Ankuj Arora*, Humbert Fiorino, Damien Pellier, and Sylvie Pesty

Learning robot speech models to predict speech acts in HRI

https://doi.org/10.1515/pjbr-2018-0015
Received November 29, 2017; accepted July 5, 2018

Abstract: In order to be acceptable and able to “camouflage” into their physio-social context in the long run, robots need to be not just functional, but autonomously psycho-affective as well. This motivates a long term necessity of introducing behavioral autonomy in robots, so they can autonomously communicate with humans without the need of “wizard” intervention. This paper proposes a technique to learn robot speech models from human-robot dialog exchanges. It views the entire exchange in the Automated Planning (AP) paradigm, representing the dialog sequences (speech acts) in the form of action sequences that modify the state of the world upon execution, gradually propelling the state to a desired goal. We then exploit intra-action and inter-action dependencies, encoding them in the form of constraints. We attempt to satisfy these constraints using a weighted maximum satisfiability model known as MAX-SAT, and convert the solution into a speech model. This model could have many uses, such as planning of fresh dialogs. In this study, the learnt model is used to predict speech acts in the dialog sequences using the sequence labeling (predicting future acts based on previously seen ones) capabilities of the LSTM (Long Short Term Memory) class of recurrent neural networks. Encouraging empirical results demonstrate the utility of this learnt model and its long term potential to facilitate autonomous behavioral planning of robots, an aspect to be explored in future works.

Keywords: human robot interaction, automated planning, SAT, LSTM

1 Introduction

For social robots, socio-communicative qualities are increasingly becoming indispensable as they strengthen the possibility of acceptance and emotional attachment to the robot [1, 2]. This acceptance is likely only if the robot fulfills a fundamental expectation that one being has of the other: not only to do the right thing, but also at the right time and in the right manner [1]. Classical Human Robot Interaction (HRI) approaches advocate that much like humans, robots can only learn with experience. Thus the most common strategy used to endow robots with social skills is to analyze, model and implement human behaviors either by observation, imitation or demonstration techniques [3]. The limitation with this approach is the misalignment of the degrees of freedom of the robot with those of the human instructor. The other approach is to “codify” speech into robots. While this ensures autonomy, encoding into a robot the subtleties of a social interaction is not trivial. A standard dialogue exchange integrates the widest possible panel of signs which intervene in the communication and are difficult to codify (the tone of the voice, emotion being conveyed etc.). Human emotions are masked in subtlety, making them complicated as it is to understand and respond to, rendering them even more difficult to program. Scripting these subtleties is time consuming, impractical and requires a lot of programming. In such a complex scenario, learning the underlying speech model of the robot from HRI dialog sequences is a promising alternative.

This learning can be done with the help of Artificial Intelligence (AI) techniques, by viewing the entire scenario in the Automated Planning (AP) paradigm. In this study, the body of HRI dialog exchanges is drawn from the Vernissage corpus [4] and consists of exchanges between a Nao robot (operated by a skilled operator in a Wizard-of-Oz setup) posing as a museum guide and two human visitors to the museum. An example of the dialog exchange can be seen in Listing 1.

Listing 1: Example of dialog exchange between NAO as a museum guide and museum visitors.

Nao : Hello, my name is Nao, what is yours?
Human: My name is Alan.
Nao: It’s a pleasure to meet you Alan, can I tell you something about these paintings?
Human: Yes, please.

In the context of AP, each utterance is considered ideologically equivalent to an action (represented in the corpus in the form of an action name and signature). Thus, each dialog sequence is represented as a sequential and orchestrated action name-signature sequence which effectuates transitions in the world state, gradually propelling it towards a predetermined goal. By means of these utterances, the speaker plans to influence the listeners’ beliefs, goals and emotional states [1, 2]; with the intent of fulfilling his own goals. These utterances, interleaved with head and body movements (i.e. multi modal acts) can then be modeled as sequences of actions in an AP system, and frameworks can be developed for providing their semantics [5, 6]. However, for the sake of simplicity, in this work we ignore the definite uncertainty in communication, as well as the head and body movements comprising the acts, and concentrate solely on utterances. The objective is then to leverage the advancements in Machine Learning (ML) to learn the underlying action model from the data i.e. learning the preconditions and effects of each of the constituent actions from their signatures. Learning the underlying model comprising action descriptions from action name-signature sequences could save the effort from having to code these action descriptions from scratch, thus promoting re usability. This model can further be fed to an automated planner to generate fresh dialogs, thus allowing the robot to communicate autonomously in future scenarios.

Our contribution in this paper is the following: given a HRI dialog corpus, our approach learns the robot’s behavioral model comprising of the utterances encoded in the form of actions alongwith their signatures, preconditions and effects. The novelty of the approach lies in bridging of symbolic and connectionist machine learning methods to make progress on the problem of learning an interactive model of a human for use in human-robot interaction. The work can be considered a progressive step in the automatic generation of dialog sequences, an aspect which will become more and more relevant as social robotics makes inroads into the real world and manual interaction script generation becomes economically inviable. The approach is divided into three phases. In the first phase, HRI dialogues taken from the Vernissage corpus [4] are annotated with the following pre-engineered speech acts: (sayHello, farewell, inform, say, ask, claim, deny, advise, thank, autoFeedback (feedback on last heard dialogue e.g. “oh”, “great” etc.). These acts are drawn from the encoding of speech acts introduced in [7]. The reader is invited to read the aforementioned paper for more detail on the encoding scheme. The second phase consists of learning the speech model. The MAX-SAT framework is used for this purpose. The constraints used as the input for this framework are constituted by intra-action and interaction specific constraints. These constraints are then fed to a weighted MAX-SAT solver, the solved constraints being used to reconstruct the underlying action model. Feeding this learnt model to a planner to generate fresh interactions and testing them in a real scenario is beyond the scope of this paper. We, however, demonstrate the utility of this learnt model by predicting speech acts likely to be executed by the robot for the same dialog corpus. For example, using the speech model, we predict whether in the scenario of the robot meeting a subject for the first time, the probability that the act of greeting the subject is followed by the act of introducing itself. This is performed with the aid of sequence labelling capabilities of recurrent neural networks. Deep learning, in general, offers multiple layers of representation wherein the features are not designed by human engineers but learn from data using a general-purpose learning procedure [8]. It has also been extensively used in the field of robotics [9]. In particular, the successes of LSTM [10], a deep learning technique being increasingly used in learning long range dependencies in the fields of speech and handwriting recognition [8], are exploited. A plan is an orchestrated execution of a series of actions which are by virtue interdependent in order to execute. There is thus a direct link between the principle of operation of the LSTM and plan execution, which this paper seeks to explore.

This paper is divided into the following sections: we firstly present some related work in Section 2, followed by the definition of our learning problem in Section 3. We then detail the functioning of our learning system in Section 4, following it up with empirical evaluations in Section 5. We conclude the paper with some perspectives and future work in Section 6.

2 Related work

We elaborate the state of the art of two different fields: machine learning (ML) in AP and ML in robotics. Learning action models in the field of AP have a considerable history. Some prominently used ML techniques to learn action models include: inductive techniques (absence of back-
ground knowledge, e.g. PELA [11]), transfer learning techniques (extraction of knowledge from one or more source tasks to apply to the target task when the latter has fewer high-quality training data [12] e.g. TRAMP [13]), reinforcement learning based approaches (e.g. [14]). More specific to our case, certain approaches have used the MAX-SAT framework to learn deterministic actions (e.g. ARMS [15]), macro-actions [16], models in Hierarchical Task Networks (HTNs) [17], and models in multi-agent setups ([18]). In particular, our approach is on the same lines of ARMS, which also generates intra-action and inter-action constraints (mined with the Apriori algorithm [19]). These approaches, however, use a partially ordered series of actions as input. Since in our problem multiple speech acts can co-occur at the same time (e.g. multiple participants speaking at the same time), we are primarily interested in co-occurring actions. On the robotics side, there has been considerable use of ML for task-oriented robot behavior learning [20] and robot motion planning [21]. Deep learning has also been extensively used in the robotics community [9]. Our work, however, is concentrated on the learning of PDDL (Planning Domain Description Language) [22], a syntax for planning domains used for behavioral models for robots; a requirement not catered for in the aforementioned works. The literature also speaks of other approaches which have treated dialogues as planning operators. In [23] the authors introduce a new language called MAPL (Multi Agent Programming Language) which represents speech acts as operators with qualitative and quantitative temporal relations between them. Other works use AP to generate natural language sentences for communication [24] or treating utterances in the form of actions [25]. Contrary to the previous works, our problem is centered on the learning of the action model which serves as a prerequisite to be fed to a planner prior to it being utilized in the forms mentioned as in the aforedescribed works.

3 Definitions and problem formulation

We begin by laying out some important definitions of AP which symbolize the HRI interaction. In the field of AP, agents interact with the environment by executing actions which change the state of the environment, gradually propelling it from its initial state towards the agents’ desired goal. In the classical representation, both the world state and actions are pre-engineered and constituted by properties called predicates. States are defined to be sets of ground (positive) predicates. Here, each action $a \in A$ where $A = \{a_1, a_2, \ldots, a_n\}$, $n$ being the maximum number of actions in the domain. We use actions and operators interchangeably in our context. These actions constitute a corpus which serves as a blueprint for these actions, called the action model. An action model $m$ is the blueprint of all the domain-applicable actions belonging to the set $A$. Each action in the model is defined as an aggregation of: (i) the action name (with zero or more typed variables as parameters), and (ii) three lists, namely $\text{pre}$, $\text{add}$ and $\text{del}$. These are: the $\text{pre}$ list (predicates whose satisfiability determines the applicability of the action), $\text{add}$ list (predicates added to the current system state by the action execution) and the $\text{del}$ list (predicates deleted from the current system state upon action execution), respectively. A planning problem is a triplet $P = (s_0, g, m)$ composed of (i) the initial state of the world $s_0$, (ii) the desired goal $g$ and (iii) the action model $m$. All the aforementioned elements in this section contribute to the formulation of a plan. A plan, given $P$, is a sequence of actions represented as: $\pi = [a_1, a_2, \ldots, a_n]$ that drives the system from the initial state to the goal. Each action sequence, complete with initial state and goal information, constitutes a trace. Each dialogue sequence in HRI can be viewed as a trace in AP. An aggregation of these traces constitutes a trace set $T$, which in this case corresponds to our HRI dialog corpus. Having described these preliminaries, we can proceed to formulating our problem.

Given the aforementioned information, our problem can be formulated as follows: given (i) a set of HRI traces $T$, each trace consisting of a dialogue sequence encoded in the form of actions; our approach produces a complete domain model $m$ encompassing all the domain-applicable operators which best explain the observed traces. This is done by encoding the inter-operator and intra-operator dependencies in the form of constraints and solving them as a satisfiability problem, then reconstructing the domain model from the satisfied constraints. We then proceed to use the learnt model to predict the label of the operators in the traces using LSTM techniques.

4 Approach

The approach can be divided into four phases, a snapshot of which is illustrated in Figure 1. The dialogs of the HRI traces are first annotated to symbolize a sequence of actions (each consisting of a name and a signature). In the first phase, we generalize the grounded actions in the trace set by replacing the variables by their pre-engineered types to obtain a trace set of operators. The second phase is dedicated to constraint generation; namely intra-operator
and inter-operator ones. In the third phase, these constraints are supplied to a MAX-SAT solver and the satisfied ones are used to reconstruct the action model. This model consists of operator signatures, preconditions (alias for \texttt{pre list}), and effects (alias for \texttt{add} and \texttt{del} lists). It conforms to the semantics of STRIPS [26]. In the fourth phase, using the operators learnt in the previous phase as features, we predict the operators likely to be chosen by the robot based on previous ones during the dialog exchange highlighted in the corpus. This classification is done by encoding the operator sequences of the corpus in the form of input and output vectors to an LSTM. We elaborate these phases in the forthcoming subsections.

### 4.1 Annotation and generalization

In this phase, dialogues in the dialogue corpus are first annotated to actions. These dialogues are taken from the Vernissage corpus which is a multimodal HRI dataset [4], and has a total of 10 conversation instances between Nao and human participants. In the scenario, the robot explains paintings in a room and then performs a quiz with the participants. We first annotate with predefined predicates the initial state of the world before the beginning of each interaction. We then annotate each robot dialogue within each trace as an action drawn from the following set of actions: \texttt{sayHello}, \texttt{farewell}, \texttt{inform}, \texttt{say}, \texttt{ask}, \texttt{claim}, \texttt{deny}, \texttt{advise}, \texttt{thank}, \texttt{autoFeedback} (feedback on last heard dialogue e.g. “oh”, “great” etc.). The participant responses are encoded in the form of \texttt{(speech, silence, laughter)}. These actions and predicates are derived from the encoding of speech acts in [7]. This encoding is chosen as it depicts speech acts in logical form, rendering it more conducive to translate into PDDL operators. These actions are represented in the form of constituents of a PDDL model [22] whose snippet is represented in Listing 3. This model is referred to as the ground truth model, or the correct baseline model that the empirically generated model will compare to in order to gauge the accuracy of our proposed learning approach. This ground truth model consists of the names of the 11 actions described above, along with each of their preconditions and effects in the form of predicates. The interpretation of these predicates is detailed in Table 2. An example of the annotation process can be seen in Table 1.

The next step is generalization, which is done by scanning each action in each of the traces and substituting its instantiated variables with their corresponding types. This produces a trace set of operator sequences, with the generalized actions constituting an operator schema \( O_s \). We then create a dictionary of all possible relevant predicates to each operator, the keys of the dictionary identified by the operator names. Each operator in the operator schema is associated with its relevant predicates, where a predicate \( p \) is said to be relevant to an operator \( o \in O_s \) if they share the same variable types. We denote the relevant predicate dictionary as \( \text{relPre} \), with the set of relevant predicates to an operator \( o \) represented as key can be denoted as \( \text{relPre}_o \) (represented as value of key \( o \)). The generalization procedure is represented in Figure 2. The complexity of this phase comes out to be \( l m n \), where \( l \) is the number of actions per trace, \( m \) is the number of traces, and \( n \) is the number of unique predicates in the dictionary.
Learning robot speech models to predict speech acts in HRI

Variables Resolved by MAX-SAT: \{1, 7, 12, \ldots\}

Learned Model:
- (action key p = \text{say} parameters \{?i - agent ?j - agent ?p - proposition\})
- precondition (\text{and} (\text{bel} ?i ?j) (\text{goal} ?i ?j ?p))
- offset (\text{bel} ?i ?j ?p)

Figure 3: Illustration of Phases 3 and 4 of our approach. Phase 3 involves solving of the formulated constraints as a SAT problem to reconstruct the underlying model. Phase 4 involves using the previously learnt model to perform sequence labelling on the speech acts comprising the training data.

Listing 2: Predicates and their implications in speech acts.

\text{bel} (?i ?p) - agent ?i believes in the proposition ?p

\text{goal} (?i ?p) - proposition ?p is the goal of agent ?i

\text{ideal} (?i ?p) - proposition ?p should ideally be true for agent ?i

\text{belbelapproval} (?i ?j ?p) - agent ?i believes that agent ?j believes that ?i gives its approval for proposition ?p

\text{belideal} (?i ?j ?p) - agent ?i believes that proposition ?p should be the ideal for agent ?j

\text{goal} (?i ?p) - proposition ?p is not the goal of agent ?i

\text{notbel} (?i ?p) - agent ?i does not believe in the proposition ?p

\text{belbel} (?i ?j ?p) - agent ?i believes that agent ?j believes in the proposition ?p

\text{belgoal} (?i ?j ?p) - agent ?i believes that the goal of agent ?j is proposition ?p

\text{notbelbel} (?i ?j ?p) - agent ?i believes that agent ?j does not believe in the proposition ?p

\text{goalbel} (?i ?j ?p) - the goal of agent ?i is that agent ?j believes in proposition ?p

\text{goalresp} (?i ?j ?p) - the goal of agent ?i is that agent ?j assumes responsibility of proposition ?p

\text{notgoalresp} (?i ?j ?p) - the goal of agent ?i is not that the agent ?j assumes responsibility of proposition ?p

\text{belbelgoalresp} (?i ?j ?p) - agent ?i believes that agent ?j believes that the goal of ?i is ?j assumes the responsibility of proposition ?p

\text{belbelnotgoalresp} (?i ?j ?p) - agent ?i believes that agent ?j believes that the goal of ?i is not that ?j assumes the responsibility of proposition ?p

\text{goalbelgoalrespWeak} (?i ?j ?p) - goal of agent ?i is that agent ?j believes that the goal of ?i is that ?j assumes the responsibility of proposition ?p to a lesser magnitude

\text{goalbelstr} (?i ?j ?p) - goal of agent ?i is that agent ?j believes strongly in proposition ?p

\text{belgratitude} (?i ?j ?p) - goal of agent ?i is to express to agent ?j its gratitude for proposition ?p

\text{belresp} (?i ?j ?p) - goal of agent ?i is that agent ?j assumes responsibility for proposition ?p


\text{look} (?i ?j) - agent ?i looks at agent ?j

\text{at} (?i ?r) - agent ?i is at room ?r

\text{already-seen} (?i ?j) - agent ?i has already seen agent ?j

\text{never-seen} (?i ?j) - agent ?i has never seen agent ?j
4.2 Constraint generation

In this phase, we detail the intra-action constraints for individual operators and inter-operator temporal constraints among operators.

4.2.1 Intra-operator constraints

In order to satisfy the semantics of STRIPS [26], each operator in \( O_s \) must satisfy certain intra-operator constraints. Thus, for each operator \( o_i \in O_s \) and relevant predicate \( p \in \text{relPre}_o \): (i) \( p \) cannot be in the \( \text{add} \) list and the \( \text{del} \) list at the same time, and (ii) \( p \) cannot be in the \( \text{add} \) list and the \( \text{pre} \) list at the same time. The relevant predicates of each operator are encoded to generate variables. Each association of a relevant predicate with one of a \( \text{(pre, add, del)} \) list of an operator can be encoded as a variable. These variables and constraints are illustrated in Figure 2.

4.2.2 Inter-operator constraints

Any operator sequence exhibits inter-operator dependencies, which can be uncovered by means of data mining techniques to facilitate the process of learning. In the case of an HRI scenario, operators may be temporally concurrent: for example, two agents may be speaking at the same time. These temporal dependencies can be explored with the help of pattern mining techniques. The aforementioned trace set \( T \) can thus be treated as a set of transactions \( S \) in the pattern mining domain. \( S \) can be written as \( S = [s_1, s_2, \ldots, s_n] \), where each transaction \( s_n \) represents a trace. A transaction is a set of symbols, in our case representing operators. A time series is a set of unique time points. A symbolic time series is generally used to represent series of ordered (but non time-sampled) events (e.g. \( < a, b, d, a, c > \) ). A symbolic time sequence is a multi-set of temporal points. Contrary to time series, symbolic time sequences can deal with several events defined at the same temporal location. This difference is also illustrated in the Figure 7. We symbolize the concurrent utterances in the Vernissage corpus in the form of operators chaining together in a symbolic time sequence [29].

Listing 3: Ground truth speech model called “hri” to symbolize dialog exchange in Vernissage corpus. The pre-engineered types are \( \text{(agent, proposition, room)} \) are defined adjacent to the \( \text{types} \) keyword. The predicates are defined adjacent to the \( \text{predicates} \) keyword. This is followed by the definition and description of the constituent actions, (their preconditions and effects defined adjacent to the \( \text{precondition} \) and \( \text{effect} \) keywords respectively.

```scheme
(define (domain hri)
  (:requirements :strips :typing)
  (:types
    agent - object
    robot human - agent
    proposition - object
    room - object
  )
  (:predicates
    (bel ?i - agent ?p - proposition)
    (goal ?i - agent ?p - proposition)
    (ideal ?i - agent ?p - proposition)
    (approval ?i - agent ?p - proposition)
    (belbelapproval ?i - agent ?j - agent ?p - proposition)
    (belideal ?i - agent ?j - agent ?p - proposition)
    (notgoal ?i - agent ?p - proposition)
    (notbel ?i - agent ?j - agent ?p - proposition)
    (belbel ?i - agent ?j - agent ?p - proposition)
    (belgoal ?i - agent ?j - agent ?p - proposition)
    (belbelgoal ?i - agent ?j - agent ?p - proposition)
    (belbelnotgoal ?i - agent ?j - agent ?p - proposition)
    (goalbel ?i - agent ?j - agent ?p - proposition)
    (goalresp ?i - agent ?j - agent ?p - proposition)
    (notgoalresp ?i - agent ?j - agent ?p - proposition)
    (goalbelgoal ?i - agent ?j - agent ?p - proposition)
    (goalbelgoalresp ?i - agent ?j - agent ?p - proposition)
    (goalbelgratitude ?i - agent ?j - agent ?p - proposition)
    (belresp ?i - agent ?j - agent ?p - proposition)
    (belbelmoralsatisfaction ?i - agent ?j - agent ?p - proposition)
    (look ?a - agent ?b - agent)
    (at ?a - agent ?l - room)
    (already-seen ?i - agent ?j - agent)
    (never-seen ?i - agent ?j - agent)
  )
  ;; One agent greets another agent
  (:action sayHello
    :parameters (?i - agent ?j - agent)
  )
)```

Unauthenticated
Learning robot speech models to predict speech acts in HRI

Various approaches to mine temporal time sequences are presented in the literature. Winepi [27] is a well known algorithm which learns episodes and association rules based on the episodes. The face algorithm allows for mining of chronicles from symbolic time sequences [28]. These approaches are, however, not equipped to deal with temporal inaccuracies in the temporal events. We choose the Temporal Interval Tree Association Rule Learning (Titarl) algorithm [29] as it allows the representation of imprecise
association rules. An association rule is a means of uncovering relationships between operators can be uncovered by means of association rule learning, which is the search for association rules. An association rule is a conditions \rightarrow implications pattern.

We hypothesize that if an association rule frequently correlates two operators, there must be a reason for their frequent co-existence. The input traces are parsed and fed to the Titarl algorithm, which produces temporal association rules [29]. The operators featuring in the frequent co-existence of two operators, there must be a reason for their frequent co-existence of these operators. More precisely, if there is an operator pair \((o_i, o_j)\), \(0 \leq i < j \leq (n - 1)\) where \(n\) is the total number of operators in the plan; and \(\text{pre}_i\), \(\text{add}_i\) and \(\text{del}_i\) represent \(o_i\)'s pre, add and del list, respectively:

- A predicate \(p\) which serves as a precondition in the pre lists of both \(o_i\) and \(o_j\) cannot be deleted by the first operator.
- A predicate \(p\) added by the first operator \(o_i\) \((p \in \text{add}_o)\) which serves as a prerequisite for the second operator \(o_j\) \((p \in \text{pre}_o)\), cannot be deleted by the first operator \(o_i\).
- A predicate \(p\) that is deleted by the first operator \(o_i\) is added by \(o_j\). In other words, an operator re-establishes a predicate that is deleted by the previous operator.
- The above plan constraints can be combined into one constraint and restated as:

\[
\exists p((p \in \text{pre}_i \cap \text{pre}_j) \land p \notin \text{del}_i) \lor (p \in (\text{add}_i \cap \text{pre}_j)) \lor (p \in (\text{del}_i \cap \text{add}_j))
\]

A snippet of the aforementioned constraints is illustrated for the action pair \((\text{ask}, \text{say})\) and relevant predicate already-seen in the Figure 2.

The details pertaining to the resolution of the intra and inter-operator constraints with the help of a SAT solver to reconstruct the underlying action model is detailed in the Figure 3.

### 4.3 LSTM based operator classification

As mentioned before in section 3, a plan is a chained series of interdependent operators directed towards the accomplishment of a goal. Thus, extracting patterns from sequences of previously executed operators is likely to provide strong evidence to predict the label of the next operator in the chain; inspiring our investigation of long short-term memory networks (LSTMs) [10] for action sequence labelling. In the following subsections we present our data encoding method for the input and output vector of the LSTM.

#### 4.3.1 Data encoding for labelling of operator sequences

We use the sequence labelling capabilities of LSTM to identify the most likely operator that succeeds a given one. The input to our LSTM system is a large corpus comprising vector representations of each operator of each trace. Each trace is taken one by one, and the comprising operators are sequentially encoded into input and output vectors; thus producing a large corpus of vectors. Each operator of each trace is represented by two distinct vectors: an input vector which encodes the operator, and an output vector which classifies the successive operator. These vec-

---

**Table 1:** Sample annotation of traces. The “Utterance” in the second column at the timestamp (in seconds) from the beginning of the interaction mentioned in the column “Timestamp” is annotated with the speech act in the column “Annotation”. The (...) represents the operator parameters.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Utterance</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>35.803</td>
<td>Hello</td>
<td>sayHello(…)</td>
</tr>
<tr>
<td>35.803</td>
<td>Should I tell you something about these paintings?</td>
<td>ask(…)</td>
</tr>
<tr>
<td>395.619</td>
<td>So, I am starting the quiz!</td>
<td>claim(…)</td>
</tr>
<tr>
<td>556.885</td>
<td>Great, your answer is perfectly right.</td>
<td>autoFeedback(…)</td>
</tr>
<tr>
<td>629.532</td>
<td>So, it's the end of this quiz.</td>
<td>say(…)</td>
</tr>
</tbody>
</table>

**Figure 4:** Vector representations for the operator “ask” and successive operator “say” for the learnt behavioral model.
The input vector representing an operator in a trace is encoded in the following fashion. It is divided into two sections: one section which labels the entire set of operators in the domain, and the other which labels the relevant predicates for the operators in the domain. In the first block, there is a slot for each operator in the domain. The slot representing the operator currently being encoded is set to 1, the others being set to 0. For example, as illustrated in the Figure 4, the slots in the vector which correspond to the relevant predicates in each operator of the empirical model are switched to 1, the others being switched to 0. For example, let us assume that the operator currently being encoded is ask and the next operator in the trace is say. The input and output vectors for the ask operator are represented in Figure 4. While the number of operators in a trace thus the number of vectors representing all the operators in a trace may vary, the LSTM requires a fixed sized input. This is ensured by calculating the maximum trace length \( batchLen \) (maximum number of operators per trace) for all the traces, and padding the shorter lists with \( d \)-dimensional vectors filled with zeros. This padding is done for all the traces till all the traces have the same \( batchLen \) number of operators. The same padding procedure is adopted for the output vectors.

The input vectors are identical in the way they are labelled for the training and validation set. The first section is represented in the same way, with the label of the currently encoded operator set to 1. In this case, there are as many slots as the number of operators in the domain, one for each operator. The slot representing the succeeding operator to the operator being currently encoded is set to 1, the others being set to 0. For example, let us assume that the operator currently being encoded is ask and the next operator in the trace is say. The input and output vectors for the ask operator are represented in Figure 4. While the number of operators in a trace is thus the number of vectors representing all the operators in a trace may vary, the LSTM requires a fixed sized input. This is ensured by calculating the maximum trace length \( batchLen \) (maximum number of operators per trace) for all the traces, and padding the shorter lists with \( d \)-dimensional vectors filled with zeros. This padding is done for all the traces till all the traces have the same \( batchLen \) number of operators. The same padding procedure is adopted for the output vectors.

The input vectors are identical in the way they are labelled for the training and validation set. The first section is represented in the same way, with the label of the currently encoded operator set to 1. In this case, there are as many slots as the number of operators in the domain, one for each operator. The slot representing the succeeding operator to the operator being currently encoded is set to 1, the others being set to 0. For example, let us assume that the operator currently being encoded is ask and the next operator in the trace is say. The input and output vectors for the ask operator are represented in Figure 4. While the number of operators in a trace is thus the number of vectors representing all the operators in a trace may vary, the LSTM requires a fixed sized input. This is ensured by calculating the maximum trace length \( batchLen \) (maximum number of operators per trace) for all the traces, and padding the shorter lists with \( d \)-dimensional vectors filled with zeros. This padding is done for all the traces till all the traces have the same \( batchLen \) number of operators. The same padding procedure is adopted for the output vectors.

The input vectors are identical in the way they are labelled for the training and validation set. The first section is represented in the same way, with the label of the currently encoded operator set to 1. In this case, there are as many slots as the number of operators in the domain, one for each operator. The slot representing the succeeding operator to the operator being currently encoded is set to 1, the others being set to 0. For example, let us assume that the operator currently being encoded is ask and the next operator in the trace is say. The input and output vectors for the ask operator are represented in Figure 4. While the number of operators in a trace is thus the number of vectors representing all the operators in a trace may vary, the LSTM requires a fixed sized input. This is ensured by calculating the maximum trace length \( batchLen \) (maximum number of operators per trace) for all the traces, and padding the shorter lists with \( d \)-dimensional vectors filled with zeros. This padding is done for all the traces till all the traces have the same \( batchLen \) number of operators. The same padding procedure is adopted for the output vectors.

The input vectors are identical in the way they are labelled for the training and validation set. The first section is represented in the same way, with the label of the currently encoded operator set to 1. In this case, there are as many slots as the number of operators in the domain, one for each operator. The slot representing the succeeding operator to the operator being currently encoded is set to 1, the others being set to 0. For example, let us assume that the operator currently being encoded is ask and the next operator in the trace is say. The input and output vectors for the ask operator are represented in Figure 4. While the number of operators in a trace is thus the number of vectors representing all the operators in a trace may vary, the LSTM requires a fixed sized input. This is ensured by calculating the maximum trace length \( batchLen \) (maximum number of operators per trace) for all the traces, and padding the shorter lists with \( d \)-dimensional vectors filled with zeros. This padding is done for all the traces till all the traces have the same \( batchLen \) number of operators. The same padding procedure is adopted for the output vectors.

The input vectors are identical in the way they are labelled for the training and validation set. The first section is represented in the same way, with the label of the currently encoded operator set to 1. In this case, there are as many slots as the number of operators in the domain, one for each operator. The slot representing the succeeding operator to the operator being currently encoded is set to 1, the others being set to 0. For example, let us assume that the operator currently being encoded is ask and the next operator in the trace is say. The input and output vectors for the ask operator are represented in Figure 4. While the number of operators in a trace is thus the number of vectors representing all the operators in a trace may vary, the LSTM requires a fixed sized input. This is ensured by calculating the maximum trace length \( batchLen \) (maximum number of operators per trace) for all the traces, and padding the shorter lists with \( d \)-dimensional vectors filled with zeros. This padding is done for all the traces till all the traces have the same \( batchLen \) number of operators. The same padding procedure is adopted for the output vectors.
of the speech model. In this alternative scheme, the first section is represented in the same way, with the label of the currently encoded operator set to 1. For the second section, the slots in the vector which correspond to the relevant predicates in each operator are set to 1. For example, as illustrated in the Figure 5, if the action currently being encoded is ask, the predicates relevant to the action ask i.e. the first block of the second section corresponding to the first operator are labelled to 1, the rest of the blocks of this section being kept at zero.

5 Evaluation

The objective of our evaluation is to obtain the best possible accuracy of: (i) the learnt model vis-a-vis the ground truth model and (ii) sequence labeling obtained with the learnt speech model. Concerning inter-operator constraint representation, two measures are defined for association rules of the form $a \Rightarrow b$, $b$ being the head and $a$ being the body of the rule. The confidence of a rule is the percentage of occurrences of the rule's body which also matches the body and the head of the rule i.e. $\text{support(body)} / \text{support(head + body)}$. The support of a rule is the percentage of occurrences of the rule's head which also matches the body and the head of the rule i.e. $\text{support(head)} / \text{support(head + body)}$ [29].

The Titarl-mined association rules along with their confidence and support are recorded. The rules with a highest value of confidence and support (empirically determined) are retained for inter-operator constraint generation (see Table 2).

Finally we encode all the intra and inter-operator constraints in the form of a weighted MAX-SAT problem. The weights of the CNF clauses representing the constraints are determined differently for the inter and intra-operator cases. While the weights of the intra-operator clauses are empirically determined, the weights of the inter-operator clauses are equal to the support of the association rules. This problem can be stated as: Given a collection $C$ of $m$ clauses, $(C_1, \ldots, C_m)$ involving $n$ logical variables with clause weights $w_i$, find a truth assignment that maximizes the total weight of the satisfied clauses in $C$ [15]. We use 2 SAT solvers: the MaxSatSolver [30] and the MaxWalkSat [31]. The solution produced by either solver contains all the variables which evaluate to true, which are then used to reconstruct the empirical model.

The difference between the ground truth model and the empirically determined model is represented in the form of a reconstruction error. This error is based on the similarity between the relevant predicates and the empirically determined predicates per operator. Let $\text{diffPre}_{o_i}$ represent the difference in pre lists of operator $o_i$ in the ground truth model and the empirical model. Each time the pre list of the ground truth model presents a predicate which is not in the pre list of the empirical model, the count $\text{diffPre}_{o_i}$ is incremented by one. Similarly, each time the pre list of the empirical model presents a predicate which is not in the pre list of the ground truth model, the count $\text{diffPre}_{o_i}$ is incremented by one. Similar counts are computed for the add and del lists as $\text{diffAdd}_{o_i}$ and $\text{diffDel}_{o_i}$ respectively.

This total count is then divided by the number of relevant constraints for that particular operator $\text{relCons}_{o_i}$ to obtain the cumulative error per operator. This error is summed up for every operator and averaged to obtain an average error $E$ for the entire model. The cumulative error for the model is represented by:

$$E = \frac{1}{n} \sum_{i=1}^{n} \frac{\text{diffPre}_{o_i} + \text{diffAdd}_{o_i} + \text{diffDel}_{o_i}}{\text{relCons}_{o_i}}$$

The obtained cumulative error is mentioned in the third column of Table 3. The relatively high error rates can be attributed to the fact that owing to the linear structure of the relevant predicate dictionary, there is little variation among the constraints produced specific to each operator, as well as their weights. A great deal of constraints are thus solved, thus introducing a great deal of noise in the reconstructed model. This noise can be illustrated with a snippet of the learnt model in the Listing 6. We conclude from these results that the ground truth model needs to be fine tuned and reworked upon to ensure that the operators being learnt are not semantically as close to ensure a better learning rate.

5.1 LSTM based sequence labeling

In this work, we explore two hyperparameters: the number of hidden units (set between (100, 200)), and the dropout
Learning robot speech models to predict speech acts in HRI

Table 2: Representation of the temporal rules mined with Titarl algorithm.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Confidence (%)</th>
<th>Support (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ask → say</td>
<td>73</td>
<td>96</td>
</tr>
<tr>
<td>inform → deny</td>
<td>69</td>
<td>100</td>
</tr>
<tr>
<td>autoFeedback → ask</td>
<td>100</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 3: Cumulative error in model learning (* = with MaxWalkSat, ** = with MaxSatSolver).

<table>
<thead>
<tr>
<th>Variables</th>
<th>Clauses</th>
<th>Model Error (E)</th>
<th>Execution Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>759</td>
<td>3606</td>
<td>41.78*, 39.78**</td>
<td>36.44*, 36.25**</td>
</tr>
</tbody>
</table>

The batch size is set to batchLen. We also use categorical cross entropy for the loss function and an adam optimizer (gradient descent optimizer). Each of the 10 sequences consists of about 400 dialogues each, which is divided using five-fold cross validation. Every training example is presented to the network 10 times i.e. the network is trained for 10 epochs. The results are summarized in the Figure 8 and are the obtained with the encoded behavioral model. The accuracy represented in the figure is the validation accuracy (accuracy measured on the validation set), which is the proportion of examples for which the model produces the correct output. It is represented as the fraction of examples classified correctly. The accuracy is recorded for 1, 2, 5 and 10 traces. As can be seen, the accuracy improves with the number of traces. If we consider the learning performance with the encoding scheme sans model in Figure 8, this prediction accuracy is not so impressive. The highest accuracy obtained is in the range of the upper forties. This demonstrates the fact that the presence of the behavioral model in the feature vector boosts the sequence labeling capacity of the learning system.

6 Conclusion

In order to be acceptable and able to “camouflage” into their physio-social context in the long run, robots need to be not just functional, but autonomously psycho-affective as well. This motivates a long term necessity of introducing behavioral autonomy in robots, such that they can autonomously communicate with humans without the need of “wizard” intervention. This paper proposes a technique to learn robot speech models from human-robot dialog exchanges. It views the entire exchange in the Automated Planning (AP) paradigm, representing the dialog sequences (speech acts) in the form of action sequences that modify the state of the world upon execution, gradually propelling the state to a desired goal. We then exploit intra-action and inter-action dependencies, encoding them in the form of constraints. We attempt to satisfy these constraints using a weighted maximum satisfiability model known as MAX-SAT, and convert the solution into a speech model. This model can be put to many uses, such as planning of fresh dialogs, automatic generation of dialogs etc. In this study, the learnt model is used to predict speech acts in the dialog sequences using the sequence labeling (predicting future acts based on previously seen ones) capabilities of the LSTM (Long Short Term Memory) class of recurrent neural networks. While this work represents a first step in this direction of behavioral autonomy via Automated Planning, it leaves a major scope of improvement as well. While this study has focused on learning speech models, true behavioral autonomy can only be achieved using behavioral models (models which consist of speech acts interleaved with co-verbal gestures). This speech model thus needs to be enriched with gestural information in order to render it complete and realistic. The planning for fresh dialog sequences can only be obtained with a behavioral model under the hood. Devising techniques for further reduction of the learning error between

Figure 7: Difference between symbolic time series and symbolic time sequences. While items in a symbolic time series are partially ordered and do not repeat, items in a symbolic time sequence may be co-occurring and repeat as well [29].

Figure 8: LSTM operator labelling accuracy (128 hidden units, 0.8 dropout rate).
the learnt model and ground truth model, and exploring the interleaving between physical and speech acts is in the scope of future work.

References

[27] H. Mannila, H. Toivonen, A. I. Verkamo, Discovery of frequent episodes in event sequences, Data Mining and Knowledge Discovery, 1999, 3(3), 259-289