General

Knowing that I was dealing with transcriptions of audio recordings, I set off on this task having decided to use an RNN to leverage phonological dependence within words. All models were trained through Tensorflow's Sequential module.

Baseline

To get an idea of baseline performance, for my first training run, my only goal was to get the model running. In this run, I did basic preprocessing where I one-hot encoded all that targets and changed feneme names to integers in the training data. I then padded all the training data to be the length of the largest sample (400); I did not use endpoint data for this trial in an attempt to see if the model could notice changes in intensity (i.e. transitions from 'whitespace' fenemes, to utterance, back to silence). The first structure consisted of an input layer and an LSTM layer with 100 units. Performance was very poor with a training accuracy of .15 and a validation of about .05. To further use phonology, I added a bidirectional connection to the LSTM layer to see how a feneme predicts those that come before. While I saw significant improvement, the model severely overfits the training data with a nearly perfect training accuracy and a validation accuracy with ~.20.

Improvements

My next test included the training endpoint information. I first used the true endpoints to train the same model (Bidirectional with some additional modifications) with a very considerable improvement in accuracy (.95 train, .80 validation). The final structure was a 2-layer bidirectional LSTM with 64 neurons each, a dropout layer set to .4, and a dense layer to project to output dimensions with a 'softmax' activation for classification. I was impressed by these results; however, I knew I needed a reliable way to trim test samples in a similar way. I first tried trimming at the lowest start point and the maximum end point to ensure I got the full utterance of each word. My next strategy was to trim at the average start and end points. Both of these resulted in a train accuracy of around .80 and a validation accuracy around .30.

While looking at the train features file, I realized that finding the endpoints can almost be done visually by imagining the feature vector as a low-resolution image, so this led me to try training a CNN to find the start and end point as a regression problem. I trained a separate model for the start and end to reduce the solution space. After experimentation, the best structure was a 2-layer with 64 units CNN with a max pooling layer, again, ending in a dense layer to match the output shape. For a loss function, I went with mean absolute error to help with outliers and some values that may look extreme due to data sparsity. In the end, this model had a validation MSE ranging from 33 to 70 for finding the start points and up to 90 for the end points. Even though this appears high, when comparing the predicted list of endpoints with the true list, I believed the models would be serviceable.

As a test, I trained the LSTM model using the clips trimmed by the endpoint models, I got a train accuracy of .99 and a validation accuracy of ~.60. I attributed some of the discrepancy between accuracy and validation scores to overfitting so at this point I started experimenting with dropout, testing values in .05 increments from .3 to .5 where I got a .99 and .75 train/validation for .3, .95 and .77 for .4, and .98 and .65 with .5. While .4 produced the best results in this experiment, I noticed accuracy, while training with the same hyperparameters, could vary as much as .10 (see discussion). The final decision was whether to train the model with the predicted endpoints so it could be trained on data it would actually come across (ie noisy from the CNN) or train using the true endpoints so the model can learn strong characteristics of the word. The final model used the true endpoints, but I would need to experiment with more data to truly test the effects.

Out of curiosity, I trained a CNN instead of using the LSTM layer. This model ended training with a .99 training accuracy and .50 validation accuracy. Although performance was lower, this model trained much faster with similar results which leads me to believe a CNN can be a viable option for this task with more data and smarter preprocessing techniques.

Discussion

The ASR challenge was a very rewarding experience; however, as an experiment, my project is still in its rudimentary phases. Given more time and resources I would have liked to run several more tests. First, I would have studied further into the data distributions such as the endpoint locations and also feature frequency to see if any feneme classes can be weighted differently. A deeper understanding of the data would have helped in creating clever preprocessing techniques. I would have also liked to do a more thorough hyperparameter testing including layer numbers and size as well as different models entirely, such as vectorizing each feneme, averaging over the utterance and training an SVM. As is usually the case, I believe these models would have greatly benefitted from more training data. Without access to that, I would like to see how cross validation can affect performance. I would have also liked to look into the data transcription process like how the fenemes are encoded.

As a preliminary test, the results of this project are fascinating and I would be interested in seeing how the above points could improve the model.

Note on usage:

Must be run in same directory with all files.

For run_with_files.sh, expects all data (train features, endpoints, transcriptions, feneme.txt, test cases) must be in a subdirectory ‘data’

run_all.sh:

- ./run_all.sh <train_features> <train_targets> <endpoints> <fenemes> <test_features> <output_file>
- Runs the whole pipeline (ie training end point finder, training ASR RNN, and making predictions on test set.
- Allows for different data sets and output file location

run_with_files.sh:

- ./run_with_files.sh
- Runs full pipeline with hard-coded parameters.
- All data files must be in data/ subdirectory
- Outputs to output.txt in same directory

endpoints.py:

- python3 endpoints.py <train_features> <endpoints> <fenemes>
- Given feneme encoded recordings and associated endpoints, predicts the start point and end point of new recordings.

asr.py:

- python3 asr.py <train_features> <train_transcript> <fenemes> <endpoints>
- Trains model to predict word utterances based on feneme encoded recordings

predict.py:

- python3 predict.py <test_features> <output_file>
- Makes word predictions best on feneme encoded recordings