Text Generation for The Big Bang Theory

Course project for CSE 6240: Web Search and Text Mining, Spring 2020

Raghav Raj Mittal*
Georgia Institute of Technology
Atlanta, Georgia, USA
raj.raghav@gmail.com

Austin Jiang*
Georgia Institute of Technology
Atlanta, Georgia, USA
jiang2023@gmail.com

Satwik Mekala*
Georgia Institute of Technology
Atlanta, Georgia, USA
spmekala@gatech.edu

ABSTRACT

Over the past few years, recurrent neural networks have enjoyed considerable success in generating realistic text given a related corpus to train on. In this project, we aim to crate a model that will be able to generate scripts for television sitcoms. Specifically, we will be looking to generate scripts for the Emmy Award winning sitcom The Big Bang Theory (TBBT). We collect and process the scripts spanning all 10 seasons of the show, which results in a dataset comprising of 2847 scenes, 53997 lines, 22709 unique words. We find the most popular uni-grams, bi-grams, and tri-grams, as well as the average sentiment scores for the main characters. In addition, we try to determine which character is most important in the series by computing the centrality scores of the characters from a graph representing their interactions. We also obtain Word2Vec embeddings for each word in the vocabulary and find the words closest to few chosen words. We trained 3 different RNN-based models - Vanilla RNNs, LSTMs, and GRUs - and evaluated the generated text using the BLEU and the Self-BLEU metrics. We also investigated the performance and text generation capabilities of the different models when the sequence length is varied and the word-to-embedding layer is initialized with pre-trained Word2Vec word-embeddings. Pre-initialization with word2Vec embeddings does not result in an improvement over the baseline models. We see that Vanilla RNN has the highest BLEU score signifying the greatest n-gram overlap with the training dataset, whereas the LSTM model has the lowest Self-BLEU score (the lower, the better), which signifies that the output generated is the most diverse.

1 INTRODUCTION

Our objective is to create a model that will be able to generate scripts for television sitcoms. Specifically, we will be looking to generate scripts (full-fledged dialogue exchanges between characters) for the Emmy Award winning sitcom *The Big Bang Theory* (TBBT).

To use a new dataset, obtained by web-scraping scripts for 10 season of the show from a public website [4]. We process the dataset to extract scene descriptions, speaker names, dialogues, and punctuation tokens. Once we parse the entire text, we assign a unique integer ID to each unique word in the vocabulary. This way, the input text is represented as a sequence of integers, which can then be fed into the learning models.

For our experiments, we use 3 different RNN-based architectures. Each contains an Embedding layer that transforms the input word into a 256-dimension embedding vector. The embedding is passed to the one of the 3 RNN-based cells - Vanilla RNN, GRU, or LSTM - whose output is then fed into a fully-connected layer that gives us an integer corresponding to the generated word. We vary the length of the sequence of tokens as a hyper-parameter, and also propose to initialize the word-to-embedding layer with weights from a pre-trained Word2Vec model in order to ease the learning burden on the model. We evaluate the models using the BLEU and Self-BLEU metrics that measure the quality and diversity of the generated sentences respectively.

We see that such pre-initialization with word2Vec embeddings does not yield improvements over the the baseline models. The Vanilla RNN with the sequence length of 7 has the highest BLEU score signifying the greatest n-gram overlap with the training dataset, whereas the LSTM model with a sequence length of 6 has the lowest Self-BLEU score which signifies that the output generated is the most diverse.

Script generation technology is very important to writers and directors in entertainment industry. With the rise of the internet, there has been an acceleration of streaming media being generated and consumed all around the world. This has led consumers to demand more and more original content. This means that companies and writers are being pushed to the brink to bring new content and ideas. This is problematic since the most time consuming portion of a production is the writing process. In order to remedy this problem writers could use script generators to help to write new scenarios and help writers better understand their writing . One of the things that we can see is that it will help writers increase their efficiency in the writing process. It will allow them to have a starting point that they can work on. Another thing that it can help writers find what are overused tropes, scenes, and gags in their writing, since the model will be biased to generate those situations more often. This analysis will help writers be more mindful of their own pitfalls and allow for more interesting and original scripts which will make viewers more engaged. We are hoping that this model will be helpful for writers of long running series.

2 LITERATURE REVIEWS

There has been a lot of recent work in text generation, particularly because of the large number of potential applications, including but not limited to text translation, visual question answering, chatbots etc. Earlier work in this domain used language models such as n-grams and feed-forward neural networks [7] [14]. Then, the focus shifted to methods based on artificial recurrent neural networks such as LSTMs and GRUs that retain memory about the previously seen tokens, which work reasonably well in practice [12]. In training, the predicted word is compared to the ground truth words of the sentence, and the loss is back-propagated to output sentences similar to the training dataset. However, at test (generation) time,

*All authors contributed equally to this research.

the next input word is sampled from the output distribution corresponding to the previously predicted word, thereby causing the model to suffer from exposure bias. The inputs at test time are sampled from a different distribution (from that used in training), and therefore an early error made during generation will propagate and errors accumulate as more words are predicted. However, LSTMs and GRUs are shown to work reasonably well in practice when trained using maximum likelihood estimation.

There has also been some work using Generative Adversarial Networks (GANs) to generate text [9] [5] [6]. GANs generally work well with real-valued data such as images that are represented as a matrix of numbers that can be incrementally altered, as opposed to generating discrete words or tokens. Yu et al. (2016) first proposed SeqGAN, a sequence generation framework that modeled the task as an RL problem for gradient update [17]. However, GANs often experience mode collapse where the generator produces only a limited set of samples, which adds a layer of difficulty to evaluating the quality of the generated sentences [11].

More recently, there has been a lot of focus on transfer learning, which proposes pre-training on a more general-purpose architecture that can later be fine-tuned for more specific downstream tasks. The pre-training therefore, is mostly independent of how the model will be used later. However, creating these general purpose models is often expensive and time consuming. Google's *Attention is all You Need* [16] paper introduced the 'transformer', an architecture created to better handle long-range dependencies in sequences, without using RNNs or convolution.

While the above conversation was for text generation as a subject, it would also be good to note other attempts in our particular application - to generate TV show scripts. There has been some prior analysis in this domain. Namely, Mangal et al., perform a comparison between LSTM, GRUs, and Bidirectional RNNs for TV show script generation, and reach the conclusion that LSTM generates text in the quickest and most efficient way out of the three different deep learning models[10]. However, using time and training loss as metrics for evaluation are not comprehensive. The BLEU metric is a modified form of precision that uses n-gram similarity between the input and generated texts of the model [15]. The value ranges between 0 and 1 - the higher it is, the more similar it is to the input text. While BLEU is a measure for the quality of the output sentence, the Self-BLEU metric evaluates the diversity of the generated sentences. Self-BLEU measures an averaged BLEU score between each generated sentences and the rest of the generated sentences. The lower the value, the more diverse the sentences are. These manually designed metrics serve as approximations for the true objective of generating sentences similar to the training data, yet diverse in output.

3 DATASET DESCRIPTION

3.1 Data Preparation

The scripts for The Big Bang Theory episodes are available on a public website [4]. There are 10 seasons, and every season has around 23 episodes. The scripts have stage directions which divides the script into scenes, and each scene contains a specific dialogue exchange between the characters. We used BeautifulSoup[1], a python text-scraping library to download these scripts.

In order to pre-process the dataset, we extracted the scene description and corresponding dialogues storing them in a pickle (pkl) format. The size of pickle file is 4MB. The dataset is stored as a dictionary, with the keys as the season episode ids, and the content - scene description and dialogues - as its values. We iterate through each individual scene-based script, and isolating the lines from each scene. Each line is labeled as either a scene description or a character's dialogue. In each character's dialogue, we find the name of the character by parsing the text present before the first colon. Therefore, each dialogue comprises of a character name as well as the spoken text. Each line is first lower-cased, and then each punctuation is then replaced with a special token that is then part of the new vocabulary. Once we parse the entire text, we assign a unique integer ID to each unique word in the vocabulary. The input text can now be represented as a sequence of integers, with each integer corresponding to a specific word in the vocabulary.

This dataset is necessary for the success of this project as to generate novel yet coherent scripts for The Big Bang Theory, it needs access to the previous scripts from the show. We believe that this dataset will be sufficient as it contains scripts for all 229 episodes, with 53997 lines in total. Having less data could lead to memorizing the sequence of tokens in the input sentences, but with a sufficiently large dataset like this, the model can pick up on grammar and sentence structure, and generate coherent yet diverse sentences.

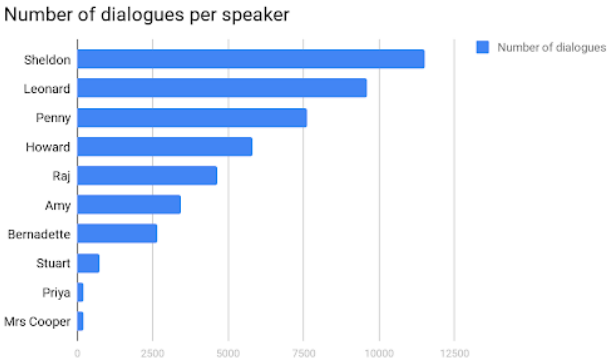
3.2 Raw Data Statistics

The dataset comprises the scripts of all the 10 seasons that the TV series aired, with a cumulative of 229 episodes that cover 2847 different scenes.

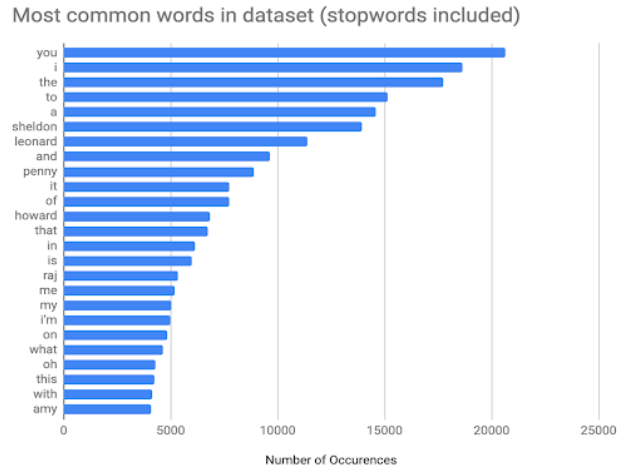
The total vocab size (number of unique words) in the entire dataset is 22709. This includes the word 'Scene' which appears when there is a scene change, the character names, and all dialogues. In our dataset, we process 10 different punctuation tokens including commas, colons, dashes, exclamation marks, question marks, quotation marks, periods, semicolons, and the left and right parentheses.

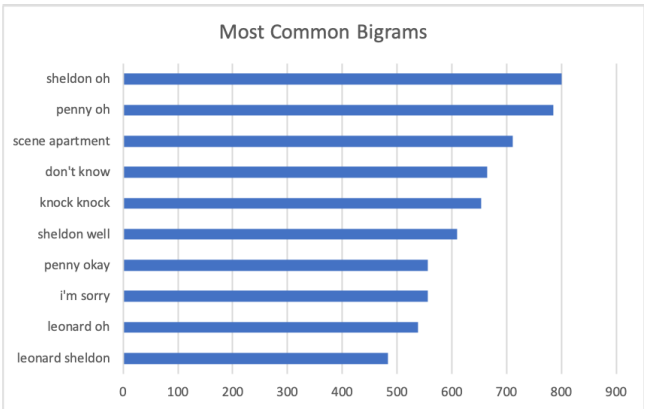
The total number of lines in the dataset is 53997. The average number of lines per scene is 18.65 and the average number of words per line in the dataset is 15.82, thereby making each scene approximately 295 words long. In addition, the longest line has 46 words, and shortest line has only 1 word.

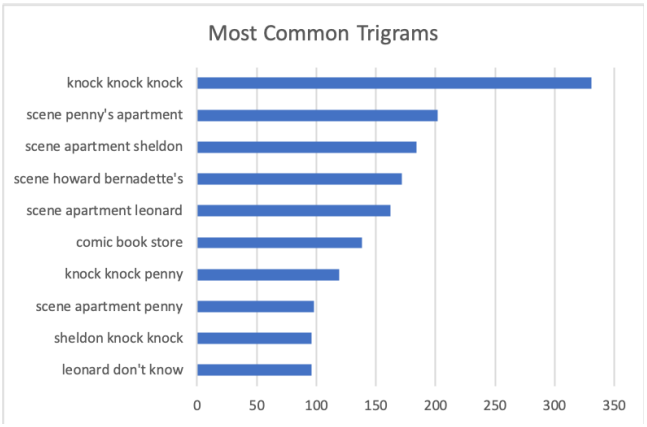
There are a total of 421 different speakers over the course of all 10 seasons. The number of dialogues for the most popular speakers are shown below:



The most common words in the dataset are show below:







3.3 Data Analysis

The most common bigrams and trigrams are shown in the figures below. Sheldon's iconic phrase 'Knock Knock Knock', along with the phrases that contain the word 'Scene' and 'Apartment' show up the most amongst the trigrams given by the protagonist's inclination to knock three times, as well as the fact that a significant part of the series is spent in the main character's apartments. The most common bigrams are mostly parts of these common trigrams itself.

Given the scripts of all the 10 seasons, we also compute the number of times each of the seven main characters mention each other, and obtain centrality scores for each of the characters from the resulting directed graph. This gives us a measure of how 'important' or 'central' the person is in the network. In this graph, the nodes are the characters and the edge weights represent the number of times a person is mentioned. We compute the Eigenvector centrality of each of the nodes, which is a measure of the influence a node has on the network. As expected, we see that Sheldon has the highest centrality score, which means that he is the most 'connected' in the group or the one with more important 'interactions'. The centrality scores are shown in the table below:

Character Name	Centrality Score
Sheldon	0.608
Leonard	0.551
Penny	0.441
Howard	0.199
Raj	0.162
Amy	0.242
Bernadette	0.082

Taking sheldon as the most centered person in the main characters, we also trained a word2vector model, and find the similarity words of sheldon in the 10 seasons, As we expected, see that Sheldon has the highest similarity score with Leonard, which is his colleague

and also best friend, which also means that he has the most conversation with Leonard. The similarity scores are shown in the table below:

Character Name	Similarity Score
Sheldon	1
Leonard	0.837
Penny	0.807
Howard	0.742
Raj	0.772
Amy	0.829
Bernadette	0.666

Similarly as above, Leonard and Sheldon both are physicist at Caltech. we are taking physicist as the sample word to fed into the word2vector model, and find the similarity words of physicist in the 10 seasons. Since now, the result start to get interesting. Some feedback words including theoretical, scientist mediocre makes sense. However, the result also shows some unexpected words including loop. The similarity scores are shown in the table below:

word	Similarity Score
physicist	1
theoretical	0.623
hero	0.544
scientist	0.541
trained	0.519
cosmology	0.512
loop	0.506
mediocre	0.498
arrogant	0.496

We decided to do a sentiment analysis of character lines in The Big Bang Theory. We passed in different line from characters to Monkey Learn [2], a machine learning API for classifying and extracting custom models. We then averaged using the generated confidences to average together a Sentiment Score. This helps with identifying the tone of the generated text, and quirks in tone that we will see in the generated text.

Character Name	Sentiment Score
Sheldon	-0.68
Leonard	-0.39
Penny	0.441
Howard	0.243
Raj	0.5624
Amy	0.242
Bernadette	-0.15

4 EXPERIMENTAL SETTINGS

The whole model training and text generating part is done in Google's Colab. We used both the NVIDIA Tesla K80 GPU and CPU. Our models were trained in the GPU setting, and it took approximately 20 minutes to train each model. The training's average GPU usage was 1GB and RAM usage was 2.46GB. We used 100% of the data for training. Once a model was trained, that trained model is then used to generate new text within the CPU setting for a few seconds. We used BLEU and Self-BLEU as the evaluation metrics. Pytorch 1.4 version is used for this experiment.

5 EXPERIMENTS, RESULTS, AND DISCUSSION

5.1 Baseline Description

We trained 3 different RNN-based architectures as our baselines. Each contains an Embedding layer that transforms the input word into a 256-dimension embedding vector. The embedding is passed to the RNN-based cell, whose output is then fed into a feed-forward neural network which gives us an integer corresponding to the generated word. In the 3 different architectures, we vary the RNN-based cell.

For the first baseline, we train a Vanilla RNN model that uses an RNN cell as its backbone. At every time step, the input embedding is multiplied with the hidden state of the previous time step and passed through a non-linearity (in our case, tanh) to get the new hidden state. This new hidden state is then passed into the fully-connected layer to get the generated word.

For the second baseline, we train a GRU model that uses a GRU cell as its backbone. The GRU cell contains a reset gate as well as an update gate, that help decide how much information to forget or retain from the previous hidden state. This way, GRUs have the ability to retain 'memory' as the hidden state is no longer completely replaced (like in the vanilla RNN).

In the third experiment, we train an LSTM model that uses an LSTM cell as its backbone. The LSTM cell contains input, output, and forget dates that help decide how much information to keep from the new state, how much information from the new cell state to pass on forward, and how much to forget from the existing memory.

For our training, we run the three experiments for 10 epochs, with a training batch size of 128, with each sequence having a length of 6 tokens. For each of the different experiments, we use 2 hidden layers at every time step that have a dimension of 256 each. The model is trained using an Adam optimizer, CrossEntropyLoss, and a learning rate of 0.002, along with a dropout of 0.25.

There are no prior experiments that are done with this dataset, but we think the baselines are suitable for this problem as artificial recurrent neural networks such as RNNs, along with LSTMs and GRUs that retain memory about the previously seen tokens, are shown to work reasonably well in practice [12]. Mangal et al. reach the conclusion that LSTM generates text in the quickest and most efficient way when compared to other RNN-based models [10]. However, we wish to obtain BLEU and Self-BLEU scores from the generated output to get a more comprehensive view when evaluating the different models. The repositories on which the code is based that have been referenced here [13] [3] [8].

5.2 Proposed Method

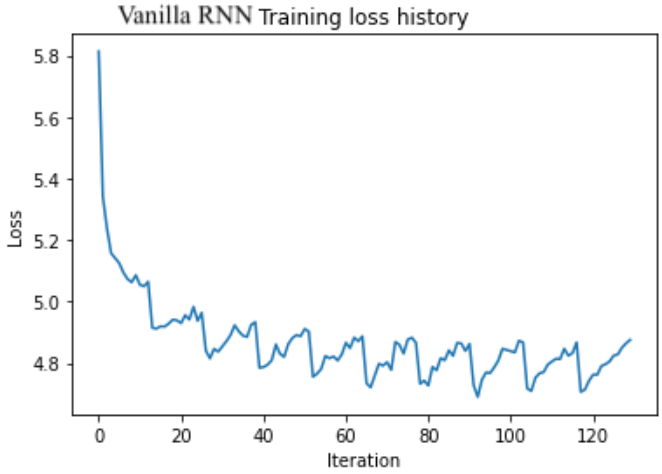
We propose to initialize the word-to-embedding matrix using different word embeddings. In our experiments, we train a Word2Vec model and then use the syn0 (word-to-embedding matrix) weights to initialize the weights of the Embedding layer. Different embedding allow us to map words or phrases from the scripts to different vectors of real numbers. We believe that these initializations would reduce the burden on the model as it would only need to fine-tune rather than learn the weights from scratch. This is due

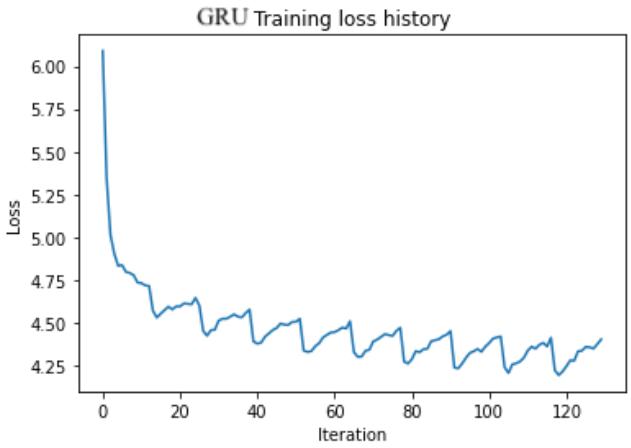
to the fact that word embeddings will contain information about how words are related in meaning, so the models will only needs to learn the specific semantics of the problem space. We used a Word2Vec model for the word embedding and we used the following parameters size=256, window=5, $\min_count = 1, workers = 4. We ran it on a cleaned data with all punctuation removed.$

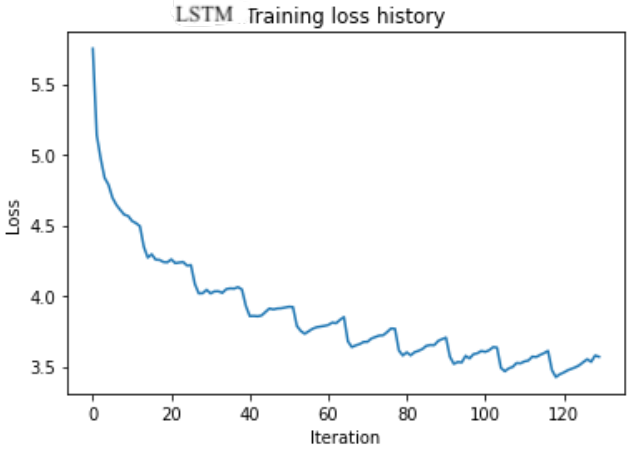
In contrast to our proposed method, the weights of the Embedding layers in the baseline models are initialized randomly. We believe that this leads to a strictly harder learning task, as not only does the model have to learn how to generate sentences, it has to also understand the syntax and semantics of the input words. Since the primary objective is to only learn how to generate sentences and not get representation of input words, initializing with learned word-embeddings would allow quicker and more efficient training as the model would only have to fine-tune these weights and no longer learn them from scratch.

5.3 Experiment Results

As we can seen from the above training loss figure, the LSTM gives the lowest training loss as compared to the other RNN-based models. All training loss eventually flatten out to a stable line as the iterations increase.







The example generated texts are showed in the next paragraph.

Example generated text

scene: the apartment.
sheldon: i'm sorry. i just wanted to talk to him.
leonard: oh, i can't tell you what?
amy: i don't know what you meant, i have no idea about it.
penny: okay. i guess i was thinking of your own culture and the pope who has crossed his own desk?
howard: i don't have coffee.
sheldon(on video): hey, i don't know why i'm on a little.

As one can see, the sentences mostly do obey the rules of grammar and do seem like reasonable sentences. However, a key takeaway is that text generation techniques usually focus on sentence-level coherence, instead of long-term structural meaning. This would mean that even though the sentences themselves make sense, there might not be a logical flow as required by a television script.

Bleu and Self-Blue Result Table

Model	Bleu	self-Bleu
Vanilla RNN (Seq Length 3)	0.746	0.645
Vanilla RNN (Seq Length 5)	0.756	0.660
Vanilla RNN (Seq Length 6)	0.703	0.748
Vanilla RNN (Seq Length 7)	**0.800**	0.700
GRU (Seq Length 3)	0.758	0.715
GRU (Seq Length 5)	0.678	0.611
GRU (Seq Length 6)	0.579	0.632
GRU (Seq Length 7)	0.700	0.634
LSTM (Seq Length 3)	0.698	0.648
LSTM (Seq Length 5)	0.724	0.589
LSTM (Seq Length 6)	0.659	**0.467**
LSTM (Seq Length 7)	0.702	0.572
Word2Vec + Vanilla RNN	0.644	0.739
Word2Vec + GRU	0.633	0.633
Word2Vec + LSTM	0.672	0.670

With a text generation project, once we generate the text, we need to know how realistic that text is. For this project, we decided

to use two metrics - the BLEU score and the Self-BLUE score - to evaluate our generated text. BLEU score is an indicator of how good/understandable out text generation is compared to human generated text. In our case we are using this score to compare the generated with all of the written scripts in our data set. For BLEU score, we want a higher score since that means there is a high correlation between our generated text and the original scripts that were used. Self-BLEU is a modification of the BLEU score where we run each sentence of the generated text against the rest of the generated text document. This score will tell us the diversity of our generated text. The lower the score the lower the correlation between the sentence and the rest of the sentences in the generated text. This means that our sentences in the test are diverse. We take the average of the scores for each sentence in the generated text, for both BLEU and Self-BLEU.

The BLEU scores of the Vanilla RNN, with a sequence length of 7, has the highest BLEU score signifying the greatest n-gram overlap with the training dataset, whereas the LSTM model has the lowest Self-BLEU score (the lower, the better), which signifies that the output generated is the most diverse.

Varying the sequence length does not seem to give a pattern, and pre-initializing with the Word2Vec embedding matrix did not improve the performance as we had hoped. The BLEU scores are a few points lower than that obtained for the baseline models, while the Self-BLEU score is around the same as that in the corresponding baseline models. The Word2Vec model may not have been able to fully distinguish the relationships between words, only throwing more chaos in to the embedding weights. This means that the models were having to about word relationships as well as the semantics. Another reason could be that we did not use enough dimensions for the Word2Vec embedding leading to words being too close in value.

6 CONCLUSION

We trained 3 different RNN-based models - Vanilla RNNs, LSTMs, GRUs, and pre-trained Word2Vecword-embeddings,and evaluated the generated text using the BLEU and the Self-BLEU metrics. The Vanilla RNN had the best BLEU whereas the LSTM model had the best Self-BLEU score.

A limitation of the proposed method, along with the baselines, is that they focus on sentence-level coherence, instead of long-term structural meaning. This means that the model learns how to generate tokens and sentences that might make sense when seen individually (local level), but fails to account for any logical flow between these sentences when view all together (global view). Tackling this limitation might possibly involve including another loss term that could address this long-term coherence and penalize generated tokens that disagree with this flow.

In regards to possible extensions of the work, one could vary the initializations of the word-to-embedding matrix and try different word embeddings models, including GLOVE and ELMO. It would also be interesting to train on a subset of data to see whether the model is capable of learning from limited data. Another direction for future work would be to vary the sentiment of the generated text as a parameter, allowing the users to have more control over the scene being generated.

7 CONTRIBUTION

All team members have contributed a similar amount of effort.

REFERENCES

[1] [n.d.]. Beautiful Soup Documentation. ([n. d.]). https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[2] [n.d.]. MoenkeyLearn. ([n. d.]). https://monkeylearn.com/api/v3/

[3] Cezanne Camacho. [n.d.]. project-tv-script-generation. ([n. d.]). https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-tv-script-generation?fbclid=IwAR2HDuogTrPprFQskcZ-j8rOGSPEscUBS-Fsf5cJv0fV4rXjSY6G6UyEghs

[4] Kristy Cecil. [n.d.]. The Big Bang Theory Transcripts. ([n. d.]). https://bigbangtrans.wordpress.com/

[5] Tong Che, Yanran Li, Ruixiang Zhang, R. Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. 2017. Maximum-Likelihood Augmented Discrete Generative Adversarial Networks. *CoRR* abs/1702.07983 (2017). arXiv:1702.07983 http://arxiv.org/abs/1702.07983

[6] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Long Text Generation via Adversarial Training with Leaked Information. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16360

[7] Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for M-gram language modeling. *1995 International Conference on Acoustics, Speech, and Signal Processing* 1 (1995), 181–184 vol.1.

[8] Koushik. [n.d.]. TV-Script-Generation. ([n. d.]). https://github.com/koushik-elite/TV-Script-Generation

[9] Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-ting Sun. 2017. Adversarial Ranking for Language Generation. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 3155–3165. http://papers.nips.cc/paper/6908-adversarial-ranking-for-language-generation.pdf

[10] Sanidhya Mangal, Poorva Joshi, and Rahul Modak. 2019. LSTM vs. GRU vs. Bidirectional RNN for script generation. arXiv:cs.CL/1908.04332

[11] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. 2016. Unrolled Generative Adversarial Networks. *CoRR* abs/1611.02163 (2016). arXiv:1611.02163 http://arxiv.org/abs/1611.02163

[12] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. *Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010* 2, 1045–1048.

[13] Mittal, Jiang, and Mekala. [n.d.]. bigBangTheoryTextGeneration. ([n. d.]). https://github.com/AustinJia/bigBangTheoryTextGeneration

[14] Frederic Morin and Yoshua Bengio. 2005. Hierarchical Probabilistic Neural Network Language Model. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, Robert G. Cowell and Zoubin Ghahramani (Eds.). Society for Artificial Intelligence and Statistics, 246–252. http://www.iro.umontreal.ca/~lisa/pointeurs/hierarchical-nnlm-aistats05.pdf

[15] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. arXiv:cs.CL/1706.03762

[17] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14344