Sentence simplification with character-level transformer

Kuan Yu Sonu Rauniyar Maya Angelova Philipp Schoneville
{kuanyu, rauniyar, schoneville}@uni-potsdam.de
maya.angelova@protonmail.com

Master's Program in *Cognitive Systems*
University of Potsdam

September 2018

Abstract

We applied the transformer, a neural network architecture developed for machine translation, to the task of sentence simplification. We modeled sentences at the character level, using automatically aligned sentence pairs from the English Wikipedia and the Simple English Wikipedia. Our system demonstrated text normalization capacities but only elementary simplification abilities, due to the limitations of the data. However we made effectual revisions to the transformer, and learned various lessons from our explorations.

1 Introduction

Text simplification can be formulated as translation from a language into a sublanguage with reduced linguistic complexities, and the task can be automated using machine translation systems (Wubben, Van Den Bosch, and Krahmer 2012; Narayan and Gardent 2014; Xu, Napoles, et al. 2016). Such systems must know the distributional properties of the source and the target languages, and model a mapping which preserves the semantic and discourse structures.

Machine translation systems commonly model sentences at the word level. However, due to the Zipfian distribution of words in a natural language, the vocabulary can never be fully modeled. In order to handle unknown words, it is common practice to replace the low frequency ones in the data with one special symbol. Among the rare words are proper names and large numbers, information which should be preserved during translation, therefore a word-level system needs to be augmented with components for named entity recognition and placement, along with the component for tokenization. Additional components inevitably introduce errors and instabilities. In comparison, a character-level system can be more robust and just as effective (Kalchbrenner et al. 2016). Sophisticated hybrid systems have also been proposed (Luong and Manning 2016; Wu et al. 2016). Ideally, such a model learns the morphology to construct new words, which makes it easier applicable to more languages such as agglutinative ones. Our system models sentences on the character level.

$$\begin{tikzcd}
T_0^* \arrow[r, bend left] \arrow[rr, bend right] \arrow[rrr, bend left] & T_1^* \arrow[r, bend right] \arrow[rr, bend left] & T_2^* \arrow[r, bend left] & T_3^*
\end{tikzcd}$$

Figure 1: An autoregressive network.

The transformer is a neural network architecture developed for machine translation with great success (Vaswani et al. 2017). The architecture relies on attention layers, as opposed to the conventional recurrent and convolutional layers. It has since been used for constituency parsing (Kitaev and Klein 2018), language generation (Liu et al. 2018), image generation (Parmar et al. 2018), and speech recognition (Zhou, Dong, et al. 2018; Zhou, Xu, and Xu 2018). The goal of our project is to apply the transformer for sentence simplification.

We made effectual revisions to the transformer in our open source implementation.[1] Our system demonstrated text normalization capacities and elementary simplification abilities. In this paper we first describe the background of the machine learning model (Section 2), with a focus on attention mechanisms. Then we discuss the experiment setup (Section 3), specifically the data and the evaluation criteria, and explain our revised architecture (Section 4). Finally we describe the training process and examine the results (Section 5), and present our conclusion (Section 6).

2 Background

2.1 Encoder-decoder

Sequence-to-sequence neural networks are a popular choice for learning text simplification (Wang et al. 2016; Zhang et al. 2017; Vu et al. 2018). A sequence-to-sequence network usually consists of an encoder and a decoder, trained jointly with pairs of sequences. The encoder learns a representation for the source sequences, and the decoder learns to reconstruct the target sequences from the representation. Since these sequences vary in their lengths, a multi-layer perceptron (MLP) is not the ideal architecture, while recurrent networks are well-suited (Sutskever, Vinyals, and Le 2014), and a convolutional one can be effective as well (Gehring et al. 2017).

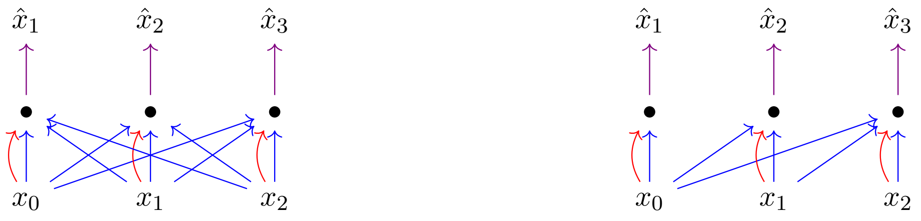
The decoder is often autoregressive, which means that it constructs a sequence one step at a time, conditioning on the previous steps, as shown in Figure 1. Here it also has to globally condition on the outputs from the encoder. Formally, a sequence-to-sequence encoder-decoder models the mapping $\forall i \in \mathbb{N}, (S^*, T^*_{0\dots i}) \to T^*_{i+1}$, where S^* are the collection of source sequences freely generated by the collection of source symbols S, and T^* the target sequences generated by T. Since each target step may have arbitrarily many dependencies from the previous target steps as well as the source steps, a good mechanism is needed to determine the importance of these dependencies and to combine them into a single representation. Attention is one such choice.

2.2 Attention mechanisms

A general description for the commonly used soft attention is as follows, given a query vector and multiple value vectors:

1. The query determines a scalar weight for each value;

[1] https://github.com/srewai/explicharr



Encoder self-attention

Decoder self-attention

Figure 2: Self-attention layers. Arrows of the same color denotes linear transformations with the same parameters. The red arrows produce the queries, the blue arrows produce the values and keys, and the black bullets are the attention cells which produce the summaries. The purple arrows produce the outputs, which may be part of the next layer.

2. The weights are then normalized to a probability distribution over the values, typically with the softmax[2] function;
3. The summary is simply the weighted sum of the values.

Bahdanau, Cho, and Bengio (2014) introduced an attention mechanism to machine translation known as additive attention, typically used in a recurrent structure. The recurrent state is used as query. Each value vector is concatenated with the query and passed into an MLP to produce the attention weight. The usual choice for the MLP has a single hidden layer with tanh activation. Luong, Pham, and Manning (2015) introduced another mechanism where the attention weight is simply the dot product between the query and the value. This is known as dot-product or multiplicative attention. Daniluk et al. (2017) introduced key-value attention, where the value vectors are split into two parts, the values and the keys. Only the key vectors are used with the query to produce the weights. This separates the tasks for the model to learn good representations for the values and the keys.

One common application of attention is for the decoder to attend to the encoder outputs, but it can also be used for both the encoder and the decoder to attend to its own data without any recurrent structure (Parikh et al. 2016). This is known as self-attention or intra-attention. In sequence learning, self-attention means that a query is simply one time step among the sequence, and the values are the whole sequence or a subsequence. The summarized value becomes an annotated representation for data at the time step queried, with contextual knowledge of all data within the attended time steps. Due to the autoregressive nature of the decoder, it must not attend to the future steps. Figure 2 illustrates the self-attention layers.

For comparison, Figure 3 illustrates the more commonly used convolutional and recurrent layers in the context of sequence learning. In a convolution layer, each step can only access information from a fixed number of input steps, but the

[2]$\text{softmax} : \mathbb{R}^n \to \mathbb{R}^n, x_i \mapsto \exp x_i / \sum_{j=1}^n \exp x_j$

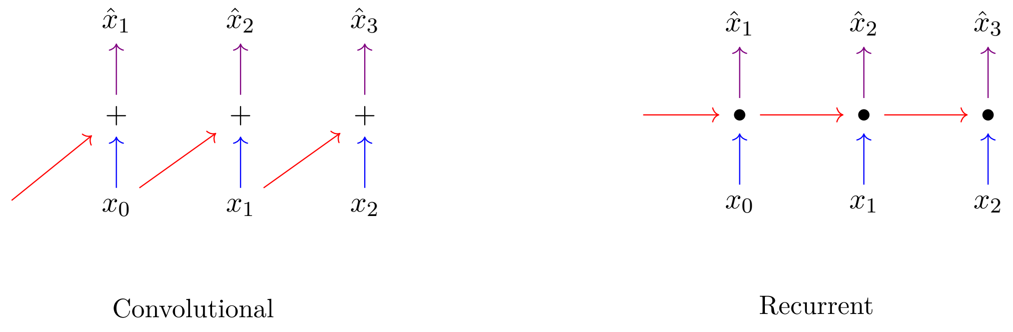


Figure 3: A convolutional layer with filter size 2 and stride 1, and a recurrent layer where the black bullets denote the recurrent cell, cf. Figure 2.

computations for all input steps are independent and can be parallelized. In a recurrent layer, the recurrent state carries information from all past steps, but the sequential nature makes training slow. The recurrent state may be an information bottleneck due to its fixed size, and long distance dependencies may be difficult to learn. While self-attention layers combine the advantages of convolutional layers and recurrent layers, its computational complexity is quadratic instead of linear with respect to the length of the sequence, since each step may query every other step.

Without either locality or temporal dependency, the attention mechanism does not respect the ordering of a sequence. Gehring et al. (2017) proposed to add positional information to the data, in the form of a learned position embedding. Vaswani et al. (2017) proposed to hard-code the positional information as sinusoids. For dimension i and position j, the encoding is defined as $p(j, 2i) = \sin\left(j/10000^{2i/d}\right)$ and $p(j, 2i+1) = \cos\left(j/10000^{2i/d}\right)$ where d is the total number of dimensions. The motivation is that the wavelengths form a geometric progression from τ to 10000τ, and that the relative distance between positions can be expressed as a linear function. Unlike with a learned position embedding, the length of sequences are not limited by the maximum position learned.

2.3 Transformer

The transformer uses scaled dot-product attention, a novel variant of key-value dot-product attention. The attention weights are divided by the square root of the representation dimension before normalized by the softmax function. The scaling normalizes the variance of dot products. On top of that, it introduced multi-head attention. The query space is split into multiple disjoint subspaces. Each split is an attention head. The same is done for the key space and the value space. For each head, the scaled dot-product attention produces a summary, and the combined summary is their concatenation, namely the direct sum of the subspaces. Since each head decides its own list of attention weights, the same position can be treated with different importance in different subspaces of its representation. Vaswani et al. (2017) explained the splitting of heads as projections to subspaces with learned linear transformations. In order to simplify the

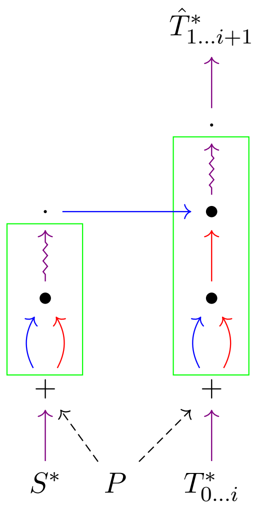


Figure 4: A transformer with the encoder on the left and the decoder on the right. P is the position encoding.
The straight arrows are linear transformations, the dashed arrows are identity functions, and the squiggly arrows are position-wise MLPs.
The coloring matches the scheme in Figure 2. The same color signifies the same interpretation, and not (necessarily) the same parameters.
The green boxes mark the transformer layers, which can be stacked to increase the depth.
The graph omits the residual connections and layer normalizations after each sublayer.

interpretation, we treat these linear transformations as procedures immediately before the attention mechanism, and the projection maps are reduced to fixed orthogonal projections, implemented as taking subvectors.

Since the attention mechanism alone merely produces a summary from a linear combination of values, the transformer also uses position-wise MLPs to increase the depth. Each MLP consists of two layers with relu[3] activation between, applied to each position separately. Each attention layer or MLP is followed by dropout (Srivastava et al. 2014), residual connection (He et al. 2016), and layer normalization (Ba, Kiros, and Hinton 2016). They are the sublayers in a transformer layer. A transformer layer in the encoder consists of a self-attention layer followed by an MLP, and a decoder layer has a secondary attention layer attending to the encoder outputs in order to condition the construction of target sequences on the source sequences. Figure 4 illustrates the full architecture.

3 Setup

3.1 Data preparation

The Simple English Wikipedia provides abundant data for training text simplification systems (Napoles and Dredze 2010). Efforts have been made to align its sentences with the ones in the standard English Wikipedia (Coster and Kauchak 2011; Hwang et al. 2015). It has also been argued that the Wikipedia datasets are not ideal for the task of learning text simplification (Xu, Callison-Burch, and Napoles 2015). However they remain a popular choice due to the accessibility and the quantity. In our experiments, we used the automatically aligned sentences classified as good and good partial by Hwang et al. (2015).[4]

The Wikipedia data come with a large set

[3] relu : $\mathbb{R} \to \mathbb{R}, x \mapsto \max(0, x)$

[4] http://ssli.ee.washington.edu/tial/projects/simplification/

of characters, over two thousand in either the standard or the simple parts. However the top 256 most frequent ones account for 99.97% of the character count. We replaced the rest with the non-breaking whitespace, and cleaned up a few text normalization errors. We also removed aligned sentences which are identical or longer than 256 characters, after which we are left with 226 208 instances. We randomly picked 1% to be used for validation, and the rest for training.

3.2 Evaluation

Machine translation systems are commonly measured by the BLEU score, a value indicating how similar the outputs are to some reference translations produced by human experts. In our case, the reference translations are the aligned sentences from the Simple English Wikipedia. Štajner, Béchara, and Saggion (2015) argued that the BLEU score is not a good measure for text simplification systems, with which we agree. For our validation data, matching the source sentences against the target sentences yield a high BLEU score of 29.53, which means that the model simply has to learn an identity function to cheat the metric. Even though we removed identical sentence pairs, ~78% of the character pairs in the training data are still identical in matching positions. Learning to replicate the source is indeed an easy strategy for the model.

We did not have a solution to this problem. In our experiments, we primarily judged the goodness of learning by monitoring the validation loss and accuracy. As a secondary criterion, we took the BLEU scores and inspected the outputs. Our baseline model (BLEU = 25.44) was a word-level transformer.

4 Architecture

We started with a faithful reimplementation of the original transformer, but made various changes to the design during our experiments. In order to explain our design choices, here is an reexamination for the attention mechanisms.

4.1 Attention reexamined

Figure 5 demonstrates the general forms of additive attention and dot-product attention. Given a query vector of q dimensions, and a sequence of value vectors of v dimensions and t time steps, packed into a value matrix of shape vt, the attention mechanism produces a summarized value vector of v dimensions. Two matrices kq and kv performs the linear transformations from the query space and the value space to the key space, where the query and values can be matched to produce the attention weights. Another linear transformation wv produces the actual values to be weighted. Finally vw transforms the summary to a value vector for the next layer. This process is repeated for each time step. The interpretation of these linear transformations may vary in specific designs. For example, with additive attention, the key transformation kv is in fact part of the MLP after concatenation, and the value transformation wv is usually the identity function, unless key-value attention is used, in which case it is an orthogonal projection. Also the final transformation vw may be part of the next layer.

Additive attention is computationally expensive. The MLP has to operate on $kt + k$ for all time steps, which is a tensor with a shape of batch size times kt^2 during training. In dot-product attention, that tensor is immediately contracted. Unless we restrict k to a very small

$$\begin{tikzcd}
v \\
wt \cdot t \arrow[u, "vw"] & t \arrow[l, "\text{softmax}"'] \\
vt \arrow[u, "wv"] \arrow[r, "kv"'] & kt + k \arrow[u, squiggly, "\text{MLP}"'] & q \arrow[l, "kq"]
\end{tikzcd}$$

Additive attention

$$\begin{tikzcd}
v \\
wt \cdot t \arrow[u, "vw"] \\
vt \arrow[u, "wv"] \arrow[r, "kv"'] & kt \cdot^{T} k \arrow[ul, "\text{softmax}"'] & q \arrow[l, "kq"]
\end{tikzcd}$$

Dot-product attention

Figure 5: Attention mechanisms with t time steps, q query dimensions, k key dimensions, v value dimensions, and w intermediate value dimensions. For the simplicity of notation, we use the shape of an object to denote the object itself. To unify matrix multiplication and batched dot product, we treated vectors as column matrices, and define $A \cdot B := AB$ whereas $A \cdot^{T} B := A^{T}B$.

$$\begin{tikzcd}
v \\
vt \cdot t \arrow[u, squiggly, "\text{MLP}"'] \\
vt \arrow[u, dashed] \arrow[r, dashed] & vt \cdot^{T} v \arrow[ul, "\text{softmax}"'] & q \arrow[l, squiggly, "\text{MLP}"']
\end{tikzcd}$$

Figure 6: Revised attention cell, cf. Figure 5.

number, it is not feasible to apply additive attention in a fully attention-based model. We experimented with $k = 2, 4, 8, 16$ and found training to be much more difficult than models with dot-product attention using a similar amount of memory. The model learned slower per training step and each training step takes much longer. Therefore we focused our experiments on dot-product attention.

4.2 Revisions

The linear transformation for keys and values are useful if we wish to set the dimensions k and w to be different from v. However in the our transformer and the original, all dimensions (q, k, v, and w) are kept the same. We observed that in dot-product attention, the key transformation kv is in fact redundant. By replacing kq with $kv \cdot^{T} kq$ and kv with identity, the resulting attention weights would remain the same. Hence the model could simply learn the key transformation, if it is necessary at all, as part of the query transformation. Similarly, the value transformation wv is also redundant. The model could simply learn vw as $vw \cdot wv$, due to the associativity of linear transformations.

Since the dot product is a bilinear form, and both the key and the query transformations are linear, the original dot-product attention can only model bilinear maps, whereas the additive attention uses an MLP which is a universal function approximator (Hornik, Stinchcombe, and White 1989). Vaswani et al. (2017) observed that additive attention outperforms dot-product attention for large representation dimensions while for small dimensions they perform similarly, which motivated the proposed scaling factor. We used an MLP for the query transformation. While the attention weights are still linear with the value space, the resulting summary is no longer a simple linear combination of values. This is important for the self-attention layers, where the query comes from the same space as the values.

Figure 6 illustrates our revised dot-product attention. These changes expand the modeling capacity of the attention layers without increasing the computational complexity. In fact, omitting the key and value transformations makes dynamic caching easier. When running the decoder autoregressively, the transformation for one time step is needed for all future steps. Caching the keys and values is necessary to avoid repeating the same computation. However, the query transformation is always new to the current step. Therefore having only the query transformation means only the results after each layer need to be cached. This problem can be understood by counting the number of connections in Figure 2.

4.3 Multi-head or single-head

After replacing the linear query transformation with an MLP, we found that the multi-head design was no longer beneficial, judging from the validation loss and accuracy. We hypothesize that the real benefits of multi-head attention was due to the combination of two factors: The softmax function used for weight normalization, and the linear transformation used for the query. The softmax function normalizes the exponentiated dot products by the l_1 norm. During exponentiation, the large positive weights are inflated, exaggerating the importance of the corresponding values over all the other values, which likely causes a winner-take-all situation. Having multiple attention heads mitigates this problem, while having an MLP to transform the query offers the same flexibility.

To test this hypothesis, we tried squaring instead of exponentiating the dot products. We found that with linear query transformations, the training dynamics of single-head attention by squaring was similar to multi-head attention with softmax, which were both notably better than single-head attention with softmax. With MLPs for query, single-head attention learned faster than multi-head, and it was more stable with softmax than with squaring.

Although these findings supports our hypothesis, we believe the true efficacy of multi-head attention still needs to be investigate in a setup with a metric of success more meaningful than the validation loss and accuracy we used for this task. However we did consistently find scaling helpful for softmax. Note that in the case of normalized squares, scaling simply has no mathematical effect. In summary, we decided on single-head attention with softmax and scaling.

4.4 Final architecture

Figure 7 illustrates our revised transformer architecture. We used two encoder layers and two decoder layers. The dimension of hidden representations was 256 throughout the model, except for the inner relu layer of the MLPs, which was doubled to 512. This was adopted from the original transformer, where the inner dimensions were quadrupled but with no explanation given. We found that having a larger inner layer was effective in preventing the dying relu problem, where all neurons in a layer were updated to always produce zero activation during training which would render the whole model useless and unable to recover.

We experimented with input and output em-

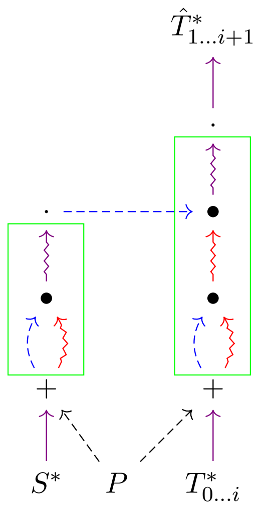


Figure 7: Revised transformer, cf. Figure 4.

bedding sharing (Press and Wolf 2016), which was used in the original transformer. It seemed to hurt the training dynamics of our character-level model, where these embedding layers had only a moderate amount of parameters. We also did not enforce parameter sharing between the two input embeddings.

5 Training and results

5.1 Training

Autoregressive networks with temporal updates are commonly trained using teacher forcing, a technique derived from the maximum likelihood criterion (Williams and Zipser 1989). Instead of feeding the outputs of the model back to itself to train for the next time step, the ground truth values are used as inputs. This way, the training for all time steps can take place in parallel. The attention weights can be computed for all steps against every other step simultaneously, both in the encoder and the decoder. However in the decoder, the upper triangle of the self-attention weight matrix has to be masked out in order to maintain the autoregressive structure. This is done by setting the unnormalized weights to $-\infty$ (Vaswani et al. 2017).

We discovered that masking is also necessary for the source sequences. In batched training, all sequences in one batch are padded to the same length. Ideally the model would learn to ignore the paddings, but we observed that this was not what happens. The trained model behaved differently when fed the same sequence with different numbers of paddings. Therefore we masked the padding positions for all attention layers.

Our training scheme is almost identical to the original transformer. The training objective is to minimize the cross entropy between the predictions and the ground truth targets with a smoothing rate of 0.1 (Szegedy et al. 2016). We used the Adam optimizer (Kingma and Ba 2014) with $\beta_1 = 0.9, \beta_2 = 0.98, \varepsilon = 10^{-9}$ and a learning rate $d^{-0.5} \min\left(s^{-0.5}, w^{-1.5}s\right)$ where s is the training step, $d = 256$ the model dimension, and $w = 4000$ the number of warmup steps. We also used a dropout rate of 0.1, but we did not apply dropout to the position encoding. We found that the model took much longer to reach the same validation loss and accuracy when the position encoding was dropped out, and it did not improve the results in the long run. We tried the time-sharing dropout scheme suggested for recurrent networks by Gal and Ghahramani (2016), but it did not seem to suit the transformer.

We experimented with other training schemes. With teacher forcing, the decoder only receives ground truth sequences during training, which might differ significantly from the autoregressively generated sequences which it depends on for inference. Therefore we tried training the model with free-running inputs. We ran the model autoregressively to produce a sequence, and used that sequence as the input to predict the truth sequence. We mixed the training schedule with teacher forcing. It was expected that the predictions would worsen in free-running mode, but in the long run, the model should become more robust. However that was not what we found. Free-running training always worsened the model, even when evaluated in teacher forcing mode.

We also tried training with backpropagation through time (BPTT). Since the softmax outputs describe the predicted distribution for the next character, we can multiply the probabilities with the embedding matrix for a weighted representation of the next input, and the computation for all steps can be chained for backpropagation. Training with BPTT is slow and expensive in memory usage, but it was also ineffective. The accumulation of uncertainty caused the prediction to degrade over time. the highest probability started from ~60% and dropped to ~15%. the second highest stayed ~10%. When the highest prediction dropped below 20%, the model would simply repeat a character, usually the whitespace.

As a result, we committed to teacher forcing training. We used a batch size of 64, and trained the model for 180 epochs. The training speed was ~10 iterations per second on a graphics card. One significant drawback of modeling at the character level is that the sequences are much longer. We suspected that it would be difficult to apply the transformer architecture to long sequences, due to its quadratic complexity with lengths. However thanks to its high parallelizability, it is still efficient for our task.

5.2 Results

We used eager decoding for the autoregressive model. The predicted character with the highest probability was taken as the next input. It produced a BLEU score of 29.59 on the validation set. Over 80% of the output sentences were identical to the source.

The model performs some basic text normalization. It replaces unknown characters with the non-breaking whitespace (A1, A2), adds or removes brackets for balancing (A2, A3), and has its own preferences for the format of punctuations (A4) and dates (A5).

Some source sentences are incomplete. The model always produces normal-looking sentences with initial capitalization and terminal punctuations. This sometimes reduces the incomplete sentence to a noun phrase (A6, A7), or a trivial sentence (A8, A9). It tries to make use of existing information, but the outcome is often unsatisfying (A10, A11).

For simplification, the model tends to remove the initial adverbial of a sentence (A12, A13, A14), but not always (A15). It removes components in the middle of a sentence which are usually omitted in the Simple English Wikipedia (A16, A17, A18). It also removes words with no additional information (A19, A20), and may add simple words such as particles (A21), or substitute some words with simpler ones (A22).

On rare occasions, the model behaves erratically (A23, A24). This happens when the decoding steps lose alignment with the source positions, as shown in Figure 8.

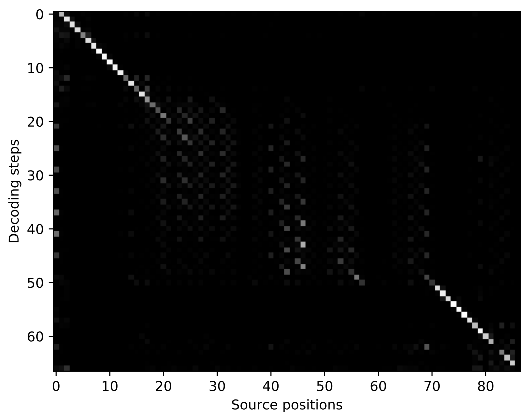


Figure 8: Attention alignment for example A23.

6 Conclusion

Our model was able to learn the common forms of English sentences, and the general differences between the two Wikipedias, as demonstrated by its text normalization capacities. However as a text simplification system, its performance is unsatisfying. It has only limited abilities to simplify whole constituents, often by deletion. It shows no ability for syntactic transformations and no understanding for semantic structures.

Nevertheless, we do not consider our architecture unsuitable for the task. As shown in Appendix A, many aligned sentence pairs differ in content. Even among the pairs which do correspond in content, the ground truth targets are rarely good simplifications for the sources. However wherever good examples exist, the model does produce the same or similar outputs. We need more suitable examples of sentence simplification with interesting linguistic properties.

References

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5, pp. 359–366.

Williams, Ronald J and David Zipser (1989). “A learning algorithm for continually running fully recurrent neural networks”. In: *Neural computation* 1.2, pp. 270–280.

Napoles, Courtney and Mark Dredze (2010). “Learning simple Wikipedia: A cogitation in ascertaining abecedarian language”. In: *Proceedings of the NAACL HLT 2010 Workshop on Computational Linguistics and Writing: Writing Processes and Authoring Aids*. Association for Computational Linguistics, pp. 42–50.

Coster, William and David Kauchak (2011). “Simple English Wikipedia: a new text simplification task”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*. Association for Computational Linguistics, pp. 665–669.

Wubben, Sander, Antal Van Den Bosch, and Emiel Krahmer (2012). “Sentence simplification by monolingual machine translation”. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics, pp. 1015–1024.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473*.

Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.

Narayan, Shashi and Claire Gardent (2014). "Hybrid simplification using deep semantics and machine translation". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1, pp. 435–445.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*, pp. 3104–3112.

Hwang, William et al. (2015). "Aligning sentences from standard wikipedia to simple wikipedia". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 211–217.

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning (2015). "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025*.

Štajner, Sanja, Hannah Béchara, and Horacio Saggion (2015). "A deeper exploration of the standard PB-SMT approach to text simplification and its evaluation". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Vol. 2, pp. 823–828.

Xu, Wei, Chris Callison-Burch, and Courtney Napoles (2015). "Problems in current text simplification research: New data can help". In: *Transactions of the Association of Computational Linguistics* 3.1, pp. 283–297.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). "Layer normalization". In: *arXiv preprint arXiv:1607.06450*.

Gal, Yarin and Zoubin Ghahramani (2016). "A theoretically grounded application of dropout in recurrent neural networks". In: *Advances in neural information processing systems*, pp. 1019–1027.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Kalchbrenner, Nal et al. (2016). "Neural machine translation in linear time". In: *arXiv preprint arXiv:1610.10099*.

Luong, Minh-Thang and Christopher D Manning (2016). "Achieving open vocabulary neural machine translation with hybrid word-character models". In: *arXiv preprint arXiv:1604.00788*.

Parikh, Ankur P et al. (2016). "A decomposable attention model for natural language inference". In: *arXiv preprint arXiv:1606.01933*.

Press, Ofir and Lior Wolf (2016). "Using the output embedding to improve language models". In: *arXiv preprint arXiv:1608.05859*.

Szegedy, Christian et al. (2016). "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

Wang, Tong et al. (2016). "An experimental study of lstm encoder-decoder model for text simplification". In: *arXiv preprint arXiv:1609.03663*.

Wu, Yonghui et al. (2016). "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144*.

Xu, Wei, Courtney Napoles, et al. (2016). “Optimizing statistical machine translation for text simplification”. In: *Transactions of the Association for Computational Linguistics* 4, pp. 401–415.

Daniluk, Michał et al. (2017). “Frustratingly short attention spans in neural language modeling”. In: *arXiv preprint arXiv:1702.04521*.

Gehring, Jonas et al. (2017). “Convolutional sequence to sequence learning”. In: *arXiv preprint arXiv:1705.03122*.

Vaswani, Ashish et al. (2017). “Attention is all you need”. In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Zhang, Yaoyuan et al. (2017). “A constrained sequence-to-sequence neural model for sentence simplification”. In: *arXiv preprint arXiv:1704.02312*.

Kitaev, Nikita and Dan Klein (2018). “Constituency Parsing with a Self-Attentive Encoder”. In: *arXiv preprint arXiv:1805.01052*.

Liu, Peter J et al. (2018). “Generating wikipedia by summarizing long sequences”. In: *arXiv preprint arXiv:1801.10198*.

Parmar, Niki et al. (2018). “Image Transformer”. In: *arXiv preprint arXiv:1802.05751*.

Vu, Tu et al. (2018). “Sentence Simplification with Memory-Augmented Neural Networks”. In: *arXiv preprint arXiv:1804.07445*.

Zhou, Shiyu, Linhao Dong, et al. (2018). “Syllable-Based Sequence-to-Sequence Speech Recognition with the Transformer in Mandarin Chinese”. In: *arXiv preprint arXiv:1804.10752*.

Zhou, Shiyu, Shuang Xu, and Bo Xu (2018). “Multilingual End-to-End Speech Recognition with A Single Transformer on Low-Resource Languages”. In: *arXiv preprint arXiv:1806.05059*.

A Output examples

Each output example is a triple of sentences: the source, the prediction, and the ground truth target. The examples are from the validation set, cf. Section 3.1.

Example 1

```
The enshrined kami is Isonotakeru no mikoto ( 五十猛命 ? )

The enshrined kami is Isonotakeru no mikoto (      ? )

This place is special to the kami named Isonotakeru no mikoto ( 五十猛命 ? )
```

Example 2

```
Edo ( 江户 ?

Edo (    ? )

Edo ( 江户 ? )
```

Example 3

```
Buddha-Bhagavan ) .

Buddha-Bhagavan .

Buddhists believe there have been many Buddhas who were alone , called pacceka-buddhas .
```

Example 4

```
Punch 's wife was originally called `` Joan . ''

Punch 's wife was originally called `` Joan '' .

Punch 's wife was originally `` Joan '' .
```

Example 5

```
Their first child , Prince George of Cambridge , was born on 22 July 2013 .

Their first child , Prince George of Cambridge , was born on July 22 , 2013 .

On July 24 , it was announced that the baby would be called Prince George of Cambridge ,
with the full given name George Alexander Louis .
```

Example 6

```
is a 1982 arcade-style platform video game by Nintendo .
```

```
In 1982 , the style platform video game by Nintendo .

Donkey Kong Jr. is an arcade video game made by Nintendo in 1982 .
```

Example 7

```
reflects the influence of the sexagenary cycle as a count of years .

The influence of the sexagenary cycle as a count of years .

Ancient records show that the sexagenary cycle was used to count years in the 3rd century B.C. .
```

Example 8

```
and Mario 's Time Machine .

The machine is a machine .

In the Dr. Mario games first seen in 1990 , Mario is a doctor .
```

Example 9

```
and Airplane II : The Sequel .

The Sequel is a sequel .

( 1980 ) and Airplane II : The Sequel ( 1982 ) .
```

Example 10

```
and it is because they do n't understand what Parkour is ; ` Who is the best ? '

They are the best ?

Star Jumping Parkour does involve risks and many people get injured every day .
```

Example 11

```
It is officially known as Avtomat Kalashnikova ( Russian : Автомат Калашникова ) .

It is known as Avtomat Kalashnikova ( Russian : : 1950 : 1998 ) .

The letters AK stand for Avtomat Kalashnikova , which is Russian for Kalashnikov 's Automatic Rifle .
```

Example 12

```
With one huge blow from his olive-wood club , Hercules killed the watchdog .

Hercules killed the watchdog .

Herakles killed her .
```

Example 13

For example , the speed of sound in gases depends on temperature .

The speed of sound in gases depends on temperature .

Loudness depends on sound intensity , sound frequency , and the person 's hearing .

Example 14

In it , Goldilocks is re-imagined as a girl in her 20s .

Goldilocks is re-imagined as a girl in her 20s .

She finally became Goldilocks sometime in the early 20th century .

Example 15

In 2008 , she starred in the comedy film Baby Mama , alongside former SNL co-star Amy Poehler .

In 2008 , she starred in the comedy movie Baby Mama , alongside former SNL co-star Amy Poehler .

In 2008 , she starred in the comedy movie Baby Mama , with Amy Poehler .

Example 16

The spines , which may be up to 50 mm (2 in) long , are modified hairs , mostly made of keratin .

The spines , which may be up to 50 mm long , are modified hairs , mostly made of keratin .

The spines , which may be up to 50 mm long , are modified hairs , mostly made of keratin .

Example 17

Sodium iodate (NaIO3) is the sodium salt of iodic acid .

Sodium iodate is the sodium salt of iodic acid .

Sodium iodate is a chemical compound .

Example 18

The palate / ˈpælɨt / is the roof of the mouth in humans and other mammals .

The palate is the roof of the mouth in humans and other mammals .

The palate is the roof of the mouth in humans and other mammals .

Example 19

It is located in the Market Square , Aylesbury , Buckinghamshire .

It is in the Market Square , Aylesbury , Buckinghamshire .

The King 's Head is a National Trust property in Aylesbury , Buckinghamshire , England , UK .

Example 20

Home Alone received mixed reviews from film critics .

Home Alone received mixed reviews from critics .

Home Alone 3 (1997) and Home Alone 5 : The Holiday Heist (2012)
have completely different characters and actors .

Example 21

However , low frequency also causes noticeable flicker in arc lamps and incandescent light bulbs .

The low frequency also causes noticeable flicker in arc lamps and incandescent light bulbs .

This happens so fast that a light bulb does not stop glowing .

Example 22

In the film 's story the location was named as Footscray station .

In the movie 's story the location was named as Footscray station .

In the movie it was called Footscray station .

Example 23

For example , 2 + 3 = 3 + 2 = 5 and 2 · 3 = 3 · 2 = 6 , but 23 = 8 , whereas 32 = 9 .

For example , 2 + 3 = 3 = 3 = 3 = 3 = 3 = 3 = 3 , whereas 32 = 9 .

Then people write powers using the ^ sign : 2 ^ 3 means .

Example 24

1 and − 1 divide (are divisors of) every integer .

In the 19th century , the state of the state of the state of the state of the state of the state
of the state .

For example , if the last digit is even (0 , 2 , 4 , 6 or 8) , then 2 is a divisor .