CS4120 Final Project Report

Due: Monday, April 19, 2021 @ 11:59pm
Group Members: Nathaniel Gordon
Professor: Felix Muzny

1 Artist's Statement

Media that we look back on as defining experience often embodies a specific tone or feeling that ushers in feelings of comfort and familiarity. For me, the episodes of the popular television program How It's Made evoke strong memories of my childhood, and how the series helped inspire me to pursue a career in technology.

Through the use of language modeling techniques, my work seeks to capture this nostalgia and replicate it with completely novel generated content. Specifically, the script for this video was produced with the transformer language model GPT-2, and was trained on nearly 500 episodes of How It's Made. The accompanying footage was sourced from real episodes of the program, with the clips cut in such a way to mimic the meaning of the script.

The experience of nostalgia is highly individual. Without the context of past experiences, many viewers may feel nothing but mild confusion when watching the video. Others may associate aspects of the media with any number of past experiences, both positive and negative. Regardless of what you feel, challenge yourself to consider what sorts of experiences would evoke positive nostalgic feelings in yourself, and how the ability to replicate the stimuli of those memories could be cathartic.

2 Project Dataset

The dataset used for this project was created by hand specifically for this project. I found the text for about 450 *How It's Made* topics on a subtitle-providing website and copied them into a file [1]. I stripped out action directions (like '[upbeat music plays]') and the speaker annotations (of which there is only 'narrator:').

Once the episode scripts were each sourced into individual files, I wrote a short program to unpack the zipped dataset. This consisted of copying the text from each episode in a random order and compiling them into a single document. Special tokens (`<BOS>` and `<EOS>`) were inserted and the the beginning and end of each script.

3 Character-Level Text Generation With a RNN

The first approach utilized for generating a script was a character-level generation technique in combination with the recurrent neural network (RNN) architecture that was covered in class [2]. For the most part, character-level generation is almost identical to word-level generation: the model simply learns to predict the next character in a string, as opposed to the next word in a sentence. The goal for this approach, considering that subsequent experiments with GPT-2 were likely to produce a higher-quality result, was simply to try and train a model that would produce somewhat convincing text in the style of *How It's Made*.

Before compiling the model, an encoding first has to be produced to convert strings into numeric vectors for the model to process. In the case of character-level generation, this encoding is quite small, as it encodes the text's charset as opposed to vocabulary. This means that the encoding lookup contains only a few dozen entries, as opposed to potentially tens or hundreds of thousands of words in the vocabulary. After the encoding is generated, the input corpus must be split up to be fed into the model as inputs. A common technique for character-level generation is to split the text into relatively small chunks, with each input being offset from the last by only a few characters. This gives the model a chance to process each piece of text multiple times at the risk of exacerbating overfitting [3].

Once the input vectors have been encoded, the model can be trained. The model architecture utilizes 3 LSTM layers, each separated by a dropout layer to combat overfitting. Much of this architecture was inspired by my past experiences on some of the in-class assignments as well as the fifth homework assignment. The model was trained over several sessions for 30 total epochs, with each epoch running on local hardware in about 15 minutes.

A key difference between character- and word-level generation with a RNN is that the former is not guaranteed to produce tokens in the input corpus. To investigate this further, the text-generation code was modified to record any generated words not present in the input text. These 'novel tokens' ranged from actual English words to utter nonsense. However, the majority of the novel tokens were extremely close to the English lexicon: they utilized English prefixes and suffixes, diphthongs, and other common features. Moreover, many of these words were similar in composition to the technical jargon present in the *How It's Made* corpus, indicating that the model was successfully mimicking the style.

The character-level generation model was not without its shortcomings: namely, text cohesion. While many generated sentences would be fairly coherent in isolation, at a larger scale the text simply rambled without any central focus. While there are ways to combat this with better pre- and post-processing, this was not the focus of my project.

4 Text Generation with gpt-2-simple

The second text-generation approach for this project revolves around the use of the Generative Pre-trained Transformer 2 (GPT-2) algorithm. GPT-2 was created by OpenAI in 2019, and its extensive pre-trained models are suitable for several critical language-modeling tasks, text generation included [4].

The Python library `gpt-2-simple` was utilized to harness the power of GPT-2. This library, written by Max Woolf, builds on fine-tuning code written by Neil Shepperd by adding an enhanced functional interface to make the fine-tuning process streamlined and simple [5][6].

While the eponymous simplicity of `gpt-2-simple` meant the immense process of implementing the advanced transformer could be skipped, the parameter choices for this project will be briefly discussed. The training was run on a Google Colaboratory instance running a Tesla T4 GPU. This allowed for 1000 epochs to be run in approximately 50 minutes. The model was trained in several sessions for a total of 5000 epochs. GPT-2 comes pre-trained with models of varying sizes: this project utilized the 355M-parameter model, which was the largest that could be easily stored in the Colaboratory instance's memory.

When generating text with GPT-2, several unique features of the library were used. First, top-k sampling was used to enable the model to run at a fairly high temperature without risking an overly chaotic output. Nucleus sampling was also used to make sure that sub-optimal choices were balanced with more expected ones. Additionally, GPT-2 is able to truncate its output to correspond to keyword tokens. Accordingly, the `<BOS>` and `<EOS>` inserted into the dataset are used to seed and truncate the sample respectively. One downside of this (particularly at high temperatures) is that the model may generate an `<EOS>` token either too early or too late, causing the outputs to either end prematurely or overrun the generation word limit [7].

The text-generation capabilities of GPT-2 appear more compelling than that of the RNN. Providing the model with a small piece of seed text not only results in a fully-formed, cohesive episode, but also one that somewhat accurately reflects the manufacturing properties of the seed's subject. For instance, seed text about a vehicle may produce results that focus on metals and machining, while a seed that mentions a food product may discuss adding various ingredients to a mixture. That being said, there is a high degree of variation in the outputs: generating 10 samples from the same seed text may produce wildly differing results.

5 Producing the *How It's Made* Episode

With cohesive episode scripts being generated by the GPT-2 model, a fake *How It's Made* episode could be created. To begin, a script was selected and fed through a text-to-speech program [8]. This would serve as the voiceover track for the episode. Next, relevant clips from existing *How It's Made* episodes were cut together to form the video. A royalty-free synth track was selected to be the background music. Finally, subtitles were added to showcase the generated script. The video was edited in Lightworks, a free editing system [9].

6 Future Work

While the scripts generated for this project possess accurate quality and tone, their greatest shortcoming is that they fail to describe how any real products are made. In order for a model to generate accurate non-fiction content like *How It's Made*, it would need to possess a knowledge of how manufacturing processes work, rather than simply an understanding of language. However, projects like AI Dungeon show that language models can produce convincing fictional scenarios [10]. One potential way to increase the impact of an application like AI Dungeon would be to include visual stimuli, similar to the episode produced for this project. This could be achieved through a game engine that is able to interpret the text generator's output into characters, settings, and interactions. Creating a fully automated multimedia system like this could be the subject of future work.

7 Data Availability

The project dataset and code can be found at this GitHub repository:
https://github.com/gordonng123/HowItsGenerated

The completed video can be found on YouTube:
How It's Made: Synthetic Cereals (CS4120 Final Project)

8 Works Cited

1. How It's Made (2001–. . .) - Episodes With Scripts
2. Working with RNNs
3. Character-level text generation with LSTM
4. Better Language Models and Their Implications
5. How To Make Custom AI-Generated Text With GPT-2
6. GitHub: GPT-2
7. GitHub: gpt-2-simple
8. Text-To-Speech
9. Lightworks: The Professional Editor for Everyone
10. AI Dungeon