Implementation project
Neural Networks

SoSe2020

Lecturer: Michael Staniek

Controlling Linguistic Style Aspects in Neural Language Generation

implementation of *Ficler, Goldberg (2017)*

Denis Logvinenko
Department of Computational Linguistics
Heidelberg University, Germany
logvinenko@cl.uni-heidelberg.de

1 Introduction

The main contribution of *Ficler et al. (2017)* [1] was the introduction of a language model that can be trained while being conditioned on some metadata extracted from the training data set. This can help ensure that various linguistic style aspects derived from these annotations be preserved in the generated text.

The authors used a corpus based on the Rotten-Tomatoes website, where they collected 1,002,625 movie reviews.

The aim of this work is mainly to reimplement their approach and test other parameters, such as different LSTM architectures and smaller vocabulary sizes, especially using another corpus to ascertain whether the proposed method is scalable enough and can be replicated in other settings.

2 Data set

In scope of this work a subset of the Amazon movie review data set collected by Stanford SNAP Group (~8 million reviews) [2] [3] was taken for training a language model (cf. section 1.2 of the *Jupyter notebook*). The subset consists of 50,000 randomly selected professional and 50,000 unprofessional reviews (all reviewers who wrote more than 100 reviews were rated as professional). The benefit of the SNAP data set and the reason it was selected is that each review in it is annotated with 7 descriptions including those that are relevant for implementing the approach of *Ficler et al. (2017)*:

- `review/userId`, which is used to count all reviews associated with the given user ID. This is useful to determine if a user is a professional reviewer (based on the heuristic that the more reviews a user produces, the more probable it is that his/her text is written in a more professional style.)
- `review/score` from 1 to 5, which tells us whether the review is positive (4 or 5), neutral (3) or negative (1 or 2).
- `review/text` – the text of the review.

3 Metadata

The authors state that the most challenging part consists in obtaining all the necessary content derived annotations. However, the task can be largely simplified by leveraging the potential of modern high-performant NLP-libraries, such as spaCy.

The metadata were collected according to the definitions in the original paper, except for the label `descriptive`, for which a lower boundary was selected (*Ficler et al. (2017)* labelled a sentence as `descriptive:true` if at least 35% of its part-of-speech tags were adjectives). In this work one has come to the conclusion that it is a very unrealistic heuristic (especially for English sentences, as there are too many function words, such as articles, prepositions, auxiliary words etc. that constitute a great proportion of POS-tags). As there weren't enough sentences for the model to learn from using sentences with at least 35% adjectives, the boundary was set to 25%.

Before labelling the sentences, the original SNAP-corpus was converted (cf. section 1.2 of the *Jupyter notebook*) to a more easily parsed *tsv*-format with the following columns:

1. User ID
2. Review
3. Review text

In total, 18 different labels were used in *Ficler et al. (2017)* to encode linguistic aspects of the sentences. To avoid dealing with categorical variables later when defining the LSTM's feature embedding layer, an integer from 0 to 17 was used as an annotation (this helps indexing the embedding matrix later on.)

The sentence labelling pipeline comprises the following steps (cf. section 2.1 of the *Jupyter notebook* for details):

1. A user id is extracted from the first column of the *tsv*-file. Using the ID, we can check, how many reviews this user has written by looking it up in the dictionary created earlier. If this number is ≥ 100 reviews, then all the sentences in the review are deemed professional `(professional:0-1.)`
2. A review score is extracted from the second column and converted to fall within the range of 3 values: negative, neutral, positive `(sentiment:2-4.)`
3. After some cleaning, the review text has to be split into sentences. *spaCy's* statistical sentencizer is used for this separately from the main *nlp.pipe()*. The results are surprisingly slightly better (and also faster) than the ones obtained by a more sophisticated default sentencizer that utilizes dependency parsing to determine sentence boundaries.
4. After that the list of sentences is fed to *nlp.pipe(),* which, among a million other things that one should disable to get fast processing (such as *'parser', 'ner', 'textcat'*), tokenizes the input and assigns POS-tags to the tokens. From the former we acquire the sentence length `(length:5-8)`, and from the latter whether the sentence is descriptive according to the heuristic described above `(descriptive:16-17.)`
5. Using the list of tokens, we also check if the sentence is written in a personal voice, i.e. if it contains 'I' or 'my' `(personal:9-10)`, and determine the main theme of the sentence: here we verify which category of the predefined list of words created by the authors of the original paper is the most prevalent in the sentence `(theme:11-15.)`
6. After this analysis we save the results again as a *tsv*-file with 7 columns: Sentence, Professional, Sentiment, Length, Personal, Theme, Descriptive.

Below, the distribution of the metadata labels in the subset of the SNAP-data set is shown:

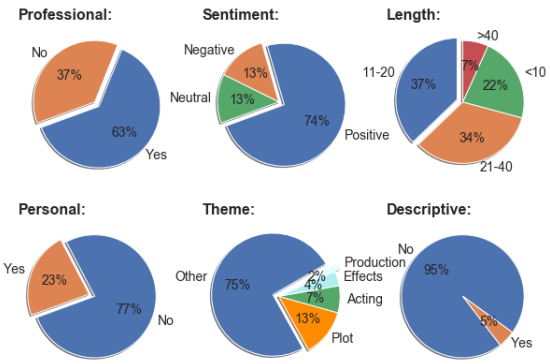

Figure 1: Distribution of the sentence annotations

4 Model

The language model is created using an LSTM [4], a technology that has already proved its worth in language generation. Some technical details (cf. sections 4.2 and 4.3 of the *Jupyter notebook* for more details) about the network architecture, two variations of which will be discussed later in this section:

- All text is divided into subwords using Byte Pair Encoding, which are later converted to numbers using the BPE [5] dictionary. These numbers index the embedding layer later on.
- The model has 3 LSTM layers, all of which comprise 1024 cells.
- These layers are followed by a fully connected layer of size 1024.
- The embedding dimensions are 256 for subwords and 20 for each of the 6 metadata features (resulting in an embedding of size 376).

In total, two different approaches to the model architecture were used and compared against one another (subsequently they are called model 1 and model 2 respectively):

1. All corpus sentences are merged together to a single text, which is divided into N batches, whereas all batches are M subwords long (160 in the tests conducted in this work.)

   In theory, the main problem with this approach is that there are batches, where the model will predict subwords, which are not in the same review. As all of the reviews are independent of each other, this can be theoretically problematic. Example: the model uses the linguistic context of review 1 to predict the next 160 words. If len(review 1) ≤ 160, then the network will also compute loss on the targets that are not in the scope of review 1 and adjust its weights correspondingly.

2. To prevent possible issues discussed in the section above, yet another approach was adopted: We use N batches, where each one consists of exactly one review, whereas the others are padded to the length of the longest review. As there are some very long reviews and also comparatively short ones, it is imperative to sort them by their length to avoid unnecessary padding (that's why shuffling was implemented but not tested in the scope of this work). If the longest review contains more than 160 subwords, batches are also divided into len(longest_review) // 160 minibatches.

   Some technical details: the sequences also have to be packed and the PAD-tokens should be excluded from the computation of loss, so that the network doesn't learn on PAD-tokens and doesn't make unnecessary computations (there are also cases where one of the minibatches is filled completely with zeros, with which PyTorch can't work directly, so it has to be clamped to the length of 1 first, and then the corresponding row of the LSTM hidden layer matrix can be filled with zeros).

5 Results

The quality of the language models is assessed using perplexity, as suggested in the original paper and can be seen in the following table:

Model 1	Model 2
161.68	501.74

Table 1: Perplexity of the 2 best language models

Somewhat surprising is the fact that model 1, albeit theoretically more naïve, actually provides better performance and, as we will see in the next section, by far more coherent and cohesive sentences, where one can also tell that they vary linguistically according to the provided metadata, which was the main purpose of the original paper in the first place.

6 Examples

In the following seven random examples (the first one would be the "default value", where only one metadata feature at a step is changed to see, whether it hat any influence on the generated text) are sampled from each model. In these examples the temperature is set to 0.6. Every example is meant to have two sentences (measured in the number of predicted <EOS> tokens) in it or be 300 subwords long. The priming sequence is "this movie"

6.1 Model 1

1	*Metadata:*	pro, +, 21-40, pers, plot, descr
	Generated:	this movie to the top of my favorite days. the main story of a man is a small interest in the movie but i realize that the movie was very good.
2	*Metadata:*	nopro, +, 21-40, pers, plot, descr
	Generated:	this movie is a classic movie and a great film with great action, and a very well written story. the story of a film that is not really my favorite of the movie and i enjoyed it so much.
3	*Metadata:*	pro, -, 21-40, pers, plot, descr
	Generated:	this movie is the case where i are in the mood for a good movie. i would buy it with all the flaws get the same character to the point of a end or a good movie.
4	*Metadata:*	pro, +, ≤10, pers, plot, descr
	Generated:	this movie to all the people. the characters are not real and i have no pleasure
5	*Metadata:*	pro, +, 21-40, nopers, plot, descr
	Generated:	this movie is to be the best western that it is written with a classic score. and the story line is not a real solid thriller which is one of the best films of the year.
6	*Metadata:*	pro, +, 21-40, pers, effects, descr
	Generated:	this movie to all of the films of the 00's. as i said the first one is good, and the song is good and the one which is a bit of the show.
7	*Metadata:*	pro, +, 21-40, pers, plot, nodescr
	Generated:	this movie the good, the worst comedy had the characters that i have ever seen, but it is great. i would recommend this movie to anyone who likes this film, so much that you will want to enjoy the story of the film and the best part of the film.

Table 2: Examples of text generated by model 1

6.2 Model 2

1	*Metadata:*	pro, +, 21-40, pers, plot, descr
	Generated:	this movie came for the film a bit of the movie
2	*Metadata:*	nopro, +, 21-40, pers, plot, descr
	Generated:	this movie is a great movie that it has to the story the plot
3	*Metadata:*	pro, -, 21-40, pers, plot, descr
	Generated:	this movie has a good story several reviewers get the movie
4	*Metadata:*	pro, +, ≤10, pers, plot, descr
	Generated:	this movie is the great line it is the story
5	*Metadata:*	pro, +, 21-40, nopers, plot, descr
	Generated:	this movie has great plot of <a hrefhttp a good story of 0 review
6	*Metadata:*	pro, +, 21-40, pers, effects, descr
	Generated:	this movie is just a touching plot a really good dvd is a good movie
7	*Metadata:*	pro, +, 21-40, pers, plot, nodescr
	Generated:	this movie was a good movie that in my opinion one thing i say

Table 3: Examples of text generated by model 2

It becomes obvious that model 2 is neither capable of generating coherent text, nor text with controllable linguistic style aspects (it also didn't learn to make sentence boundaries clear by generating some punctuation mark rather than merely the <EOS>-token), whereas the generated sentences from model 1 are clearly not the most cohesive or coherent ones, but they are at least examples of text that can be read and understood by a human being. This model also makes the necessary adjustments as required by the metadata in most cases.

7 References

[1] J. Ficler und Y. Goldberg, „Controlling Linguistic Style Aspects in Neural Language Generation,“ *CoRR,* Bd. abs/1707.02633, 2017.

[2] J. Leskovec und A. Krevl, *SNAP Datasets: Stanford Large Network Dataset Collection,* 2014.

[3] J. J. McAuley und J. Leskovec, „From Amateurs to Connoisseurs: Modeling the Evolution of User Expertise through Online Reviews,“ *CoRR,* Bd. abs/1303.4402, 2013.

[4] S. Hochreiter und J. Schmidhuber, „Long Short-Term Memory,“ *Neural Computation,* Bd. 9, p. 1735–1780, 1997.

[5] R. Sennrich, B. Haddow und A. Birch, „Neural Machine Translation of Rare Words with Subword Units,“ *CoRR,* Bd. abs/1508.07909, 2015.