Convolutional Neural Networks for Speech Recognition

Berkan Hiziroglu, Adnan Alperen Demirci

Koç University

***Abstract*—The speech recognition models previously included Gaussian Mixture Models with a hidden Markov model (HMM). The Gaussian mixture models are replaced with the deep neural network structures. The convolutional neural networks (CNN) have improved significantly the performance of the speech recognition systems, as they model the complex correlations of the speech features. In this project, we trained a CNN - HMM hybrid model, and experimented on the TIMIT phone recognition dataset. In the report, we describe the structure and the use of CNN model applied to the speech features. Then, we describe the decoding and training process of the HMM.**

I. Introduction

THE purpose of this project is to create an automatic speech recognition framework using convolutional neural networks (CNN) and Hidden Markov models (HMM). This paper proposes a framework which transcripts a human speech into spoken words. Human speech signals can vary tremendously from other human speech signals due to the speaker's attributes, style, tone and other environmental noises. Automatic speech recognition framework maps a given input variable-length speech signal into variable-length sequences of phonemes. HMMs are used in this framework for modeling the temporal behavior of speech signals using state sequences. The conventional model before the use of deep neural networks is the Gaussian mixture models with a HMM. The Gaussian mixtures are used for modeling the speech features and calculating the posterior probabilites.For acoustic modeling, these models are replaced by the CNNs which were used as the likelihood or observation probability extractors. These observation probabilities are used as the emission probabilities in the HMM. For language modeling which is the other important part of the speech recognition, the n grams method is used which is also used in this project. But, this convention has changed with the use of recurrent neural networks which do not make the Markov assumption, so make better assumptions.

II. Convolutional Neural Networks and Their Use In ASR

A. Organizing the Input Data to CNN

To use Convolutional Neural Networks for speech processing, the input data should be organized as a *features* of the input speech. The feature extraction can be applied in variety of ways. The usage of MFCC causes problems because of the locality problems proposed in the original paper, therefore we have use *MFSC* features for the input data. The log-energies computations are made directly from the mel-frequency spectral coefficients(*MFSC*). The MFSC library for Julia or MFSC toolbox for MATLAB can be used for feature extraction. CNNs run a filter over the input data at both training and test time. This filter learns from various data samples. The use of features of the input data make the filter more generic to unseen data. *Weight sharing* refers to the decision to use the same weights at every position of the window. This procedure can be called local because the individual units of the input are calculated at a particular positioning of the filter. By convolving this filter throughout the features, the filter also learns the connection between a feature and it's following feature. This increases the recognition of the convolutional neural network. If the feature maps were to given one at a single time for each input speech, then the CNN would not have the information about the connection between frames. Therefore, the usage of frames increase the accuracy of the model significantly. We used 40 features for every input speech data.

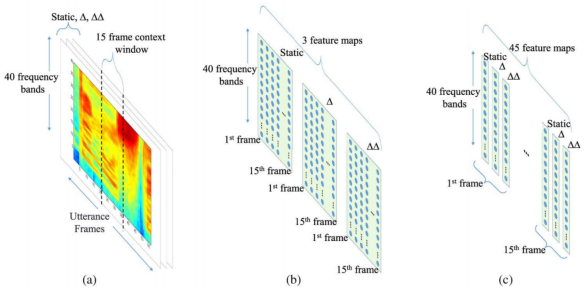

Fig. 1. CNN Layers

There are various ways of organizing the MFSC features while mapping the features for the inputs of convolutional neural network.

The input to the convolutional neural network is a 4D vector. The first two dimensions are 40 and 15. The number of features equal to 40 and at each iteration 15 frames are shown to convolutional neural network. The third dimension is the number of filters which is 150 or 80 (for *full weight sharing* and *least weight sharing* respectively). We used 150 filters because the model has *full weight sharing*. The last dimension is the batch size. We have used 500 as batch size. Normally, a larger batch size (i.e., 1000) would be more efficient in terms of time complexity. However, this model is trained on Amazon GPU instances and the relation between the frame number

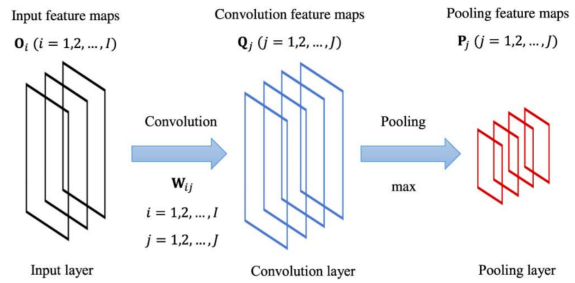

Fig. 2. The scheme of the convolutional network shown for an arbitrary input. The features maps are first convolved with the filter. The result of the convolution is transformed into likelihood probabilities using a max pooling layer. This procedure maps a frame of features into corresponding likelihood probabilities.

and batch size led the model to exceed the GPU memory. Therefore, the frame rate is set to 15 as the original paper and the batch size is set to 500. The first layer of the CNN is the convolution of the input features with the filters.

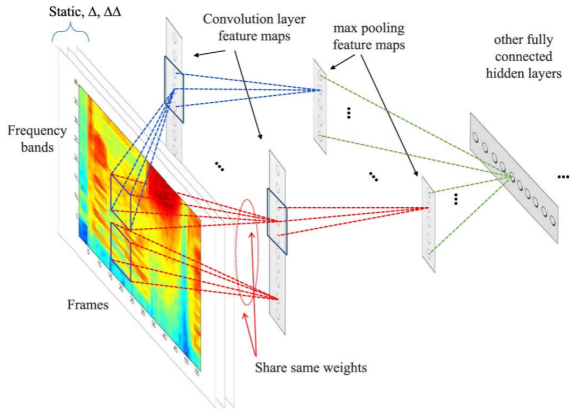

Fig. 3. CNN Layers with full weight sharing

Seen on Figure 3, the full weight sharing approach uses three layers of input. The first one is the input itself with feature maps. The second and the third layer is the second and third derivative of the input features respectively. This way, the information given to CNN is better. After the convolutional and pooling layers, CNN passes the weights from a fully-connected layer to yield likelihood probabilities for each frame. The last layer is a standard multi layer perceptron with 1000 neurons. It maps the weights to 39 possible phonemes with different probabilities. With the largest one indicating the decision for that frame. The pooling layer is useful in terms of computation time and also it finds a relation within the frames.

1) Subsubsection Heading Here: Subsubsection text here.

B. The Use of HMM in Speech

Hidden Markov Models are an example of unsupervised learning where there are no target labels for the model to decrease the loss on. These models are commonly used in speech recognition for finding the state transition and emission probablities. The state transition and emission probabilities of the HMM is trained by using Baum-Welch algorithm which is an expectation maximization algorithm. These probabilities fit themselves according to the given observation sequences. After the HMM is trained, the hidden state sequence are decoded by Viterbi algorithm which finds the sequence that finds the maximum likelihood path.

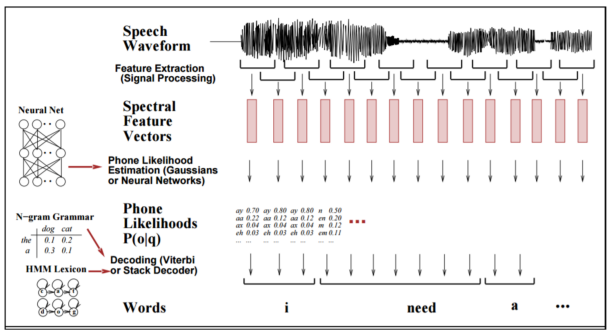

Fig. 4. Train error

For speech applications the HMM is used for decoding the speech observations which can be phonemes or words. The idea is to find the phoneme that maximizes the product of the likelihood and the prior probabilities. The likelihood probability includes the acoustic information which is given the acoustic features, which phoneme is observed. The prior probability includes the language information which is given a sequence of words, what is the probability of the next word or phoneme. For language modelling, the conventional method is to use the n-grams method, but recently this method is replaced by the recurrent neural networks (RNN), or gated recurrent neural networks, such as LSTM. For this project, we have implemented a bigrams method as language modelling. This predicts the probability of given the previous phoneme what is the probability of the next phoneme. For the acoustic observation, the convention was to use the Gaussian Mixtures Models (GMM) which model the likelihood probability for each state. However, the GMM's are replaced recently by CNN's which are used for extracting the observation probabilties used in HMM. Figure 5 shows the main parts of a HMM/DNN hybrid. The speech are seperated into its frames, and the features are extracted by using method such as mel frequency cepstral coefficients. Then, the hidden states are trained, and after the training, the likelihood or the observation probabilities are used as the emission probabilities from the hidden states of the HMM. Then, the observation sequence which was given by the deep neural network was used in the Viterbi decoder with the state transition and emission matrices. The decoded output gives the likely states that are results of the Viterbi decoder.

The paper that we implemented use a CNN for extracting these probabilities. It first trains a GMM/HMM model for finding the hidden state allignments. Then, it uses these state

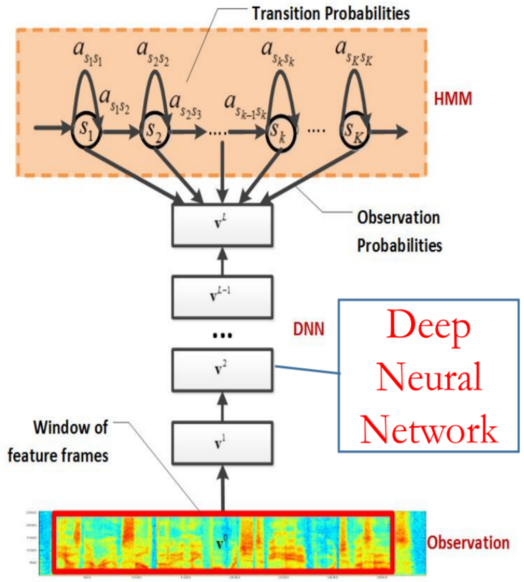

Fig. 5. HMM with DNN

allignments as target variables for the CNN model to train on. This method is called forced allignment. After the CNN is trained, the HMM model is used for decoding the hidden states by using the Viterbi decoder. The HMM used in the paper is a 3 state HMM for each of the given phonemes, and it trains one HMM for each of the phonemes.

In the project, we have implemented the HMM by using 39 states which correspond to the 39 output neurons of the CNN. This 39 states represent 39 different phonemes. So, as the acoustic modeling, we used the output of the HMM, but unlike the paper implementation, we have taken the phoneme that has the highest probability and send the probabilties of the other states to a small value. Then, as the language modeling, we used bigrams method for each of the transition probabilities of the hidden states that emit a single phoneme. The state transition probabilities are calculated using the frequency of the phoneme, or the output of the CNN layer. This determines the state transition matrix, and with the emission matrix and the given observation sequence, the Viterbi algorithm finds the most likely states to generate the phoneme.

III. Implementation Details

The following are the hyper-parameters and the implementation details about this model. The implementation is made using Knet[1] Library for Julia. The model is trained on Amazon GPU instances. Optimization methods used are Adam and SGD. Adam seems to work better for this framework. It takes longer time for SGD to converge. No learning rate is used. Adam optimizer configures a learning rate by itself.
The convolutional layer sizes are taken from the original paper. The sizes of the convolutional layers can be changed within the limitation of the memory of the current workspace. However, it should be noted that larger convolutional layer sizes increase the total time it takes for the model to train.

IV. Results

We have trained our model on GPU for approximately 2 hours. Figure 6 [train error] shows the train loss using Adam optimizer. After 60 iterations the train error is 0.5. Continuing to train the model after this

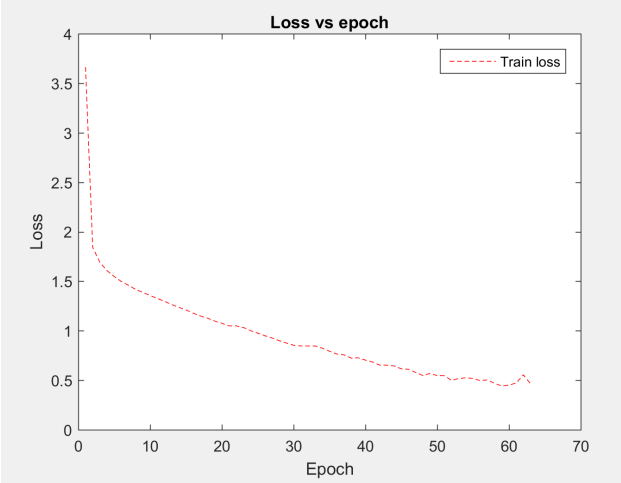

Fig. 6. Train error

V. Conclusion

In this paper, we have described how to apply hidden Markov models and Convolutional Neural Networks to speech recognition. The paper shows that a performance improvement can be obtained relative to the standard use of Deep Neural Networks with the use of weight parameters. The most significant improvement is obtained using the multiple frame per iteration approach. Giving multiple frames at a time to convolutional neural network improved the accuracy of the model significantly. (about 20% relative increase). The proposed framework of hybrid CNN-HMM approach increase the performance of hidden Markov model with the use of starting the hidden Markov model with the likely predicted observation from the output of convolutional neural network. In addition, this paper also proposes a new limited weight sharing that is standard in convolutional neural network applications for image processing but not used in speech processing that often. This limited weight sharing technique leads to fewer number of units in the pooling layer which lead to a smaller model size. This is efficient because it reduces the computational complexity of the model which normally uses full weight sharing technique.
TIMIT phone recognition performance can be increased with a variety of convolutional neural network parameters and model design settings. One should note that the use of random weights initialization for the hidden Markov model yields a decrease in the accuracy of the model and Gaussian Mixture Model should be used instead of random weights initialization

if possible. Finally, the result of this experiment is feasible even with a slight error rate of the model.

ACKNOWLEDGMENT

The authors would like to acknowledge Engin Erzin for his helpful lectures and advices.
In addition, the authors would like to thank the owners of TIMIT dataset.

REFERENCES

[1] Yuret, Deniz *Knet: beginning deep learning with 100 lines of Julia, 2016* Machine Learning Systems Workshop at NIPS 2016
`https://github.com/denizyuret/Knet.jl`