Arpagen: A Corpus and Baseline for Phoneme-Level Text Generation

Jared Acord, Brendan Devlin, Camille Dunning, Siu Wu
University of California, San Diego

***Abstract* – We explore the performance of a phoneme-based text generation model. Character based models have a limited amount of potential inputs and as such require high computation costs to model long term dependencies. Word-based models are accurate and require less computational costs, but in contrast to character-based, have an overwhelming input size with tens of thousands possible unique words. A phoneme-based attempts to bridge this gap by offering a greater amount of unique inputs as compared to the character-based but substantially less than a word-based model. We evaluate the performance of this phoneme-based model against a character and word based using BLEU, ROUGE, and human based metrics.**

***Keywords* – Neural Networks, Phonetics, RNN, Text Generation**

I. INTRODUCTION

Text generation is the act of generating text with the goal of the text generated to be nearly indistinguishable from human written text. There have been many tools used to model natural language, in particular, RNNs have been shown to be not only accessible but practical at a high level for many standard tasks (Mikolov 2012). In this realm of practice, the RNN model has been generally utilized by two distinct parties, one of word-based generation and the other of character-based generation. It is hard to say if either generational method is better than the other as each has their own merits and weaknesses. Word-based generation enjoys higher accuracy and lower computation cost when compared to a char-based generation due to the larger hidden layer a char-based RNN would need to model long-term dependencies. Char-based RNNs main advantage of word-based can be attributed to vastly smaller vocabulary size, with English having only 26 alphabet characters and up to 40 including symbols in contrast to the tens of thousands of unique words. Would mean a drastic decrease in memory usage. Our final project is Arpagen, a phoneme-based text generation model attempting to bridge the gap between current word-based and character-based NLP text generation models. Arpagen is a phoneme-based model that uses the 44 unique phonemes in the arpabet, nearly double compared to the standard alphabet. This keeps the input size for the phoneme-based model low, while also keeping some of the benefits of increased training data size that the word-based model has. The basic structure of this phoneme-based model will be to convert text to phonemes, run RNN on the phonemes to generate sequential phonemes, then convert the phonemes back into English text. This extra layer of conversion between phonemes and words adds an additional layer of complexity and problems which we will discuss later. Nevertheless, we were able to create a working model which shows exceptional performance in text generation. In order to test this model, we sought to create an additional two models; a word-based and a character-based model which we had mentioned earlier. The purpose of these models is to establish a baseline for which we are able to test our output against known and existing models. Our testing metrics included a variety of tools including BLEU, ROUGE, and a custom made human centric testing method.

II. DATASET

Our corpus was compiled using all or some of the text in the following books, freely available from Project Gutenberg: American Fairy Tales, Arabian Nights, Blacky the Crow, A Child's Garden of Verses, A Christmas Carol, Heidi, Little Lord Faunleroy, Mrs Peter Rabbit, Old Granny Fox, Penelope English Experiences, Peter Pan, Rambow Valley, Secret Garden, Swiss Family Robinson, Tanglewood Tales, The Jungle Book, The Three Golden Apples, Through the Looking Glass, Wind in the Willows, and The Wizard of Oz. These books were selected because they have a similar reading level (they are all children's books), were written around the same time, and have similar styles of language.

There were two steps involved in compiling our corpus from the text contained in the above books: text preprocessing and text transcription into its phonetic representation. The corpus used in training the character- and word-based text generation models was taken after the text preprocessing step, while the corpus used in the phoneme-based text generation model was taken after the additional text transcription step.

For text preprocessing, we first extracted the text from each book, then removed all of the legal and administrative text in the header and footer. We then split the text into an array of sentences, where we classified the division between sentences as being a '?', '.', or '!' character, or an uppercase letter preceded by a newline character '' (such as would be found for a chapter title). We then removed all non-alphabetic characters (numbers, punctuation, et cetera), converted all uppercase letters to lowercase, and removed all sentences of length less than 10 characters. In the end, the preprocessing of the raw text yielded a list of suitably long, strictly alphabetic lowercase sentences contained in all of the books.

For text transcription into its phonetic representation, we used the Carnegie Mellon University Pronunciation Dictionary (CMUdict) [1]. This dictionary, which is easily utilizable in python through the NLTK package, contains a list of 127,069 [2] English words with their associated phonetic transcription(s) in ARPAbet format (each phonetic representation is given as a list of phonemes for the given word). This dictionary was first relieved of all entries for words not present in the preprocessed text, reducing errors from words that were

[1] https://www.kaggle.com/rtatman/cmu-pronouncing-dictionary
[2] https://www.nltk.org/api/nltk.corpus.reader.cmudict.html

phonetically similar, but syntactically different. This reduced the CMUdict size to approximately 5,500 entries. Then, using this modified dictionary, we transcribed word-by-word for each sentence in the preprocessed text, compiling the phonetic transcription. For words that did not appear in the modified dictionary, we implemented the Jaccard distance to determine the 'Nearest-Neighbor' dictionary entry. In the end, the text transcription yielded a 3-dimensional array, with data presented in the form [sentence][word][phoneme].

III. MODEL BASELINE

For a fair head-to-head comparison, our LSTM model retains the same architecture for all three tasks: character-level, word-level, and phoneme-level generation. However, due to contrasting dataset complexities, and time constraints, we vary the training parameters (i.e., number of epochs and mini-batch sequence size) [3][4]. After flattening our three-dimensional data structure, we perform simple index encoding on each sequence of words, characters, and phonemes. The result is a list of integers, with the character encoding having the smallest number of possible integers, and the word encoding having the largest.

We also create two dictionaries, for example, with the phoneme generation task: one to map the integer indices to their corresponding phonemes, and another to map the phonemes to their indices. As the LSTM cell expects a one-hot-encoded input, we add functionality to convert the input unit to an integer, via the mapping, and create a column vector where the unit's corresponding index will have a value of 1, and the rest of the indices a value of 0.

To control the dynamics of our learning algorithm, we create training mini-batches, which each consist of multiple sequences of some desired number of sequence steps. The creation of mini-batches depends on two parameters: the number of sequences in each mini-batch (i.e., the number of equal-length parts into which we split the input array of encoded indices), and the length of each sequence. The process for creating mini-batches for phonemes is as follows:

1. Let n be the number of sequences, m be the number of steps, and k be the number of batches. Then, our input sequence must contain a total of $m \times n \times k$ units. Thus, we strip some text from longer sequences to preserve a consistent step length throughout each sequence.
2. Divide the array into n sequences, using NumPy's array reshaping functionality.
3. We now have an $n \times (m \times k)$ array. We create a $n \times m$ window on the array, moving m steps. In iterating, we also create input and target sequences, where the target sequence is shifted one phoneme forward from the corresponding input sequence.

Finally, we implement our network architecture in the PyTorch framework. The first key component of the architecture is the LSTM (long-term short-term memory) layer, so that the network can learn the order dependence in the input sequence. An LSTM layer holds an internal state that encodes past inputs and context. Vanilla RNNs fail to learn in the presence of long-term dependencies, i.e., lag of 5-10 discrete time steps. Given that we are handling longer sequences in this dataset, we find the LSTM to be the most appropriate network type for this task.

LSTM layers maintain a cell state C_t that are regulated by gates, which are composed of a pointwise multiplication operator and a sigmoid activation layer. The output of the sigmoid function is between zero and one, and it indicates how much information should be let through the gate to influence the next cell state. Consider an LSTM layer at time t. The "forget gate" within the layer is a sigmoid function that outputs a number between zero and one for cell state C_{t-1} based on hidden state h_{t-1} and input x_{t-1}. In text generation, we may want to forget information, such as gender or age, about a previous subject, when a new subject comes up within the sequence. We hypothesize that this information could be partially derived by translating input text into phonemes, but might work better with languages that have masculine and feminine words, as the "gender" of the word directly affects its phonetic translation. This new value for the cell state is denoted as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$

The next component of the LSTM layer is the "input gate", which decides which values to update (i.e., change gender pronouns of the new subject). After the "candidate values" are derived, a *tanh* layer creates a vector $\tilde{C}_t$ of these values that could alter the next cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3}$$

We update the cell state by multiplying it by f_t (dropping old information), and adding a composition of the input gate output and candidate vector (adding new information).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4}$$

Finally, the output of the LSTM layer and the next hidden state are denoted as follows:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{5}$$

$$h_t = o_t * tanh(C_t) \tag{6}$$

We hope to explore LSTM variants for this problem, such as GRU's and peephole variants. In implementing the LSTM, we consider the following parameters: input size (the number of possible phonemes), a hidden layer size, a number of layers, and the dropout probability, which we set to 0.5. Dropout is necessary for LSTM layers as a regularization technique where connecting to LSTM units are probabilistically excluded from weight updates.

We define a single dropout layer after the LSTM, with the same dropout probability as defined in the LSTM layer. After, we append a fully connected layer. We initialize the weights

[3]http://www.datascienceassn.org/sites/default/files/Subword%20Language%20Modeling%20with%20Neural%20Networks.pdf

[4]https://betterprogramming.pub/intro-to-rnn-character-level-text-generation-with-pytorch-db02d7e18d89

as a random uniform distribution between -1 and 1, and the bias as a tensor of all zeros. We initialize the hidden state and cell state with two zero tensors of shape (number of layer $\times$ number of sequences $\times$ hidden layer size). The cell states are updated during forward propagation.

IV. Training and Prediction

We use an Adam optimizer with a learning rate of 0.001 and cross-entropy loss for our criterion. We allocate a different number of epochs and batch dimensions between character-, phoneme-, and word-level models due to limited compute time.

Our output from the fully-connected layer is a distribution of scores for the next, which is converted to a categorical probability distribution via a softmax function. The next phoneme is predicted by sampling from the probability distribution, which is necessary to introduce noise and randomness into the output text. We reduce our sample space to only the K most probable phonemes. We set K equal to 5 for this project.

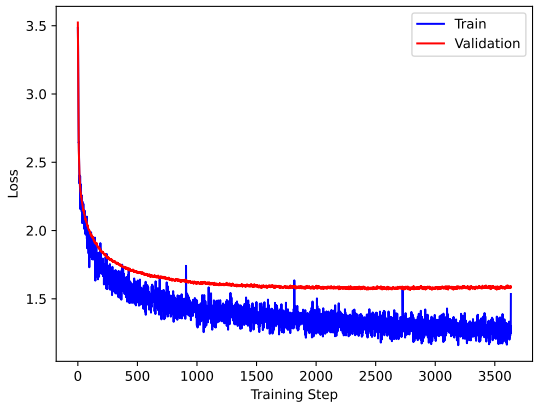
V. Results

Firstly, we shall mention that we were very ambitious with our goals. During the creation of our models, there were many issues with getting results. Because our phoneme-based model works through phoneme translation, there were many different conjunctive concepts to code, like the model itself, the translation, preprocessing the corpus, and how those things interact. Also, it was a vast challenge to create and debug 3 separate models from the ground up. We had to create and prepare 3 separate corpora, and then run preprocessing and training on each. This was a huge time sink as well as conceptually difficult at times. In addition, there were many potential problems that arose either from the initial translation or the translation of phonemes out of the NN. It should be noted that our translation requires all characters to be lower case. Regardless of these challenges, we have outputs from our phoneme-based, character-based and word-based model, from which we can start to draw conclusions.

To have approximately equal size lines of text from the output of each model, we set the number of phonemes to be 80, the number of words to 20, and the number of characters to be 80 when generating from the models.

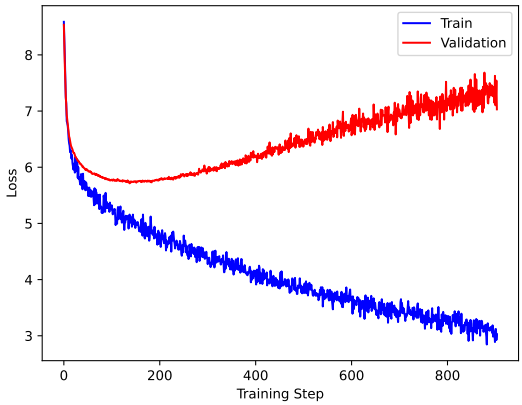
Here are some sample output lines of text and lost plots, primed with "<BOS>", for our models, trained for 20 epochs with 50 for the number of sequences and 20 for the number of steps:

Phoneme model:



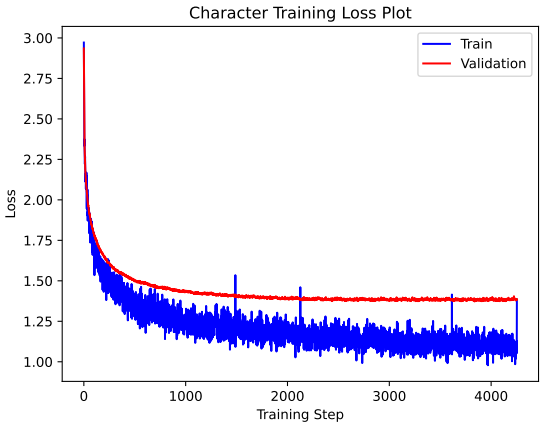
What a little boy wood let her go with me and aye don't want to b said to b hear said the
The spectators were. One of his friend and they were stopped and then when moglia had been
To her until aye could not come but he can tell myself at his heart with which their had been

Word model:



I think i should have to be told to you. She said. I am not going to
hey are all right to come. I shall tell you that i am sure that is not the same
He was a great deal of time that had been so much in a secret to know that. It

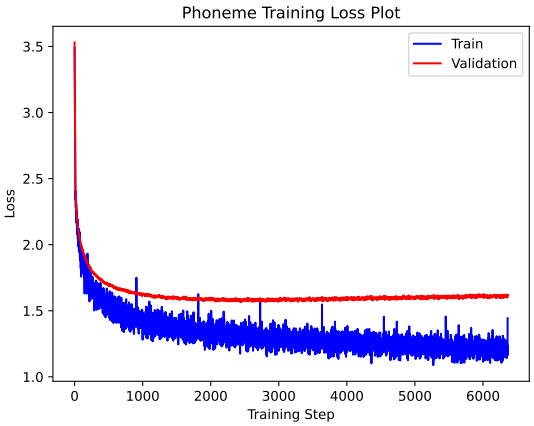
Character model:



And without horses. Said bagheera to the palace. And the sultan how to the people
Tink and striggled brother and shorted the things this marriage and said to her a
The sultan that i wouldn't have not been informed of a considerable self of the p

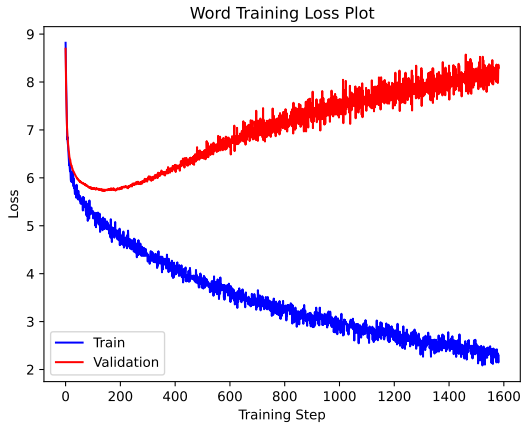
After training for 35 epochs, with 50 for our number of sequences and 20 for the number of steps, our models' outputs and loss plots were thus:

Phoneme model:



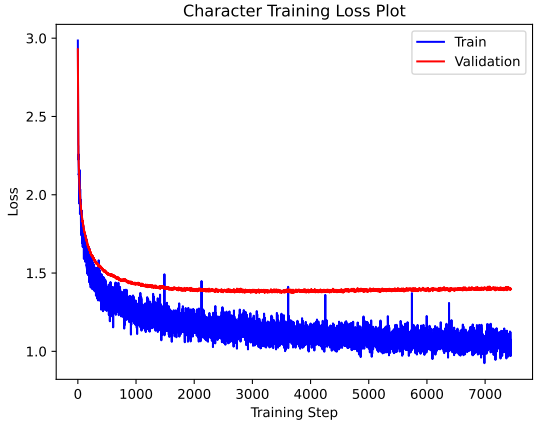
He had said the king has know read. Having looked up and was so. Aye will give me
When he had been born. What is the present for heidi and then they were so many
Said so he said to him that the princess hu was. He has stare in her friend's as he

Word model:



The earl had not been very fond of him. He had been fond of him and he did not
The boy did not know why they were all so there. The child had never seen a child before
He had seen the big man in the same manner. The sultan who has been in his life had

Character model:



That stirred he had been set out of the stairs and his feet was as if he were to
Happiness who and what was happening and sat alone to be sent to him so much the
Steady sound to bed in a moment the strange captain and the sultan and there were

After predicting and generating text using ground truth lines of text from the corpus, we used multiple comparison metrics to judge our Neural Network. For a more quantitative metric; we used BLEU to measure precision and ROUGE to measure recall, common metrics in Data science and to compare neural networks. We used the code provided by Divish Dayal[5] to do this.
We used the 50 20 20 models (num sequences, num steps, epochs) to generate the text for these metrics.

We also used a qualitative approach for comparison, a survey of comprehension. The survey works by a surveyee judging between a control and our model outputs, marking the line of text that made the most sense to the surveyee, and tabulating the results. At the end, we will have scores accumulated for the control and our models, and the selection with the highest score had the highest amount of understandability overall. Our survey also uses the 50 20 20 models to generate the text. Besides the 25 control lines of text from the corpus, we have generated, from each model, 10 lines of text primed with "<BOS> what", 10 lines of text primed with "<BOS> a", and 5 lines of text primed with "<BOS> you" for the survey.

To compare against the ground truth for phonemes, it was necessary to first convert part of the ground truth sentences into their phoneme equivalents. We used translation tools already developed, to prime the Phoneme model. To do the same for characters, we had to prime with characters separated by spaces, using the space token to mimic the space between the words to do so. For words, we simply had to prime with the given words. We should also note that, regardless of translation, all models were primed with strings that were all lower case and had no punctuation.

These quantitative metrics were done on a sample of 5 ground truth lines of text from the corpus and 5 generated

[5]https://towardsdatascience.com/how-to-evaluate-text-generation-models-metrics-for-automatic-evaluation-of-nlp-models-e1c251b04ec1

outputs from each model primed with the start of those ground truths in either phoneme, character or word form. The default is for our 50 20 20 model, where 50 is the number of sequences, 20 is the number of steps, and 20 is the number of epochs. I have also done them for our 50 20 35 models, to compare how epoch length affects the BLEU and ROUGE scores. All scores are rounded to 10 decimal places.

BLEU SCORE:
Phoneme model: 0.0227496220
Phoneme model 50 20 35: 0.0227496220

Word model: 0.0982142857
Word model 50 20 35: 0.0937499910

Character model: 0.0776699029
Character model 50 20 35: 0.0714285714

ROUGE-1 SCORE:
Phoneme model: 0.2592592593
Phoneme model 50 20 35: 0.2592592593

Word model: 0.4629629630
Word model 50 20 35: 0.4259259259

Character model: 0.4444444444
Character model 50 20 35: 0.4074074074

ROUGE-2 SCORE:
Phoneme model: 0.0327868852
Phoneme model 50 20 35: 0.0163934426

Word model: 0.1967213115
Word model 50 20 35: 0.1639344262

Character model: 0.1311475410
Character model 50 20 35: 0.1639344262

These scores are between 0 and 1, where the closer the score is to 1, the higher amount of precision or n-gram overlap there is between the ground truth text and the generated text, either for BLEU or ROUGE-N scores respectively. A high ROUGE-N score indicates a high degree of similarity between the n-grams of the two texts. Our models perform very poorly for the BLEU metric, as a score less than 10 is almost useless[6] for a NLP. Our Rouge scores are not very good as well, except the Rouge-1 scores are mediocre rather than bad. However this is simply because the Rouge-1 scores calculate the unigram overlaps between the two texts. We would expect this score to actually be much better than it is, since it is simply checking each character's overlap between the texts, it should be extremely high for a good NLP model.

Also, when considering epoch training time, it appears that increasing the number of epochs for the word model actually makes it perform worse on these precision and recall metrics. Similarly, the phoneme model performs the same or worse, when trained for more epochs. The character model performed worse for the BLEU score and the ROUGE-1 score, after increasing the number of epochs; however the ROUGE-2 score was actually better for this model.
Using the quantitative metrics, our word model performs best.

Using the qualitative metric, we would expect the control would score the highest. Therefore the 2nd highest score is the best version of our model based on human comprehension.

1st: Control with 60
2nd: word with 39
3rd: character with 16
4th: phoneme with 10

As we expected, for 5 surveyees, the control group scored the highest with a 60. The control is based on real lines of text from the corpus, so it is understandable it is the most intelligible. The next highest scoring model, which shows the best version of our base NN, was the word-based model. This shows, for this human-based comprehension metric, the word-based model is the best version of our model. The next highest was our character-based model. Finally, our phoneme model came in last, though it sometimes was chosen. This might indicate that our phoneme-based approach has some value to it, but we need to do many more optimizations to improve it to gain parity with the more typical word and character NLP models.

VI. Conclusion

We had experimented with a new type of phoneme-based text generation model, which had hoped to bridge the gap between the existing char and word based models. A corpus was generated through various selected texts, which was parsed to be run in our LSTM model which could handle word, char, and phoneme based generation. The phoneme model needed additional processing both in pre and post to convert to phonemes and to convert back to English text. The end result allowed us to be able to generate readable text from all three models and test them against each other.

The results from our testing metrics showed that our phoneme model actually performed worse compared to the word-based and char-based models. Although disappointing, such a result could be expected as there are many more clear improvements which could be done for the phoneme-based model. Initially starting this project, we had thought the most challenging part may be the creation of the LSTM model, and although development was hard, what we had failed to take into account is numerous nuances that english has and how it would affect phoneme to word or word to phoneme processing. One of the greatest challenges in this area was the presence of words which were the same but were phonetically the same. Words like altar-alter, cell-sell, die-dye, earn-urn, and many more would significantly impact our accuracy as the jaccard distance will mark these words with the same value. As such, given the values of same they will then be matched alphabetically first. We were able to slightly remedy this by compressing our arpabet dictionary and removing

[6]https://cloud.google.com/translate/automl/docs/evaluate

words which had not appeared in the corpus but, nevertheless, testing our dictionary shows that there are still over 600 words in our 5.5k long dictionary which have a matching phonetic pair. There have been solutions proposed by our peers such as a probabilistic model or a separate RNN to match phonetics to words, however, this would go beyond the scope of what we can achieve within this timeframe. Additionally, we have found that the phoneme model carries the same weakness of a char-based model with its difficulty in modeling long term dependencies.

The phoneme model does show to be an interesting alternative to char and word based models, however, the additional processing needed to convert between phoneme and words may prove to be too great a weight to justify the merits. One potential use of phoneme based generation could be in speech generation where phonemes do not need to be translated to text, allowing us to skip the exhaustive step of post processing, however, no testing has been done to that end. Before phoneme based models can be utilized at the same level as those of word and char, significant testing and research still needs to be done.