CABbAGE

An AI Generated Pink Noise Approach to Improve Sleep

Emelia Blankenship Ike Lyons Jacob Shomstein Nicole Gizzo
Sam Cohen Sen Francis

December 7th, 2020

Abstract

The CABbAGE project aims to improve sleep though using a AI generated pink noise, variable frequency noise generated to promote better sleep. Given the novel research surrounding sleep and white/pink noise. As a type of pink noise, through personal findings the project aims to experiment with podcast-like noise and compare it sleep-performance of standard white and pink noise. To generate the audio, CABbAGE uses a range of generative techniques like LSTM and Markov Models trained on a variety of seed data ranging from Science and Tech, Zen and Nature and Sports.

1 Motivation

Sam likes noise while he sleeps so he likes idea.

FILL IN

2 Speech Text Generative Approaches

Given the lack of certainty around which model could produce text, and in turn speech, that would promote sleep the best. We opted to implement more than models that would perform text generation on our training set. This would show us the drawbacks and potential upsides of chosing one model over another.

For the two types of models, we decided to use a Markov Model approach along with a LSTM Deep Learning approach.

2.1 Training Data

We sourecd our traning data from ...

podcast and so on

types of podcasts

FILL IN

2.2 Markov Model

The work and science behind this.

FILL IN

2.3 LSTM Apprach

The second model that we used for text generation was an LSTM (Long Short Term Memory) approach where we had a LSTM layer with 64 neurons. By using a type of a recurring neural network, we were able to input a sequence of inputs as data. The inputs that we used to the model were normalized indices of word tokens from the body of text. We chose to the last 10 tokens as the input sequence and the output vector was a softmax layer.

For training, we created sequences of the past 10 tokens and for the label or target we used the following word. This way, the model would be able to avoid loss through training by matching what the following word would be given the previous 10 tokens.

To run the training, validation and generation, we used a Machine Learning API Keras to simplify the work in describing the model. This allowed us to experiment and test quickly without rewriting gradient descent.

To prevent over fitting we also added a dropout layer with a relatively high weight that provided some protection.

The following Keras code was used to represent the neural network.

```
Sequential([
  LSTM(
    64,
    input_shape=(
        ready_input_data.shape[1],
        ready_input_data.shape[2])),
  Dropout(0.10),
  Dense(
    ready_target_data.shape[1],
    activation='softmax')
]),
```

2.3.1 Word Tokenization

One issue we faced was in generating text was the model only creating stop words, the most common words in natural language. To avoid the model from continuously selecting stop words, we abstracted tokens to not only be words but to be joined with common stop words like "to", "a", ... that would otherwise lead to text with just "to to to to to to".

In doing so we improved the model's performance.

2.3.2 Limited Training Set

Due to time and computing constraint, we only had a relatively small size of training data for the channels of text. This lead to repetitive generated text. To avoid repetitiveness, we opted to select not the argmax of the resulting output vector as the following word, but rather a random choice between the 10/20/30/40/50 highest activated neurons of the output vector. This lead to a more diverse output, but started to degrade the grammatical validity. We settled on 30 as this felt like a decent trade-off between grammatical correctness and randomization.

2.3.3 Text Generation

3 Use of Cognitive Technologies

We used IBM Watson technologeis that made it easy to convert text to speech and vise versa. This allowed us to traing the models easier by importing podcast audio into the model as plain text after watson tranformation process finished.

FILL IN*

4 Application

The application was integral to providinga a usbale interface to the audio.

Something sometihing. maybe write a bit about electron

FILL IN

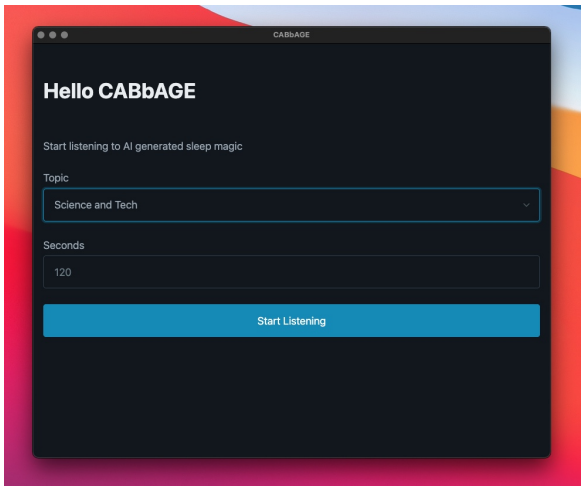

Figure 1: Image of the ElectronJS App

4.1 ElectronJS

To generate the application, we chose to build a native application that would possible to download and run on a variety of machines. This would make it possible for us to not have a constraint on a host machine's operating system due to the way that ElectronJS is essentially an encapsulated web

view. In addition to the interoperability benefit, ElectronJS allowed our team to use a widely used language that allowed us to quickly build an app in the limited time frame.

5 Problems not Solved

Given the limited time scope of the project, 3 weeks was not enough time to collect and train the models on enough data to create perfect text. Through we see he promise, the text often feels like it is missing grammatical marks that could be detracting form the goal of creating fluid sounding speech to go to sleep to.

Additionally, we did not parametrize or customize the type of speech output. This lead to very robotic sounding speech that is more difficult to go to sleep to. Though the speech had human-like vocal inflections, the robotic sound alike to the grammatical issues detracted from the end goal.

These problems could potentially be solved by longer processing time and with more training data. We did not have a strong machine learning fleet of machines to train the models on, so we did the best we could with our personal machines. This problem was further expanded since we were using the CPU version of Tensorflow that does not support massive parallelism with the GPU version that would vastly speed up training and generation times leading to higher quality text, hence speech output.

Lastly, the system architecture used for deployment lacked the Model API that would serve the model output on a separate server. To simplify the workload, we generated the text generation from the three types of output and randomized the start points to provide essentially a stub of the API. If we chose to deploy this application, a more service based deployment approach would be necessary to allow for independence of changes between the Electron application and the two models.

6 Future Work

Essentially, the future work would entail building on the issues of the current implantation through creating a stronger infrastructure, changing Tensorflow from CPU to GPU based along, providing more training data and training the model for more epochs.

The are however a few interesting areas that we did not have a chance to explore, one of which was implementing a custom loss function that would factor in the English grammar. This would entail appending to the loss function grammar issues like a breaking in the common Subject Verb Object form of sentences, run on sentences and misuse of conjunctions. In theory, this would skew the output to be more grammar focused than only trying to match the following word from the previous subsequence of words.

In addition to improving the general correctness of the text generation, we will want to work on experimenting with the types of voices and the different variations of speech inflections or styles. This would fall into the experimental and validation aspects of the project that we did not have a chance to fully go through. This would be composed of running a standardized trial with comparison of REM and NREM sleep between the various possibilities. Since we are using white noise as a baseline, we would use that as the control and compare the performance between standard pink noise generators, and our various channels, lengths and voices of speech output.

This would allow us to show that our approach would create a statistically significant change in sleep that would prompt people to experiment with out technology.

7 References

- https://www.webmd.com/sleep-disorders/pink-noise-sleep
- https://www.sleepfoundation.org/bedroom-environment/white-noise
- http://xpo6.com/list-of-english-stop-words/
- https://ashutoshtripathi.com/2020/04/06/guide-to-tokenization-lemmatization-stop-words-and-phrase-matching-using-spacy/
- https://base64.guru/converter/encode/audio/flac