RNN-based Seq2Seq model for Hutter Prize Wikipedia Dataset

Bernal, Jason
Department of Computer Science & Engineering
Texas A&M University
College Station, United States
ojasonbernal@tamu.edu

***Abstract*—Sequence-to-sequence models have revolutionized machine translation. They cover a range of applications and continue to be improved to this day. The construction of a sequence-to-sequence model implements various machine learning concepts ranging from recurrent neural networks, long-short term memory models, activation functions, and much more. In this paper, I discuss the procedures I took to construct a sequence-to-sequence model step-by-step and the evaluation metrics I used to evaluate the performance of the model.**

***Index Terms*—sequence-to-sequence, recurrent neural networks, long-short term memory models, model creation, model evaluation**

I. Introduction

According to [19] Machine Translation is a 20th century concept which was given birth during a time of war between countries. It's main purpose at the time was to translate foreign languages to gain an advantage between opposing countries. During the late 20th century, a new model was developed known as Statistical Machine Translation. Statistical Machine Translation placed an emphasis on the statistical principle behind the probabilities of the next predicted word. It wasn't until the 21st century that Neural Machine Translation was discovered. Neural Machine Translation revolutionized machine translation models and currently leads in performance regarding translation models. A Neural Machine Translation uses neural networks, primarily recurrent neural networks, to take a source sequence and predict a target sequence. Although it is common practice for Neural Machine Translation models to focus on the words in a given text, the sequence-to-sequence model discussed in this paper takes the perspective that text is a sequence of characters and thus aims to predict a sequence of characters.

The sequence-to-sequence (seq2seq) model makes use of several concepts ranging from Recurrent Neural Networks, Long-Short Term Memory (LSTM) models, and One-Hot Encoding. Various real-world applications of the seq2seq model range from natural language processing to human interaction with machines. The use of the seq2seq model comes with both advantages and disadvantages. Advantages of using the model include high performance and less overhead required while disadvantages include complex implementation and diminishing accuracy with longer sequences. This project restricts the input sequences to be 30 characters and the output sequences to be 10 characters. This allows for the created model to be able to have consistent high performance in training and in accuracy. The aim of this paper is to provide a deep understanding of the underlying principles behind the seq2seq model.

II. Data Preprocessing

A. Overview

The dataset that was given for the sequence-to-sequence model to work with was the Hutter Prize Wikipedia dataset. The Hutter Prize Wikipedia dataset is a well known dataset that consists of a subset of the website known as wikipedia. The dataset contains a character representation of a select few topics of the wikipedia website and its existing format in the style it is presented in. Although all characters in the dataset could be used in the training of the seq2seq model, the English characters that formed sentences were the most interpretable. I made the decision to disregard characters that aided in the style the information was presented in as it could further complicate the seq2seq model. The high-level approach I took to extract desired data was by pre-selecting the number of lines to go through the dataset, verify the first character in the beginning of a line is a character contained in the English alphabet, and use a variant of the sliding window technique to obtain combinations of 30 and 10 characters to later feed the seq2seq model. The intention behind selecting lines in a file where the beginning of a line is a character contained in the English alphabet is a heuristic approach in selecting desired data from the dataset as creating complex rules has the potential to drastically lead to suboptimal performance of the model. Based upon observation of the data, lines that start with an English character are likely English sentences, which is the desired data we seek. The data was then saved onto a file named "charInfo.csv" where the first column contained the extracted 30 sequence of characters and the following column contained the additional 10 sequence of characters extracted.

B. Implementation

Preprocessing such data was implemented through the use of user-defined methods. The following methods were created to aid in this task: read_file(), read_lines(), process_string(),

store_chars(), preprocess_data(). The method read_file() takes in a string parameter, which represents the name of the file, and is then used to open that file and store the contents of the file into a string, and returns that string. The method read_lines() takes in a string parameter, which is a line in the file, and returns lines that are English sentences as previously described. The method process_string() takes in a string parameter, which represents an English sentence, and processes the string to only contain alpha, or space characters, and returns that string. The store_chars() method takes in a string parameter, which represents a processed string, and uses a variation of the sliding window technique to store combinations of 30 and 10 character sequences into the charInfo.csv file. The method preprocess_data() does not take in any arguments but calls the mentioned methods to preprocess a file and output sequences of 30 and 10 characters into a csv file to later feed into the seq2seq model.

C. Results

Since the preprocess of data was implemented using user-defined methods, minimal space is used on the machine once fully executed. The resulting process leads to a file with sequences that are then ready to be one-hot encoded to feed into the seq2seq model.

III. Encoding Sequences

There are multiple ways to one-hot encode sequences, as mentioned in [2]. One such was is using Keras' built-in methods but I found that manually encoding such sequences is much easier to follow and understand. In order to encode the input and output sequences, it is important to identify the number of unique characters that the input and output sequences can possess. This type of information is much more important for models that translate languages since there is not always a one-to-one mapping between the characters of a language. In this model, we are strictly dealing with English character inputs and outputs, so some assumptions can be made regarding the length of the input and output sequences. Some complications that can be made are described in [6]. Although there are much more efficient methods to find such unique characters from the input and output sequences, I currently have it set to iterate through every character in all sequences to determine if each character has been seen before. I have two hash sets that contain all of the unique characters seen so far for input and output sequences respectively. Using the hash sets, I am able to determine the number of unique characters of each respective sequence using python's built-in method len(). The length of the unique characters for the input becomes the number of encoder input tokens while the length of the unique characters for the output characters becomes the number of decoder input tokens. It is important to also note that the max length that an input and output sequence can possess is 30 characters. Using this information, we can create numpy arrays, which is an acceptable form of input for the neural networks, to encode the sequences with the appropriate sizes.

IV. Model

A. Overview

Venugopalan et al. [3] mention that the sequence-to-sequence model is made up of two LSTM models. For quick reference, an LSTM or long-short term memory model is based on a recurrent neural network (rnn) model without the disadvantage of "forgetting" information after long periods of training. From there, appropriate input vectors, vectors must be in 3-dimensional, are fed into both LSTM models. One LSTM model is dedicated for the encoding component while the other is dedicated for the decoding component of the seq2seq model. In the instance of this project, we input the 30 character sequence into the encoding component and the resulting output vector, known as the context vector, is the input vector for the decoding component of the model. The output of the rnn becomes the input for the step of the rnn. The rnn continues until a stopping condition is met, either reaching the end of the sequence or reaching the max length of the intended output sequence. The resulting output vector is the predicted output sequence given the input sequence. A greedy approach takes place when obtaining the token id of the output characters. The greedy approach takes the character with the largest probability of being the desired output character based on previous input and output sequences. From there, it is common practice to then do a reverse table lookup of the decoded sequence in order to make the resulting sequence interpretable. Once the model is created and trained, sequences can be generated through the use of the seq2seq model. The LSTM models can be represented by the equations (1) and (2).

$$h_t = RNN_{enc}(x_t, h_{t-1}) \tag{1}$$

$$s_t = RNN_{dec}(y_t, s_{t-1}) \tag{2}$$

B. Implementation

Regarding implementation, I used Keras, which is a library which compliments Tensorflow in simplifying code in the creation of a neural network. I began by importing various features in Kears' library such as Model, Input, LSTM (Long Short-Term Memory), etc, to aid in the development of the seq2seq model.

I then specify certain characteristics of the model such as the batch size, the number of epochs, and the number of hidden dimensions. Using the number of input and output tokens I gathered from the previous stage, I then create encoder and decoder input layers respectively. Using the LSTM feature, I then create a LSTM for the encoder and decoder components of the neural network. Once I pass the encoder input layer in the encoder LSTM, then I receive the states of the encoder. I follow a similar procedure for the decoder LSTM but it is a bit more involved. I first make sure that the return sequence in the decoder LSTM is set to true to use the output for later use. I then pass the decoder input layer along with specifying the initial state as the encoder states previously found in the

encoder LSTM. The Dense feature is then used to predict the output sequences and the number of decoder tokens and the softmax activation function is passed as an argument for this purpose. The decoder dense method then uses the decoder outputs to redefine the decoder outputs. Using the Model feature, then I am able to pass in the encoder and decoder input layers along with the decoder outputs to produce the model.

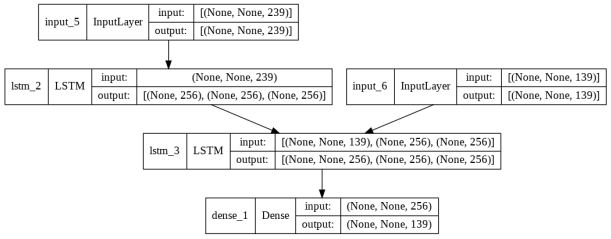

Fig. 1. Model Visualization.

The model is then compiled with the adam optimizer and the categorical cross entropy loss function. The model is then fitted with the encoder and decoder input sequences, decoder output sequences, previously specified batch size, number of epochs, and with a standard validation split of 20%. The model can be visualized in figure 1.

C. Hyperparamter Tuning

The main hyperparameter tuning procedures for the seq2seq model was by specifying the number of samples used, the batch size, number of epochs, and the number of hidden dimensions. I used Google Colab to execute my model and Google Colab does come with its limitations in machine space and GPU use. I noticed a great difference in performance when I would increase the number of samples used in conjunction with the number of epochs used. Some odd behavior that I noticed from the model was that if I specified a low number of samples and/or a low number of epochs, the model would generate sequences of words such as “and”, “the”, and “a”. It is my understanding that the model would generate these sequences since there are likely the most seen words in the dataset. Throughout my experiments, I mainly kept the number of hidden dimensions and batch size relatively the same and modified the number of epochs and the number of samples used. I trained the model with 50 epochs and 50,000 lines of sample data to obtain a highly accurate model.

D. Decoding Sequences

I created a user-defined method output_seq() which takes in a numpy array parameter, which represents an input sequence, and returns that sequence decoded. The method began by passing the input sequence into the encoder model to create an encoded sequence. A numpy array was then created to represent the output sequence. A variable that would store the decoded sequence was initialized and is the variable that is returned in the method. I also reversed the hash set to have the token id point to the character rather than the opposite that was created earlier in the stages of the model’s development. A while loop is then used to have the model continuously predict with the use of the context vector. After every iteration of the loop, the model returns the state of the output tokens and the state of the decoder model. Numpy argmax is then utilized to select the token id that has the high probability of being the desired character. To emphasize, this approach is a greedy approach which may lead to local optimizations but may suffer from long term results. The reversed hash set is then used in conjunction with the token id to obtain the corresponding output character. The state of the encoder model is then preserved for re-usability. A conditional statement handles the stopping conditions of the while loop. The stopping conditions include that of reaching the end of sequence character, which is a newline character in this project, or reaching a length that is greater than that of the max decoder sequence. Once the stopping condition has been met, the sequence has been successfully decoded and is returned as a result.

E. Generating Sequences

To generate output sequences of the model, I utilized the encoded sequences previously created in the Encoding Sequences stage of this paper. I used a for loop to be able to slice through the encoded input sequence array, which would return a given encoded input sequence. That encoded input sequence is then passed in as an argument for the earlier mentioned output_seq() method. The return of the output_seq() method is the decoded sequence of the input sequence. I then print the results for the user to be able to visually see the predictions the model makes. I print the input sequence, decoded sequence, and the complete English sentence the sequence makes. I print out the decoded sequence in bold so that it’s easier for the user to see which 10 characters were decoded.

F. Performance

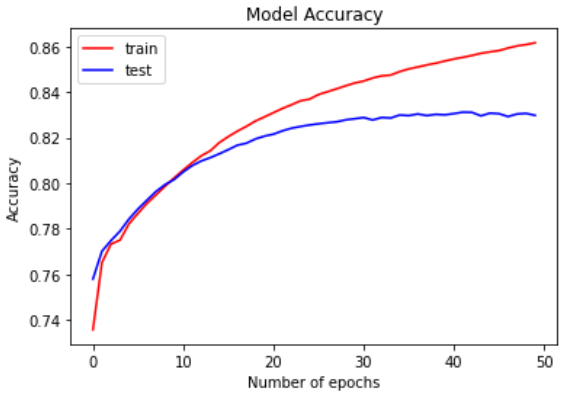

Fig. 2. Accuracy Plot with 50 epochs.

To visualize the performance of the model, I have created the following figures. The following model uses 50 epochs and 50,000 lines of sample data. With my implementation of the model, currently using more than 50,000 lines of data, more specifically near 100,000 lines of data can lead

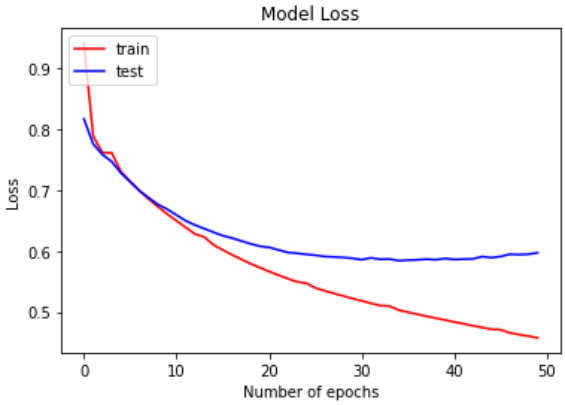

Fig. 3. Loss Plot with 50 epochs.

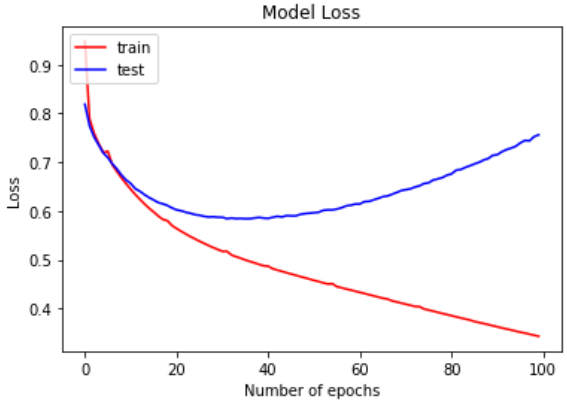

Fig. 5. Loss Plot with 100 epochs.

to my machine to terminate execution due to not having enough RAM. I find that 50,000 lines of data is just enough to interpret the performance of the model. As observed in figure 2, as the number of epochs increases, the accuracy of the model increases. This is the expected behavior of the model since the model has more time to train on the given dataset then, the model is expected to perform better over time. One can say that since there is minor overfitting with the dataset as the training accuracy increases much more than the validation accuracy. As observed in figure 3, as the number of epochs increases, the loss of the activation functions decreases since the model is performing well. The overall trajectory of the plots are performing standard to that of correct seq2seq models. This execution of the model led to a 86.16% in model accuracy and a 0.4576 in model loss.

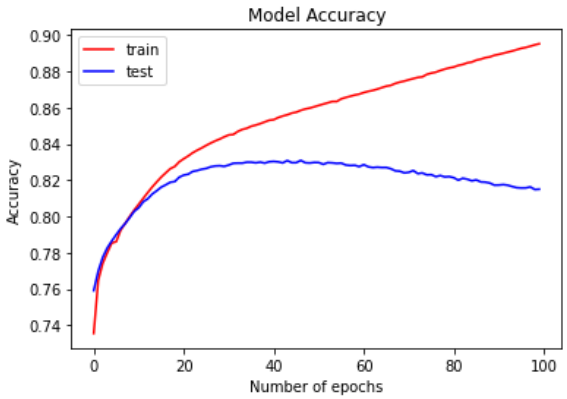

Fig. 4. Accuracy Plot with 100 epochs.

I ran another execution of the model with 100 epochs and again 50,000 lines of sample data. As well, observing figure 4, there thus seem to be much more overfitting with the training data. After increasing the number of the epochs to be 100, it is expected that the model overfits much more since the dataset has not changed but the model continues to train on it. It is important to note that a disadvantage of using neural networks is that they generally tend to overfit. The figure 5 shows that there is a decrease in both loss and validation loss of the model. This trajectory of the graph is to be expected since the accuracy of the model improved, the loss of the model decreases. This execution of the model led to a 89.52% in model accuracy and a 0.3433 in model loss.

G. Teacher Forcing

There are various techniques that can improve the seq2seq model. For more details, see [4]. One method to improve a simple seq2seq model is through the implementation of teacher forcing on the model. Teacher forcing is an architectural change to the seq2seq model for improved performance. Teacher forcing decreases the computation costs of back-propagation in the recurrent neural network and uses an output from previous time steps to feed input in future steps. [9] In our project, teacher forcing was implemented when I passed the encoder states in the decoder LSTM component of the model. The model then uses this information to assist in the prediction of sequences. One major disadvantage of using teacher forcing is that since we are using already seen sequences, it may be very difficult, if not impossible, for the model to produce a new unique output sequence that has not been seen before. This is supported in the following research paper [5].

H. BLEU Score

There are additional metrics to evaluate how well a seq2seq model has performed. The Bilingual Evaluation Understudy (BLEU) score can measure how well the output sequences of a model match that of actual human translation. This paper won't go in depth as to how the score is calculated but a surface level understanding is that it measures how similar the human and machine translation sequences are and gives an appropriate score. Although this metric is very accurate in evaluating model performance, this metric was ultimately not implemented in my seq2seq model.

V. Conclusion

Sutskever et al. [5] have noted that neural networks are very powerful models which combine a lot of machine learning concepts and principles. It can be said that neural networks are the pinnacle of machine learning knowledge. When it comes to sequence-to-sequence models, there are currently the best models in terms of performance and accuracy for machine translation. This paper discussed the procedures I took when performing this project. The journey of completing this project began by first preprocessing the Hutter Prize Wikipedia dataset. This step is arbitrarily one of the more difficult steps as there are no strict guidelines or rules as to what kind of data the model should be fed and what the output should be. Then once the data was preprocessed, to one-hot encode the data so that the sequences took an appropriate form for the model to train on. LSTM model creation and hyperparameter tuning was implemented to create the seq2seq model. The longest step was to train the model. Training the seq2seq model took a lot of time, especially to debug. After the model was trained, it was then evaluated based on the accuracy of the expected output sequences and the predicted output sequences. It was concluded that the model tends to overfit and that is a characteristic of using neural networks. In summary, sequence-to-sequence models have their limitations but perform well in the domain they are used in.

Acknowledgment

J. B. thanks the Department of Computer Science and Engineering at Texas A&M University for the quality of education he has received, especially from the Machine Learning instructors. All sources are cited in this paper, if not, they are cited in the attached Jupyter Notebook.

References

[1] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

[2] Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. N. (2017, July). Convolutional sequence to sequence learning. In International Conference on Machine Learning (pp. 1243-1252). PMLR.

[3] Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., & Saenko, K. (2015). Sequence to sequence-video to text. In Proceedings of the IEEE international conference on computer vision (pp. 4534-4542).

[4] GChiu, C. C., Sainath, T. N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., ... & Bacchiani, M. (2018, April). State-of-the-art speech recognition with sequence-to-sequence models. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 4774-4778). IEEE.

[5] Luong, M. T., Le, Q. V., Sutskever, I., Vinyals, O., & Kaiser, L. (2015). Multi-task sequence to sequence learning. arXiv preprint arXiv:1511.06114.

[6] Vinyals, O., Bengio, S., & Kudlur, M. (2015). Order matters: Sequence to sequence for sets. arXiv preprint arXiv:1511.06391.

[7] "How to write to CSV files in Python," Python Tutorial - Master Python Programming For Beginners from Scratch, 25-Oct-2020. [Online]. Available: https://www.pythontutorial.net/python-basics/python-write-csv-file/. [Accessed: 14-Dec-2021].

[8] "13.1. CSV - csv file reading and writing," 13.1. csv - CSV File Reading and Writing - Python 2.7.18 documentation. [Online]. Available: https://docs.python.org/2/library/csv.html#examples. [Accessed: 14-Dec-2021].

[9] "Character-level recurrent sequence-to-sequence model," Google Colab. [Online]. Available: https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/nlp/ipynb/lstm_seq2seq.ipynb#scrollTo=GseD3b1jWYXr. [Accessed: 14-Dec-2021].

[10] "Reading CSV files in Python," pythonspot, 24-Aug-2016. [Online]. Available: https://pythonspot.com/reading-csv-files-in-python/#: :text=CSV%20Files%20can%20be%20read%20by%20the%20Pandas,manager.%20Panda%E2%80%99s%20read_csv%20function%20can%20read%20multiple%20columns. [Accessed: 14-Dec-2021].

[11] A. Takezawa, "How to implement Seq2Seq LSTM model in Keras," Medium, 08-Mar-2021. [Online]. Available: https://towardsdatascience.com/how-to-implement-seq2seq-lstm-model-in-keras-shortcutnlp-6f355f3e5639. [Accessed: 14-Dec-2021].

[12] "Numpy.zeros," numpy.zeros - NumPy v1.21 Manual. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.zeros.html. [Accessed: 14-Dec-2021].

[13] "Python zip()," Programiz. [Online]. Available: https://www.programiz.com/python-programming/methods/built-in/zip. [Accessed: 14-Dec-2021].

[14] "NLP from scratch: Translation with a sequence to sequence network and attention," NLP From Scratch: Translation with a Sequence to Sequence Network and Attention - PyTorch Tutorials 1.10.0+cu102 documentation. [Online]. Available: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html. [Accessed: 14-Dec-2021].

[15] A. Jana, "Machine translation using recurrent neural network and pytorch," A Developer Diary, 24-Oct-2020. [Online]. Available: http://www.adeveloperdiary.com/data-science/deep-learning/nlp/machine-translation-recurrent-neural-network-pytorch/. [Accessed: 14-Dec-2021].

[16] J. Brownlee, "How to develop a Seq2Seq model for neural machine translation in Keras," Machine Learning Mastery, 07-Aug-2019. [Online]. Available: https://machinelearningmastery.com/define-encoder-decoder-sequence-sequence-model-neural-machine-translation-keras/. [Accessed: 14-Dec-2021].

[17] J. Brownlee, "Display deep learning model training history in Keras," Machine Learning Mastery, 03-Oct-2019. [Online]. Available: https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/. [Accessed: 14-Dec-2021].

[18] Jia-LuoJia-Luo, "How do I print bold text in python?," Stack Overflow, 01-Feb-1960. [Online]. Available: https://stackoverflow.com/questions/8924173/how-do-i-print-bold-text-in-python. [Accessed: 14-Dec-2021].

[19] "Stanford CS224N: NLP with Deep Learning - YouTube." [Online]. Available: https://www.youtube.com/watch?v=XXtpJxZBa2c. [Accessed: 14-Dec-2021].