CS 194 Project Report: TalkToTransformer

Nicholas Lee

1 Overview

For my final project, I created TalkToTransformer, a website that extends the traditional language model demos online by allowing the user to speak into their microphone and send it through a text-to-speech model in order to "speak" to the transformer. The transformer's response is then read back to the user by passing the response through a text-to-speech model. The website and github are hosted here:
`https://talktotransformer.onrender.com/` and `https://github.com/dragon18456/talktotransformer`

2 Technical Aspects of the Project

The following section will detail the more technical aspects of the project.

2.1 Machine Learning Models

A total of 4 machine learning models were used for this project. They are:

- WavToVec2.0: A state-of-the-art self-supervised framework for speech that I used as an Automatic Speech Recognition (ASR) system. This system was provided by the Huggingface repo as was the pretrained weights of the model. On the whole, the model actually does a pretty good job at speech recognition, even understanding punctuation and some capitalization.
- GPT-2: A transformer-based Language Model that was once state of the art (GPT-3 exists, but no mortal can run that on their laptop) that still generated pretty good responses to the input. This model was also provided by Huggingface.
- Tacotron-2: A state of the art text to spectrogram model. This specific version was trained on DDC, or Double Decoder Consistency, which improves the quality of the alignments at test-time. This model was provided by Mozilla TTS.
- MultiBand-Melgan: This is a waveform generative model that takes mel-spectrograms and generates the corresponding waveforms for the audio. This vocoder was also provided by the Mozilla team and I chose it and Tacotron-2 because they seemed to have pretty good audio quality and fairly quick generation (more on this later)

2.2 Website Structure.

All of the most relevant code for this project is on my github. My project sported a Flask backend coupled with a React.js frontend. I chose the Flask backend because it was in python and that made it really easy to create a REST-style api for my static frontend website to call. I chose React.js for my frontend mainly because most of the frameworks for frontend didn't have everything that I wanted to have, mainly microphone support for a variety of tasks. Here, React.js is an industry standard, so I took this opportunity to learn React and build my website using only React on the frontend. Basically, I have a static frontend that calls a Flask Rest-API on the backend. The backend was containerized in Docker to make moving and sharing the backend environment easier.

2.3 Deployment

The website is deployed on an ec2 t2-large instance as a REST api. The frontend is hosted on render.com, a website that gives free hosting for static websites. I simply gave them my github repo to my frontend and they gave me a way to route all of the api calls from the frontend to my ec2 backend. What is great about this is that I get a free SSL certificate from the render people, so that he microphone works on normal browsers. Chrome and Firefox disable the microphone on unsecure websites, so this was pretty important. On the backend, I used a Nginx server configures to reroute HTTP calls from port 80 to port 5000 where the backend is running on gunicorn.

3 Most Difficult Parts of the Website

Definitely web development is a "you know it or you don't" kind of area. The entire field is extremely rich in minute details and technical details that made progressing through this project more difficult than it seemed at first, especially since I had never done web development work in my life. However, because I had never done this kind of work before, I wanted to take on a challenge and make the best website that I could.

One point of contention was getting the Flask and React.js parts of the website to play well. Different formatting issues came up when trying to transfer the data from the frontend to the backend and vice versa.

In this case, the audio encoding of the blobs on the frontend played a big role in the preprocessing in the backend. Getting the audio to play on the frontend was a huge pain.

Deploying the website was a whole other world to me. Once I had gotten the website to run perfectly adequately on my computer and a small ec2 instance, I thought that deployment would not be as difficult.

3.1 Socket.io

I had spent about 40+ hours getting the website up and running in its current form. However, there was a whole other 20-30 hours that I spend trying to get sessions and sockets implemented on the website. In it's current form, the backend is really just a stateless API, it takes inputs and returns outputs without remembering anything. However, this meant that generating longer and longer responses would prove to be a challange. In its current form, the website can take significant time (¿1min) to generate and play 300-500 words in response. My plan was to create a websocket between the backend and the frontend so that it could iteratively generate words and audio so that the user wouldn't have to sit for so long. However, this ultimately didn't work as well as I thought. One, I had tried to use greedy search on GPT-2, but I found that the quality of the responses if you didn't use beam search fell off pretty hard and made less intersting responses in general, so I didn't want to sacrifice that. However, that meant that generation was fairly difficult. Add onto that the fact that the audio generation was significantly slower than the generation itself.

The other part that I didn't realize was that in order to accomplish this, I would need to create session for different users on different browsers. We would need to keep track of what generation is happening for what users all at once so that the responses wouldn't get messed up. However, this meant implementing sessions, and after spending 40+ hours on the project, I couldn't spend another 40+ hours getting socket.io and session authentication from users up and running.

3.2 Deployment

While that solution to deployment seems easy, I spent a lot of time struggling to deploy the website. At first, I tried a more standard and practical approach which was to host the frontend on a static website on S3, and host the containerized backend on EBS and Elastic Beanstalk and link them together to get elastic compute. However, try as I did, I just couldn't get my docker image to run on EBS. Even though it ran perfectly fine on my normal EC2 instance, it just wouldn't run at all. In addition to that, I tried adding SSL to the nginx server itself for a while, but I couldn't get a certificate for an Amazon EC2 url, and domain names cost money, so I decided that having a free domain and SSL certificate from render was worth the hassle. While this solution definitely seems more hacky than others, it was the one that worked after hours of struggling through tutorials.

4 Future Directions

On the Machine Learning side, I could stand to perhaps train a language model to generate responses more organically like a chat bot. Going along with the sessions and sockets work that I partially completed, I could then try to get my project to have long standing session and then I would be able to have users actually have a conversation with a transformer.

On the web dev/ deployment side, I could stand to make the frontend look better. Because I didn't use any frameworks, it does mean that the frontend looks a bit bare. In addition, I am not sure if my website works on all browsers and devices, I probably will need to come up with some way to make the website useable on all platforms in the future.

5 Closing Thoughts

I definitely took on a lot by choosing to do a web development project when I had not done web development ever in my life. Indeed, I did learn a lot about Javascript and Deployment and Latency while trying to make the website the way I wanted it to be, but it does feel like people who are actually good at web development would have spent significantly less time on this project than I alone. However, I learned a lot about deployment for machine learning models, especially that when latency gets in the way, you can really feel it drag the whole performance of the website down. I can take a lot of experience away from this project.