Learning a Loss Function for Text Simplification

Rebekah Cramerus, Malte Klingenberg, Paul Opuchlich

September 15, 2018

Abstract

Neural networks are a promising tool in many areas of machine learning, including natural language processing. Applying them to the field of automatic text simplification poses a problem, as most difficulty measures, for example parse tree-based measures, are not differentiable. However, a differentiable loss function is required for training a neural network using gradient descent algorithms. In this work, we present a neural network that is able to predict traditional difficulty measures with high accuracy. On a difficulty scale from 0 to 1, the network achieves a root mean square error of 0.021. Since the calculation performed by a neural network is a differentiable operation, this network can then be used as a loss function for other neural networks.

1 Introduction

Automatic text simplification is a current open problem in the field of natural language processing in which the goal is to take a text and transform it, while retaining the same meaning, into a more comprehensible and altogether simpler version of itself [Siddharthan, 2014] [Coster and Kauchak, 2011]. Text simplification has many applications, either as a step in preprocessing (for example, in text summarization or translation), or as a method of increasing accessibility to texts otherwise too complex for some readers. Automating this process is a new area of research in natural language processing (NLP), and various methodologies have been used in approaching it.

With the rise of neural networks in the field of machine learning, researchers now can apply these new models to text simplification. A problem, however, arises in the training step: training a neural network using gradient descent requires a differentiable loss function. In the case of text simplification, a loss function would analyze a text and determine its level of complexity. But typically the features which are used to calculate text complexity are not continuous, but defined stepwise - for example, counts of certain part of speech tags, or syntactic dependencies, which in themselves require a parser to find - and as such do not comprise a differentiable function.

Therefore, we chose to train a "helper" network to calculate complexity scores for given texts, for the purpose of being used as a loss function in a text simplifier neural network. In order to train this network, of course, data is required: texts with associated complexity scores. We found a lack of appropriate English language, labeled data in what was available. A lack of data in text simplification has previously hindered data-driven techniques, and many early attempts at the task employed handcrafted rules [Coster and Kauchak, 2011] [Siddharthan, 2014].

In that data that is available, sometimes the labels were discrete and not continuous, as in the Newsela texts, roughly divided by grade level, or in the Wikipedia Simple English corpus, with only two classes [Coster and Kauchak, 2011]. Most sources did not cover a wide enough range of difficulty. Newsela articles, for example, range from elementary to high school, which does not represent the whole range of text complexity in English. A network trained on this data might not be able to handle texts of complexities outside of its domain. Lastly, the features behind the given complexity scores in many datasets are opaque or undocumented. To serve as a loss function the helper network ideally would identify varied linguistic features - not, for example, only shallow features like sentence or word length, which can show high correlation with difficulty but do not encapsulate the syntactic or lexical background to complexity [Siddharthan, 2004]. Training a loss function on a dataset with explicit features behind the labels at least increases the chance that the neural network will converge to those same features. For these reasons, we built a rule-based system to determine text complexity, in order to gather a labeled dataset for use in training the loss function network.

The first part of our project was the creation of this rule-based function. We used existing literature to support our choice of features, wrote methods to extract those features from given texts, and transformed the resulting feature vec-

tors into continuous scores. We detail the results of this process in Section 3, naming the features we chose and how we extract them.

The second part of our project was the selection of an appropriate architecture for a neural network, followed by the training of this network on data with scores given by our complexity function. The texts we chose for training and the methodology behind the network are discussed in Section 4.

Finally, the results of our loss function neural network, analysis and conclusions are covered in Sections 5, 6 and 7.

2 Literature Review

2.1 Linguistic Research

Looking at previous literature was an integral part of our feature selection process. To properly determine text complexity, it was necessary that each of our chosen features have strong linguistic motivation. We looked at papers published on datasets for text simplification, previous approaches to the text simplification task, and experiments similar to ours on determining text complexity.

2.1.1 Available Data

In recent years research on text simplification has been dominated by Wikipedia Simple English and the different corpora it has produced [Siddharthan, 2014] [Coster and Kauchak, 2011]. The data provided by Wikipedia Simple English helped fill a lack that previously had defined the field. In 2015, Xu et al. published a paper asserting that the Wikipedia dataset is deficient and should no longer be considered the benchmark corpus for text simplification [Xu et al., 2015]. They analyzed the aligned dataset (by Zhu et al. 2010), and found that only 50% of the pairs were simplifications at all, and of those only 12% involved both deletion and paraphrasing, the other 38% being either one or the other. Because Simple English Wikipedia was collaboratively written by volunteer contributors with no specific guidelines or objectives, it is unreliable as a dataset. In response, Xu et al. assembled a new corpus based on the Newsela website, which stores a collection of news articles simplified to different levels. Unlike the Wikipedia corpus, which necessarily involves complex vocabulary in both simplicity levels due to its status as an encyclopedia, the Newsela corpus vocabulary drops by 50.8% at its simplest level. Because the articles are also consistent in length, the Newsela dataset also can allow document-level comparison - for example, on rhetorical structure. We reached out to Newsela requesting the dataset; however, lack of a timely response meant that we gathered the articles ourselves [Xu et al., 2015].

2.1.2 Previous Approaches

Siddharthan (2014) provides a comprehensive overview of research in the field of text simplification up to that point. Many systems focus either on syntactic, hand-crafted rules or on statistical models, although the divide between the two is mostly artificial and not necessarily binary. Syntactic simplification tends to involve a finite number of constructs, most commonly relative clauses, apposition, coordination, subordination and voice, and hand-crafted rules can cover this well. The switch to data-driven methods happens when lexical simplification is the focus (or syntactic rules with lexical components). In this case there are far too many possible substitutions (for example) to write out by hand. Word frequency statistics often are necessary, and metrics for word difficulty - some of which have been formulated using the Simple English Wikipedia corpus. Synonym substitution is a common method, and word ambiguity also must be taken into account. There have been recent attempts at hybrid approaches, using handwritten syntax plus rules acquired during machine learning for the lexical side [Siddharthan, 2014].

One of the earliest approaches was [Chandrasekar et al., 1996]: a hand-crafted system of syntactic rules for simplification. They simplified relative clauses, appositives and coordination. Another early approach was [Dras, 1999], which focused on complex verb constructions, clausal components, cleft constructions, and genitive constructions. Multiple changes happening in individual sentences could cause issues with coherence, though, and so the author chose to limit operations to one per sentence. This brought attention in the field to issues on discourse structure and coherence of a text as a whole [Siddharthan, 2014].

Other papers such as [Canning, 2002] and [Carroll et al., 1998] used parsers in addition to their hand-crafted rules, taking parse tree outputs and making the transformations there. They focused on coordination, passive voice, anaphora or pronoun replacement, and lexical tasks like synonym substitution, using word frequency to determine relative difficulty. Parsers, however, more so then than now, were slow and could time out. In Siddharthan (2004) a goal was to do simplification tasks without parsers, instead using machine learning techniques to identify clauses and attachments. Although parsers are faster now, it was an important find

that machine learning could perform as an alternative to parser use. The other purpose of Siddharthan (2004) was an analysis of the implications in discourse structure of syntax-based simplification, which had not often been studied before in favor of an only sentence-level approach. Siddharthan (2004) aimed to preserve text cohesion while attaining simplicity, using syntactic features such as relative clauses, apposition, coordination and subordination before using models of discourse structure to minimize disruption [Siddharthan, 2004] [Siddharthan, 2014].

More recent systems have approached text simplification as a translation task that exists in one language. The Simple English Wikipedia corpus, as mentioned, opened doors for many researchers in the search for data-driven simplification techniques. Many of these systems still use parsed tree-to-tree translation. [Candido Jr et al., 2009] works on Brazilian Portuguese, using hand-crafted rules on parse trees to change passive voice to active, rearrange clauses and subject-verb-object word order, and analyze the topicalization of adverbial phrases. However, with the increased use of dependency parsers, tree structures are no longer the only option. [Bott et al., 2012], working on Spanish instead of English, uses dependency parsing to simplify relative clauses, coordination and participle constructions. Siddharthan (2014) also uses dependency parsing, similarly working on relative clauses, apposition, voice conversion (passive to active), coordination, and quotation inversion [Siddharthan, 2014].

2.1.3 Evaluation and Readability Measures

There is still a lack of agreement in the field on how to evaluate text simplification systems. Two metrics used in evaluating machine translation output, BLEU and ROUGE, have been used on text simplification results, but there is still skepticism about these methods. A monolingual 'translation' (complex English to simple English) is a different case from a bilingual one (English to, say, French). Essentially, there are more ways to simplify a sentence than there are to translate one, and so fluency judgments may be trickier. Another common method used to evaluate simplified text is readability metrics, such as a text's Flesch score. The Flesch metric originally was published in 1951, and remains quite simple: it is calculated using the average number of syllables per hundred words in a text multiplied by the average number of words per sentence, and then mapped to a specific reading level. While it is simple, reliable and valid in the sense that it has been shown to match judgments by educators, Siddharthan asserts that the Flesch metric, like other readability metrics, does not directly measure complexity. Readability and comprehensibility are not the same thing, just as shorter words are not always simpler ones. The Flesch metric can be abused by purposefully creating short sentences with difficult words [Siddharthan, 2014].

Consensus on this topic does not seem to be fully reached, though. Despite the problems with readability vs. complexity, scores like the Flesch metric are still often used as evaluation metrics, including by Siddharthan in his earlier thesis [Siddharthan, 2004]. Xu et al. (2015) use features traditionally used in readability scoring, such as the number of characters per word and words per sentence, to support the claim that there is better and more appropriate simplification across levels in the Newsela dataset than in the Wikipedia Simple English dataset [Xu et al., 2015].

Finally, Aluisio et al. (2010) developed a readability assessment tool for Brazilian Portuguese meant to assist in the process of text simplification (specifically SIMPLIFICA, a Portuguese authoring tool) [Aluisio et al., 2010]. The authors at times use the two terms interchangeably, and any distinction drawn between them is unclear. They use machine learning techniques to train a classification model which assigns a label of rudimentary, basic or advanced to a text. A total of 59 features were chosen for their experiment. In their features were included those often used in readability metrics (referred to as 'cognitively motivated'), such as the Flesch index itself, counts of words, sentences, paragraphs, and average numbers of words in a sentence, syllables in a word, sentences in a paragraph. There were also syntactic constructions commonly used in text simplification research: NP modifiers, clauses, adverbial phrases, apposition, relative clauses, coordination, subordination, and passive voice. Lexical features such as type-token ratio, ambiguities, word frequencies and connectives were also involved, as well as a variety of features involving n-gram probabilities and perplexities. Feature experiments found that the strongest correlation to complexity (or readability) was found in the following features, in descending order: words per sentence, apposition, clauses, Flesch index, number of words before the main verb, sentences per paragraph, relative clauses, and syllables per word. Aluisio et al. contend that these different features can complement one another, and show that while they did split the features into subsets (cognitively motivated, n-grams, syntactic, and so on), the combination of all features

consistently yields better results than any of the subsets [Aluisio et al., 2010].

2.2 Text Simplification Using Neural Networks

Text Simplicity Scoring is a new topic in the neural network domain that is the reason why the number of papers dealing with this issue is quite limited. The few papers concerning simplification and neural networks like Rush et al. (2015) and Wang et al. (2016) deal with the creation of a more simple version of a difficult text [Wang et al., 2016], [Rush et al., 2015]. They tried to simplify a text by using a neural machine translation algorithm and summarization techniques with attention based neural networks, respectively. They use sequence to sequence models, which doesn't fit with our purposes of calculating a score.
Other models which cover our sequence-to-context problem better are text scoring techniques, such as essay scoring or sentiment analysis. They all are using word embeddings to replace the words by numerical vectors followed by convolutional networks to filter the amount of parameters. In the end several types of RNNs use the aggregated information to generate a score or a classification over labels. The next sections will discuss these approaches for the different levels. For a better notion for these architectures we show the architecture of Tang et al. (2015) in figure 1 [Tang et al., 2015].

2.2.1 Word Embedding

To make a string calculable by a neural network each approach has to embed it into a multidimensional representation. Alikaniotis et al. (2016) uses a augmented C&W model which learns the representation of a word in a text by its local context [Alikaniotis et al., 2016]. Tang et al. (2015) worked with a pre-trained word vector which is still adjusted by a linear layer [Tang et al., 2015].

Another commonly used approach is to use a Skip-gram model like in Lai et al. (2015), which essentially trains a small neural network with words as input and the surrounding words as the target variable. The learned fixed-length weights are then used as a word embedding [Lai et al., 2015].
An approach that highly reduces the number of possible inputs but also loses some semantic information is to use characters as input. Taghipour and Ng (2016) propose a way to aggregate all characters of a word using a convolutional network [Taghipour and Ng, 2016].

2.2.2 Aggregation

After each word has a numerical representation, the models reduce the number of parameters so that they can produce an actual score or classification afterwards. Most of the papers use convolutional neural networks (CNNs) before they apply recurrent neural networks (RNNs) to get the final labels. Taghipour and Ng (2016) justifies the use of CNNs by extracting local information of the words and using these windows of words as n-grams [Taghipour and Ng, 2016]. To reduce the dimensionality further Dong et al. (2017) proposes an attention pooling right after the convolutional layer, while Tang et al. (2015) uses a sequence-to-context RNN to squeeze the information of all word embeddings into one sentence composition [Dong et al., 2017], [Tang et al., 2015].
A completely different approach is used by Lai et al. (2015), which covers the representation of a word not only by a singular vector, but also through their "left context" and "right context", whose weight matrices are learned like filters in a convolution neural network. In the end all of these word embeddings and their left and right context are transformed by another hidden layer and aggregated by a max-pooling operation [Lai et al., 2015].

2.2.3 Combine Feature Aggregations

Finally, most of the elaborated papers use a RNN to generate the final score. This type of architecture can deal with sequences of variable length, which is a essential key in natural language processing. Taghipour and Ng (2016) and Dong et al. (2017) feed the sentence representations directly into a Long Short-Term Memory (LSTM) network, which is considered to be a good RNN for covering relations over long distances [Taghipour and Ng, 2016], [Dong et al., 2017]. It consist of 4 gates, which can control what information the neural network should keep from the input, which to forget, to output and to memorize [Hochreiter and Schmidhuber, 1997]. A special form of this network used by Alikaniotis et al. (2016) is a bi-directional LSTM, which not only takes into account information from the past, but also from the future [Alikaniotis et al., 2016].

The LSTM of Tang et al. (2015) is modified in a way such that the output gate is always on, so that according to them no semantic information of the previously seen sentences are discarded (also called a gated neural network). To calculate the final score they either used each output vector of the RNN or just the last one generated. Experiments showed that the difference in

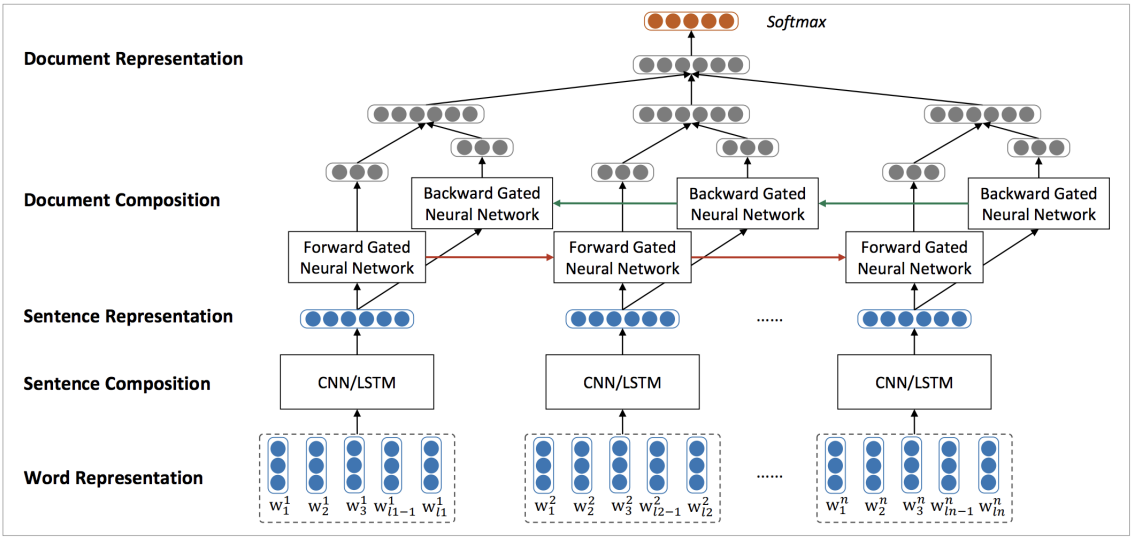

Figure 1: Architecture of Sentiment Classification proposed in [Tang et al., 2015]

accuracy is marginal [Tang et al., 2015].
The final layer of our neural network has to output a regression vector, so that it can be evaluated with the target score. Alikaniotis et al. (2016) concatenate the first and last representation of their bi-directional LSTM and transform them with a normal fully connected layer to the final predicted target vector [Alikaniotis et al., 2016]. The idea is that the first and last vector represent only the final information about the text. Another idea outlined by Dong et al. (2017) is either to use all RNN outputs and average them over time or to use a additional attention pooling layer, which focuses on each output to a different extent [Dong et al., 2017].

To optimize the loss function Taghipour and Ng (2016) and Alikaniotis et al. (2016) use RMSProp for a better calculation of the learning rate. The algorithm works by exponentially decaying the learning rate for each weight in relation to the past squared gradients [Taghipour and Ng, 2016], [Alikaniotis et al., 2016].

Dong et al. (2017) additionally used momentum, which takes the gradients of the previous steps into account for the calculation of the next gradient [Dong et al., 2017]. Thereby, the direction of the gradient is stabilized even if the surface of the loss function implies a change of the direction in each step. Applying both of these methods yields the AdamOptimizer [Kingma and Ba, 2014].
All these architecture perform quite well in relation to previous approaches in their task. We took several ideas out of their architectures to implement our neural network for text simplification, which is a similar task to essay scoring in that both produce a regression output in the end and use linguistical features as input. Nevertheless, our network needs to extract other information to finalize the score.

3 Rule-based Function for Difficulty Score

3.1 Data

We used the Newsela corpus, described in 2.1 as a dataset meant to improve upon the previously standard Wikipedia Simple English corpus, to train our model of text complexity. The Newsela corpus includes a large range of text levels, the lowest among them being suitable for lower elementary school students, according to their website. The original articles, given a Newsela scoring label as well, are also included.
Although the lower end of text simplicity is adequately covered by the Newsela corpus, we were unconvinced that the maximum levels of the corpus (generally the original news articles) would represent the higher degrees of complexity possible in language. For that reason, we added fifty passages taken manually from practice Law School Admission Tests (LSAT) used globally and in English [LSA, 2018b] [LSA, 2018c] [LSA, 2018a] [LSA, 2018d]. These passages are written intentionally to be very difficult, but must still be comprehensible. Therefore, we used them to represent the highest levels of difficulty in our training data.

3.2 Preprocessing

The Newsela corpus still required some cleaning before reaching a usable state. Line breaks and subheaders had to be removed, along with

HTML tags, non-word dividers, markdown segments such as image URLs, multiple whitespaces, and other general noise. After cleaning, the corpus and the LSAT texts require a preprocessing stage. We converted all words to lowercase, and then using spaCy found lemmas, part of speech (POS) tags, and dependency tags for all words.

3.3 Features

The relative newness of the field and the variety in previous literature that we found meant that there were many potential features that could be used. Features used by others ranged in number from two to 100: ours sit at a moderate 20, enumerated below in Table 1 [3.3].

Our features can be split up into different categories: syntactic, lexical, and those that, as previously discussed, are often used in readability metrics. Of the syntactic features, there are those which involve dependency tags, for which we used spaCy's dependency parser [Honnibal and Johnson, 2015], and there are those which use part-of-speech tags, for which we used the Natural Language Toolkit (NLTK) [Bird et al., 2009]. We had considered adding features related to text cohesion or coherence, namely discourse structure, but decided against it due to the lack of literature on what features exactly would be used.

In 2.1 we gave an overview of previous literature, focusing on the linguistic justification for the features that we chose. Next we provide a more precise definition for the chosen features and go into further detail on how each one was extracted.

3.3.1 Syntactic Features

Syntactic complexity is integral to determining complexity of a text as a whole; more syntactically dense sentences not only require more effort to parse, but also provide more semantic information. We chose to use simple part of speech tags and dependency parsing to obtain our features. All tags (dependency and part of speech) were generated using spaCy [Honnibal and Johnson, 2015].

Some of our features had straightforward corresponding dependency tags, specifically: **Coordination** (`cc`), **Apposition** (`appos`), **Parataxis** (`parataxis`), **Negation** (`neg`), and **Prepositional Phrases** (`prep`). **Passive Verbs** were counted as a sum of `nsubjpass` (for main verbs) and `csubjpass` (for clauses). **Auxiliary Verbs**, similarly, were a sum of `aux` and `auxpass`, so that passive auxiliary verbs were also counted.

Subordination was counted using the dependency tags `acl` (adjectival clauses, or clausal modifiers of nouns), `advcl` (adverbial clauses) and `relcl` (relative clauses). **Complements**, clauses which function as objects, are counted by `ccomp` (clausal complements with their own subjects) and `xcomp` (clausal complements with no subject). **Modifiers** were counted as a sum of `advmod` (adverbial modifiers), `amod` (adjectival modifiers), `nummod` (numeric modifiers), and `nmod` (noun modifiers).

Simple POS tags were collected according to the Penn Treebank POS tag system, for **Nouns**, **Verbs**, **Adjectives**, **Adverbs**, and **Pronouns**.

3.3.2 Lexical Features

Ideally, to measure how difficult the vocabulary of a given text is, one would have access to a difficulty score for every possible word. Then, it would be simple to obtain an overall lexical complexity score for a text. Unfortunately, this is at the moment unrealistic. There are many aspects of a word to take into account when measuring its difficulty: linguistic origins, length of the word, morphological structure and complexity, and frequency used in speech, among others.

We chose two straightforward measures to include as features: the **Type Token Ratio** and the **Basic English Ratio**.

The type-token ratio (TTR) is a standard to evaluate lexical diversity. It compares the unique number of words in a text (types) with the total number of words in the text (tokens). A high TTR implies more lexical diversity than a low value.

The basic English ratio, on the other hand, takes a dictionary of basic English words [Ogden, 1932]. It is a proportion of the number of basic words in the dictionary that appear in the text to the number of unique words in the text. When more difficult words are used in a text, this ratio decreases.

3.3.3 Readability Features

While we did not want to focus on traditional readability features, we still chose to include three major ones with the understanding that, as suggested in Aluisio et al., they can complement the usage of syntactic and lexical features [Aluisio et al., 2010]. We will later assess to which extent the neural network learned just from these features.

Mean Word Length is the number of characters per word. **Sentence Length** is the number of words per sentence. Finally, we counted the number of **Syllables per Sentence** using

Feature Type	Feature Name
Syntactic: Dependency Parsing	Subordination
	Complements
	Coordination
	Apposition
	Passive Verbs
	Parataxis
	Auxiliary Verbs
	Negation
	Prepositional Phrases
	Modifiers
Syntactic: POS Tags	Nouns
	Verbs
	Adjectives
	Adverbs
	Pronouns
Lexical	Type Token Ratio
	Basic English Ratio
Readability	Mean Word Length
	Syllables per Sentence
	Sentence Length

Table 1: Features for Text Complexity Function

the Python package Pyphen to hyphenate given words [Pyp, 2013].

3.4 Methods

To create the target variable that the supervised neural network needs we tried two approaches:

- Take the features directly as target variables. In this case, the neural network would have 20 output units, one for each feature.
- Calculate a regression score from all features and take this single value as the target.

3.4.1 Feature Normalization

As mentioned before, we used several LSAT texts to determine our upper limit for the text score. The calculated features of the LSAT texts were averaged and used as a maximum threshold in a min-max-normalization to squeeze every feature scale into a range between 0 and 1:

$$z_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

Each text can be theoretically really short, so that x_{min} will be 0 in most of the cases.

Newsela does not provide information on how they determine their difficulty scores. We therefore explored the relationship between the score given by Newsela and the sum of all our feature values. The relationship is shown in figure 2. There does seem to be a roughly linear relationship between the feature sum and the (normalized) Newsela score. We placed the LsAT texts on this linear regression curve (not shown in the figure) based on their feature sum and normalized the Newsela scores to the resulting maximum value.

3.4.2 Regression Score

The normalized feature sum already is a suitable target variable for our first approach (see 3.4). To increase the agreement between the feature values and the Newsela score, we performed a regression using common regression techniques. We experimented with Lasso-, Ridge- and Elastic Net-Regression; Ridge Regression gave the best results.

3.4.3 Optimization Criterion

$$\min_{\theta} \sum_{i=1}^{n} (x_i\theta - y_i)^2 + \eta\theta^2 \qquad (1)$$

The optimization criterion seen in equation 1 for Ridge-Regression is the minimization of the squared loss $(x_i\theta - y_i)^2$ and the L2 regularizer $\eta\theta^2$, which forces the values of the parameter vector θ to be as small as possible. η functions as a hyperparameter that determines the effect of the regularizer on the result [Gelman et al., 1995].

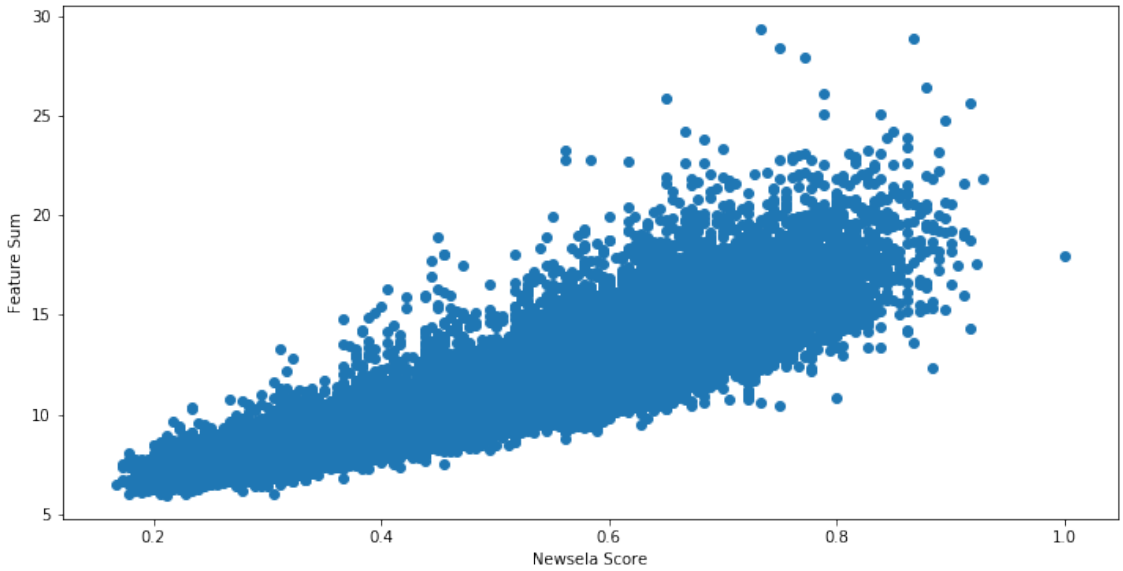

Figure 2: Relation between Score and Feature Sum

3.4.4 Evaluation Metric

$$R^2 = \frac{\text{Explained Variation}}{\text{Total Variation}} = \frac{\sum_{i=1}^{n}(y_i - f_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (2)$$

We took the commonly used coefficient of determination (R^2) as our evaluation metric [Dangeti, 2017]. The measure divides the explained variation (between true value and predicted value) by the total variation (between value and mean). The result explains the model variance compared to the data variance.

3.4.5 Cross Validation

To fine tune η we used generalized cross validation proposed by Golub et al. (1979), which shuffles the data and splits it at different positions into several train and test sets. A model is trained on each train/test pair with η and the performance will be averaged over all of them. This procedure is done for each given hyperparameter value and in the end, the best parameter is chosen by selecting the best average performance score [Golub et al., 1979].

3.4.6 Results

We trained the model on around 17000 texts and tested it on 4000 texts and received a R^2 of around 94%. This is a convincing result, indicating that the regression model can explain most of the observed variance.

3.4.7 Feature Coefficients

The features calculated for predicting a simplicity score seem to be good indicators for generating the Newsela score. To see which features were more important than others we looked at the feature coefficients, which are the entries of the vector θ (see 3.4.2).

Table 3.4.7 shows the feature parameters for the Ridge-Regression model with the largest values in bold, which are the sentence length, the number of syllables, the mean word length and the Basic English Ratio. If we remove these features from the regression the accuracy drops just by around 2% and other features seem to replace the most important features (nouns, verbs and adjectives).

3.4.8 Feature Correlations

Another assessment of each feature is their correlation matrix, which compares each feature with the others to show their behavior while changing the target variable. If their change is similar than the features tend to be correlated to each other. This matrix is included in the Python package, pandas [McKinney, 2010]. The matrix is shown in figure 3.
The Pearson Correlation Coefficient determines the linear relationship between two features. Red indicates a positive correlation, while blue indicates the opposite. Most of the features are slightly positively correlated. This makes sense, as all of the features measure some aspect of text difficulty.
The negative correlation between the basic english ratio and the other features is a result of the basic english ratio decreasing for a difficult text, while the other features increase with rising text difficulty.

Feature Name	Coefficient	Coefficient without strongest features
Subordination	0.04	0.01
Complements	0.01	-0.006
Coordination	0.03	-0.04
Apposition	-0.004	-0.0006
Passive Verbs	-0.005	0.001
Parataxis	0.001	0.001
Auxiliary Verbs	0.01	0.003
Negation	-0.01	-0.01
Prepositional Phrases	0.03	0.06
Modifiers	0.02	0.04
Nouns	0.11	**0.3**
Verbs	0.07	**0.2**
Adjectives	0.03	**0.08**
Adverbs	0.03	0.04
Pronouns	-0.01	-0.002
Type Token Ratio	-0.05	-0.07
Basic English Ratio	**-0.15**	-
Mean Word Length	**0.39**	-
Syllables per Sentence	**-0.41**	-
Sentence Length	**0.72**	-

Table 2: Features for Text Complexity Function

4 Neural Network Loss Function

4.1 Data

Due to the fact that a neural network cannot process text data directly we followed the approaches of Lai et al. (2015) and used a Skip-gram model to embed the words to a numerical feature space [Lai et al., 2015]. If a word ("king") appears in a similar content as another word ("queen") their feature vectors would point to the same region in the feature space. For this purpose we used the Word2Vec model proposed by Mikolov et al. (2013), which applies Skip-grams (predict the words around the current word) and continuous bag-of-word models (predict the current word by the surrounding words) [Mikolov et al., 2013].
After changing every word into a 50-sized vector (limited by the computational resources) we padded each text and sentence with zero-values. We used Tensorflow to implement the neural network, which requires input values of the same shape in each dimension, in our case number of sentences, number of words and number of embedding values [Abadi et al., 2015]. Due to the distribution of sentences per text and words per sentence (see figure 4), we decided to pad each text into a 100 times 100 times 50 vector (skipping each word and sentence behind this threshold) without losing information for most of the texts. Moreover, we applied a dynamic recurrent neural network structure, that ignores the applied padding, so that zero-values are not taken into account for learning.

4.2 Architecture

The architecture is strongly related to the approaches shown in chapter 2.2. Figure 5 gives an overview of our architecture. There are three main steps: sentence information retrieval, text information aggregation and scoring.

Sentence Information Retrieval
The first step consists of an LSTM cell (blue cell in the picture), which takes a word and a hidden state as its input and calculates another hidden state. This hidden state and the next word in the sentence deal then as a input for the next iteration. Finally, the last output vector for each sentence is taken for the next step. We decided against a CNN, like proposed in some of the papers, because our sentence length is shorter than 100 words, which is effortless manageable by a LSTM [Hochreiter et al., 2001].

Text Information Aggregation
After the previous LSTM cell incorporates all information from each sentence, a second LSTM cell (green) takes the last output of each sentence and learns the final representation of the text. Therefore, each word and each

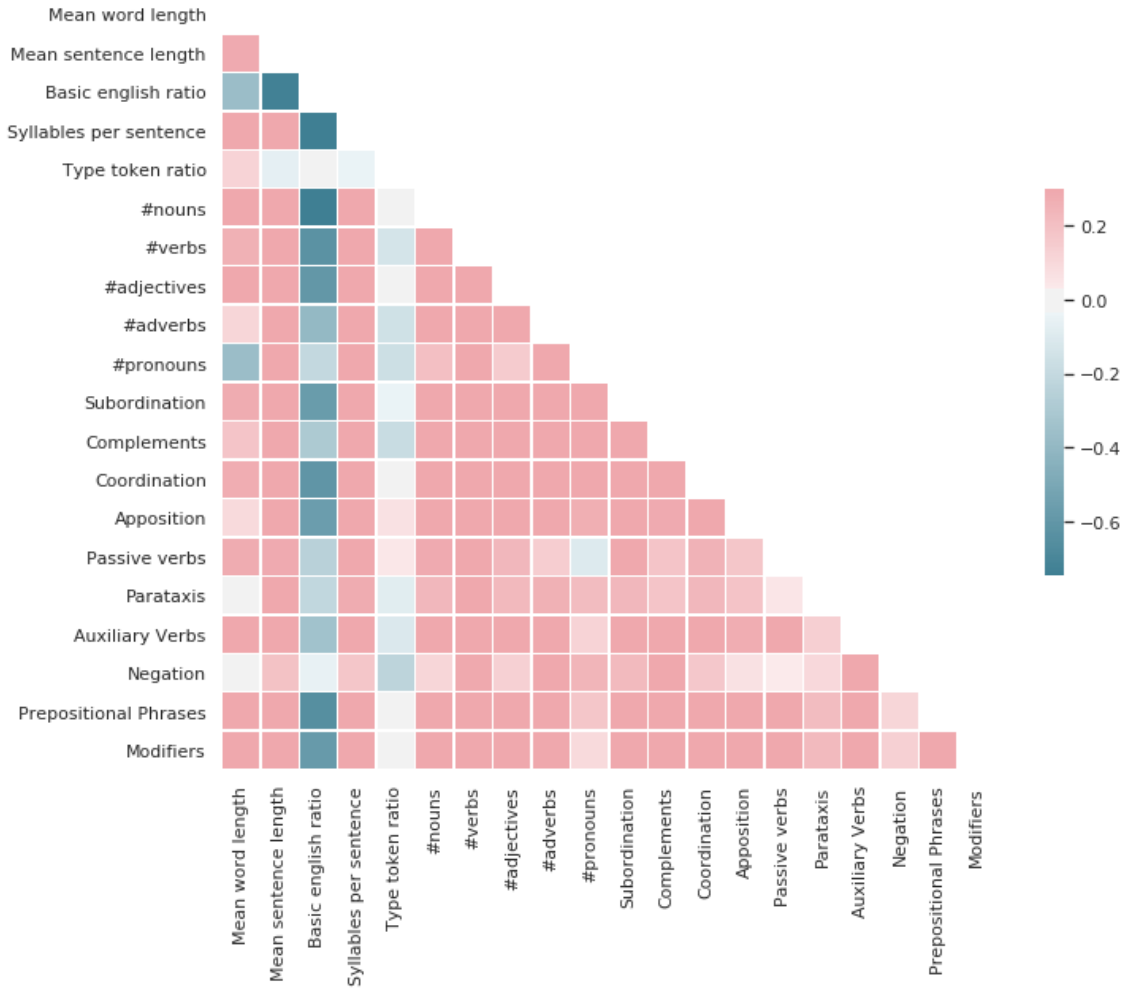

Figure 3: Pearson's r Feature Correlation Matrix

sentence can influence the final result.

Scoring
The last step takes the final output of the sentence LSTM cell and hand it over to a single dense layer (orange), which reduces the vector size to a single value (simplification score) or to the number of our selected features. To evaluate the result we compared the predicted scores with the actual scores using the mean squared error, which was also used in the mentioned papers. This loss function will be optimized by the Adam Optimizer, which incorporates RMSProp and Momentum (see 2.2.3).

5 Results

To train the neural network, we used the same train-test-ratio as for the regression model above, giving around 17000 texts for training and 4000 for evaluation. We used a stratified split to ensure that the difficulties of the training texts are distributed evenly. This prevents the neural network from adapting to a possibly uneven distribution.

We trained the network using mini-batch gradient descent, with a batch size of 10 and a learning rate of 0.01. Due to computational constrains, we only performed one epoch of training, which took around 27 hours. While in some scenarios this may lead to underfitting, the network performed very well, as shown below.

We first tried to predict the overall regression score as described in chapter 3.4. On this task, the network achieved an overall root mean square error of 0.021. The results are shown in more detail in figure 6. The evaluation texts were binned by their true difficulty. The x-axis shows the text difficulty, while the y-axis shows the difference between the true and predicted difficulties. The error in the predicted difficulty seems to be more or less independent of the actual difficulty. The only visible trend is a slight increase in the error for texts on the high end of the difficulty range. This is probably due to

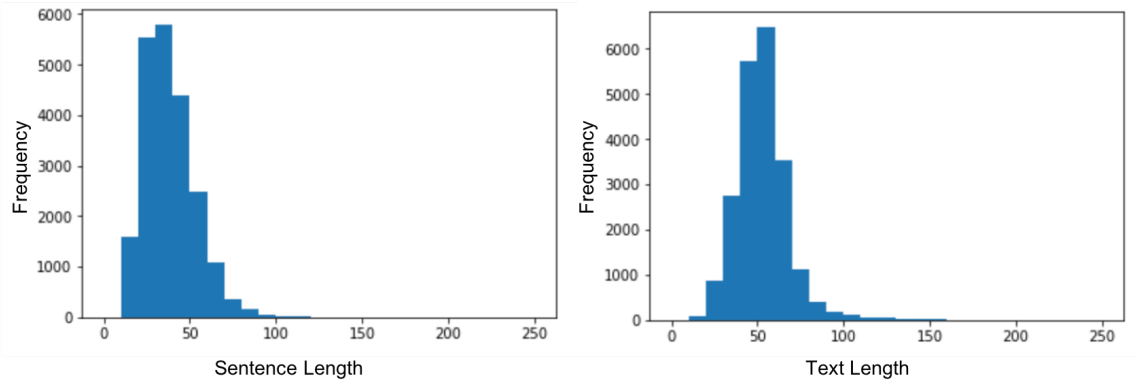

Figure 4: Distribution of Sentence and Text Length

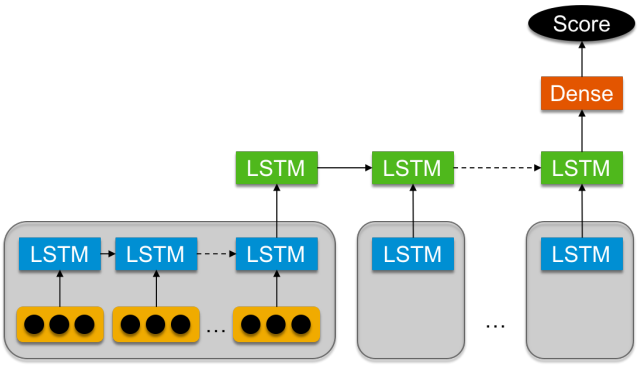

Figure 5: Final Architecture of the Neural Network

the higher variability of difficult texts, as a high difficulty can result from the presence of a wide variety of features (and types of features), but a low difficulty is simply caused by their absence.

As a second step, we trained the network using the individual features as the target values. The mean square error for the different features is shown in figure 7. As expected, the network performs best on simple structural features, such as the mean sentence length. Interestingly, the network is also able to predict the mean word length very well, even though we only provide it with a fixed-length representation of the words. The most likely explanation is that the network simply learned the correlation between the mean word length and the other features.
The network also performs well on most syntactical features, both the counts of part-of-speech tags and the dependency relations. It does, however, struggle with some of the dependency relations such as complements, appositions and negations, and is completely unable to pick up on parataxes. While it is to be expected that the syntactical features are harder for the network to learn than the structural features, the failure to identify parataxes might also result from their very sparse occurrences in the training data.
Even though the network was only trained for one epoch, it is clearly able to both identify the overall difficulty of the texts as well as many of the separate features with a high accuracy.

6 Analysis

Since the good performance of the network on the dependency relations might just have been a result of the network learning the easier features and the correlation between the different features, we analyzed the activations of the LSTM cells using the tool LSTMVis ([Strobelt et al., 2016]). This tool visualizes the activation of the recurrent units on the different input steps, and thereby allows to discover patterns in these activations.

The analysis was performed for the case of predicting the single difficulty value. We were unfortunately unable to discern any patterns relating to syntactical features. None of the recurrent units showed a consistent activation pattern for the syntactical features of the texts, at least some of which could be easily identifiable, for example negations marked by ”not”. It is likely that the reason for this is the short training process. One epoch is apparently not enough time for the units to specialize to detect

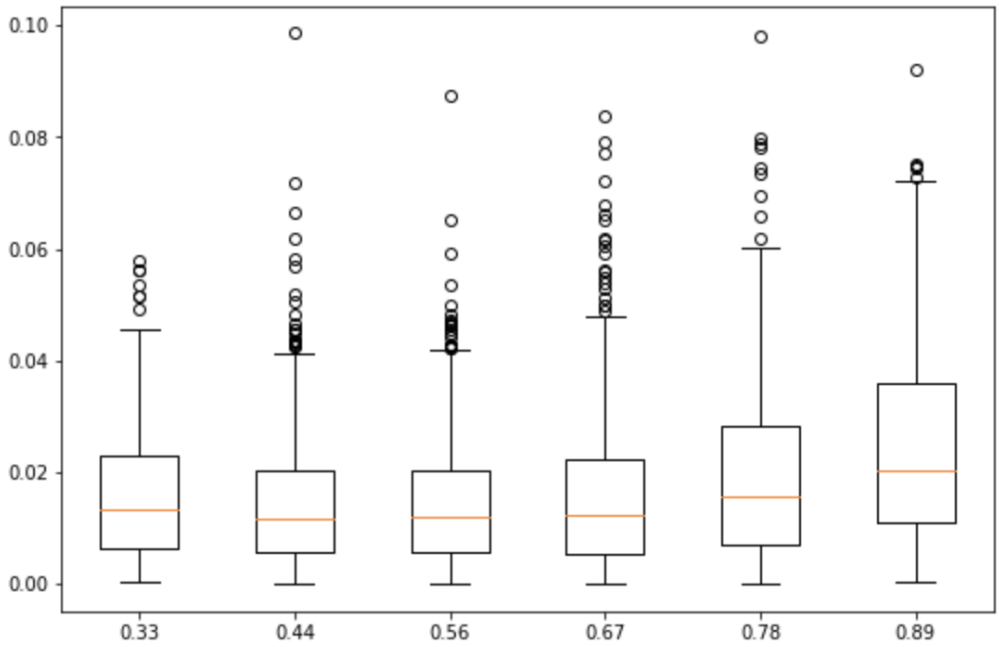

Figure 6: Bias of the predicted difficulty score for several difficulty bins

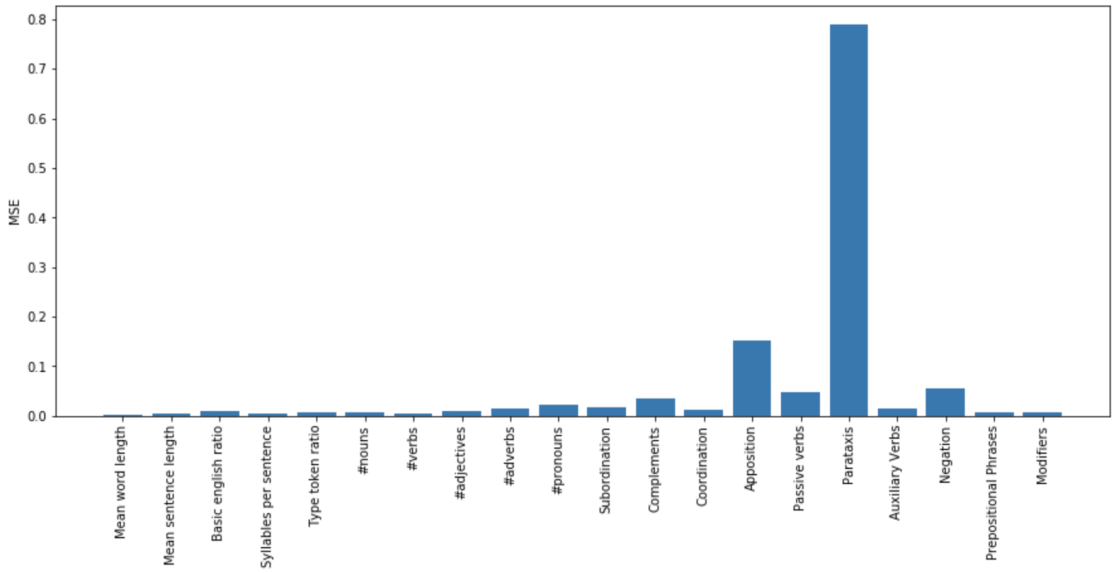

Figure 7: Mean square error values of the separate features

single features. The specialization might also be hindered by the two-layer recurrent layout. A longer training time might lead to detectable activation patterns.

As an alternative, we performed a manual analysis of the average activations of the first-layer recurrent cells after reading a sentence with a specific number of words. This is shown on the left in figure 8. While most of the units do not show any reaction to a variation in sentence length, there are some units for which a pattern is discernible. For some units, their activation increases as sentences get longer, while for some other units it decreases. The oscillations for high sentence lengths are due to a low sample size giving a rougher average.

Apparently, the network did learn to measure the sentence length to some extent. For comparison, the right side of figure 8 shows the average activation of the second-layer recurrent cells after processing entire texts with a varying number of sentences. While some units do show slight dependencies on the text length, these are very limited, especially compared to the clear

trends in the other diagram. This makes sense, as a text that contains only simple sentences will have a low difficulty score, no matter the number of simple sentences, while the number of words in a sentence does have a strong influence on the overall text difficulty.

7 Conclusion

We have shown that a neural network is capable of predicting the difficulty score of a given text. This allows for the application of neural networks in many problems regarding text simplification, as the network calculating the text difficulty can be used as a loss function for another neural network.

Future work could expand on this result in several ways. First, it would of course be interesting to train the network for more than one epoch to see whether the results improve further or whether patterns become detectable in LSTMVis. Different network architectures, for example including convolutional layers, could also be explored.

We saw in chapter 3.4 that a high sentence length is a strong indicator for high text difficulty. Since this is feature that is easily learned by a neural network, it could be interesting to try a way to penalize the network for focusing on the sentence length alone. This might give further insights into how the network picks up on syntactical features.

References

[LSA, 2018c] (2007-2018; accessed May 2018c). *LSAT Practice Tests*. Varsity Tutors.

[Pyp, 2013] (2013). Pyphen: Hyphenation in pure Python. [Online; accessed May 2018].

[LSA, 2018a] (2018; accessed May 2018a). *Free LSAT Practice*. Kaplan, Inc.

[LSA, 2018b] (June 2007; accessed May 2018b). *Law School Admission Test*. Law School Admission Council, Inc.

[LSA, 2018d] (March 2004; accessed May 2018d). *LSAT Sample Practice Questions*. Blue Coast Math Tutoring.

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

[Alikaniotis et al., 2016] Alikaniotis, D., Yannakoudakis, H., and Rei, M. (2016). Automatic text scoring using neural networks. *arXiv preprint arXiv:1606.04289*.

[Aluisio et al., 2010] Aluisio, S., Specia, L., Gasperin, C., and Scarton, C. (2010). Readability assessment for text simplification. In *Proceedings of the NAACL LT 2010 Fifth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 1–9.

[Bird et al., 2009] Bird, S., Loper, E., and Klein, E. (2009). Natural language processing with python. O'Reilly Media Inc.

[Bott et al., 2012] Bott, S., Saggion, H., and Mille, S. (2012). Text simplification tools for spanish. In *LREC*, pages 1665–1671.

[Candido Jr et al., 2009] Candido Jr, A., Maziero, E., Gasperin, C., Pardo, T. A., Specia, L., and Aluisio, S. M. (2009). Supporting the adaptation of texts for poor literacy readers: a text simplification editor for brazilian portuguese. In *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 34–42. Association for Computational Linguistics.

[Canning, 2002] Canning, Y. M. (2002). *Syntactic simplification of Text*. PhD thesis, University of Sunderland.

[Carroll et al., 1998] Carroll, J., Minnen, G., Canning, Y., Devlin, S., and Tait, J. (1998). Practical simplification of english newspaper text to assist aphasic readers. In *Proceedings of the AAAI-98 Workshop on Integrating Artificial Intelligence and Assistive Technology*, pages 7–10.

[Chandrasekar et al., 1996] Chandrasekar, R., Doran, C., and Srinivas, B. (1996). Motivations and methods for text simplification. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 1041–1044. Association for Computational Linguistics.

[Coster and Kauchak, 2011] Coster, W. and Kauchak, D. (2011). Simple english wikipedia:

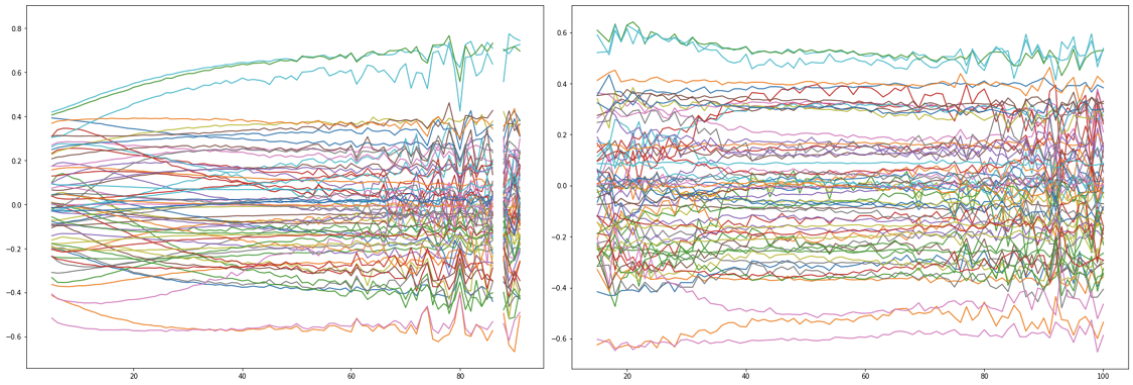

Figure 8: Mean activation of the first-layer LSTM cells for varying sentence lenghts (left) and second-layer LSTM cells for varying text lenghts (right)

A new text simplification task. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics:shortpapers*, pages 665–669. Association for Computational Linguistics.

[Dangeti, 2017] Dangeti, P. (2017). *Statistics for Machine Learning*. Packt Publishing.

[Dong et al., 2017] Dong, F., Zhang, Y., and Yang, J. (2017). Attention-based recurrent convolutional neural network for automatic essay scoring. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 153–162.

[Dras, 1999] Dras, M. (1999). *Tree adjoining grammar and the reluctant paraphrasing of text*. Macquarie University Sydney.

[Gelman et al., 1995] Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (1995). *Bayesian data analysis*. Chapman and Hall/CRC.

[Golub et al., 1979] Golub, G. H., Heath, M., and Wahba, G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223.

[Hochreiter et al., 2001] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8).

[Honnibal and Johnson, 2015] Honnibal, M. and Johnson, M. (2015). An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal. Association for Computational Linguistics.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Lai et al., 2015] Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. In *AAAI*, volume 333, pages 2267–2273.

[McKinney, 2010] McKinney, W. (2010). Data structures for statistical computing in python. In van der Walt, S. and Millman, J., editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Ogden, 1932] Ogden, C. (1932). *The A B C of Basic English (in Basic)*. K. Paul, Trench, Trubner.

[Rush et al., 2015] Rush, A. M., Chopra, S., and Weston, J. (2015). A neural attention model for abstractive sentence summarization. *CoRR*.

[Siddharthan, 2004] Siddharthan, A. (2004). Syntactic simplification and text cohesion. University of Cambridge.

[Siddharthan, 2014] Siddharthan, A. (2014). A survey of research on text simplification. In *Special issue of International Journal of Applied Linguistics, 165(2)*.

[Strobelt et al., 2016] Strobelt, H., Gehrmann, S., Huber, B., Pfister, H., and Rush, A. M.

(2016). Visual analysis of hidden state dynamics in recurrent neural networks. *CoRR*, abs/1606.07461.

[Taghipour and Ng, 2016] Taghipour, K. and Ng, H. T. (2016). A neural approach to automated essay scoring. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1882–1891.

[Tang et al., 2015] Tang, D., Qin, B., and Liu, T. (2015). Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432.

[Wang et al., 2016] Wang, T., Chen, P., Rochford, J., and Qiang, J. (2016). Text simplification using neural machine translation.

[Xu et al., 2015] Xu, W., Callison-Burch, C., and Napoles, C. (2015). Problems in current text simplification research: New data can help. TACL.