NLP Task3: Automatic Political Speech Writer

G. Breiner, V. Rupprecht, R. Schotola, and O. Stritzel

Technische Universität, Wien, Austria
`{e1447920, e01525090, e01225193, e11920598}@student.tuwien.ac.at`

1 Task Description

The task was to create an automatic political speech generator. For this task we opted to look at 3 different approaches to generate speeches: A classical n-gram language model based on probability chains, a character-level predictor and a neural network encoder/decoder architecture based on LSTMs. We trained each of these algorithms on 4 different presidential candidates: Ronald Reagan, George W. Bush, Barack Obama and Donald Trump. We put the data from Millercenter[1] into .txt files. These were the raw inputs for each of the respective text generation pipelines. The website provides around 50 presidential speeches per president, all of them well spread throughout the presidential periods of their orators.

2 Algorithms

2.1 Character Prediction

A length of the history determines the size of the window that is analyzed before each character. A starting character is prepended on the start of each speech to indicate starts. The trained model computes probabilities for a character following a prepending history. These probabilities are used to create letters, given the desired length of the text. Each letter is selected via a random value and their probability that the letter occurs after the history given. This ultimately leads to words and sentences. Therefore, words that are not in the original text can be created in theory, based on the given history lengths.

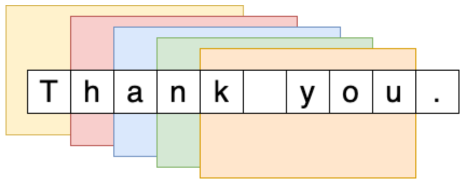

Fig. 1: Illustration of History and next character for a window size of 5

2.2 N-Gram Language Model

We also implemented a n-gram language model, that not only considers text, but also the grammatical structure (POS tags). Compared to the Character Prediction model, the n-gram model works with word tokens, instead of character tokens. In the training phase, the speeches are tokenized, POS tagged, and the number of occurrences of each n-gram, (n-1)-gram, ..., and unigram are counted. For the speech generation we first determine the grammatical function (POS tag) and then based on that the word token. With this we ensure that the generated speech is grammatically correct and the choice of words is based on that.

2.3 LSTM Encoder & Decoder

This is a fairly standard encoder decoder network architecture implemented using torchtext. The network was trained on a word-level basis. The input is given as a window of words and the prediction is the next word in the sequence. In order to produce words the network adds some noise to the word probabilities, then chooses the most likely word for the input sequence. Afterwards it moves the window and repeats the process.

[1] https://millercenter.org/the-presidency/presidential-speeches

3 Evaluation

- **TFIDF Cosine Similarity**: Describes how similar the importance of all words is within the the original speeches as compared to the generated speeches. A higher value is better (max 1).
- **TFIDF Rank Distance**: Describes how different the ordering of the top 15 words within the speeches is to each other. A lower value is better (min 0).
- **ROUGE Score**: Describes how many words of the original appear in the generated text and vice versa. A higher value is better (max 1).
- **Mean Sentence Length Ratio**: Describes how similar the length of generated sentences is to the length of original sentences. The closer to 1 the better.
- **Mean Word Length Ratio**: Describes how similar the length of generated words is to the length of original words. The closer to 1 the better.

4 Results & Discussion

4.1 Quantitative

	tfidf_cosine	tfidf_distance	rouge	mean_sentence_len_ratio	mean_word_len_ratio
lstm	0.049	559.239	0.401	0.682	0.711
charpred	0.094	4681.692	0.571	0.990	0.997
ngram	0.090	5286.658	0.433	0.889	1.033

CharPred scored best overall with NGram very close behind. LSTM was by far the best at identifying the most important words but generally ended up producing words and sentences that were quite a bit shorter than they should.

4.2 Qualitative

When looking at how natural the generated text sounds, the character prediction model comes out on top with some very lifelike speech-parts, although the generated speeches still lack in coherence. The LSTM network in contrast is able to produce more coherent speeches, keeping a topic over several sentences, but does not reach the authentic feel of language, that the character predictor can produce.

4.3 Run time

Considering run time we observed that character prediction is by far the fastest, being able to train on a single presidents speeches in a matter of seconds and generating a full speech in near instant's notice. The classical n-gram model trains around half a minute per president and generates a speech in a minute. The slowest model by far was the LSTM network training 10 minutes per president and generating a single speech in a minute.

5 Conclusion

Overall character prediction really surprised us with its great performance both on a time scale as well on the quality of the generated text, especially with usage of punctuation marks. Both other approaches also have their strengths, but given the extreme simplicity of the character prediction model and its great performance we would rank it as best overall.