Deep Reinforcement Learning for Conversational Robots Playing Games

Heriberto Cuayáhuitl[1]

***Abstract*— Deep reinforcement learning for interactive multimodal robots is attractive for endowing machines with trainable skill acquisition. But this form of learning still represents several challenges. The challenge that we focus in this paper is effective policy learning. To address that, in this paper we compare the Deep Q-Networks (DQN) method against a variant that aims for stronger decisions than the original method by avoiding decisions with the lowest negative rewards. We evaluated our baseline and proposed algorithms in agents playing the game of Noughts and Crosses with two grid sizes (3x3 and 5x5). Experimental results show evidence that our proposed method can lead to more effective policies than the baseline DQN method, which can be used for training interactive social robots.**

I. INTRODUCTION AND MOTIVATION

Training conversational robots that can learn useful skills for humans is an extremely challenging task [13], [4], [1], and deep (reinforcement) learning is promising for such a challenge. Deep reinforcement learning agents jointly learn their feature representations and interaction policies by using multi-layer neural networks. They are suitable for high-dimensional spaces and update their weights from numerical rewards. This form of learning is interesting because it avoids manual feature engineering, and it has shown promise for training intelligent machines. Example systems use visual inputs to control the speed of a car [11], to play Atari games [16], and to play the game of Go [22], among others.

Previous work in this field applied to conversational/social robots remains largely unexplored. Some exceptions include the following. [19] train a humanoid robot with social skills using the DQN method [16] and a two-stage approach. While the first stage collects data (grayscale and depth images) from the environment, the second stage trains two Convolutional neural nets with fused features. The robot's goal is to choose one of four actions: wait, look towards human, wave hand, and handshake. The robot receives a reward of +1 for a successful handshake, -0.1 for an unsuccessful handshake, and 0 otherwise. [3] train a robot to play games also using the DQN method. In this work a Convolutional neural net is used to predict game moves, and a fully-connected neural net is used to learn multimodal actions (18 in total) based on game rewards. Other previous works have addressed multimodal deep learning but in non-conversational settings [25], [18], [23], [12]. From these works we can observe that learning robots use small sets of actions in single-task scenarios. This is not surprising given the complexity and computational requirements for training robots with complex behaviour(s).

[1]H. Cuayáhuitl is with the Lincoln Centre for Autonomous Systems, School of Computer Science, University of Lincoln, Brayford Pool, Lincoln, LN6 7TS, United Kingdom hcuayahuitl@lincoln.ac.uk

Fig. 1. A humanoid robot playing the game of noughts and crosses (with different grid sizes) using multiple modalities and learnt behaviour

This paper investigates the effectiveness of deep reinforcement learning for conversational robots. In particular, using the Deep Q-Networks (DQN) method of [16]. Our long-term scenario is a humanoid robot playing multiple games with people that suffer from brain illnesses, where the robot's goal is to slow down cognitive decline through keeping the brains of people active with social games [8]. Our robot plays games using multiple input and output modalities including speech, touch and gestures. The game that we focus on is the game of *Noughts and Crosses* also known as 'Tic-Tac-Toe' and use two levels of difficulty (grids: 3x3 or 5x5) – see Figure1.

To measure the effectiveness of DQN for conversational robots we compared the original DQN method against a simple but effective extension that restricts the action space of agents to only good or safe actions. The latter is addressed by identifying actions with the lowest negative rewards before making any decision, e.g. loosing a game. This requires measuring the effects of every available action for allowing the agent to choose strong actions (such as those that avoid loosing games). Experimental results show that the proposed method can induce more effective (with higher winning rates) behaviours than the baseline. Although our results are only a small step towards the longer term goal of trainable interactive robots for useful purposes, the proposed method can easily be applied—due to its simplicity—to other methods aiming for larger-scale skill learning.

II. BACKGROUND

In reinforcement learning, an agent trains its behaviour from interaction with an environment – typically virtual due to the complexity of real environments, where situations are

mapped to actions (**dialogue acts** in the case of communicative multimodal interaction) [3]. The robot's task is to maximise a long-term reward signal [24]. A reinforcement learner is typically characterised by a set of input features called states $S = \{s_i\}$; a set of actions $A = \{a_j\}$, which can be verbal, non-verbal or both in the case of multimodal robots; a state transition function $T(s, a, s')$ that specifies the resulting state after executing action a in state s; a reward function $R(s, a, s')$ for choosing action a in state s; and a behaviour or policy denoted as $\pi : S \rightarrow A$, where there is typically a very large set of possible policies. The goal of a reinforcement learner is to find an optimal policy by maximising its cumulative discounted reward defined as

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + ... | s_t = s, a_t = a, \pi], \quad (1)$$

where function Q^* represents the maximum sum of rewards r_t discounted by factor γ at each time step. While reinforcement learners take actions with probability $Pr(a|s)$ during training, they select the best at test time according to

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a). \quad (2)$$

To induce the Q function above our agent approximates Q^* using a multilayer neural network as in [16]. The Q function is parameterised as $Q(s, a; \theta_i)$, where θ_i are the parameters (weights) of the neural net at iteration i. Training a deep reinforcement learner requires a dataset of learning experiences $D = \{e_1, ...e_N\}$ also referred to as 'experience replay memory'. A replay memory is used to draw samples from it in order to avoid correlations in the data. Every learning experience in a replay memory is described as a tuple $e_t = (s_t, a_t, r_t, s_{t+1})$. Inducing the Q function consists in applying Q-learning updates over minibatches of experience $MB = \{(s, a, r, s') \sim U(D)\}$ drawn uniformly at random from the full dataset D. A Q-learning update at iteration i is thus defined according to the loss function

$$L_i(\theta_i) = \mathbb{E}_{MB}\left[(r + \gamma \max_{a'} Q(s', a'; \overline{\theta}_i) - Q(s, a; \theta_i))^2\right], \quad (3)$$

where θ_i are the parameters of the neural net at iteration i, and $\overline{\theta}_i$ are the target parameters of the neural net at iteration i. The latter are held fixed between individual updates. This procedure is implemented in the learning algorithm *Deep Q-Learning with Experience Replay (DQN)* described in [15].

III. PROPOSED METHOD

We extend the method above with the ability to make strong decisions by avoiding weak decisions in both training and testing. To put this in context let us imagine a multimodal robot playing games, where a human tries to beat the robot in the game and the other way round. In the middle of a game the robot has a number of actions or decisions available, which may lead to winning or loosing the game. Typically, the robot gets a positive numerical reward for winning and a negative numerical reward for loosing the game. This setting raises the following question: *Why robots do not avoid such really weak decisions from their behaviour?* We denote such subset of actions as $\hat{A}$ (referred as *"worst negative actions"*), which obtain the lowest negative immediate rewards. Our proposed method, formalised in algorithm 1, considers $\hat{A}$ as empty in the case of non-worst negative rewards. This represesents a simple but powerful extension to the original DQN algorithm. The main changes are in lines 5 and 6, which require the identification of worst actions $\hat{A}$ so that decision making can be made based on actions in A not in $\hat{A}$, also denoted as $A \setminus \hat{A}$. A reinforcement learning agent using our proposed method selects actions according to

$$\pi^*_\theta(s) = \arg\max_{A \setminus \hat{A}} Q^*(s, a; \theta). \quad (4)$$

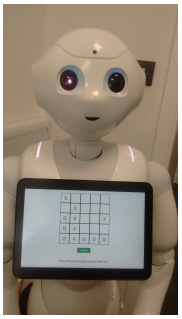
Notice that identifying $\hat{A}$ requires look ahead rewards. This means that actions in set A can be taken temporarily in the environment to observe the consequent rewards, and then undo such temporary actions to remain in the original environment state s – before looking for $\hat{A}$.

Algorithm 1 DQN Learning with Strong Decisions

1: Initialise Deep Q-Networks with replay memory D, action-value function Q with random weights θ, and target action-value functions $\hat{Q}$ with weights $\hat{\theta} = \theta$
2: **repeat**
3: $s \leftarrow$ initial environment state in S
4: **repeat**
5: $\hat{A} = \begin{cases} \text{actions with } \min(r(s,a) < 0 \ \forall a \in A) \\ \emptyset \text{ otherwise} \end{cases}$
6: $a = \begin{cases} rand_{a \in A} \text{ if random number} \leq \epsilon \\ \max_{a \in A \setminus \hat{A}} \hat{Q}(s', a'; \hat{\theta}) \text{ otherwise} \end{cases}$
7: Execute action a and observe reward r and next state s'
8: Append transition (s, a, r, s') to D
9: Sample random minibatch (s_j, a_j, r_j, s'_j) from D
10: $y_j = \begin{cases} r_j \text{ if final step of episode} \\ r_j + \gamma \max_{a \in A} \hat{Q}(s', a'; \hat{\theta}) \quad \text{otherwise} \end{cases}$
11: Set $err = (y_j - Q(s', a'; \theta))^2$
12: Gradient descent step on err with respect to θ
13: Reset $\hat{Q} = Q$ every C steps
14: $s \leftarrow s'$
15: **until** s is a goal state
16: **until** convergence

IV. EXPERIMENTS AND RESULTS

We equipped the Pepper humanoid robot (see Section IV-C) with multimodal deep reinforcement learning agents for playing the well-known game of Noughts and Crosses (N&C) with two grid sizes. A player wins by getting 3 symbols of the same in a row in the N&C 3x3 game and 4 in a row in N&C 5x5. Rather than training our robot using real human-robot interactions with thousands of games, we used simulated interactions.



A. CHARACTERISATION OF LEARNING AGENTS

a) States: The state spaces of our learning agents include 62 and 95 input features (for both grids) that describe the game moves and words raised in the interactions. While words derived from system responses are treated as binary variables (i.e. word present or absent), words derived from noisy user responses are treated as continuous variables by taking speech recognition confidence scores into account. Since we use a single variable per word, user features override system ones in case of overlaps. In addition and in contrast to word-based features that describe the last system and user responses, game moves (treated as binary features) do take into account the whole history of each game.

b) Actions: The action spaces includes 18 and 34 multimodal dialogue acts for both games N&C 3x3 and N&C 5x5[1]. Rather than training agents with all actions in every state, the actions per state were restricted in two ways. First, dialogue acts are derived from the most likely actions, $Pr(a|s) > 0.001$, with probabilities determined from a Naive Bayes classifier trained from example dialogues – see Appendix at the end of the paper. Second and in the case of game moves, all valid physical actions were allowed – but only those game moves not taken by any other player.

c) State Transitions: The input features in every situation are based on numerical vectors representing the last system and user responses, and game history. The latter means that we kept the game move features as to describe the game state rather than resetting them at every turn. The system responses are straightforward, 0 if absent and 1 if present. The user responses correspond to speech-based confidence levels [0..1] of noisy user responses. The language generator used template-based responses similar to those provided in the example interactions.

d) Rewards: Given that the robot's goal was to win as many games as possible, the rewards are simple and based on game scores according to the following function:

$$r(s,a,s') = \begin{cases} +5 \text{ for winning or about to win the game} \\ +3 \text{ for a draw or about to draw in the game} \\ -1 \text{ for a repeated (already taken) action} \\ -5 \text{ for loosing or about to loose the game} \\ 0 \text{ otherwise.} \end{cases} \quad (5)$$

e) Model Architectures: The neural networks consist of fully-connected multilayer neural nets with 5 layers organised as follows: 62 or 95 nodes in the input layer, 100 nodes in each of the three hidden layers, and 18 or 34 nodes in the output layer. The hidden layers use RELU (Rectified Linear Units) activation functions to normalise their weights. Other learning parameters include experience replay size=10000, burning steps=1000, discount factor=0.7, minimum epsilon=0.005, batch size=2, learning steps=100K, and max. number of actions per dialogue=50.

[1]GameMove(gridloc=$loc) × 9 in N&C3x3 or × 25 in N&C5x5, Provide(feedback=draw), Provide(feedback=loose), Provide(feedback=win), Provide(name), Reply(playGame=yes), Request(playGame), Request(userGameMove), Salutation(closing), Salutation(greeting).

f) User Simulation: Semi-random user behaviour is derived from random but legal game moves. The user responses are randomly selected from observed examples. While random speech recognition confidence scores are used during training, actual confidence scores are used for testing.

B. EXPERIMENTAL RESULTS

In this section we compare agents using the standard DQN method versus our proposed method described in Section III. While the former uses all available actions for training, the latter avoids weak actions with the lowest negative rewards. Our domain to evaluate such agents is the popular game of noughts and crosses using two grid sizes 3x3 and 5x5. Both methods use the same data, resources and hyperparameters for training. The only difference between both systems is the learning method (DQN baseline or DQN proposed).

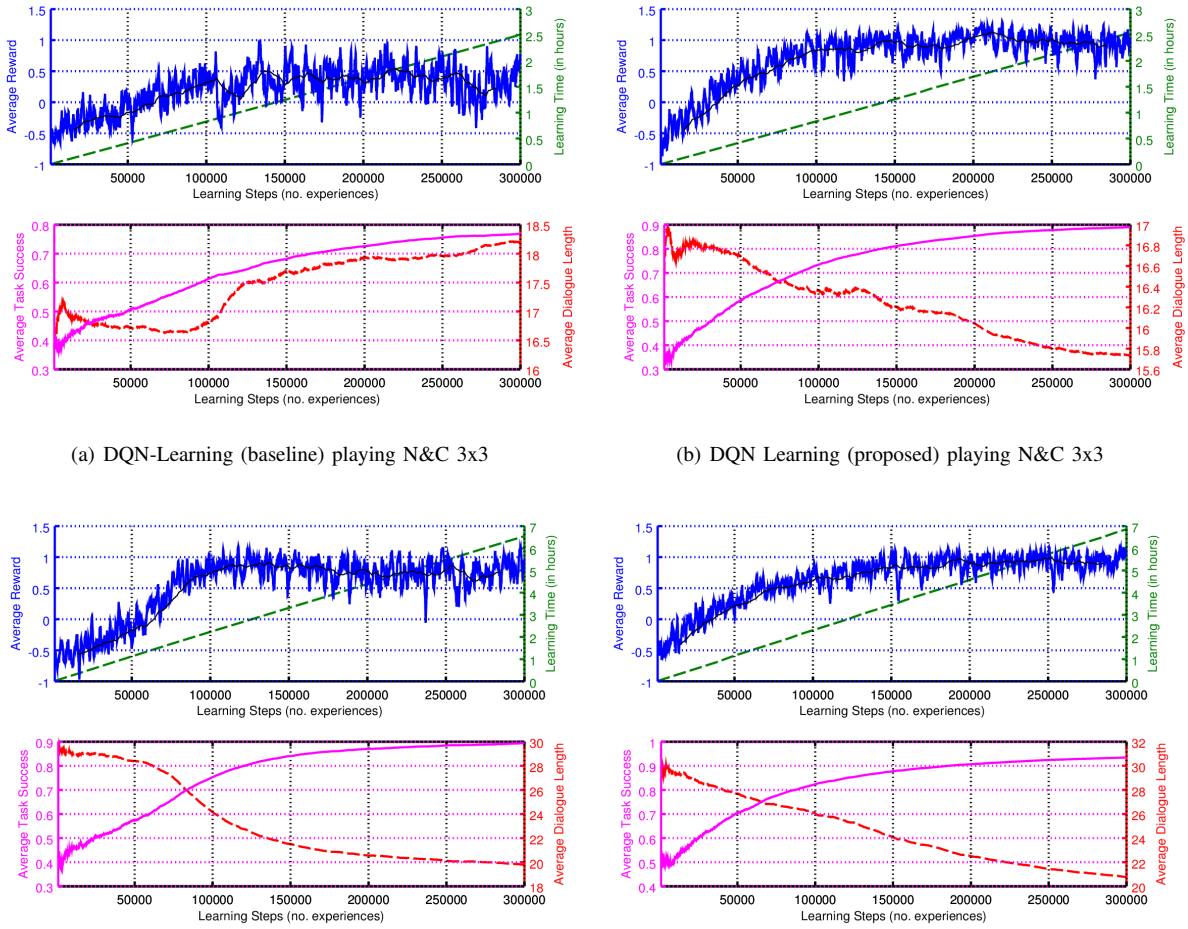
We use four metrics to measure system performance: avg. reward (sum of rewards divided by learning steps), learning time[2], avg. task success (win/draw rate), and avg. dialogue length (avg. number of actions per game). The higher the better in avg. reward and avg. task success, and the lower the better in training time and dialogue length.

Figure 2 shows learning curves for the baseline agents (left), and agents using the proposed algorithm. Both the baseline and proposed agents report results over 300K learning steps (about 17K dialogues in N&C 3x3 and 15K dialogues in N&C 5x5). Our results report that our proposed method can train more successful agents. The evidence for this claim can be seen in the learning curves of avg. reward and avg. task success on the right versus those on the left – they have higher values and more consistently. While the training times in the proposed method are slightly higher, the differences at test time are more relevant. Even when the baseline policy in the 5x5 game (see Figure 2(c)) indicates more efficient games, they are not as successful as the policies induced by the proposed method (see Figure 2(b)(d)).

To add additional evidence to our claim, we tested (offline) the performance of the learnt policies over 1000 simulated games for each of the four agents and obtained the results shown in Table I. It can be noted that indeed the proposed method performs better than the baseline DQN method.

An error analysis of of the learnt policies revealed the following. The learnt behaviours of all agents exhibited repeated actions such as "I am Pepper",...,"I am Pepper" or "Nice, let me start.",...,"Nice, let me start." This explains why the avg. dialogue length in Figure 2(a) grows instead of decreasing over time. The proportion of games with repetition for each agent is quantified as $1 - TaskSuccess$. Excluding such repetitions resulted in a task success of 1.0 for all agents, i.e. no lost games. This observation tells us that successful games can be learnt by both methods – but only in the case of coherent interactions. The repetition problem in the proposed method was solved by replacing the -1 reward for -5 in Eq. 5. The fact that avg. reward and task success are higher in the game with larger grid (5x5) than the game

[2]Ran on Intel Core i5-3210M CPU@2.50GHzx4; 8GB RAM@2.4GHz.



(a) DQN-Learning (baseline) playing N&C 3x3

(b) DQN Learning (proposed) playing N&C 3x3

(c) DQN-Learning (baseline) playing N&C 5x5

(d) DQN Learning (proposed) playing N&C 5x5

Fig. 2. Learning curves of DQN-based agents using the baseline algorithm (left) and the proposed one with strong decisions (right), which show that the proposed method can train more successful agents as measured by average reward and average task success – see text for further details

with smaller grid (3x3) suggests that it is easier to beat the simulated user in the larger grid than the smaller one.

Game	Learning Algorithm	Average Reward	Task Success	Dialogue Length
N&C 3x3	$DQN_{baseline}$	0.6370	0.9085	17.24
	$DQN_{proposed}$	**1.0970**	**0.9558**	**16.28**
N&C 5x5	$DQN_{baseline}$	0.8020	0.9339	18.03
	$DQN_{proposed}$	**0.9170**	**0.9801**	**17.93**

TABLE I

TEST RESULTS AVERAGED OVER 1000 GAMES (N&C=NOUGHTS AND CROSSES) USING THE BASELINE AND PROPOSED DQN ALGORITHMS

Game / Method	$DQN_{baseline}$	$DQN_{proposed}$
N&C 3x3	84.8%	**90.1%**
N&C 5x5	92.9%	**95.8%**

TABLE II

WINNING RATES OBSERVED FROM 1000 GAMES PER AGENT-GAME

To extend our error analysis further we ran 1000 simulated games per agent to quantify their winning rates (only winning instead of winning plus drawing, without game exclusion), obtaining the results shown in Table II. These results show further evidence that our proposed method can yield more effective robot policies than the baseline method.

C. INTEGRATED SYSTEM

Our humanoid robot[3] used multiple modalities – including speech, touch and gestures. To do that we used both built-in components and components built specifically for our Naoqi-based integrated system – see Figure 3. While the interaction manager, game move recogniser and game visualiser were implemented in Java and Javascript, the rest was implemented in Python using the Naoqi API[4]. These components run concurrently, via multi-threading, communicate though Web Sockets and are briefly explained as follows.

[3] www.aldebaran.com/en/a-robots/who-is-pepper

[4] http://doc.aldebaran.com/2-4/naoqi/index.html

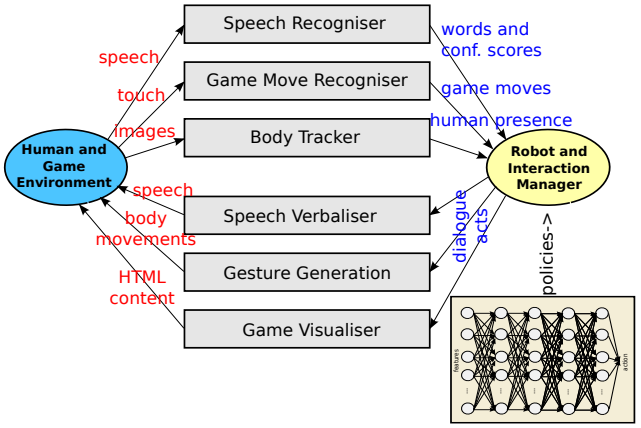


Fig. 3. High-level architecture of our integrated system

1) Speech Recognition: This component runs the Nuance Speech Recogniser, which is activated once the robot finishes speaking. These perceptions are used as features in the state space of the deep reinforcement learning agents.

2) Game Move Recognition: This component maps screen touches to game moves. These touch screen-based perceptions are used as features in the state space of the deep reinforcement learning agents. A system based on vision-based perceptions of handwriting is left as future work.

3) Body Tracker: The built-in body tracker was used to detect changes in orientation of the human player's head and body. This allowed the robot to know where the user is located at in order to follow the player in focus.

4) Speech Synthesis: The verbalisations (in English), translations of high-level actions from the interaction manager, used a template-based approach and the built-in Acapela speech synthesizer. The spoken verbalisations were synchronised with the arm movements, where the next verbalisation waited until the previous verbalisation and corresponding arm movements completed their execution.

5) Arm Movement Generation: This component receives commands from the interaction manager for carrying out gestures. We used built-in arm movements for non-game moves and pre-recorded arm movements (human demonstrations) to indicate game moves, which notified the interaction manager when their execution was completed. In this way, a future verbalisation waited until the drawing gestures were done and the arms were at the initial position (downwards).

6) Visualisations: This component displayed HTML content in the Tablet including a launch game button, the game grids and the words of the last system turn. The tablet's role was only to display the state of the game and to notify events (screen touches) to the interaction manager.

7) Interaction Manager: The interaction manager, based on the publicly available tools *SimpleDS*[5] [2] and ConvNetJS[6] , can be seen as the robot's brain due to coordinating the components above by continuously receiving speech-based and touch-based perceptions from the environment, and deciding what and when to do next. Regarding what to do next, it chooses actions based on the learning agents described above. Most actions are verbal and non-verbal; for example, action $GameMove(gridloc = loc31)$ can be unfolded as "I take this one $[who{=}rob \wedge what{=}draw \wedge where{=}loc31]$", where the square brackets denote a physical action (drawing a circle or cross at the given location).

This integrated robot system has been fully implemented as can be seen in this video: `https://youtu.be/OqEtt4rwIBY`. While the system performed well and according to expectations in pilot tests, a human evaluation with elderly people in particular is one of our future works.

[5]`https://github.com/cuayahuitl/SimpleDS`
[6]`http://cs.stanford.edu/people/karpathy/convnetjs`

V. CONCLUSION

We present a learning algorithm for training interactive conversational robots. Our policies are trained in simulation due to the large amount of examples required to learn optimal policies. The inputs to the neural nets are based on words and game moves detected from a touch screen. The outputs to the neural nets are multimodal actions containing words and arm movements. A natural extension to this work is training agents to play multiple games, but this will require more scalable methods combining ideas from [17], [14], [21], [9], [10], [6], [5], [20], [7]. The degrees of engagement using different sets of modalities also remains to be investigated.

Our experimental results showed that successful interactions can be obtained from induced policies using the baseline and proposed methods. This was especially true in the case of coherent interactions without repetition of actions as described in Section IV-B, where both methods were able to induce policies that do not loose games. The proposed method however showed be more robust against those repetitions, and also showed to lead to higher winning rates than the baseline. One can argue that the user simulator is too simple, but it used all possible valid actions and all experiments were carried out under the same settings. Our experiments were thus carried out under fair conditions, which is in favour of our claim stating that $DQN_{proposed}$ is more effective—with higher winning rates—than $DQN_{baseline}$.

How important is our main result? On the one hand, if our goal is to train robots that aim to stimulate human brain activity then they should be competitive enough as done by the proposed method. On the other hand, a robot that wins all the time may not be the ultimate solution. How much and when should a robot win or loose is still an open question to address in the future. For example, a fun and challenging interaction may be better than always winning. But this robot will require neural nets with support for a larger amount of inputs and outputs, which brings us back to the scalability issue raised above. These represent exiting research avenues to explore for training conversational humanoid robots.

VI. ACKNOWLEDGEMENT

The robot used in this paper was funded by the Engineering and Physical Sciences Research Council (EPSRC). We would like to thank Alexandra Gorry-Pollet & Arnauld Alex for helping with photo/video shooting and system testing.

REFERENCES

[1] H. Cuayáhuitl. Robot learning from verbal interaction: A brief survey. In *4th International Symposium on New Frontiers in HRI*, 2015.

[2] H. Cuayáhuitl. *SimpleDS: A Simple Deep Reinforcement Learning Dialogue System*, pages 109–118. Springer, Singapore, 2017.

[3] H. Cuayáhuitl, G. Couly, and C. Olalainty. Training an interactive humanoid robot using multimodal deep reinforcement learning. *CoRR*, abs/1611.08666, 2016.

[4] H. Cuayáhuitl, M. van Otterlo, N. Dethlefs, and L. Frommberger. Machine learning for interactive systems and robots: A brief introduction. In *IJCAI Workshop on Machine Learning for Int. Sys. (MLIS)*, 2013.

[5] H. Cuayáhuitl and S. Yu. Deep reinforcement learning of dialogue policies with less weight updates. In *International Conference of the Speech Communication Association (INTERSPEECH)*, 2017.

[6] H. Cuayáhuitl, S. Yu, A. Williamson, and J. Carse. Scaling up deep reinforcement learning for multi-domain dialogue systems. In *International Joint Conference on Neural Networks (IJCNN)*, 2017.

[7] N. Dethlefs and H. Cuayáhuitl. Hierarchical reinforcement learning for situated natural language generation. *Nat. Lang. Eng.*, 21, 5 2015.

[8] N. Dethlefs, M. Milders, H. Cuayáhuitl, T. Al-Salkini, and L. Douglas. A natural language-based presentation of cognitive stimulation to people with dementia in assistive technology: A pilot study. *Informatics for Health and Social Care*, 0(0):1–12, 2017.

[9] F. S. He, Y. Liu, A. G. Schwing, and J. Peng. Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *CoRR*, abs/1611.01606, 2016.

[10] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*, 2016.

[11] S. Lange, M. A. Riedmiller, and A. Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *International Joint Conference on Neural Networks (IJCNN)*, 2012.

[12] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015.

[13] N. Mavridis. A review of verbal and non-verbal human-robot interactive communication. *CoRR*, abs/1401.4994, 2014.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[17] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015.

[18] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. Multimodal deep learning. In *ICML*, 2011.

[19] A. H. Qureshi, Y. Nakamura, Y. Yoshikawa, and H. Ishiguro. Robot gains social intelligence through multimodal deep reinforcement learning. In *HUMANOIDS*, 2016.

[20] A. H. Qureshi, Y. Nakamura, Y. Yoshikawa, and H. Ishiguro. Show, attend and interact: Perceivable human-robot social interaction through neural attention q-network. *CoRR*, abs/1702.08626, 2017.

[21] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484 – 489, 2016.

[23] N. Srivastava and R. Salakhutdinov. Multimodal learning with deep boltzmann machines. *J. of Machine Learning Research*, 15(1), 2014.

[24] C. Szepesvári. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.

[25] S. Wermter, C. Weber, M. Elshaw, C. Panchev, H. R. Erwin, and F. Pulvermüller. Towards multimodal neural robot learning. *Robotics and Autonomous Systems*, 47(2-3), 2004.

APPENDIX

The dialogue below is one of the manually-specified seed demonstrations for inducing the agents features and actions, and for training classifiers to predict the most likely actions given the previous system and user responses. The grid below is how the game would look like at the end of the interaction – assuming that the agent uses noughts and the user uses crosses. Given that we aim for a sample efficient training methodology, we provided 10 example interactions for the Noughts and Crosses game with grid 3x3 and 3 interactions for the game with grid 5x5. These features and dialogue acts were used by the DRL agents described in Section III for inducing their dialogue behaviour. In the dialogue below the square brackets on the left denote system actions, double quotes denote system or user responses, and functions within double quotes denote physical actions (gestures in our case).

o		o	x	o
o	o	x	x	x
	o	x	x	x
	x		o	x
x	o	o		o

```
SYS:[Salutation(greeting)]"Hello"
USR:null
SYS:[Provide(name)]"I am Pepper"
USR:null
SYS:[Request(playGame)]"Would you like to play a game?"
USR:"Okay yes"
SYS:[Reply(playGame=yes)]"Nice. Let me start."
USR:null
SYS:[GameMove(gridloc=loc00)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc14"
SYS:[GameMove(gridloc=loc33)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc24"
SYS:[GameMove(gridloc=loc04)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc34"
SYS:[GameMove(gridloc=loc44)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc22"
SYS:[GameMove(gridloc=loc10)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc23"
SYS:[GameMove(gridloc=loc21)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc12"
SYS:[GameMove(gridloc=loc02)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc03"
SYS:[GameMove(gridloc=loc11)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc31"
SYS:[GameMove(gridloc=loc41)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc13"
SYS:[GameMove(gridloc=loc42)]"I take this one Draw(...)"
USR:null
SYS:[Request(userGameMove)]"your turn"
USR:"I take this loc40"
SYS:[Provide(feedback=loose)]"No, I lost."
USR:null
SYS:[Salutation(closing)]"Good bye!"
USR:null
```