Background

Currently, the giant tech companies are fighting to build the best speech-based assistant. Nevertheless, Siri, Alexa, Cortana and Google now are good at recognizing words but they still make many mistakes when there is a background noise. In addition, no single assistant is good enough at understanding complex human intonations such as “Is what is an easy task to code for?” or “What is an easy task to code for?”. However, by using spectral features, we can differentiate between words. To build a more robust model which can be used in the outdoors, we augmented the audio data using various background noises in order to make our training data set more generalized.

Data and preprocessing

We preprocess the data from its raw 16kHz form into a 2D MFCC matrix. The steps of the transformation size:
1. Take the Fourier transform of a windowed excerpt of a signal. We used 80ms windows (one time step every 15ms). We removed the first and last time step.
2. Map the powers of the spectrum obtained above onto the real scale, using triangular overlapping windows.
3. Take the log of the powers at each of the real frequencies. Based on our literature review, we found that the number of bins to use for the MFCC fingerprint is between 18 and 40. We used 40 bins.
4. Take the discrete cosine transform of the log of real powers, as if it were a signal.

The MFCCs are the amplitudes of the resulting spectrum. Overall, our MFCC data representation is a 2D matrix of 40x88, for 40 frequencies and 88 time steps.

Baseline model

1) Model: One Fully connected layer
   Input: 100
   FC: 40
   Softmax
   Accuracy: 48.2%
   Parameters: 1590

2) Model: BI-GRU
   Input: 100
   GRU: 100
   Softmax
   Accuracy: 73.8%
   Parameters: 1590

3) Model: 2-layer CNN
   Input: 100
   Conv1: 64, 64
   Pool1: 64, 64
   Conv2: 64, 64
   Pool2: 64, 64
   FC: 40
   Softmax
   Accuracy: 97.5%
   Parameters: 5120

4) Model: 3-layer CNN
   Input: 100
   Conv1: 64, 64
   Pool1: 64, 64
   Conv2: 64, 64
   Pool2: 64, 64
   FC: 40
   Softmax
   Accuracy: 97.5%
   Parameters: 15360

5) Model: Single LSTM cell
   Hidden layers: 320
   Learning rate: 0.05 for 5000 iterations
   Accuracy: 93.6%

6) Model: Two LSTM cells
   Hidden layers: 2 x 320
   Learning rate: 0.05 for 5000 iterations
   Accuracy: 93.4%

7) Model: Bidirectional LSTM
   Hidden layers: 320
   Learning rate: 0.05 for 5000 iterations
   Accuracy: 93.4%

Advanced models

2) Model: 2-layer CNN (input -> conv2D -> RELU -> maxpool -> conv2D -> RELU -> fully connected -> softmax)
   Hyperparameters: learning rate, filter size, number of filters per layer
   Accuracy: 96.5%

3) Model: 3-layer CNN (input -> conv2D -> BatchNorm -> RELU -> maxpool -> conv2D -> BatchNorm -> RELU -> conv2D -> BatchNorm -> fully connected -> softmax)
   Hyperparameters: learning rate, filter size, number of filters per layer
   Accuracy: 97.5%

4) Model: 4-layer CNN (input -> conv2D -> BatchNorm -> RELU -> maxpool -> conv2D -> BatchNorm -> RELU -> conv2D -> BatchNorm -> fully connected -> softmax)
   Hyperparameters: learning rate, filter size, number of filters per layer
   Accuracy: 97.5%

Results / Hyperparameter Tuning

Hyperparameters in CNNs:
- The hyperparameters are the learning rate and filter size. Iterating over different learning rates shows that the best learning rate and an initial convolutional layer filter size is the best filter size. Other performance metrics tested were F1 score and accuracy. Having trained over 1000 training steps. This was the most learning secure.

Figure 5: Accuracy versus iteration for the top CNN architecture

Figure 6: Loss versus iteration for the top CNN architecture

Future Directions

As seen above our best CNN model outperformed our optimized bi-directional LSTM network. The best CNN network achieved 93.7% testing accuracy while the best RNN model achieved 88.7% testing accuracy. This may be the case because of how the error signal flows through the RNN network. In the RNN the error signal might have to travel up to 88 time steps to modify the weights of a sound input based on another future input. Given more time we would have experimented with different time steps. Specifically, it may be the case that using attention in our RNN model could help speed up the learning process since the relationship between audio features of different times would be better captured by the gating weights of an attention model.