3. Proposed method

3.1 Fundamental algorithm

There are two main phases in speech-recognition system, training phase and recognition phase. Feature extraction and HMM, which would be briefly introduced below, are important issues in both phases. Viterbi and viterbi-like algorithms used in the recognition phase would be discussed, too.

3.1.1 Feature extraction—MFCC

There are usually six steps to calculate MFCC:

(1) Pre-emphasis—emphasizes the high frequencies

(2) Windowing—diminishes both ends of a frame

(3) FFT—transfers the signal to frequency domain

(4) Triangular band-pass filters—smooth the spectrum

(5) Log—transfers the spectrum to log scale

(6) DCT—transfers the spectrum to cepstrum

Pre-emphasis could be ignored with little effects on the recognition results. Windowing would also increase the continuity between adjacent frames. Triangular band-pass filters should be chose averagely according to Mel Frequency, which represents the sensitivity of human ears to sound frequencies. According to the steps above, we could get 13 coefficients, including 12 MFCC and the logarithm of energy. With two more steps, we could get 39 coefficients:

(7) 1^{st} order regression—delta coefficients

(8) 2^{nd} order regression—acceleration coefficients

Fig.2 demonstrates the whole procedure.

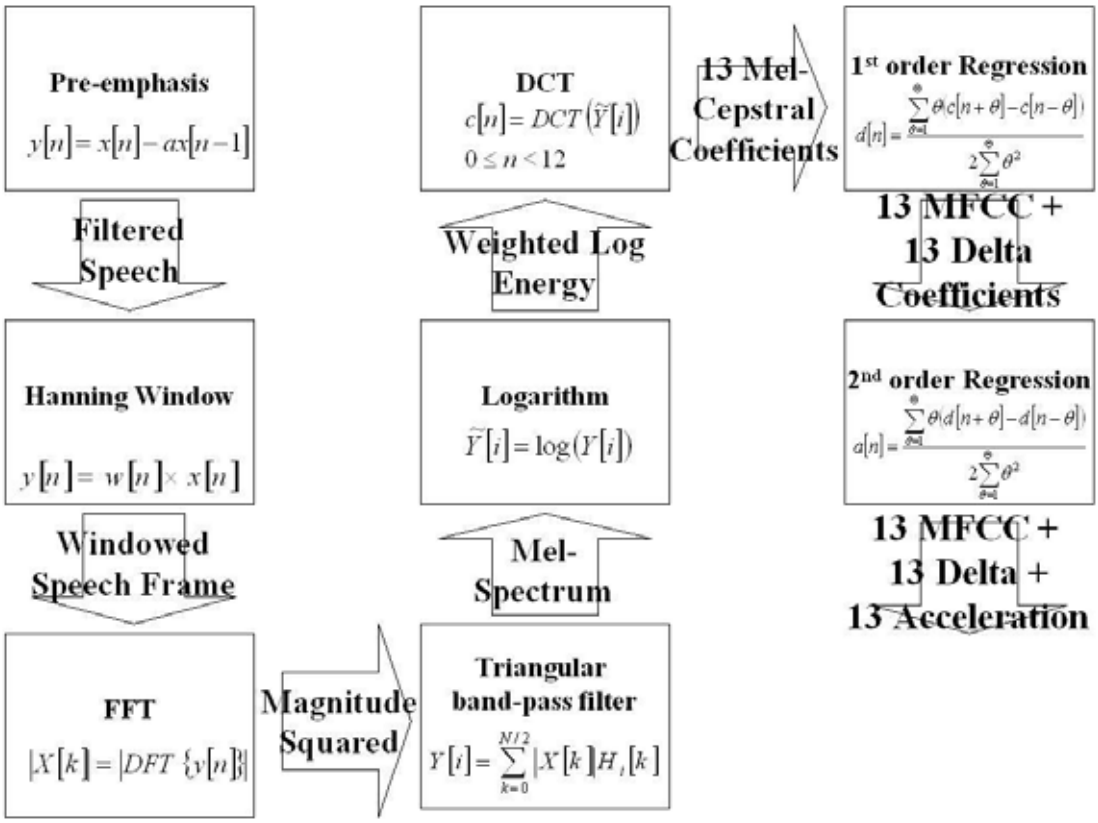

Fig. 2 Each step to calculate MFCC

3.1.2 Building acoustic model – HMM

DTW, Artificial Neural Network and HMM are all known methods to implement the speech-recognition system. HMM is popularly adopted recently because it requires less memory and is speaker-independent. We also use HMM, as shown in Fig.3, to implement the speech-recognition system on the embedded system. $b_j(o_t)$ determines the probability of generating observation o_t at time t. a_{ij} determines the associated transition probability between state i and j. We can compute $b_j(o_t)$ as shown in Eq. (3-1), where M_{js} is the number of mixture components in state j for stream s, c_{jsm} is the weight of the m'th component and $N(o;\mu,\Sigma)$ is a multivariate Gaussian with mean vector μ and covariance matrix Σ as Eq. (3-2). We can build the acoustic model according to features in the training phase, and the model will be used to do the

Viterbi decoding later in the recognition phase. For more details, please refer to [8, 9].

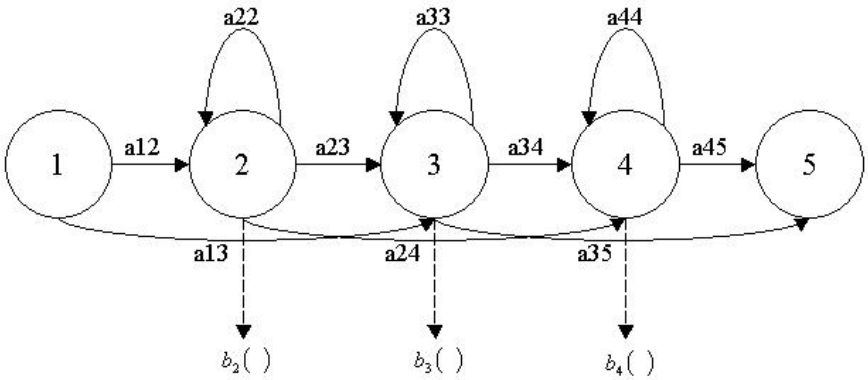

Fig. 3 Simple left-right HMM

$$b_j(o_t) = \prod_{s=1}^{S} \left[\sum_{m=1}^{Mjs} c_{jsm} N(o_{st}; \mu_{jsm}, \Sigma_{jsm}) \right]^{\gamma_s} \tag{3-1}$$

$$N(o; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(o-\mu)'\Sigma^{-1}(o-\mu)} \tag{3-2}$$

3.1.3 Searching the most-likely path—Viterbi algorithm

Viterbi algorithm is a dynamic programming approach to find the best state sequence in an HMM, as shown in Fig.4. The algorithm consists of four steps:

(1) Initialization—calculate the initialized probability

(2) Induction—calculate the rest probabilities according to the acoustic model

(3) Termination—find the max probability

(4) Backtracking—find the most-likely sequence

These steps are shown in Fig.5. Backtracking is optional depending on the purpose of the application.

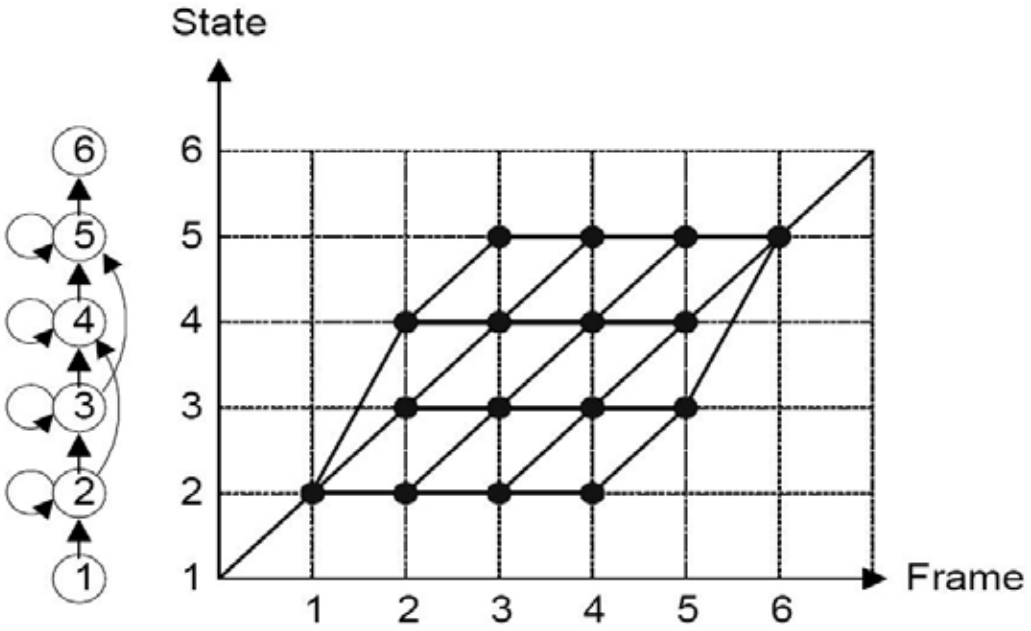

Fig. 4 Viterbi decoding

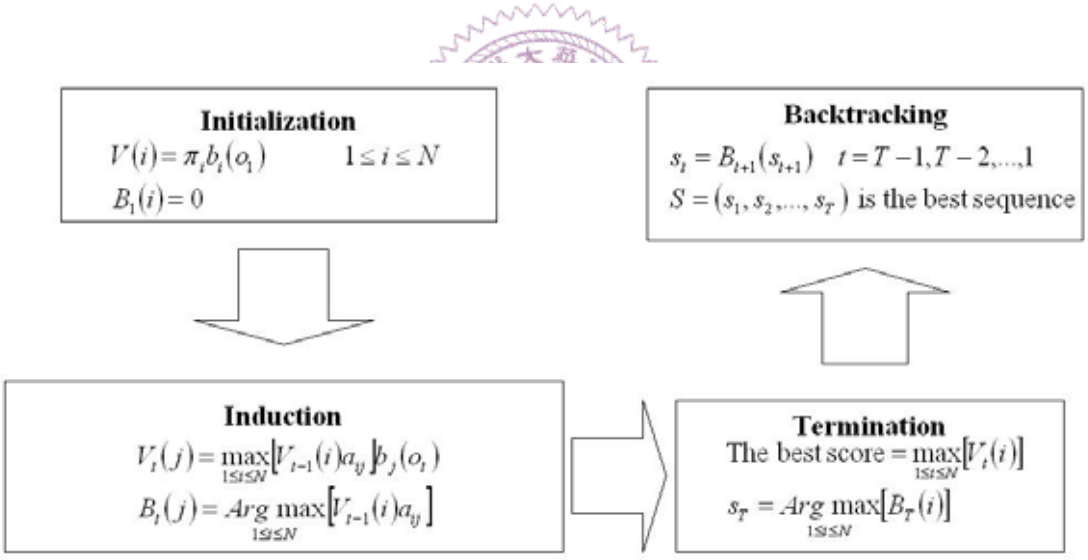

Fig. 5 Each step to do Viterbi decoding

Since the computation of the induction step in Fig.4 requires repeated multiplications with small values, it could lead to underflow. Hence, we could rewrite the equation as Eq. (3-3) by using the log likelihood instead.

$$\psi_t(j)=\max_{1\le i\le N}\{\psi_{t-1}(i)+\log(a_{ij})\}+\log(b_j(o_t)) \qquad (3\text{-}3)$$

The classic Viterbi algorithm computes the probability of the most probable path, without actually retaining the path. It is not enough for larger networks, where we are interested in the sequence of words spanned by the winning path. Thus we use a Viterbi-like algorithm, called the token-passing algorithm [10] instead.

The token-passing algorithm, as displayed in Fig.6, makes the concept of state path explicit. Imagine each state *j* of an HMM at time *t* holds a single movable token which contains the partial likelihood $\psi_t(j)$ and other information. The induction step represented by Eq. (3-3) is replaced by the following steps executed at each time frame *t*:

(1) Pass a copy of the token in state *i* to all connecting states *j*, incrementing the log likelihood of the copy by $\log(a_{ij})+\log(b_j(o_t))$.

(2) Examine the tokens in every state and discard all but the one with the highest probability.

The above-presented algorithm assumes each state to be an emitting one. We need a set of history records for the algorithm to keep track of the history of a token's route, and every token must carry a pointer to one of these records. The transition of a token from the exit state of a word to the entry state of another represents a potential word boundary. A new history record containing the value of the new word and a reference to the original history record is created. The token would be re-point to this new record.

The token emerging from the final state of the network at the end of the recognition process would refer a history record that can be traced back to obtain the full sequence of words the final token has passed through.

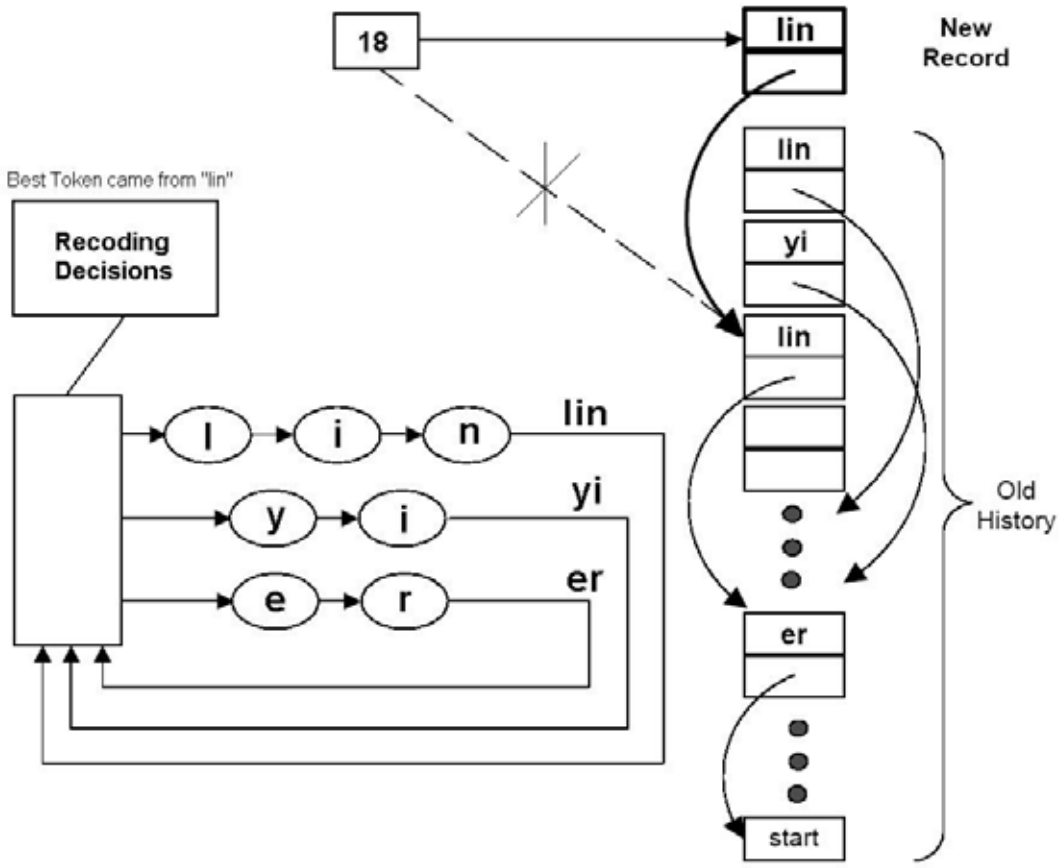

Fig. 6 Token-passing algorithm

3.2 Implementation on embedded system

We use “HP iPAQ H4100 Series-4150” pocket pc as the platform. Specifications are listed in Table 1.

Table 1: iPAQ H4150 specifications

System Feature	Description
Processor	Intel(R) PXA255 400 Mhz
RAM	64 MB SDRAM (55 MB main memory, user accessible)
Audio	Microphone, Speaker
Floating-point computation	Simulated by fixed-point computation

Since the floating-point computation is not supported directly on the 32-bit fixed-point processor, all computations in the recognition steps should be replaced with fixed-point operations.

3.2.1 MFCC

As we said in the Sec. 3.1.1, there are eight steps to calculate 39-dimensions MFCC. We will introduce fixed-point computations of each step, including problems we met and solutions to these problems. During the feature-extraction process, there are four kinds of error measures, which are listed below:

(1) Max absolute error $= \max_{i=0}^{N-1} \{ |dataA[i] - dataB[i]| \}$

(2) Max relative error $= \max_{i=0}^{N-1} \{ |(dataA[i] - dataB[i])/dataA[i]| \}$

(3) Avg. absolute error $= \left(\sum_{i=0}^{N-1} |dataA[i] - dataB[i]| \right) / N$

(4) Avg. relative error $= \left(\sum_{i=0}^{N-1} |dataA[i] - dataB[i]| \right) / \sum_{i=0}^{N-1} |dataA[i]|$

(*N: frame size*):

The dataset used to estimate these errors are digits 0 ~ 9. Each digit is recorded twice by a speaker and the details are listed in Table 2.

Table 2: Feature verification dataset

Speaker	38 males and 12 females
Sampling rate	16 kHz
Bits per sample	8 bits
Total	1000 files = 2000 seconds

1. Pre-emphasis

In this step, we replace the floating-point coefficient 0.95 with 15974>>14. In order to increase the accuracy, we also scale up the minuend with 2^{14}. The overflow will not occur in this step because 16-bit wave data are only scale up with 2^{14}, which

are still within the 32-bit range of the system.

The original equation could be modified as follows:

$$y[i] = (x[i] << 14 - 15974 \times x[i-1]) >> 14, \ 0 \le i < N \tag{3-4}$$

The errors in this step are listed in the following table:

	Absolute error	Relative error
Max	1.6016	6.2535%
Avg.	0.4815	0.0388%

2. Hamming window

We build look-up table of Hamming window. The window values are scaled from 0 ~ 1 to 0 ~ 16384 (2^{14}) and saved in the table, HamTable.

The original equation can be modified as follows:

$$y[i] = (\text{HamTable}[i] \times x[i]) >> 14, \ 0 \le i < N \tag{3-5}$$

The errors in this step are listed in the following table:

	Absolute error	Relative error
Max	4.5742	48.9457%
Avg.	0.5269	0.0786%

3. FFT

Due to the complicated computations of FFT, we adopt the "FFT of pure real sequences" algorithm [11] to speed up this step. With this algorithm, $\frac{N}{2}$-points complex sequences could substitute for the original N-points real sequence. If N is not 2^n (n is an integer), we need to pad N with zeros until N = 2^n. The basic FFT equation is like

$$F(f) = \sum_{n}^{N-1} f(n) e^{-j2\pi kn/N} \tag{3-6}$$

According to the Euler's equations,

$$e^{-jk} = \cos(k) - j\sin(k) \tag{3-7}$$

, which means lots of trigonometric functions would be used. Therefore, we build look-up table to replace trigonometric functions. The table sizes of cosine and sine are N, which is zero padded. The cosine and sine values are scaled from -1 ~ 1 to -1024 ~ 1024 (2^{10}) and saved in tables, cosTable and sinTable.

For instance, if we want to compute

$$y[n] = x[n] \times \sin(i \times \pi)$$

We could replace it with

$$y[n] = x[n] \times \text{sinTable}[i \times N] >> 10$$

Based on the experimental result, we scale up the values with 2^{10} instead of 2^{14} in this step.

The errors in this step are listed in the following table:

	Absolute error	Relative error
Max	47526.2281	3170883.4949%
Avg.	2598.6019	34.5058%

4. Triangular band-pass filter

We use 26 filters in this step. Again, look-up table is used for the filters. We scale up the filter values by 2^{11}, which is also based on our observation on the experimental data.

The original equation could be modified as follows:

$$y[i] = \sum_{k=0}^{N/2} \left(|x[k]| \times \text{filterTable}[i][k] \right) >> 11, \ 0 \le i < 26 \tag{3-8}$$

Because we use "FFT of pure real sequences" algorithm in the previous step, $x[k]$ is actually $x_r[k] + jx_i[k]$, and $|x[k]|$ should be $\sqrt{x_r[k]^2 + x_i[k]^2}$. Since $x_r[k]^2$ and

$x_i[k]^2$ exceed 2^{32}, which cause overflow, we make some adjustment to avoid this situation.

$$x_r[k] = x_r[k] >> 4$$

$$x_i[k] = x_i[k] >> 4$$

$$|x[k]| = \sqrt{x_r[k]^2 + x_i[k]^2} << 4$$

The errors in this step are listed in the following table:

	Absolute error	Relative error
Max	614.625	5.5027%
Avg.	104.0894	0.092%

5. Log

We build look-up table for log function by scaling up the log values with 896, which is an experimental result to meet the constraint of recognition process.

The original equation could be modified as

$$\tilde{Y}[i] = \log\text{Table}[Y[i]] \tag{3-9}$$

For example, $\log(2) = 0.693 \Rightarrow \log[2] = 621$.

$\log[Y[i]]$ and $\log[Y[i] + a]$ would be identical when $[Y[i]]$ is large enough, *a* grows with $[Y[i]]$. To prevent this redundancy, we group $[Y[i]]$ (dividing it by 148, approximate to n^5), and the Eq. (3-6) could be modified as

$$\tilde{Y}[i] = \log\text{Table}[Y[i]/148] \tag{3-10}$$

For example, $\log(2^{15}) = 10.397 \Rightarrow \log\text{Table}[2^{15}/148] = 9318$.

The errors in this step are listed in the following table:

	Absolute error	Relative error
Max	2.4587	0.0221%
Avg.	0.6782	0.0071%

6. DCT

Just like what we did with FFT, we use the look-up table instead of trigonometric functions in this step.

The errors in this step are listed in the following table:

	Absolute error	Relative error
Max	158.7221	2217.4588%
Avg.	21.6741	2.0115%

The six steps above could generate the basic 12 MFCC. The logarithm of energy is combined to form 13 coefficients. Because the frame size of 20ms (= 320 points with 16kHz sample rate) is used in this research, the sum of the square of energy would exceed 2^{32}. We divide the square of energy by 403 (approximate to n^6) first and compensate it after operating the logarithmic function. The logarithm of energy also needs to be scaled up by 896 for the same reason mentioned in the Log step.

$$E = \sum_{i=0}^{N-1} (x[i] \times x[i]) / 403$$

$$c[12] = (\log[E] + 6) \times 896$$

The errors of energy are listed in the following table:

	Absolute error	Relative error
Max	0.9423	0.0046%
Avg.	0.4057	0.0019%

7. 1st order regression

There is no complicated computation in this step so we just compute the coefficients with fixed point instead of floating point.

The errors in this step are

	Absolute error	Relative error
Max	1.2732	46.3005%
Avg.	0.4948	0.0556%

8. 2nd order regression

As mentioned in the 1st order regression step, we just compute the coefficients with fixed point instead of floating point.

The errors in this step are

	Absolute error	Relative error
Max	1.4165	55.8275%
Avg.	0.6009	0.1679%

Since all floating-point values are scaled, the overflow will possibly occur. We create a new class Int to deal with this problem. Operations that generate values exceeding -2147483648 (-2^{31}) ~ 2147483647 (2^{31}-1) would cause an overflow and be detected. An error message "overflow occurs" will be shown. With this class, we could conveniently estimate the scaling factors at each step.

To verify if the extracted features in this system is efficient, we test a minor dataset of digits 0 ~ 9, as listed in Table 2, with DTW and HTK (HMMs toolkit). The result is displayed in Table 3, which shows that the recognition rates of DTW and HTK using fixed-point features are even higher than that of the floating-point features. These features really have the ability for recognition.

Table 3: Feature verification result

	Fixed point	Floating point
Frame size	20 ms	20 ms
Overlap	10 ms	10 ms
DTW	80.0%	79.6%
HTK	79%	75%

3.2.2 HMM

Building acoustic model needs lots of computations so we leave this process on the PC, which has strong computational power. We use HTK to build HMMs in this research. We need to take care of some issue while using HTK to train acoustic model with fixed-point features, because HTK is basically used for floating-point features. At first we directly use fixed-point features without any adjustment. We found that "bad data or over-pruning" would be shown and ruin the training process. Therefore, scaling is used to solve this problem. Fixed-point features are scaled down as floating-point values and used for training. After training, we scale up the acoustic model for recognition. Fig.7 displays this adjustment.

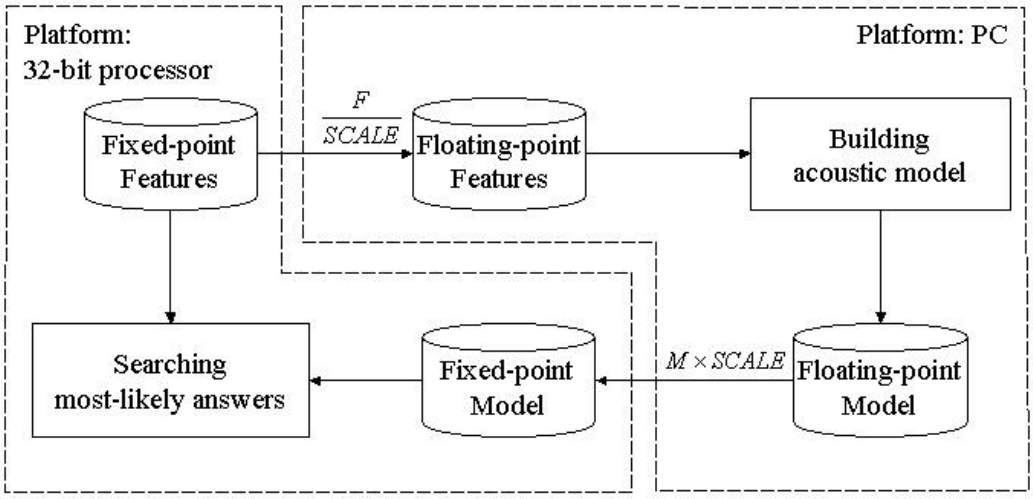

Fig. 7 Adjustment of training process

As the experimental result listed in Table 4, the mixture weight, mean, variance and transition probability of a model need to be scaled differently according to their range of the values.

Table 4: Statistic of model content

	Range	Scale
Mixture weight	0.7121449~0.00003	2^{17}
Mean	-2.053871~1.862582	896
Variance	1.000e-30~119.1154	896^2
Transition probability	< 1	2^{17}

There is no specific scaling rule except the relation between scales of mean and variance. According to Eq. (3-11), if the scale of mean is *s*, the scale of variance must be s^2.

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x[i]-\mu)^2}{N} \tag{3-11}$$

The HTK parameters used for training are shown in Table 5. For more details, please refer to [9].

Table 5: HTK parameters

Parameter kind	MFCC_E_D_A
State number	5
Stream number	1
Mixture number per stream	2~6

3.2.3 Token-passing algorithm

As we mentioned in Sec. 3.1.3, the computation of probability could lead to under- flow, thus we use the log likelihood instead. With log likelihood, the multiplication of probability could be replaced by simple addition. Due to the logarithm is considerably used, we need to speed it up by building a look-up table. We scale up the log value by 256.

For example, $\log_2(2)=1 \Rightarrow \log 2\text{Table}[2]=1\times 256=256$.

Since mixture weight and transition probability are scaled up by 2^{17}, the log value

should minus 17, as shown next:

$$\log_2(a\times 2^m)=\log_2(a)+\log_2(2^m)=\log_2(a)+m$$
$$\longrightarrow\quad \log_2(a)=\log_2(a\times 2^m)-m \tag{3-12}$$

For example, $\log_2(2)=1 \quad\Longrightarrow\quad \log 2\text{Table}[2]=(1-17)\times 256=-4096$.

As the same reason mentioned in the log step of MFCC, we also group the log value (dividing it by 1024) to save memory.

For example, $\log_2(2^{15})=1 \quad\Longrightarrow\quad \log 2\text{Table}[2^{15}/1024]=(5-7)\times 256=-512$.

The log of the observation probability $b_j(o_t)$ could be derived from Eq. (3-1) as

$$\log(b_j(o_t))=\log\left(\prod_{s=1}^{S}\left[\sum_{m=1}^{Mjs}c_{jsm}N(o_{st};\mu_{jsm},\Sigma_{jsm})\right]^{\gamma_s}\right)$$
$$=\sum_{s=1}^{S}\gamma_s\log\left(\sum_{m=1}^{Mjs}c_{jsm}N(o_{st};\mu_{jsm},\Sigma_{jsm})\right) \tag{3-13}$$

From Eq. (3-2), the computational complexity of $N(o;\mu,\Sigma)$ could be decreased if we take the log of it. To achieve this purpose, the log of summation needs to be revised based on Eq. (3-14).

$$\log(x+y)=\log\left(x\left(1+\frac{y}{x}\right)\right)=\log(x)+\log\left(1+\frac{y}{x}\right),\ \text{if } x\geq y \tag{3-14}$$

In Sec. 3.1.3, only the best token at each word boundary is saved. In fact, we save the best N tokens and this idea is called lattice N-best. With this idea, the potential tokens would not be discarded at the very beginning. But the drawback is the increasing computation and memory requirement. To solve this problem, a pruning process is adopted. The concept of pruning is only propagating tokens that have some chance of being amongst the eventual winners. It could be implemented at each time step by keeping a record of the best token overall and those whose log likelihood still stay within a beam-width below the best. If the pruning beam-width is set too small, the most likely path might be pruned before its token reaches the end. Therefore, setting the beam-width is a compromise between speed and recognition rate.

We also use the Int class, described in Sec. 3.2.1, to estimate the scaling factor in this process. Some relation between the scaling factor and the pruning beam-width is consequently revealed. If the scaling factors are too small, the error will be too large and the recognition rate will decrease. On the other hand, with large scaling factors, the pruning beam-width must be small, or tokens with overflowed log likelihood would still compete with the eventual winner. The pruning beam-width would also affect the recognition rate as we mentioned above. Therefore, there is a trade-off among pruning beam-width, scaling factor and the recognition rate.