**Project 3 – Eva L. Scheller and Eric Han**

**Introduction**

Team name: CaltechDanceTeam  
Group members: Eva L. Scheller and Eric Han

Division of labor:

Eva L. Scheller focused on the pre-processing for HMM and RNN and implemented the RNN model. Additionally, she provided interpretations discussion of visualizations.

Eric Han implemented the HMM model as well as additional goals that primarily focused on the HMM, including rhymes, generating haikus, and incorporating additional texts.

Design decisions of all parts were generally the product of collaboration.

**1. Preprocessing**

We had two different pre-processing methods for HMM and RNN.

***For HMM***

For HMM, we tokenized each word and each word from shakespeare.txt. We chose this because tokenizing the words would let us control the amount of syllables in each line of the poem. Additionally, this seems like an appropriate way for the HMM to attempt to recreate meaning in our poems, as certain words are more likely to follow certain words. Conceptually, the word-to-word probability is easy to comprehend qualitatively.

We set 1 sequence to be 1 line. If number of total lines is N and number of words in each line W, then we created a NxW matrix in order to represent the shakespeare.txt file. We thought this would be a good trade-off between capturing meaning within a sentence. However, we did observe that this compromised the meaning captured within the full poem, between lines.

Words with characters such as hyphens and apostrophes were included. We chose this because the use of hyphens and apostrophes in words can change their meaning dramatically. Hence splitting the words would mean that we would not be able to recreate their meaning.

Additionally, end punctuations such as comma, period, exclamation mark, question mark, colon, and semicolons were treated as separate “words”. We analyzed that Shakespeare often had punctuations within single lines in the poem, not just at the end after each line. So we thought that this method of separating the punctuations from words would help simulate punctuations within lines. Additionally, we analyzed that Shakespeare uses a lot of different punctuations. Hence, we thought that treating the punctuations separately might help the program understand when it should use a colon instead of a comma for example.

We let each of our unique words and our punctuations be represented by a number in our NxW matrix, such that there was a separate number for each unique word/punctuation. Lastly, we did not include any of the integers that separate the sonnets.

The final pre-processing pipeline worked as follows:

1. Load all word-items from shakespeare.txt
2. Remove any integers in the array of “words”
3. Create an array with all unique words and assign each unique word an index-integer
4. If a word ends with a punctuation such as comma, period, question mark, exclamation mark, color, semicolon, then we removed the last character of the word.
5. Aforementioned punctuations were added to the array of unique words and were assigned their own index-integer
6. Read the shakespeare.txt line by line and split the line by spaces
7. Reconstruct the line/sentence by finding the corresponding word/punctuation from our previous word-array. Store the index-integer in a list that represents the collection of items in the line.
8. Keep adding the list representing one line to a list of lists that represent all lines in the shakespeare.txt file.

***For RNN***

Our pre-processing for RNN was different because it was specified that it needed to be a character-based model. Through online research, it seemed that there is a general way that people usually pre-process their data for character-based LSTM. Hence we followed these methods closely. Specifically, we followed the code given by Jason Brownlee in a blog post called How to Develop a Character-Based Neural Language Model in Keras.

In the character-based RNN, of course each character has to be tokenized. We chose to tokenize separately all characters (including all letters, hyphens, apostrophes, commas, periods etc. etc.) of the text except for the integers. We removed all the integers as they were not actually used in the actual sonnet text. We saw that if we did not remove the integers, we would end up with random integers in the middle of our poems. Each of these characters was mapped to a specific integer in a dictionary. From the project description, it was given that each sequence should be 40 characters long, so that’s what we did. We chose to one-hot-encode out y-dataset. This is because we have been suggested to do this many times earlier in the course. It was also suggested by Jason Brownlee in the blog post. This is because it might exclude any correlation between the integers themselves.

The final pre-processing pipeline worked as follows:

1. All integers were removed from the text itself and it was stored as a new text-file
2. This text was loaded and each word was split by spaces. Then new sequences of 40 characters based on these words were stored in a new text-file called char\_sequences.txt. This file contained all 40 character sequences as lines with each 40 sequence starting at the next character comparative to the first character of the previous sequence.
3. Each character in the char\_sequences.txt file was load, and we constructed a dictionary that mapped each character to an integer line by line. As we read the char\_sequences.txt file, each line was stored in an array where each character was recorded as the corresponding integer from the dictionary.
4. The data were split in X and y. Here X represented all characters in the lines except the last character of each line. y represented the last character of each line.
5. y was one-hot-encoded. These final X and y were fed to the model.

**4. Poem Generation: Recurrent Neural Network**

We tried implementing two different models from online. The one that we ended up using for the project was implemented from code given by Jason Brownlee in a blog post called How to Develop a Character-Based Neural Language Model in Keras. The other model was also given by Jason Brownlee in a blog post called Text Generation With LSTM Recurrent Neural Networks in Python with Keras. Both models used only the keras package.

Of course, we modified these models to the specifics of our project, although the pre-processing was left much similar to the original code by Jason Brownlee. We created only one LSTM layer, and one output layer. We inserted a lambda layer that would control the temperature of out softmax activation, and then we had a softmax activation at the end. For our loss, we chose categorical\_crossentropy as stated in the problem description.

We tuned the following parameters: number of hidden units, number of epochs, optimizer type, and batch size. All the parameter tuning was done with only one epoch because each epoch took ~5 minutes to run (accumulated this was a pretty long time). We saw that using 200 hidden units in the LSTM layer gave an accuracy of 0.3187, loss of 2.4694, while using 100 hidden units gave accuracy of 0.3208, loss of 2.4450. Hence, using 100 hidden units seemed to be best option. We also tested rmsprop vs. adam optimizers, where adam appeared to give the highest accuracy and the smallest loss. Lastly, we ran different batch sizes of 10, 100, and 130. Here, a smaller batch size appeared to give a better result. Hence, we chose batch size of 10 as this seemed to be a good trade-off for minimizing loss and also having an effective performance rime. We also observed that generally having a higher number of epochs (we tried different epochs within a range of 1-50) maximized accuracy and minimized loss. However, due to time constraints we chose the numbers of epochs to be 20.

The final model included: batch size of 10, 100 hidden units in the LSTM layer, adam optimizer, LSTM-layer, output dense layer, Lambda-temperature layer, and 20 epochs.

Poems generated with the 3 different temperatures from our model with the seed line: ***shall I compare thee to a summer's day?\n*** can be seen below.

**Temperature = 0.25**

's alse the summer that thou shouldst bear,

My sear have I will be it the world beare,

The summer that the sweet summer that thee show,

The likened buries the praise that the stain:

So ill in the far the bright do not be song.

I like a wortated with the time and state,

And therefore that the summer that thou art,

In me that thou shouldst that the sweet state,

Like althour the time and the praise thee,

When in the far the strong that the praise.

Like a too ad a torment that the sweet state,

And therefore that the summer that thou art,

In me that thou shouldst that the sweet state,

And therefore led the sweet summer that true**,**

**Temperature = 0.75**

the stars of thee their shadows for thee,

The more that their shadow of the world,

And then the seasons of the seasons of thee.

Then the season their shadow of the seas,

Which like the willing shall be their stay,

The beauty shall the sun the sears to stay,

The sun the sears the world shall the state,

And that their shadow of the world thee thee.

Then the seasons of the seasons of their things

As their staing me to thee,

The summer's staing not so shall I thee.

Then the seasons of the seasons of their things,

As their staing me to thee,

The summer's staing not so shall I thee.

Then the seasons of the seasons of their things

**Temperature = 1.5**

the world the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

the stard the stard the stard the stard the stard

In our model we actually did not fix the line structure, because we wanted to observe whether the LSTM would be successful in learning the sonnet structure on its own. We can see that with a low temperature, the LSTM is actually very good at learning the sonnet structure. Almost all sentences carry 10 syllables, and most of them actually have the unstressed-stressed alternating pattern. Additionally, the model actually succeeded in adding punctuation at the correct places (most often at the end of sentences).

The RNN took much longer to run compared to the HMM (~2-3 hours for each training, whereas the HMM only took 5-10 minutes per training). When comparing the low-temperature RNN to the poems resulting from the HMM, they appear to be of similar quality. Of course, the character-based RNN has some problems with generating real words (e.g. *alse*, *althour*, *sear* etc.) that we don’t observe in the HMM, since the HMM is word-based. However, neither of the models generate intelligible poems. The level of grammatical structure seems to be somewhat similar in both poems.

We do observe that there is a lot of difference between poems generated by different temperatures. With the low temperature of 0.25, we observe that the sentences are very varied. All the sentences are different and they have a large variety of words. With temperature of 0.75, we observe that the poem starts off relatively good. However, we do observe a lot of repeated words and later in the poem many of the sentences are basically repeated. With temperature of 1.5, the same two words are repeated over and over again showing that no real sentence structure is captured by the model. We know that for high temperatures, all characters will have the same probability. For low temperatures, the differences between probabilities will be very high. For low temperatures, selection of the next character becomes “easier”. However, it may induce less diversity in character selection. For our model, we observed that using a lower temperature created a better result. We suspect that this is because with a low temperature the most likely next character is more easily chosen. However, we would have suspected there to be more random characters juxtaposed for the high temperature rather than the same pattern repeated over and over.

**5. Visualization & Interpretation**

In order to visualize and interpret our HMM, we did two things: 1) we created lists of the 10 words with highest probabilities for each hidden state, and 2) we used the visualize\_sparsities function from homework 6 in order to visualize the transition matrix. Based on these two visualizations, we provided interpretations on the words associated with hidden states and their respective transitions.



Figure : Here we see the transition matrix from the visualize\_sparsities function.

Our visualization of the transition matrix basically just visualizes the probability of transition from a certain state to all states. From our visualization of the transition matrix, we observe that state 0 has a preference for transitions to state 4. State 1 has a strong preference for transitions to state 0. State 2 may transition to either 1, 2, and 4 with a slight preference for state 2. State 3 transitions to either 0, 1, and 4 with a slight preference for 4. State 4 has a strong preference for transitions to 3, while state 5 has a strong preference for transitions to 2.

For our 6 hidden states, we observed that the 10 most common words observed for each state were:

***State 0***

***State 1***

***State 2***

***State 3***

***State 4***

***State 5***

From the cluster of words, we can observe that state 0 is generally associated with XX, state 1 is generally associated with XX, state 3 is generally associated with XX, state 4 is generally associated with XX, and state 5 is generally associated with XX.

Summarizing our results, it seems that XX are most likely to transition to XX etc. (see figure below).

