Learning Conversational Web Interfaces

Izzeddin Gur*
Google
izzeddin@google.com

Xifeng Yan
University of California, Santa Barbara
xyan@cs.ucsb.edu

Abstract

Automating user tasks with natural language instructions, such as booking a movie ticket, while keeping them engaged is a nontrivial and open problem. Previous work has focused on a particular scenario where users need to give entire instructions before a task can be handled. Aside from the difficulty of uttering a long instruction, this setup is also less realistic as the instructions could depend on future observations and needs to be delayed. In this work, we introduce the dialogue-based web navigation problem where the objective is to fulfill a hidden user goal by having multi-turn conversations with users and navigating a given web page simultaneously. We study joint learning of dialogue and navigation policies using reinforcement learning with actor-critic method. An architecture where user dialogue and web page observations are attentively encoded into policy actions is developed. We build a novel dialogue-based web environment by wrapping a user simulator and the Fandango movie ticket booking website into a single environment. We evaluate the performance of our models and discuss their biases and shortcomings.

1 Introduction

With the growing popularity of dialogue systems, their capabilities are challenged and improved in a broader set of new tasks. A recent example is Google's Duplex system which is able to have voice-enabled conversations with users to automate a range of tasks such as managing appointments. [2] In this work, we are interested in adding yet another capability to task-oriented dialogue systems by enabling them to navigate web pages while interacting with real users.

Recent work on navigating web pages [1, 2, 3, 4] focuses on a narrow scenario: A user needs to decide and utter an entire instruction to the system which then runs in a closed loop without any user interaction. Consider the task of booking a movie ticket on Fandango website. To abide by the previous assumption, we need to know which movie plays in which theatre and at what time that becomes infeasible to satisfy without first navigating the website. We argue that it is crucial to incorporate users into the loop to make the task scale to more realistic use-cases.

To address the aforementioned challenges, this paper introduces a new task: Navigating a designated web page to accomplish a hidden user goal by having multi-turn dialogues with users. We first develop a novel dialogue-based web navigation environment by carefully wrapping the Fandango movie ticket booking website and a rule-based user simulator into a single Gym environment [5]. We keep the core structure and look-and-feel of the Fandango website to enable more realistic experimentation and transferability of the trained models to real websites. A system can seamlessly interact with the user simulator and web environment to collect and learn from a rich set of experiences.

In addition to the introduced task and environment, we also train a new policy network that can learn actions from dialogue and web page observations. We first decompose the policy into two entangled

*Work done while the author was in University of California, Santa Barbara

[2] https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Figure 1: A dialogue with an example web page flow. Detected slot names and values from natural language responses are shown underlined. At time step-2 and time step-3, the system decides to *skip* interacting with user and continues with acting on the web page (A_2 and A_3).

sub-policies — dialogue policy, which outputs dialogue responses, and navigation policy, which navigates the web environment as suggested by the dialogue. A dialogue response is represented as a single (act, slot, value) tuple while a web page is navigated by interacting with HTML elements using keyboard or mouse. Both policies share the same low-level layers which encodes dialogue and web page observations into hidden representations by jointly attending different parts of the inputs; output layers are adapted to generating corresponding actions. Each policy runs continuously with up-to-date observations until a special *skip* action is generated which skips executing the corresponding policy for the current time step.

We train our policy network via actor-critic method with Proximal Policy Optimization (PPO) [6]. We first pre-train the network with behavioral cloning (BC) by collecting a small set of labeled dialogues (125 dialogues). We then fine-tune the pre-trained model using a combined objective which is the sum of the PPO objectives for each sub-policy. Each sub-policy is trained with a shared reward signal which is collected from web environment after performing an action. We test both BC and PPO policies with simulated and real users using task success metric, whether a ticket is successfully booked or not. While our policy network trained with PPO can successfully learn from a single reward and increase the performance of BC from 54% to 72%, there is still a large room for improvement. We believe there is a large set of research areas that our task might benefit from: Semi-supervised learning from available unlabeled task-oriented dialogue corpora, joint training of user simulator with dialogue and navigation policies, transferring learned policies into different web navigation tasks, etc. While our problem domain is web navigation, the policy architecture in this work might also be relevant to other reinforcement learning problems where a dialogue policy is learned from structured observations.

2 Task Definition

Given a dialogue and a web page, our task is to jointly learn a dialogue policy and a web navigation policy. At each time step, the system generates a response according to the dialogue policy and updates dialogue with the latest user and system responses. With the updated dialogue, the system navigates the web page according to the navigation policy and collects a new page and a reward.

For the example in Figure 1, the user initiates a dialogue with an *intent* and the system takes the lead. Based on the current observation of the web page, the dialogue policy generates a new request R_1, user responds with U_1, and the system collects a reward r_1 by executing a navigation action A_1 on the page. In this example, the system asks *movie* name and also `clicks` on the search field to open a new page.

When we come to time step-2, the dialogue policy decides to reflect previous user response on the web page before continuing with user dialogue; hence *skips* interacting with user. Navigation policy refers back to the dialogue and suggests the system to type the movie name into the `<search-2>` field. This generates a list of candidate movies to select from. By comparing each candidate movie with previous user responses, the system clicks on the corresponding element in time step-3. Note

that slot names and values do not necessarily match the attributes of elements in the web page and our policies need to learn their relationship (ex. *movie* and `search`). It is important to incorporate dialogue history when navigating a page as actions on the page might depend on previous turns.

We represent a dialogue as a list of last T user and system responses. A dialogue action (or system response) can be in dialogue act level which is represented as an *(act, slot, value)* tuple or in natural language level which is transformed into the former using a slot filling model. In our example, we represent R_1 as *request(movie)* and dialogue corresponds to $[U_0, R_1, U_1]$. Following previous work [2, 4, 3], we represent each observation of the web page as a Document Object Model (DOM) — a list of HTML elements linked in a tree structure. Each element in the DOM tree is denoted by a list of attributes: *tag, value, label, id, and text*. There are two different navigation actions: (i) Clicking an element or (ii) Typing the value of the requested slot from the last turn into an element. When the system executes a navigation action, attributes of elements as well as the structure of the DOM tree might alter as seen in Figure 1. At each time step, we constrain the system to generate at most one dialogue and one navigation actions. We also define *skip* actions for both dialogue and navigation sub-tasks that skips executing the corresponding policy for the current time step. We name a time step as a *turn* if there is a system-user conversation. As an example, in time step-1 the system generates both a dialogue (R_1, hence a turn) and a navigation (A_1) action while in time step-2 it generates a dialogue skip action and stops interacting with user.

3 Dialogue-Based Navigation Environment and Data Collection

We address the design of our web environments and dialogue setup that enables scalable training and evaluation of dialogue and navigation policies.

In this work, we focus on a *movie ticket booking* domain and design a new web environment in the Miniwob [1] framework by carefully cloning the Fandango website [3]. We preserve the core functionality of the Fandango while also unlinking some of the noisy hyperlinks that are unrelated for our task such as the "Help" page. There are several different ways of booking a movie ticket in Fandango, all of which are captured in our environment design such as typing into the search field or navigating to the movies page to find the correct movie. We build a database with approximately 500 different movies and 300 different theatres which are randomly populated with a set of available days, times, and number of tickets at the beginning of each episode. At each episode, a new web site and a new user goal is created by randomly sampling from this database. This randomness prevents policies from overfitting to the environment design and also enables evaluating the performance under stochasticity.

To automate the training and evaluation of our policies, we follow the traditional task-oriented dialogue literature and implement a simple yet extendable rule-based user simulator for the movie ticket booking domain. A new user goal as a list of slot and value pairs is generated at the beginning of each episode. After each system response, user simulator returns a response by selecting correct values from user goal as in Figure 1. To understand the performance of policies under more realistic scenarios, it is also important to generate natural language utterances for user and system responses. We use a template based natural language generation (NLG) approach by extracting templates from an existing dialogue dataset collected for a general purpose movie ticket booking task [7] and filling a randomly picked template for a given act-level input. Each natural language response in this dataset is annotated with the slot names, values, and corresponding substrings in the response. We use *inform* and *request* as the set of acts and *movie*, *theatre_name*, *day*, *time*, and *number_of_tickets* as the set of slots in this work.

While it is possible to train policies purely from environment rewards via trial-and-error, current reinforcement learning methods such as actor-critic algorithms exhibit high variance and low sample efficiency. Collecting a large dataset for every task is infeasible yet a small set of labeled data was important to have a reasonable performance and make more informative conclusions. We collected a dataset of 125 different dialogues with 884 time steps using Wizard-of-Oz paradigm [8]. The users are given a random goal, hidden from the wizards, as in Figure 1 and asked to type responses to wizard requests. The wizards are given the main page and asked to request values from users and also navigate the page based on user responses until the correct ticket is booked. Note that the collected data has all the labels to train both dialogue and navigation policies.

[3] https://mobile.fandango.com/

4 Learning Dialogue and Navigation Policies

4.1 Notations

Let's assume that we are at time step-t. We denote a dialogue by D_t which is a list of previous user and system responses. i-th response in dialogue is denoted by $D_t[i]$ and acts, slots, and values are represented as a, s, and v, respectively. Following recent work [3], we linearize tree structured representation of web page observations into a sequence of elements and denote by S_t. j-th element in this sequence is denoted by $S_t[j]$ and an attribute is represented as $attr$.

4.2 Neural Policy Architecture

We use an actor-critic algorithm which requires the training of two networks — a policy network, mapping observations into actions, and a value network, predicting sum of rewards from a given observation. We use two policy networks with similar architectures for mapping dialogue and web page observations to dialogue and navigation actions. Both policy networks use the same type of observations as inputs but with two different outputs; dialogue or navigation action. Value network takes the same input as policy networks and outputs a value for the current observation. All policy and value networks share the same low-level layers but output layers are trained separately.

4.2.1 Policy Network

There are three different components in our policy network architecture (Figure 2): (i) Low-level shared sub-network to jointly encode dialogue and web page observations, (ii) Output layers for dialogue and navigation policies, and (iii) Output layer for value prediction.

Encoding Dialogue and Web Page Observations
Each act, slot, and value is first embedded into fixed-length vectors by taking mean of the corresponding token embeddings in each sequence. Following [9], if any token in an act, slot, or value is found in any element in current web page, the exact match flag (f_a, f_s, or f_v, respectively) is activated and concatenated to corresponding embeddings. We encode a single turn i by concatenating the embeddings of the corresponding act-level representations and using a 2-layer neural network:

$$h_i = NN([e_a, e_s, e_v]) \tag{1}$$

$$M_i^D = NN([h_i, max(f_a, f_s, f_v)]) \tag{2}$$

where NN is a linear transformation followed by a ReLU activation function, e denotes an embedding vector with flags, $[.]$ denotes vector concatenation, and $max(.)$ denotes an activation if any of the input flags is activated. A dialogue is represented using a memory where each cell in the memory represents the embedding of individual turns, i.e., M_i^D corresponds to i-th cell in the memory.

Similar to turn-level embeddings, we first embed each attribute by taking mean token embeddings and concatenating with a flag which is activated if any token in the attribute is found in any dialogue turn. Encoding of an HTML element j is computed by concatenating attribute level embeddings and using a 1-layer neural network:

$$E_j = NN([e_{1:5}, f_{1:5}]) \tag{3}$$

where $e_{1:5}$ and $f_{1:5}$ denote the sequence of embeddings and flags of 5 different attributes used in this work, respectively. Using the sequence of element vectors as input, we encode the linearized view of the web page into a memory representation using a bidirectional LSTM (biLSTM) network:

$$M_{1:L}^S = biLSTM(E_{1:L}) \tag{4}$$

where $E_{1:L}$ is the sequence of element embeddings and M_j^S is the memory cell corresponding to the j-th element in the web page.

While we can increase the representational power of this architecture by stacking more layers, dialogue and page encoders are still independent of each other which loses any alignment information in encoding layers. Inspired by recent sentence matching approaches [10], we practise a soft-alignment layer with attention mechanism. Given a memory index j, we generate a context vector by computing

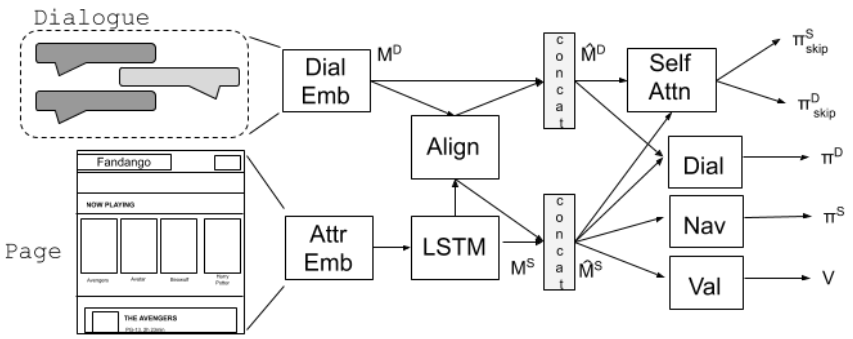

Figure 2: A general overview of our policy network. Top part denotes dialogue representation and bottom part denotes web page representation. Skip action is generated from both dialogue and page representations (top-right). Dial, Nav, and Val are indepedent neural networks that output dialogue policy, navigation policy, or value prediction, respectively (bottom-right).

attention [11] of M_j^S against each vector in M^D:

$$C_j = \sum_i \alpha_{j,i}.M_i^D \tag{5}$$

$$\alpha_{j,i} \propto exp(NN(M_j^S).Q.NN(M_i^D)) \tag{6}$$

where Q is a trainable matrix. We compute soft-aligned encoding of M_j^S as:

$$\hat{M_j^S} = [NN(M_j^S), C_j] \tag{7}$$

We also compute soft-aligned encodings of M^D using the same procedure with separate parameters. Alignment layers can also be stacked to yield a deeper representation of the current observation.

Policy Representations

Given final dialogue and page memory representations, $\hat{M}^D$ and $\hat{M}^S$, we first generate a single representation, m^D and m^S, by reducing each memory matrix into vectors using self attention mechanism as in [12].

We decompose the policy into two sub-policies — dialogue policy and navigation policy. Dialogue policy is composed of two different distributions over act and slot spaces. Both distributions are computed independently from the same web page and dialogue representations:

$$\pi^D = [\pi_{act}, \pi_{slot}] \tag{8}$$

$$\pi_{act} \propto exp(l([m^D, m^S])), \quad \pi_{slot} \propto exp(l([m^D, m^S])) \tag{9}$$

where l is a linear layer that outputs logits over corresponding action spaces.

Similarly, navigation policy is composed of two different distributions over available elements and `{click, type}` actions in the current observation:

$$\pi^S = [\pi_{action}, \pi_{element}] \tag{10}$$

$$\pi_{action} \propto exp(l(M^S)), \quad \pi_{element} \propto exp(l(M^S)) \tag{11}$$

First linear layer generates 2-D logits for each action and element pair while the second linear layer generates 1-D logits for each element. A navigation action is sampled by first sampling an element from $\pi_{element}$ and then sampling from the corresponding row of π_{action}.

Skip actions are defined as distributions over *{skip, continue}* for both dialogue and navigation sub-tasks and their distributions are computed as:

$$\pi^S_{skip}, \pi^D_{skip} \propto exp(l([m^D - m^S, m^D + m^S, m^D * m^S])) \tag{12}$$

At each time step-t, the same network architecture is run twice. The system first samples from dialogue skip distribution. If the action is *skip*, user interaction is skipped and dialogue is kept the

same; otherwise, a dialogue action is sampled from π^D and dialogue is populated with the latest user response. Using the updated dialogue, the system samples from navigation skip distribution. If the action is *skip*, navigating the page is skipped and page observation is kept the same; otherwise, a navigation action is sampled from π^S and a new page is collected. Note that both dialogue and navigation policies depend on the same web page observation while navigation policy uses the updated dialogue.

Value Predictions
To compute value prediction, we use self attention to generate a new hidden vector, m_V^D, from M^D. Value prediction is computed from this hidden vector as:

$$V_t = NN(m_V^D) \tag{13}$$

4.2.2 Training policy networks

We use Proximal Policy Optimization (PPO) [6] with advantage actor-critic [13] to train policy and value networks. We sample a new minibatch of transitions from replay buffer and define the following loss function for dialogue policy (for both act and slot distributions):

$$L_{PPO}^D = \mathbb{E}\left[min(\frac{\pi^D}{\pi_{old}^D}.A_t, clip(\frac{\pi^D}{\pi_{old}^D}, 1-\epsilon, 1+\epsilon).A_t\right] \tag{14}$$

where $\frac{\pi^D}{\pi_{old}^D}$ is the ratio of the probability of taking a dialogue action under policy π^D to the probability of taking an action under behavior policy π_{old}^D and ϵ is hyper parameter. A similar loss function for navigation policy is also computed and our final objective function becomes:

$$L_{PPO} = L_{PPO}^D + L_{PPO}^S + \|V_t - R_t\|^2 \tag{15}$$

where R_t denotes the cumulative reward from current state. A_t is the advantage function which is shared between both dialogue and navigation policies and computed as: $A_t = R_t - V_t$.

5 Experimental Results

We train and test our policies on a dialogue-based Fandango environment implemented in Miniwob framework.

5.1 Training Setup

Behavioral Cloning (BC). We pre-train our policy networks using the labeled dataset that we generated in Section 3. We have labels for each dialogue and navigation skip actions as well as actual dialogue and navigation actions. We train each supervised model for 50 epochs with a batch size of 32 and learning rate of 3e-4. We use 300 dimensions for word embeddings and 100 dimensions for hidden vectors. After every 5 epochs, we execute the last policy in the environment for 100 dialogues and accumulate rewards (development performance).
Reinforcement Learning (PPO). We initialize the parameters of our policies with behavioral cloning and further fine-tune for 250k samples. We use the same batch size, embedding and hidden dimensions but reduce the learning rate considerably, 1e-5, to avoid large gradient updates as is a common problem in reinforcement learning. We use ϵ of 1 in our PPO training. After every 5 batch of training, we execute the last policy in the environment for 20 dialogues to populate replay buffer (with a size of 5000 time steps) and to accumulate rewards (development performance).

5.1.1 Metrics

We report mean accumulated reward (R) and mean task success rate (TS), which equals 1 if movie ticket is successfully booked and 0 otherwise. We also report best reward (BR) and best task success rate (BTS) to understand the performance gap between running policy and the best policy. We analyze the distribution of collected dialogue and navigation trajectories to understand failure and bias cases. To evaluate the performance of our models with real users, we train a bidirectional slot filling network similar to [14] using labeled dataset generated in Section 3. User responses are converted into act-level responses and fed into our policies.

Best performing BC and PPO models are selected w.r.t. development performance and tested for 500 steps to collect test samples. Each setting is run at least 3 times and mean performance is reported.

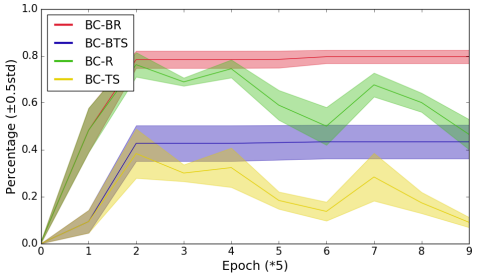

Figure 3: Development performance of BC over number of epochs.

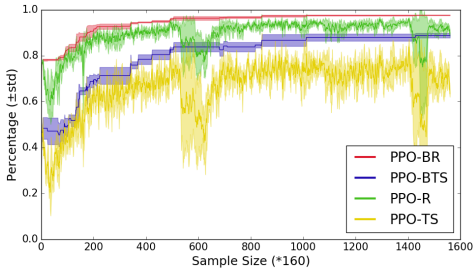

Figure 4: Development performance of PPO over sample size

Table 1: Testing Performance

Model	Reward	Task Success
BC - Sim	0.81	0.51
BC - Real	0.52	0.27
PPO - Sim	**0.93**	**0.72**
PPO - Real	0.73	0.43

Table 2: Error Analysis

Error Category	PPO	BC
Incorrect navigation action	0.3	0.52
Focus on wrong turn in dialogue	0.46	0.22
Missing request	0.68	0.62
Incorrect follow-up request	0.48	0.64
Request is not reflected by page	0.0	0.34

5.2 Performance of PPO and BC

In Figure 3 and 4, we evaluate the performance of BC and PPO policies over development set across training steps. Behavioral pre-training model (BC) provides a reasonable starting performance reaching 50% task success with only a handful of demonstrations.

After reaching the peak performance, BC starts to overfit even with a significant dropout regularization which is largely attributed to the small size of our labeled dataset. The large gap between total accumulated reward and task success shows that BC model learns partial sub-tasks such as searching and selecting the correct movie but fails to learn the overall task of booking tickets in almost half of the cases.

On the other hand, the performance of PPO steadily improves, surpassing the performance of BC with the best performing model reaching up to 90% task success over development set. We argue that the intrinsic exploration of PPO via sampling from policy distributions behave as a regularization step which keeps the development performance high for much longer. Similar to BC policy, there is a gap between total reward and task success performances albeit smaller in PPO policy.

Table 1 compares PPO and BC policies using testing performance over 500 and 50 dialogues with user simulator and real users, respectively. PPO policy, pre-trained with BC, improves the best task success performance with 18% and generalizes to unseen user goals and web page settings better. When evaluated with real users, the performance of our policies drop significantly where the most significant factor is noisy predictions of slot filling model. In majority of the error cases, slot filling model produced no outputs.

5.3 Error Analysis

We randomly sampled 50 unsuccessful testing dialogues from both PPO and BC and analyzed the types of errors in Table 2. While first two cases mostly reflects the ability of a model to understand the dialogue, the last three cases are mostly related to understanding the current page.

In some errors, user responses are reflected into incorrect navigation actions such as clicking on the incorrect date. This error is more apparent in BC policy showing the ability of PPO to match dialogue responses to navigation actions better. The most prominent error in both PPO and BC models is missing a required dialogue response such as theatre name. One plausible reason is that, when the informed time of the movie is unique in the current page, only a single theatre has exactly the same time slot, both models tend to ignore asking theatre name and overfit. Another error case is focusing on the wrong dialogue turn where models occasionally use an earlier turn to act on the web page. In majority of the cases, the problem is caused by having multiple flags activated at the same time

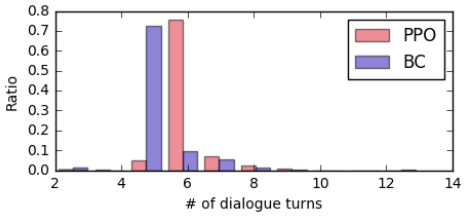

Figure 5: Ratio of average number of turns per dialogue

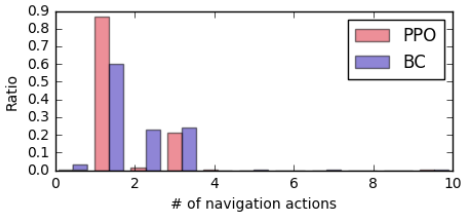

Figure 6: Ratio of average number of navigation actions per time step

in dialogue memory. When the date of the movie is a single digit, policies sometimes misinterpret this signal as the number of tickets. We also observe that in 34% of the cases, BC makes a request which is unrelated to the current page. As an example, BC sometimes request the theatre name on the final ticket selection page which populates the dialogue with a noisy entry and causes performance degradation. Note that an error can belong to multiple categories.

5.4 Distributional Characteristics of PPO and BC

In Figures 5 and 6, we show the distribution of average number of dialogue turns and average number of navigation actions per time step. In general, PPO tends to interact with users longer as longer dialogues are not externally penalized; while BC conforms to the labeled dataset and has 5-turns on average. Since there are multiple trajectories that a policy can follow, we observe that PPO tends to focus more on a main trajectory; hence has a more peaked navigation action distribution (Figure 6).

We also examine the most frequent dialogue and navigation trajectories. We represent a dialogue trajectory with the sequence of requested slots and a navigation trajectory with the sequence of tags of visited HTML elements. Similar to previous observation, PPO has longer dialogue trajectories; however the overall structure of the most prominent dialogue trajectory is common between PPO and BC. Both models favor a unique navigation trajectory which begins with searching movie name and clicking on the correct movie from a drop-down list of candidates. The most common dialogue and navigation trajectories are $movie \rightarrow date \rightarrow time \rightarrow num\_tickets$ and $input\_text^{(2)} \rightarrow h4 \rightarrow span^{(3)} \rightarrow label$ where (x) means the last item is repeated x times.

6 Related Work

Web Navigation. Liu, et al. (2018) and Shi, et al. (2018) both uses human demonstrations to improve the performance of reinforcement learning models. Gur, et al. (2019) uses curriculum learning and shallow encoding to guide the agent towards high potential states. All three models in [3, 2, 4] use DOM tree observations as in this work while [1] uses input pixels with manual features. Jia, et al. (2019) uses a graph neural network approach to encode DOM tree observations while in [3] a linearized view of DOM tree structure is used similar to ours.

Task-Oriented Dialogues. Recently, modular architectures of dialogues systems are replaced with end-to-end trainable neural network models [15, 16, 17]. Peng, et al. (2017) proposed a hierarchical reinforcement learning model for training dialogues across multiple time scales. Lei, et al. (2018) developed a holistic seq2seq approach by extracting belief spans for natural language responses. Similar to our memory representation, Eric, et al. (2017) and Bordes, et al. (2016) use memory networks to store information and repeatedly query and reason over.

7 Conclusion

We introduced a new problem of navigating web pages to fulfill unknown user goals while having multi-turn conversations with users. We built a new dialogue-based environment by combining Fandango movie ticket booking website with a rule-based user simulator. By jointly encoding dialogue and web page observations, we developed a new policy architecture with shared sub-structures between dialogue and navigation sub-policies. We trained our policy using PPO and BC and evaluated with user simulator and real users. We showed that while PPO significantly improves the performance over BC, there is still a large gap to fully solve the task. As a future work, we plan to develop augment our task with diverse set of environments and user simulators. We also plan to build end-to-end models that directly operates on natural language responses.

References

[1] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[2] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *Proceedings of the International Conference on Learning Representations*, 2018.

[3] Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. Learning to navigate the web. In *International Conference on Learning Representations*, 2019.

[4] Sheng Jia, Jamie Ryan Kiros, and Jimmy Ba. DOM-q-NET: Grounded RL on structured language. In *International Conference on Learning Representations*, 2019.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[7] Pararth Shah, Dilek Hakkani-Tür, Gokhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry Heck. Building a conversational agent overnight with dialogue self-play. *arXiv*, 2018.

[8] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve J. Young. A network-based end-to-end trainable task-oriented dialogue system. *CoRR*, abs/1604.04562, 2016.

[9] Yichen Gong, Heng Luo, and Jian Zhang. Natural language inference over interaction space. *CoRR*, abs/1709.04348, 2017.

[10] Seonhoon Kim, Jin-Hyuk Hong, Inho Kang, and Nojun Kwak. Semantic sentence matching with densely-connected recurrent and co-attentive information. *AAAI*, abs/1805.11360, 2019.

[11] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[12] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130, 2017.

[13] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[14] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tür, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, and Geoffrey Zweig. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23:530–539, 2015.

[15] Bing Liu, Gokhan Tur, Dilek Hakkani-Tur, Pararth Shah, and Larry Heck. End-to-end optimization of task-oriented dialogue model with deep reinforcement learning. In *NIPS Workshop on Conversational AI*, 2017.

[16] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve J. Young. A network-based end-to-end trainable task-oriented dialogue system. *CoRR*, abs/1604.04562, 2016.

[17] Bing Liu and Ian Lane. End-to-end learning of task-oriented dialogs. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 67–73, New Orleans, Louisiana, USA, June 2018. Association for Computational Linguistics.