STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLV**, Number 2, 2000

A METHOD FOR TRAINING INTELLIGENT AGENTS USING HIDDEN MARKOV MODELS

GABRIELA ŞERBAN

Abstract. It is well-known that, in this moment, the field of *intelligent agents* represents an important research direction in Artificial Intelligence, which offers a new method for problem solving and a new way for interaction between the computer and the user. The use of mathematical statistic-methods represents a leading topic in this field. Hidden Markov Models (HMM) are often used as a mathematical tool for modeling the environment of intelligent agents. In this paper we propose a learning method for an agent which recognize characters, based on training an Hidden Markov Model.

Keywords: Artificial Intelligence, Hidden Markov Models, learning.

1. Intelligent agents

The field of *intelligent agents* is in connection with another field of Artificial Intelligence (AI), the field of *machine learning*. *Machine learning* represents the study of system models that, based on a set of data (training data), improve their performance by experiences and by learning some specific experimental knowledge. The attempt of modeling the human reasoning leads to the concept of *intelligent reasoning*. The *reasoning* is the process of conclusion deduction; the *intelligent reasoning* is a kind of reasoning accomplished by humans. Most of the AI systems are deductive ones, able for making inferences (draw conclusions), given their initial or supplied knowledge, without being able for new knowledge acquisition or to generate new knowledge. The learning capability being connected to the intelligent behavior, one of the most important research directions in AI is to implement in the machines the learning capability.

An *agent* [3] is anything that can be viewed as **perceiving** its environment through *sensors* and **acting** upon that environment through *actions*. An *intelligent agent* is an agent with an initial knowledge, having the capability for learning. In the

2000 *Mathematics Subject Classification.* 68U05.
1998 *CR Categories and Descriptors.* I.2.6. [**Computing Methodologies**] : Artificial Intelligence – *Learning* .

followings we present how an agent can be modeled using a Hidden Markov Model, and how the agent can be trained by learning the associated HMM.

2. Hidden Markov Model (HMM)

The Hidden Markov Model (HMM) is a generalization of Markov decision processes, being possible more transitions from a state for the same input. For the same input sequence (of actions) we can have more paths in the HMM, which implies that $P(a_{1,n})$ (the probability to have as input a sequence of n actions, $a_1\,a_2\,\cdots a_n$, shortly written as $a_{1,n}$) is calculated as the sum of the probabilities on all the possible paths. Probability on a given path is calculated by multiplying the probabilities of transitions on the path.

Definition.
An HMM is a 4-tuple $< s^1, S, A, P >$, where S is a finite set of states, $s^1 \in S$ is the initial state, A is a set of input symbols (actions), and $P : SXSXA- > [0,1]$ gives the probability of moving from state s_1 to s_2 on performing action a. Let us consider the following order of the elements of the sets S, A, P: $S = (s^1, \cdots s^\sigma)$; $A = (a^1, \cdots a^\omega)$; $P = (p^1, \cdots p^\epsilon)$., where σ is the number of states, ω is the number of actions and ϵ is the number of transitions.

Let us notice that [4] a_i means the i-th element (action) of an input sequence, while the a^i represents the i-th element of the A set. A transition is defined as a 4-tuple: (s^i, s^j, a^k, p) , which means that the input action a^k in the state s^i transitions to the state s^j with the probability p. For a given input sequence of actions there are more possible paths in the HMM, so, the sequence of states that it has been passed through is not deductible from the input, but *hidden* (this gives the name of the model). The sequence of states $s_1, s_2, \cdots, s_{n+1}$ that has been passed through for an input $a_{1,n}$ is marked shortly with $s_{1,n+1}$.

2.1. **Agents and Hidden Markov Models.**

Let us consider a passive learning agent in a known environment represented as a set of *states*. At each moment, the agent executes an *action*, from a set of actions. In such a passive learning model, the environment generates transitions between states, perceived by the agent. The agent has a model of the environment using a *model of actions* (P), where $P(x'|x,a)$ represents the probability for reaching the state x' by taking the action a in the state x.

With the above considerations, the behavior of the agent in a given environment can be seen as a Markov decision process. If the state transitions are nondeterministic (a given action a in a given state x transitions to a set of *successor states*, not to a single successor state), then the Markov model is an HMM where:

- S is the set of the environment states;

- $s^1 \in S$ is the initial state for the agent;
- A is the set of the actions of the agent;
- P is the set of transitions between the states (conditioned by actions).

2.2. **Algorithm for computing the likelihood of an input sequence of actions.** In the followings, we mention a very simple algorithm for computing the likelihood of an input sequence of actions in an HMM. This algorithm, the *"forward"* algorithm [1] calculate the probability of an input sequence of actions ($a_{1,n}$) using the *"forward" probability* (α) and the *"backward" probability* (β).

The *"forward" probability* is defined [1] as the probability of being in state i after seeing the first t observations, given the input sequence.

Let us note by $\alpha_i(t+1)$ the probability of the input sequence $a_{1,t}$ having s^i as final state. In other words:

$$\alpha_i(t+1) = P(a_{1,t}, s_{t+1} = s^i), t > 0 \tag{1}$$

The idea of the algorithm is to calculate the probabilities for all the input subsequences ($a_{1,t}, t = 0, \cdots, n$) having as final state the state s^i, $i = 1, \cdots, \sigma$, where σ is the total number of states of the Markov model. Having all $\alpha_i(n+1)$ values calculated, the probability $P(a_{1,n})$ is given by:

$$P(a_{1,n}) = \sum_{i=1}^{n} \alpha_i(n+1)$$

Considering that $a_{1,0}$ is the empty sequence, which has the acceptance probability 1, we have that $\alpha_j(1) = 1$ if $j = 1$ and is 0 otherwise , corresponding to the fact that the initial state of every path is s^1.

Using the dynamic programming principle, we can make the following remark: the probability of the input sequence $a_{1,t+1}$ having s^j as final state, is obtained by summing for all state s^i, $i = 1, \cdots, \sigma$ the products between the probability of the input sequence $a_{1,t}$ having s^i as final state and the probability of the transition between the state s^i and the state s^j for the action a_t. Thus, calculation of $\alpha_j(t)$ [4] is made starting with $\alpha_j(1)$, $\alpha_j(2)$ and going until $\alpha_j(n+1)$, using the recursive relation:

$$\alpha_j(t+1) = \sum_{i=1}^{\sigma} \alpha_i(t) P(s^i \xrightarrow{a_t} s^j).$$

Recall that $\alpha_i(t)$ are called "*forward*" probabilities. Using the above considerations, let us notice that the algorithm for finding the highest-probability-paths for a given entry is based on the "*backward*" variant of the dynamic programming principle (using the backward variant of the optimality principle).

As we have mentioned above it is also possible to calculate "*backward*" probabilities, $\beta_i(t)$, with the following definition: $\beta_i(t)$ represents the acceptance-probability of the input $a_{t,n}$, if the state at step t is s^i. In other words, the "*backward*" probability $\beta_i(t)$ computes the probability of seeing the observation from time t+1 to the end, given that we are in state i at time t (given the input sequence).

So [4]:

$$\beta_i(t) = P(a_{t,n} \mid s_t = s^i), t > 1.$$

The probability we are looking for will be

$$\beta_1(1) = P(a_{1,n} \mid s_1 = s^1) = P(a_{1,n})$$

Calculation of β function is made starting with values:

$$\beta_i(n+1) = P(\epsilon \mid s_{n+1} = s^i) = 1, i = 1, \cdots, \sigma.$$

For the recursive case, we have:

$$\beta_i(t-1) = P(a_{t-1,n} \mid s_{t-1} = s^i) = \sum_{j=1}^{\sigma} P(s^i \overset{a_{t-1}}{\to} s^j)\beta_j(t)$$

2.3. **Training Hidden Markov Models.** In the followings, we use the Baum-Welch algorithm [1]("*forward-backward*") for training a Hidden Markov Model. This algorithm, that has given a certain *training* input sequence (an observation sequence $a_{1,n}$), adjusts the probabilities of transitions in the HMM, in order to maximize the probability of the observation sequence. Having an HMM structure already defined, the algorithm will let us train the transition probabilities of the HMM. In fact, we can estimate the probabilities of transitions using a very simple algorithm: for each transition (arc) t which begins in a state s, we calculate how often this arc is used when the entry sequence is $a_{1,n}$. Thus, $P(t)$ is given by

$$P(t) = \frac{how\ often\ the\ arc\ t\ is\ used}{how\ often\ an\ arc\ beginning\ from\ s\ is\ used}$$

More exactly, the probabilities of transitions are calculated with the formula [2]

$$P(s^i \overset{a^k}{\to} s^j) = \frac{C(s^i \overset{a^k}{\to} s^j)}{\sum_{l=1,m=1}^{\sigma,\omega} C(s^i \overset{a^m}{\to} s^l)} \tag{2}$$

Let us notice that the formula (2) is used only if the sum $\sum_{l=1,m=1}^{\sigma,\omega} C(s^i \overset{a^m}{\to} s^l)$ is non-zero, otherwise the probability $P(s^i \overset{a^k}{\to} s^j)$ remains unchanged.

The C function (the "$numberingfunction$") in the above formula is calculated like this [2]:

$$C(s^i \overset{a^k}{\rightarrow} s^j) = \frac{1}{P(a_{1,n})} \sum_{t=1}^{n} \alpha_i(t) P(s^i \overset{a^k}{\rightarrow} s^j) \beta_j(t+1) \tag{3}$$

Let us notice that for the calculation of $C(s^i \overset{a^k}{\rightarrow} s^j)$ we have to know the probabilities of transitions for the HMM model. The main idea of the algorithm is the following: we will start with an estimate for the probabilities, and then use these estimated probabilities to derive better and better probabilities - we calculate the new values of function $C(s^i \overset{a^k}{\rightarrow} s^j)$ using the formula (3) and finally we adjust the probabilities of transitions using the formula (2). The measure of the improvement level of probabilities after a training sequence is given by the growth of the probability ($P(a_{1,n})$) of the input sequence compared to it's previous estimation. The process of recalculating the probabilities of transitions is finished when $P(a_{1,n})$ suffers no more significant modifications (in comparison with a given approximation error).

3. Experiment

In this section our aim is to test how a system represented as an HMM (in our example an agent for recognizing characters) works.

3.1. **An agent for recognizing characters.**

Let us consider an agent for recognizing two characters "*I*" and "*U*". We assume that each character is represented by a binary matrix (for simplification, we consider that the matrix has 4 lines and 3 columns). So, the matrix corresponding to the character "*I*" is [[100][100][100][100]] and the matrix corresponding to character "*U*" is [[101][101][101][111]]. For this issue, we propose the model described in Figure 1.

Using the considerations made in subsection 2.1, the model is *hidden*, in other words is an HMM.

Of course, the structure of the Markov model chosen for the modeling of the problem, it is important.

Having as initial state the state "*a*", the above described HMM accepts the entries **100100100100** (the character "*I*") and **101101101111** (the character "*U*") (the entry for a character is obtained by juxtaposing the rows of the corresponding matrix in the following order: the first, the second, the third and the fourth line). The initial probabilities of transitions are calculated in comparison with the two entry sequences which are accepted by the HMM (the characters "*I*" and "*U*").

Let us notice that the dimension of the matrix (number of rows and columns)

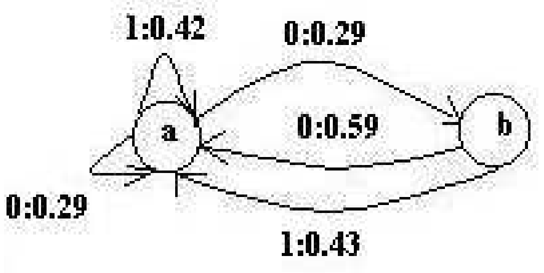

Figure 1. The Hidden Markov Model

used to represent the characters has no influence in the recognition process (only the probabilities of transitions after training the HMM change).

In this HMM, using the algorithm described in subsection 2.3, we observe that:

- the acceptance-probability for the entry **100100100100** (corresponding to the character "*I*") is $3.9190411 \cdot 10^{-4}$;
- the acceptance-probability for the entry **101101101111** (corresponding to the character "*U*") is $8.2214139 \cdot 10^{-5}$;
- the acceptance-probability for the entry **100100100111** (corresponding to the character "*L*") is $2.07292009 \cdot 10^{-4}$.

3.2. **First training.** First, we train the HMM to recognize the character "*I*" (we use the training algorithm described in subsection 2.4 for the entry sequence **100100100100**).
Considering the approximation error 10^{-7}, the HMM is trained in 13 steps. The probabilities of transitions during the training are described in Table 1 (the columns correspond to the probabilities of transitions).

In the HMM trained to recognize the character "*I*", we observe that:

- the acceptance-probability for the entry **100100100100** (corresponding to the character "*I*") is $3.906248 \cdot 10^{-3}$;
- the acceptance-probability for the entry **101101101111** (corresponding to the character "*U*") is $7.017882 \cdot 10^{-25}$;
- the acceptance-probability for the entry **100100100111** (corresponding to the character "*L*") is $1.953123 \cdot 10^{-3}$.

3.3. **Second training.** The second training of the HMM is for recognizing the character "*U*" (we use the training algorithm for entry sequence **101101101111**).

Considering the approximation error 10^{-7}, the HMM is trained in 26 steps, described in Table 2.

TABLE 1. The probabilities of transitions during the first training process

Step	P(a, 1, a)	P(a, 0, a)	P(a, 0, b)	P(b, 1, a)	P(b, 0, a)
1	0.34895903418	0.32739092305	0.32365004277	0.28061623839	0.71938376161
2	0.36807390057	0.30667030898	0.32525579045	0.21734816939	0.78265183061
3	0.39085481661	0.26963514927	0.33951003412	0.15079989110	0.84920010890
4	0.41877605321	0.21221865027	0.36900529651	0.08661076916	0.91338923084
5	0.45055613731	0.13549261529	0.41395124739	0.03595643299	0.96404356701
6	0.47879919631	0.05899019566	0.46221060803	0.00857366994	0.99142633006
7	0.49432319346	0.01463521470	0.49104159184	0.00080952252	0.99919047748
8	0.49898253274	0.00224291095	0.49877455631	0.00001805465	0.99998194535
9	0.49985762067	0.00028976704	0.49985261230	0.00000006090	0.99999993910
10	0.49998185346	0.00003637704	0.49998176950	0.00000000003	0.99999999997
11	0.49999772587	0.00000454959	0.49999772454	0.00000000000	1.00000000000
12	0.49999971564	0.00000056874	0.49999971562	0.00000000000	1.00000000000
13	0.49999996445	0.00000007109	0.49999996445	0.00000000000	1.00000000000

In the HMM trained to recognize the character "*U*", we observe that:

- the acceptance-probability for the entry **100100100100** (corresponding to the character "*I*") is $8.111088 \cdot 10^{-22}$;
- the acceptance-probability for the entry **101101101111** (corresponding to the character "*U*") is $3.251364 \cdot 10^{-3}$;
- the acceptance-probability for the entry **100100100111** (corresponding to the character "*L*") is $6.103893 \cdot 10^{-17}$.

After the agent was trained for recognizing the characters "*I*" and "*U*", the agent receives an entry, for example **100100100111** (the character "*L*"), which he tries to recognize. The recognition performs the following steps:

- first, the agent computes the probability $p1$ for the given entry in the first environment (trained for "*I*");
- second, the agent computes the probability $p2$ for the given entry in the second environment (trained for "*U*");
- third, the agent compares $p1$ and $p2$ and determines the maximum;
- fourth, because $p1$ is greater than $p2$ the agent recognize the character "*I*" as the most probable for the given entry.

This is a kind of supervised learning, the agent is trained for a few models, and after the training he tries to recognize a given entry. We chose this experiment with two characters because it is simple and illustrates very clearly the idea of training the agent using the training of the HMM.

Table 2. The probabilities of transitions during the second training process

Step	P(a, 1, a)	P(a, 0 , a)	P(a, 0, b)	P(b, 1, a)	P(b, 0, a)
1	0.71380471380	0.14141414141	0.14478114478	1.000	0.00
2	0.70681329384	0.12043988151	0.17274682466	1.000	0.00
3	0.69971006779	0.09913020337	0.20115972884	1.000	0.00
4	0.69291260238	0.07873780715	0.22834959047	1.000	0.00
5	0.68680005870	0.06040017610	0.25279976520	1.000	0.00
6	0.68162626416	0.04487879249	0.27349494335	1.000	0.00
7	0.67748162778	0.03244488333	0.29007348889	1.000	0.00
8	0.67431367967	0.02294103901	0.30274528131	1.000	0.00
9	0.67198200820	0.01594602459	0.31207196721	1.000	0.00
10	0.67031480817	0.01094442451	0.31874076732	1.000	0.00
11	0.66914788095	0.00744364286	0.32340847618	1.000	0.00
12	0.66834348425	0.00503045274	0.32662606302	1.000	0.00
13	0.66779488597	0.00338465792	0.32882045610	1.000	0.00
14	0.66742348964	0.00227046892	0.33030604144	1.000	0.00
15	0.66717331840	0.00151995519	0.33130672641	1.000	0.00
16	0.66700537657	0.00101612971	0.33197849371	1.000	0.00
17	0.66689289418	0.00067868255	0.33242842326	1.000	0.00
18	0.66681767274	0.00045301824	0.33272930902	1.000	0.00
19	0.66676742102	0.00030226306	0.33293031592	1.000	0.00
20	0.66673387346	0.00020162039	0.33306450615	1.000	0.00
21	0.66671148776	0.00013446328	0.33315404895	1.000	0.00
22	0.66669655476	0.00008966429	0.33321378095	1.000	0.00
23	0.66668659534	0.00005978602	0.33325361864	1.000	0.00
24	0.66667995391	0.00003986172	0.33328018438	1.000	0.00
25	0.66667552547	0.00002657642	0.33329789811	1.000	0.00
26	0.66667257283	0.00001771848	0.33330970870	1.000	0.00

Also, we considered an experiment with four characters: "*A*", "*L*", "*U*", "*I*". The associated HMM has four states and the recognition process works well.

On our opinion, it would be interesting the combination of the above described method of training with certain dynamic programming methods.

4. The application

The application is written in Microsoft Visual C++ 5.0 and implements the algorithms described in the previous sections.

Examples

1. For the sequence *100100100111* the application displays the following results:

```
The entry is
*
*
*
* * *
The maximum probability for the entry is 0.003906
The character recognized for the given entry is:
*
*
*
*
```

2. For the sequence *000101111101* the application displays the following results:

```
The entry is

*   *
* * *
*   *
```

The maximum probability for the entry *000101111101* **is** $2.2966411 \cdot 10^{-22}$

```
The character recognized for the given entry is:
*   *
*   *
*   *
* * *
```

5. Conclusions

In certain situations, the behavior of an intelligent agent can be modeled using an HMM. In such situations, it would be interesting to use mathematical methods for working on these models.

In the case of the proposed experiment, some future research would be:

- how could the proposed model be generalized for as many characters as possible;
- how could the structure of the HMM be generated dynamically;
- how would such probabilistic methods be more appropriate than others;
- how could the probabilistic methods be combined with others (that may also be heuristic) for obtaining a higher performance of the model;

- what would happen in certain "plateau" situations (where a given entry would have the same probability in more environments).

Anyway, which we wanted to emphasize in this paper is another way of working on problems of training intelligent agents.

References

[1] D.Jurafsky, James H. Martin : "Speech and language processing", Prentice Hall., 2000.
[2] E. Charniak: "Statistical language learning", MIT Press, 1996.
[3] S.J.Russell, P.Norvig: "Artificial intelligence. A modern approach", Prentice-Hall International,1995.
[4] D.Tatar, G.Serban: "Training probabilistic context-free grammars as hidden Markov models", Studia Universitatis "Babes-Bolyai", Series Informatica XLV (2), 2000, 69–78.

"Babeş-Bolyai" University, Cluj-Napoca, Romania
E-mail address: gabis@cs.ubbcluj.ro