CNN model for speech recognition

Tao Wang
941015-5338
taowan@kth.se

Gao Yuan
940410-0662
yga@kth.se

Abstract

The combination of deep neural network (DNN) and hidden Markov model has been proved to outperform Gaussian mixture model, mainly due to the fact that DNN can model features with more complex correlations. In this paper, we investigate Convolutions Neural Network (CNN) for continuous speech recognition. Compared with DNN, CNN has shown remarkable progress in speech recognition task nowadays. It can reduce the size of the neural network model significantly and at the same time achieve an even better recognition accuracy. We experiment on different network structures on TIMIT dataset and give an understanding of the factors that control the accuracy of CNN.

1 Introduction

1.1 Automatic speech recognition & related works

With the development of the technology, the interaction between the computer and human beings becomes more and more prevalent. To achieve this objective by voice interaction, we need to achieve three goals: Automatic Speech Recognition (ASR), Speech Encoding, and Speech synthesis. What we focus on in this paper is Automatic Speech Recognition. The traditional method for ASR is to use the hybrid structure of Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM) where HMM models the temporal behavior and GMM estimates the posterior probability for each phone. However, this method comes with some limitations. To make a breakthrough, researchers start to apply artificial neural network to solve the problem.

By 2006, people applied the artificial neural networks to estimate the posterior probabilities of a continuous density HMMs' state given the acoustic features and then backpropagation algorithm was used to train that network[3], however most of them failed because the gradient in the traditional network is unstable and the structure of a network with more than two layers is complicated. Even though the deep neural network is developed, the high requirement of the initial parameter setting in the network makes it relatively hard to train. Until 2006, Hinton[4] proposed the layer wise pre-training deep network, and later on, Microsoft's research based on Context-Dependent Deep Neural Network[3] made deep neural network become increasingly popular in Automatic Speech Recognition.

1.2 Convolutions neural network

Even with Hinton's pre-training method, it is still difficult to introduce the traditional training method in DNN. Whereas Convolutional Neural Network(CNN) is a special case which can use Back Propagation to train the network. CNN is actually a variant of DNN, different from the fully-connected DNN. It uses a small shared filter to connect layers and each filter can produce a feature map of a current layer. Thus compared with the fully-connected DNN, CNN has fewer parameters, which can accelerate the training process with Back Propagation algorithm.

At the beginning of the CNN study on speech recognition, researchers usually focus on the convolution along the time axis[5], while Ossama et.al[2] found a hybrid CNN-HMM approach delegating

temporal variability to the HMM, which concerned more about the convolution along the frequency axis. And they also found that[1] the success of CNN was attributed to the learned features that are invariant to small frequency shifts of speech patterns (e.g. formants), which increases the robustness to speaker variations, explaining why CNN works well on phoneme classification.

In the paper, we consider to use Ossama's data processing method, using *static, delta, delta-delta* three-layer data instead of simply using MFCC to fit and build a CNN model. We try different parameter settings and make some slight changes to the network structure in order to evaluate and compare the performance with the result of DNN.

The paper is organized as follows. In Section 2, we introduce how we extract features and the network structure of CNN including convolutional layer, pooling layer, activation function, fully-connected layer and decoding. In Section 3, we describe in detail that how we perform the experiments including data preparation, tool, training and testing process. Then we report our results generated by our CNN with different hyper-parameter settings and final error rate. In Section 4, we discuss and conclude what limitations we are facing currently and how we plan to improve our network model.

2 Methods

2.1 Feature extraction

When a pattern recognition task is solved by CNN, the input data needs to be organized into a series of feature maps. When processing with images, RGB can be viewed as three channels of input and each channel is a feature map. In the process of extracting speech features, the image can be loosely represented as a spectrogram with static, delta and delta-delta features(first and second temporal derivatives). Each kind of spectrogram can be considered as one channel which corresponds to that of R,G,B in a real image. For example, we divide the spectrogram with a small window in DNN like the context-dependent trick, where each window is an input image, as shown in figure 1. As for the frequency, Ossama's work[2] shows that if MFCC is used, the discrete cosine transform projects the spectral energies into a new basis that can not maintain locality, thus we turn to utilize mel-frequency spectral coefficients(MFSC) and its delta and delta-delta as our feature map. In fact, MFSC is the MFCC feature before doing DCT transform.

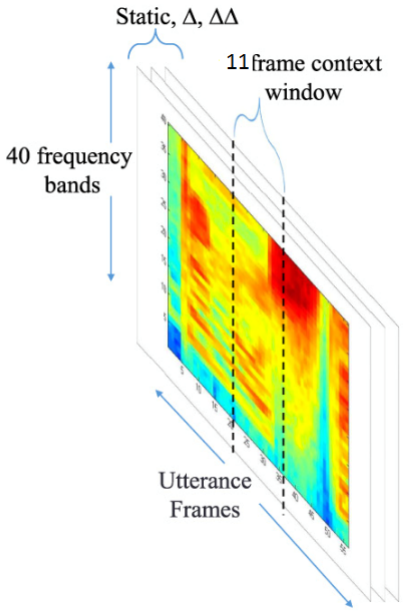

Figure 1: We use 40 MFSC features and the first and second derivatives with a context window of 11 frames for each speech frame, the figure is from[2]

.

2.2 Network structure

2.2.1 Convolutional layer

The convolution layer is the core part of the CNN. It can generate a series of feature maps using several filters. The convolution process for feature maps can be written as:

$$A = f(\sum_{d=0}^{D-1} X_d * W_d + b) \tag{1}$$

where A is the output feature map, and D represents the number of input channels. The multiply sign here is actually the convolutional process. We can also express it more specifically, that is:

$$a_{i,j} = f(\sum_{d=0}^{D-1}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} W_{d,m,n} * x_{d,i+m,j+n} + w_b) \tag{2}$$

where filter size is $M \times N$. It has D dimensions and w_b is the bias added to the result after the summation. As shown in figure 2, D is 1, which means we only have one channel as input. The blue

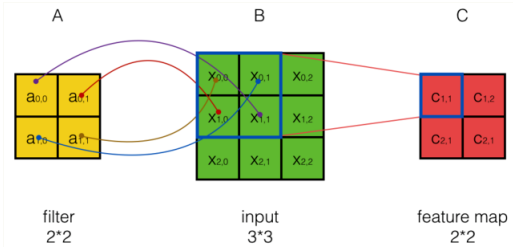

Figure 2: An example of convolution process

square frame in graph B of figure 2 is filter window (also called kernel), it will be slided to cover all "pixels" in the map. And "stride" is used to define the way we move the window. For example, stride of size 1 means every time we move the window with only one row or one column. And in our case, as the input feature map is already small enough($3 \times 11 \times 40$), the stride will always be 1.

2.2.2 Pooling layer

Pooling layer is commonly used in CNN with the purpose of reducing the spatial size of the representation gradually to decrease the number of features and complexity of the model. Every time a filter process is over, we need to carry out a down-sampling to reduce the number of parameters further (The filter has already reduced the number of parameters). This down-sampling is also known as pooling. Pooling helps to remove unimportant messages and reduce the size of the input map to accelerate the training.

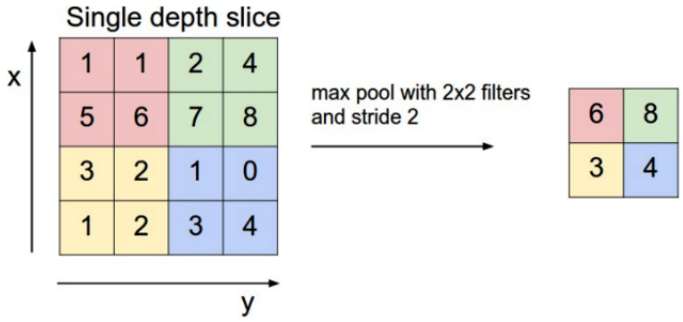

Figure 3: An example of 2x2 pooling process from [8]

The most common pooling method is max-pooling, which is shown in figure 3. In each pooling window, the largest pixel value is selected to represent this pooling window. The other pixels in

this pooling window will be considered as irrelevant and being ignored. Also, there are some other methods like mean-pooling. But we choose to use max-pooling in our CNN and the pooling size will be described in the experiment section.

2.2.3 Activation function

With the experience of multi-layer perception network, we know that if a layer is just a linear combination, then we can use only one layer to represent this multi-layer structure. Thus to avoid this situation, we need to transform our cost calculation into a non-linear combination. That is why an activation function is needed. We choose sigmoid function and tanh function that are commonly used as the activation functions in CNN (Because of the version incompatibility between toolkits that we use, we are not able to implement the ReLu activation function.) After each pooling process, an activation function is applied to the output feature map.

2.2.4 Fully-connected layer

After two or three iterations of Convolution and Pooling we will get a group of small feature maps. Then they are flattened into a long vector as the input of a fully-connected layer. Also like DNN, we could have a few hidden layers which are also fully-connected with each other. And for the last layer, we use softmax to calculate the posterior probability of each node.

2.3 Decoding

In the DNN part, just similar to what we have done in lab3. The outputs of DNN are not the exact phonemes. Instead, they are the sub-phonemes. A phoneme can be split into several sub-phones. For example, a phoneme `aa` could be split into several sub-phones, such `aa_1`, `aa_2`... .And in the decoding part, we will merge them together. To achieve this goal, we use decoding functions provided by Kaldi. When decoding, as it is suggested in [6], we map the 61 phonemes to 39 classes as shown in figure 4.

aa, ao	aa
ah, ax, ax-h	ah
er, axr	er
hh, hv	hh
ih, ix	ih
l, el	l
m, em	m
n, en, nx	n
ng, eng	ng
sh, zh	sh
uw, ux	uw
pcl, tcl, kcl, bcl, dcl, gcl, h#, pau, epi	sil
q	-

Figure 4: Mapping from 61 phonemes to 39 classes. The phonemes in the left column are folded into the labels of the right column and q is discarded

3 Experiments & results

We decided to use this system to finish the phoneme recognition task . So in this section we present the steps we take to achieve this goal. A first overview is given by the following summary:

- **Data preparation**
  - We use TIMIT dataset and split the original training set into training and validation set.
  - Using Kaldi to extract and normalize the MFSC feature.

- **CNN training**
  - We use PDNN to train CNN network.
  - Try different hyper-parameters setting and obverse the change of the error rate on training and validation set.
- **Decoding**
  - Firstly put the test file into CNN and get an output by PDNN.
  - Then, Use Kaldi to decode and evaluate this output.

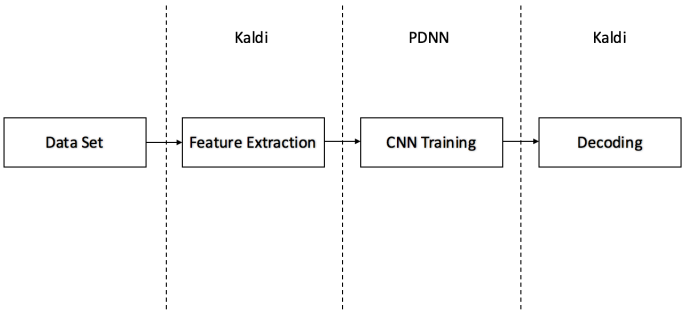

Figure 5: An overview of the working process

3.1 Data preparation

We use TIMIT (Acoustic-Phonetic Continuous Speech Corpus) dataset for the training and testing data. Different from TIDIGIT dataset that is to recognize digit 0-9 in the lab, TIMIT contains a total of 6300 sentences (5.4 hours), consisting of 10 sentences spoken by each of 630 speakers from 8 major dialect regions of the United States. And TIMIT is split on phone-level and labeled with phones manually.

There are five steps of data pre-processing:

- **Split dataset**
  Choose around 95% training data files as the training set and the remaining 5% as validation set.
- **Forced aligned transcriptions**
  Create level time aligned transcription for the training to increase the readability with MFSC features as Lab3.
- **Feature extraction**
  Like we have already talked in Section 2, we will extract the MFSC features and its first and second derivatives using Kaldi. Actually, the MFSC features are the FBANK features. The sampling rate of the TIMIT is 16kHz and we use 40 mel bins to get the MFSC features.
- **Feature normalization**
  Features are normalized to make sure that every feature coefficient has unit variance and zero mean. The computed original mean and variance from the whole training set are also used for normalizing test set.
- **File formats**
  As speech is a type of sequential problem, we need to keep track of the sequence boundaries. Thus, we choose to use specially designed data format pfile from a number of feasible ways.

3.2 CNN training

3.2.1 Training basic networks

For the first training, the size of input is $3 \times 11 \times 40$. We apply two filters with the size of 9×9 and 3×4. We also add two max-pooling layers after each convolutional transformation with the size of

1×3 and 1×1, which means we only focus on the frequency, assuming that the time variability can be modeled by HMM. Two fully-connected layers with 1024 hidden units per layer are followed by convolutional layers and the dimension of output is 1947, which is the default class number used in Kaldi while using the GMM model instead of neurual network. So we will keep this class number and consider the phone is generated from 1947 different probability density functions(PDF). Figure 6 shows our CNN network architecture for the training. The number before @ sign indicates the depth of feature representation for each layer. Also, we set momentum to be 0.9 since CNN over filterbanks seems to converge slowly. Thus we increase momentum to speed up convergence.

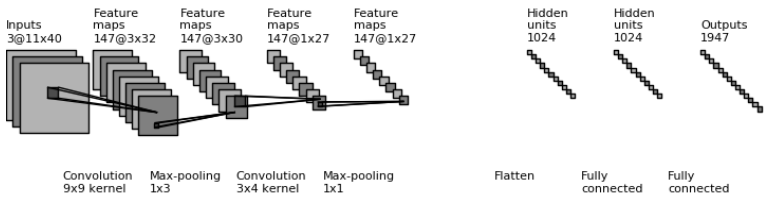

Figure 6: The network architecture

3.2.2 Network structure parameters

The network parameters in the convolutional layers we use are represented in the format: $t \times n \times m$: $a, b \times c, pd \times e$, where the first part $t \times n \times m$ means there are t input feature maps, each with the dimension of $n \times m$. The second part $a, b \times c, pd \times e$ describes one convolutional layer and a indicates the number of feature maps (the number of filters) where $b \times c$ means the size of local filters (kernels). $d \times e$ means the max-pooling size. We set our basic network structure as: $3 \times 11 \times 40$:$147, 9 \times 9, p1 \times 3$:$147, 3 \times 4, p1 \times 1$, two fully-connected layers with 1024 hidden nodes and the dimension of outputs is 1947. The learning rate of our network is defined as follows: D:0.08:0.5:0.2,0.2:4, that is, the learning rate starts with 0.08, if the validation error reduction between two consecutive epochs is less than 0.2, the learning rate is scaled by 0.5 during each of the remaining epochs. Training finally terminates when the validation error reduction two consecutive epochs falls below 0.2. And 4 is the minimum epoch number after which scaling can be performed.

The following figure is the training and validation error of the network training (not the final error rate of phoneme recognition) after each epoch for the first four networks in the table shown in section 3.3. network (a) has two convolutional layers and two fully-connected layers, network (b) has one convolutional layer and two fully-connected layers, network (c) has two convolutional layers and one fully-connected layer, network (d) has two convolutional layers and three fully-connected layers.

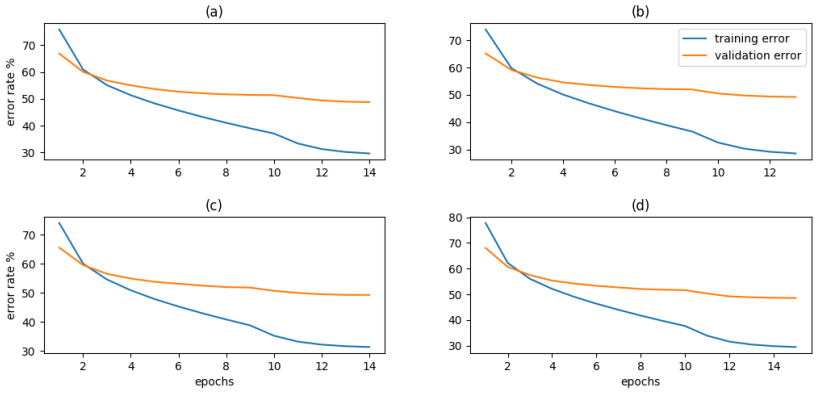

Figure 7: The training and validation error after each epoch

3.2.3 Dropout

We also add dropout with rate 0.2 to the best network from the first-round experiment for each default activation layer. Dropout is a regularization technique where randomly selected neurons are ignored during training for reducing overfitting, which means the contribution of these drop-out units to the activation is temporarily removed on the forward pass and weight updates are not applied to the neuron on the backward pass[7].

3.2.4 Activation function

After comparing the test accuracy of the above four networks, we find that the network (a) as the best with two convolutional layers and two fully-connected layers is superior than the others. The default activation function for both convolutional and full-connected layer is sigmoid. We now change the activation function on the network into tanh and test the performance on this new network. The phoneme error rate on validation and test set are shown in next section.

3.3 Decoding

Since the time cost of decoding is huge, we only select 24 utterances from the original test set as our own test set. We perform decoding process on the validation set with 15057 words and test set with 7215 words. The phoneme error rate (PER) is computed as follows:

$$PER = \frac{S + D + I}{N} \tag{3}$$

where N is the number of words, S is the number of substitutions, D is the number of deletions and I is the number of insertions.

Table 1 shows the average phoneme error rate on the validation and test set for different network structures, regularization techniques and activation functions. We find that the network (e) with two convolutional layers, two fully-connected layers, sigmoid activation function and dropout 0.2 gives the lowest test error rate, which is 18.0%. From network (a) and (f), we cannot make an obvious distinction between using sigmoid and tanh function. However, once dropout technique is added, both validation and test error rate decline significantly, which can be seen from network (a) and (e). Typically, a CNN network with more convolutional and fully-connected layers within normal limits tends to produce a better result. However, in our case network (d) with the most layers gives a relatively high error rate. Due to the complexity of the network caused by number of hidden nodes and layers, we analyze that there is an overfitting situation in the model.

Table 1: Phoneme error rate on validation and test set

No.	Conv layer	Fully-con layer	Dropout	Acti func	Valid err	Test err
a	2	2	0.0	sigmoid	17.4	18.4
b	1	2	0.0	sigmoid	17.4	18.6
c	2	1	0.0	sigmoid	17.5	19.2
d	2	3	0.0	sigmoid	17.5	19.2
e	2	2	0.2	sigmoid	17.1	18.0
f	2	2	0.0	tanh	17.3	18.9
g	2	2	0.2	tanh	17.4	18.7

We also carry on a statistical analysis for network (a) to (d), to calculate the correct rate for each phoneme. Here is the result:

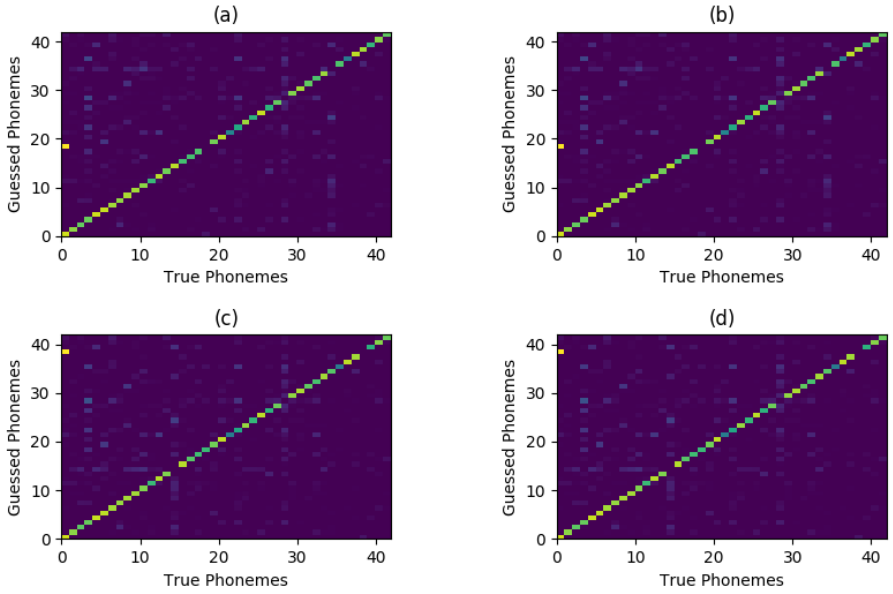

Figure 8: The statistic results for the phone recognize result with network(a)-(d).

In this color mesh, each point represents a normalized statistical count. Lighter color means the count is higher. Thus the diagonal of this map is very shallow, indicating that most phonemes can be correctly recognized.

We also experiment on DNN with the same setting as our best model and the average error rate is 19.9% on validation set and 22.3% on test set in total. Therefore, we conclude that the performance of phoneme recognition with CNN is better than that with DNN for TIMIT dataset.

4 Discussion & conclusions

In the paper, we describe and explore how CNN can be used in Automatic Speech Recognition with different network structures from feature extraction to classification of recognized phonemes. The experimental results of PER indicates that there is an obvious improvement in relation to traditional DNN methods. We delegate HMM to finish the temporal variability instead of doing convolutional process over time, rather convolving along the frequency axis, which provides small frequency shifts invariance to some extent.

In the future work, we intend to adjust our model on large vocabulary task where there are a lot of challenges. For instance, words may not be distinguishable based on their acoustic information alone in continuous speech and the memory and computational requirements will become very large especially in terms of decoding. To implement this, we will perform CRBM-based pretraining on convolutional layers in CNN referred to [1]'s attempt, which is found to improve performance on a large vocabulary speech recognition task.

References

[1] Ossama Abdel-Hamid, Li Deng, and Dong Yu. Exploring convolutional neural network structures and optimization techniques for speech recognition. In *Interspeech*, pages 3366–3370, 2013.

[2] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[3] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[4] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[5] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

[6] Carla Lopes and Fernando Perdigao. Phone recognition on the timit database. *Speech Technologies/Book*, 1:285–302, 2011.

[7] Vieira Sandra, H.L. Pinaya Walter, and Mechelli Andrea. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience Biobehavioral Reviews*, 2017.

[8] Stanford. Cs231n convolutional neural networks for visual recognition. 2016.