The Thirty-Second International Florida
Artificial Intelligence Research Society Conference (FLAIRS-32)

Learning Behavioral Memory Representations from Observation

Josiah Wong, Avelino J. Gonzalez
Department of Computer Science, Univ. of Central Florida — Orlando, FL, USA
{josiah.w.ucf@knights.ucf.edu, gonzalez@ucf.edu}

Abstract

Learning from Observation (LfO) is highly useful for modeling behaviors through nonintrusive observation of some actor's performance. However, an actor's performance is often influenced by unobservable internal influences, such as emotions, agendas, and memory of past events. Therefore, new techniques are needed to infer the structure of these influences and their effect on an actor's decisions. In this paper, we propose a novel approach called Memory Composition Learning (MCL) for capturing one internal influence: memory of past events. We hypothesize that memory influences on a behavior can be modeled through parameterized memory features that can be learned from observation of traces of an actor's behavior; these memory features can then be presented as additional input to a performance modeling application. We demonstrate the efficacy of our approach in a simulated vacuum cleaner domain and show that hidden memory influences can be detected, modeled, and then used to improve machine learning performance.

Learning from Observation (LfO) is how we learn from others on a daily basis. We often learn how to perform a task or exhibit a behavior simply by observing someone else doing it. The same is true for computer systems that learn how to perform complex tasks, such as steering a car (Pomerleau 1989) or playing air hockey (Bentivegna and Atkeson 2001). LfO is an advantageous machine learning paradigm because LfO transfers the burden of converting raw observations into machine readable procedural knowledge from the developer to the learning system itself (Floyd 2013); this allows LfO to learn the nuances of human behavior that are implicit or difficult to articulate (Sidani and Gonzalez 2000).

However, the biggest limitation of LfO is that it is currently impossible for an external observer to directly see what a person is thinking as they act. A person's behavior may be affected by various internal influences, such as emotions, goals, and memory of past events. The importance of these influences necessitates the development of techniques in LfO that can approximate such influences. Fortunately, certain internal influences can be inferred by analyzing patterns in observed behavior that coincide with likely stimulants of internal processing.

In this paper, we specifically address the problem of modeling how an observed entity appears to use *memory* of past events in their decisions. It is difficult to determine specifically what an actor chooses to remember from the past, but it is feasible in some cases to capture aspects of memory that are highly correlated with certain decisions by the actor. By making this information from memory available to a machine learning algorithm, it becomes possible to leverage influences from memory in modeling observed actions.

In this paper, we introduce a new technique, *Memory Composition Learning* (MCL), a Case-Based Reasoning (CBR) approach that learns a set of *memory features* that capture the aspects of memory that are most influential in an observed behavior. We demonstrate in a simulated vacuum cleaner domain that these learned memory features improve machine learning performance in tasks that involve memory and that the memory features reveal the influences of memory that affect the observed behaviors in this domain.

This paper is organized as follows. First, we provide background on LfO and how memory plays a crucial role in the learning task. Next, we describe our MCL technique and how it learns the appropriate memory features that describe memory influences on observed behavior. Then, we present the results of our approach in a simulated vacuum cleaner domain, followed by related work and conclusions.

Learning from Observation

Learning from Observation (LfO) is defined by (Fernlund et al. 2006) as:

> The agent shall adopt the behavior of the observed entity solely from interpretation of data collected by means of observation.

Here, the observed behavior need not be optimal for achieving the desired task, and so the observed actor need not necessarily be an "expert". This is advantageous for modeling human-like behaviors that are not by-the-book and which reflect inherent human sub-optimality (Fernlund et al. 2006).

LfO is a subfield of machine learning that learns correct behavior from *traces* or records of behavior to emulate. It is most similar to standard supervised learning, except:

1. Each decision in the LfO task cannot be handled independently of all prior decisions because decisions are temporally-based. Decisions cannot be assumed

to be identically and independently distributed (i.i.d.) (Ontañón, Montaña, and Gonzalez 2011).

2. The objective of LfO is to capture the probability distribution of a stochastic process, not to minimize the prediction error (Ontañón, Montaña, and Gonzalez 2014).

There are three levels of behaviors that can be learned by LfO, described in (Ontañón, Montaña, and Gonzalez 2014):

1. *Strict Imitation*: This behavior is strictly a function of time. A learning agent is expected to learn to perform the observed behavior exactly.
2. *Reactive Behavior*: This behavior depends solely on the current environmental perception at a given point in time. The learning agent is expected to generalize its learned behavior to novel situations not observed during training.
3. *Memory-based Behavior*: This behavior depends on the current perception and on memory of all past events. Decisions cannot be made independent of prior decisions.

Standard supervised learning algorithms are unable to learn the general class of Level 3 memory-based behaviors (Ontañón, Montaña, and Gonzalez 2014) because it is necessary to map entire *runs* up to a given point to the appropriate action instead of just mapping the current perception to the right action. In other words, memory-based behaviors do not conform to the *Markov assumption*, which states that all information that is needed for a decision exists within the current state. However, it is possible to an extent to capture the influences of memory in an auxiliary feature set that can be added into the current state, largely restoring the Markov assumption for machine learning algorithms that depend on it. We discuss our technique for doing this next.

Approach

Memory Composition Learning (MCL) is our novel approach for modeling the memory influences that affect an observed actor's decisions. It does this by learning *memory features* (MFs) that capture salient aspects of memory that are highly correlated with an actor's decisions. First, we describe the MFs that are learned by MCL. Next, we describe the overall MCL algorithm. Then, we describe in more detail each stage in our MCL approach.

Memory Features

Perception aliasing is the phenomenon where a single perception can correspond to two different situations; because of this, an entity can appear to have many different responses to the same perception (Faltersack et al. 2011). To compensate for the insufficiency of a single perception to account for differences in behavior, it becomes necessary to delve back into memory of the past to find the true causes for an actor's decisions. Such causes may be a specific *value* of a feature at a fixed time before the present moment or they may be a function of *how much time* has passed since the last observation of a given value of a feature. We can capture these aspects of memory with *memory features* (MFs).

MFs are parameterized features that store some past value of a feature (even an action) or a temporal relation between a past observed value and the present. There are two types of MFs that can be learned by MCL:

- *Value-back MF*: At a given time step t, store the value α of a feature f at time $t-k$ for some constant k. We represent this MF as $f(k)^V$ and its value at time t as $f(k)^V_t = \alpha$.
- *Time-back MF*: At a given time step t, store the number of time steps β before t when a value v for a feature f was last observed. We represent this MF as $f(v)^T$ and its value at time t as $f(v)^T_t = \beta$, where $\beta > 0$. ($\beta \neq 0$ so that the memory of the past and its effects are distinct from the perception of the present and its effects.)

The combination of instances of these MF types comprises the set of memory influences that affect an observed behavior. To illustrate how these MFs work, suppose we have a trace of some observed behavior over five time steps, which consists of two binary features, A and B (see Table 1).

Table 1: Example Trace

t	0	1	2	3	4
A	+	-	-	+	+
B	-	+	-	+	-

Just as an example, suppose we add two MFs to our trace (see Table 2):

- $A(2)^V$: a value-back MF that stores the value of feature A at time $t-2$ for $t \geq 2$.
- $B(+)^T$: a time-back MF that stores the number of time steps before time t (for $t \geq 1$) since the last time the value of feature B was '+'.

Table 2: Example Trace with Memory Features

t	0	1	2	3	4
A	+	-	-	+	+
$A(2)^V$			+	-	-
B	-	+	-	+	-
$B(+)^T$			1	2	1

In Table 2, we see that the value-back MF $A(2)^V$ at time $t = 2$ stores the value of A at time $(t-2) = 0$, which is '+'. This MF stores no value at time $t = 0$ or $t = 1$ because at those times, $(t-2) < 0$. At times $t = 3$ and $t = 4$, $A(2)^V$ stores the value of A at times $(t-2) = 1$ and $(t-2) = 2$, respectively.

The first time $B =$ '+' is at time $t = 1$. Therefore, the time-back MF $B(+)^T$ stores nothing at times $t = 0$ and $t = 1$ because there is no time *before* time $t = 1$ when $B =$ '+'. At times $t = 2$ and $t = 3$, the number of time steps before t since the last time $B =$ '+' (time $t = 1$) are 1 and 2, respectively. At time $t = 4$, $B(+)^T = 1$ because $B =$ '+' at time $t = 3$, which is one time step before time $t = 4$.

We now proceed to discuss how MCL automatically learns the most appropriate MFs for modeling memory influences on an observed behavior.

Memory Composition Learning

MCL is a trace analysis technique that learns which MFs capture the most salient influences of memory in an observed behavior. The learned MFs are then added to the original traces to produce *memory-enhanced* traces. The MCL task can be described as:

Input: A set of traces of an observed behavior.
Output: A set of memory-enhanced traces that incorporate a set of learned MFs that describe the memory influences on the observed behavior.

The memory-enhanced traces contain the time-stamped features found in the original traces plus the set of learned MFs reflecting memory influences. The MFs can be treated in the same way as the original features by a machine learning algorithm that uses the memory-enhanced traces.

MCL is a case-based reasoning (CBR) approach to learning MFs from traces. Algorithm 1 reflects the major operations in MCL. These are:

1. Create a set of cases from the given set of traces. Each case consists of a subset of a trace up to a given time step (the problem component) and the action the observed actor took at that time in the trace (the solution component).
2. Perform *MF Extraction* for each case in the case base in the procedure `extract` to get a list of MFs that make a given case's run conform to the Markov assumption.
3. Perform *MF Refinement* in the procedure `refine` to reduce the set of extracted MFs into a minimal refined subset of MFs that appear often in the case base and traces.
4. Perform *Trace Enhancement* for each trace in the procedure `enhance` by adding the refined MFs to the trace, which creates a *memory-enhanced* trace.

The set of memory-enhanced traces are the final output of MCL. We now discuss the MF Extraction, MF Refinement, and Trace Enhancement operations in greater detail.

Algorithm 1 Memory Composition Learning

```
1:  procedure MEM-COMP-LEARN(traces)
2:      casebase ← create-casebase(traces)
3:      mf-lists ← (empty set)
4:      for each case in casebase do
5:          tmp-mfs ← extract(case, casebase)
6:          mf-lists.add(tmp-mfs)
7:      refined-mfs ← refine(traces, casebase, mf-lists)
8:      mem-traces ← (empty set)
9:      for each trace in traces do
10:         tmp-trace ← enhance(trace, refined-mfs)
11:         mem-traces.add(tmp-trace)
12:     return mem-traces
```

Memory Feature Extraction

MF Extraction is the process of learning which MFs would capture the necessary memory influences needed to predict the solution component of a given case. This means that the MFs must be useful in distinguishing between cases with a similar solution and those with a different solution.

MF Extraction relies on Temporal Backtracking (TB) (Floyd 2013), [1] which is a case-based retrieval algorithm that compares entire runs, element-by-element, in reverse chronological order. As an example, suppose we have the case base in Table 3 encoding traces with a single binary feature F. TB will determine the appropriate next action for the *test case* by comparing its elements to corresponding elements of the cases at time step t, then $t-1$, then $t-2$, etc., where the value of t for a given case is the *last* time step for that case. Cases 1-5 in the case base prescribe action α or β, as shown in the table. In our example, Cases 1, 2, and 3 are discarded as dissimilar to the test case in TB iterations 1, 2, and 3, respectively. At Iteration 3, the solution components of the remaining cases (Cases 4 and 5) agree, so there is no need to compare elements for earlier time steps; TB will return the solution agreed upon by the remaining cases, action α, after three iterations.

Table 3: Temporal Backtracking Example

Iter	Time	Cases						Solns
		Test	1	2	3	4	5	
		?	α	β	β	α	α	
1	t	-	+	-	-	-	-	α, β
2	$t-1$	-		+	-	-	-	α, β
3	$t-2$	-			+	-	-	α
	...	...				...	...	

Our research goes one step further and tries to explicitly capture the memory influence that caused TB to terminate after three iterations. In the example, TB terminated at time $t-2$, at which point the value of the test case's singular feature F is '-'. Thus, we can create the following MFs to capture this information from memory:

- $F(-)^T$: a time-back MF to track the number of time steps since the last time $F =$ '-'.
- $F(2)^V$: a value-back MF to track the value of F at time $t-2$ for $t \geq 2$.

With this information, TB would need fewer iterations to determine the appropriate solution. With additional MFs, TB would only need one iteration.

Algorithm 2 shows the major operations of MF Extraction for a given case C. These are:

1. Repeat the following until the number of TB iterations required to terminate for C is just one:
   (a) Get the number of iterations k that TB uses for C.
   (b) Create one value-back MF and one time-back MF from information about C at time $t-k$.
   (c) Create two copies of C and the case base. Incorporate one MF into each copy and run TB for each copy.
   (d) For the MF (value-back or time-back) whose TB run had the fewest iterations, do the following:

[1] https:/github.com/sachag678/LFO_Framework

- Add that MF to the list of extracted MFs.
- Permanently add this MF to the test case C and to each case in the case base.

2. Return the list of extracted MFs.

Algorithm 2 Memory Feature Extraction

```
1: procedure EXTRACT(case, casebase)
2:     T ← case.lastTimeStep
3:     extractedMFs ← (empty set)
4:     while true do
5:         numIter ← TB(case, casebase)
6:         if numIter == 1 then
7:             break
8:         valBackMF ← ValBack(case, T− numIter)
9:         timBackMF ← TimeBack(case, T− numIter)
10:        valCase, valCasebase ← addMF(valBackMF)
11:        timCase, timCasebase ← addMF(timBackMF)
12:        valNumIter ← TB(valCase, valCasebase)
13:        timeNumIter ← TB(timCase, timCasebase)
14:        if valNumIter ≤ timeNumIter then
15:            extractedMFs.add(valBackMF)
16:            case, casebase ← addMF(valBackMF)
17:        if timeNumIter ≤ valNumIter then
18:            extractedMFs.add(timBackMF)
19:            case, casebase ← addMF(timBackMF)
20:    return extractedMFs
```

In our example, Step 1a is illustrated with Table 3 and Step 1b is illustrated with the creation of MFs $F(-)^T$ and $F(2)^V$. Step 1c is illustrated with Table 4, which shows the incorporation of MF $F(2)^V$ (assuming F is '-' for any time steps not shown). The feature values at each time step are of the form ($F(2)^V$, F). Here, we see that TB now terminates after just two iterations. With the addition of more MFs, TB will use fewer and fewer iterations until it only requires one iteration because the MFs have captured all memory influences.

Table 4: MF Extraction Example

Time	Cases						Solns
	Test	1	2	3	4	5	
	?	α	β	β	α	α	
t	-, -	-, +	-, -	+, -	-, -	-, -	α, β
$t-1$	-, -		-, +		-, -	-, -	α
...	...				...	...	

The final output is a list of MFs for a given test case C that make C's sub-run conform to the Markov assumption. The extracted MFs encapsulate all memory influences from prior time steps for a given case and they allow TB to terminate after just one iteration. This process is performed for each case in the case base. We now describe the next task: reducing all extracted MFs to a refined subset.

MF Refinement and Trace Enhancement

After a set of MFs is extracted for each case in the MF Extraction stage, this multitude of MFs (many of which are repeated) must be reduced to a concise set of MFs that appear the most frequently and are therefore the most useful for capturing the major memory influences of the observed behavior. To do this, we compute the following values for each *unique* MF:

- p_c: the proportion of *cases* for which the MF is extracted
- p_t: the proportion of *traces* for which an MF appears at least once for some case derived from the trace

The proportions p_c and p_t are compared to experimentally determined domain-specific thresholds, $thresh_c$ and $thresh_t$, respectively. Each MF for which the conditions $p_c > thresh_c$ and $p_t > thresh_t$ hold will be retained in the final *refined* MF set.

The purpose of these case and trace coverage thresholds is to eliminate *artifacts* (extracted MFs) that appear as a result of a particular scenario or sequence of events instead of as a result of true memory influence on the observed behavior. For example, in a car driving domain, if a driver lowers her speed to match the value of the speed limit sign she just passed, then we wish to capture memory of the speed limit sign's value (a true memory influence on her behavior) and not memory of the presence of a red truck that just happened to pass her in approaching traffic (a coincidental artifact).

Blip artifacts pass neither threshold. These MFs are aberrations in the MF Extraction stage that bear no relation to the memory influences on an observed behavior. MFs that pass the case coverage threshold, but not the trace coverage threshold, are *trace artifacts*; such MFs are indicative of memory of specific aspects of a scenario faced in a particular trace instead of memory that affects the behavior overall. MFs that pass the trace coverage threshold, but not the case coverage threshold, are *rare memory influences* that affect the observed behavior so infrequently that they are negligible. Finally, MFs that pass both thresholds are the *true memory influences* that affect the observed behavior frequently in multiple scenarios — these are the MFs we want to retain.

Once the final set of MFs is captured in the MF Refinement stage, these MFs are added to the original traces to create memory-enhanced traces in the Trace Enhancement stage. The value of each MF is computed for every time step in each of the original traces and incorporated into the new memory-enhanced traces. Then, a machine learning algorithm that uses these memory-enhanced traces will have access to the salient aspects of memory that affect the observed behavior. The MFs can be treated the same way as the regular features. The intent is for the learned MFs to make the trace conform to the Markov assumption so that the information at each time step is sufficient for predicting the decision that is made by the observed actor at that time. We now discuss the assessment of our approach.

Assessment

We assessed the efficacy of our approach in modeling how an observed entity uses memory to improve machine learning performance by using the same simulated vacuum cleaner domain used in (Ontañón, Montaña, and Gonzalez 2014). In this domain, a vacuum cleaner agent moves up,

right, left, and down in a wall-enclosed 2D grid world, cleaning dirt patches and bumping into walls. The vacuum cleaner agent perceives the environment through eight binary features, two per direction that indicate the closest object in that direction (dirt or wall) and the distance to it (close or far). There are several agent behaviors defined in (Ontañón, Montaña, and Gonzalez 2014) for this domain:

- **Sequential (SEQ) — Level 1**: This agent repeatedly executes the same 20-move sequence.
- **Random (RD) — Level 2 Non-deterministic**: This agent performs a random action at each time step.
- **Wall Follower (WF) — Level 2 Deterministic**: This agent follows the left wall, but goes right if there is none.
- **Straight Line (SL) — Level 3 Non-deterministic**: This agent goes straight until it hits a wall, then it begins moving in a new random direction. The agent must remember its direction from the previous time step.
- **Zig Zag (ZZ) — Level 3 Deterministic**: This agent goes right until it hits a wall, then moves down one and goes left. At the bottom wall, it repeats its horizontal behavior going up. It must remember its short-term horizontal direction and long-term vertical direction.
- **Smart Agents**: The RD, WF, and SL agents have "smart" variants (named SRD, SWF, and SSL, respectively) that behave the same as their non-smart counterparts except that they prefer moving toward any sighted dirt.

We simulated each vacuum cleaner agent in seven maps to generate traces of 1000 time steps. Then, we ran MCL ($thresh_c = 0.05$, $thresh_t = 0.80$) on these traces to generate a set of memory-enhanced traces for each agent. (We set $thresh_c$ relatively low to capture memory influences that only manifest for a single time step, such as those influencing the ZZ agent's vertical movement.) We provided both the original traces and the memory-enhanced traces to three machine learning algorithms in the Weka framework (Hall et al. 2009): J48 decision tree, Bayes Network, and Multilayer Perceptron. We then recorded the F1-score using 10-fold cross validation. The average F1-score over all maps for each agent/classifier combination is shown in Table 5. For a given agent and learning algorithm, if the average F1-score for the MCL traces exceeded that for the original traces by at least 0.10, we bolded the MCL F1-score.

Table 5: Learning Performance Results - F1-score

Agent	Bayes Net		J48 Tree		MLP	
	Orig	MCL	Orig	MCL	Orig	MCL
SEQ	0.71	**0.82**	0.58	**1.00**	0.59	**0.99**
RD	0.35	0.35	0.17	0.20	0.18	0.20
SRD	0.35	0.32	0.30	0.33	0.33	0.33
SSL	0.53	**0.89**	0.52	**0.94**	0.52	**0.94**
SWF	0.99	0.99	0.99	0.99	1.00	1.00
SL	0.50	**0.89**	0.49	**0.94**	0.50	**0.94**
WF	1.00	1.00	1.00	1.00	1.00	1.00
ZZ	0.55	**0.99**	0.52	**1.00**	0.52	**0.98**

The trends observed for all machine learning algorithms are similar. For the Level 1 SEQ behavior, the original traces' F1-score was superseded by that of the memory-enhanced traces generated with MCL, showing that the learned MFs improved machine learning performance for SEQ. Performance improvements are also seen for the Level 3 SL, SSL, and ZZ behaviors when the memory-enhanced traces are used. For the Level 2 RD, SRD, WF, and SWF behaviors, there was no performance increase (because these behaviors don't use memory at all), however, there was no performance degradation. These results show the positive impact that MCL's learned MFs had on learning.

The learned MFs also shed insight into which memory influences MCL was able to detect for each agent behavior. The perception features are represented as $ObjX$ or $DistX$, which indicate the closest object in direction X and the distance to it; X is U (Up), D (Down), L (Left), or R (Right). Possible values for ObjX are 'dirt' or 'wall', possible values for $DistX$ are 'close' and 'far', and possible values for $Move$ (the agent's choice of movement) are U, D, L, R. The MFs learned for each agent behavior are presented below. The MFs we deemed "necessary" are underlined; the rest are considered "extraneous" (though possibly still helpful):

- **SEQ**: $DistU(Far)^T$, $\underline{Move(D)^T}$, $DistU(1)^V$, $DistU(2)^V$, $DistU(4)^V$, $\underline{Move(3)^V}$, $\underline{Move(5)^V}$
- **RD / SRD**: too many (over 25 each), all extraneous
- **WF / SWF**: none
- **SL**: $Move(U)^T$, $Move(R)^T$, $Move(L)^T$, $Move(D)^T$, $\underline{Move(1)^V}$
- **SSL**: $DistU(far)^T$, $DistD(2)^V$, $Move(U)^T$, $Move(R)^T$, $Move(L)^T$, $Move(D)^T$, $\underline{Move(1)^V}$
- **ZZ**: $\underline{Move(U)^T}$, $Move(R)^T$, $Move(L)^T$, $\underline{Move(D)^T}$, $\underline{Move(1)^V}$, $Move(8)^V$

For the Level 1 SEQ behavior, the learned MFs track the Action taken at time $t-3$ and $t-5$ as well as the time since the last "Down" action. These indicate an attempt by MCL to codify where in the 20-move sequence an agent is at a given time. The other learned MFs likely should have been categorized as trace artifacts, but were retained anyway.

For the Level 2 deterministic WF/SWF behaviors, no MFs were learned, which is good because these behaviors don't require memory. For the Level 2 non-deterministic RD/SRD behaviors, too many (unhelpful) MFs were learned because MCL does not yet have the capability for determining that a behavior is random and thus cannot benefit from memory.

For the Level 3 SL/SSL behaviors, time-back MFs were learned for each possible value of $Move$ to determine the most recent action taken, but only the value-back MF $Move(1)^V$ is needed for this. Therefore, future work should better compare the utility of time-back and value-back MFs for a given case. Finally, the MFs learned for the ZZ behavior appropriately captured memory of the agent's short-term horizontal and long-term vertical directions. Thus, the memory influences of each behavior were learned automatically by MCL with minimal knowledge of the domain and future work can further minimize the final refined MF set.

Related Work

There are some works in the literature that specifically address Level 3 memory-based behaviors by using one of two approaches. The first approach, probabilistic reasoning, seeks to model a behavioral probability distribution that captures stochastic memory-based behaviors. Dynamic Bayesian Networks (DBNs) were used to prove that standard supervised learning algorithms and metrics were insufficient for properly modeling and assessing stochastic Level 3 memory-based behaviors (Ontañón, Montaña, and Gonzalez 2014). However, new metrics and algorithms based on DBNs were shown to fulfill this task. Probabilistic Finite Automata (PFAs) were used to see how different memory constraints affected learning performance (Tîrnăucă et al. 2016); PFAs were generally the best approach for behavior recognition and behavior cloning with mild difficulty in differentiating between deterministic/stochastic variants of a behavior. The limitation of these approaches is that the structure for memory is too simplistically predefined; memory influence is either confined to a limited subset of prior time steps or it is constrained to a predefined discrete value set.

The second approach to learning memory-based behaviors uses case-based reasoning (CBR). CBR is considered a "lazy" machine learning algorithm that waits until run-time to generalize a solution to a novel problem. Temporal Backtracking (Floyd 2013) was used to represent entire runs in a case base and compare them to an agent's current run during deployment to select the next action. Various inductive biases were compared to that of Temporal Backtracking as part of a more general CBR approach to memory-based behaviors in (Gunaratne, Esfandiari, and Fawaz 2018) — no single inductive bias proved superior to the others. The limitation of these approaches is that memory influence must be inferred at each decision because no memory structure is maintained over time. For each new case in the case base, the computational cost increases quadratically.

Our approach overcomes the limitations of the prior approaches because 1) it does not predefine the memory structure like in the probabilistic reasoning approaches and 2) the MFs that MCL learns remove the need for machine learning to compare arbitrarily long history segments like in the CBR approaches. A set of MFs specifically tailored to an observed individual is learned automatically with no predefined memory structure. These MFs then help restore the i.i.d. assumption used by standard supervised learning algorithms.

Conclusions

Learning from Observation is an advantageous machine learning paradigm for modeling observed behaviors and capturing implicit behaviors that are hard to articulate but easier to demonstrate. However, the major challenge of LfO is modeling crucial unobservable internal processes that influence an entity's behavior. In this paper, we introduced a novel approach, Memory Composition Learning (MCL), that learns memory features (MFs) that capture salient aspects of memory of past events in an observed behavior that are highly correlated with an actor's decisions. These MFs are used to produce memory-enhanced traces that consolidate all pertinent information from memory into the current state and enable learning algorithms that rely on the Markov assumption to learn memory-based behaviors. We showed the efficacy of our approach in a simulated vacuum cleaner domain and found that MCL determined which aspects of memory were useful for modeling and that the resultant memory-enhanced traces improved machine learning performance of memory-influenced behaviors.

Acknowledgments: This material is based on work partially supported by the National Science Foundation under Grant No. IIS-1521972.

References

Bentivegna, D. C., and Atkeson, C. G. 2001. Learning from Observation Using Primitives. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 1988–1993. IEEE.

Faltersack, Z.; Burns, B.; Nuxoll, A.; and Crenshaw, T. L. 2011. Ziggurat: Steps Toward a General Episodic Memory. In *Proceedings of the AAAI Fall Symposium: Advances in Cognitive Systems*, 106–111. AAAI Press.

Fernlund, H. K.; Gonzalez, A. J.; Georgiopoulos, M.; and DeMara, R. F. 2006. Learning Tactical Human Behavior through Observation of Human Performance. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 36(1):128–140.

Floyd, M. 2013. *A General-Purpose Framework for Learning by Observation*. Ph.D. Dissertation, Dept. of Computer Science, Carleton University.

Gunaratne, A. S. E.; Esfandiari, B.; and Fawaz, A. 2018. A Case-Based Reasoning Approach to Learning State-Based Behavior. In *Proceedings of the FLAIRS-31 Conference*, 377–382. AAAI Press.

Hall, M.; Franke, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The Weka Data Mining Software: An Update. In *Proceedings of SIGKDD Explorations*, 10–18. ACM.

Ontañón, S.; Montaña, J.; and Gonzalez, A. 2011. Towards a Unified Framework for Learning from Observation. In *IJCAI Workshop on Agents Learning Interactively from Human Teachers (ALIHT)*.

Ontañón, S.; Montaña, J. L.; and Gonzalez, A. J. 2014. A Dynamic-Bayesian Network Framework for Modeling and Evaluating Learning from Observation. *Expert Systems with Applications* 41(11):5212–5226.

Pomerleau, D. A. 1989. Alvinn: An Autonomous Land Vehicle in a Neural Network. In *Proceedings of Advances in Neural Information Processing Systems*, 305–313. NIPS.

Sidani, T. A., and Gonzalez, A. J. 2000. A Framework for Learning Implicit Expert Knowledge through Observation. *Transactions of the Society for Computer Simulation* 17(2):54–72.

Tîrnăucă, C.; Montaña, J. L.; Ontañón, S.; Gonzalez, A. J.; and Pardo, L. M. 2016. Behavioral Modeling Based on Probabilistic Finite Automata: An Empirical Study. *Sensors* 16(7):958.