Generating TV Scripts using Word-Level LSTM

By Madeleine Frost and Arun Subramaniam

Abstract

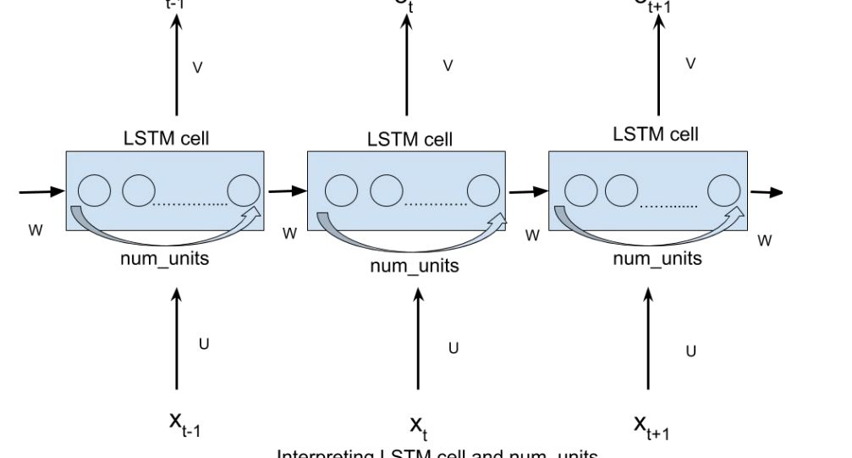
The media industry is burdened with the task of writing interesting and unique scripts. However, in our current day and age the majority of original stories have already been written in one form another. For our project we propose to generate TV-scripts for the children's television show SpongeBob SquarePants. We implemented a method of recurrent neural networks with a LSTM specifically to complete this. We demonstrate the method using a word based LSTM network and display the results within this poster.

Introduction

Recurrent Neural Networks are a class of artificial neural network that take not just their current input but also a memory of what they have previously seen. LSTM's are a variation of recurrent neural networks that were proposed in the mid 90s by the German researchers Sepp Hochrieter and Juergen Schmihuber. LSTMs have allowed for great advancements in Natural Language Processing and are used daily by major companies such as Facebook and Google, with Facebook alone performing 4.5 billion automatic translations every day using LSTMs.[1]

Aims

- Research relevant topics which relate to our project.
- Be able to acquire and process a dataset for use in a LSTM model.
- Train a model which produces TV -scripts that would pass human evaluation.
- Evaluate the success of the model and identify areas for improvement.



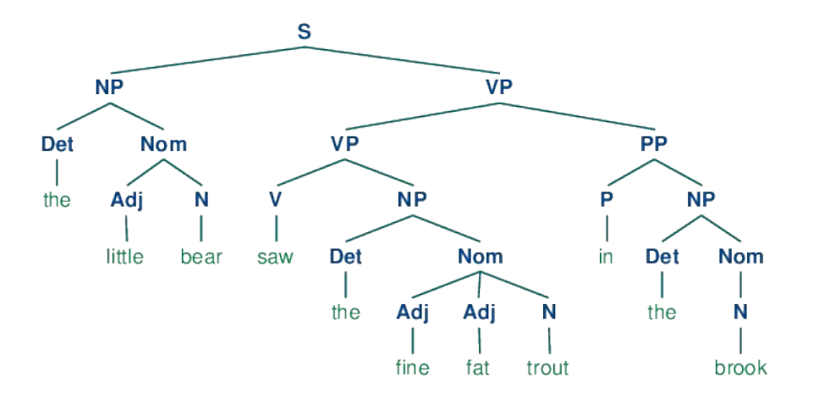
Interpreting LSTM cell and num_units

Literature Review

LSTMs are an advancement in the field of recurrent neural networks and have provided innovative advancements to many areas of NLP (Natural Language Processing) and NLG (Natural Language Generation). Our project focuses on Natural Language generation so this will therefore be the focus of this literature review.

Sentence Aggregation

When generating texts at a more advanced level there are situations when multiple sentences can be combined into a single more readable sentence. This has generally been domain dependant and previously would have to be hand crafted. Stent and Molina (2009) describe an approach to the acquisition of sentence-combining rules from a discourse tree-bank, by using only domain-independent features which are then incorporated into SPaRKY (Sentence Planning with Rhetorical Knowledge. A freely generating text data to text generator).

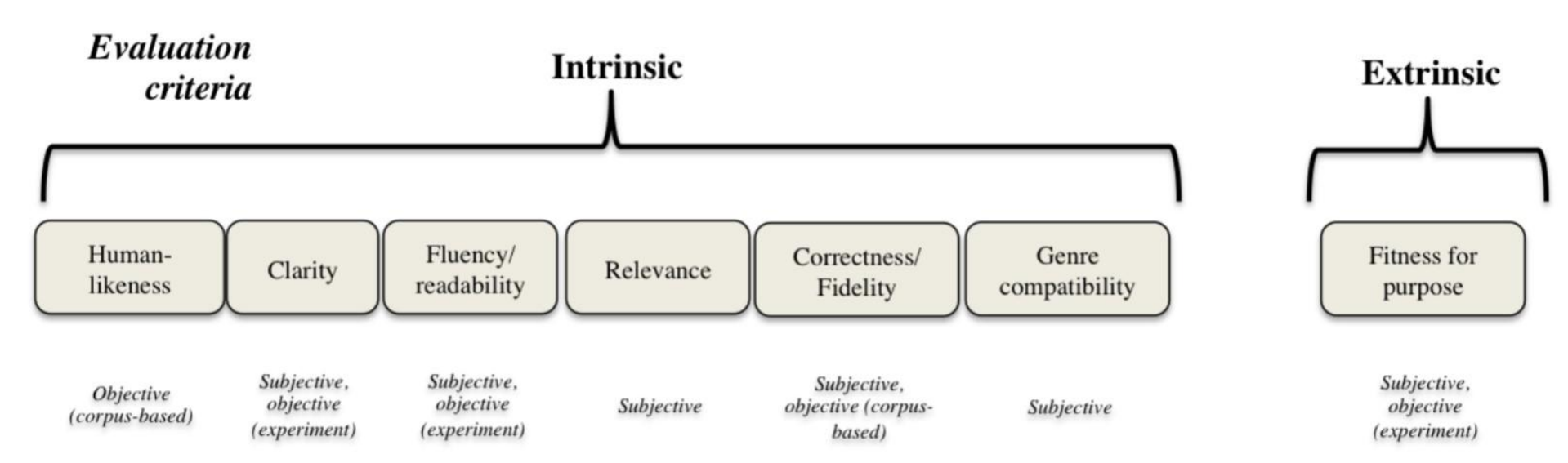


Generating with Feeling

When generating text in order to pass as human written text, many factors otherwise ignored must be introduced. Displaying emotion within text is one of them. Gupta, Walker, and Romano (2007) implemented POLLy,a system which combines a spoken language generator with an artificial intelligence planner. In doing so dialog is split into 4 strategies first exemplified by Walker et al. (1997):
"Direct strategy: Chop the tomatoes!
Approval strategy: Would it be possible for you to chop the tomatoes?
Autonomy strategy: Could you possibly chop the tomatoes?
Indirect strategy: The tomatoes aren't chopped yet."

Evaluating NLG

NLG is not easy to evaluate as it offers a lot of variety making it difficult to compare the ability of different systems. For a single input, the range of outcomes is unlimited and shows a lot of variation. It it unclear which outputs are preferred. Although variation is often preferred there are times when variation is not a goal. Weather predictions and statistical NLG tasks are an example where this is the case. The two most common methods of intrinsic evaluation are subjective human evaluation and objective measures using a corpus. Measuring N-gram overlap, String distance or content overlap.[6].



Analysis of the Problem

The following section will analyse problems which should be considered before the implementation of this project. From this, the most appropriate approaches will be chosen and justified, as well as highlighting possible challenges which may be faced because of this decision.

Firstly, a choice had to be made between using a word-based or a character-based LSTM network. A word-based LSTM network was picked as the method of choice as character-based requires a much bigger hidden layer to successfully model long-term dependencies, resulting in a higher computational cost. For example, for a sentence of 5 words, the word-based LSTM network only has to take into consideration the dependencies for 5 tokens, whereas the character-based LSTM network will have to consider the dependencies for a token size equal to the number of characters. A word-based model can therefore give a higher accuracy on a lower computation cost in comparison. Furthermore, character-based models must also learn spellings in addition to semantics and syntax, making it more prone to errors.

However, it must be taken into consideration that the corpus size for a word-based model will be very large and so will require more memory when it comes to training. Moreover, as the goal of the project is to generate TV scripts, a challenge will be to incorporate punctuation and structure into the final generated result. This is not done as simply as it would be in character-based LSTM networks, and so must be coded for accordingly.

The quality of the generated script will depend on the model's ability to understand context. Therefore, the decision between using a bidirectional or unidirectional LSTMs is important. A bidirectional LSTM will run the input in both the forwards and backwards direction, allowing the model to preserve information about both the past and the future, whereas a unidirectional LSTM will only be run in the forwards direction, only preserving information about the past. Bidirectional LSTMs are more suited to understanding context and can improve the accuracy of the model when it comes to picking the next word in the sequence. It is for that reason that the bidirectional LSTM is the chosen approach. However, it should be noted that the bidirectional approach doubles the LSTMs layers trainable parameters, making it more computationally expensive.

Another factor which should be considered whilst carrying out this project is the defining of the number of LSTM units. The number of units in an LSTM establishes the number of units in the cell, and increasing this value makes the training process more computationally expensive. Therefore, a value should be chosen which provides good results without taking too long to process. The number of units is typically defined as 128, however it is worth exploring other values to see if there is an improvement in the results produced.

After analysing some of the problems facing this project, and making decisions about which approaches to take, as well as being limited to training the model on the university labs computers, it is clear that reducing computational power and minimising memory while still producing high quality results is going to be one of the biggest challenges facing this project.

Dataset

Effective training of the LSTM model requires a large dataset of sequential data. The dataset for this project was created by scraping a website which contained the transcripts for all 469 episodes of SpongeBob and saving the results in a text file. (website available at https://spongebob.fandom.com)

Once the dataset had been created, it must undergo some initial preprocessing before it can be used to train the model. This process involves firstly removing characters which are not critical to the dataset. All letters are converted to lowercase, double spaces are changed to single spaces and unnecessary punctuation is removed. As the goal of this project is to generate a tv script, punctuation which was considered necessary to the final result was kept by converting it into the word representation of itself – for example, '!' was changed to 'exclamationmark. This was a necessary step as leaving the punctuation in its original form would result in the model not being able to distinguish between expressions such as 'hello' and 'hello!'. By making the necessary punctuation symbols their own words, it allows the neural network to incorporate the punctuation effectively into its predictions. Line breaks were also left in the dataset by being replaced by their word representation ('newline') to allow the generated output to have a similar structure to the original transcripts.

After this initial preprocessing was performed, a list of all words in their original order was created by splitting the dataset by spaces. From this, a set of all the unique words and their frequencies was generated, so that those with a frequency of less than 10 can be identified and ignored.

To generate the training set, a sentence length of 16 was defined based on the average sentence length across the entire dataset. The initial list of all words in their original order was traversed 16 words at a time, plus the next word. This 16 word sentence was then added to a list of sentences if none of the words in that sequence had a frequency of less than 10. The next word to each sentence was also added to another list, and together these two lists defined the features and labels of the training set.

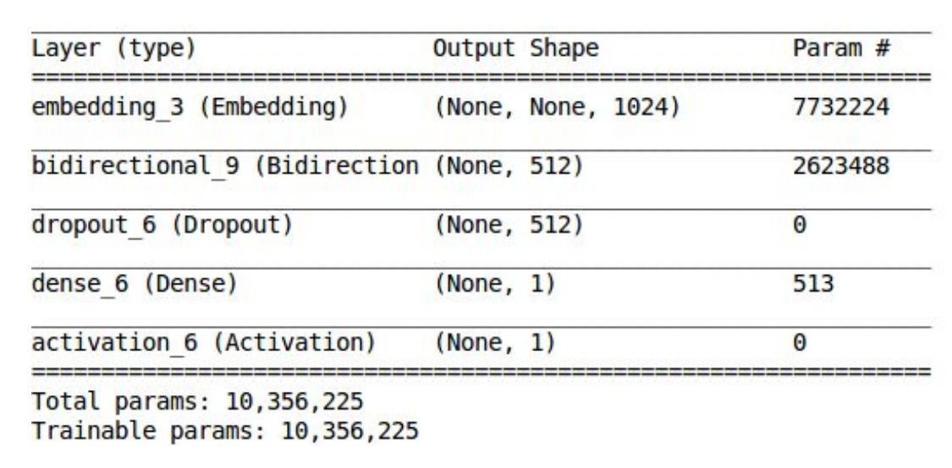
The sample below shows the dataset after all the preprocessing has been performed.

leftsquarebracket in jellyfish fields rightsquarebracket newline leftsquarebracket spongebob rightsquarebracket colon breaker breaker outer perimeter looks clear fullstop over fullstop newline leftsquarebracket patrick rightsquarebracket colon robert robert fullstop fullstop fullstop uhh fullstop fullstop fullstop ronald ronald ryan fullstop newline leftsquarebracket spongebob rightsquarebracket colon are you sure youre not trying to say roger questionmark newline leftsquarebracket patrick rightsquarebracket colon oh wait i got it fullstop ringo fullstop fullstop fullstop newline leftsquarebracket spongebob sees jellyfish rightsquarebracket newline leftsquarebracket spongebob rightsquarebracket colon patrick we have visual contact fullstop now taking evasive action fullstop subject still in close proximity fullstop over fullstop newline leftsquarebracket patrick rightsquarebracket colon hello questionmark newline leftsquarebracket spongebob rightsquarebracket colon please reply fullstop newline leftsquarebracket patrick rightsquarebracket colon i wonder if i can order pizza with these things fullstop newline leftsquarebracket spongebob rightsquarebracket colon please contact immanent patrick fullstop respond now fullstop please fullstop please exclamationmark newline leftsquarebracket patrick rightsquarebracket colon spongebob youre gonna need to speak up fullstop my eardrums arent what they used to be fullstop newline leftsquarebracket spongebob rightsquarebracket colon i cant speak up patrick theres a jellyfish here and im worried it might sting me if i make any loud fullstop fullstop fullstop leftsquarebracket spongebob bumps patrick and he screams rightsquarebracket fullstop fullstop fullstop noises fullstop newline leftsquarebracket patrick rightsquarebracket colon oops fullstop leftsquarebracket they run away fullstop the jellyfish shrugs it off rightsquarebracket

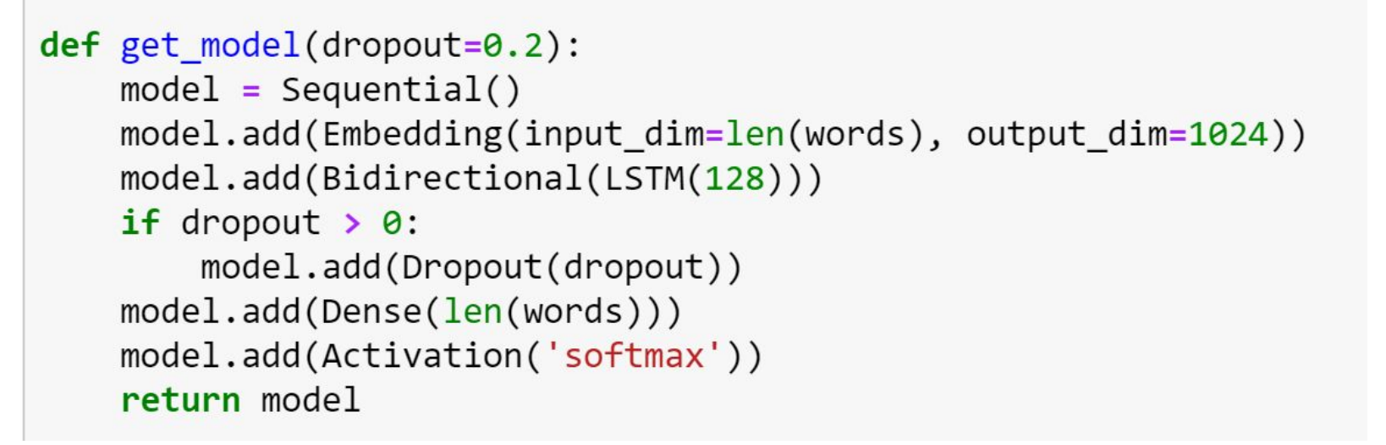
Implementation

When using Keras, training the model on the GPU is preferred due to its improved performance - however we are limited with the dedicated memory the GPU has to offer on the university computers. When implementing the model, due to the large size of the training set (sentences length was 770290 and a sentence was 16 words), embedding the words to their indices for the embedding layer cannot be completed in one go. Instead the model must be fitted using a data generator. This feeds the model data in batches which the GPU can handle.

For our implementation we used an Embedding layer to map the label encoded words to a dense vector representation. The Embedding layer is first initialized with random weights and will learn an embedding for all of the words in the training dataset. This is followed by a single Bidirectional LSTM layer with 128 units and a final dense layer of the length of the corpus. The number of trainable parameters for our model can be calculated by adding the parameters of the Embedding layer + Bidirectional LSTM layer + Dense layer



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 1024)        7732224
_________________________________________________________________
bidirectional_9 (Bidirection (None, 512)               2623488
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 513
_________________________________________________________________
activation_6 (Activation)    (None, 1)                 0
=================================================================
Total params: 10,356,225
Trainable params: 10,356,225
```



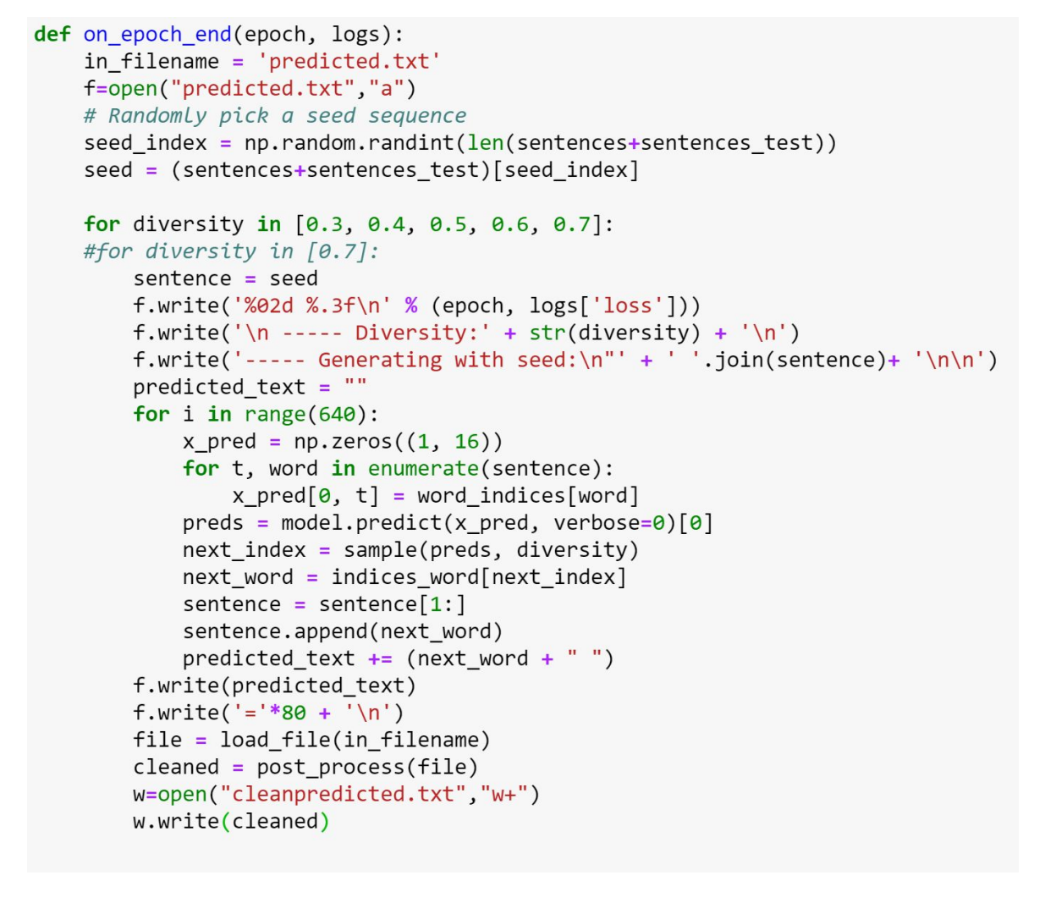
```python
def get_model(dropout=0.2):
    model = Sequential()
    model.add(Embedding(input_dim=len(words), output_dim=1024))
    model.add(Bidirectional(LSTM(128)))
    if dropout > 0:
        model.add(Dropout(dropout))
    model.add(Dense(len(words)))
    model.add(Activation('softmax'))
    return model
```

Embedding Layer = corpus size * output_dim

Bidirectional LSTM = 2 * 4 * ((1024 + 1) * 128 + (128^2)) Formula =`4(nm+n^2)`

Dense 128 * 2 * 4970 + 4970

We trained our model for 100 epochs however due to the long computational time, and the fact the models were trained on the university lab computers we were unable to finish training the model. To still gather results we implemented a method placed to be ran at the end of each epoch. In this method we gathered a random seed sentence and then had the model predict the next 200 words of the script.

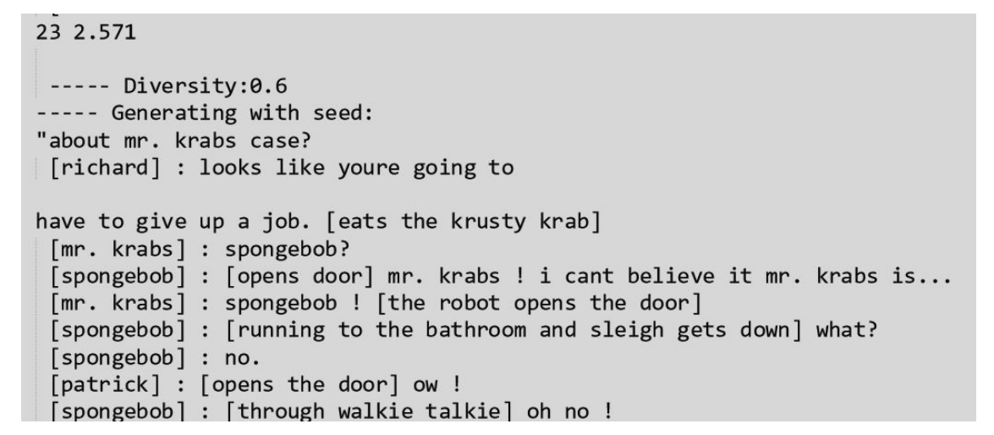


```python
def on_epoch_end(epoch, logs):
    in_filename = 'predicted.txt'
    f=open("predicted.txt","a")
    # Randomly pick a seed sequence
    seed_index = np.random.randint(len(sentences+sentences_test))
    seed = (sentences+sentences_test)[seed_index]

    for diversity in [0.3, 0.4, 0.5, 0.6, 0.7]:
    #for diversity in [0.7]:
        sentence = seed
        f.write('%02d %.3f\n' % (epoch, logs['loss']))
        f.write('\n ----- Diversity:' + str(diversity) + '\n')
        f.write('----- Generating with seed:\n"' + ' '.join(sentence)+ '\n\n')
        predicted_text = ""
        for i in range(640):
            x_pred = np.zeros((1, 16))
            for t, word in enumerate(sentence):
                x_pred[0, t] = word_indices[word]
            preds = model.predict(x_pred, verbose=0)[0]
            next_index = sample(preds, diversity)
            next_word = indices_word[next_index]
            sentence = sentence[1:]
            sentence.append(next_word)
            predicted_text += (next_word + " ")
        f.write(predicted_text)
        f.write('='*80 + '\n')
        file = load_file(in_filename)
        cleaned = post_process(file)
        w=open("cleanpredicted.txt","w+")
        w.write(cleaned)
```

Results

The following section will use discuss the results of our system. These results were generated with the finalised model discussed in the implementation section of the poster. What is most noticeable about our results is that the punctuation and structure of a script was picked up and is clearly visible. The sample result seen below was generated on the 23rd epoch. The seed sentence given for prediction is "about mr. krabs case? \n [richard]: looks like youre going to". Everything onwards was generated by our model.
.



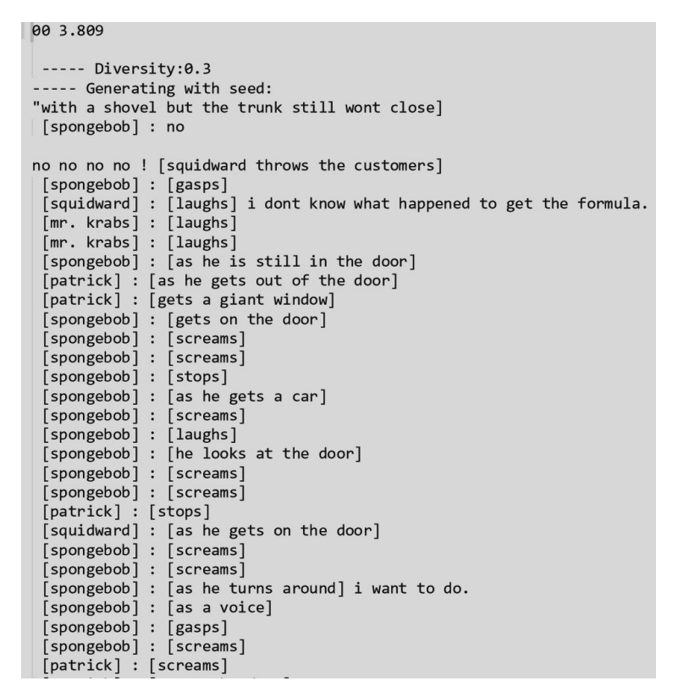
```
23 2.571

 ----- Diversity:0.6
----- Generating with seed:
"about mr. krabs case?
 [richard] : looks like youre going to

have to give up a job. [eats the krusty krab]
 [mr. krabs] : spongebob?
 [spongebob] : [opens door] mr. krabs ! i cant believe it mr. krabs is...
 [mr. krabs] : spongebob ! [the robot opens the door]
 [spongebob] : [running to the bathroom and sleigh gets down] what?
 [spongebob] : no.
 [patrick] : [opens the door] ow !
 [spongebob] : [through walkie talkie] oh no !
```

The structure of "[character] :" at the start of each line, with proper punctuation and scene directions is common throughout our results.

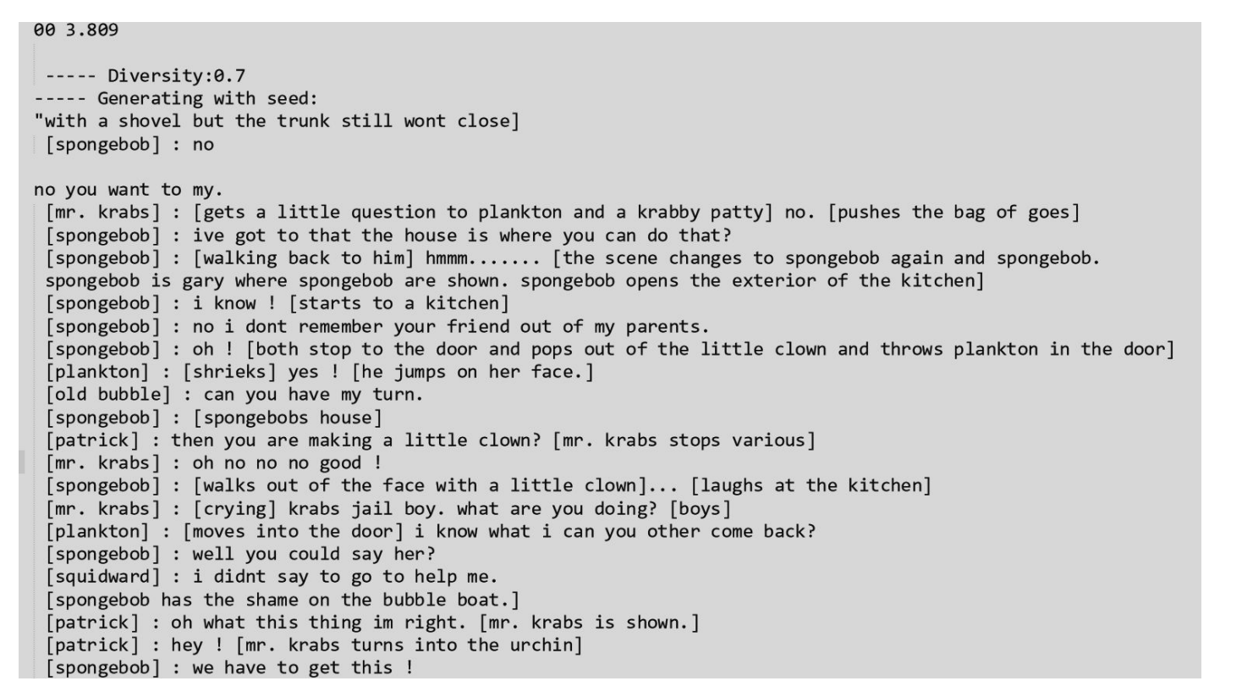
When predicting the next word the hyperparameter of temperature was adjusted to display a range of results. The temperature is used to control the "randomness" of predictions. Using a smaller temperature results in a more conservative prediction due to the meaning it is less likely to sample from unlikely candidates however using a higher temperature produces a softer distribution and results in more diversity in predictions which also leads to more mistakes. This generally lead to a lower diversity having worse results in the earlier results but better results in the later epochs and the reverse for a higher diversity.[2] This is most clearly visible in the earlier epochs, The sample results below are from epoch 1.



```
00 3.809

 ----- Diversity:0.3
----- Generating with seed:
"with a shovel but the trunk still wont close]
 [spongebob] : no

no no no no ! [squidward throws the customers]
 [spongebob] : [gasps]
 [squidward] : [laughs] i dont know what happened to get the formula.
 [mr. krabs] : [laughs]
 [mr. krabs] : [laughs]
 [spongebob] : [as he is still in the door]
 [patrick] : [as he gets out of the door]
 [patrick] : [gets a giant window]
 [spongebob] : [gets on the door]
 [spongebob] : [screams]
 [spongebob] : [screams]
 [spongebob] : [stops]
 [spongebob] : [as he gets a car]
 [spongebob] : [screams]
 [spongebob] : [laughs]
 [spongebob] : [he looks at the door]
 [spongebob] : [screams]
 [spongebob] : [screams]
 [patrick] : [stops]
 [squidward] : [as he gets on the door]
 [spongebob] : [screams]
 [spongebob] : [screams]
 [spongebob] : [as he turns around] i want to do.
 [spongebob] : [as a voice]
 [spongebob] : [gasps]
 [spongebob] : [screams]
 [patrick] : [screams]
```



```
00 3.809

 ----- Diversity:0.7
----- Generating with seed:
"with a shovel but the trunk still wont close]
 [spongebob] : no

no you want to my.
 [mr. krabs] : [gets a little question to plankton and a krabby patty] no. [pushes the bag of goes]
 [spongebob] : ive got to that the house is where you can do that?
 [spongebob] : [walking back to him] hmmm....... [the scene changes to spongebob again and spongebob.
 spongebob is gary where spongebob are shown. spongebob opens the exterior of the kitchen]
 [spongebob] : i know ! [starts to a kitchen]
 [spongebob] : no i dont remember your friend out of my parents.
 [spongebob] : oh ! [both stop to the door and pops out of the little clown and throws plankton in the door]
 [plankton] : [shrieks] yes ! [he jumps on her face.]
 [old bubble] : can you have my turn.
 [spongebob] : [spongebobs house]
 [patrick] : then you are making a little clown? [mr. krabs stops various]
 [mr. krabs] : oh no no no good !
 [spongebob] : [walks out of the face with a little clown]... [laughs at the kitchen]
 [mr. krabs] : [crying] krabs jail boy. what are you doing? [boys]
 [plankton] : [moves into the door] i know what i can you other come back?
 [spongebob] : well you could say her?
 [squidward] : i didnt say to go to help me.
 [spongebob has the shame on the bubble boat.]
 [patrick] : oh what this thing im right. [mr. krabs is shown.]
 [patrick] : hey ! [mr. krabs turns into the urchin]
 [spongebob] : we have to get this !
```

This left diagram shows a low temperature with highly conservative predictions. This is due to the resulting effect of computing the softmax on the logits / temperature. It is clear from this sample (where the only difference was the temperature) that a lower temperature leads to a smaller pick of words which often leads to a loop for the next word generated.

Evaluation

Evaluating the output of generative models is difficult due to the fact there is no automated metric which can access its accuracy, unlike with classification tasks. Therefore, the results generated by project were assessed using human evaluation to determine their quality.

A good aspect of this project was that the resulting generated scripts followed the same structure and use of punctuation as those used in the original dataset. The results had character names at the beginning of each line surrounded by square brackets and a colon to demonstrate the words which were spoken by that specific character. Furthermore, each time a new character spoke, a new line was started. The model was able to generate this structure due to the appropriate handling of the punctuation and new lines as mentioned in the preprocessing of the dataset. This allowed the model to incorporate the desired structure into its predictions, and generate better results.

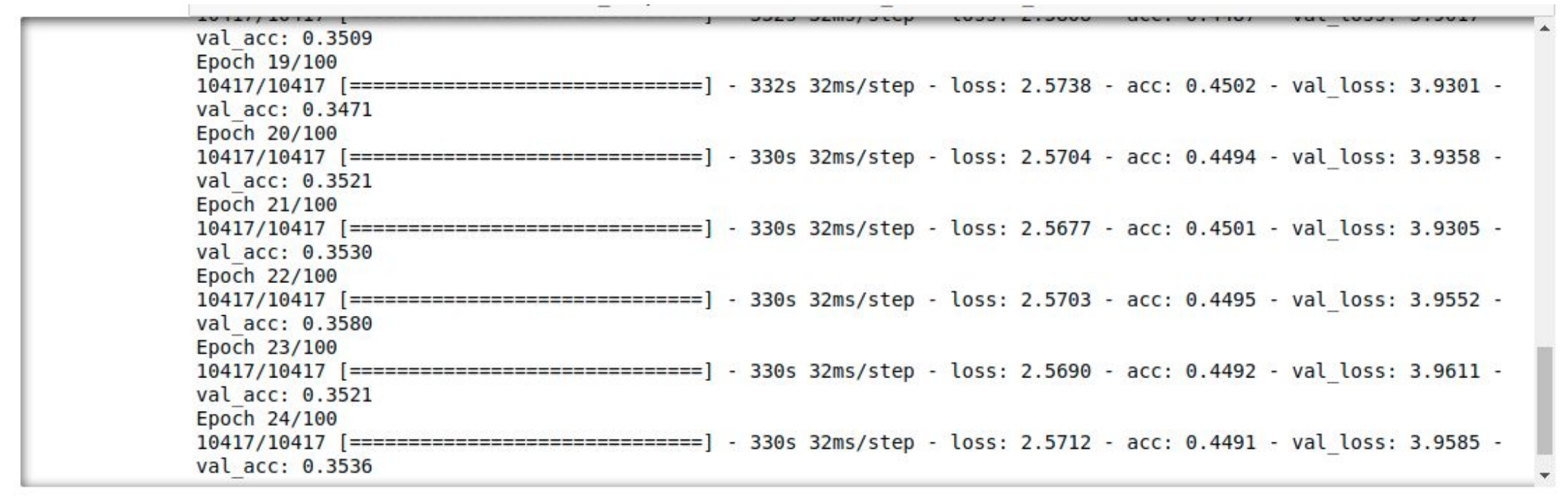
The script also produced scene directions on top of sentences spoken by the characters. Although they didn't always make sense, these directions were clearly different to how the spoken words were written and did describe the scene or behaviours of the characters.

Moreover, after looking through many of the scripts produced, it is clear that the system captured some of the individual characters personality traits. This was mainly due to the large size of the dataset and that it consisted of all the available episodes. This factor improved the quality of the results produced, and was only made possible by the use of a generator to train the model. Without it, the dataset would need to have been cut down and this would have negatively impacted on the standard of the generated scripts.

When training the model, each epoch produced an accuracy value which plateaued at around 45%. This metric is not a valid representation of the quality of the scripts produced as there is no right answer due to this not being a classification problem. A print out of a sample of their epochs and the corresponding accuracy can be seen in the figure below.

A problem with the final results was that the scripts didn't make sense the majority of the time. Although the lines seemed to have a good structure, they would not pass as human generated, which was one of the aims of this project. Achieving this is a big challenge for any generated text project, but the results could be improved by adjusting the hyper-parameters used in training, such as the batch size and learning rate. If we had access to more computational power and memory, the trainable parameters could also be increased, and together these changes could result in a higher quality of results.

Overall, this project met the aims that were set out at the beginning. Research was carried out into relevant related topics which helped with the implementation of this system, as well collecting appropriate data to create a dataset which effectively trained the model to produce tv-scripts.



```
val_acc: 0.3509
Epoch 19/100
10417/10417 [==============================] - 332s 32ms/step - loss: 2.5738 - acc: 0.4502 - val_loss: 3.9301 -
val_acc: 0.3471
Epoch 20/100
10417/10417 [==============================] - 330s 32ms/step - loss: 2.5704 - acc: 0.4494 - val_loss: 3.9358 -
val_acc: 0.3521
Epoch 21/100
10417/10417 [==============================] - 330s 32ms/step - loss: 2.5677 - acc: 0.4501 - val_loss: 3.9305 -
val_acc: 0.3530
Epoch 22/100
10417/10417 [==============================] - 330s 32ms/step - loss: 2.5703 - acc: 0.4495 - val_loss: 3.9552 -
val_acc: 0.3580
Epoch 23/100
10417/10417 [==============================] - 330s 32ms/step - loss: 2.5690 - acc: 0.4492 - val_loss: 3.9611 -
val_acc: 0.3521
Epoch 24/100
10417/10417 [==============================] - 330s 32ms/step - loss: 2.5712 - acc: 0.4491 - val_loss: 3.9585 -
val_acc: 0.3536
```

Conclusion and Future Work

To conclude, the results of this project prove that LSTMs are a very powerful subset of Recurrent Neural Networks. From evaluating the relevant research areas and the results we obtained, it is clear that LSTMs can greatly contribute to the advancements in Natural Language Generation for a wide range of uses.

Our implementation was generally evaluated as a success because it met the majority of our aims. Although there were some lines which were questionable and the context of the character placement did always make sense, the structure, differentiation between lines and scene directions as well as its ability to concatenate words into readable sentences could be concluded to be a success.

The biggest problem that we faced with our project was the lack of automated metrics for this type of natural language generation. The most common NLG metric is BLEU. This however is intended for use in evaluating text translation so does not relate to a accurate representation of our task. Finding appropriate evaluation methods is a common problem in the NLG domain.

As we were unable to finish the training of the model due to the large time it would take and the university computers not working in our favour, we were unable to save the finalized model. If we were able to save this model we could have generated a larger number of samples on command which we could have used to conduct a survey on whether the scenes created were passable. This would have given us a better metric to evaluate the projects success.

In terms of future works, this project could be tweaked and implemented for many other uses such as creating song lyrics, short stories and for use in improving human technological problems such as next word prediction in keyboards.

Arun's Evaluation

Madeleine and I collaborated for the entirety of both the implementation and write up of this project, so no credit can be solely given to one of us for any aspects as everything was a team effort.

We worked well together on each part on the project, with the combined knowledge of both of us helping to produce the final results. There were some areas that I did not understand but Madeleine did, and vice versa, and we could help to teach each other the missing gaps in our knowledge.

What I found the most challenging part of this project was understanding the number of parameters used in an LSTM and how the embedding layer worked.

The confusion for me with calculating the number of LSTM parameters was because the formula for calculating the number of trainable parameters is `4(nm+n^2)`with m being the bias + input layer and n being the size of output and n being the size of output. However as I did not initially understand the flow of the model I was unaware that the output of the Embedding layer was the input for the LSTM layer. Also I was not taking into account the fact the LSTM was Bidirectional which makes the formula* `4(nm+n^2)* 2.`

I thoroughly enjoyed contributing to this coursework as it has widened my knowledge on the differences between NLP and NLG as well as giving me a deeper understanding of both Keras and LSTMs. I believe the project was an overall success as the generated scripts were of a high quality. However, they did not always make full sense due to the context lost when generating.

Madeleine's Evaluation

Throughout the entirety of this project, Arun and I equally contributed on every aspect. We worked on the implementation of system together and both provided good inputs to help generate better quality results. By combining our knowledge, we were able to produce the finished working model and achieve the aims that we set out at the beginning of the project.

For the poster, we split up the work evenly, with each of us working on our own sections to effectively outline the work which was undertaken.

For me, the most challenging part of this coursework was overcoming the limited access to memory and computational power provided by the lab computers.

After collecting a large amount of data for the dataset, trying to make the most of it was challenging as many times the computers simply could not run the training of the model. Therefore researching and implementing the solution to this was vital as it meant that the dataset could be used to its entirety, and as a result, the generated scripts were of a better quality.

I have enjoyed undertaking this coursework, as it has taught me a lot about natural language generation. I believe this project has been successful in generating tv transcripts, however the results could better if they made more sense.

References:

[1] Pino, J, (2017) "Transitioning entirely to neural machine translation" [online] Available at: https://code.fb.com/ml-applications/transitioning-entirely-to-neural-machine-translation/

[2] Xie, Z, (2018). "Neural Text Generation: A Practical Guide". [online] Available at: https://cs.stanford.edu/~zxie/textgen.pdf

[3] Stent,A, Molina A (2009) "Evaluating Automatic Extraction of Rules for Sentence Plan Construction." Available at: https://www.researchgate.net/publication/220794547_Evaluating_Automatic_Extraction_of_Rules_for_Sentence_Plan_Construction
[4] Gupta, Walker, and Romano (2007) How Rude Are You?: Evaluating Politeness and Affect in Interaction Available at: https://link.springer.com/chapter/10.1007/978-3-540-74889-2_19
[5] Walker, M., Cahn, J. and Whittaker, S. J. (1997). Improving linguistic style: Social and affective bases for agent personality. In Proc. Autonomous Agents'97, 96–105. ACM Press
[6]Gatt, A. and Krahmer, E., 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, *61*, pp.65-170.