Introduction and Data Analysis

This dataset is the scripts from 9 seasons of an American television sitcom. The data type includes text and symbols, for example, dash (-), question mark (?), and semicolon (;). This dataset acts as training data in order to train the model which generates a new script based on patterns recognized.

Data Characterization

- Volume:
    - Roughly the number of unique words: 46,367
    - Words: 605,614
- Quality: This dataset is collected from 9 seasons of an American television sitcom named Seinfeld
- Bias: The data is collected from Seinfeld's scripts only, thus the dataset might limit the model to learning natural and various dialogues

Algorithm design

Pre-processing

1) Lookup Table

Create words embedding by creating two dictionaries to embed words to ids and ids to words and return results in the tuple form

2) Tokenize Punctuation

To avoid creating multiple ids for the same word that contains different punctuation marks, we implement the function to return a dictionary that tokenize punctuation marks into strings

3) Pre-Process Saving

To save the pre-processing data into a file, after the completion of the pre-processing procedure.

Build a model

We chose LSTM as RNNcell since it is more accurate with large datasets. LSTM, on the other hand, requires more memory and executes longer than GRU.

Train a model

1) Train Loop

Looping is implemented to repeatedly train the network over the batches for the number of epochs, while also showing the progression every couple of batches (The number of batches that the progression will be shown depends on the input, without the input, the progression will be shown every 100 batches, which is the default value).

2) Hyperparameters

We have several hyperparameters to train the model, i.e. sequence_length, batch_size, num_epochs, learning_rate, embedding_dim, hidden_dim and n_layers. However, the ones we adjusted for training are embedding_dim, hidden_dim and n_layers. Assuming having a high learning rate may result in skipping the optimal solution so we keep it at 0.001 to avoid the problem. From our experiments, increasing the hidden_dim has a higher chance of getting lower loss. Though, having higher hidden_dim also results in taking longer to train the model. As for n_layers after using other numbers, we found that 2 get the best result for our model.

3) Training

We trained the network using the data we previously obtained from the pre-processing procedure. In our training process, we used Adam as an optimizer, this is due to its fast computation time, the needs of only a few parameters for tuning, and its dynamic learning rate. Cross entropy loss was used as a loss function, to minimize the loss.

Evaluation

Long short-term memory (LSTM)

- LSTM can be used to train on data that is in form of time series or text sequential form since there is no memory associated with the model which is a problem for these kind of data and it is also a great choice for anything that has a sequence since the meaning of a word depends on the ones that preceded it.
- LSTM takes relatively longer time to train and also higher chance to overfit to a simpler model such as a CNN model due to much more parameters.

Bidirectional Long short-term memory (BiLSTM)

- BiLSTM is great for text generation since the prediction in this task not only depends on the previous input but also on the future input as well.
- BiLSTM took higher computational costs compared to LSTM simply because it has double LSTM cells.

Hyperparameters

In both network models, the performance can be improved by adjusting the hyperparameters: sequence length, number of epochs, learning rate, batch size, number of layers, hidden dimension.

- **The number of epochs** should be increased until the validation accuracy starts to decrease.
- **The learning rate** shouldn't be set too low or too high since a low learning rate will slow down the learning drastically which will take too long to run. Though, the high learning rate takes quicker to run, it may not allow the model to converge.
- **The batch size** is usually used in a form of multiples of 32, the higher it is the larger gradient steps, so it should be experimented to find the appropriate value for a data set. And as always the number of layers and the hidden layers has no final number on how many nodes or layers one should use, so depending on the individual problem a trial and error approach will give the best results.

- **The number of RNN layers** defines how many models are combined. Since the upcoming models will use the output from its previous one as an input, the more RNN layers result in more compacted result, ultimately greater loss. The number of RNN layers that is commonly used is 2 RNN layers and through our several tries of adjustment of RNN layers, we also concluded that the generally used number of 2 layers gives the lowest loss.
- **The hidden dimension** is the number of features in each hidden state. If the number is too high, it will require more time and memory to execute. However, experiments to find the proper value is needed to build the model with good results and practical runtime.

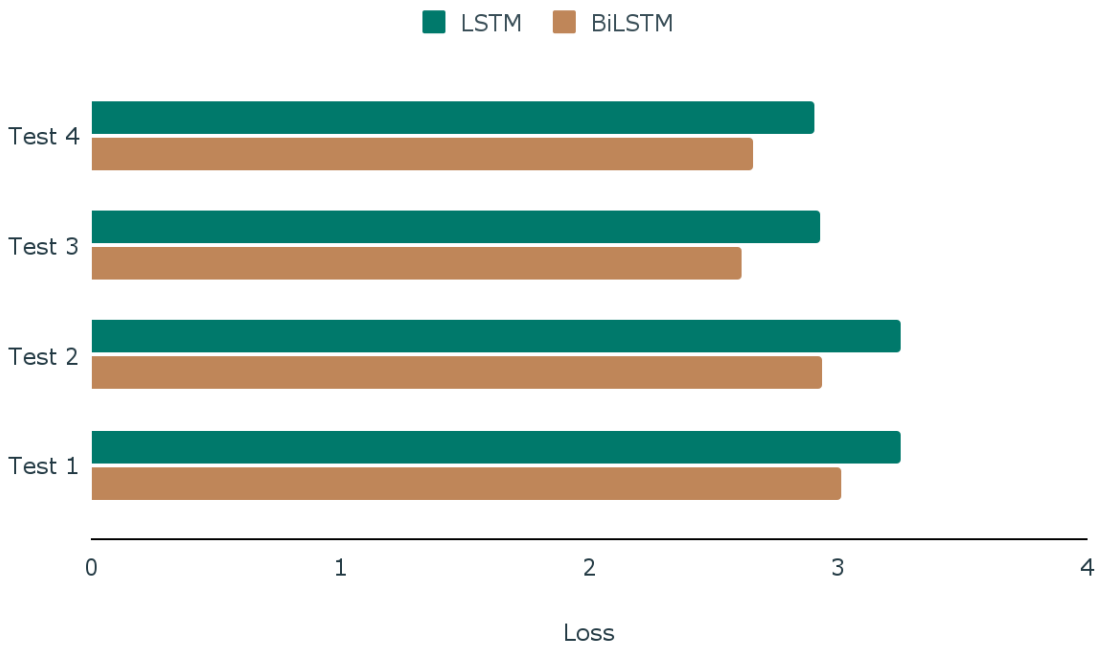

Figure 1: Compare loss value between LSTM and BiLSTM using the same hyperparameter

Conclusion

In this project, we aimed to train a model that is able to generate our own Seinfeld TV scripts. We trained 2 models using LSTM and BiLSTM in which BiLSTM takes longer to run the more the number of hidden dimensions is set and gives lower loss. After adjusting hyperparameters, we came to the conclusion that the ones with the least loss are not always appropriate to use since it may take too long to run. The most suitable model is the one that gives satisfactory results with suitable runtime which depends on which project the model will be used for.

Hyperparameters	**LSTM**		**BiLSTM**	
	Loss	*Runtime*	Loss	*Runtime*
hidden_dim = 256 embedding_dim = 256 n_layers = 2	3.2478	*33 mins*	3.0142	*37 mins*
hidden_dim = 512 embedding_dim = 256 n_layers = 2	2.9063	*44 mins*	2.6599	*1hr 20 mins*
hidden_dim = 512 embedding_dim = 512 n_layers = 2	2.9301	*57 mins*	2.6106	*1hr 11 mins*

Table 1: Compare loss value and runtime between LSTM and BiLSTM

References

Brownlee, J. (2017, July 3). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Machine Learning Mastery. Retrieved October 30, 2022, from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Hopfield, J. (n.d.). *What is the reasonable range of sequence length that RNN can handle?* Quora. Retrieved October 30, 2022, from https://www.quora.com/What-is-the-reasonable-range-of-sequence-length-that-RNN-can-handle

Koech, E. (2020, October 2). *Cross-Entropy Loss Function. A loss function used in most… | by Kiprono Elijah Koech*. Towards Data Science. Retrieved October 30, 2022, from https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

LSTM — PyTorch 1.13 documentation. (n.d.). PyTorch. Retrieved October 30, 2022, from https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html