Generating Dialogue Based on Emotions

Thivija Thavarajah 500772532

Abstract— This project examines a way to produce character dialogue given a movie script (or Multiple scripts). A small chunk of dialogue based on the script will be generated using a Neural Network. There are three parts to this process, the first part will be parsing the scripts to get the required pharses and dialogue. The second part will be taking the dialogue and assigning an emotion to it. Finally, the third part will be to take the text based on emotion and create a Computer Generated dialogue made specifically for that character. Using these three parts, the computer can produce a dialogue based off of the character and an emotion given by the user.

1 INTRODUCTION

This document describes the process behind creating a Text Generator based on character dialogue. This was created using the python code to implement a Long Short-Term Memory (LSTM) reccurent neural network to create the final dialogue and a Word to Vector (Word2Vec) model to get the context of the words and the emotions they are commonally associated with to generate the emotion-based dialogue. I applied this algorithim to several movies on one specific character (ie. *Joker*) to generate the text. Although the program does produce a computer-generated text, the accuracy and understandability of the generated text is poor.

1.1Existing Work

There are already existing works similar to this product in terms of text generation. The most common style involves the development of computer generated text from a popular book, Shakesphere, etc. I have decided to use the same concept to generate text but take it further by recognizing the emotions associated with the text first and then training the neural network to produce results based on the dialogue and the emotion behind it.

This work would be considered a text classification. Text classification is a model trained using multiple inputs that classify a text. It is often used to recognize toxic comments, spam, negative reviews, etc. I wanted to combine both of these concepts in order to generate an algorithim that would generate dialogue based on emotion for a character.

I wanted to implement both models together in order to produce unique dialogue for a character based on their emotions and generate new content for that character. This can be used to generate new scripts in the series, new dialogue for books, or even to create a chat box using the dialogue.

2 PROCEDURE

2.1 Pre-Processing

In order to get the text that the RNN will train on, the raw text data obtained from the internet must first be preprocessed. Since movie scripts have specific formats, the algorithim iterates through each line of the script and find the lines of text that were said by the one specific character we would like to use to generate dialogue for. After the information is obtained, the processed text is stored in a text file that will be used to train the RNN. The raw data used for this project were 3 movie scripts: *Joker* (2019), *Batman* (1989), and *The Dark Knight* (2008).

2.2 Recognizing Emotion Using Word2Vec

To recognize emotion, a Word2Vec model was used on the dialogues generated. A Word2Vec model is a neural network that processes text and outputs a set of vectors. Word2Vec can group vectors and similar words together in a vector space, meaning it can detect similarities between the vectors. Given a large enough data set, it can establish a word's association. Using this concept, I thought that the Word2Vec model could also associate words with emotions. Using this as a base, I trained the Word2Vec model on the script and then input emotions I would like to have the algorithim generate dialogue for. In this case I have chosen 'Happy' and 'Sad' to quickly generate text.

2.3 Recognizing Emotion using a Database

Similarily to Word2Vec, I used a database of tweets to create recognition between the tweets and the emotions associated with the tweets. Before I could train the RNN, I had to preprocess the data to get the features necessary and remove all noisy data. In this way, the emotions associated with the tweets were obtained. In order to train the model, I used an LSTM RNN to train it based on the training data (tweets) and the labels (emotions). I also added an embedding layer using the GloVe vectors. GloVe is an unsupervised learning algorithim for producing word vectors. This was used in a word-based LSTM, meaning that a collection of words was used to predict the label. The use of this embedding layer should produce more accurate results for this model that will predict the emotions of a text. Similar words will have similar vectors, therefore, the corresponding GloVe embedding for the words in the data would be obtained and stored. Using this as an embedding layer allows the model to be trained to produce an emotion from the tweets or sentences provided.

Given this, I realized this would not be efficient for the algorithim I was planning on implementing. I decided it would be better to use the Word2Vec model since it would be a more accurate representation of the script and the words that are associated with it. This model bases the emotions on the given tweets. When the tweet data is compared to the dialogue data, it is clear that they are

completely different. This means that if this method is used, it would be less accurate than the Word2Vec method.

2.4 Generating Text Using an RNN

Finally, the component that generates the text was created. I used Keras to create a generative model for text by using a character by character LSTM RNN. Since the data will be generated from a script, I thought a Char-based Neural Language model would be ideal. Compared to other models, using the Char-based model allows for a general and flexible language model. It can can handle any words and other document structures which are ideal to generate dialogue for a character, since they tend to use words that may not appear at all later in the script. There is also a smaller discrete space since only the character and its punctuation are considered, unlike vocabulary which is thousands of words. The con of a Character based RNN would be that it can also cause typos or nonsense words to be outputted from the model.

The model takes in either the entire dialogue said by the character or the lines that have an 'emotion' associated with them to generate the output. In this model, extra Embedding layers were not added. However, previous output of the model was loaded in order to generate a more accurate solution with less loss.

3 ALGORITHIMS

A main component of this program was the Keras module from python using tensorflow as backend and the Word2Vec.

3.1 Background

The Word2Vec algorithim is used to see vocabular similarities between words. In this project, it was used to find the similarity between words and the emotions associated with those words. I used the genism Word2Vec model, and algorithim that includes both the skipgram and continuous bag of word (CBOW) method.

The vector space model allows a document to be represented as a vector. Representing each word as a vector can help determine the similarities between the word itself and other words. Mapping word vectors allow for better performance in a natural language processing task.

The continuous bag of words (CBOW) predicts the probability of a word given a context. It takes the whole vocabulary and makes a Bag of Words model with the word vectors. The word vectors are fed into the neural network and then a softmax layer to get the probabilities.

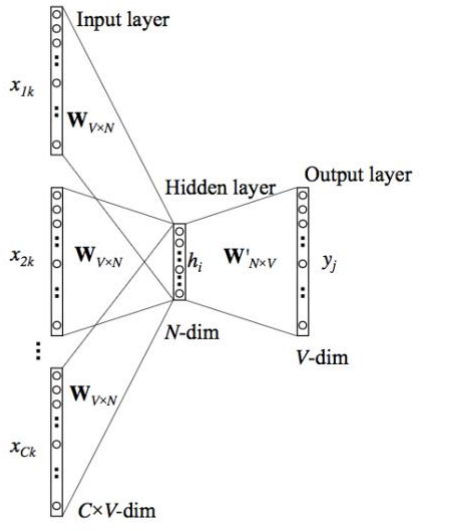

Fig 1. The Continous Bag of Words model architecture

The skip-gram method predicts the context of the word given the input words. Skip-gram will learn the word vector representations that are good at predicting nearby words in the associated context[1]. The model architecture of skip gram is similar to CBOW model, but it does the opposite. Instead, multiple words would be generated given an input word.

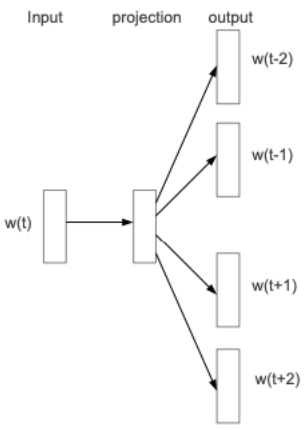

Fig 2. Skipgram architecture

To generate text using an RNN, I used the keras LTSM model. A LSTM (Long short-term memory) model is a recurrent neural network that models sequence dependent behaviour. This was used to generate a model for language.

A Recurrent Neural Network works by feeding the output of the neural network back to the input of the same neural network. A recurrent neural network is a dynamic system where the hidden state is dependent on the current observation and also the previous hidden state. It is possible to extend the hidden state into multiple layers and create a deep learning algorithim. However, the RNN can cause the vanishing gradient. Since the network receives an update proportional to the previous step with respects to the weight, it can result in the gradient being extremely small. This can prevent the weight from changing its value in the current step[6].

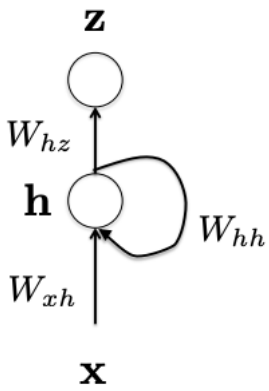

Fig 3. Reccurent Neural Network, a recursive description.

A Long short-term Memory (LTSM) is an extension of the RNN model. It has two

advantages compared to the RNN. It can contain the memory information and handle long squences better. It is an effective and scalable model for learning problems related to sequential data. It has feedback connections which help prevent the vanishing gradient problems that are common with the RNN. These problems are prevented by keeping track of arbitrary long-term dependencies in the input sequencies. Although this helps prevent it most times, it can still suffer from the exploding gradient problem. The most common architecture is composed of a cell and three regulators (the gates). Figure 4 (below) depicts a common LTSM block.

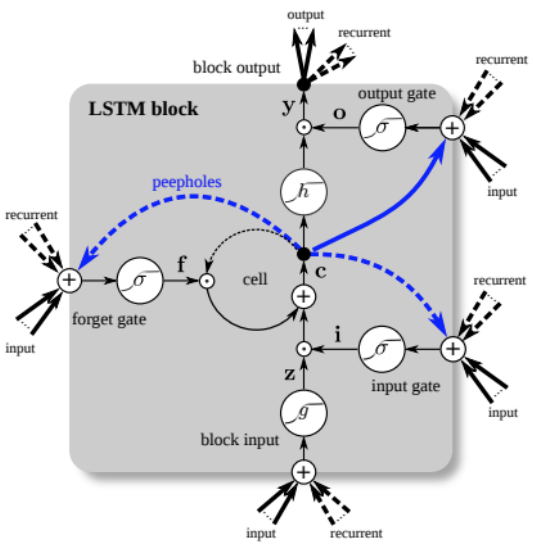

Fig 4. A simple LSTM block

3.2 Algorithim Used

For the Word2Vec algorithm, I used the genism model and loaded the vocabulary into it from the movie scripts. I decided to use action verbs instead of the names of the emotions since the action verbs would be surrounded by words that impy the emotion associated with them. For example, the word "cry" would be associated with the emotion "sad". When testing the model, I tested whether continuous bag of words would fair better when compared to the skip gram model, or if a combination of both would be better. The results of the test are shown in Figures 5-7 (below).

```
[('resident', 0.9767457246780396), ('name', 0.976630
8665275574), ('ta', 0.9765453338623047), ('tries',
0.9764868021011353), ('hoyt', 0.9764647483825684),
('mr', 0.9762318134307861), ('looks', 0.976184725761
4136), ('waiting', 0.9761800765991211), ('knock', 0.
9761164784431458), ('gary', 0.97611004114151)]
```

Fig 5. The result using only Bag of Continous Words to find words similar to "cry".

```
[('ta', 0.9994391202926636), ('questions', 0.9994210
600852966), ('easy', 0.9994189739227295), ('closin
g', 0.9994187355041504), ('tries', 0.99941539764404
3), ('ellis', 0.9994111657142639), ('bother', 0.9994
101524353027), ('serious', 0.9994097948074341), ('cl
ip', 0.9994093179702759), ('producer', 0.99940860271
45386)]
```

Fig 6. The result using only skipgram method to find words similar to "cry".

```
[('resident', 0.9977442622184753), ('name', 0.997699
2607116699), ('looks', 0.9976601004600525), ('go',
0.9976441860198975), ('hoyt', 0.9976415038108826),
('glances', 0.9976367950439453), ('mr', 0.9976356029
510498), ('aback', 0.9976320266723633), ('tries', 0.
997617781162262), ('woman', 0.997593879699707)]
```

Fig 7. The result using only skipgram method to find words similar to "cry" and setting min-count to 1.

From these results, I decided that a skipgram model would be more appropriate given the context of the movie since it had more words that could be tied to the emotion. Setting the min-count to 1 takes into account all the words that have appeared only once. Since only 3 movie scripts were used, every single word that appeared needed to be considered and have the most similar emotional words generated based on that.

To generate the text, I used the Keras LSTM model. It is a Character-based RNN that takes the sequence of characters and predicts the following character. I split the sequence length to 65 since that was the average length of the sentences in the dialogue. After I had the sequence of characters, I turned them into integers using their char values. Then I rescaled the integers into the range 0-1 so that it makes it easier for the LSTM, since it uses sigmoid activation (the gating function for the LSTM). I then set the neural networks layer, number of memory units (320), the dropout probability, and a temporary softmax. I also loaded in the network weights from the previous epoaches in order to get a better accuracy and loss score. At first, I ran 20 epoaches to estimate what would be the best number of epoaches for this given text data set.

```
model = Sequential()
model.add(LSTM(320, input_shape=(X_modified.shape[1],
               X_modified.shape[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(320))
model.add(Dropout(0.2))
model.add(Dense(Y_modified.shape[1], activation='softmax'))
model.add(Lambda(lambda x: x ** 2)) #temp softmax
```

Fig 8. Model code

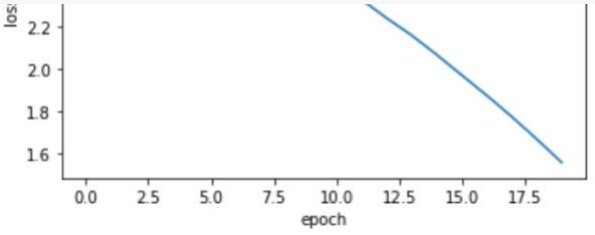

Fig 9. Model train vs. validation loss.

Layer (type)	Output Shape	Param #
lstm_8 (LSTM)	(None, 65, 320)	412160
dropout_8 (Dropout)	(None, 65, 320)	0
lstm_9 (LSTM)	(None, 320)	820480
dropout_9 (Dropout)	(None, 320)	0
dense_5 (Dense)	(None, 34)	10914
lambda_5 (Lambda)	(None, 34)	0

Total params: 1,243,554
Trainable params: 1,243,554
Non-trainable params: 0

None
Train on 124423 samples, validate on 16967 samples

Fig 10. Keras LSTM Model Summary

Based on the graph shown in Figure 9 (above), it is clear that the validation loss plateaus around 11 epoaches. Using this information, I decided to make the epoaches for this LSTM model have 11 epoaches when training.

4 RESULTS

After running the model for 7 epoches the following results were obtained (Figures 11 & 12 below).

Fig 11. Model train vs. validation loss

```
Epoch 00006: loss improved from 2.71498 to 2.64609, saving model to model-data/baseline-improvement-06-2.6461.hdf5
Epoch 7/7
24861/24861 [==============================] - 288s 12ms/step - loss: 2.5877 - accuracy: 0.2644 - val_loss: 2.5904 -
val_accuracy: 0.2760

Epoch 00007: loss improved from 2.64609 to 2.58766, saving model to model-data/baseline-improvement-07-2.5877.hdf5
```

Fig 12. Final Epoch Results

Although the loss is low, the accuracy is also low. This is most likely because I only did 7 Epoches instead of 11. I could only do 7 before my computer started to over-heat or freeze. Using this model will not be accurate but the following results were generated.

Given just the character dialogue. the Program generates the following result:

```
‘ective burke steps close to him, h
olds up the card that joker han
‘
```

from the seed, which is:

```
" ective burke steps close to him,
holds up the card that joker han "
```

It seems like the model is padding the ends with ‘ ‘ instead of generating characters.

Next, the test on the emotional text dataset was run. Only 5 epoaches for this model we run since there was less data.

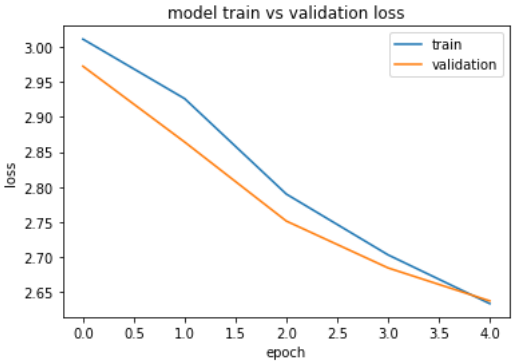

Fig 13. Model Train vs. Validation Loss

```
Epoch 00004: loss improved from 2.78984 to 2.70297, saving model to model-data/baseline-improvement-04-2.7030.hdf5
Epoch 5/5
49751/49751 [==============================] - 299s 6ms/step - loss: 2.6332 - accuracy: 0.2545 - val_loss: 2.6371 - v
al_accuracy: 0.2604

Epoch 00005: loss improved from 2.70297 to 2.63322, saving model to model-data/baseline-improvement-05-2.6332.hdf5
```

Fig 14. Final Epoch Results

Simlar to the general dialogue, there were almost similar graphs and val accuracy was also low (shown in Figures 13 & 14 above).

When this model was used to predict the text, the following result was obtained:

```
‘ginning to me. you what? what? wha
t kind of woman are you? who does t
hat?  why not? no. don't listen to
the tou to the tou to the tou to th
e tou to the tou to the tou to the
tou to the tou to the tou to the to
u to the to’
```

From the seed text:

```
" ginning to me. you what? what? wh
at kind of woman are you? who does
that?  why not? no. don't listen "
```

This model seems to repeat the characters at the end.

5 Conclusion

In conclusion, there is not enough information to come to a general consenus on whether or not this method is efficient. In order to generate the character dialogue, sentences with emotions attached to them were obtained using a Word2Vec model, which then trained the LSTM RNN model with that as an input. The LSTM RNN was also used on the general dialogue for the character to see if there would be any differences.

5.1 Improvements

There could be many improvements made to this project. I could have used a bigger database so the model could have had more data to train on and possibly produce more accurate results. There also could have been better reasources (computer) so that more epochs could have been run. This would have generated a more accurate graph to determine the parameters for the model and have a faster compile time.

This project did not produce any meaningful results and should be redone with better reasources and input data to produce better results.

References

[1] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

[2] Meyer, D. (2016). How exactly does word2vec work?. *Recuperado de http://www*, 1-4.

[3] Nicholson, C. (n.d.). *A Beginner's Guide to Word2Vec and Neural Word Embeddings*. Retrieved from https://pathmind.com/wiki/word2vec

[4] S, N. N. (2017). *An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec*. Retrieved from https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

[5] Malik, U. (2019). *Python for NLP: Word Embeddings for Deep Learning in Keras*. Retrieved from https://stackabuse.com/python-for-nlp-word-embeddings-for-deep-learning-in-keras/

[6] Chen, G. (2016). A gentle tutorial of recurrent neural network with error backpropagation. *arXiv preprint arXiv:1610.02583*.