CS 6120: TV Script Generation using Deep Learning

Rohit Awate, Shubham Bhagwat, Rohan Devasthale

Code: https://github.com/RohitAwate/TheOfficeScriptGenerator

Abstract

This project aims to develop a natural language processing (NLP) model capable of generating a script for the popular TV series "The Office (US)". Utilizing dialogues from all nine seasons that aired on the show as a dataset, we experiment with various deep learning architectures, such as Bidirectional Long Short-Term Memory (LSTM) networks, deep neural networks. The resulting model will produce a script containing character names followed by their respective lines of dialogue, capturing the essence of the show's unique humor and style.

1 Introduction

The goal of this project is to build a language model that can generate a script for the popular TV show "The Office" based on a given prompt. We employ Bidirectional Long Short-Term Memory (LSTM) models on a large corpus of "The Office" scripts.

The motivation for this project is to explore the capabilities of language models in generating coherent and engaging text that resembles written dialogue. It also demonstrates the potential of these models in generating content for entertainment. Our goal is to develop a model that can generate coherent lines for a scene that sound similar to what the characters would say on the show.

2 Background/Related Work

Recent advances in NLP have resulted in the development of language models that can generate human-like text. Examples include OpenAI's GPT series and Google's BERT. These models have been successfully applied to tasks such as text summarization, translation, and content generation. Our project aims to build upon these advancements and tailor a model specifically for script generation. Previous studies have demonstrated the capabilities of these models in tasks like poetry generation, storytelling, and dialogue generation. We will investigate how these techniques can be applied to generate engaging and contextually relevant scripts for a popular TV series.

Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) models are two commonly used NLP methods that can be used for script generation. RNNs are neural networks that are designed to handle sequential data, such as text. They are particularly useful for tasks that require the model to remember previous inputs and use them to generate the next output. LSTMs are a type of RNN that can remember longer sequences of data and are better suited for more complex tasks.

Previous work has shown that RNNs and LSTMs can be used for script generation in various contexts. For example, such models have been used to generate dialogue for TV shows such as "The Simpsons" and "Friends". They have also been used to create movie scripts and short stories.

3 Data

The dataset [2] used in this project consists of dialogues from all nine seasons of "The Office (US)." Each sample in the CSV is a line of dialogue from the show along with the season,

episode, and scene number along with the speaker of the dialogue. Following is an example from the dataset:

```
{'season': 2, 'episode': 5, 'title': 'Halloween' 'scene': 32, 'speaker': 'Creed'', line': 'No, you have the power to undo it.'}
```

We preprocess the data by tokenizing the text, converting the tokens into integer sequences, and generating training samples of length=100.

First, we performed some simple exploratory data analysis on the dataset to gather statistics about characters and lines. One thing we noticed was that the dataset was seemingly compiled from a variety of online sources and thus, parts of the data weren't as good as others. Specifically, there were some entries where the speaker's name was followed by a colon and space, for example "Michael: ". The lines of dialogue associated with these samples were poorly formatted. Thus, we removed these.

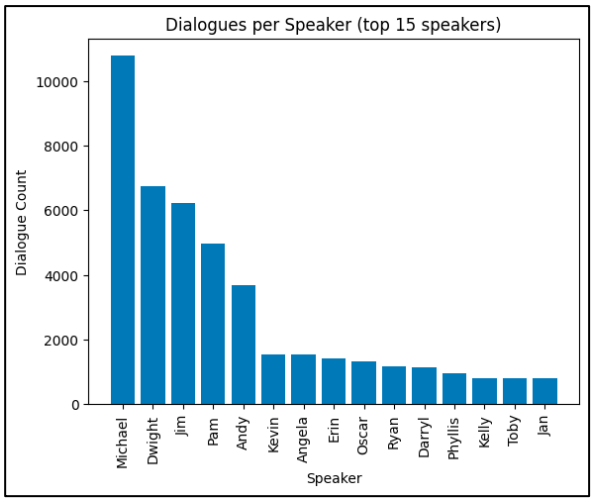

fig 1. Dialogues count Vs Speaker

Next, we found that there are 750+ distinct speakers in the dataset. However, in line with Zipf's law, only a few major characters accounted for most of the lines. So, we decided to prune the data to contain lines only from the top 15 characters that have the most lines. We arrived at this number based on the distribution that can be seen in *(fig 1)*. The top 15 characters have close to a thousand lines each, and the number steeply declines past that. We also corrected some misspellings. For example, a character named DeAngelo was misspelled "Deangelo".

Using this dataset, we generated two different textual datasets which we use further to tokenize and generate input sequences to the model. The first one contained additional meta tokens, such as <SCENE_START>, <SPEAKER_START>, <LINE_START> and their corresponding end tokens such as <SCENE_END>. These were inserted in the text to provide more context to the model. The *fig 2* shows the extract from this dataset.

```
<scene_start>
    <speaker_start> Michael <speaker_end>
        <line_start>
            <sent_start> All right Jim <sent_end>
            <sent_start> Your quarterlies look very good <sent_end>
            <sent_start> How are things at the library <sent_end>
        <line_end>
    <speaker_start>
```

fig 2. Extract of the dataset

The other textual dataset we generated contained the same data, but without any of the meta tokens. Each line in this file is simply in the format "speaker: line … ". Following is a small extract from this dataset:

```
Michael: All right Jim. Your quarterlies look very good. How are things at the library?

Jim: Oh, I told you. I couldn't close it. So…

Michael: So you've come to the master for guidance? Is this what you're saying, grasshopper?
```

We used both of these datasets with our models, however, there wasn't a significant difference in the quality of dialogue produced. The one with meta tokens required an additional post processing step to remove the meta tokens from the final output. For our final model, we ended up using the latter.

We created a heatmap (*fig 3*) to visualize the dialogue between the top 15 characters based on the next character they interacted with. This allowed us to identify which characters tended to speak with each other more frequently and which ones had less interaction. The resulting visualization can be used to gain insights into the

dynamics and relationships between characters in the show.

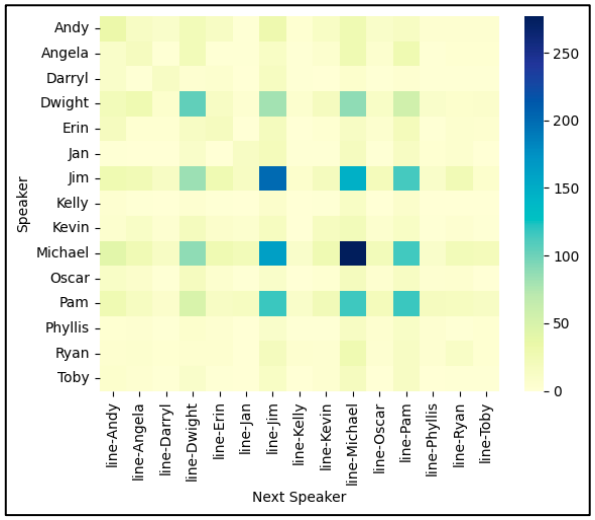

fig 3. Heatmap of Speaker and Next Speaker

4 Methods

4.1 Baseline with n-grams + LSTM

Our baseline implementation used an LSTM network with a dropout layer which was trained on n-gram sequences (with n ranging from 4 to 10) similar to HW4. We used simple sampling strategies to generate text viz. picking the token with maximum probability as well as random sampling based on the softmax distribution. The generated dialogue was extremely incoherent and generally poor.

4.2 Word Embeddings (Word2Vec)

The next thing we tried was training word embeddings on the input data and feeding those to the model using a Keras Embedding layer before the LSTM one. This moderately improved the coherency of the output.

5 Experiments

5.1 Architecture Tweaks

At almost every step of our experimentation, we tried tweaking model parameters such as number of layers, number of units per RNN layer, adding or removing dropout layers, etc. However, this didn't make a significant difference in the ultimate output.

5.2 Using longer sequences

We took inspiration from [1] and decided to use longer sequences of tokens to feed to the model. We experimented with input sequences of size 50 with a step size of 5. This resulted in the most dramatic improvement in the quality of output text. The text sounded almost English and there was coherency at least at the sentence-level if not the entire set of lines.

5.3 One-hot encoded input sequences

We noticed that [1] used one-hot encoded vectors as input to the model instead of word embeddings. While we were skeptical of this approach over the more superior word embeddings that we were already using, we decided to give this a shot as well. Surprisingly, this also produced excellent results. Overall, however, the improvement in quality was marginal compared to the previous approach and came at the cost of extremely long training times thanks to the massive one-hot encoded vectors used as inputs to the model.

5.4 GloVe

At this point, our outputs, while decent, were still far from our expectations of coherent dialogue that would capture the essence of the characters and the show. We felt that the word embeddings trained by us were perhaps something we could improve upon. Thus, we decided to try pre-trained embeddings from GloVe.

5.5 GRU + Bidirectional LSTM

To further explore RNNs, we then decided to try a Bidirectional LSTM architecture, which allows the model to learn from both past and future contexts. This was implemented using the Bidirectional wrapper available in Keras. The bidirectional LSTM model outperformed the baseline model and improved the model's ability to generate coherent and contextually relevant dialogue. The bidirectional LSTM achieved this by processing the input sequence in both forward and backward directions, allowing the model to capture both the preceding and succeeding context for each word.

5.6 Transformers

This was a stretch goal of our project. We were able to train a transformer-based network using the official tutorials for KerasNLP. Unfortunately, our code crashed on Google Colab after predicting about 4-5 tokens. Due to time constraints, we were unable to pursue this approach further.

5.7 Top-p sampling & temperature

Further, we defined a function for generating text using top-p sampling and temperature techniques. The function takes in a seed speaker and generates a specified number of lines, or a maximum number of words based on the provided parameters. It uses a pre-trained LSTM model and a tokenizer to encode the seed text, and then generates the predicted word using top-p sampling and temperature. The function filters out the least likely words and selects one at random based on the probabilities. The generated word is appended to the generated text and the seed text is updated for the next iteration. The output generated is evaluated based on the coherence and creativity of the generated script. This technique allows for more control over the diversity of the generated text and can produce more creative and coherent results.

5.8 LR and scheduler

In our bidirectional LSTM model, we used a learning rate (LR) scheduler to adjust the learning rate during training. Specifically, we used the ReduceLROnPlateau callback function from Keras, which reduces the learning rate by a factor of 0.5 if the validation loss does not improve for a certain number of epochs. We set the initial LR to a default value and then allowed the scheduler to adjust it as needed.

The purpose of using an LR scheduler was to allow the model to adjust the learning rate based on the performance of the validation set, which can prevent the model from getting stuck in a local minimum or overfitting the training data. By reducing the learning rate when the validation loss does not improve, the model can potentially escape a local minimum and find a better solution. Overall, using an LR scheduler improved the performance and stability of the model during training.

6 Evaluation

We also aimed to evaluate the quality of the generated script using a neural network classifier. To accomplish this, we fed the dialogue generated by our bidirectional-LSTM model as input to a neural network, which aimed to predict the speaker. Our initial attempt involved a simple Feed Forward Neural Network, which provided a baseline for comparison. However, to improve the accuracy, we incorporated LSTM in this neural network. The accuracy of this model was marginally better than the Feed Forward Neural Network, with a training accuracy of 56% and a test accuracy of 15%. This is because the characters of the show have dialogues related to day-to-day life and those are difficult to pinpoint to a particular character. Also, we tested the evaluation metric on one of the speaker's most frequent/famous dialogue and the model predicted it accurately.

It is important to note that we are not relying solely on the accuracy of the neural network classifier to evaluate the quality of our generated script. We are also assessing the coherence, grammar, and overall appearance of the generated dialogue, as well as how well it captures the essence of the original show. Ultimately, the success of our project will be judged by the subjective evaluation of the quality of the generated script.

7 Results

Our language model produces moderately coherent results. We observed that while there isn't a lot of coherence across lines, the model is able to generate fairly coherent lines as can be seen in the output example (*fig 4*). Having watched the show, we also noticed that the lines do capture some essence of the respective characters. They often contain words that are closely associated with a certain character. Bombastic characters such as Michael have

longer lines while quieter characters such as Angela have shorter lines. Overall, while the results are decent, we believe that a transformer-based network can outperform our model.

```
dwight:  thanks. made.
erin: this is great, right? you want to get that worse
with your girlfriend?
michael: well, uh, come on. it's not the best idea.
dwight: yeah.
jim: nice.
jim: um, that's highway! it's stairmageddon.
erin: i miss cece.
andy: yeah.
donna: yeah, okay, thank you.
pam: yeah.
```

fig 4. Output Example

8 Conclusion

We conclude that using various advanced models of modern NLP, we can build highly sophisticated language models for a particular use case. In this case, we built a language model using bidirectional LSTM to generate the script for "The Office". Further, models like transformers can help to generate coherent scripts that capture the essence of the actual show. That being said, the efficient methods to evaluate these models are ambiguous and these models can be more properly evaluated by judging the overall content, coherency, humor and style instead of statistical metrics like accuracy or perplexity.

9 References

- [1]https://gilberttanner.com/blog/generating-text-using-a-recurrent-neuralnetwork/
- [2] Dataset (Link)
- [3] TV script generation using deep learning (Link). By Pradhyo Bijja
- [4] Visualizing Word2Vec Word Embeddings using t-SNE. By Sergey Smetatin
- [5] How to generate your own "The Simpsons" TV script using Deep Learning.
- [6]https://shiva-erma.medium.com/generating-a-tv-script-using-recurrent-neural-networks-dd0a645e97e7

Artist's Statement

TV Script Generation using Deep Learning

Our NLP project is a unique exploration of the intersection between art and technology. By leveraging the power of natural language processing and machine learning algorithms, we sought to capture the essence of a beloved TV show and use it as a basis for generating new, original content.

As fans of "The Office", we were drawn to the show's distinctive humor and writing style. Our goal was to create a tool that could help other writers and creators generate content that feels authentic to the show's voice and tone. To achieve this, we processed the show's scripts, so that they are suitable for training a model. We trained on this data to build a 15M parameter language model using advanced deep learning techniques such as bidirectional LSTMs as well as smart sampling strategies that are able to leverage the model to generate interesting results. Our model is able to recognize the nuances of dialogue and character interactions, and generate new content that feels like it could have come straight from the show's writers' room.

Ultimately, our project is a testament to the incredible potential of NLP and machine learning in creative endeavors. By using these cutting-edge technologies to analyze and replicate the artistic sensibilities of a beloved TV show, we hope to inspire other artists and creators to explore the many ways in which technology can enhance and enrich their work.