Estimation of HMM parameters using Particle Swarm Optimization technique

Amir Hossein Harati Nejad Torbati

Temple University, PA, USA

amir.harati@gmail.com

Abstract

Nowadays, Hidden Markov Models (HMMs) are used widely in speech recognition and many other fields due to their powerful modeling capabilities for sequential data. However, one problem which always comes with HMMs is their suboptimal training. The traditional method for training a HMM is Baum-Welch algorithm, but this algorithm is always find a local optimum point, and in many applications it is a serious restriction. Recently, heuristic based techniques like Particle Swarm Optimization (PSO) have been proposed to overcome these limitations. It has been shown that PSO can improve the results of many problems facing with local optimums. It has been shown that PSO can be used to improve the training process of HMMs too. However, no one tried to train a state of the art speech recognizer using PSO technique yet, and all HMMs trained so far using this technique were a part of a relatively simple system. Our motivation to do this research is to evaluate the possibility of applying PSO technique to a complex speech recognition system.

Introduction

 Most of speech recognition systems today are based on Hidden Markov Models (HMM) with Gaussian mixtures. For many years, people have been trying different topologies, but they have ended up with relatively simple and constrained left to right models.

 A typical modern speech recognition system, contains a simple (usually 3-5 states) HMM model for each of its sub-word units (usually phonemes). The task, like almost all other machine learning applications, can be divided into two different phases.

 The first step is to train models using training data. The training data usually consists of several hours of speech (acoustic data) along with corresponding transcriptions. It should be noted that, because of the existence of the transcriptions, training of speech recognizer often classified as a supervised learning task, but practically acoustic data is not segmented and so the boundary of each word or phoneme is not known. In other words, we have continues speech utterance and its transcriptions (at word level) and our goal is to train several sub-word units (e.g. phoneme models) from this data. Therefore, the training task is somewhat more difficult in practice than one might thought initially.

 After training all models, we can use these trained models to transcribe new utterances. This is the second phase of a speech recognition system and is usually can be achieved using the famous Viterbi algorithm. In many applications of HMM models and particularly in speech recognition systems, training, is by far more difficult than recognition, and in fact , at least theoretically, if we are given prefect models, recognition could be done almost perfectly using the so called Viterbi algorithm or Forward algorithm.

 In this report, we just focus on the first stage or training of HMM models. Traditionally, people have been using Baum-Welch (BW) algorithm to train HMMs. It has been shown that this algorithm is working well and is relatively fast. Baum-Welch algorithm is an iterative based algorithm and like many algorithms of this family its performance is strongly dependent on starting point. In other words, if the optimizing surface has many local minimums then Baum-Welch will find one of these local minimum instead of the global one. Generally speaking, finding the global minimum is a very difficult problem and there is no full-fledged analytical solution existed to address this problem. During recent years, people proposed many heuristic approaches, like Genetic Algorithm (GA) [1], Tabu search algorithm [2], API algorithm [3] and Particle Swarm Optimization (PSO) algorithm [4], to resolve this problem. Experiments show that these algorithms and their different combinations with classical algorithms have the capability to find the global minimum or at least find a better local minimum on the average. Recently, people start to apply some of these algorithms to train HMMs and particularly to train speech recognizer [4]. It has claimed that these approaches can improve the performance of the system when applied carefully. In this paper, we will focus on particle swarm based approach to train HMMs. The primary purpose of this paper is to provide a simple and complete tutorial for the topic. Beside this, we have tried to do a literature review for some of the topics presented here, and also we will critically examine claims regarding to PSO based training approaches results and performance. Hopefully, this paper can be used as a starting point for the next stage of the research and especially implementing the PSO training method on a state of the art speech recognizer system (ISIP prototype system [5]) and for a relatively difficult task (Wall street Journal corpse.). Only after this, we can become sure about the usability of PSO algorithm (and other similar algorithms) on real world problems.

The organization of this paper is as follow: In the next section we will introduce Hidden Markov Models and Baum-Welch algorithm. After that Particle Swarm Optimization technique would be introduced and finally the application of Particle Swarm Optimization in training HMMs and some reported results would be presented.

Hidden Markov Models

 Hidden Markov Models has been around for more than 30 years and during these years have been used extensively for different applications, but perhaps, speech recognition was the most important and most successful application of these models. In this section, we will introduce the theory behind this powerful model and Baum-Welch algorithm to train HMMs.

Markov Models

 It is a well-known fact that, sequential data (i.e. time series) could not be treated as an independent and identically distributed (i.i.d.) process [6,7], rather we have to relax i.i.d. assumption and one of the easiest way to do this is to assume a Markov model(or Markov chain).

 Relation (1) is the formal definition of an nth orders Markov chain [7].

 (1)

The event happens if the model is in state i at time t. It can be shown [7] that an nth orders Markov model can always be converted to a first order model. Figure 1 shows the graph for a first order model.



 Figure 1- A First Order Markov Model

A classic example [8], for Markov models is the weather example. Figure 2 shows a simple model for this example. The probability of a being in each state at time t is a conditional probability of being in other states in time (t-1)



Figure 2- Markov Model for weather example

From Markov Models to Hidden Markov Models

 So far we have introduced Markov Models, and we have seen that they are capable of modeling sequential data. One natural extension of this concept could be obtained by introducing hidden states in the model. The idea is to make each observation in the sequence independent of all other observations. This leads to the definition of hidden Markov models.

 **Definition 2.1. Hidden Markov model [7] “**A hidden Markov model (HMM) is collection of random variables consisting of a set of T discrete scalar variables and set of T other variables which may be either discrete or continuous (and either scalar or vector valued). These variables, collectively, possess the following conditional independence properties”:

 (2)

 (3)

Equation (2) states that the future is independent of the past given the present. One implication of above relationship is that all of the pervious observations have no effect on the current state if we know the pervious state. Equation (3) states that observation is independent of all other variables if is known.

 To understand the idea of observable sequence and hidden sates better, consider the weather example again. This time, we could not see the weather condition directly and instead of that we are told if the weather is dry, humid, soggy, etc. This example is depicted in Figure 3.



Figure 3- Hidden Markov Model for Weather example

 HMMs could be parameterized with five parameters [8]:

1-N: Number of states in the model

2-M: Number of distinct observation symbols per state (in the case of discrete output HMMs)

3-: The state transition probability distribution where is determined by:

 (4)

4-B: The observation symbol probability distributions in state j:

 (5)

Where in the discrete case, is the probability of observing symbol k when we are in state j and in continues case we just replace the mass probability function with probability density function.

5-: The initial state distribution. Where:

 (6)

Traditionally, complete parameter of the model compactly indicated by (N and M are known from A and B):

(7)

 After a bit of contemplating, we can easily see there exists 3 different problems associated with HMMs [8]:

1-The first problem is evaluation problem. Given an observation sequence and a model compute

2-The second problem is the decoding problem: Given an observation and a model what is the most likely sequence?

3-And finally, the third and most difficult problem is the learning problem: Given observation sequence and the model determine the parameters of the model.

 The simplest solution for problem 1 involves enumerating all possible state sequences of length T [8]. The probability of observation sequence O for the state sequence of is given by:

 (8)

(8) could be written in this way because of (3). In other words, each observation is independent of other states and observations given the current state. The probability of occurrence of state sequence Q can be computed by:

 (9)

The joint probability for O and Q can be obtained by multiplying (8) and (9) and finally, is computed by summing the result of this production over all possible state sequences. Unfortunately, direct computation of involves 2T\*NT computations and therefore is not feasible. To resolve this, we should use conditional independence properties intelligently, the resulting algorithm is called Forward-Backward algorithm and can be derived as follow [7, 8]:

 We start from , we can write the probability inside the summation as follow (In the following we will drop λ from all equations for the sake of simplicity.):

 (10)

The last equation could be written because of conditional independence properties. By plugging this into we obtain:

 (11)

By defining we obtain this recursion:

(12)

So we can compute

 (13)

This method just takes TN2 computations and is called Forward algorithm. Similar to this approach and by defining we can obtain following recursion which is called backward algorithm:

 (14)

And similar to (13) we have:

 (15)

It is also shown that [7]:

 (16)

 And this is completing the Forward-Backward algorithm.

 For problem 2, there exist many solutions due to different criteria for optimality. Perhaps, most widely accepted and rational criterion is to maximizing. Usually Viterbi algorithm is used to find this optimum path [8]. The basic idea behind Viterbi algorithm is to just save the best path (among all paths) terminating to each state, by doing this number of computations decreases dramatically and the algorithm can be used in real time applications. The algorithm is as follow [8]:

1-Initialization:

 (17)

(18)

2-Recursion:

 (19)

 (20)

3-Termination:

 (21)

 (22)

4-Path backtracking:

 The answer to the third problem is the subject of next subsection so we skip it here.

 It has been shown [7] that even the simplest HMM models can capture correlation between feature vectors. In fact, hidden nodes indirectly encode dependency information between temporally separated observable variables.

 One of the questions, often arise when studying HMMs is about their inherent state duration distribution. It can be shown [7] that this distribution is geometric and therefore is not consistent with many practical applications. Fortunately, this is not a really serious problem because there existed some techniques like state tying [7] (Several states use the same observation) to cure this problem.

 One important property of HMMs is that they are structurally discriminative. “Sampling from a model of one class produces an object containing attributes only that distinguish it from samples of the other class’s model. The sample will not necessarily resemble the class of objects its model represents. This, however, is of no consequence to classification accuracy” [7]. In other words, HMMs could not be used to synthesize spoken utterances because as we have mentioned HMM presents specific properties of speech utterance to distinguish it from other utterances and such a model could not necessarily produce high quality speech utterances [7].

Baum-Welch Training

 To answer to the third problem, we should first notice that this is an extremely hard problem to solve. First of all, as problem 2, the optimality criterion should be determined first. There exists many different criteria that we can select from, and they lead us to different answers. The most common, not the best possible, criterion is Maximum Likelihood (ML). Therefore, in this paper we confine our discussion to this; however, all results and discussions can be easily expended to other cases too. Secondly, the optimization surface is a very complex one with many local optimums, so almost all methods ends up by finding a local optimum answer instead of the best possible answer (relative to the selected criterion). This is a serious problem and we will return to it in subsequent sections.

 Briefly, the basic idea behind the Baum-Welch algorithm is to start from initial model (selected arbitrary) and using this model and observed sequences to re-estimate parameters of the model and use new estimation for another re-estimation step until successive models converge. The algorithm is as follow [8]:

1. Start from an initial model λ0 .
2. Compute αt (i) and βt(i) using Forward-Backward Algorithm.
3. Compute ζt(i,j) using (24) for all i and j :

 (24)

1. Computing ϒt(i) using:

(25)

1. Re-Estimation:

 (27)

 (28)

1. If model is not converged go back to 2.

To make the algorithm clearer, in the following subsection, we give a numerical example.

## Numerical Example for Baum-Welch Training

Suppose that we have some data and we want to use Baum-Welch algorithm to estimate the parameter of the model. We will assume that this data produced by the similar model and we just want to estimate the parameter of the model. In this example, we assume that model has 2 states and the observation sequence has length 2. Our corps consists of 2 strings {AA, BA} and each of these strings occurred 10 times in the corps. We start by assuming an arbitrary model. Following lines show one iterations of the Baum-Welch algorithm as applied to this example:

1. Initializing the model λ0 :

 (29)

1. Computing αt(i) and βt(i):

For {AA}:

 (30)

 (31)

Similarly for {BA} we have:

 (32)

 (33)

1. Computing

 (34)

(35)

1. Computing

For {AA}:

 (36)

 (37)

1)=0.2146

 (38)

For {BA}:

 (39)

1)=0.2146 (40)

 (41)

1. Re-estimation:

 (42)

 (43)

As this numerical example shows Baum-Welch algorithm converges toward the solution relatively fast. An interesting observation for this particular example is that after one iterations we can see the probability of emitting {A, B} is practically the same for both of the states. In other words, in the above example, we can model the data with a 1 state machine too.

Particle Swarm Optimization

As we have discussed in the last section, Baum-Welch algorithm can be used to find the optimum parameters of HMMs, however, it always find a local optimum. Because of this serious restriction, people tried to invent new algorithms capable of finding the global optimum or at least find a local optimum that performs not too much worse than the global one.

 There existed many algorithms to address this problem; none of them can guarantee a satisfactory solution in general case. Perhaps, the simplest method used from the early days of machine learning is to run many training process in parallel and then select the best among them. Particle Swarm Optimization (PSO), though is completely different from this later method, but has some common philosophy with this and as we will see, one of its common extensions can be thought as a generalization of parallel processing.

 PSO has been first proposed by James Kennedy and Russell Eberhart [9, 10], and is inspired from swarm of birds and school of fishes. Consider a swarm of n particles. We assume there is no leadership in the swarm and every particle can behave on its own. The goal of a swarm in nature is to find the best food source in the space. In that case, each particle performs a local search in the space and moves from one place to another according to its own experiences and information shared by the swarm. If the search space is bounded after some finite time the whole swarm should be concentrated around the best food source. In Particle Swarm Optimization, we have a very similar problem. In this case, the space is the D-dimensional space of parameters (parameter space), the food source is the optimum point of the function that we want to optimize, and each particle is a simple processor that can evaluate the target function. (The input to the target function is the position of the particle in the parameter space.). At the initialization step, each particle positioned randomly in the space. Particles start moving in a semi randomized manner by considering their own best position and swarms best position. In other words, we have many parallel processors with a communication channel between them and each of them is just evaluating the function at each iteration and moves its position (in other words, change the input parameters for the function to be evaluated.) by considering its own best position (cognitive information) and the swarm’s ever best position (social information).

 We can say the basic assumption behind this algorithm was the hypothesis that “social sharing of information among conspeciates offers an evolutionary advantage” [9]. Beside the above basic algorithm there are many other extensions proposed during recent years to improve the performance of the algorithm. For example, in one of these versions [10], instead of having a communication channel between all particles we have many communication channels between proximate particles. In other versions, we have all kinds of adaptability [11], hybridizations [12] and many other kinds of modifications to improve the original algorithm.

 Other than global search property of PSO, another important and in many applications vital property of PSO is the fact that it does not need to calculate the gradient for its work because it works directly on the function itself. This is very good property because many functions of interest do not have a gradient and therefore we could not perform a gradient based algorithm on them.

 So far PSO looks very promising but unfortunately as we will see there are some problems associated to this algorithm that limits its success in practice. Though there are some solutions to address these problems (all of the extensions were initially proposed to solve these limitations.) but none of them yield to a satisfactory result yet and this is an active research area.

 The standard GBEST PSO algorithm is as follow [9]:

1. Initialization: Initialize a vector of N particles with random positions and velocities on D-dimensional parameter space.
2. Evaluate the target function for each particle.
3. Compare the evaluation with particle’s pervious best value (PBEST). If current value is better (according to the fitness criterion) than PBEST then PBEST=current value and PBESTx=current position in the hyperspace.
4. Compare the evaluation with group’s pervious best position (GBEST). If current value is better than GBEST then replace it with GBEST.
5. Change the velocity with the following formula:

 (45)

In above relationship, v(k) is the velocity; ω is the inertial weight and controls the amount of recurrent in the particle’s velocity [13]. φp and φg are two constants and rp and rg are two uniform random numbers between zero and one.

1. Move particles according to:

 (46)

1. If number of maximum iterations does not reach go back to 2

As we have already mentioned, PSO has some problem that limits its performance. This problem can be seen easily in above algorithm: How should we determine the value for different parameters of the algorithm (ω, φp , φg, N,Max-itr ) ?

 It is shown that selecting these parameters greatly affects the performance of the algorithm; there is no optimum approach to select these parameters. One of the reasons, for the lack of consensus on this, is related to the fact that no one has done a complete mathematical analysis of this algorithm yet. (It is a really a difficult problem). Recently, an optimization technique (meta-optimization) has been proposed to find optimum values of these parameters for a problem or a class of problems [13]. But at this moment, people generally use experimental results to tune the algorithm for a specific problem. Because the size of problems are generally very big this tuning process is a difficult and time consuming problem itself and as we have mentioned it is a subject of active research right now.

## PSO example: training a simple Neural Network

 So far we have introduced the PSO algorithm and we claimed that this algorithm, if tuned carefully, can lead to more optimum results in compare to traditional methods. Before proceeding to the next section which dedicated to training paradigm of HMMs using PSO algorithm, it seems beneficial to prove our argument for a simpler problem. Here we have chosen a relatively simple but at the same time important problem. This problem was also addressed in the first appearance of PSO algorithm [9], but here we implement it independently.

 XOR problem is a traditional example in many machine learning text books. The problem is simply to train a neural network to perform like a XOR gate. This problem is not linearly separable and in order to solve it with need at least one layer of hidden nodes [14].

 In this example we will use a Neural Network with one hidden layer which consists of 3 sigmoid neurons. Naturally we have two inputs and one output. The output node is sigmoid too. All in all, we have 13 weights to determine. We try to minimize Mean-Square-Error (MSE). We have used 2 configurations in one of them number of particles were set to 20 and in other the number of particles are 50 and in both of them ω=1, φp=2 and φg=2 , we have also ran a standard back-propagation as our baseline system. Figure 4 shows MSE versus iteration number for various algorithms.



Figure 4- Comparing the performance of PSO and back-propagation algorithms for training XOR function

It is evident from this picture that PSO has the potential to surpass traditional gradient based method, but its performance is a subject of its parameters and so should be tuned carefully to the problem.

PSO training of HMMs

 Till now we have discussed HMMs and PSO separately. Now it is the time to merge these concepts together and see how we can use PSO technique to train HMMs.

 The basic idea behind HMM training using PSO algorithm is fairly simple. We can present each HMM using a string of real numbers. So for a typical HMM we would have a D-dimensional vector that represents it in the space of all parameters. It is in fact the approach followed by all considered publications [5, 12, 15, 16 and 17]. However, there are some problems with this approach which is not discussed clearly in these papers. One of the obvious problems is that parameters are not completely independent of each other and also they are following certain restrictions. For example, π is a probability vector so the sum of its element should be 1 and all elements should be greater than or equal to zero. One solution that looks is used in most publications is to normalize the result after each movement (step 6 in PSO algorithm) furthermore, we can restrict the search space to [0, 1] for all of these parameters. However, it not clear if this is a good solution or not. The same argument is correct for other probability vectors and matrices. In the case of using Gaussian mixtures (GMMs) we have another complication because of the constraint on the covariance matrix which requires it to be positive definite. In order to address this problem, most people just use diagonal covariance matrices and in many application this is a reasonable solution but in [18] authors provide a solution to deal with this problem too. (In the case of full covariance matrix) We will restrict our discussion to the case with just diagonal covariance matrix.

 It should also mentioned that there are different mapping of parameter space that we can implement our algorithm [19], but here we just consider the simplest one which is directly the parameter space itself.

 There are few implementation reports that we will cover in the next subsection but before proceeding to there, it would be better to mention some of the extensions proposed to improve the original algorithm. The most important extension is to hybridize PSO with Baum-Welch (BW) algorithm to obtain PSO-BW algorithm [12]. Essentially we can think of this hybrid algorithm as many parallel BWs running on the same optimization surface along with an information exchanging channel. The partial information from status of other running BWs could be used to make the effort more efficient (by moving on the optimization surface toward more promising areas.) This idea has been implemented and it is claimed to be computationally fast (relative to pure PSO algorithm.) and also can yields better results relatively to pure BW or pure PSO algorithms.

 Another extension to the original PSO algorithm described in [15]. This algorithm is named Improved Particle Swarm Optimization (IPSO) algorithm and is based on this idea: In a typical swarm, many particles become inactive after few iterations (when algorithm is still immature.) The proposed solution in [15] is to detect these particles and add Gaussian noise to the position of 25% of these inactive particles; this operation is named receptor editing. It has been claimed that this algorithm can outperform both of BW and PSO algorithms in certain applications [15].

## Reported Results

The first attempt to train HMMs for speech recognition is reported in [5]. The baseline system is isolated word recognition for English digits task. There were 100 examples for each digit in the training corps and 128 examples in the testing corps. 24 MFCC coefficients have been used. HMMS were 5 states left to right with 3 Gaussian mixtures per state. 30 particles have been generated, 3 of them have been initialized with k-mean algorithm and other randomly, and finally number of iterations has been set to 3000. Table 1 shows the average log-likelihood of each model for both of BW and PSO algorithms. Table 2 shows the error rate for these 2 different approaches. It is seen from these two tables that PSO based algorithm could get better results in expense of using more computational power. One can always argue that if we give both algorithms the same computational power then probably they do not be different at all. This is an important issue and should be investigated.

Table 1- Log-likelihood for each model [5]

|  |  |  |
| --- | --- | --- |
| Model | BW | PSO |
| 0 | -84.427 | -78.988 |
| 1 | -85.172 | -83.564 |
| 2 | -74.304 | -70.835 |
| 3 | -82.792 | -75.612 |
| 4 | -82.907 | -75.103 |
| 5 | -89.387 | -81.718 |
| 6 | -70.284 | -67.201 |
| 7 | -85.226 | -77.596 |
| 8 | -75.712 | -69.087 |
| 9 | -86.151 | -80.941 |

Table 2- Error Rate for different methods [5]

|  |  |  |
| --- | --- | --- |
|  | BW | PSO |
| Close | 1.75 | 1.43 |
| Open | 10.38 | 8.56 |

 In [12] and [17], authors have used PSOBW algorithm to train HMMs used in continues digit recognition task. In this paper, AN4 corps has been used. 39 MFCC coefficients have been used. HMMs were 3 states left to right with one Gaussian Mixture per state. Tables 3 and 4 shows evaluations on both of training and testing datasets for this experiment. We see some improvement relative to BW method. Again, we can discuss that increasing the computational power for BW algorithm might fade out the improvement gained by PSOBW algorithm.

One of the problems with both of these papers is the poor results of their baseline experiments. The performance of their baseline experiments is far from state of the art systems. One reason for this is related to the complexity of state of the art systems. In order to implement PSO or PSOBW algorithm in those system we should be very careful and we should find a practical way to merge it to the standard recipe. Another problem with both of these reports is related to selection of the parameters. None of them supplied any information on selecting parameters for the PSO algorithm.

 Finally table 5 shows the results for IPSO algorithm. In this case, the algorithm is not applied to speech data. It has been used to the problem of multiple alignments for biological sequences. If the result of this table proves to be correct then we are facing with a very interesting situation because both of the computation time and the performance show significant improvement which is a very rare occurrence in science and engineering.

Table 3- Recognition Performance on Training Dataset [17]

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sentence Correct | Word Correct | Word Accuracy |
| BW | 40.56 | 90.70 | 75.24 |
| PSO | 62.22 | 92.97 | 87.03 |

Table 4- Recognition Performance on testing Dataset [17]

|  |  |  |  |
| --- | --- | --- | --- |
|  | Sentence Correct | Word Correct | Word Accuracy |
| BW | 27.27 | 88.37 | 67.44 |
| PSO | 60.61 | 91.86 | 84.3 |

Table 5- The Benchmark Alignment Problem and Simulation Results [15]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | BW | Time(s) | IPSO | Time(s) |
| IaboA | 0.622 | 121,28 | 0.678 | 19.67 |
| 451c | 0.321 | 159.73 | 0.557 | 25.01 |
| 9rnt | 0.782 | 178.78 | 0.845 | 34.15 |
| kinase | 0.308 | 437.65 | 0.467 | 86.80 |
| 9cba | 0.653 | 408.63 | 0.794 | 91.38 |
| 1ppn | 0.605 | 383.86 | 0.714 | 64.49 |
| 2myr | 0.236 | 589.71 | 0.472 | 81.86 |
| 1eft | 0.728 | 512.34 | 0.754 | 84.93 |
| 1taq | 0.747 | 4278.93 | 0.889 | 256.13 |
| 1ubi | 0.267 | 853.42 | 0.438 | 132.10 |
| kinase | 0.186 | 2119.87 | 0.347 | 160.88 |
| lidy | 0.295 | 996.06 | 0.412 | 107.96 |

Conclusion

 In this paper, we have reviewed HMMs, Baum-Welch training, Particle Swarm Optimization technique and applicability of PSO for training HMMs. We have seen that PSO has been used successfully to train HMMs. However, the computational cost paid for this improvement is generally high and it is not clear that if we use the same computational power with traditional Baum-Welch then we could get the same result or not. At the other hand, PSO is very sensitive to its parameters and needs a very careful tuning. To our best knowledge there exists no widely accepted method to determine PSO parameters yet.

 All in all, PSO and its variations show some potential to be used in problems similar to training of HMMs. For the next stage of this research, we will start some initial experiments using this technique to train few HMMs and if we get reasonable results then we will proceed to merge this into ISIP prototype system.

# References

1. S.K. wong, C.W. Chau "Analysis of Parallel Genetic Algorithm on HMM Based Speech Recognition System", IEEE Trans. on Consumer Electronics,pp.1229-1233,1997
2. T.Y. Chen, X.D. Mei,J.S. Pan, S.H. Sun, Optimization of Hmm by the tabu search algorithm, Journal of Information Science and Engineering 20 (5) (2004):
3. S. Aupetit, N. Monmarche, M. Slimane, P. Liardet , "An Exponential Representation in the API Algorithm for Hidden Markov Models Training" , proceeding of the 7th International Conference on Artificial Evolution, Lille, France, 2005
4. http://www.isip.piconepress.com/projects/speech/software/
5. Xue, L.;   Yin, J.;   Ji, Z.;   Jiang, L, A Particle Swarm Optimization for Hidden Markov Model Training, 8th International Conference on Signal Processing, Nov 16-20 2006
6. C. Bishop,"Pattern Recognition and Machine Learning", Springer 2006
7. J. Bilmes, "What HMM can do", University of Washington Tech Report, 2002
8. L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In Proceedings o f the IEEE, Vol. 77(2), pp. 257- 285, 1989.
9. Kennedy, J. and Eberhart, R. C, “Particle swarm optimization,” Proc. IEEE int'l conf. on neural networks, vol. IV, pp. 1942-1948.
10. Eberhart, R. C. and Kennedy. J, “A new optimizer using particle swarm theory,” Proc. Sixth int'l international symposium on micro machine and human science, pp. 39-43, 1995.
11. Zhan, Z-H.; Zhang, J.; Li, Y; Chung, H.S-H. (2009). "Adaptive Particle Swarm Optimization". IEEE Transactions on Systems, Man, and Cybernetics 39 (6): 1362–1381
12. Fengqin Yang;   Changhai Zhang, An Effective Hybrid Optimization Algorithm for HMM, ICNC '08. Fourth International Conference on Natural Computation, 18-20 Oct 2008.
13. M. Pedersen , et all, " Simplifying Particle Swarm Optimization, Applied Soft Computing 2010
14. M. Minsky,S. Papert, Perceptrons: an introduction to computational geometry, MIT press, 1969
15. Hong-Wei Ge and Yan-Chun Liang, A Hidden Markov Model and Immune Particle Swarm Optimization-Based Algorithm for Multiple Sequence Alignment , AI 2005: Advances in Artificial Intelligence Lecture Notes in Computer Science, 2005, Volume 3809/2005, 756-765
16. Somnuk Phon-Amnuaisuk, Estimating HMM Parameters Using Particle Swarm Optimisation , Applications of Evolutionary Computing, 2009
17. Fengqin Yang;   Changhai Zhang;   Tieli Sun, Comparison of Particle Swarm Optimization and Genetic Algorithm for HMM training, . 19th International Conference on Pattern Recognition, 2008.
18. C.Ary, S. Aksoy, "Maximum Likelihood Estimation of Gaussian Mixture Models Using Particle Swarm Optimization," in 20th IAPR International Conference on Pattern Recognition, Istanbul, Turkey, August 23-26, 2010
19. Sébastien Aupetit, Nikolas Monmarché and Mohamed Slimane, Hidden Markov Models Training Using Population-based Metaheuristics, Advances in Metaheuristics for Hard Optimization Natural Computing Series, 2008, 415-438