Pronounce this!

Tarun Sunkaraneni
CS6956: Deep learning for NLP
April 28, 2019

Abstract

Deep learning in NLP has made great advances in many domains such as NER, Machine translation and language inference to name three. This project explores the ability of these RNN sequence models to understand pronunciation of words. In the domain of NLP tasks, this should be one of the easier tasks to handle, as there is only less than 30 characters which constitute every word, and there are relatively few ways to pronounce all the phonemes in English. The dataset this project uses is the CMU pronouncing dictionary, which has 134,000 words and their ARPAbet pronunciations (87 symbols). The maximum word length is 19 letters long, and the longest phoneme sequence is 18 symbols long. Therefore, there is very little compute requirements for this task. Although this task is very important for tools such as voice assistants, this task has probably already been "solved" for English by 2019. Nevertheless, it is a very good project for understanding proof of concepts covered for seq2seq models.

Ideas/ Process

Some important ideas explored in this project include *learned* character and phoneme encodings through the process of training, the encoder-decoder pattern and how to feed data the pattern, and recurrent dropouts. Different types of evaluation metrics such as BLEU score and syllable accuracy were also necessary for a fair judgement. In order to combat overfitting on the model when word embeddings were used, recurrent dropout was necessary.

In addition, heavy knowledge of the computation graph was needed to setup different decoder architectures for training and testing phases. The training phase required a decoder which would be able to take gold pronunciations as inputs and try to predict the next pronunciation, while the testing phase the same decoder to feed its own outputs as input for the next timestamp, generating outputs until it is outputs the terminator symbol or reaches the maximum word length.

An embedding table/layer greatly helped out learning character level and phoneme level representations better because some characters are obviously closer to some than others, which is something that can be learned by an embedding instead of just relying on the hidden states to handle the one hot vectors.

No pretrained embeddings were used for this project so that there would be fair evaluation on how much the model could improve by learning embeddings. Even so, normal character level embeddings are most likely for language modeling. Although language modeling embeddings will be correlated to pronunciation, they will not be really helpful for the task at hand.

Results

Evaluation was done by training the model on 100,000 words and their pronunciations and evaluating metrics on the remaining 34,000 words. After preprocessing all the test dataset and running through the algorithm, I evaluate on the following metrics: syllable count, overall accuracy, and Bleu score.

Syllable count is found by extracting the number of phonemes in the output with a stress marker. This makes it a very easy metric to attain, and one which should be a high percentage because it does not consider whether the phoneme with the stress marker is the gold phoneme after all. When ran on the baseline model (1-hot encoded inputs/outputs), I got an accuracy of 91.0%. After adding an embedding layer, the accuracy went up to 95.7%. Significant advancement. With the addition of both recurrent and time distributed fully connected dropout, I get 96.2% accuracy. Not a really significant gain because it is easily learned anyways, so overfitting is not a big problem.

Overall accuracy and Bleu score are a measurement of how the model performs with the task at hand. Bleu score is more forgiving to offsets in indices than accuracy, so it is expected to have a higher score than accuracy. With the baseline model, I got a Bleu score of 0.66 and accuracy of 0.52, with the embeddings I got Bleu score of 0.69 and accuracy of 0.55, finally with the embedding model and dropout, I got Bleu score of 0.74 and accuracy of 0.64.

The Embedding layer doubles the number of trainable parameters from 600k to 1.1 million, so adding dropout is crucial to prevent overfitting. And out accuracy and Bleu advances (~10% accuracy and 0.05 Bleu) definitely support that belief.

Future studies

The next direction to go with this project is make the RNN a BiLSTM, as word pronunciation is determined by the next character as much as the previous character. Indeed, because of the Encoder phase, the whole representation *does* get captured before we start decoding, so it is not as important as it would be in a task such as a language model. However, this would still help capture relationships between immediate character neighbors.

The next addition would be to add beamsearch to make a more robust classifier. This would help making robust choices without being greedy. It is not as much of a concern dealing with intractability because our words are not too large, but we would nevertheless need to keep a restricted k amount of choices before making out prediction.

The final addition would be to add attention. Since there’s a definite correlation between the characters and the phonemes, attention would be able to pick up and remember character sequences better when the input word gets is longer. This would be better at capturing relationships than just the “handoff” our encoder-decoder goes through.

References/Further Readings

- Predicting Pronunciations with Syllabification and Stress with Recurrent Neural Networks - Daan Esch
- Text-to-Phoneme mapping using neural Networks – Eniko Bilicu
- English Phoneme predictions using RNN – Ryan Epp
- Joint-Sequence Models for Grapheme-to-Phoneme Conversion- Max Bisani