Research Article **Open Access**

Mohammad Ali Zamani*, Sven Magg, Cornelius Weber, Di Fu, and Stefan Wermter

Deep reinforcement learning using compositional representations for performing instructions

https://doi.org/10.1515/pjbr-2018-0026
Received January 31, 2018; accepted October 30, 2018

Abstract: Spoken language is one of the most efficient ways to instruct robots about performing domestic tasks. However, the state of the environment has to be considered to plan and execute actions successfully. We propose a system that learns to recognise the user's intention and map it to a goal. A reinforcement learning (RL) system then generates a sequence of actions toward this goal considering the state of the environment. A novel contribution in this paper is the use of symbolic representations for both input and output of a neural Deep Q-network (DQN), which enables it to be used in a hybrid system. To show the effectiveness of our approach, the Tell-Me-Dave corpus is used to train an intention detection model and in a second step an RL agent generates the sequences of actions towards the detected objective, represented by a set of state predicates. We show that the system can successfully recognise command sequences from this corpus as well as train the deep-RL network with symbolic input. We further show that the performance can be significantly increased by exploiting the symbolic representation to generate intermediate rewards.

Keywords: deep reinforcement learning, spoken language instruction

***Corresponding Author: Mohammad Ali Zamani:** Knowledge Technology, Department of Informatics, University of Hamburg, Vogt-Koelln-Str. 30, 22527 Hamburg, Germany;
E-mail: zamani@informatik.uni-hamburg.de
Sven Magg, Cornelius Weber, Stefan Wermter: Knowledge Technology, Department of Informatics, University of Hamburg, Vogt-Koelln-Str. 30, 22527 Hamburg, Germany;
E-mail: {surename}@informatik.uni-hamburg.de
Webpage: http://www.informatik.uni-hamburg.de/WTM/
Di Fu: CAS Key Laboratory of Behavioral Science, Chinese Academy of Sciences, Beijing, China;
Department of Psychology, University of Chinese Academy of Sciences, Beijing, China;
Knowledge Technology, Department of Informatics, University of Hamburg, Vogt-Koelln-Str. 30, 22527 Hamburg, Germany;
E-mail: fud@psych.ac.cn

1 Introduction

In the future, robots are expected to work as companions with humans in various areas ranging from domestic to care-giving scenarios. Even with well-engineered robots, it would be unrealistic to transfer them directly from factories to home environments to perform complex tasks such as caregiving [1], [2]. One of the main reasons is safety [3]. Moreover, the robots also have to adapt to new environments to perform the given tasks. Hence, we need adaptive learning algorithms that can be "programmed" by the users easily.

Spoken language can be considered as one of the most effective communication channels to instruct robots to perform a sequence of actions to fulfill a task. Assigning tasks to robots by verbal instructions has been studied for example in [4], [5] and [6] but the problem had to be limited to small domains due to the variability in language and the corresponding problem to understand the human's intention.

Intention detection in spoken language has been studied to some extent on short texts [7] with a focus on applications like web search. Related research was started in the Facebook DeepText project [8] inspired by Collobert et al. [9]. In the DeepText project, all the sentences "I need a ride", "Take a cab" , and "But, I need to take a taxi" are interpreted as `Request a ride` which reflects the user intention in a unique phrase.

Deep neural networks have achieved a significant improvement in recent years especially in domains like machine translation [10] and can also be used for intention classification. The encoder-decoder architecture enables training different lengths of sequences. One of the most successful implementations [11] used Long Short-Term Memory (LSTM) for both encoder and decoder [12].

Such an end-to-end approach for translation could be applied to map the spoken utterance (sequence of words) into a sequence of actions. The input and output consist of the sequences of text and actions respectively. The proposed approach should be able to map the sentence "Put the mug into the microwave" into a sequence of low-level robot instructions like `Move-to Mug, Grasp`

`Mug`, `Move-to Microwave`, `Open Microwave`, and `Put Mug in Microwave`. Although it seems similar to a machine translation task, due to the high variability of both language and actions that lead to the accomplishment of the objective, the amount of training data and training time needed makes this approach currently not feasible. It is made even more complex since the current state of the environment has to be taken into account as well.

Reinforcement learning has proven to be an effective method to learn a task through interaction with the environment [13] and has been used in human-robot interaction (HRI) scenarios. In the RL framework, an agent interacts with the environment and receives rewards. The overall goal of the agent is to maximize the expected return (which is a kind of accumulation of future rewards), often to reach a terminal state. To improve RL in an HRI context, Thomaz et al. [14] introduced a feedback message from the human that can be used in the interactive reinforcement learning (IRL) framework, either as guidance for action selection or as reward for evaluating the selected action. Thomaz and Breazeal [15] implemented IRL in a simulated environment called "Sophie's Kitchen" in which the robot agent learned to bake a cake. The simulation allows the human to click on the object to guide the robot and also give a reward through clicking on a sliding bar [15]. However, the scenario was only designed for one fixed intention in order to limit the state space and thus the size of the Q-table.

Using deep convolutional neural networks as value function approximators [16] boosts RL to gain more powerful generalization capabilities for larger state spaces. Mnih et al. [16] implemented deep reinforcement learning that could surpass human performance in many Atari-2600 games by reading only raw pixels from the screen. Deep Q-networks have also been implemented for text-based games [17]. The agent in the game environment receives a description of the situation in a few sentences which are encoded by an LSTM to represent the change in the game state. Then, a multi-layered neural network is used to score possible actions in the given state. Although the agent learns to perform the given quest, it does not use the environment's state representation directly but builds a representation of the instruction instead.

To successfully perform a planning task (e.g. boiling water in a kitchen), we need to have a more interpretable representation of the environment. For example, in the list of the Atari 2600 games, "Montezuma's Revenge" has the lowest score performed by the deep Q-network [16]. The reason is that the agent requires to obtain a complex strategy to complete the game which cannot be obtained from past n visual frames like in the "Breakout" game. Moreover, it also provides sparse reward signals during the game.

A compositional representation for the environment's state on a more abstract level can help to solve the task. This representation can be obtained from the vision modality. For instance, Kumar and Oates [18] trained a deep convolutional neural network to represent a symbolic description of the given image. Garnelo et al. [19] implemented a system which extracts a symbolic representation from the environment. However, their action-value function used for RL was learned through discrete Q-learning which is not extendable to complex problems.

A compositional representation enables hybrid systems that also include classic planning and deduction systems and, on the other hand, helps us to easily generate intermediate rewards for the agent because of compositionality in the state representation. The objective of our research is to develop a system that understands the intention of an utterance and uses it to generate a sensible sequence of actions. Therefore, we separated the problem into two components, intention detection and action planning. We propose a novel deep reinforcement learning (DRL) architecture that directly learns from the compositional representation. In the results section, we show the feasibility of the approach for two tasks in a kitchen scenario.

2 Corpora for instructing robots

There are only a few annotated corpora available for instructing robots, which are large enough to be used for supervised learning. The Human Robot Interaction Corpus (HuRIC) was introduced by Bastianelli et al. [20] for the home service domain. Given sentences like "Could you please move to the sink and open the cold water" are assigned to action frames like "motion" and "change operational state". The corpus consists of only 302 sentences annotated with part-of-speech tags, lemmatized tokens, dependency trees, frame semantics and frame elements. However, no particular environment is assumed for this corpus. Moreover, the size of the corpus is not big enough to train a large network.

A corpus for natural language instruction to a robot is the "Tell-Me-Dave" corpus [21], [22]. This corpus has 20 particular environments including 10 different kitchens and 10 different living rooms (with a complete symbolic description of the objects' position, orientation and their states). In the VEIL-500 (Verb-Environment-Instruction-Library) version of the corpus, the human users ask the

robot to perform specific tasks, i.e. providing sentences for 10 objectives (`boiling water`, `cleaning the kitchen`, etc.). The total vocabulary size in the Tell-Me-Dave corpus is 687 and there are 469 valid samples in the VEIL-500 version of the corpus. The "Tell-Me-Dave" corpus is the largest language corpus annotated for action planning to the best of our knowledge. The environments in the corpus are designed similar to a home environment and are thus suitable for our work and future extensions.

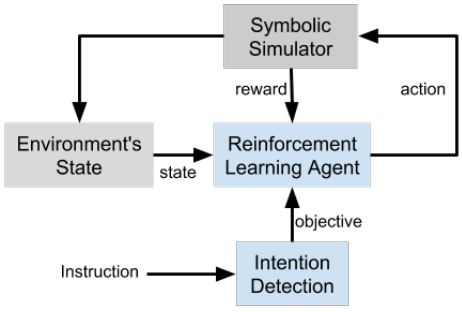

Figure 1: The modular approach using intention detection and reinforcement learning trained for each objective to generate the sequence of actions.

3 Method

We propose a modular approach that is divided into subsystems (Figure 1) with measurable sub-goals including intention detection and reinforcement learning (action planning). In the case of a significant change of the environmental setup, it is thus not necessary to retrain all modules, but only the RL agent has to be updated. Unlike supervised learning, the RL agent does not need any corresponding pair of instructions and actions. Instead, it learns through interaction with the environment towards a goal and receives sparse feedback which consists only of a positive scalar number for completing the task (or subtasks).

In the current format different RL agents can be trained for each intention (e.g. boil water, clean the room) or hierarchical RL could be applied where a high-level learner modifies the context or goal of a low-level learner [23]. Therefore, a classifier detects the intention of the given instruction and triggers the associated RL agent to generate the sequence of actions.

The intention detection and RL agents are trained separately. The reason is that they are from different domains and there are few datasets or simulators available that bring instructions and action planning together in one framework. For instance, simulators such as AI2thor [24] for high-level action planning do not contain any language instructions corpus. In case of a mismatch between the user's intention and the produced sequence of actions, online negative feedback from the user can be incorporated into the system to modify/stop the RL agent since it was generating a wrong action sequence. For now, feedback of the user is ignored to focus on the effects of the autonomous sub-goal reward calculation (see section 3.3.3).

3.1 Intention detection

The Tell-Me-Dave corpus already provides the ground truth objectives of the instructions, which we can use to extract a goal for reinforcement learning. With this goal, our intention detection module learns the objective associated with the given instruction. After learning, this module is used to call the corresponding RL agent to generate the correct action sequence. A given instruction was represented as a binary vector (of size 687 of the vocabulary) which indicated whether each word was present in the given instruction. A Multi-Layer Perceptron (MLP) with one hidden layer was used to classify among 10 existing objectives in the corpus (see Figure 2). In the proposed architecture, the hidden layer has 50 nodes with ReLU [25] as activation function and 20% dropout. The output layer includes 10 nodes equal to our classes with softmax as the activation function. As loss function, Categorical Cross-entropy minimized by an Adam optimizer was used as introduced in [26].

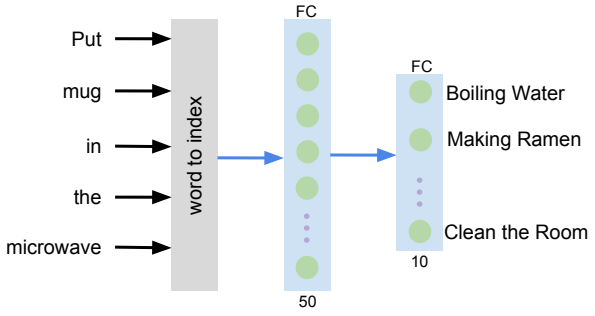

Figure 2: An MLP neural network is used to implement the intention detection. The Tell-Me-Dave corpus is used to train the neural network.

3.2 Compositional representation

Given the objective, the corresponding RL agent performs the action sequence. The RL agent receives the compositional representation of the environment consisting of predicates that describe a spatial relation between two objects, e.g. `(On Mug Sink)`, or the state of an object, e.g. `(State Mug Water)`. Combinations of these predicates, regardless of their order, describe the state of the environment. This can lead to a huge number of possible combinations of the predicates in the state representation which would make a Q-table an intractable model. The number of the predicates describing the current state is also not constant, not only for different environments but also during the performance of a specific task. Moreover, we are seeking to see a generalization with combinations of predicates that are seen by the agent for the first time.

Formally, the symbolic information of the environment is given as

$$\chi_t = \{x_1, x_2, \ldots, x_k\} \tag{1}$$

where χ is the set of predicates which represent the state of the environment, t is the time step, and k is the maximum number of existing predicates in each time step. Each predicate x_i is a member of all possible predicates

$$\Pi = \{x_1,\ x_2,\ \ldots,\ x_d\} \tag{2}$$

where d is the total number of possible predicates which in our case is 108. We convert the symbolic state $\chi(t)$ to a binary vector of fixed length (see Appendix)

$$s_t = [q_1,\ \ldots,\ q_d],\ where \underset{i\in\{1,\ldots,d\}}{q_i} = \begin{cases} 1, & if\ x_i \in \chi_t \\ 0, & if\ x_i \notin \chi_t \end{cases} \tag{3}$$

We developed a symbolic simulator based on the domain knowledge (i.e. Planning Domain Definition Language (PDDL) file) in the Tell-Me-Dave corpus to train the RL agent. PDDL is the standard to represent an action planning task [27]. It defines preconditions for each action as well as the effects on the environment. We developed the symbolic simulator on the basis of this description based on the OpenAI gym environment [28].

3.3 Deep reinforcement learning

The compositional state representation, s_t, is the input of our RL agent and the output units will estimate state-action values (Q) on performing the actions. Similar to the approach presented in Narasimhan et al. [17], outputs are grouped into actions and objects. For our scenario, we need to add two more output groups for a possible second object and a preposition which is necessary for some actions (i.e. for the action `keep` in `(keep mug on sink)`). The output of the network is thus divided into four groups of outputs as *action*, *object 1*, *object 2* and *preposition*. This indicates that there are four state-action values (Q_i , i = 1, 2, 3, 4) in our problem.

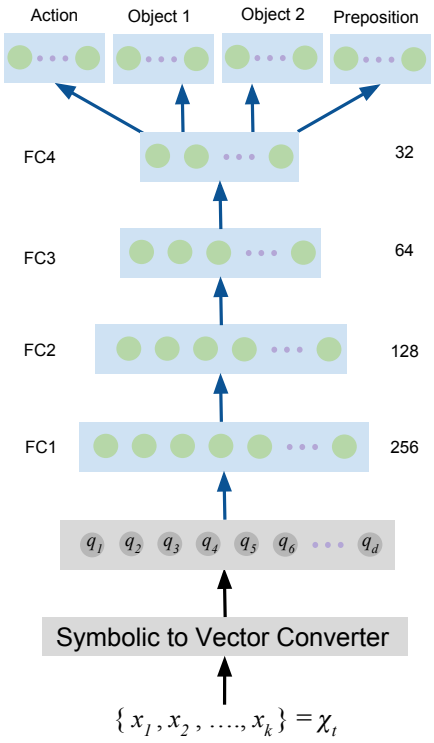

Figure 3: The deep reinforcement learning architecture. An MLP neural network is trained to approximate the action-value functions. The compositional linguistic state, χ_t, is presented to the network as a compositional vector s_t (see Section 3.2).

3.3.1 Network architecture

We do not require convolution as in LeCun et al. [10] nor an LSTM as in Narasimhan et al. [17], because the input of the RL agent consists of an abstract state representation instead of raw image pixels or text sentence. Since the model has the Markov Property (see [13]) the state is already adequate to fully represent the environment. Therefore, a fully-connected feedforrward MLP architecture is used to approximate the action-value (Q) function, which is presented in Figure 3.

We investigated the effect of number of layers in the RL agent but did not do an extensive search for optimal hyper parameters of the network. Our architecture with 4 hidden layers is depicted in Figure 3 which is referred to

as *DRL4*. Following the same naming scheme, we tested also *DRL3*, *DRL2*, and *DRL1*, where *DRLx* consists of x fully connected hidden layers with *FCx* connected to the output layer. For example, *DRL2* has only two hidden layers *FC1* and *FC2*, and *FC2* is connected to the output layer. We expected more capabilities of a network that has more layers. The hidden layers' activation functions are ReLU, and the output layers' activation functions are linear. The other parameters of the network like learning rate and number of units in each layer were found empirically.

3.3.2 Target network

The agent selects an action based on an epsilon-greedy policy ($\epsilon = 0.1$) and the observed experience was recorded as $e_t = \{s_t,\ a_t,\ r_t,\ s_{t+1}\}$ in memory. The memory recorded the last 100,000 (= N) experiences. Meanwhile, in each time step, a batch of experiences (32 in our experiment) was replayed to train the network (θ) to avoid teaching highly correlated samples in a row, as introduced by [16]. Besides the experience replay, to gain more stability, a duplicate neural network which is called "target network" ($\bar{\theta}$) [16] is used to estimate the next state value (see section 3.3.5). The target network is updated every 5,000 iterations from the trainable network (i.e. θ).

3.3.3 Reward

We implemented two approaches for obtaining rewards from the environment. First, the agent got reward only when it reached the goal state (i.e. reward of 1). In the second approach, the agent received an intermediate reward (0.1) when it found a predicate that is present in the goal state only for the first time in every episode (each episode had a maximum of 500 iterations in our implementation). If the goal state had k predicates, the agent was able to receive a maximum of $k-1$ intermediate rewards (regardless of required steps) plus one terminal reward (see Table 1). For all other steps, the agent received no reward. In both rewarding systems, the agent receives a small negative reward (-0.01) every step to be encouraged to accomplish the task faster.

3.3.4 Loss function

The loss function for the deep RL is defined as introduced in [16]:

$$L_i(\theta) = \mathbb{E}_{e\sim\sigma(E)}\left[\left(r_t + \gamma V_i(s_{t+1}) - Q_i(s_t, a_t; \theta)\right)^2\right] \qquad (4)$$

where e represents samples of experience which are drawn from the experience replay E (see Section 3.3.6). γ is the discount factor (0.9), r is the reward that the agent receives from the environment and $V_i(s_{t+1})$ is the estimated next-state value. $L_i(\theta)$ is the loss function corresponding to ith Q. The output of the network consists of a maximum of 4 groups ($action$, $object_1$, $object_2$, $preposition$), which represent state-action values (i.e. Q_i). The Q value is not used or updated when the output group was not applicable (e.g. `(grasp, kettle)` needs only two outputs). It means that the gradient signal for those groups is masked and does not backpropagate through them.

The relevant action arguments (i.e. $object_1$, $object_2$ and $preposition$) for each individual *action* is thereby predefined. Each action has a fixed relation with its corresponding arguments. Therefore, it was not necessary to learn these relations in our approach.

Moreover, similar to the standard procedure in the DQN, within each applicable output group, only one of the output nodes is used, because the network is trained to estimate the state-action value for one particular state and action each time (i.e. $Q(s_t, a_t; \theta)$). Therefore, two masking layers are necessary for the gradient signal.

3.3.5 Estimation of the next state value

In the original Deep Q-Network [16], the next state value is estimated by

$$V(s_{t+1}) = \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \bar{\theta}) \qquad (5)$$

which is not directly applicable to our variable action groups. Depending on the current state and selected action, some of the action groups are not available. For example, considering `(grasp, kettle)` as the selected action means only two Q-values are selected to be updated. However, the next state might have more or less the same number of available action groups compared with the current one. So, we propose two alternatives: (a) *global* and (b) *local* estimation of the next state value.

A *global* estimation of the state value is the maximum value among all Q-values of all action groups:

$$V_i(s_{t+1}) = \max_{j,\ j=1,\ldots,M}\left(\max_{a_{t+1}} Q_j(s_{t+1}, a_{t+1}; \bar{\theta})\right) \qquad (6)$$

Table 1: The reward function for the optimal sequence of the boiling water objective. The environment with only-terminal reward (OTR) does not provide any reward until the terminal state. However, the agent with the intermediate reward receives a small reward whenever the agent achieves predicates from the terminal state (S8).

ID	State	Action	IR	OTR
S0	{(on, kettle, sink)}	(move_to, sink)	0	0
S1	{(on, kettle, sink), (near, robot, sink)}	(turn_sink_knob)	0.1	0
S2	{(on, kettle, sink), (near, robot, sink), (state, sink_knob, tap_on), (state, kettle, water)}	(turn_sink_knob)	0	0
S3	{(on, kettle, sink), (near, robot, sink), (state, kettle, water)}	(move_to, kettle)	0	0
S4	{(on, kettle, sink), (near, robot, kettle), (state, kettle, water)}	(grasp, kettle)	0	0
S5	{(grasping, robot, kettle), (state, kettle, water)}	(move_to, stove)	0.1	0
S6	{(near, robot, stove), (grasping, robot, kettle), (state, kettle, water)}	(keep, kettle, stovefire2, on)	0.1	0
S7	{(on, kettle, stovefire2), (near, robot, stove), (state, kettle, water)}	(turn_stove_fire2)	1	1
S8	{(state, stove, stovefire2), (on, kettle, stovefire2), (near, robot, stove), (state, kettle, water)}	-		

As a result,

$$V_i(s_{t+1}) = V_j(s_{t+1}),\ for\ all\ i, j \in \{1, \ldots, M\} \tag{7}$$

where M is the number of used output groups for the experience (this is variable depending on the action, e.g., in *(grasp, cup)*, $M = 2$ and in *(keep, kettle, stove, on)*, $M = 4$). Since, in the global estimation, the value is chosen regardless of availability in the next state, this can lead to an overestimation. Over-estimation can be compensated by applying methods such as a Double Deep Q-Network which is extensively discussed in [29].

A *local* estimation of the state value is the maximum Q-value within each action group

$$V_i(s_{t+1}) = \max_{a_{t+1}} Q_i(s_{t+1}, a_{t+1}; \bar{\theta})\ \ (for\ i = 1, \ldots, M) \tag{8}$$

which means the estimated value of the state in local Q is less than the global one.

$$\max_{a_{t+1}} Q_i(s_{t+1}, a_{t+1}; \bar{\theta}) \le \max_j \left(\max_{a_{t+1}} Q_j(s_{t+1}, a_{t+1}; \bar{\theta}) \right) \quad (for\ i, j = 1, \ldots, M) \tag{9}$$

The local estimation is restricted to Q-values within each action group, and therefore, it might underestimate the state value.

There is a third way to estimate the value of the state by finding the maximum value among all available actions in the given state. However, this needs a complete knowledge of the problem, which is not directly available in most cases. Moreover, the DQN is a model-free approach.

3.3.6 Memory

In our RL problem, an agent receives sparse rewards during the learning process. A random batch sampling from the memory is not efficient to train such an agent (we call it Uniform Experience Replay (UER) in this paper). Therefore, we applied rank-based prioritized experience replay (PER) [30]. In this approach, samples are ranked based on the Temporal Difference (TD) error. Due to multiple outputs in our model, the TD-error was averaged as

$$\delta_{e_t} = \frac{1}{M} \sum_{m=1}^{M} \left| r_t + \gamma V_m(s_{t+1}) - Q_m(s_t, a_t; \theta) \right| \tag{10}$$

where δ_{e_t} is the TD-error for the experience (e_t). Similar to Schaul et al. [30], the priority of the sample was

$$p(e_t) = \frac{1}{rank(e_t)} \tag{11}$$

and the probability of selecting the sample thus

$$P(e_t) = \frac{p(e_t)^\alpha}{\sum_l p(e_l)^\alpha} \tag{12}$$

where α, the prioritization factor, was 0.7 as recommended by [30]. After each sampling, the probability of each experience was updated in the memory. For an efficient sampling, the memory was also implemented as a binary heap, which was rebalanced every 1000 iterations.

We also implemented importance sampling weights recommended by Schaul et al. [30] to correct the bias caused by the PER, since it violates the distribution of experiences as they are normally expected. Therefore, a weight is assigned to each experiment in the sampled batch as introduced in [30]

$$w_{e_t} = \left(\frac{1}{N \cdot P(e_t)} \right)^\beta \tag{13}$$

where β adjusts the weight, ranging from equal weights ($\beta = 0$) to a full compensation of the non-uniform probabilities ($\beta = 1$). The weights are normalized by scaling them with

$$\frac{1}{max(w_{e_t})},\ e_t \in E_b \text{ and } E_b \sim E \tag{14}$$

where e_t is among the batch samples (E_b) which are sampled (based on the priority) from the experience replay (E). So, the weight can be rewritten as

$$w_{e_t} = \frac{\left(N \cdot P(e_t)\right)^{-\beta}}{\max_{e_t \in E_b} \left(\left(N \cdot P(e_t)\right)^{-\beta}\right)}, \; E_b \sim E \tag{15}$$

As recommended in [30], we also linearly increase β from $\beta_0 = 0.5$ to 1 at the end of learning.

4 Results and discussions

4.1 Intention detection

The intention detection network was first trained for the Tell-Me-Dave corpus. The results of a 5-fold cross validation show (see Table 2) that the objectives of the instructions can be classified accurately on average in 89.57% of the cases from the test set. An early stopping criterion was used by monitoring the loss function on a validation set of size 5% of the training set each time. A single layer perceptron (i.e. no hidden units) was selected as the baseline which could obtain 85.74% accuracy. The current intention detection mechanism could also be implemented with a non-neural method such as keyword spotting. However, the proposed architecture can more easily be extended to also classify not only objectives but also feedback and warning from a human in the future.

Table 2: Intention classification results. The numbers show the mean and standard deviation percentage.

	Training Acc.	Validation Acc.	Test Acc.
MLP	99.73 ± 0.18	99.37 ± 1.39	89.57 ± 4.27
Perceptron	97.44 ± 0.07	96.87 ± 4.42	85.74 ± 3.91

The confusion matrix (Figure 4) shows that the most confusion happened between `Prepare for party`, `Preparing for the study night`, and `Clean the room` which use many common words. For example, two similar sample instructions from these two classes are: "Take all bottles, cans, and trash and put them in the trash can ..." as the "Preparing for the study night" intention and "Put beer and chips in trash can" as the `Clean the room` intention.

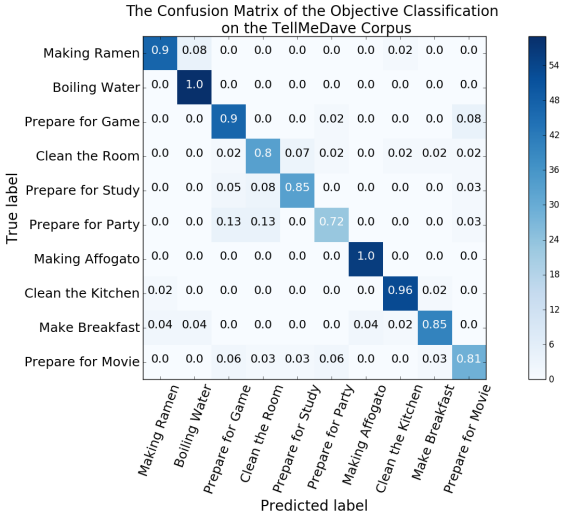

Figure 4: Confusion matrix for the classification of objectives from the Tell-Me-Dave corpus.

Table 3: Different factors and their alternatives which are used to create a DRL agent.

Factor	Alternatives
Reward Type	Intermediate Reward
	Only Terminal Reward
$V(s_{t+1})$ Estimation	Global
	Local
$V(s_{t+1})$ Source	Target Network
	Trainable Network
Network Architecture	*DRL1*
	DRL2
	DRL3
	DRL4
Replay Memory	Uniform Replay Memory
	Prioritized Replay Memory

4.2 Reinforcement learning agent

We used two criteria to measure the performance of our presented approach: success rate and the learning time (i.e. number of iterations) for the given tasks. The success rate indicates how many times the instances of the agent were successful in finding the optimal policy. The learning time criterion is the total number of iterations until the first success in the test mode (i.e. no exploration), which indicates how fast our agent can accomplish the task. Both criteria are based on the test episodes, which means the agent generated the sequence of actions without exploration.

Of course, the learning speed represents those instances that did not fail to find the optimal policy. Every 5 episodes, there was a test episode in which the agent's per-

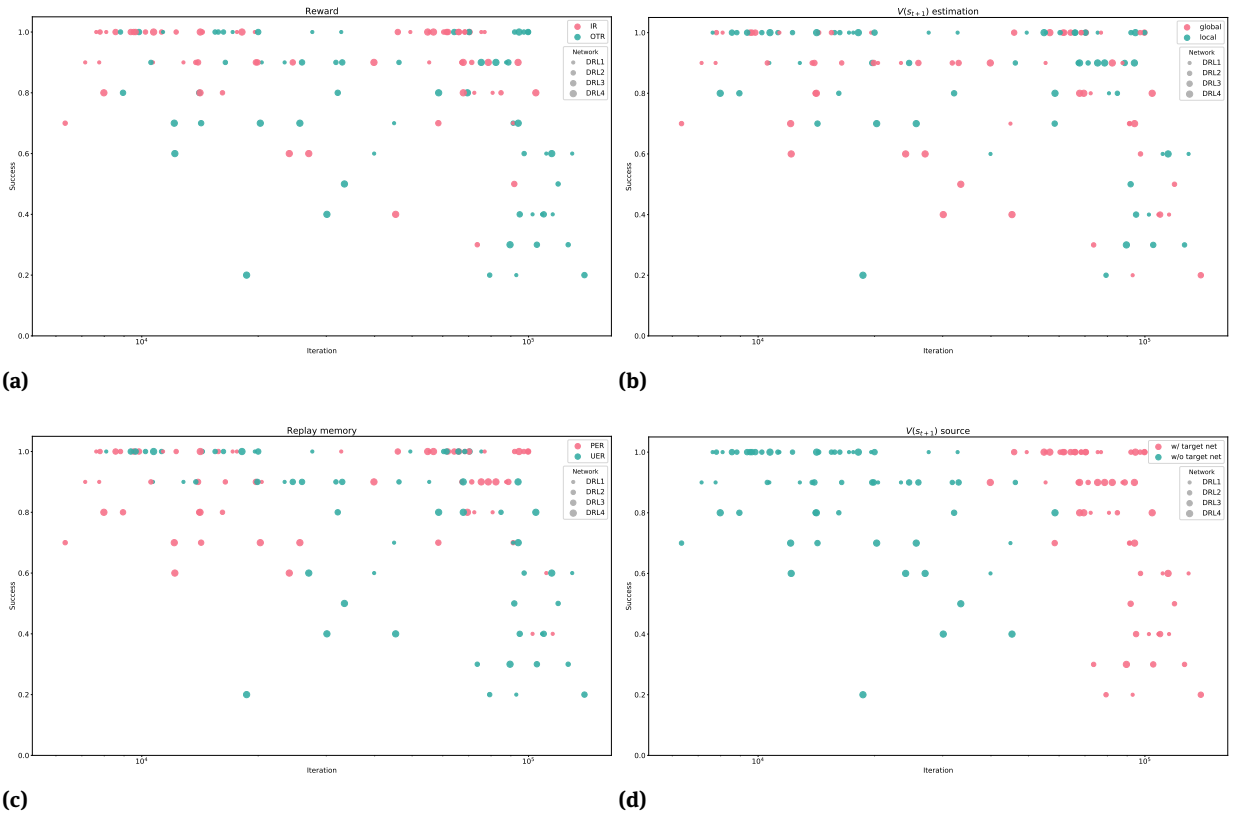

Figure 5: Scatter plot of the agents based on their success rate and number of iterations to learn the given tasks. Each circle in the figures represents an agent with a unique configuration and task which were trained 10 times. The y-axis shows the success rate of the agent out of 10 trials and the x-axis (log scale) shows the first iteration which agent could solve the task in the test mode (no exploration). The dots at the top-left corner in each plot represent the agents with the best performance. The agents are separated based on (a) reward type, (b) $V(s_{t+1})$ estimation, (c) replay memory, and (d) $V(s_{t+1})$ source. It can be seen that the results are independent of the size of the network (indicated by the size of the dots).

formance was tested (i.e. without exploration). It should be noted that each episode has a maximum of 500 iterations and the experiments are limited to 150K iterations (there is no limitation of the number of episodes). Therefore, the experiments with a better performance have a shorter duration within the episodes.

The deep RL performance was evaluated mainly for the two factors of reward type and estimation of the next state value (in short: $V(s_{t+1})$ estimation). However, other factors like different sampling methods from the replay memory, network architecture, estimating $V(s_{t+1})$ using either target or trainable network (short: $V(s_{t+1})$ source) were also investigated to understand the efficiency of our compositional state representation (see Table 3). We combined the presented five factors to create different RL agents to solve the two tasks of `boiling water` and `making ramen`. We built 64 (2×2×2×4×2) alternative RL agents. Each of the alternative agents had 10 trials to learn to solve the tasks. The goal state for the training was derived by executing the annotated minimal action sequence in the simulator. The RL network thus had to learn the action sequence that leads to this goal state.

The performance of all the alternative agents are depicted in Figure 5. The fastest RL agent with a 100% success rate has used the IR reward and the UER memory with local estimation of $V(s_{t+1})$ and trained on the *DRL2* network without the target network (see the first row of Table 4). We also searched for the largest combinations of factors which are essential to achieve a 100% success rate and found three combinations. Each of these three combinations represents 2 RL agents performing 2 tasks in 10 trials (see Table 4).

To understand the effect of the reward type and $V(s_{t+1})$ estimation, we present the results in Table 5 with permutation of the sampling method from the replay memory.

It can be seen that the agents which were trained with the intermediate reward type showed a better performance for the given objectives than the ones that learned with

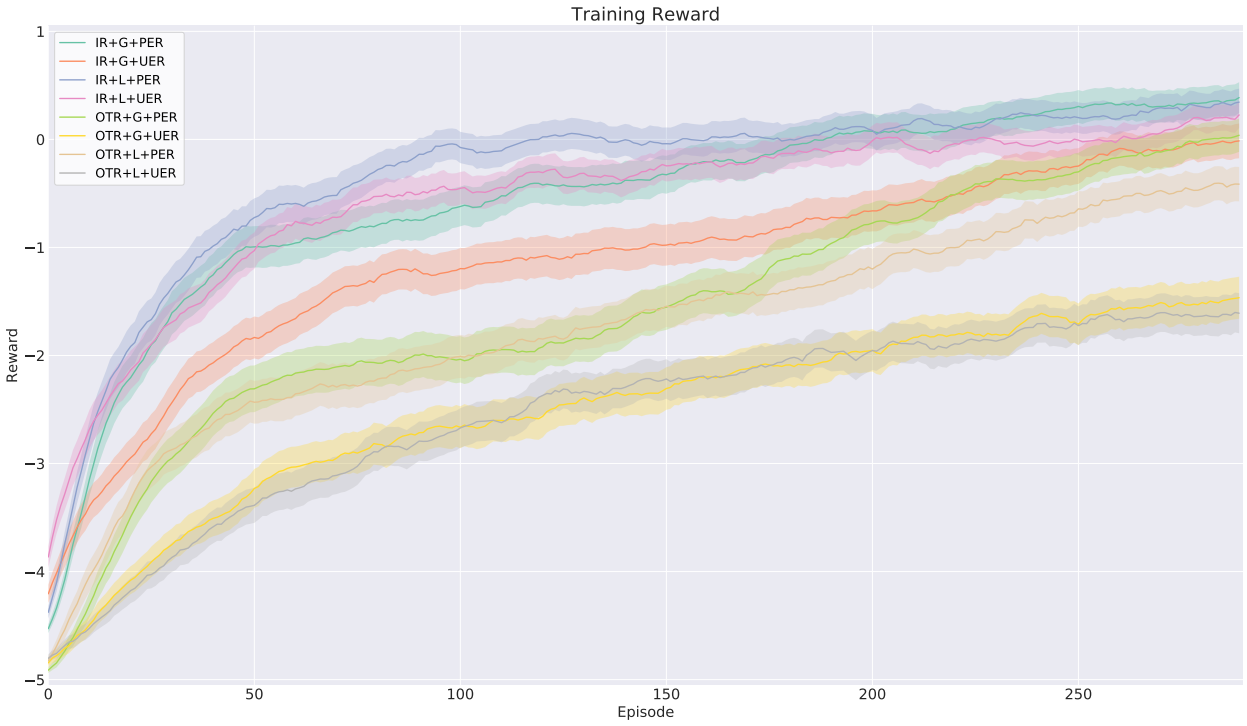

Figure 6: The collected reward in each episode during training. The graphs include only the first 300 episodes. The RL agents learn faster when they receive the intermediate reward. Moreover, prioritized experience replay outperforms the uniform experience replay. The curves are smoothed with a moving average with a sliding window of 10 episodes. The confidence interval shows the standard error and each curve is the average of 160 experiments (16 RL agents × 10 trials).

Figure 7: The collected reward by the RL agent during the test episodes. The agent performs best when it receives the intermediate reward with prioritized experience replay. The curves are smoothed with a moving average with a sliding window of 1000 iterations. The confidence interval shows the standard error and each curve is the average of 160 experiments (16 RL agents × 10 trials).

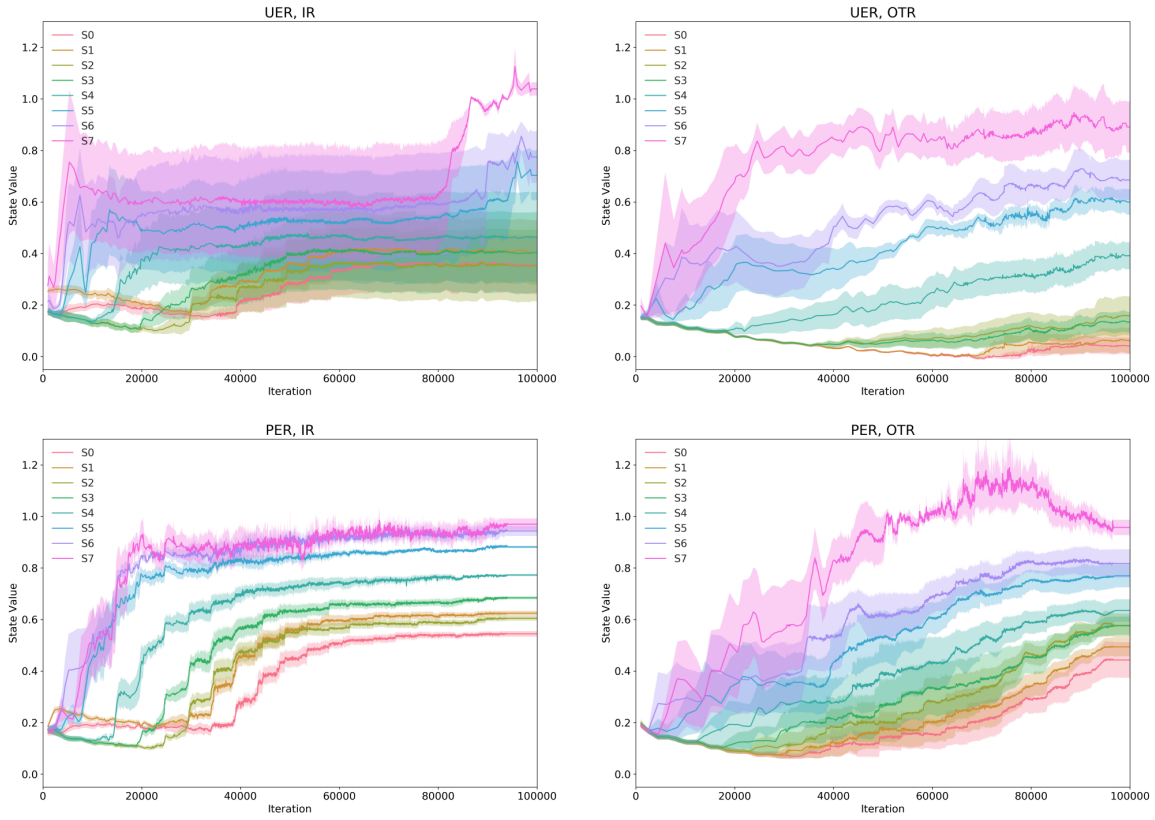

Figure 8: Estimated state values obtained by different methods for sequences of states required for the boiling water task (see Table 1). Each curve is the average of 5 trials. UER= Uniform Experience Replay, PER = Prioritized Experience Replay, IR= Intermediate Reward, OTR= Only-Terminal Reward. The *DRL4* was used as the network of the agent and $V(s_{t+1})$ estimation was global. The confidence interval shows the standard error.

only the terminal reward signal. This shows that using intermediate rewards improves the speed of learning for this scenario despite ignoring the necessary order of goal predicates in the reward process, e.g., putting the kettle on the stove without water leads to an intermediate reward. This means, no knowledge about the order of sub-tasks was used in the assignment of the intermediate rewards. Local estimation seems slightly better for our agents. It should be noted that using all actions (i.e. the global estimation) is the most common technique in the literature.

Figures 6 and 7 show that the intermediate rewards lead to faster learning compared with the OTR counterparts. They also show that the agent with the intermediate reward and the PER has the best performance among all other methods. We can observe that the $V(s_{t+1})$ estimation does not have significant effect in terms of collected reward.

Another benefit of intermediate reward is to obtain distinguishable well-ordered state values. This is visible in its best form in Figure 8 (PER+IR) with sorted values of the state clearly distinguishable from each other (i.e. more robust). However, in the case of only terminal reward, especially UER+OTR (see Figure 8), there is no proper and clear order of the state values.

The global estimation could cause overestimation of the state values as depicted in Figure 8. The S7 state, which is the just before the terminal state, should converge to the value of 1 according to the terminal reward presented in Table 1. In the UER+IR and PER+OTR experiments, it slightly overshoots from the true value, but later converges to 1. As can be seen in Table 5, the global estimation (even with the side effect of over-estimation) is more efficient than the local estimation of the state value. Therefore, we show the performance of the agent only with the global state value estimation in the figures.

We also investigated the performance gains of using the prioritized experience replay method by comparing it to a standard random sampling of experiences. As table 5 shows, RL with a terminal reward signal as well as with intermediate rewards benefits clearly by prioritizing experiences.

Table 4: The first row of the table presents the fastest RL agent with 100% success rate. The next three rows show the largest combinations of factors (which is 2) that can create an agent with a 100% success rate.

Reward	Memory	$V(s_{t+1})$ estimation	Network	$V(s_{t+1})$ source	Success rate	Learning speed
IR	UER	Local	*DRL2*	Trainable Target	1.00	$10.73K \pm 5.12K$
IR/OTR	PER	Local	*DRL1*	Trainable Network	1.00	$22.69K \pm 5.56K$
OTR	PER	Local	*DRL2*	Target/Trainable Network	1.00	$53.57K \pm 23.98K$
IR	PER/UER	Global	*DRL3*	Target Network	1.00	$61.39K \pm 6.23K$

Table 5: Performance of the agent during the first 150K iterations. UER= Uniform Experience Replay, PER = Prioritized Experience Replay, IR= Intermediate Reward, OTR= Only Terminal Reward, G= Global and L= Local Estimation of the state value. Each row represents 160 experiments (16 RL agents × 10 trials).

Approach	Success (%)	Learning iteration (x1K)
IR + G + PER	90.63 ± 3.09	36.15 ± 6.85
IR + G + UER	83.13 ± 5.60	47.28 ± 7.90
IR + L + PER	90.00 ± 3.16	39.57 ± 7.24
IR + L + UER	92.50 ± 3.23	43.21 ± 8.26
OTR + G + PER	86.25 ± 3.64	50.95 ± 10.14
OTR + G + UER	57.50 ± 7.50	66.60 ± 10.66
OTR + L + PER	85.63 ± 4.74	56.86 ± 10.06
OTR + L + UER	56.25 ± 8.21	66.62 ± 10.30

We used Analysis of Variance (ANOVA) to test the significance of our results. Before the ANOVA analyses, we tested the distribution of success rate and learning speed results. The skewness of the success rate and learning speed was -1.266 and .441 respectively. The kurtosis of the success rate and learning speed was .593 and -1.071. Since the skewness and kurtosis were both below 1.00, the data of the success rate and learning speed in the current study are believed to be normally distributed. Therefore, to investigate the effect of all factors, 2 reward types (intermediate/only terminal) × 2 $V(s_{t+1})$ estimations (global/local) × 2 $V(s_{t+1})$ source (trainable/target network) × 2 replay memories (UER/PER) × 4 network architectures (*DRL1*, *DRL2*, *DRL3*, *DRL4*), mixed design ANOVAs were performed on the success and learning speed criteria. In the ANOVA analyses, architectures were used as the between-setups independent factor, and the remaining factors were within-setup independent factors.

4.2.1 Success rate

There were significant main effects of the reward type ($F = 36.38$, $p < .001$, $\eta_p^2 = .36$), replay memory ($F = 29.06$, $p < .001$, $\eta_p^2 = .31$), $V(s_{t+1})$ source ($F = 14.36$, $p < .001$, $\eta_p^2 = .18$) and network architecture ($F = 3.70$, $p < .05$, $\eta_p^2 = .15$). However, $V(s_{t+1})$ estimation was not significant ($F = .35$, $p = .559$, $\eta_p^2 = .01$).

The Bonferroni post hoc test indicated that the RL agents with the IR significantly outperformed the OTR (.891 vs .734, $p < .001$) and the agents were more efficient with the PER memory rather than UER (.891 vs .714, $p < .001$). Interestingly, the agents without target network were more successful (.881 vs .723, $p < .001$).

The interaction effect among the reward, memory and $V(s_{t+1})$ source was significant ($F = 7.41$, $p < .01$, $\eta_p^2 = .10$). The interaction among network architecture and $V(s_{t+1})$ source, memory and $V(s_{t+1})$ source, reward and $V(s_{t+1})$ source, and reward and memory were also significant ($ps < .01$).

Reward, Memory, and $V(s_{t+1})$ Source: The simple effect analysis of the interaction effect indicates that using the IR boosts the agents that use the UER memory with target network (+.494, $p < .001$) as well as without target network (+.125, $p < .05$). Moreover, the agent with the OTR and target network gains a significant success by choosing the PER (+.500, $p < .001$).

Reward and Memory: The agent which receives only the terminal reward is more successful with PER (+.291, $p < .001$). However, the agent with the IR does not gain a significant improvement by the PER memory $p = .548$. An explanation could be that each IR is presented only once in every episode, but not when a loop occurs (see section 3.3.3). This leads to a fluctuating TD error, and hence overly frequent selection of these states within the PER.

Network Architecture and $V(s_{t+1})$ Source: The simple analysis indicates the RL agents with shallower networks (*DRL1-DRL3*) work better without the target network ($ps < .01$), However, the *DRL4* agent performance increased by +.100 with the target network ($p = .09$).

4.2.2 Learning speed

The main effects of reward type ($F = 73.24$, $p < .001$, $\eta_p^2 = .54$), memory ($F = 23.74$, $p < .001$, $\eta_p^2 = .28$), and $V(s_{t+1})$ estimation ($F = 765.61$, $p < .001$, $\eta_p^2 = .93$) were significant. However, the main effect of $V(s_{t+1})$ source ($F = .64$, $p = .428$, $\eta_p^2 = .01$) and network architecture ($F = .93$, $p = .43$, $\eta_p^2 = .04$) were not significant. The Bonferroni post hoc test indicated that the agents with the IR (41.5K vs 61.7K $p < .001$), and the PER memory (45.9K vs 57.3K $p < .001$) were significantly faster on average. Interestingly, the agents without the target network were significantly faster in accomplishing the tasks (19.1K vs 84.1K $p < .001$). Using a global $V(s_{t+1})$ estimation leads to 1.9K iterations faster learning which is not significant. Also, it was interesting to see that the network architecture (i.e. number of hidden layers) was not significant for faster learning. The interaction effect between reward and memory ($F = 18.30$, $p < .01$, $\eta_p^2 = .23$), reward and the $V(s_{t+1})$ source ($F = 10.30$, $p < .01$, $\eta_p^2 = .14$) were significant.

Reward and Memory: The agents with the PER memory are on average 16.0K iterations ($p < .001$) faster when they are trained with the IR. When they use the UER memory, they are on average 24.2K iterations ($p < .001$) faster with the intermediate reward.

Reward and $V(s_{t+1})$ Source: The simple effect analysis indicates that the agents with the intermediate reward were significantly faster on average without the target network (14.2K vs 68.8K $p < .001$). Also, the agents with only terminal reward were significantly faster without the target network (23.9K vs 99.5K $p < .001$).

Overall, the analysis shows the effectiveness of the intermediate reward in both success and learning speed criteria. We also analyzed other factors and their interaction to build an agent which can learn faster and more robust. As a limitation of this analysis, each of the presented factors can depend on parameters which might affect the overall analysis. For example, the number of neurons in each layer for the network architecture, the update interval for the $V(s_{t+1})$ source, the re-balancing interval of the binary heap in the PER memory, and the adjustment of β in the PER memory (or alternatively, it can be dynamically adjusted with the approach introduced in [31]) can have effects on the results. However, our main focus was to investigate the effect of the reward type in combination of different factors and analysis of the other factors were not our primary goal.

5 Conclusions and future work

Our approach uses two steps to combine deep reinforcement learning with compositional representations: first, detect the objective of the given linguistic instruction and second, generate the sequence of actions based on the symbolic representation of that objective. In this paper, we presented a proof of concept to show the general utility of this hybrid approach. Furthermore, we show, by using a compositional representation, that not only large state spaces can be handled but also intermediate rewards can easily be determined, which boosts the RL learning performance.

The current implementation of the intention classification is only based on spotting keywords in the given sequence to obtain the objective. The performance could be improved if the input is represented using word embeddings [32] and using recurrent neural networks [33] because the instructions are given as word sequences. Our current model is trained based on a fixed set of objectives and each of them corresponded to a terminal predicate. Therefore, we plan to extend the model by accepting arguments in the objective if the human user prefers the task to be performed differently (i.e. with a different set of predicates). We also intend to not only classify objectives in the future, but also to extend the classifier to distinguish user feedback such as warnings and to incorporate this in the learning process (Interactive RL). With this extension, it would then be possible to replace the current goal state extraction through simulation with a learning system that learns which predicates are necessary to satisfy the user's intention.

The selected set of actions for our model can be extended depending on a robot's capabilities. In a real-life scenario, the abstract state representation enables the robot to internally plan for new tasks. Once the given task can be performed successfully in an internal symbolic simulation, the robot can perform the sequence of actions in the real scenario.

By splitting up the intention detection and the goal-directed learning, the presented RL framework only has to rely on the signal from the environment to understand whether the agent is in the terminal state. By using compositional representations at both ends of the Deep-Q Network, the input and output can be interpreted easily and the network also be used within a hybrid system. We showed, by using two objectives, that such a system can be effectively trained towards a simple task in a home scenario. Several objectives can be used by training different networks and selecting the correct one for the given task

after intention detection. The RL network also offers the complexity to be trained towards several goals and thus to combine several objectives within the same network. By doing this, common sequences to reach sub-goals can be efficiently exploited.

Our main finding was that large performance gains can be obtained by using the individual predicates of the symbolic goal state as intermediate rewards in the learning process. These intermediate rewards can be easily generated, as we have shown, by rewarding the appearance of individual predicates that need to be satisfied once when they are encountered first. Since no domain knowledge was necessary to order those predicates, our method provides a simple way to speed up learning when using compositional representations. Compared to other symbolic approaches, our model does not need any explicit internal representation of the world model. Moreover, the deep neural network, which approximates the action-value function, generalizes over the states to select an action. We thus showed that combining compositional representations with neural reinforcement learning not only makes it possible to implement a hybrid system but it also improves the performance by generating intermediate rewards which are readily available in this representation.

Acknowledgement: The authors gratefully acknowledge partial support from the German Research Foundation DFG under project CML (TRR 169) and the European Union under project SECURE (No 642667).

References

[1] S. Schaal, The new robotics – towards human-centered machines, HFSP Journal, 2007, 1(2), 115–126

[2] S. Schaal, C. G. Atkeson, Learning control in robotics, IEEE Robotics & Automation Magazine, 2010, 17(2), 20–29

[3] J. Peters, S. Schaal, Learning to control in operational space, The International Journal of Robotics Research, 2008, 27(2), 197–212

[4] S. Lauria, G. Bugmann, T. Kyriacou, E. Klein, Mobile robot programming using natural language, Robotics and Autonomous Systems, 2002, 38(3), 171–181

[5] S. Lauria, G. Bugmann, T. Kyriacou, J. Bos, E. Klein, Converting natural language route instructions into robot executable procedures, In: Proceedings of the 11th IEEE International Workshop on Robot and Human Interactive Communication, IEEE, 2002, 223–228

[6] T. Nishizawa, K. Kishita, Y. Takano, Y. Fujita, S. Yuta, Proposed system of unlocking potentially hazardous function of robot based on verbal communication, In: 2011 IEEE/SICE International Symposium on System Integration (SII), IEEE, 2011, 1208–1213

[7] W. Hua, Z. Wang, H. Wang, K. Zheng, X. Zhou, Short text understanding through lexical semantic analysis, In: 2015 IEEE 31st International Conference on Data Engineering (ICDE), IEEE, 2015, 495–506

[8] A. Abdulkader, A. Lakshmiratan, J. Zhang, Introducing DeepText: Facebook's text understanding engine, https://code.facebook.com/posts/181565595577955/introducing-deeptext-facebook-s-textunderstanding-engine [Accessed: 2018-01-30]

[9] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, Journal of Machine Learning Research, 2011, 12, 2493–2537

[10] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature, 2015, 521(7553), 436–444

[11] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, In: NIPS'14 Proceedings of the 27th International Conference on Neural Information Processing Systems, 2014, 2, 3104–3112

[12] S. Hochreiter, J. Schmidhuber, long short-term memory, Neural Computation, 1997, 9(8), 1735–1780

[13] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, vol.1, MIT Press Cambridge, 1998

[14] A. L. Thomaz, G. Hoffman, C. Breazeal, Real-time interactive reinforcement learning for robots, In: AAAI 2005 Workshop on Human Comprehensible Machine Learning, 2005

[15] A. L. Thomaz, C. Breazeal, Teachable robots: understanding human teaching behavior to build more effective robot learners, Artificial Intelligence, 2008, 172(6-7), 716–737

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, et al., Human-level control through deep reinforcement Learning, Nature, 2015, 518(7540), 529–533

[17] K. Narasimhan, T. Kulkarni, R. Barzilay, Language understanding for text-based games using deep reinforcement learning, In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, 2015, 1–11

[18] A. Kumar, T. Oates, Connecting deep neural networks with symbolic knowledge, In: The 2017 International Joint Conference on Neural Networks (IJCNN), May 2017, 3601–3608

[19] M. Garnelo, K. Arulkumaran, M. Shanahan, Towards deep symbolic reinforcement learning, arXiv:1609.05518, 2016

[20] E. Bastianelli, G. Castellucci, D. Croce, L. Iocchi, R. Basili, D. Nardi, HuRIC: a human robot interaction corpus, In: the Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14), Reykjavik, Iceland, 26-31 May, 2014, 4519–4526

[21] D. K. Misra, J. Sung, K. Lee, A. Saxena, Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions, The International Journal of Robotics Research, 2016, 35(1-3), 281–300

[22] D. K. Misra, K. Tao, P. Liang, A. Saxena, Environment-driven lexicon induction for high-level instructions, In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Beijing, China, July 26-31, 2015, 992–1002

[23] D. Rasmussen, A. Voelker, C. Eliasmith, A neural model of hierarchical reinforcement learning, PLOS ONE, 2017, 12(7), 1–39, https://doi.org/10.1371/journal.pone.0180234

[24] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, A. Farhadi, AI2-THOR: An interactive 3D environment for visual AI, arXiv:1712.05474, 2017

[25] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011), 2011, 315–323

[26] D. Kingma, J. Ba, Adam: a method for stochastic optimization, In: 3rd International Conference for Learning Representations, San Diego, 2015

[27] M. Ghallab, A. Howe, C. Knoblock, D. McDermott A. Ram, M. Veloso, et al., PDDL – The Planning Domain Definition Language, Technical Report TR-98-003, Yale Center for Computational Vision and Control, 1998

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, arXiv:1606.01540, 2016

[29] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, In: AAAI'16 Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016, 16, 2094–2100

[30] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, In: International Conference on Learning Representations (ICLR), May 2016

[31] M. Khamassi, G. Velentzas, T. Tsitsimis, C. Tzafestas, Active exploration and parameterized reinforcement learning applied to a simulated human-robot interaction task, In: 2017 First IEEE International Conference on Robotic Computing (IRC), April 2017, 28–35, 10.1109/IRC.2017.33

[32] J. Pennington, R. Socher, C. D. Manning, GloVe: global vectors for word representation, In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 2014, 1532–1543, ISSN 10495258

[33] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, In: NIPS 2014 Workshop on Deep Learning, December 2014

APPENDIX: Details of the compositional representation

This section is a brief explanation of the compositional representation which is used in this paper. Given $\chi_t = \{x_1, x_2, ..., x_k\}$ as the symbolic state, each predicate $x_i,\ i \in \{1, ..., k\}$ consists of three elements: *relation* (e.g. in), *object* (e.g. `kettle`) and *object/state* (e.g. `stove/ON`).

$$x_i = (re_l,\ ob_m,\ oc_n)$$

The relation (re_l) describes the relations between two objects. The overall *relation* set consists of 4 elements RE = `near, on, grasping, state` ($d_{RE} = 4$). The first two relations describe spatial relations, *grasping* is used only for the robot, and *state* describes the object's status if any (e.g. (`state, mug, water`)).

The *object* (ob_m) describes the relation objects that exist in the environment (OB = `kettle, stove, ...` and $d_{OB} = 6$). The third element of the predicates can be either an object (e.g. (`on, mug, sink`)) or a state (e.g. (`state, kettle, water`), and the state set is C= `water, ON, ...` , $d_C = 6$). To fit both objects and states in one variable (i.e. oc_n), we use the following definition:

$$oc_n \in \begin{cases} C, & if\ re = state \\ OB, & otherwise \end{cases}$$

The total number of elements that exists in Object/state set (OC) is $max(d_{OB}, d_C) = d_{OC} = 6$. The state's predicates are converted to the indices (see Figure 9). Then, the state is represented as a sparse binary 3D tensor. If a predicate (x_i) exists in the environment, its corresponding value is 1, otherwise 0. Then, the tensor is reshaped in the form of a vector that can be used as the state in reinforcement learning (see Figure 9). We assume all possible predicates as $\Pi = \{x_1,\ x_2,\ ...,\ x_d\}$, where d is the total number of the predicates which in our case is $(d_{RE} - 1) \times d_{OB} \times d_{OC} = d = 108$. Then, the binary vector which represents the state is

$$s_t = [q_1,\ ...,\ q_d],\ where \underset{i \in \{1,...,d\}}{q_i} = \begin{cases} 1, & if\ x_i \in \chi_t \\ 0, & if\ x_i \notin \chi_t \end{cases}$$

We performed an analysis to show how our agent interprets the predicates. Each predicate (e.g. (`state, kettle, water`)) can be part of the state and it fully describes the environment. However, we can use our trained agent to see the assigned value to each predicate. The value of each predicate can depend on three factors. First, how frequently it appears in the sequence based on the agent's policy. Second, where it appears in the sequence (e.g. at the beginning of the sequence). Third, how frequently it co-occurs with other predicates. For example, the predicate (`state, kettle, water`) has the highest value (see Figure 10). As can be seen in Table 1, the agent's state has this predicate after two actions, and this predicate remains in the state until the end. If a predicate has a higher value, then it either happens at the end of the sequence or earlier but remains until the end.

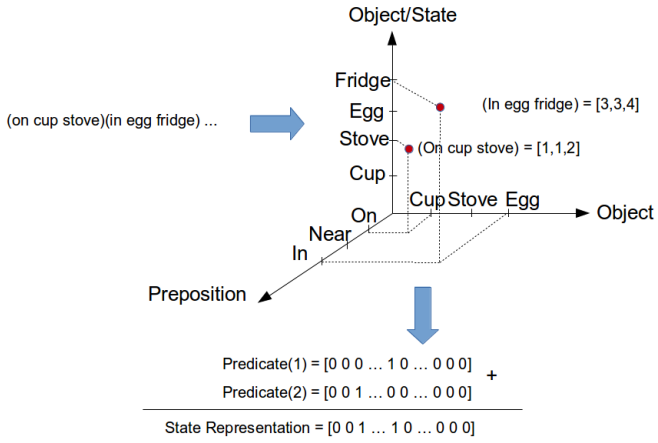

Figure 9: Converting symbolic representation to a binary vector. First the existing predicates are represented in a 3D tensor as 1. If the predicate does not exist then the corresponding value in the tensor is 0. Then, the tensor is flattened as a binary vector which uniquely describes the state.

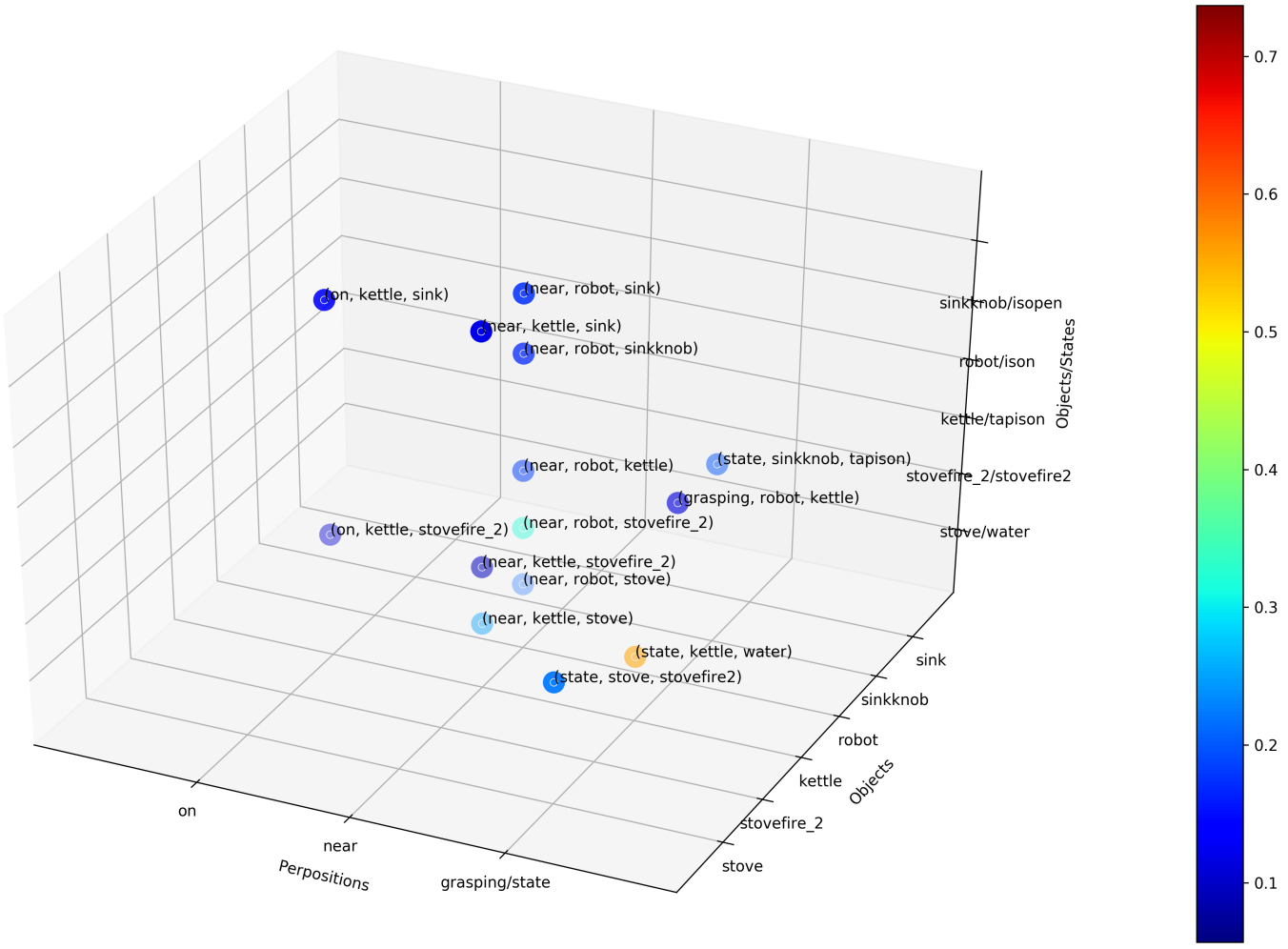

Figure 10: The value of each predicate is examined. For example, (*state, kettle, water*) has the highest value since it appears early in the optimal sequence of actions for the boiling water task and it lasts until the end as it is part of the terminal state.