Journal of Electronic Research and Application

Research Article

A Speaker Recognition System Based on Deep Learning

Haowei Li

Georgia Institute of Technology, Atlanta 30332, Georgia, United States.

Abstract: This paper lies in the field of digital signal processing. This is a speech recognition system that identifies the different speakers based on deep learning. The invention consists of the following steps: Firstly, we collect the voice data from different people. Secondly, the data having been selected is preprocessed by extracting their Mel Frequency Cepstral Coefficients (MFCC) and is divided into training set and test set randomly. Thirdly, we cut the training set into batches, and put them into the convolutional neural network which consists of convolutional layers, max pooling layers and fully connected layers. After repeatedly adjusting the parameters of the network such as learning rate, dropout rate and decay rate, the model will reach the optimal performance. Finally, the testing set is also cut into batches and put into the trained neural network. The final recognition accuracy rate is 70.23%. In brief, the research can automatically recognize different speakers efficiently.

Publication date: December, 2019

Publication online: 31 December, 2019

****Corresponding author:*** Haowei Li, zhaogechong@xyzrgroup.com

1 Introduction

Speech is an important biological information and the most effective way of human communication. With the rapid development of information technology, computer is widely used in various aspects of human social life and speech recognition can improve work efficiency in many situations such as telephone communication, mobile payments, machine control and airport security. How to make the computer recognize the identity of the speaker in the human-computer interaction is of great significance to public safety, information security and property security.

In the 1950s, a speech recognition system that recognized 10 English digital pronunciations was accomplished in Bell Labs, marking the beginning of modern speech recognition research[1]. With the wide application of computer technology in the field of speech recognition, a series of important research results have been obtained. The concept of Mel Frequency Cepstral Coefficients (MFCC) was proposed in the 1980s and is still an important feature parameter in speech recognition[2]. The introduction of artificial neural network (ANN) has injected new vitality into speech recognition technology and promoted its development[1].

Deep learning is a branch of machine learning research which can be understood as the development of ANN. It is essentially a method of training deep structural models and it is also an algorithm for modeling complex relationships between data through multiple layers[3]. With the development of high-performance computing platform and big data processing technique, deep learning has become a powerful method for speech recognition. In many popular models, such as VGG, more convolutional layers and pooling layers are needed, and the processing of data is more complicated[4].

In this paper, TensorFlow is used to implement the deep learning framework. We first collect voice clips of different people in the environment of daily life and use MFCC to preprocess the data. After that, we randomly feed the training data set into a simpler convolutional neural network in batches. Parameters including learning rate, dropout rate and decay rate are optimized by observing the model's average accuracy and variance of the testing set until the model achieves the best performance. In addition, the model can also be used in fields such as image recognition.

2 Methods and Results

2.1 Data Preprocessing

In this project, we collect voice data from 5 people. All data is saved as WAV files and cut into 2000 segments of 7 seconds. The total amount of recording for each person is more than 45 minutes. In order to get sufficient data, each 7-second fragment is cut into 21 2-second parts with each part overlapping a little with each other. Then, we obtain 8400 recording files for each person. All files are renamed in form of *X_Y* where *X* stands for different people and *Y* is its serial number. This step helps us to load in data more efficiently.

Next, we extract Mel Frequency Cepstral Coefficients (MFCC) for all 42000 segments. There are mainly three steps to gain MFCC from WAV files.

(1) The spectrum of the signal is extracted by short-time Fourier transform.

(2) The energy spectrum is obtained by squaring the original spectrum and effective fragments is extracted by bandpass filtering.

(3) The logarithm of the output signal is taken from the filter and inverse Fourier transform is applied to get MFCC. A simplification of the formula is as follow:

$$C_n=\sum_{k=1}^{M}\log X(k)\cos\left[\frac{\pi(k-0.5)n}{M}\right] \quad n=1,2,\ldots,L \tag{1}$$

L is the coefficient of step numbers, and *M* is the number of triangle filters.

The final output of MFCC is a matrix of shape [*Z*×16] where *Z* is related to the bit rate of the source and its value should be more than 2000. However, we just need the first [1021×16] characteristic values to get a matrix of [32×32×16] for each fragment, so the surplus parts are cut off. We append a label corresponding to each person who recorded their voice. Finally, 80% of the whole data is randomly chosen as the training set while the other data composes the testing set. Both sets are saved as MATLAB files (.mat) for further use in the following steps[2].

Before training, we load the .mat files into storage to do some preprocessing work.

(1) The sets are reshaped and labels are transformed into one-hot encoding. For example, we transfer the label of the first speaker to [1,0,0,0,0], and the label of the third speaker to [0,0,1,0,0]

(2) All characteristic values are normalized to data ranging between -1 and 1. The range of all characteristic values is unknown at the beginning. We obtain the maximum and minimum of the characteristic values which are -112 and 85 respectively. So we normalize all characteristic values by:

$$N=\frac{C+13.5}{98.5} \tag{2}$$

N is the normalized value, and *C* is the original characteristic value.

Both methods allow the computer to process the characteristic values more efficiently.

2.2 Network Design

The network has in sum four convolutional layers, two max pooling layers and two fully connected layers. The input each passes through convolution layer 1, convolution layer 2, and max pooling layer 1, convolution layer 3, and convolution layer 4, max pooling layer 2, fully connected layer 1 and fully connected layer 2.

The function of convolutional layer is to extract diverse features from input. The first convolutional layer can only extract elementary features such as edge, line and corner. More layers of network can extract more complex features from low-level features through iteration.

There are some parameters referred to the calculation of convolutional layer that needs to be declared:

(1) Filter: the convolution kernel in convolutional neural network.

(2) Depth: it controls the depth of the output unit, which is equal to the number of filters.

(3) Stride: step size of the convolution kernel in convolution

(4) Receptive Field: the width(height) of the convolutional kernal.

Zero-padding: the value of zero-padding has two situations: “Valid” means there is no padding, while “Same” means the output image is the same size as the input image. In this program we use “Same”.

The width(height) of the output data matrix can be calculated by the formula below:

$$\frac{W-F+2P}{S}+1 \tag{3}$$

where *W* is the width(height) of input, *F* is the width(height) of the convolution kernel, *P* is the value of zero-padding and *S* is step size.

After convolutional layers, an output with large dimension is obtained. The function of the max pooling layer is to cut the matrix into several regions, take the maximum value in each region and combine them into a new feature with a smaller dimension.

Fully connected layer connects every node in one layer to every node in the next layer. The input matrix goes through a fully connected layer, and the activations of nodes can thus be computed. Eventually, the classify output is calculated.

We use Rectified Linear Unit (ReLU) in convolutional layers and fully connected layer1. The function of ReLU is

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} x, & x > 0 \\ 0, & x \le 0 \end{cases} \quad (4)$$

The using of ReLU unit can effectively alleviate the issue of gradient disappearance and overfitting.

The specific situation of each layer in this project is introduced below.

(1) Convolutional Layer 1

The input data of convolutional layer 1 has the shape [32×32×1]. It is convoluted by a [3×3×1] convolution kernel. The convolution kernel moves through the horizontal and vertical direction of the input data. Its stride is 1 and zero-padding is 1. Therefore, the width(height) of the output data is

$$\frac{32+2\times1-3}{1}+1=32$$

The number of filters is 16, namely the depth of output data is 16. As a result, the shape of output data is [32×32×16].

After convolution, we input the result into ReLU unit. After this procedure, the size of data is still [32×32×16].

(2) Convolutional Layer 2

The input data of convolutional layer 2 has the shape [32×32×16]. It is convoluted by a [3×3×16] convolution kernel. The convolution kernel's stride is 1 and zero-padding is 1. Therefore, same as the calculation of convolution layer 1, the width(height) of the output data is 32. The number of filters is also 16, namely the depth of output data is 16. As a result, the shape of output data is [32×32×16].

After convolution, we input the result into ReLU unit. Still, after this procedure, the size of data is [32×32×16].

(3) Max Pooling Layer 1

In this invention, we set max pooling layer 1 with filter of size [2×2], with a stride of 2. It slices the input by 2 along both width and height. When doing max operation every time, a 2×2 region will be taken and the max value will be calculated to substitute the region. The depth dimension remains the same. Therefore in the project the initial input volume has the size [32×32×16], and it is pooled with filter of size 2, stride 2. The output of size [16×16×16] is finally produced.

(4) Convolutional Layer 3

The input data of convolutional layer 3 has the shape [16×16×16]. It is convoluted by a [3×3×16] convolution kernel. The convolution kernel's stride is 1 and zero-padding is 1. Therefore, the width(height) of the output data is

$$\frac{16+2\times1-3}{1}+1=16$$

The number of filters is 16, namely the depth of output data is 16. As a result, the shape of output data is [16×16×16].

After convolution, we input the result into ReLU unit. After this procedure, the size of data is [16×16×16].

(5) Convolutional Layer 4

The input data of convolutional layer 4 has the shape [16×16×16]. It is convoluted by a [3×3×16] convolution kernel. The convolution kernel's stride is 1 and zero-padding is 1. Therefore, same as the calculation of convolution layer 3, the width(height) of the output data is 16. The number of filters is also 16, namely the depth of output data is 16. As a result, the shape of output data is [16×16×16].

After convolution, we input the result into ReLU unit. The size of output data is still [16×16×16].

(6) Max Pooling Layer 2

Same as max pooling layer 1, max pooling layer 2 has the filter of size [2×2], with a stride of 2. When doing max operation every time, it will substitute a 2×2 region with the max value of the four values. The depth dimension remains the same. Therefore it changes the input volume of size [32×32×16] with filter of size 2, stride 2. The output size is [16×16×16].

(7) Fully Connected Layer1

After finishing convolution, we rearrange our data. We reshape the data matrix from shape [batch, 8, 8, 16] to [batch, 1024]. We took the batch size of training set as 64, and test set's as 400. Therefore for the training data, the input matrix has the shape of [64, 1024] and for test data it is [400, 1024]. Accordingly, the input of the fully connected layer 1 has 1024 nodes, each of them is connected to all nodes of the next layer.

Then we define the value of weights of fully connected layer 1. We give each weight a value according to the law of normal distribution. The values of weights are stored in a matrix with the shape of [size× size× channels, number of hidden nodes], which in this program is [1024, 128]. We also define the referring bias. The value is 0.1 with shape [number of hidden nodes], namely [128].

To calculate the output of fully connected layer 1, we use the formula:

$$y = w^T x + b \tag{5}$$

in which y is the output of fully connected layer 1, w is the weight matrix, x is the input matrix, and b is bias. Here we choose ReLU as the active function. From this procedure we get the output from the fully connected layer 1. Its shape is [64, 128].

(8) Fully Connected Layer 2

We put the result above into the fully connected layer 2 as its input. Similarly, we define the value of weights of fully connected layer 2 according to normal distribution. The value of weights are stored in a matrix with the shape of [number of hidden nodes, number of layers], which in this layer is [128, 5]. We also define the referring bias. The value is 0.1 with shape [number of layers], namely [3].

In this layer, we use Softmax function to calculate the output. Softmax is a function that can output the probability that each classification is taken. Its formula is:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{N} \exp(x_j)} \tag{6}$$

where x_i is the value of the i-th element. The output gives out the probability of each label, namely the probability for the data to be judged as label 0 to 4.

To adjust weights to optimal values, we use back propagation. Back propagation is achieved by calculating the loss of each nerve node, which is:

$$\frac{\partial \ell}{\partial x} = w \cdot \frac{\partial \ell}{\partial \mathrm{y}}, \frac{\partial \ell}{\partial w} = x \cdot (\frac{\partial \ell}{\partial y})^T \tag{7}$$

where $y \in R^{m\times 1}$ is the output of node, $x \in R^{n\times 1}$ is the input of node, $w \in R^{n\times m}$ is the weight of node, and ℓ is the loss.

2.3 Optimization

During the implementation of our project, we found that the overfitting phenomenon occurred during the running of the program. Overfitting is that the accuracy of the training is very high, while the accuracy of the test is low. So we have these several ways to solve the problem.

(1) Regularization

Regularization has two forms- L1 loss and L2 loss. L2 loss has a unique value, while L1 loss doesn't have this feature. We finally use L2 loss to optimize because it can distinguish the result which is better. The formulas for L1 loss and L2 loss are:

$$\text{L1: } S = \sum_{i=0}^{n} \left| y_i - h(x_i) \right| \tag{8}$$

$$\text{L2: } S = \sum_{i=0}^{n} \left(y_i - h(x_i) \right)^2 \tag{9}$$

L2 can minimize the sum of the square of the differences between the target value and the estimate values. We use TensorFlow's function to make L2 loss work. The function utilizes the norm of L2 to calculate tensor loss value.

(2) Dropout

Dropout is dropping some nodes randomly in fixed probability at fully connected layer in training. If we drop some nodes randomly, it may avoid overfitting. We use TensorFlow's dropout function to drop nodes. The function can input tensor x and output value after applying drop ratio.

(3) Update

We have three methods to update loss and find the deep point.

The steepest descent method. We need to find a minimum value, so we can use the method to iteration search reversely in a fixed step size on the basis of current point gradient.

Momenta. Momenta give a direction which combines with the steepest descent method in parallelogram law. And the final direction obeys the law.

Adam. Adam is one of the fastest algorithms in all methods. It can find the deep point effectively.

Finally, we use Adam to update loss. Adam is more effectively and wider than the other methods. For our thousands of data, Adam can optimization the problem directly.

Adam calculate gradient of time, the formula is:

$$g_t = \nabla_\theta J(\theta_{t-1}) \tag{10}$$

First, it calculates the exponential moving average of the gradient, the formula is:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{11}$$

Second, it calculates the exponential moving average of the square of the gradient, the formula is:

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{12}$$

Third, we make v_0 and m_0 initialization which is 0, so we need to correct average of the gradient and average of the square of the gradient, the formula is:

$$\hat{m}_t = m_t / (1-\beta_1^t) \tag{13}$$

$$\hat{m}_t = m_t / (1-\beta_1^t) \tag{14}$$

Finally, Adam updates the parameter.

(4) Learning rate optimization

We hope the learning rate is neither high nor low. TensorFlow's decay function can keep the learning rate. We use this function to make learning rate rise first then

decay to make sure learning rate keeps medium. The learning rate will decay once in decay rate at a stable decay step. Our team use staircase decay mode, because its scale is larger, and it is easier to reduce outfitting.

(5) Framework optimization

In file 'dp_refines.py', models after training are saved to make us record. We also use API to simplify the code so that we can change or modify the parameters easier.

2.4 Training and Testing

Firstly, the file is converted into the form of mat according to the ratio that train and test is 4:1 before the training. The amount of collected data is 8400×5×16 and initial size of each data is defined as [32×32×1]. And then, to keep it from running out of memory, all the data cannot be processed simultaneously, we take the approach of batching data: define a variable of 'batch size' as the size of each batch, the computer only processes one batch of data at a time. We use 400 as the batch size of testing set, and 64 as the batch size of training set. Therefore, testing shape is [400×32×32×1], and training shape is [64×32×32×1]. After being handled by the structure of convolution layers and max pooling layers, the size of output data is changed to [8×8×16]. The loss generated during the training is what we use to judge the average accuracy rate of training and the value of loss depends on the parameters-base learning rate (initial valued 0.001). At the end of training, we adopt the method of 'Adam' to optimize result and reduce the loss generated. Besides, Softmax function gets the probability that each data belongs to each label. [4] The progress of testing is basically same as training except data set. The accuracy of testing is also presented in the confusion matrix, which shows the accuracy rate of each label.

3 Results

Table 1 shows the result when we take different values of parameters. We can achieve the optimal recognition accuracy of 70.23% when dropout rate is 0.95, base learning rate is 0.001, decay rate is 0.3, iteration steps is 20000, and λ is 5.0×10^{-4}.

Table 1. Recognition Result

Dropout rate	Base learning rate	Decay rate	Iteration steps	λ	average accuracy	standard deviation
0.95	0.001	0.3	23000	5.00E-04	67.97	2.23
0.98	0.001	0.3	23000	5.00E-04	66.84	2.3
0.92	0.001	0.3	23000	5.00E-04	66.47	2.26
0.95	0.0018	0.3	23000	5.00E-04	67.35	2.26
0.95	0.0002	0.3	23000	5.00E-04	64.37	2.31
0.95	0.001	0.36	23000	5.00E-04	66.36	2.29
0.95	0.001	0.26	23000	5.00E-04	66.54	2.43
0.95	0.001	0.2	23000	5.00E-04	69.17	2.38
0.95	0.001	0.18	23000	5.00E-04	67.49	2.28
0.95	0.001	0.3	26000	5.00E-04	66.86	2.41
0.95	0.001	0.3	22000	5.00E-04	65.1	2.27
0.95	0.001	0.3	20000	5.00E-04	70.23	2.13
0.95	0.001	0.3	19000	5.00E-04	64.87	2.41
0.95	0.001	0.3	23000	5.00E-04	67.97	2.23
0.95	0.001	0.3	23000	6.00E-04	67.39	1.8

4 Conclusions

Our invention uses Mel Frequency Cepstral Coefficients (MFCC) to process audio data shape into [32×32×1]. This approach makes data processing in subsequent convolutional networks more efficient and faster. In this invention, we randomly divide the original data into training set and testing set proportionally, which ensures the rigor and reliability of the experimental results. Our invention introduces regularization and dropout to avoid overfitting. The regularization makes the neural network intend to learn smaller weights. The dropout helps lessen dependence between neurons which makes the network more robust. By optimizing the parameters, our invention can effectively identify the speakers.

References

[1] Davis KH, Blddulph R, Balashek S. Automatic Recognition of spoken Digits[J]. Journal of the Acoustical Society of America, 1952, 24(6): 637-642.

[2] Mahboob B, Tahira H, Khanum M, et al. Speaker Identification Using GMM with MFCC[J]. International Journal of Computer Science Issues (IJCSI), 2015, 12(2): 126-135.

[3] Hou YM, Zhou HQ, Wang ZY. Overview of speech recognition based on deep learning. Application Research of Computers,1001-3695(2017) 08-2241-06.

[4] Feng GH. Small-scale Picture Classification Based on Convolutional Neural Network VGG[D]. Lanzhou University, 2018.