Sentence Simplification with Transformer-XL and Paraphrase Rules

Fei Fang[1] **& Matthew Stevens**[2]
Stanford University
Stanford, CA 94305
[1]feifang@stanford.edu, [2]mcs0042@stanford.edu

Abstract

Neural sentence simplification is an emerging technology which aims to automatically simplify a sentence with complex syntax and/or diction while preserving its meaning. Most of the existing work on neural sentence simplification has modeled the task as monolingual machine translation (MT), and has investigated applications of architectures such as RNN- or Transformer-based seq2seq encoder-decoder networks. In this project, we explore a new model by adapting Transformer-XL, the cutting-edge architecture for language modeling (LM), to sentence simplification. In addition, we propose and investigate two original approaches to incorporate the Simple Paraphrase Database (PPDB), a large database of reduction rules for words and phrases. The character-level Transformer-XL model achieved performance close to SOTA. Though our experiments using the SimplePPDB variants do not achieve desirable results, they reveal important insight regarding the integration of accurate paraphrase rules into neural models. This project is the first known application of Transformer-XL to a non-LM task, as well as the first known sentence simplification model that can use character embeddings.

1 Introduction

Given a sentence with complex syntax and/or diction as input, a sentence simplification model aims to output a sentence that is easier to comprehend and yet preserves the semantics of the input. This technology helps readers process and retain information more efficiently. Furthermore, it is especially useful to children, nonnative readers, as well as those with reading disabilities such as dyslexia [5]. Since simplification can be thought of as monolingual translation, neural sentence simplification has been studied with the same architectures that are applied to standard interlingual neural machine translation (NMT): seq2seq architectures involving RNNs or Transformers [6, 8, 9]. These models were trained on a large parallel corpus of complex-simple sentence pairs in English.

Aside from neural methods, many statistical methods have also been applied to sentence simplification. Relative to neural methods, statistical methods require significantly more extensive feature engineering [9]. However, even though neural architectures have shown promising results for interlingual MT, when it comes to sentence simplification, neural models have rarely outperformed statistical models [6, 8, 9]. We conjecture that this is due to the fact that unlike in interlingual MT, in sentence simplification, most tokens of the simplified sentences are copied over from the originals [8]. As a result, a standard supervised-learning encoder-decoder model becomes "too" good at pointing to parts in the source sentences and then copying them over to the predictions, instead of making significant simplifying changes. In comparison, statistical rule-based systems consistently perform simplifying changes. To combine the strengths of both neural and statistical methods, it is worth exploring hybrid models that are based on neural networks and yet still make use of paraphrase rules [9].

In this project, we explore a potential improvement to existing neural models for sentence simplification. We adapted Transformer-XL, a novel architecture that had not been applied to sentence simplification before, to our task at hand. Transformer-XL is a variant of the Transformer architecture that incorporates a recurrence mechanism. In comparison with the vanilla Transformer, Transformer-XL is able to learn temporal dependencies beyond a fixed-length context, and aims to outperform Transformer on language modeling (LM) [1]. Indeed, it has demonstrated stronger performance on LM than RNNs and Transformers, thanks to its ability to learn longer dependencies. Since the lack of ability to learn long dependencies is a common shortcoming of both RNNs and Transformers, we decided to explore the application of Transformer-XL architecture to sentence simplification. In fact, our model is the first to apply the Transformer-XL architecture to a task other than language modeling.

In an effort to build a hybrid neural-statistical sentence simplification model, we further investigate two approaches to integrate the Simple Paraphrase Database (PPDB) into our neural model. SimplePPDB is a collection of approximately 4.5 million reduction rules of words and phrases in English, and there has been effort done on integrating these reduction rules into the loss function of a neural model [9]. Inspired by previous work, we explore the application of SimplePPDB to neural sentence simplification tasks via two integrations: a loss function using the paraphrase rules, and a "weighting" system that assigns a "confidence score" to each gold label, which then in turn goes into the computation of losses.

The character-level Transformer-XL model achieved a SARI score of 27.39. While neither the word-level model nor the SimplePPDB-integrated variants performed well, our experiments with the latter variants reveal important insight discussed in the Analysis section.

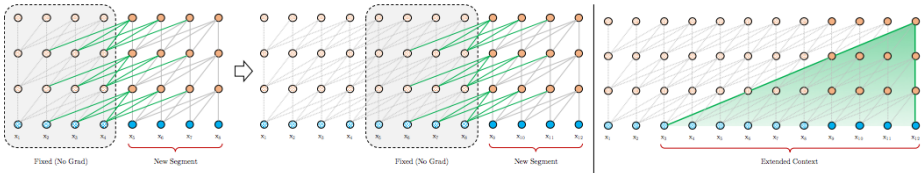
2 Related work

In the past few years, deep learning (DL) techniques, especially sequence-to-sequence architectures such as RNNs and LSTMs, have led the progress in natural language processing (NLP), including machine translation. In particular, these architectures have also been applied to intralingual translation, such as paraphrase and text simplification. The first and most straightforward DL-based approach to sentence simplification is supervised learning: take a simplified parallel corpus (e.g. semantically aligned sentences from the normal English Wikipedia and the Simple English Wikipedia), and train a seq2seq encoder-decoder model (often with LSTM layers) on the corpus [8]. It was shown that although the baseline could generate fluent sentences, it often fails to simplify the complex sentences [8]. This baseline model was then refined by adding in a reinforcement learning (RL) component, where the REINFORCE algorithm uses the output of the baseline model as input [8]. Another approach to refine the baseline was to adapt the Neural Semantic Encoder [3], an encoder architecture with augmented memory, to the encoder component of the LSTM-based encoder-decoder model, so that the model could learn longer dependencies [6].

Since it has been observed that neural models still have not outperformed statistical, rule-based models on sentence simplification, a recent study proposed a neural-based model that integrates simplifying paraphrase rules from the SimplePPDB [9]. The neural component of this model is based on the Transformer, a multi-layer, multi-head attention-based encoder-decoder. It was claimed that the integration of SimplePPDB ensured that infrequent simplifying paraphrases, which are often ignored by neural models, are still applied [9]. In particular, the SimplePPDB [4] was integrated into the loss function. The SimplePPDB database includes approximately 4.5 million simplifying rules r, each of which is given a "simplification" score w_r between 0.0 and 1.0, with high scores assigned to simplifications which are simpler compared to their target sentences. Given a sentence I containing a word or phrase A which can be simplified to a using a reduction rule r in the SimplePPDB and a corresponding prediction I', the authors' loss function L_{critic} is defined as

$$L_{\text{critic}}(I') = \begin{cases} -w_r \log P(a|I) & \text{if model generates } A \\ w_r \log P(A|I) & \text{if model generates } a. \end{cases} \quad (1)$$

Informally, the loss function rewards the model for applying reductions in the SimplePPDB, and penalizes the model for not doing so, in proportion to the simplification scores for each reduction rule. We have several concerns regarding the specification of this loss function; because the function is essentially defined by example, it is unclear what score will be applied if a word or phrase has multiple



(a) Training phase. (b) Evaluation phase.

Figure 1: An illustrated diagram of the Transformer-XL training and evaluation phases. Each segment/subsequence is a layer in the Transformer-XL. In the training phase, the gray box indicates the previous subsequence, while the filled blue and orange dots make up the current subsequence. Source: [1].

simplifications in the PPDB, or how simplifiable phrases which overlap with other simplifiable phrases are handled by the function. To the best of our knowledge, the L_{critic} loss function is not implemented in the publicly released codebase and all attempts to contact the authors to clarify the situation were unsuccessful.

It was claimed that the Transformer-SimplePPDB model achieved state of the art results among all existing models.[1] However, as mentioned earlier, the paper failed to elucidate important aspects of the implementation of the integration of the SimplePPDB into the loss function. Furthermore, there was no qualitative analysis on example outputs in the paper. In addition, we failed to reproduce the results of that study from the public repository.

Inspired by the proposal of the neural-statistical model [9], in our project, we conducted an investigative project on different approaches to build a hybrid neural-statistical model. Since the Transformer-XL is an extension of the Transformer, we believe that the neural component could benefit from the deployment of Transformer-XL. In addition, we investigated and implemented two new approaches to integrate the SimplePPDB into our loss function.

3 Approach

3.1 Main approaches

3.1.1 Transformer-XL

The basic neural-based model in our project is based on the Transformer-XL architecture. The Transformer-XL is an architecture proposed for language modelling. Given a sequence of characters $x_1, x_2, \ldots, x_n$, the goal of the Transformer-XL is to predict the following character, x_{n+1}. Similar to the vanilla transformer, when the context is long, the Transformer-XL splits a sequence into subsequences, uses the attention framework of the Transformer on each subsequence. Yet instead of fragmenting the context like the vanilla Transformer, the Transformer-XL incorporates a recurrence mechanism to learn dependencies across subsequences.

The recurrence mechanism enables the learning of dependencies beyond fixed-length context by using information from previous subsequences. Like in the vanilla Transformer, when a subsequence is processed in Transformer-XL, each hidden layer in the current subsequence receives the output of the previous hidden layer in the same subsequence as input (represented by the gray arrows in Figure 1). Yet unlike in the vanilla Transformer, in Transformer-XL, each hidden layer in the current subsequence also receives the output of the previous hidden layer in the *previous* subsequence as input (represented by the green arrow in Figure 1). The latter mechanism thus links consecutive subsequences to each other, and as Figure 1(b) shows, in evaluation/prediction phase, the dependencies that the Transformer-XL can learn and use actually extend beyond any two consecutive segments.

Since Transformer-XL is proposed for LM, we took steps to adapt it to sentence simplification. More specifically, the goal of our model is as follows. The model is first given a complex sentence

[1] On one metric (SARI) and on one test set (*WikiSmall-AMT*). See details in 4.1 Datasets and 4.2 Evaluation Metric.

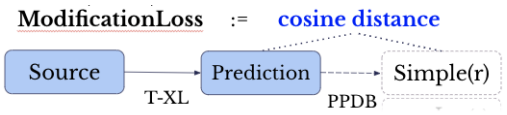


Figure 2: A visual description of ModificationLoss.

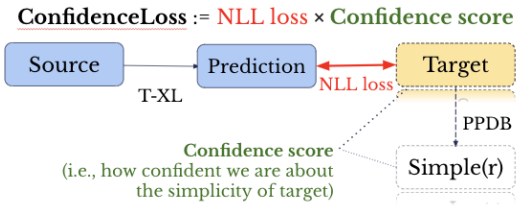


Figure 3: A visual description of ConfidenceLoss.

$I = (x_1, x_2, \ldots, x_n)$ as input, where for each $i \in \{1, 2, \ldots, n\}$, x_i is the ith token in the complex sentence, and n is the length of the complex sentence I. Then, the model outputs a simple sentence $O = (y_1, y_2, \ldots, y_m)$, where for each $j \in \{1, 2, \ldots, m\}$, y_j is the jth token in the simple sentence, and m is the length of the simple sentence O. The model is trained to minimize the negative log-likelihood loss of the simple sentence O,

$$L_{nll} = -\log p(O|I, \theta),$$

where θ represents the current parameters in the model.

To our knowledge, this is the first known adaptation of the Transformer-XL to a non-LM task.

3.1.2 Incorporation of SimplePPDB

The extension of our model makes use of the SimplePPDB, a large database of simplifying paraphrases for words and phrases. The current iteration of SimplePPDB is derived from the second iteration of the (general purpose) Paraphrase Database (PPDB) and consists of a subset of approximately 4.5 million reduction rules, selected automatically using a multi-class logistic regression classifier trained to identify rules which output a phrase simpler than the input phrase. Each simplification rule in the database is accompanied by a decimal "quality" score between 1.0 and 5.0 indicating the quality of the paraphrase (i.e. how well the paraphrase preserves the meaning of the original phrase) and a decimal "simplicity" score between 0.0 and 1.0 equal to the regression classifier's confidence score for the phrase in question (paraphrases with confidence score less than 0 are of course not included in the database). Note that these paraphrases were generated automatically, not by humans [7].

We investigate two **original** methods for integrating SimplePPDB reduction rules into our model.

First, we construct a loss function, ModificationLoss, inspired by the L_{critic} loss function in Zhao et al. [9]. ModificationLoss takes as input a prediction $\hat{y}$ and replaces all words in Simple PPDB with the one-word simplification scoring highest in terms of quality, producing a sentence $\hat{y}'$. The final loss value is CosineDistance$(\hat{y}, \hat{y}')$. In essence, this loss function penalizes predictions which can be simplified extensively using the SimplePPDB and rewards those that cannot. When integrating with Transformer-XL, the loss is a weighted sum of ModificationLoss and negative log-likelihood loss.

Second, we devise a method of "weighting" loss values between predictions and targets as follows. Let $\hat{y}$ be a prediction, y be the corresponding target, and n be the number of words in y. Let s be the number of words in y which have simplifications in the SimplePPDB. Then, the final loss value between $\hat{y}$ and y is

$$\text{ConfidenceLoss}(\hat{y}, y) = \text{NLL}(\hat{y}, y) \cdot \left(1 - \frac{s}{n}\right) \cdot c$$

for $0 < c < 1$ a hyperparameter set to $\frac{1}{5}$ in our tests. Informally, if the target y corresponding to a prediction $\hat{y}$ can be extensively simplified using the SimplePPDB, we decrease the loss between y and $\hat{y}$ by a multiplicative factor if a large number of words in the target sentence can be simplified using SimplePPDB, indicating that the target sentence provided in the corpus is possibly of poor

quality and the model should be less eager to decrease the distance between $\hat{y}$ and y. We refer to this loss function as ConfidenceLoss.

3.2 Baseline

Our baseline is the performance of an attention-based encoder-decoder model with LSTM layers (EncDecA). The implementation, results, and outputs of this system are provided by Zhang and Lapata, 2017 [8].

3.3 Implementation details

Our model was implemented in PyTorch. The neural architecture of our model is adapted from the original repository of Transformer-XL [2], which implements word and character language modeling using the Transformer-XL architecture. Because the original Tranformer-XL codebase is intended for language modeling tasks and supports only a small collection of corpora, a number of significant modifications were necessary in order to enable the Transformer-XL model to train for our non-LM task.

Note that because Transformer-XL is intended for language modeling, the native preprocessing step involves loading input sentences from a single folder of files and producing output sentences by shifting the input sentences one token (character or word level) to the right. On the other hand, sentence simplification tasks require that complex sentences and their simplified counterparts, which oftentimes differ in length, to be loaded separately. As such, we implemented a data loading module which loads complex and simple sentences from separate filepaths, constructs token embeddings (character-level or word-level, per request), pads sentences to a user-defined length, and tokenizes sentences. Since the implementation of the Transformer-XL requires that both source and target tensors are of shape `(batch_size, sequence_length)`, we then constructed a batch iterator for each of source and target, where each batch contains the tokenized and padded sentences in the required shape. In addition, we modified the Transformer-XL architecture so that it takes in an index for the pad token and knows to not assign attention to any pad token.

For our SimplePPDB-dependent variants, we also loaded the SimplePPDB database along with complex and simple sentences, and used SimplePPDB entries in the construction of token embeddings. We then constructed a dictionary which maps a complex word or phrase to its entry in the SimplePPDB, and used the dictionary to construct a custom PyTorch loss function, which implements ModificationLoss, and a utility which computes the multiplicative factor used in the implementation of ConfidenceLoss.

4 Experiments

4.1 Datasets

For training, we used the *WikiLarge* dataset, constructed by Zhang and Lapata, 2017. This is the largest parallel Wikipedia corpus, containing over 296k normal-simple sentence pairs [8]. For development and test, we used the same dev and test sets as in previous work [8], created by Xu et al., 2016 [7]. The sources of dev and test data come from the *WikiSmall-AMT* dataset. The *WikiSmall-AMT* dataset consists of 2,359 original sentences taken from *WikiSmall*, a parallel Wikipedia corpus, each with 8 reference simplifications, collected from Amazon Mechanical Turk workers. The 2,359 examples have been split into 2,000 for dev and 359 for testing [8].

When training on word-level embeddings, in order to reduce the vocabulary size, we followed the preprocessing scheme of previous work [8, 9], and replaced proper nouns of people's names, locations, and organizations with tokens in the format of PN@n, where PN $\in$ {PERSON, LOCATION, ORGANIZATION} is the category of the proper noun, and n denotes the nth distinct proper noun of a given category in a given sentence.

We followed the same train-dev-test split as baseline and SOTA models [8].

4.2 Evaluation metrics

SARI The System Output Against References and Against the Input Sentence (SARI) metric is a novel metric proposed by Xu et al. [7] in 2016 specifically for the evaluation of text simplification models. The SARI metric evaluates a simplification based on three criterion: addition, under which a simplification is rewarded for including a word not present in the unsimplified sentence, but present in human reference sentences; keeping, under which a simplification is rewarded for retaining a word which is also retained in reference sentences; and deletion, under which a simplification is rewarded for erasing a reward which is also erased in the reference sentences. Note that SARI has been shown to outperform BLEU in both quantitative and qualitative tests [7]. To ensure consistency of computation of SARI with respect to existing work on sentence simplification, we use the implementation of SARI provided by the authors of SARI [7].

It should be noted that, strictly speaking, the SARI metric functions by comparing outputs with target and reference sentences *at the word level*. Thus, it is not entirely appropriate for evaluating character-level models. We address this concern in the Analysis section.

4.3 Experimental details and results

4.3.1 Transformer-XL

Character-level We performed a randomized grid search to find the optimal hyperparameters for our model. Hyperparameters tuned include the number of layers, model dimension, number of attention heads, the attention head dimension, and the batch size. Given the restrictions of computing power, the best-performing model has 2 layers, where the dimension of each layer is 64. Each layer has 8 attention heads, each of dimension 8. The positionwise feed-forward layer has an inner dimension of 4096. The batch size was 32, and the length of each example was capped at 128. The model was trained with a learning rate of 0.001, with a decay rate of 0.5, and a dropout rate of 0.05. The training deployed an NVIDIA Tesla M60 GPU with 56GB of memory, and completed in approximately 3 hours.

Upon convergence, the model achieved a val loss of 2.26 and perplexity 9.597, and a test loss of 2.26 and perplexity 9.605. The set of test predictions achieves a SARI score of 27.39.This result is close to our expectations. As we can see in Table 1, we outperformed the baseline by a huge margin, and achieved a SARI score close to SOTA.

Word-level When we trained the model on word-level embeddings on the entire training set, we observed that both the train loss and val loss would fluctuate substantially. We thus began our debugging process by trying to overfit on a small dataset. While our word-level model successfully overfit on a small set of four sentences, it failed to overfit to a sample of 100 sentences, despite extensive hyperparameter tuning and debugging with tensor shape and value checks. We offer possible explanations for the failure of our word-level model in the following section.

4.3.2 Transformer-XL with SimplePPDB Integrations

When we trained the Transformer-XL model with either of the two approaches to integrate the SimplePPDB on the entire training set, we observed that both the train loss and val loss would fluctuate or increase. To debug the model, we tried to overfit on a small daaset by training and testing both loss methods on a small set of four complex-simple pairs, at both the character and word

Model	SARI (Eval Metric)
Baseline (LSTM Enc-Dec w/ Attention)	13.61
SOTA (Statistical model)	**30.46**
Transformer-XL w/ character embeddings	27.39

Table 1: Automatic SARI Evaluation on the WikiSmall-AMT dataset. Source of results of baseline and SOTA: [8].

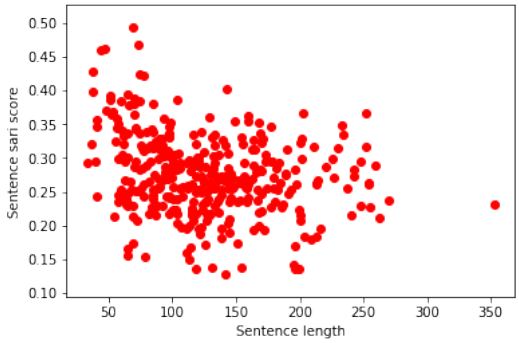


Figure 4: Scatterplot of complex sentence length (in characters) against prediction SARI score

levels. As expected, our model successfully overfit. We then tried to overfit on a larger dataset of 100 complex-simple pairs. Unfortunately, despite extensive experimentation with hyperparameters, all models either exhibited worse performance relative to the pure Transformer-XL architecture, or did not converge at all, with the best performing models performing approximately 10% worse compared to pure Transformer-XL. As such, we did not evaluate the SimplePPDB variants using SARI. We offer possible explanations for the failure of our SimplePPDB variants in the following section.

5 Analysis

Given that the *WikiSmall-AMT* Dataset has a substantial number of references, we performed our analysis on outputs for a portion of this dataset. In particular, we would like to analyze the strengths, weaknesses, and potential of the multiple variants of sentence simplification systems we have proposed in this project.

5.1 Correlation between SARI and sentence length

We produced a scatterplot of the lengths of the complex sentences against the SARI scores that their corresponding predictions yield. We observe a weak negative correlation between the two. Relative to sentence lengths against BLEU in MT, this correlation is extremely subtle. We conjecture that this is because a significant portion of the tokens in the target sentences are copied over from their source counterparts. And as a result, even if the model had only learned the identity mapping, the predictions would not yield an extremely low SARI score.

5.2 Character-level Transformer-XL

The use of the SARI metric As mentioned above, SARI is a word-level metric and is thus not entirely appropriate for evaluating character-level models. In particular, an output sentence produced by a character-level model which makes a single character error and a character-level model which outputs an entirely erroneous word may be penalized by the same amount.

Evidence of this instability can be seen in our test predictions: consider the prediction "The book , Political Economy , was publise", whose corresponding source sentence is "The book , Political Economy , was published in 1985 , but had limited classroom adoption .", which receives a SARI score of 32.82. While the word "published" is in 7 of 8 reference sentences, the word "publise" is in none of them. As such, the prediction does not receive a reward for retaining "published". However, if the word "publise" is corrected to "published", an edit requiring only two insertions, the prediction receives the significantly higher SARI score of 34.89, an increase of more than two full points. Note, however, that a misspelled word does not always negatively affect the SARI score. If all references had decided to *delete* the word "publish" instead of retaining it, the SARI score of the above prediction would be artificially high, rather than artificially low.

While this, in general, means that character-level and word-level models cannot be compared according to SARI, we can estimate the effect that minor misspellings have on the overall SARI score by

preprocessing predictions with a spellchecker before computing the SARI value. We perform this computation using PyEnchant by replacing all words which are not in the English dictionary by the first suggestion offered by the library; modified predictions receive a collective SARI score of 25.59, lower than the original SARI score. This is likely due in part to a reduction in the deletion component of the SARI score, as PyEnchant restores words deleted in references previously considered to have been deleted in the predictions. However, some of this discrepancy may be due to the presence of proper nouns in the corpus.

Incomplete sentence endings We note that many predictions made by the character-level model are often exceptionally short compared to both the source and target sentences. The mean and median line lengths among predictions were 51.43 characters and 51 characters, respectively, while the mean and median line lengths among the source sentences were 125.2 characters and 119 characters, and the mean and median line lengths among target sentences were 117.00 characters and 109 characters.

This suggests that many of the predictions produced by the model end inappropriately early, and the direct inspection of specific examples confirms this result. Indeed, we observe that the model predicts

- "His real date of birth was never recorded , b t it", while the target sentence was "Since his actual date of birth was not recorded , it is believed to be between 1935-1939 .",
- "This quantitative measure indicates how much of a particuar", while the target sentence was "This quantitative measure indicates how much of a drug or other substance is needed to inhibit a biological process by half .", and
- "The string can vibrate in different modes just as a", while the target sentence was "The string can vibrate in different modes just as a guitar string can produce different notes , and every mode appears as a different particle : electron , photon , gluon , etc. .".

That is, sentences are often initially coherent for the first 45-50 characters, at which point they quickly become nonsensical and stop.

This is likely due to the fact that prediction sentences are constructed in a character-by-character fashion based on characters which have already been predicted; once a small number of faulty character predictions are made, the errors "snowball" at which point the predicted sentence begins to become nonsensical. We observe that as a language model, character-level Transformer-XL at its inception was evaluated on an output sequence of $\frac{1}{4}$ the length of the target sequence in the training set. Note that source and target sentences often contain pad tokens near the end of the sentence, and thus we expect that our model tends to predict that pad tokens follow pad tokens. Since pad tokens are removed before predictions are output, this means that sentences often stop soon after a pad token is erroneously inserted when the sentence begins to become incoherent. Therefore, it is possible that with character-level embeddings, the current Transformer-XL cannot make good predictions that are as long as targets in the training samples. If so, the model architecture of Transformer-XL would need some significant changes in order to be appropriate for this task.

5.3 Word-level Transformer-XL

The current implementation of Transformer-XL requires a 2-dimensional tensor of shape `(batch_size, sequence_length)`. Therefore, when it comes to giving word embeddings to each token, we were forced to either assign an index to each unique token, or revamp the existing implementation such that it can take in a 3D tensor where each word is embedded using a dense representation (e.g., GLoVe, word2vec). We decided upon the former given the consideration of time limits. However, this means that essentially, one-hot embeddings were used for our word-level model, which hinders effective learning. This is our conjecture for the significant fluctuations of training loss and val loss during the training of our word-level model. Given more time, we would look into how to modify the implementation of the model such that it would allow dense vector representations of words.

5.4 SimplePPDB integrations

As mentioned above, neither SimplePPDB integration performed well enough for testing on the full WikiLarge set. We offer possible explanations for the failure of both integrations.

5.4.1 ModificationLoss

Poor SimplePPDB reduction rules As mentioned above, the SimplePPDB is an automatically generated paraphrase database, and prediction rules were not devised by human writers. While efforts are made to ensure that simplification rules in the database a) preserve grammaticality and meaning, i.e. simple phrases are "drop-in replacements" for their unsimplified counterparts; and b) are actually simplifications, numerous errors exist in the SimplePPDB.

A significant number of reduction rules fail to satisfy b). For example, the SimplePPDB maps "limit values" to "restriction" and "limitation" to "qualifier," while in both of those cases, the former phrase is arguably a simplification of the latter.

However, a far larger number of reduction rules seem to fail to satisfy a, i.e. reductions commonly fail to preserve grammaticality. For example, the SimplePPDB replaces

- "unemployed" with "lose their job," even though the former is an adjective and the latter is a verb phrase;
- "parlamentarian" with "all meps," even though the former is a singular noun and the latter is a collective noun;
- "statistician" with "statistics," even though those two words do not refer to the same idea;
- "nowhere" with "go anywhere," even though the former is a pronoun/adverb (i.e., same lexical category as "anywhere"), the latter is a verb with an adverb, and meaning is not preserved by the reduction (the reduction is also arguably not a simplification).

In principle, these issues might be remedied using Uniform Cost Search to ensure that reductions are applied in such a way that they preserve meaning and grammaticality, as well as produce a simpler sentence; however, this solution is computationally infeasible for medium or large datasets. Thus, an improvement on the original SimplePPDB itself is likely necessary to address these shortcomings.

Unstable gradient descent Note that the final ModificationLoss value is the cosine distance between a batch tensor of unmodified predictions and a batch tensor of predictions modified using the SimplePPDB. While the modified batch tensor is computed from the original batch tensor, the function mapping the unmodified tensor to the modified one is *not* differentiable, or even continuous. As such, the modified batch tensor is essentially treated as a constant in the backpropagation calculation, on the false but simplifying assumption that the modified tensor will usually remain constant in a small neighborhood around a given point. One possible explanation for the failure of ModificationLoss is that this assumption is an impractical one; that is, treating the modified batch tensor introduces enough noise into the gradient descent process so as to render it ineffective.

5.4.2 ConfidenceLoss

Informally, ConfidenceLoss reduces the loss between a prediction and target if that target contains many words which can be simplified using the PPDB. This may be problematic for small datasets. If a particular pattern appears in a large number of source sentences, this method may effectively prevent the model from learning to map the pattern to a poor target pattern. For small datasets, where many patterns appear infrequently for no other reason than the size of the dataset, this approach may prevent the model from learning to effectively map the pattern to *anything*. Testing on far larger datasets is likely necessary to determine if this is the sole reason for the poor performance of the models trained with ConfidenceLoss; if it is, then results should significantly improve.

We conjecture that one way to counter this tendency for small datasets (and possibly improve the performance of the entire model) might be to modify the ConfidenceLoss function such that instead of reducing the loss for a predicted sentence x if its corresponding target $\tilde{x}$ contains many words which are simplifiable by the SimplePPDB, the function reduces the effect of the sentence on the loss of its batch. That is, where ℓ is the loss of the batch $B \ni x$ (recall that the loss of the batch is the mean of the negative log likelihoods for each predicted character accross the sentence batch), we multiply the column vector by a scalar such that the loss ℓ' of the resulting batch is closer to the loss of the batch $B - \{x\}$, by a factor depending on how many words in $\tilde{x}$ can be simplified using the SimplePPDB. Given more time, we would have implemented this new loss function.

6 Conclusion

We implemented a Transformer-XL-based model for sentence simplification and investigated two methods of integrating paraphrase rules. To our knowledge, this is the first application of the Transformer-XL architecture to a non-LM task, as well as the first study on character-level sentence simplification. We find that Transformer-XL's exceptional ability to perform language modeling on character level has led us to successfully build a character-level model for sentence simplification, with a straightforward encoding and decoding framework. This suggests that with some more sophisticated adaptation of Transformer-XL, the architecture has tremendous potential in tasks including but not limited to sentence simplification. Future avenues of research include improving the SimplePPDB for use in real-world sentence simplification tasks and exploring the efficacy of weighting losses based on target sentence quality using larger datasets. We also look forward to seeing BERT-XL, where the representations from BERT are trained on Transformer-XL, and can facilitate a wide range of tasks in NLP.

7 Additional Information

We would like to thank our mentor, Abi, for her remarkable patience and guidance regarding the direction of this project.

We would like to acknowledge Nishith Khandwala for his invaluable advice regarding the efficient implementation of this project.

References

[1] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019.

[2] kimiyoung. transformer-xl. `https://github.com/kimiyoung/transformer-xl`, 2019.

[3] Tsendsuren Munkhdalai and Hong Yu. Neural semantic encoders. *CoRR*, abs/1607.04315, 2016.

[4] Ellie Pavlick and Chris Callison-Burch. Simple PPDB: A paraphrase database for simplification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016.

[5] Luz Rello, Clara Bayarri, Azuki Gorriz, Ricardo A. Baeza-Yates, Saurabh Gupta, Gaurang Kanvinde, Horacio Saggion, Stefan Bott, Roberto Carlini, and Vasile Topac. Dyswebxia 2.0!: more accessible text for people with dyslexia. In *W4A*, 2013.

[6] Tu Vu, Baotian Hu, Tsendsuren Munkhdalai, and Hong Yu. Sentence simplification with memory-augmented neural networks. *CoRR*, abs/1804.07445, 2018.

[7] 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, 2016.

[8] Xingxing Zhang and Mirella Lapata. Sentence simplification with deep reinforcement learning. *CoRR*, abs/1703.10931, 2017.

[9] Sanqiang Zhao, Rui Meng, Daqing He, Andi Saptono, and Bambang Parmanto. Integrating transformer and paraphrase rules for sentence simplification. *CoRR*, abs/1810.11193, 2018.