Final Report Eagle Yuan and Haley Matuszak

As mentioned in our Artists' Statement, we wanted to create a Christmas poem generator. We were both very much looking forward to Christmas break and for the semester to be over. To do this, we created a Recurrent Neural Network. While poem generation seems to be just like sentence generation, we wanted to make it more interesting by allowing the network to take a rhyme scheme as input (ex. AABB CCDD E). This added two constraints that our model had to learn: 1.) To limit the poem to the specified amount of lines and 2.) to force the lines to rhyme as specified. This made poem generation not as trivial as the sentence generation we have done in the past.

We created our Christmas dataset by hand. We gathered poems from a variety of sources. We hand-picked some of the poems and excluded ones that had dark topics in them or did not seem to relate to the ideal vision of Christmas. We realized that we needed more data as poems are usually fairly short so we also included some traditional Christmas songs, formatted the same as the poems. This gave us a lot of Christmas-themed material to work with. We wanted to include other holidays around the time from different beliefs such as Kwanzaa or Hannakaa, but were worried that they would all mix together and may make references to one another by accident. While this isn't a problem, we did not want our testers to look at the poems and just say "well, this is not accurate". In total, we collected around 24,000 lines of poems with a vocabulary size of around 12000.

We used Jupyter Notebooks for our development stage, and we present our final edition of code in a script that uses version 3.9 of Python. We split the code into a training section, which would store our model, and a running section for generating poems. This made it possible to not have to run the training every time the file is reopened. Some of the important packages we used were Tensorflow and Keras, which mostly relate to the definition, training, and running of our models. We also use the Keras Tokenizer, String, and Numpy libraries to assist with data preprocessing, making sure our data is fit for the format required by the model training. We use Carnegie Mellon University's "Pronunciation" package [1]. This allowed us to check which words would rhyme with one another. Finally, we used Gensim's pretrained "glove-wiki-gigaword-300" [2] package to use as a word embedding model.

In order to tackle this issue, we first needed to clean our data set. We looked at the text file together and saw that the main issues seemed to be the inconsistent punctuation and casing of the words. Therefore, the first thing we did was remove all punctuation and capitalized letters, as well as removing unnecessary spaces within the text. Once this was done, we used the Tokenizer function in Keras to tokenize our texts. This tokenizer essentially took the whole vocabulary of the training data and encoded the words into numbers. By mapping the words to numbers, it is much easier to process through the neural network. Once we converted the whole training set to numbers, we then had to generate our gold labels. This is actually very easy to do, as we simply do an n-gram approach. For example if our dataset was [2, 4, 5, 6], our training data is [2, 4, 5] and the gold labels would be [4, 5, 6], which essentially means given a word, pair it with the next word in the sentence. This allows us to use the network as the true data for a text prediction model. Figure X below shows an example of this separation but with strings instead of numbers. Finally we batch our data to save memory and optimize it for the best performance to feed into our training model.

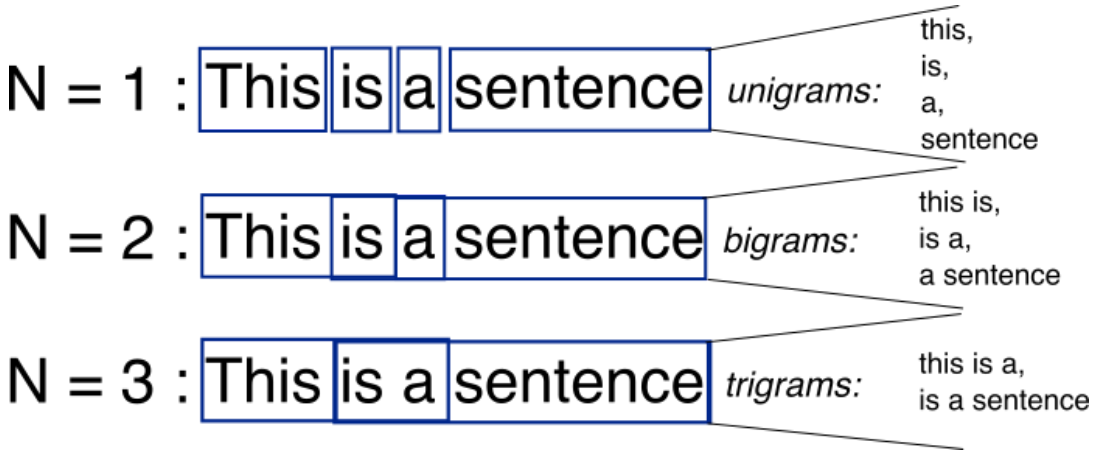

Figure 1: An example of the separation of our training data to get our gold labels. Here we use bigrams, or N = 2 [3]

When it came to defining the model, we followed the Tensorflow Text Generation Recurrent Neural Network example [4]. This helped us to define the architecture of our model: an embedding layer, a GRU layer, and a dense layer. However, where we differ in terms of our model definition is the method.

The example we found trained their network by character, however, we thought it would be more interesting to train ours by words.

Additionally, we made a few changes to our hyperparameters due to our larger and more complicated dataset:

- We increased our embedding since our vocabulary of 12,000 words was greater than their vocabulary of 60 characters
- We increased the magnitude of GRU units because we wanted greater coherence throughout the poem
- We increased the amount of epochs to 100 to try to create a unified tone. Our training file had works from many different authors and in many different styles and so we wanted to give our model a chance to learn and converge on the text better.

As a result of our changes, our runtimes greatly increased. This model should be run on accelerated systems, with a GPU or TPU. With a GPU, training takes about 15 minutes, however, a weaker processor would take days.

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        multiple                  6318592

 gru (GRU)                    multiple                  4724736

 dense (Dense)                multiple                  12649525

=================================================================
Total params: 23,692,853
Trainable params: 23,692,853
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 2: The output and summary of our model, using the tensorflow command "print(model.summary())".

It is extremely important to note that we focused on minimizing the loss, but we absolutely made sure that the model was not overfit. We were able to determine overfitting by checking to see if the model fully copied the poems from our training data. After a lot of hyperparameter testing, we discovered that a threshold loss in which it is most accurate without being overfit is around 0.4. Additionally, we discovered that the cohesion of the poems generated did not specifically depend on the sizes of the model, but rather the the value of the loss. This implies that different combinations of the embedding size, number of GRU units, and the number of epochs can produce both the same loss and same level of cohesion generation in the model.

Once the model is trained, we simply generate our poem. To do this, we first grab a starter word (which will appear as the first word in our generated poem) and rhyme scheme. The rhyme scheme is entered as a string.

Once the model makes its predictions, we then use the Keras Tokenizer again to convert the numbers keys back to words. From there, we are able to complete our rhyming. We go through the produced words, and we check the end of each sentence to see if it needs to rhyme with another line. Let's say that we need line 1 and line 3 to rhyme. We will first get all of the words that rhyme with our desired word using the CMU library mentioned earlier. So if line 1 was "ago", we find words using the pronouncing library such as "go","so","blow", etc. We then check the context for where we need the rhyme to be in line 3 by summing the Gensim pretrained embeddings (glove-wiki-gigaword-300) with the rhyming word and each word in the sentence. The word with the highest embedding count is then added at the end of the sentence. We repeat this process for all of the lines in the poem.

We will now detail out how to generate our art, using our provided Christmas poem training data or a database of your own. You will need to set up the environment and run our training python file to train the Recurrent Neural Network. This will save the model. Now you can go into our generation file and run it all the way through to generate poems. There are two parameters that you can manipulate. "next_char" is the first word in the generated poem. Some suggestions would be 'christmas','bells', or 'twas'. These must be lowercase.The second parameter is the 'rhyme_scheme'. Consider the example, "ABAB CDCD". This will produce a poem where the first and third line rhyme, the second and fourth line, the fifth and seventh and

the sixth and eighth. There will also be two stanzas, split between the fourth and fifth lines. A version of our source code is uploaded here: https://github.com/eagleyuan21/poem_rnn.

```
In [13]: next_char = ['christmas']
         rhyme scheme = 'ABBCBB'
```

Figure 3: Block 13 showing the configurable parameters in poem generation. Note that the poem will start with the word "christmas" and contain the rhyme scheme "ABBCBB".

Poetry itself is inherently hard to analyze and judge on a scale of any common or numerical metric. If we were to test with an accuracy function of simply making sure our generator produced the correct rhyme scheme, we would score near 100%. However, this is not very useful as this could still mean the poem is bad. We found that a better way to measure our performance would be to show our art to participants. We recruited 32 participants and gave them a Qualtrics Survey. They saw three sets of pairs of poems. In each pair, there would be one of our generated poems and the other would be a human-written poem from our dataset. Both would have the same rhyme scheme. In each comparison, users were asked to choose which was "the better poem". We asked them to try to ignore metaphorical and literary components and try to focus on the overall meanings and coherency. We found the following:

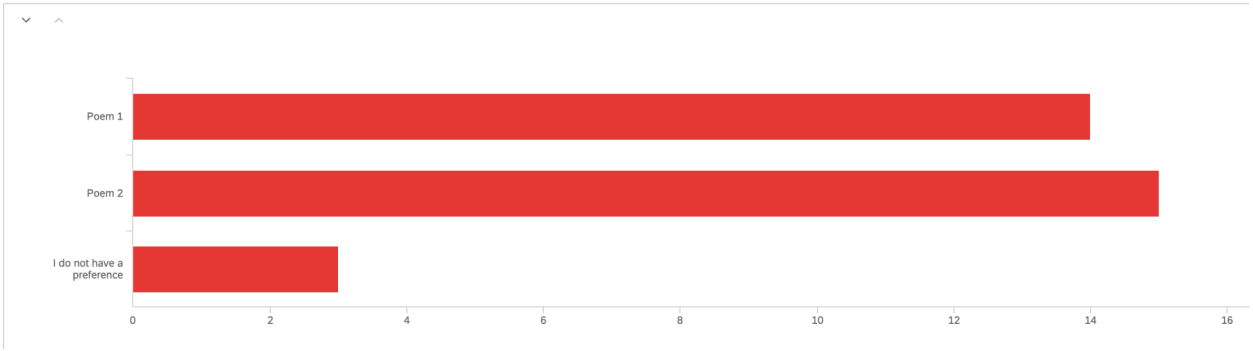

Figure 4.1: The participant votes for Comparison 1. Our poem was Poem 1

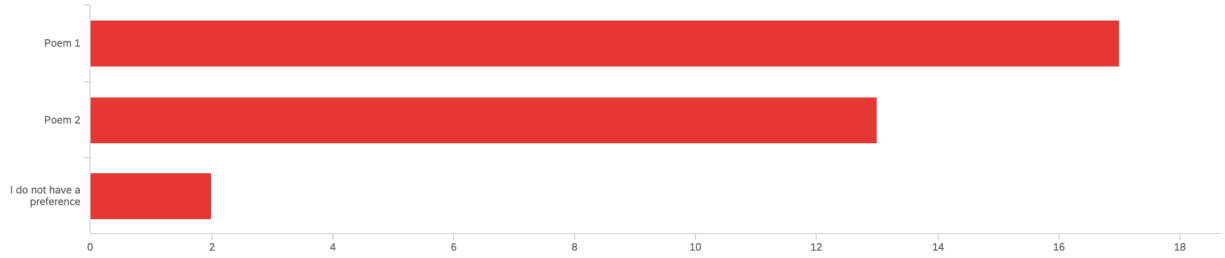

Figure 4.2: The participant votes for Comparison 2. Our poem was Poem 2

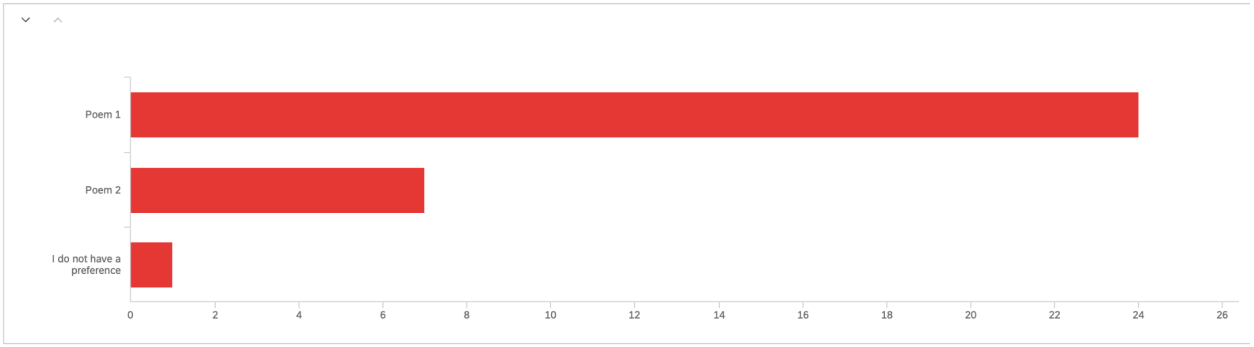

Figure 4.3: The participant votes for Comparison 3. Our poem was Poem 2

In the first two comparisons, our poems did very well. It seemed that people did not have much of a preference for the human-made poem over the generated one. In the final comparison, we think that maybe the difference can be explained by the use of the word “death” in our model’s poem. Many people commented saying that “death” did not seem “Christmas-y”.

The results we were able to see from this were full of potential. We were able to generate poems using a combination of two modern popular NLP techniques, but we hope to improve the unity of the tone even more in our model, as shown by the results of our evaluation survey. One way to do this could be to implement a different model that utilizes other techniques such as transformers. We could also continue to investigate our hyper parameters and slightly modify our layers in our model. However, what would be a great next step that would also benefit this research field is to only use one NLP technique to generate both the poem and rhyme scheme, instead of using two separate ones. Currently with a simple google scholar search, there doesn’t contain many options where the rhyming scheme is integrated into the recurrent network. It would be really interesting to create a technique in which this could be combined, however it may not be as simple as we think. Some other work includes part of speech tagging to determine the best rhyme. Although our current words rhyme and fit the context, it sometimes is not exactly grammatically correct, and maybe constraining the part of speech it needs to be would really help the integrity of the sentence. Finally if we were feeling really creative, we could investigate the possibility of generating poems that have the format of a shape we would prefer to have, making a visually appealing poem. This would probably not require any well known NLP specific tasks, but rather a brute force method or simply just using a fourier drawing.

Works Cited:

[1] Cmusphinx. (n.d.). *CMUSPHINX/CMUDICT: CMU US english dictionary*. GitHub. Retrieved December 9, 2022, from https://github.com/cmusphinx/cmudict

[2] *Gensim: Topic modelling for humans*. downloader – Downloader API for gensim - gensim. (2022, May 6). Retrieved December 9, 2022, from https://radimrehurek.com/gensim/downloader.html

[3] user2649614user2649614 44911 gold badge66 silver badges1313 bronze badges, KamranKamran 2, zoulzoul 101k4343 gold badges253253 silver badges349349 bronze badges, & codethulhucodethulhu 3. (1960, September 1). *What exactly is an n gram?* Stack Overflow. Retrieved December 9, 2022, from https://stackoverflow.com/questions/18193253/what-exactly-is-an-n-gram

[4] *Text generation with an RNN : tensorflow*. TensorFlow. (n.d.). Retrieved December 9, 2022, from https://www.tensorflow.org/text/tutorials/text_generation

Database:

Christmas poems for free -- christmas poetry, poems for Xmas. Christmas Poems for Free -- Christmas Poetry, Poems for Xmas. (n.d.). Retrieved December 9, 2022, from https://www.poemsforfree.com/xmaschristmaspoems.html

Poetry Foundation. (n.d.). *Christmas poems*. Poetry Foundation. Retrieved December 9, 2022, from https://www.poetryfoundation.org/collections/101692/christmas-poems

Shaffi, S. (2021, December 3). *25 Christmas poems that capture the spirit of the season*. Stylist. Retrieved December 9, 2022, from https://www.stylist.co.uk/books/quotes/best-christmas-poems-poetry-quotes-literature/332017

Brown Surrealist, C. W. (2021, December 11). *Christmas poems for a magic holiday atmosphere: The world of english*. The World of English | Global Language and World Culture. Retrieved December 9, 2022, from https://www.english-culture.com/christmas-poems/

Christmas poems. Poemist. (n.d.). Retrieved December 9, 2022, from https://www.poemist.com/poems/christmas

Merry Christmas quotes. (n.d.). Retrieved December 9, 2022, from https://www.quotegarden.com/christmas.html

Vogel, M. (2017, November 28). *A poem dedicated to the lighting of the Rockefeller Center Christmas Tree*. amNewYork. Retrieved December 9, 2022, from https://www.amny.com/opinion/rockerfeller-center-christmas-tree-poem-1-15234652/

What is an ogg? Christmas Poetry And Prose. (n.d.). Retrieved December 9, 2022, from https://www.hymnsandcarolsofchristmas.com/HTML/Christmas_Poetry_And_Prose1.html