CIS 5603

HW #6

Zhengkun Ye

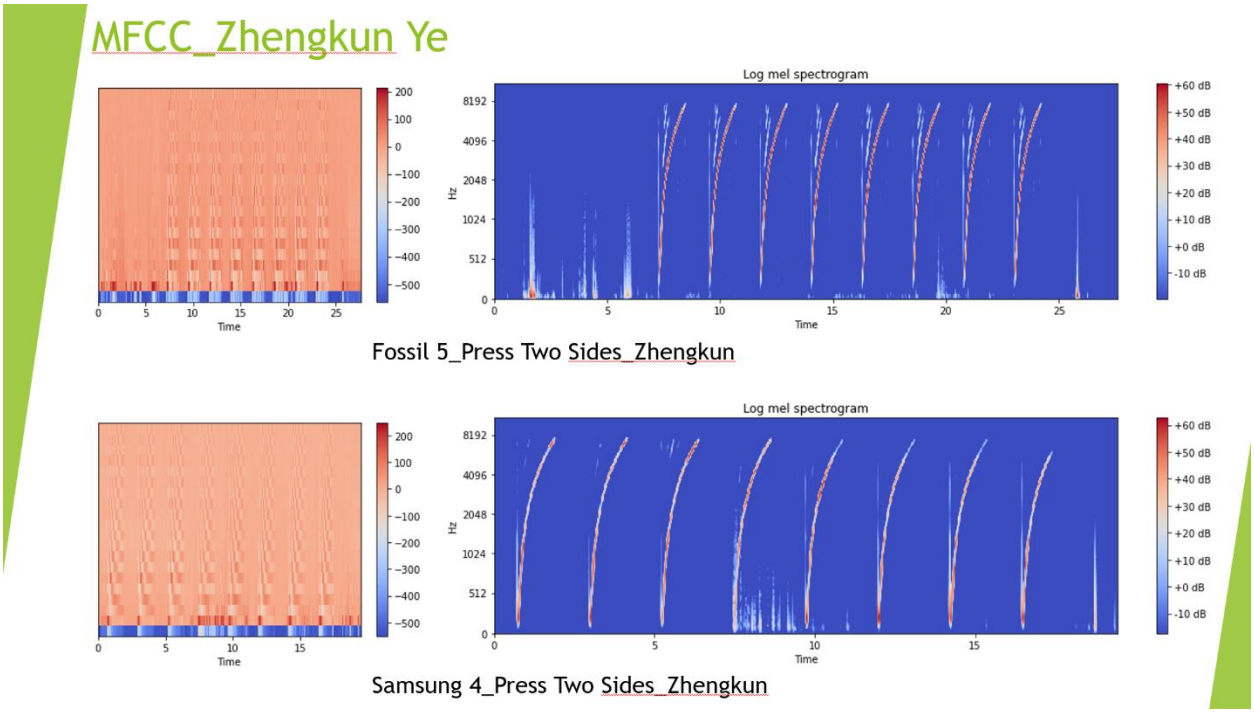
03/19/2022

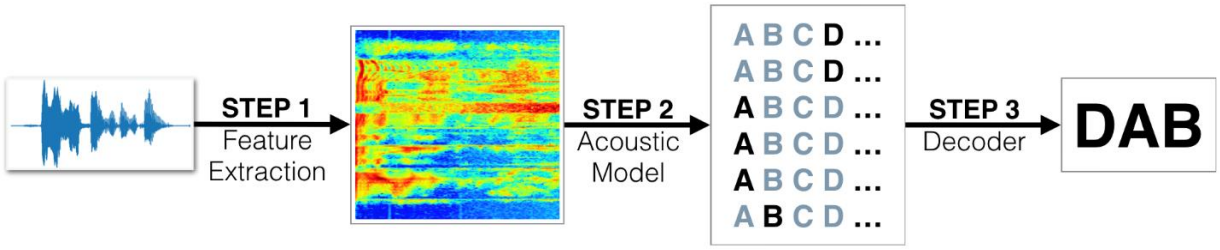
Speech Recognition using MFCCs Feature with Neural Networks

Introduction:

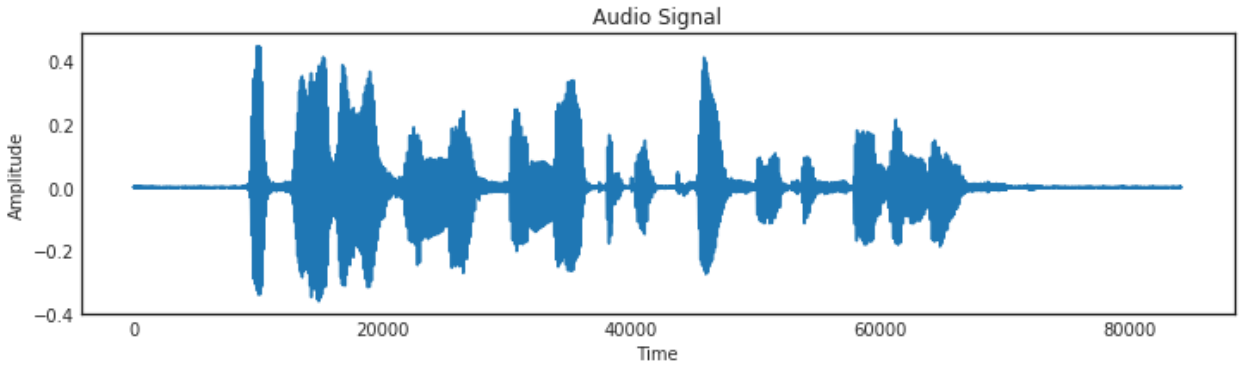
The Mel-frequency cepstral coefficient (MFCC) is widely used to represent the short-term power spectrum of acoustic or vibration signals and can represent the dynamic features of the signals with both linear and nonlinear properties. The MFCCs are state-of-the-art features for speaker identification and speech recognition. MFCC-based feature extraction is also related to my final project and my current research. The dataset LibriSpeech contains 1000 hours of speech derived from audiobooks. I only made use of a small subset in this project, since larger-scale data would take too much time to train.

Plots of MFCCS:



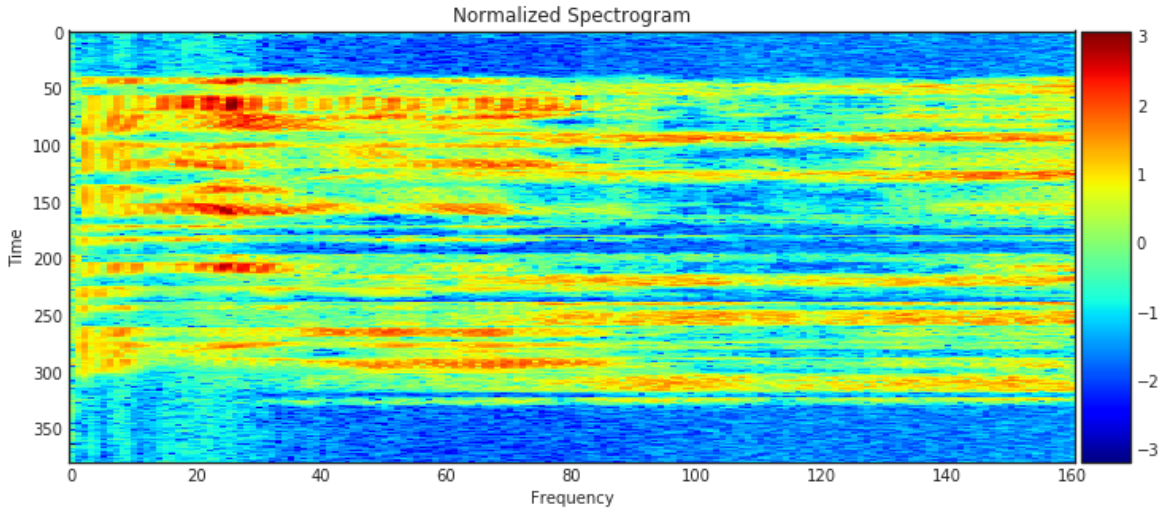


Step 1 is a pre-processing step that converts raw audio to one of two acoustic feature representations commonly used for speech recognition (user identification in our case). Step 2 is an acoustic model which accepts audio features as input and returns a probability distribution over all potential transcriptions. Step 3 in the pipeline takes the output from the acoustic model and returns a predicted transcription.

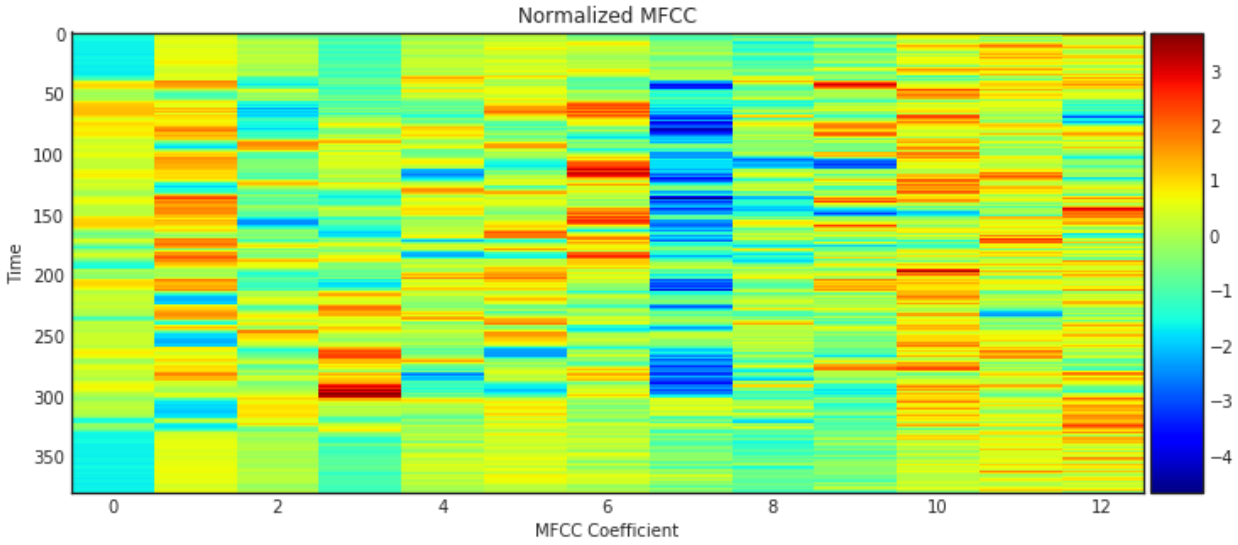


Feature Extraction:

The first audio feature representation is the spectrogram. A spectrogram is a representation of how the frequency content of a signal changes with time, we can also use as a visual way of representing the signal strength.

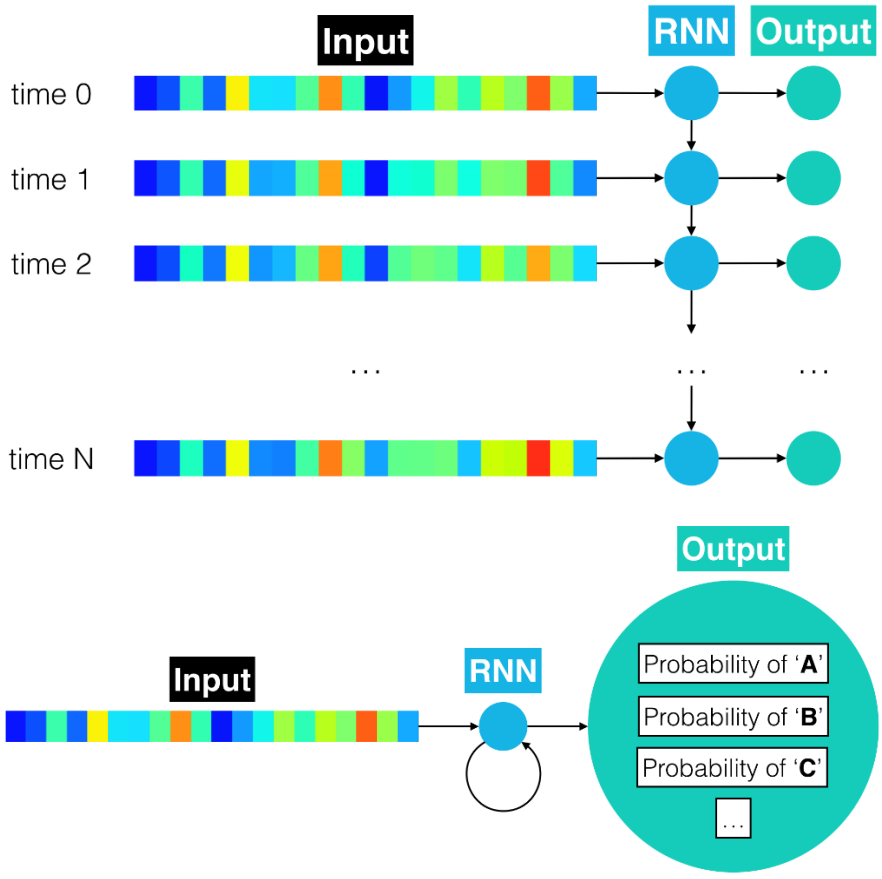


The second audio feature representation is MFCCs.

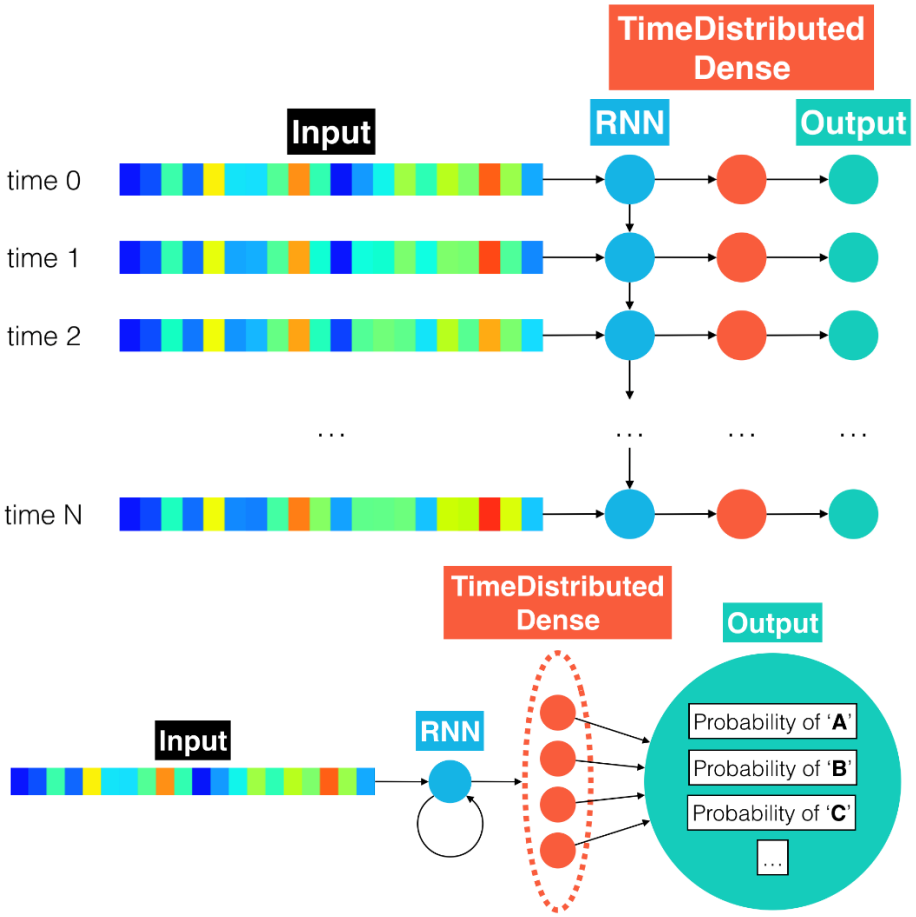


Neural Networks for Acoustic Modeling:

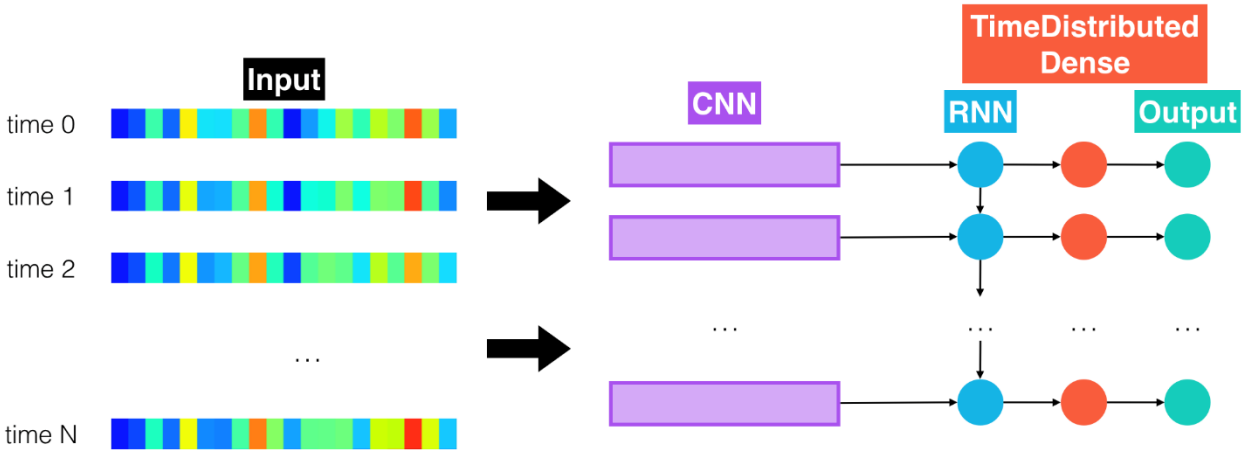
The first acoustic model is an RNN (recurrent neural network) for sequence classification. As shown in the figure below, the RNN will take the time sequence of audio features as input.



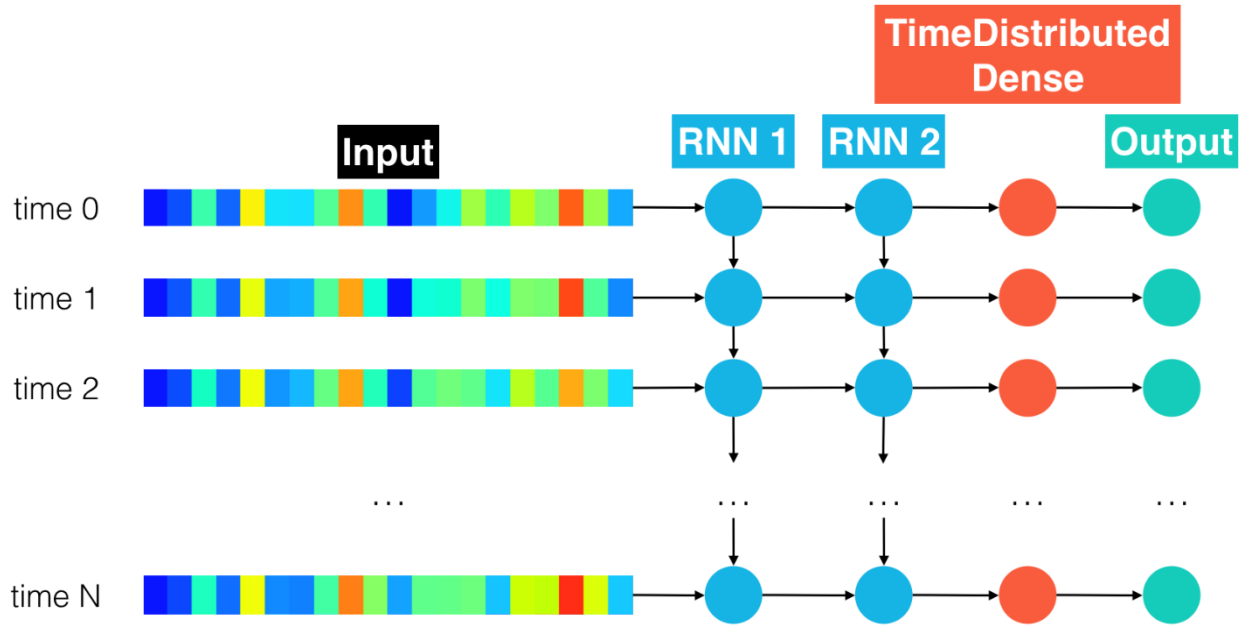
The second acoustic model is RNN + TimeDistributed Dense. The first layer of the neural network should still be an RNN that takes the time sequence of audio features as input. The TimeDistributed layer will be used to find more complex patterns in the dataset. The unrolled snapshot of the architecture is depicted below.



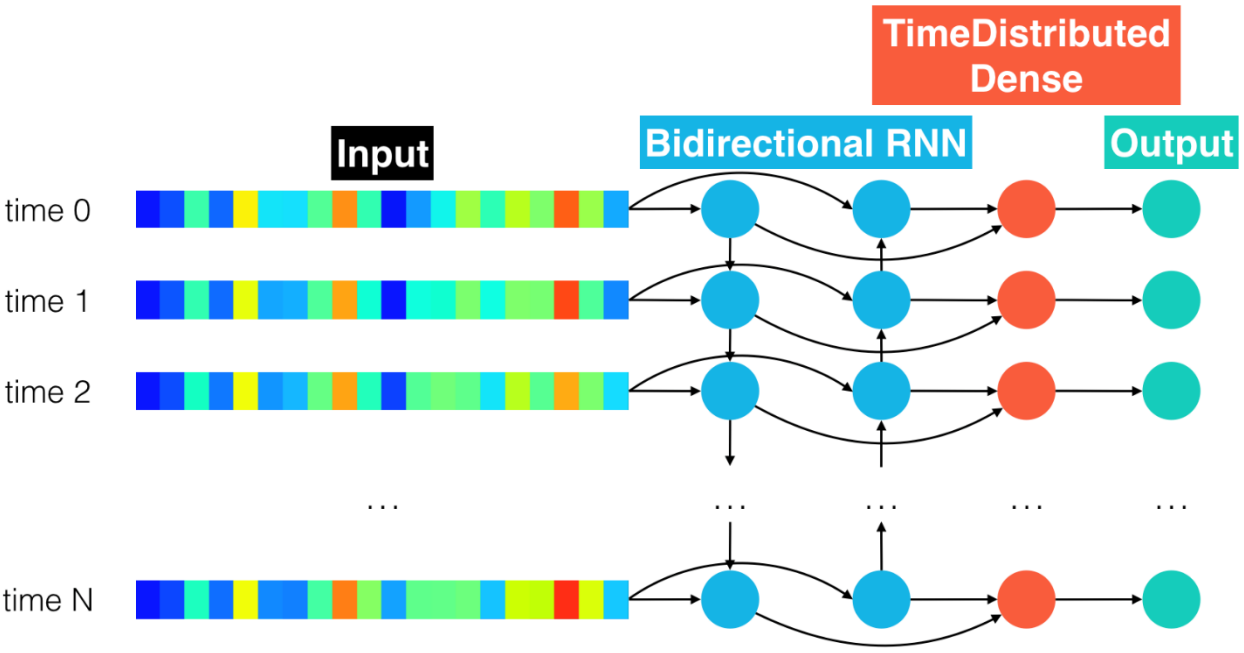
The third acoustic model is CNN + RNN + TimeDistributed Dense.



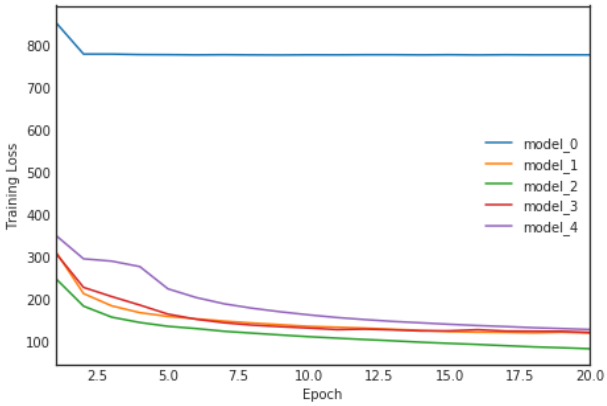
The fourth acoustic model is Deeper RNN + TimeDistributed Dense. In previous models, rnn_model, which makes use of a single recurrent layer. Now, specify an architecture in deep_rnn_model that utilizes a variable number recur_layers of recurrent layers. The figure below shows the architecture that should be returned if recur_layers=2. In the figure, the output sequence of the first recurrent layer is used as input for the next recurrent layer.

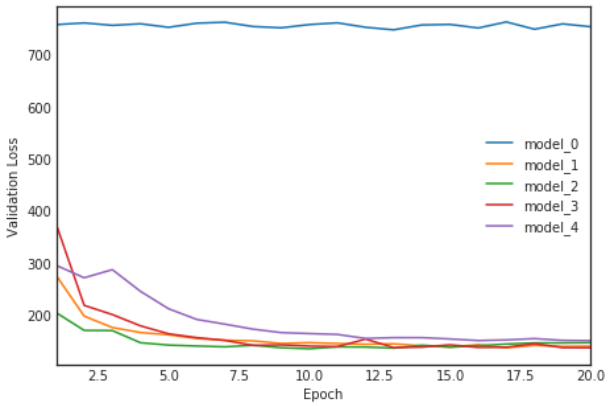


The fifth and the last model I use is Bidirectional RNN + TimeDistributed Dense. https://keras.io/api/layers/recurrent_layers/bidirectional/



Model Comparisons:





Discussion:

The first RNN model is named as model_0, and 1,2,3,4 are successively assigned to following models.

From the plots above we may see that it seems that in training model_2 has the smallest loss (with a training loss of 62.1754 after 20 epochs), but in the validation, it gets edged out by model_3. It seems to be pretty much tied between model_2, model_3, and model_4 in the validation. Model_0 is the simplest one, but it does not train very fast. It also has the worst scores and they do not seem to improve over time both in validation and testing. This would not be enough for our task. Model_1 improves the scores drastically (presumably due to the Time Distributed layer) and does not train much more slowly than model_0 (because of batch normalization). However, both training and validation scores get worse starting at about epoch 17, which is not ideal and requires further investigation. Model_2 trains well despite having a considerable number of parameters and the scores do not suffer from that same problem as model_1 whereas they start to go slightly higher in the end similarly which is probably due to some overfitting. Model_3 trains slowly and has a lot of parameters, although it does not show obvious signs of overfitting (The training loss after 20 epochs is 101.6875 and the validation loss is 116.8010), which may be an issue at some point from what I've read. Model_4 trains quickly and has a low level of complexity although it seems to get slightly worse validation scores when compared to model_2 and model_3. The final model outperforms all of these in the validation even though it gets edged out in training by model_2. It also trains quickly despite having the largest number of parameters of all the models.

In the case of models that use purely recurrent layers, the lambda function is the identity function, as the recurrent layers do not modify the (temporal) length of their input tensors. For the final model I would like to implement, by viewing previous models, we can see that the CNN model had good performance with low training times. Then I took the other best-performing model (the deep RNN one) and merged it with the CNN. Having two RNN layers improved my score when

compared to just one. I then increased the number of RNN layers to see whether the scores improved, but I got no/marginal benefits, and the training times increased drastically. So, I then settled for two RNN layers for decent training times and performance. I then added dropout between the convolutional and the first RNN layers, which also improved the scores. Finally, I added another dropout layer between the two RNN layers, which slightly enhanced the scores.

One standard way to improve the results of the decoder is to incorporate a language model. The limitations are whenever I need to train bigger, deeper models, the desired model will very likely take a long while to train. For instance, training an End-to-End Speech Recognition in English and Mandarin (https://arxiv.org/pdf/1512.02595v1.pdf) model would take few weeks on a single GPU. Although in my limited experience with research in the audio domain, I have gotten the most success in using MFCCs as the feature choice for supervised learning, I also did not benchmark my model on several publicly available test sets and compare the results. Nevertheless, this can be my plan and goal.

References:

1. Jian Liu, Yan Wang, Gorkem Kar, Yingying Chen, Jie Yang, and Marco Gruteser. 2015. **Snooping Keystrokes with mm-level Audio Ranging on a Single Phone**. In Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15). Association for Computing Machinery, New York, NY, USA, 142–154. DOI:https://doi.org/10.1145/2789168.2790122
2. Jian Liu, Chen Wang, Yingying Chen, and Nitesh Saxena. 2017. **VibWrite: Towards Finger-input Authentication on Ubiquitous Surfaces via Physical Vibration**. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). Association for Computing Machinery, New York, NY, USA, 73–87. DOI:https://doi.org/10.1145/3133956.3133964

Notebook:

The complete. ipynb notebook is available on my personal website: www.yezhengkun.com, there is a repository on the right top corner named as "SPEECH RECOGNIZER."