Abstract

Deterministic finite state (DFA) automata have emerged as an effective tool for agent modeling applications. The problem of automata learning is to determine a DFA from a series of observation and has recently been studied extensively and a number of algorithms has been proposed. These algorithms can be divided into groups: supervised and unsupervised. In supervised algorithms, we have access to a teacher who give a right answer to our queries, whereas in unsupervised algorithms, the instead of asking questions from teacher, the algorithm rely on latest generated model which changes during its execution. The idea of unsupervised learning an automata through a series of examples have advantageous over supervised version in two important counts: no teacher is needed to response queries and having final model is not necessary. US-L* is an unsupervised method of learning automata which was introduced by D. Carmel et al [1] and is based on L*. To improve the speed of algorithm we modified this method.

The proposed method was tested using a series of automaton and randomly generated set of examples and the results are presented.

Introduction

On of key issues in the field of robotic is modeling of other cooperative or adversary agents in the environment via a series of interactions or past experiences. A wide variety of algorithms exist and among them model based approach recently has recently been found widespread. In model based technique we assume that a primarily model of the target agent is available. This model is built during the specification phase or a predefined model from previous execution of the algorithm is employed. A major issue with this assumption is that in many cases no initial model of the system is available or it is inapplicable. For example in the Robo Soccer competition where two teams of robots play soccer against each other, using a predefined model which has been built during last competition or applying unique model to all opponents might be ineffective because it is not obvious if it matches to its latest update or matches all opponent.

To deal with this problem we have to exploit the approaches in which creating and growing a model is automatic and do not need any assistance. Using these techniques we can start from one initial model or the latest version of the system and later the agent can automatically update the model of the system using further observation.

Modeling of an agent using an automaton, a model based approach in some cases has been shown to be more appropriate.

In unsupervised learning automata at each step the agent take advantage of the latest approximated model to predict the next movement of the agent. Then we compare the real output with the predicted output to see the conformance of the current model with real one. It some algorithms the evolution of the model is due to positive examples (supporting examples) while in the other it is based on the negative examples (counterexample).

Angluin[2] introduced a supervised method of learning automata called as L* to build the smallest automaton from scratch in presence of actual model, a finite set of examples coming from a regular language and a teacher.

The teacher also tells automaton whether or not examples are member of the target language and also it always compares the actual and built automaton. Based on this method Carmel et al [11] proposed an unsupervised method in which
presence of teacher and target model is not necessary. To improve the speed of the algorithm we changed the original US-L*.

The objective of this work is to examine the efficiency of modified version of US-L* for inferring a finite state automata and potential bottlenecks in applying it.

**DFA**

A deterministic DFA is a modeling device that can be used to mimic the performance of a system. It consists of a set of elements called states which interconnect by arrows named as transitions in such a way it generates the required results. At t=t₀ it is in one of its initial states and under some pre specified conditioned it traverse to another state. These conditions can be viewed as a finite sequence of inputs named as events which can generate a language. It has been used in numerous of applications such as NLP, communication, control, robotic, diagnosing, forecasting, image processing , pattern recognition and etc.

Briefly a Deterministic Finite state Automata (DFA) is formally defined as a six tuple \((Q, \Sigma, \delta, q₀, F, Y)\) where :

- \(Q\) is a none-empty finite set of states
- \(\Sigma\) is a nonempty set of symbols
- \(q₀\) is an initial state
- \(\delta:Q \times \Sigma \rightarrow Q\) is a stat transition function
- \(F\) set of accepting states

A DFA can be seen as a directed graph. States and transient function in a DFA resemble nodes and arcs that connect nodes . In addition to that DFA has input and in some cases output.

**The Problem**

The general problem can be expressed as : given a finite set of examples in the form \((s,y)\) where \(s\in\Sigma^*\) and \(y\in\{0,1\}\)find an automaton which is consistent with the example set. Set of examples contain supporting examples and counterexamples. If the state reached by the DFA after reading a string \(s\) in \(\Sigma^*\) is in \(F\) (the string is accepted ) and \(y=1\) or it is not in \(F\) (the string is rejected)and \(y=1\). It is similar to Moore machine in which output of the automaton has only one of two states: 1 or 0.In the other word the language \(L\) generated by DFA should be consistent with set of examples. Another Important issue is the generalization power of DFA. Therefore challenge is to find the smallest DFA that is consistent with the training set and can predict the unseen data. These two goals are generally mutually compatible.

But what make the automaton different from other agent modeling algorithm is the predictive accuracy and also the compactness of the generated model. The problem can be simplified if the algorithm has the possibility of asking query and access to main model.

**Related works**

DFA learning seems to be a recent development. There are many ways to learn a DFA. In some quality and quantity of approaches completely depends on the set of finite examples which it receives. In some methods the learner needs the positive evidences; in others they need negative evidences. In general, these algorithms can be divided into two main groups: passive algorithm and active algorithm. In passive approaches the learner has no control over the data which it receives while in active learning the learner can obtain appropriate examples.


Pollack, 1991[7]; Watrous and Kuhn[8], Giles, Sun, Chen, Lee and Chen[9],Giles, Miller, Chen, Sun, Chen and Lee[9], 1992 used different arrangement of Neural Network to infer an automata from a series of example.

Evolutionary algorithm has been used by some researcher such as Zhou & Grefenstette [11] , Wyard [12], Lucas [13] and Dupont [14],[15] to learn a regular and context free language from a set of samples.

Schwehm & Ost [16], Kammyeyer & Belew [17], and Keller & Lutz [18] and [19], Luke et al. [20], and Lucas and Reynolds [21] applied GA to learn an automaton and described results concerning the inference of stochastic grammars with genetic algorithms.

**L***

Angluin introduced L algorithm for inferring an automaton from a given finite set of samples in his seminal paper “Learning regular sets from queries and counterexamples”. At any stage the learner collect the information in a table called as observation table \((S,E,T)\). S and E are finite sets
of strings and T is a finite function: T: (S U ΣΣ)E \rightarrow {0,1}. Observation table is a two dimensional table with one row for each element of S U ΣΣ, where ΣΣ = {σσs ∈ S, σσ ∈ ΣΣin}, and one column for each element of E. Table can be divided into two parts: top part and bottom part. The top part of the table corresponds to the states or member of S and bottom part corresponds to Concatenation of the states with events set or ΣΣ. In each step we maintain a table with rows corresponding to states and columns corresponding to experiments. States in this table can be interpreted as a set of input string which changes the state from starting state to it. So when two rows in top part of the table are equal then they can be considered as same states but if the rows are different then they are different states. Using these experiments we can determine different states. Members of table are the answer of teacher to membership query of concatenation of corresponding row and column. The table entries, T (s, e) ∈ ΣΣout={0,1}. It is 1 when the target model accepts input string otherwise 0.

The computational time is polynomial in the number of states of the machine and the longest counterexample supplied by the teacher[130].

In L* the learner receives more information via membership queries. The learners can ask two type of queries :
1-Membership query: this string can be accepted by target automaton?
2-Equivalence query: Is the candidate automaton is equal to target automata?

In L*, we assume that the teacher is able to reset the target automaton before answering any membership queries. At first this might be considered trivial presumption but without this precondition learning become complicate.

**Some Definitions:**

A table is closed if for every row in the bottom part of the table, there is an equal row in top part. In the another word a table is closed iff for any s ∈ ΣΣ there is a a string s’ ∈ CS such that row(s) = row(s’).

A table is consistent if for every pair of strings in the top part of the table with equal rows, their successors must have identical rows as well. Therefore, in a consistent table, for any pair of states that are the equal are still equal if we concatenate every experiment string with an event set member. Briefly we can call a table consistent iff for any two strings, x1, x2 ∈ S such that row(s1) = row(s2), and for any σσ ∈ ΣΣin, row(s1σσ) = row(s2σσ).

The learner starts with an initial model and then using examples it develops the current model in such a way that the examples to be covered by the model. Then it checks whether the table is close and consistent. If it is not the algorithm has to take some measures. After solving closeness and consistency it compares the current model with real automata and if they are not identical, the teacher provides another counterexample.

**Complexity and Correctness of L**

**Theorem 1**: L* is a polynomial-time algorithm for inferring the exact target DFA using membership queries and equivalence queries.

**Theorem2** : If (S, E, T ) is a closed and consistent observation table ,then the DFA M(S, E, T ) is consistent with the table T , and any other DFA consistent with T but not equivalent to M(S, E, T ), must have more states.

**Theorem3 (Gold)**. The problem of finding an optimal assignment for an observation table that covers a given sample is NP-hard.

**Theorem4 (Angluin)**. L* eventually terminates and outputs a minimal DFA equivalent to the Teacher’s DFA. Moreover, if n is the number of states of the teacher’s DFA, and m is an upper bound on the length of the longest counterexample provided by the teacher, then the...
Theorem 5. For any DFA M there exists a DFA \( M_{opt} \) such that \(|M_{opt}| = |M|\) and \( M_{opt} \) is optimal in respect to \( M \).

A summary of \( L^* \) algorithm is shown in table 1. Fig 1 shows a target DFA and also trace of execution of \( L^* \).

**US-L**

Although \( L^* \) seems to be an effective method to learn an automaton, there are applications in which applying original \( L^* \) is impossible. The major problem is due to unavailability of target automaton. Therefore the membership queries and equivalence queries can not be responded. One way to solve this problem is to use the latest table as a reference. We build an automaton based on this table, consider it as a target model and taking the advantage of this model membership queries can be answered. Actually this model which is consistent with past examples play the role of target model and we exploit it to answer our membership queries. Of course we do not get a correct answer for some input strings. It is quite clear that we are not able to answer the equivalence queries because the target model is not attainable so what we do is that we keep updating our model. These are major differences between original \( L^* \) and what we call unsupervised \( L^* \).

David Carmel and Shaul Markovitch [1] have introduced “Unsupervised \( L^*\)” US-L* algorithm

Which is a modified version of \( L^* \). In US-L* we maintain the observation table as in \( L^* \) but in addition to two differences which was mentioned above, there are some other alternation. The examples are divided to two groups: Supporting examples and counter examples. Counterexamples influence the model more than supporting examples. The supporting examples...
s’ do not change the table except it marks any s.σ ∈ T , s ∈ S U Σ, e ∈ E such that s’ = s.σ , as a permanent member. Any member of table have an attribute named as membership permanency. Any member of table is a permanent member, changed or not changed at all. If it is a not changed member, its attribute can be converted to changed or permanent but if it is changed we can not change it again. If the new example is a counter example then the table should be changed such that the new table be consistent with the new examples and its extensions. After extending the tables we check the table to be closed and consistent same as what is done in L* with some changes. Resolving a none closed table is same as L* but in the case of an inconsistent table it is different.

Correctness of US-L∗

Theorem 5. If D is a set of examples of the machine’s behavior, M is a DFA consistent with D, and t is a new example. Then US-L∗(D, M, t) eventually terminates and outputs a model consistent with D U {t}. Moreover, if k is the size of the set of all prefixes of the examples in D U {t}, then the total running time, and the size of the observation table used by US-L∗, are bounded by a polynomial in k and |M|.

Lemma 6. The consistency loop terminates after at most k²(1 + |Σin|) + k iterations and outputs a consistent table.

Consistency:

While not Consistent(S, E, T)

find two equal rows s₁, s₂ ∈ S, σ ∈ Σin, e ∈ E, such that T(s₁σ, e) ≠ T(s₂σ, e)

if both (s₁σ, e) and (s₂σ, e) are permanent or both have been changed before (we must distinguish between rows s₁ and s₂)

if s₁ or s₂ are not member of D or not changed before

Change row(s₂, e) if the length s₂ > length s₁

otherwise Change row(s₁, e)

else

E ← E \{σe\}

for each s ∈ S U Σ, T(s, σe) ← Query(s, σe)

elseif one entry is a hole which was not changed before (assume (s₂σ, e)) or both entries are holes which were not changed before and assume s₁ ≤ s₂

T(s₂σ, e) (and its tied entries) ← T(s₁σ, e)

mark (s₂σ, e) (and its tied entries) as changed

Fig 6: Modified US-L∗. The only change is in Consistency part of the algorithm.

Lemma 7. The closeness loop at the end of the algorithm, terminates after at most 2k⁶(1 + |Σin|) + |M| iterations, does not change the consistency of the table, and outputs a closed table.

A summary of US-L∗ algorithm is shown in table 2. Fig 1&3 shows a target DFA and also trace of execution of L∗.

Modified US-L∗

To improve the speed and ability of original US-L∗ we changed the resolving consistency step and it is shown in Fig 6.

Automata Learning and Opponent modeling

Automata learning algorithm can be shown to be useful for agent modeling problems. Although finding an automata that generates a language are
in some sense similar to modeling an agent in an adversary environment but still there are some differences. In both problems the goal is to discover an automaton from a series of examples and we know about the possible events.

In both problems the goal is to discover an automaton from a series of examples and we know about the possible events. In agent modeling we can have initial information about the number of states and also state constraint(s) while in learning a language we have no information about those items. In agent modeling the main concerns are finding the structure of model, inter state connection and switching condition between states but not in automata learning. In agent modeling the latest model can be exploited to predict the further situation of the system while in language learning until reaching to the target automata it can not be used.

**Experiment**

we performed a series of experiments to test the performance of our approach. For that we consider a set of 57 different automata with two events. It consists of 9 automata with two states, 20 automata with three states, 12 automata with four states and 16 automata with five states. Then we generated a series of random examples with random length and feed them into system. The initial model was same as Fig 1. These examples were considered as positive or negative examples with respect to latest discovered model. If it is a counterexample then system went through a series of evolution according to the modified models and if it is a supporting example it keeps the latest model. The training time for the larger automata was very slow comparing to automata with fewer states perhaps due to closeness of the initial model to target automata.

The results obtained are represented in Figures 7 and 8. Figure 7 depicts the average size of built model versus automata size and Figure 8 shows the number of counterexamples needed as a function of automata size. As it can be seen to learn a larger automata we need more counterexamples. For automaton with 5 states we see a slightly decrease in the number of counterexamples. The reason is that since the manual evaluation of large automata is difficult we only consider the ones with same structure or the ones with less than 8 states. Also the size of the model is increases as the size of the automaton is also increases.

**Conclusion and future work**

The main contribution of this work is to find a method to learn an automata which is an introduction to the agent modeling step in opponent modeling. The proposed technique is an initial attempt to model an agent using automata. Although modified method has a better performances over standard US-L*, it has has same complexity as US-L*. The running time is still very slow.

Future work may include a more thorough examination of some alternatives for the algorithm, as well as a further study of other methods.
1-L* and US-L* algorithms are well known to be NP-hard. It becomes more important when we need to learn automata with higher number of states. One possibility is to start with a good initial model. According to our experiences when the initial model is close enough to target mode less step is needed.

2-Quite often we confront with the situations in which the number of states suddenly growth after injecting a counterexample. It makes a lot of troubles. First of all the cost of calculation exponentially increases afterward. Second it might be far from original automaton .To come up with this problem it is recommended to minimize the number of automata states at those situations.

3-One of major problems of this method is its inability to use supporting examples. In some methods such as evolutionary algorithms they use the supporting example. Changing the method such that it can use both kind of examples can reduce number of steps needed to learn the automata.

4- The application of the automata learning to agent modeling domains is an interesting open problem that can be considered in future work.

References:


