MAIS 202 Deliverable 2: Preliminary Results

Michelle Wang

1 Problem statement

The goal of this project is to implement a grapheme-to-phoneme converter that takes in a word (a sequence of graphemes) and outputs its pronunciation as a sequence of phonemes.

2 Data preprocessing

The dataset that I am using is the CMU Pronouncing Dictionary [1]. It consists of a text file, with each row containing a word, followed by two spaces and its pronunciation (a sequence of phonemes, each separated by a single space). Here are some example entries from the dictionary:

MACHINE M AH0 SH IY1 N
LEARNING L ER1 N IH0 NG

The original dictionary contains 125074 unique words and 133854 pronunciations (some words have multiple possible pronunciations). I preprocessed the data by removing any word that contained characters that were not one of the 26 letters of the alphabet. I did this because the dictionary contained the pronunciation of some symbols, and I do not think the model needs to learn them. Here are some examples of dictionary entries that were removed:

!EXCLAMATION-POINT EH2 K S K L AH0 M EY1 SH AH0 N P OY2 N T
-HYPHEN HH AY1 F AH0 N
'APOSTROPHE AH0 P AA1 S T R AH0 F IY0

In order to convert the dataset into two three-dimensional arrays, I had to find the maximum length of the words and pronunciations. The longest word in the dictionary has 34 characters. However, I found that there are only three words that have more than 20 characters, so I removed them because this way the arrays that represent my data can be smaller.

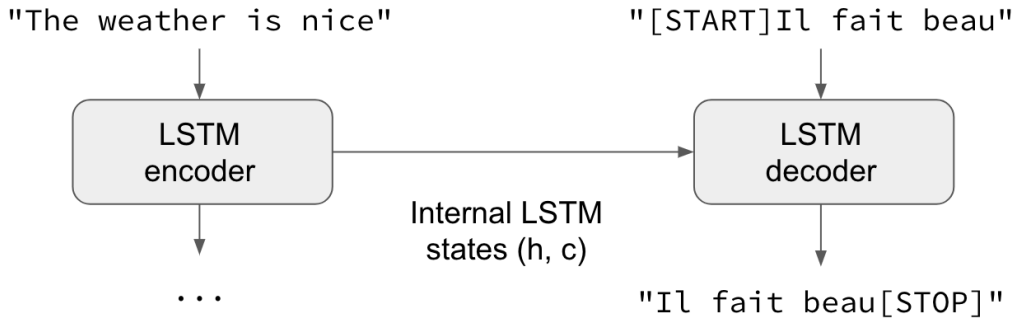
The cleaned dataset has a total of 116503 unique words and 124558 pronunciations. I first converted each word into a two-dimensional array consisting of stacked one-hot vectors, each representing a character. I then padded these word arrays with rows of zeros so that they would all have the same shape (namely 20 rows, since the maximum word length is 20). Finally, I stacked all of the words together to get a three-dimensional array. For the pronunciations, I followed a very similar procedure, using phoneme one-hot vectors instead of character one-hot vectors. I also added special START and STOP tokens at the beginning and end of each two-dimensional pronunciation array (before zero padding); they will be needed by the model later on.

3 Machine learning model

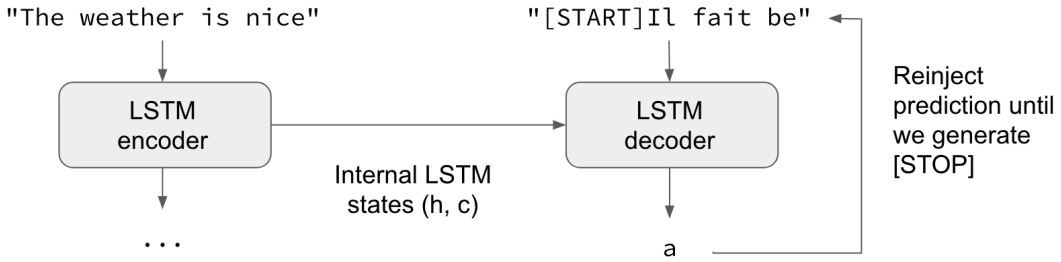
I am using a sequence-to-sequence model, with an encoder and a decoder. The encoder takes in a two-dimensional array that represents a word, then it outputs its hidden state and its internal state. The states of the encoder are used to initialize the state of the decoder.

I use a training method called "teacher forcing": during training, the decoder is first fed the START token, then it predicts the first phoneme. Next, the output of the decoder is ignored; instead, it is fed the

next true phoneme. This continues until the decoder is fed all of the phonemes. Here is a graph, taken from a tutorial on sequence-to-sequence models [2], that illustrates the teacher forcing process:



During testing, the setup is slightly different: since we cannot use the true phoneme sequence, the decoder's previous output (predicted phoneme) is used as the input in the next iteration. This continues until the decoder outputs the STOP token. This graph (from the same tutorial as the one referenced above) shows what happens during testing:



The framework I am using is the Keras library (with a Tenserflow back-end). I am using this because I found a good online example of sequence-to-sequence learning using Keras [2], and also because it seemed relatively easy to use. I am using a bi-directional Long Short-Term Memory neural network (BLSTM) for my encoder, and a uni-directional LSTM for my decoder.

I am using a 70-20-10 train-test-validation split. This seems to be a good choice because a big part of the data is used for training (87190 samples), but I still have a good amount of data for validation (12456 samples) and testing (24912 samples).

I have tried tuning some hyperparameters to improve the accuracy of my model. Specifically, I used different combinations for the batch size and the number of nodes in the hidden layer. Increasing the number of nodes generally increases model performance, as does decreasing the batch size. The drawback to doing so is that training time increases.

The optimizer that I currently use is ADAM, and the loss function is categorical cross-entropy. I have tried using another optimizer, RMSprop, but it did not give better results than ADAM. Overall, I have not looked into optimizers and loss functions in depth, but that is something that I plan on doing.

The validation set was used to determine when to stop training the model. If the validation loss did not improve for a specific number of consecutive trials (determined by the "patience" hyperparameter, which I also tuned), then training stopped.

I tested different models by using the procedure described above, getting the predicted sequence of phoneme for each word in the test set. I then computed two metrics, the accuracy (whether or not the predicted pronunciation was exactly the expected one) and the BLEU score.

4 Preliminary results

Here are some examples of hyperparameter combinations that I tried out and the performance of the corresponding model. The best set of hyperparameters is in bold.

Encoder	Decoder	Nodes	Patience	Optimizer	Batch size	Accuracy	BLEU score
BLSTM	LSTM	256	1	Adam	32	0.5756	0.7031
BLSTM	LSTM	256	1	Adam	128	0.5794	0.7059
BLSTM	LSTM	256	1	Adam	256	0.5583	0.6859
BLSTM	LSTM	256	1	RMSprop	64	0.5782	0.7042
BLSTM	LSTM	256	1	RMSprop	128	0.3197	0.4886
BLSTM	LSTM	256	1	RMSprop	256	0.5788	0.7023
BLSTM	LSTM	256	2	Adam	32	0.5865	0.7122
BLSTM	LSTM	256	2	Adam	64	0.5987	0.7174
BLSTM	LSTM	256	2	Adam	128	0.5913	0.7120
BLSTM	LSTM	256	2	Adam	256	0.5718	0.6978
BLSTM	LSTM	256	3	Adam	32	0.6063	0.7251
BLSTM	LSTM	256	3	Adam	64	0.5930	0.7142
BLSTM	LSTM	256	3	Adam	128	0.5897	0.7112
BLSTM	LSTM	256	3	Adam	256	0.5709	0.6953
BLSTM	LSTM	512	1	Adam	32	0.6018	0.7243
BLSTM	LSTM	512	1	Adam	64	0.5816	0.7079
BLSTM	LSTM	512	1	Adam	128	0.5996	0.7225
BLSTM	LSTM	512	1	Adam	256	0.5858	0.7119
BLSTM	**LSTM**	**512**	**2**	**Adam**	**32**	**0.6195**	**0.7359**
BLSTM	LSTM	512	2	Adam	64	0.6037	0.7249
BLSTM	LSTM	512	2	Adam	128	0.5966	0.7190
BLSTM	LSTM	512	2	Adam	256	0.6028	0.7218
BLSTM	LSTM	512	3	Adam	32	0.6182	0.7353
BLSTM	LSTM	512	3	Adam	64	0.5993	0.7206
BLSTM	LSTM	512	3	Adam	128	0.5996	0.7205
BLSTM	LSTM	512	3	Adam	256	0.5981	0.7203

The accuracy and BLEU score of the best model are 0.62 and 0.74 respectively. I have tested this model on the combined training and validation sets also, and the accuracy and BLEU score are 0.76 and 0.83. Therefore, I think my model might be underfitting, because its performance on the training set is a bit low, and this is a sign that the model is not able to capture all of the complexity of the data. The test accuracy and BLEU score are lower than the training ones, but it still seems like the model is able to generalize to new examples, since an accuracy of 62% means that 62% of the predictions have no mistakes at all, which is I think acceptable, especially when considering the total number of possible combinations.

Here is a plot showing the loss and the validation loss of the best model across the different training epochs. Here, the "patience" parameter was set to 2, so the training stopped after the validation loss stopped improving for 2 consecutive epochs. The circle on the plot indicates the lowest validation loss.

We can see here that the loss on the training set was lower than the loss on the validation set when the model stopped training. I think this might be why the accuracy and BLEU score for the training set was better than that for the test set.

5 Next steps

My model is working but, as previously stated, I think it might be underfitting. This means that I should increase its complexity. A way to do that would be to increase the number of nodes in the hidden layer. Other techniques that I think would probably increase my model's performance are attention and beam search. Using an attention mechanism will allow the decoder to "attend" to a specific part of the input sequence, instead of only relying on the encoder's states [3]. Beam search will allow the decoder to keep track of multiple growing predictions at the same time, with the final output being the prediction with the highest overall probability [3].

References

[1] Carnegie Mellon University. *The CMU Pronouncing Dictionary*. `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`.

[2] Francois Chollet. *A ten-minute introduction to sequence-to-sequence learning in Keras*. `https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html`.

[3] Guillaume Genthial. *Seq2Seq with Attention and Beam Search*. `https://guillaumegenthial.github.io/sequence-to-sequence.html`.