Speech Recognition: from speech to text

Ben Limonchik , Rysen Otomo {benlimon, rysotomo}@stanford.edu

March 2018

Abstract

More and more appliances in our lives are getting connected to the internet. Thermostats, garbage robots, heater and air-conditioning systems are some of these new internet connected devices. The more appliances we have the more interfaces we need to work with. The promise of speech recognition is to reduce the dependency on interfaces and instead control all electronics via our voices. In this project we attempted to dive deeper into the process of turning human speech (i.e. audio files) into text. We initially preprocessed the audio data to extract features of sound and then built various neural networks to classify the audio files' finger prints. We experimented with both Recurrent Neural Network architectures and Convolutional Neural Networks. Our best performing network was a three layered CNN achieving 93.7% Testing accuracy over our data.

1 Introduction

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 mistakes when there is background noise. In addition, no single assistant is good enough at understanding complex human intents such as "Siri what is an easy dish to cook for 6 people". In this project we explored speech recognition through classifying audio files into written words. In order to build a more robust model which can be used in the outdoors, we augmentated the audio data using various background noises in order to make our training data set more generalized.
All of our network architectures had a similar input and output. Initially we preprocessed each raw audio file into a 2D Mel-Frequency Cepstrum Coefficients matrix (MFCC) (See dataset for more details) and use this 2D matrix finger print of the audio as input to our networks. The output of our networks is a Softmax layer that classifies the input into one of 12 classes:(1) One of ten possible word or (2) Background noise or (3) Unknown word.

2 Related work

Google has performed a similar task in its *Convolutional Neural Networks for Small-footprint Keyword Spotting* paper. In this paper, the authors describe the benefit of using a CNN for recognition of small-footprint keywords over a deep neural network (DNN). In this paper, it states that a CNN can achieve 27%-44% more accuracy while reducing the number of parameters. Interestingly, the CNN performed better for noisy and non-noisy audio samples. This work has informed our choice to look at CNN networks as well and not only on RNN networks that capture the temporal nature of audio data.

3 Dataset and Features

For our project we are using 31,682 one second long audio files collected by Google. These one second audio clips include either silence, an unknown word, or one of the ten words: "yes", "no", "up", "down", "left", "right", "on", "off", "stop", or "go". For audio files with a word, the word is repeated only once in the file. The number of the examples we had for each class is fairly even:

yes	no	up	down	left	right	on	off	stop	go	silence	unknown
2377	2375	2375	2359	2353	2367	2367	2357	2380	2372	4000	4000

Notice that we artificially generated more 'silence' and 'unknown' data examples since those tend to be more common in the real world.
Our task was, given an audio file, classify which word/silence/unknown word was recorded in this file. The data was collected from a wide variety of people which helps to ensure that our model will generalize better to unheard voices.
The original words data had very little background noise and as a result we added random background noise to each word file in order to make them more realistic. We made sure not to use the same random noise in the background so the network would not learn to recognize only a few types of background noise. Furthermore, the unknown word was randomly sampled from a collection of 8 words that are of no interest (i.e. are not part of the ten words we do try to correctly classify).
Finally the dataset was split into a training , a validation and a test set according to the following distribution 80%, 10%, 10% respectively.

3.1 Preprocessing speech data

Audio data is typically represented by the frequency values across different vocal bands over a period of time. In our data the sampling rate is 16kHz, this means that we have 16,000 amplitude values for every second of data. In our initial experiment we ran this 16,000x1 vector through a fully connected layer which lead to very poor accuracy. This vector representation doesn't capture the features of the sound file and its temporal structure. Therefore, for our later experiments we pre-processed the data using Mel-Frequency Cepstral Coefficients technique (MFCC). Using MFCC allows one to convert the raw 16,000 valued audio file into a 2D matrix that better represents the features of the sound file. The steps to convert the raw audio file are (see MFCC flow summary below for more details):
1) Taking the Fourier transform of (a windowed excerpt of) a signal: we used 98 time windows (one time step for every 160 amplitude readings and we remove the first and last time step)
2) Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows.
3) Take the logs of the powers at each of the mel frequencies. Based on our literary review we saw that the recommended number of bins to use for the MFCC fingerprint is between 26 and 40 for an audio file sampled at 16kHz. In our project we used 40 bins.
4) Take the discrete cosine transform of the list of mel log powers, as if it were a signal.
5) The MFCCs are the amplitudes of the resulting spectrum.
Overall our MFCC data representation is a 2D matrix of size 40x98, for 40 frequencies and 98 time-steps. See figure 3 for a visualization of the MFCC. The white spots are areas of high amplitude for a given frequency.

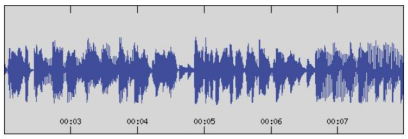

Figure 1: The waveform of an audio file

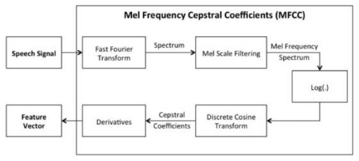

Figure 2: summary of raw sound to MFCC conversion

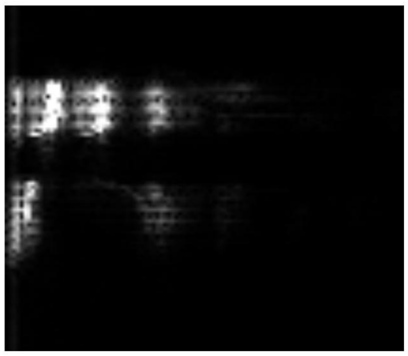
Figure 3: An 40x98 MFCC for the word 'happy'

4 Methods

4.1 Baseline Model (Single Fully Connected Layer)

Our baseline model implements a fairly naive approach. In order to determine a lower bound in accuracy, a single fully connected neural network was implemented where the weights were optimized via stochastic gradient descent. Training also incorporated dropout to help speed up iterations as well as add some regularizing effects. The output is then a softmax classifier with 10 outputs.
As expected, this naive approach does not perform particularly well. The average testing accuracy was found to be 48%.

4.2 CNN via 2D Convolutions

As mentioned in the previous section, the audio data was transformed into a spectrogram image as illustrated in Figure 3. Because the data is now represented as a 2D image, a CNN architecture with 2D filters was a natural model choice. Initially we used CNN architecture with 2 convolutional layers with dropout connected to a fully connected layer yielding 90% testing accuracy. Building on this, batch normalization and a third convolutional layer was added. In addition, gradient descent was originally used to optimize our network, but the 3-layer network described above instead used the Adam optimizer. It was found that the Adam optimizer and batch normalization helped speed up learning. In the 2 convolutional layer model, these changes helped decrease the number of training steps necessary to converge to 85% by a factor of 4. This makes sense since the Adam optimizer enjoys the benefits of both RMS prop and momentum such that the learning model can reach the global minimum faster and take big learning steps in large plateaus. On the other hand, batch normalization helps the network adjust to noisy inputs and controls the magnitude of activations between different layers. This helps learning because it reduces the network's sensitivity to noisy input.
Finally, adding an extra layer helped learning accuracy improve to 94% (4% improvement). The tuning procedure and hyperparameter search will be described in the next section. The final CNN had the following architecture:
input, convolution(64 filters)+bias, batch norm, RELU, maxpool, convolution(128 filters)+bias, batch norm, RELU, convolution(256 filters)+bias, batch norm, RELU, fully connected layer, softmax.

4.3 RNNs

Due to the temporal representation of speech data many state of the art speech recognition models are base don RNNs. At each time step a snippet of the audio data is fed to an RNN cell (e.g GRU or LSTM) and when the network has been fed with all the audio data, it uses an output layer to classify the audio file to a word. In our project we feed the RNN the 40 frequency values of a given time-step and unfold the RNN 98 times (for a total of 40*98=3920 features) as discussed previously. A regular RNN suffers from the problem of 'remembering' features from earlier time-steps. Since we unfold the RNN 98 times this can be a significant problem. Therefore in our project we used Long Short Term Memory networks that using the cell state, can let features from earlier time-steps propagate easily to later time-steps.
We also experimented with bi-directional RNN networks and found that those outperform regular forward only LSTM networks.

5 Experiments/Results/Discussion

5.1 CNN

In order to get the best accuracy from the 3 convolutional layer CNN, the hyperparameters that we tuned were the learning rate and filter sizes. The filter size of the first layer was chosen while the filters of the next convolutional layer would decrease the width by a factor of 2 and decrease the height by a factor of 2. The following table shows the dev/test accuracy given by the corresponding learning rate and filter sizes over the first 2000 steps of batched learning.

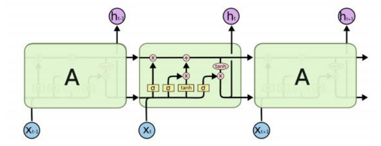

Figure 4: Our LSTM network unfolds 98 times outputing 12 classes at each time-step

$$
\begin{aligned}
i &= \sigma(x_t U^i + s_{t-1} W^i) \\
f &= \sigma(x_t U^f + s_{t-1} W^f) \\
o &= \sigma(x_t U^o + s_{t-1} W^o) \\
g &= \ tanh(x_t U^g + s_{t-1} W^g) \\
c_t &= c_{t-1} \circ f + g \circ i \\
s_t &= \tanh(c_t) \circ o
\end{aligned}
$$

Figure 5: equations of an LSTM cell, the extenet to which the past cell state is remembered is controlled via the forget and input gate and the candidate cell

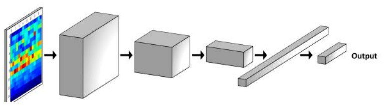

Figure 6: our three CONV-layered network, takes in a 40x98x1 image and outputs 12 classes

Hyperparameter Tuning			
	lr=0.01	lr=0.001	lr=0.0001
4x10 First Filter	77.7%/77.9%	80.3%/75.9%	78.6%/74.1%
8x20 First Filter	84.2%/84.2%	86.9%/86.6%	84.1%/83.5%
12x30 First Filter	79.2%/79.6%	85.1%/85.0%	85.3%/85.5%

The weights change the most over the first 2000 iterations. Then, it can be seen from the table that the best learning rate is 0.001 and the best filter size results from using an 8x20 filter on the first convolutional layer. No regularization was used as the validation accuracy and test accuracy were fairly close.
This CNN was trained over 15,000 steps where the first 10,000 steps were done with a learning rate of 0.001 and the last 5,000 steps were done with a learning rate of 0.0001. The validation accuracy was 94.0% and the test accuracy was 93.7%. The class-wise accuracy is shown in the tables below:
Dev Set Accuracy

silence	unknown	yes	no	up	down	left	right	on	off	stop	go
94.2%	86.7%	97.3%	92.0%	90.4%	91.4%	92.2%	98.8%	95.7%	95.1%	97.9%	97.8%

Test Set Accuracy

silence	unknown	yes	no	up	down	left	right	on	off	stop	go
98.8%	85.9%	96.0%	93.4%	88.7%	91.3%	92.8%	97.1%	95.2%	96.8%	93.1%	96.4%

Overall, the results from the development set and test set were very similar. So, the CNN did not have a variance issue. However, human performance can correctly classify all examples. This means that we still have a bias issue. An implementation with more filters was explored, but there were variance issues with 3 and 4 convolutional layers. It was found that a 3 convolutional layer model was best for the bias/variance tradeoff.

5.2 RNNs

In terms of RNN architectures we used LSTM cells throughout the project due to their memory features. We experimented with changing the learning rates, the number of hidden units, the number of stacked LSTM units and varying between feed forward only LSTM networks to bi-directional LSTM networks. We found out the following:
1) Stacking LSTM cells: Making our networks deeper by stacking more LSTM cells on top of each other had no major affect on final accuracy besides the longer time it took to train the network. This was true for both forward only and bi-directional networks. We experimented with stacking one to three LSTM cells on

top of each other and since we found no difference we continued on using only one layer of LSTM cells.
2) Hidden units: The number of hidden units used had a significant affect on the final accuracy of the model. We tested the basic feed forward LSTM network with 50, 100 and 200 hidden units and achieved the following dev/testing accuracies:

hidden units	50	100	200
dev/test accuracy	80.4%/80.2%	85.1%/83.7%	84.8%/82.9%

As it appears there was no added value to increase the number of hidden units beyond 100 units so we used this number moving forward.
3) Switching to Bi-directional networks: Instead of using only a feed forward LSTM cell we used a bi-directional LSTM structure. This meant that we used two LSTM cells one the unfolded backward and the other forward. Both cells' outputs are concatenated and then fed to a Fully connected layer. This change of architecture led to the most prominent increase in overall accuracy. The best test accuracy achieved with the bi-directional model was 88.7% about 5.0% higher then when using the forward only architecture. This maybe the case since predicting the nature of a work can be affected by both earlier sounds and later sounds. For example in the case of th word 'right' hearing the sound of 'ei-ght' can be helpful to a human to think of the word being said as 'right'. The same explanation is true for the forward direction if we only hear the letters 'Ri'. This may explain the improvement we noticed.
4) Learning rate: The learning rate for the LSTM models was baby-seated closely rather than predetermined. We noticed that starting with the optimal learning rate for the CNN model (i.e. 0.001) the Cross entropy error curve decreased linearly with a very small slope. Therefore we used a 0.05 learning rate which increased the speed of learning significantly. The learning rate was then reduced to 0.005 after 11,000 iterations when the accuracy started to plateau. This led to another 'jump' in accuracy (see graph plots below for optimal accuracy):

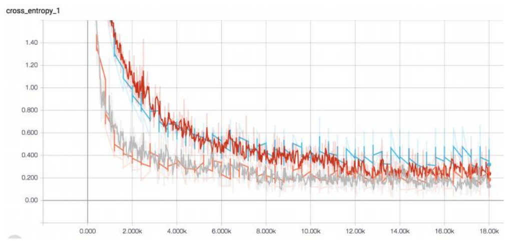

Figure 7: Y:crosss entropy error, X: iteration

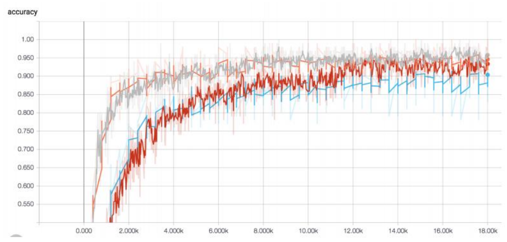

Figure 8: Y:overall accuracy, X: iteration

Figure 9: Orange curve: best CNN validation, Grey curve: best CNN training, Red curve: best RNN training, Blue curev: Best RNN validation

6 Conclusion/Future Work

As seen in figure 7 and 8 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. Furthermore, we see that the CNN network reaches higher accuracies much faster than the RNN network. This could be explained by the the way the error signal flows through these networks. In the RNN the error signal might have to travel up to 98 time-steps to modify the weights of a sound input based on another future input. Since this architecture may suffer from the vanishing gradient the weights will be updated only very little. On the other hand in the CNN case each CONV layer 'saw' all the 98 timesteps in the previous 'image' of the sound. Thus it could be the case that while all the timesteps are fed to the CNN at once, the network still managed to learn the temporal relationship between the different feature across different time periods.
Given more time we would have liked to experiment with additional network modifications. 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 alpha weights of an attention model.

7 Contributions

Both Ben and Rysen worked together to determine how to preprocessing the audio data. In addition, we both read and discussed the literary sources we found relevant for this project. Finally we came out with a plan of the different architectures we want to test and the hyperparameters we would want to tune. In terms of implementation, Ben focused on the RNN networks and Rysen focused on the CNN networks.

8 Link to Repo on Github

https://github.com/BenLimonchik/SpeechRecognition

References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[4] Sainath, T.N. & Parada, C. (2015) Convolutional Neural Networks for Small-footprint Keyword Spotting. *INTERSPEECH 2015*

[5] Sainath, T., Kingsbury, B., Saon, G., Soltau, H., Mohamed, A., Saon, G., and Ramabhadran, B. Deep Convolutional Networks for Large-Scale Speech Tasks, Elsevier Special Issue in Deep Learning, 2014.

[6] Zhang, Z., Sun, Z., Liu, J., Chen, J., Huo, Z., Zhang, X. (2016) Deep Recurrent Convolutional Neural Network: Improving Performance for Speech Recognition. [Online]. Available: http://arxiv.org/abs/1611.07174

[7] Ng, A., DeepSpeech, GitHub repository, https://github.com/mozilla/DeepSpeech

[8] Rubashkin, M. and Mollison, M. (2017) TensorFlow RNN Tutorial. [Online].
Available: https://svds.com/tensorflow-rnn-tutorial/

[9] Olah, C. (2015) Understanding LSTM Networks. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[10] Zhang, Y., Pezeshki, M., Brakel, P., Zhang, S., Laurent, C., Bengio, Y., Courville, A. (2017) Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks. [Online].
Available: https://arxiv.org/pdf/1701.02720.pdf

[11] Rao, K., Sak, H., Prabhavalkar, R. (2018) EXPLORING ARCHITECTURES, DATA AND UNITS FOR STREAMING END-TO-END SPEECH RECOGNITION WITH RNN-TRANSDUCER. [Online].
Available: https://arxiv.org/pdf/1801.00841.pdf