Deep Reinforcement Learning of Dialogue Policies with Less Weight Updates

Heriberto Cuayáhuitl[1], *Seunghak Yu*[2]

[1]University of Lincoln, School of Computer Science, Lincoln, United Kingdom
[2]Samsung Electronics Ltd., Artificial Intelligence Team, Seoul, South Korea
HCuayahuitl@lincoln.ac.uk, seunghak.yu@samsung.com

Abstract

Deep reinforcement learning dialogue systems are attractive because they can jointly learn their feature representations and policies without manual feature engineering. But its application is challenging due to slow learning. We propose a two-stage method for accelerating the induction of single or multi-domain dialogue policies. While the first stage reduces the amount of weight updates over time, the second stage uses very limited minibatches (of as much as two learning experiences) sampled from experience replay memories. The former frequently updates the weights of the neural nets at early stages of training, and decreases the amount of updates as training progresses by performing updates during exploration and by skipping updates during exploitation. The learning process is thus accelerated through less weight updates in both stages. An empirical evaluation in three domains (restaurants, hotels and tv guide) confirms that the proposed method trains policies 5 times faster than a baseline without the proposed method. Our findings are useful for training larger-scale neural-based spoken dialogue systems.
Index Terms: spoken dialogue systems, deep reinforcement learning, multi-domain dialogue management

1. Introduction

Deep Reinforcement Learning (DRL) agents aim to jointly learn their feature representations (or 'environment state') and interaction policies by using multi-layer neural networks. They are suitable for high-dimensional spaces with hundreds or thousands of dimensions, and update their weights (or strengths of connection between neurons) from numerical rewards. This form of learning has led to higher levels of automation than previous attempts by avoiding manual feature engineering. DRL agents have already shown huge promise for training intelligent systems—with particularly impressive results in agents using visual inputs to control the speed of a car [1], to play Atari games [2], and to play the game of Go [3], among others.

DRL has not gone unnoticed by the dialogue systems community. Example DRL-based dialogue systems include the following. [4] proposed to apply DRL for mapping speech recognition results to dialogue actions—bypassing the natural language understanding component. [5, 6] train a DRL agent using a feed-forward neural network for mapping game situations to strategic dialogue actions in the game of 'Settlers of Catan'. [7] train an actor-critic agent with a combined supervised/reinforcement learning method using a feed-forward neural network in the restaurant domain. [8] train a hybrid learning agent using a recurrent neural network that predicts the values of a DRL policy in the phone-call domain. [9] train an agent using a long-short term memory network for a Quiz game, where such a recurrent neural net is also used for belief tracking. Other neural-based dialogue agents have been applied to text prediction using the sequence-to-sequence learning approach [10, 11], to response generation using a long-short term memory network for chat bot dialogues[12], and to reasoning with inference for text-based question answering [13, 14, 15].

Funding from Samsung Electronics is gratefully acknowledged.

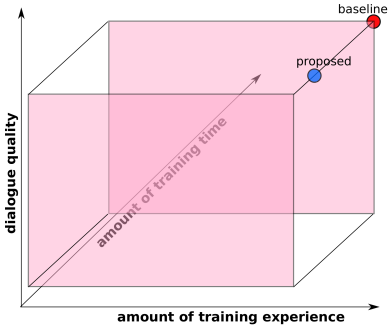


Figure 1: *A 3D viewpoint of dialogue systems training*

A drawback of DRL lies in its computational requirements, which is not surprising because recent neural nets have millions or billions of weights to update to induce stable behaviour. This computational expense can increase as more domains are taken into account due to the increased inputs (states) and outputs (actions). The research question addressed in this paper is *How to train deep reinforcement learning dialogue agents faster and without performance degradation?* Assume training a system with X number of dialogues, Y dialogue quality score (the higher the better), and Z training time (the shorter the better). Figure 1 helps to illustrate the contribution of this paper, which aims to train deep reinforcement learning dialogue systems faster—without sacrificing the quality of learnt policies. We aim to train dialogue agents of the type $\{X, Y, Z' < Z\}$.

Previous work tackling faster learning of DRL-based agents have used distributed neural nets [16, 17], prioritised experience replay by sampling from important previous experiences [18], fast reward propagation [19], and distributed policies to train specialised agents (one per task or domain) [20, 21, 22]. We propose a method that can be used on top of previous methods, which applies a reduced amount of weight updates in two ways. First, it gradually reduces the amount of weight updates over time, assuming that weights in the long run do not require to be frequently updated as in early stages of training. Second, it uses more limited minibatches of learning experiences than usual systems. Although both ways can be applied independently, they both contribute faster learning than either or none of them. The remainder of the paper describes our method in further detail and empirically validates its usefulness. Even though the proposed method is applied here to multiple domains, it can also speedup training of single-domain conversational agents.

2. Background

This paper treats neural-based dialogue agents in multiple domains as a network of Deep Reinforcement Learners as proposed in [20], for example by using a network of Deep Q-networks (DQN). A DQN agent aims to find an optimal policy by maximising its cumulative discounted reward defined as

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

where function Q^* represents the maximum sum of rewards r_t discounted by factor γ at each time step. In the Q function above θ represents the parameters (weights) of the neural net, which are updated after each decision (selected action). Furthermore, training a DRL agent requires a dataset of experiences $D = \{e_1, ...e_N\}$ (also called 'experience replay memory' [23, 24]) collected during online learning, where every experience 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 mini-batches 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

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

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 process is implemented in the learning algorithm *Deep Q-Learning with Experience Replay* described in [25].

Instead of training a single DQN agent, we train a Network of DQN agents (referred to as NDQN), where every DQN in the network represents a specialised skill to converse in a particular subdialogue—see Figure 2. The network of agents enable DQNs to be executed without a fixed structure in order to support flexible and unstructured dialogues. In contrast to Hierarchical DQNs [21] that follow a strict sequence of agents, our network of DQN agents aims to allow more flexible transitions between all DQN agents except for self-transitions. The latter uses a stack-based approach as in [26]. While user responses can motivate transitions to another domain in the network, completing a subdialogue within a domain motivates a transition to the previous domain to resume the interaction. [20] and [22] describe algorithms to train and execute NDQN agents.

An optimal policy in an NDQN performs action selection according to

$$\pi^*_{\theta^{(d)}}(s) = \arg \max_{a \in A^{(d)}} Q^{*(d)}(s, a; \theta^{(d)}), \quad (1)$$

where function F selects domain or skill $d \in \mathcal{D}$ according to

$$d_{t+1} = \arg \max_{d' \in \mathcal{D}} F(d' | d_t, s_{t+1}), \quad (2)$$

and evidence s_{t+1} takes into account all features that describe the most recent environment space exhibited in domain d at time t. While the domain transition function is essentially a classifier (Equation 2) and used for high-level transitions in the interaction, Equation 1 is used for low-level transitions within a domain (subdialogue) in the network and is subject to reinforcement learning. NDQN assumes that the domain transition function F can be deterministic or probabilistic (the latter estimated for example in a supervised learning manner). Although in this paper function F is a prior requirement for NDQN-Learning, it can also be learnt in parallel with the dialogue policies $\pi^*_{\theta^{(d)}}$.

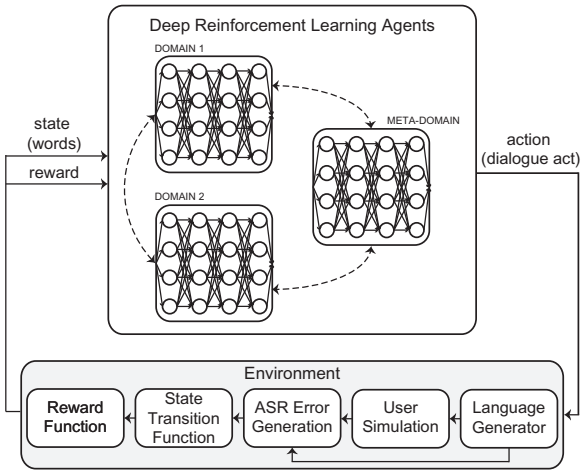


Figure 2: *Illustration of a Network of DQN dialogue agents (NDQN). The dashed arrows connecting domains denote transitions between domains to avoid rigid structures in the interaction. Section 4 describes these elements in further detail.*

3. Proposed Algorithm

The learning procedure above assumes that the parameters of the neural network of each domain $\theta^{(d)}$ are updated at every time step—after each action is chosen and executed. In other words, the backward step in gradient descent [27, 28] is applied every time the agent makes a decision. Our proposed algorithm is driven by the question *Can we apply selective weight updates to achieve faster learning without sacrificing the quality of policies?* Our experiments below suggest or indicate a positive answer to this question.

Action-selection strategies in reinforcement learning are used to encourage high degrees of exploration at early stages of training and high degrees of exploitation at later stages of training [29, 30]. We apply the same idea to updating the weights in the neural nets of our learning agents for training them faster than traditional training schemes. This means that the parameters $\theta^{(d)}$ will be updated almost always at early stages of training, and the amount of updates will gradually decrease as training progresses. This strategy can be interpreted as a learning agent only performing weight updates when it is uncertain about its decisions—assuming that the uncertainty is decreased in the long run. Algorithm 1 describes and formalises this idea.

This algorithm initialises a set of replay memories $D^{(d)}$ and random weights $\hat{\theta}^{(d)}$=$\theta^{(d)}$ for all domains. The outer loop iterates over a set of dialogues, the middle loop iterates over the domains raised in the interaction, and the most inner loop iterates over the actions per domain. Note that the most inner loop terminates when a domain is over or when there is a change of domain. While the middle loop terminates when a dialogue is completed, the outer loop usually terminates after a fixed number of dialogues (or until convergence is observed). Notice also that line 7 of this algorithm refers to an action chosen as 'exploratory' or 'optimal'. The former is used in line 10 as a selector to perform the weight update, i.e. updates to the weights occur only for exploratory actions. Note also that the memories $D^{(d)}$ are updated accordingly—in case of exploratory actions. The size of minibatch for faster learning is reported in Section 5.

Algorithm 1 NDQN-Learning with Less Weight Updates

Initialise set of Deep Q-Networks with replay memories $D^{(d)}$, action-value functions $Q^{(d)}$ with random weights $\theta^{(d)}$, target action-value functions $\hat{Q}^{(d)}$ with weights $\hat{\theta}^{(d)} = \theta^{(d)}$, and exploration rate $\epsilon^{(d)}$ per domain d
repeat
  Set initial domain d, predefined or defined by $\arg\max_{d \in D} F(d)$
  Set initial environment state $s \in S^{(d)}$
  repeat
    repeat
      $a = \begin{cases} \text{choose exploratory action } a \in A^{(d)} & \text{with probability } \epsilon^{(d)} \\ \arg\max_a Q^{*(d)}(s, a; \theta^{(d)}) & \text{otherwise} \end{cases}$
      Execute action a and observe reward r and next state s'
      Set next domain d' according to $\arg\max_{d' \in \mathcal{D}} F(d'|d, s')$
      if a is an exploratory action and $|A^{(d)}(s)| > 1$ **then**
        Append transition (s, a, r, s') to $D^{(d)}$
        Sample random minibatch (s_j, a_j, r_j, s'_j) from $D^{(d)}$
        $y_j = \begin{cases} r_j & \text{if final step} \\ r_j + \gamma \max_{a \in A^{(d)}} \hat{Q}^{(d)}(s', a'; \hat{\theta}^{(d)}), & \text{otherwise} \end{cases}$
        Set $err_j = \left(y_j - Q^{(d)}(s', a'; \theta^{(d)})\right)^2$
        Gradient descent step on err_j with respect to $\theta^{(d)}$
      end if
      Decay $\epsilon^{(d)}$
      Set $\hat{Q}^{(d)} = Q^{(d)}$
      Set $s = s'$
    until s is a terminal state or $d \neq d'$
    Set $d = d'$
  until s is a goal state
until convergence

4. Multi-Domain Dialogue System

The proposed computational framework for training multi-domain neural-based dialogue agents is a substantial extension from the publicly available software tools SimpleDS [4] and ConvnetJS [31]. It can be executed in training or test mode using simulations or speech-based interactions (via a mobile App[1]). Our dialogue system runs under a client-server architecture, where the learning agents—one per domain—act as the *clients* and the dialogue system as the *server*. They communicate by exchanging messages, where the clients inform the server the action to execute, and the server informs the clients the state and rewards observed. The elements for training multi-domain DRL-based dialogue systems are as follows.

The **state spaces** include word-based features depending on the vocabulary of each learning agent. They include 273 unique words and 200 predefined synonyms to deal with unseen words during training. An agent in a particular domain has relevant features for its own domain and it is agnostic of features in other domains. While words derived from system responses are treated as binary variables (i.e. word present or absent), the words derived from user responses can be seen as continuous variables by taking ASR confidence scores into account. Our state representations used delexicalised word-based representations and excluded words from information presentation—for increased scalability, as described in [20].

The **action spaces** include dialogue acts for the targeted domains—currently 97 unique actions in total. Example dialogue act types, dialogue acts without slot-values, are as follows: Salutation(), Request(), AskFor(), Apology(), ExpConfirm(), ImpConfirm(), Retrieve(), Provide(), among others. The set of slots and domain value sizes include the following: meta={|domain|=3}; restaurants={|food_type|=10, |area|=4, |price|=3}; hotels={|city|=20, |day|=31, |month|=12, |nights|=14}; tv_guide={|genre|=12, |day|=9, |time|=24}. Rather than learning with whole action sets, our framework supports learning from valid actions—selected in a two stage process. Firstly, actions are selected from the most likely actions, $Pr(a|s) > 0.0001$, derived from Naive Bayes classifiers (due to scalability purposes) trained from demonstration dialogues. See example demonstration dialogue in Appendix of [20]. Secondly, the most likely actions in he previous stage are extended with legitimate requests, apologies and confirmations.

The **state transition functions** are based on numerical vectors representing the last system and user responses. Taking a wider dialogue context into account is also possible but not explored here. The system responses are straightforward, 0 if absent and 1 if present (hit-or-miss). The user responses correspond to the confidence level [0..1] of noisy user responses. While system responses are generated from templates, user responses are generated from semi-random behaviour. Trainable language generation from raw text is left as future work [32, 33].

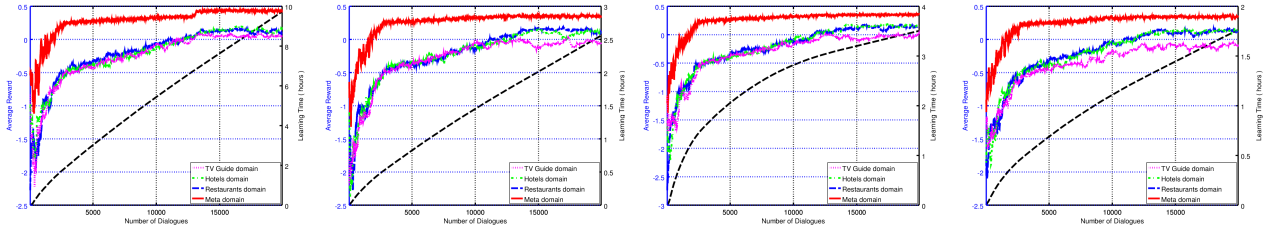
The **domain transition function** specifies the next domain or task in focus. It is currently defined deterministically, and it is also implemented as a Support Vector Machine (SVM) classifier trained from example dialogues [20]. The design of this classifier follows that of a two-deep fully connected neural net with 80 nodes in each hidden layer, with tanh activation, and an SVM output layer, using Hinge Loss. While the input layer accepts domain-independent *words-as-features* vectors representing the unique global vocabulary shared amongst all domains in a hit-or-miss approach, the output layer has classes representing system domains. We refer to meta domain as subdialogues containing domain-general system and user responses.

The **reward function** assigns partial rewards according to $R(s, a, s') = GR + DR - DL$. GR is a contribution to slot-filling in the range [-1..1]—it deducts the number of repetitions from the number of slot fillings and confirmations in the current utterance, and divides by the number of slots to confirm in the current domain. DR is a data-like probability of having observed action a in state s, obtained from the Naive Bayes classifiers mentioned above to allow statistical inference over actions given states ($Pr(a|s)$). Finally, DL is a dialogue length score (steps×-0.1 in our case) to encourage efficient dialogues.

The **model architectures** use 4-layer feed-forward multi-layer neural net (one per domain), trained with stochastic gradient descent, where nodes in the input layers depend on the vocabulary of each agent. The use of convolutional neural nets or other types of neural nets is left as future work. They include 2 hidden layers with 100 nodes per layer, and Rectified Linear Units (RELUs) [34]. Other hyperparameters for each network include experience replay size=10000, burning steps=1000, discount factor=0.7, minimum epsilon=0.001, batch size={32, 2}, and learning steps=30000. The latter means that the minimum epsilon value is reached by 30000 steps.

Our **simulated dialogues** are driven by user goals (slot-values per domain), randomly initialised before each dialogue. First, each dialogue was restricted to two domains. Second, while system actions are chosen by the dialogue policies, system responses are generated from templates (verbalisations observed in demonstration dialogues). Third, while user actions are sampled from observed interactions in the demonstration dialogues, user responses are generated from stochastic templates with 10% of mumbling. The verbalisation of a user action is randomly selected from observed examples. Fourth, while random ASR confidence scores are used for training, actual confidence scores are used for testing. Our system retrieves live information from `http://www.bookatable.co.uk`, `www.reservetravel.com`, and `www.tvguide.co.uk`.

[1] `https://youtu.be/B5fZfZ-xaKM`



(a) All Updates (MB=32) (b) All Updates (MB=2) (c) Less Updates (MB=32) (d) Less Updates (MB=2)

Figure 3: *Example learning curves of DRL agents with all or less weight updates and with typical or limited minibatch of experience (MB). Notice that training with less weight updates only requires sub-linear growth resulting in faster learning—see dashed black lines.*

5. Experimental Results

We compared NDQN-Learning [20] against our proposed algorithm (see Section 3) using a multi-domain dialogue system. Our agents use the same data, resources and hyperparameters for training. The only difference between them is the learning algorithm (i.e NDQN-Learning with all or less weight updates) and size of minibatch of experience. The following metrics were used to measure system performance: avg. reward, learning time[2], avg. task success, and avg. dialogue length (i.e. avg. system actions per dialogue).

Figure 3 shows learning curves of NDQN with all or less weight updates, and with typical or limited minibatches of experience (MB=32, MB=2) from replay memory. Each plot reports results of a single run, out of 10 runs, over 20K simulated multi-domain dialogues. Our NDQN agents exhibited the following training times: (a) the baseline DRL agent with typical and limited minibatches of experience required **9.73 hrs** and **2.56 hrs**, respectively; and (b) the proposed algorithm with typical and limitied minibatches of experience required **3.51 hrs** and **1.76 hrs**, respectively. This represents an overall speedup of 5.5. Note also that while the growth of learning time of the DRL baseline is linear, the growth of learning time of the proposed algorithm is sub-linear—see black dotted lines. Faster training is explained by less weight updates in the proposed algorithm, which is improved by the limited minibatches of experience.

Although the currently generated dialogues using the trained policies seem reasonable (see video in footnote above), we integrated an additional baseline (KNN=K-Nearest Neighbour [35], 4 neighbours), which aims to behave as the example demonstration dialogues—see Appendix of [20]. We tested our dialogue policies over 10 runs of 1K dialogues for each type of behaviour and obtained the results shown in Table 1. We found that the performance of DRL agents using small minibatches is more stable than large minibatches—as judged by the average dialogue lengths and their standard deviations. According to pairwise comparisons and with focus on dialogue length, only the difference between NDQN agents with all vs. less updates (both using MB=32) is not significant (at $p < 0.8026$), the other differences between pairs of NDQN agents are significant (at $p < 0.05$)—according to Wilcoxon signed rank two-sided tests. No significant differences were observed in task success.

The following is an example dialogue generated with policies derived from NDQN-Learning with less weight updates and minibatch of 32. Square brackets denote system actions, double quotes denote system or user responses, and words beginning with an underscore encapsulate live web-based information according to the user requests. Notice that the system makes reasonable decisions even in the presence of mumblings.

[2]Ran on Intel Core i7-6950X CPU @ 3.00GHz x 10; 32GB RAM.

Algorithm/Metric	Task Success	Dialogue Length	Training Time
KNN	.932	15.33	NA
$NDQN^{all}_{MB=32}$	.999±.001	14.19±.410	9.73 hrs
$NDQN^{all}_{MB=2}$	.999±.001	13.76±.130	2.56 hrs
$NDQN^{less}_{MB=32}$	.997±.001	14.40±.608	3.51 hrs
$NDQN^{less}_{MB=2}$	.998±.001	13.86±.105	**1.76 hrs**

Table 1: *Test results averaged over 10 runs of the KNN baseline in contrast with NDQN with all/less weight updates and two sizes of minibatches of experience (MB)*

```
SYS:[Salutation(greeting)]"Welcome to MultiDS!"
SYS:[Provide(intro)]"You may ask for information about restaurants,
                                                    hotels and tv guide."
SYS:[Request(hmihy)]"How can I help you?"
USR:"bla bla"
SYS:[Apology(domain)] "Sorry, can you repeat your request?"
USR:"i need a hotel room"
SYS:[ExpConfirm(domain=hotels)] "Did you ask about hotels."
USR:"yes please"
SYS:[Call(domain=hotels)] ""
SYS:[Request(city,day,month,nights)] "Where and when would you like a hotel,
                                    and how many nights are you looking for?"
USR:"bla bla"
SYS:[Request(city,day,month)] "Where and when?"
USR:"on the 10th of february in paris"
SYS:[ImpConfirm(city=paris,day=10th,month=february)] "Okay, in paris on the
                                                          10th of february."
SYS:[Request(nights)] "How many nights do you want to stay?"
USR:"12 nights"
SYS:[ImpConfirm(nights=12)] "Okay, for 12 nights."
SYS:[Retrieve(info)] "Let me see."
SYS:[Provide(known)] "I found the following. _hotels."
SYS:[AskFor(more)] "Anything else?"
USR:"no thanks"
SYS:[Salutation(closing)] "You are welcome. Good bye!"
```

6. Concluding Remarks

This paper is about fast deep reinforcement learning for spoken dialogue systems. It uses a Network of DQN agents (called 'NDQN'), skips weight updates during exploitation of actions, and makes use of small minibatches of learning experiences from replay memories. Experimental results using simulations report that the proposed method can train policies faster than NDQN with all weight updates. Our experimental results provide evidence to suggest that faster learning can be achieved with little or no degradation of quality of policies, and that small minibatches lead to more consistent behaviour than larger ones. Although our results in three domains report about 5 times faster training, the speedup can increase as more domains are taken into account. Our proposed algorithm also showed to be more successful and efficient than a K-nearest neighbour baseline.

7. References

[1] S. Lange, M. A. Riedmiller, and A. Voigtländer, "Autonomous reinforcement learning on raw visual input data in a real world application," in *The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, June 10-15, 2012*, 2012, pp. 1–8.

[2] 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*, vol. 518, no. 7540, 02 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[3] 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*, vol. 529, no. 7587, pp. 484 – 489, 2016.

[4] H. Cuayáhuitl, "SimpleDS: A simple deep reinforcement learning dialogue system," *CoRR*, vol. abs/1601.04574, 2016. [Online]. Available: http://arxiv.org/abs/1601.04574

[5] H. Cuayáhuitl, S. Keizer, and O. Lemon, "Strategic dialogue management via deep reinforcement learning," *CoRR*, vol. abs/1511.08099, 2015. [Online]. Available: http://arxiv.org/abs/1511.08099

[6] S. Keizer, M. Guhe, H. Cuayáhuitl, I. Efstathiou, K.-P. Engelbrecht, M. Dobre, A. Lascarides, and O. Lemon, "Evaluating persuasion strategies and deep reinforcement learning methods for negotiation dialogue agents," in *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*, 2017.

[7] M. Fatemi, L. E. Asri, H. Schulz, J. He, and K. Suleman, "Policy networks with two-stage training for dialogue systems," 2016.

[8] K. Asadi and J. D. Williams, "Sample-efficient deep reinforcement learning for dialog control," *CoRR*, vol. abs/1612.06000, 2016. [Online]. Available: http://arxiv.org/abs/1612.06000

[9] T. Zhao and M. Eskénazi, "Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning," *CoRR*, vol. abs/1606.02560, 2016.

[10] O. Vinyals and Q. V. Le, "A neural conversational model," *CoRR*, vol. abs/1506.05869, 2015.

[11] I. V. Serban, T. Klinger, G. Tesauro, K. Talamadupula, B. Zhou, Y. Bengio, and A. C. Courville, "Multiresolution recurrent neural networks: An application to dialogue response generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[12] J. Li, W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao, "Deep reinforcement learning for dialogue generation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[13] J. Weston, S. Chopra, and A. Bordes, "Memory networks," *CoRR*, vol. abs/1410.3916, 2014.

[14] J. Perez, "Dialog state tracking, a machine reading approach using a memory-enhanced neural network," *CoRR*, vol. abs/1606.04052, 2016.

[15] J. Li, A. H. Miller, S. Chopra, M. Ranzato, and J. Weston, "Dialogue learning with human-in-the-loop," *CoRR*, vol. abs/1611.09823, 2016. [Online]. Available: http://arxiv.org/abs/1611.09823

[16] 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*, vol. abs/1507.04296, 2015. [Online]. Available: http://arxiv.org/abs/1507.04296

[17] 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*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[18] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *CoRR*, vol. abs/1511.05952, 2015. [Online]. Available: http://arxiv.org/abs/1511.05952

[19] 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*, vol. abs/1611.01606, 2016. [Online]. Available: http://arxiv.org/abs/1611.01606

[20] H. Cuayáhuitl, S. Yu, A. Williamson, and J. Carse, "Deep reinforcement learning for multi-domain dialogue systems," *CoRR*, vol. abs/1611.08675, 2016. [Online]. Available: http://arxiv.org/abs/1611.08675

[21] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. B. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," *CoRR*, vol. abs/1604.06057, 2016.

[22] H. Cuayáhuitl, S. Yu, A. Williamson, and J. Carse, "Scaling up deep reinforcement learning for multi-domain dialogue systems," in *IJCNN*, 2017.

[23] L. J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992. [Online]. Available: http://dx.doi.org/10.1007/BF00992699

[24] S. Adam, L. Busoniu, and R. Babuska, "Experience replay for real-time reinforcement learning control," *IEEE Trans. Systems, Man, and Cybernetics, Part C*, vol. 42, no. 2, pp. 201–212, 2012. [Online]. Available: https://doi.org/10.1109/TSMCC.2011.2106494

[25] 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.

[26] H. Cuayáhuitl, I. Kruijff-Korbayová, and N. Dethlefs, "Nonstrict hierarchical reinforcement learning for interactive systems and robots," *ACM Transactions on Interactive Intelligent Systems (TiiS)*, vol. 4, no. 3, 2014.

[27] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 5 2015.

[28] C. M. Bishop, *Pattern recognition and machine learning, 5th Edition*, ser. Information science and statistics. Springer, 2007. [Online]. Available: http://www.worldcat.org/oclc/71008143

[29] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

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

[31] A. Karpathy, "ConvNetJS: Javascript library for deep learning," 2015, http://cs.stanford.edu/people/karpathy/convnetjs/.

[32] H. Cuayáhuitl, N. Dethlefs, H. F. Hastie, and X. Liu, "Training a statistical surface realiser from automatic slot labelling," in *Spoken Language Technology Workshop (SLT)*, 2014.

[33] N. Dethlefs, H. W. Hastie, H. Cuayáhuitl, and O. Lemon, "Conditional random fields for responsive surface realisation using global features," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL*, 2013, pp. 1254–1263. [Online]. Available: http://aclweb.org/anthology/P/P13/P13-1123.pdf

[34] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[35] D. Aha and D. Kibler, "Instance-based learning algorithms," *Machine Learning*, vol. 6, pp. 37–66, 1991.