A Reinforcement Learning Approach for Dynamic Search

Georgetown University at TREC 2017 Dynamic Domain Track

Zhiwen Tang
Georgetown University
zt79@georgetown.edu

Grace Hui Yang
Georgetown University
huiyang@cs.georgetown.edu

Abstract

TREC Dynamic Domain (DD) track is intended to support the research in dynamic, exploratory search within complex domains. It simulates an interactive search process where the search system is expected to improve its efficiency and effectiveness based on its interaction with the user. We propose to model the dynamic search as a reinforcement learning problem and use neural network to find the best policy during a search process. We show a great potential of deep reinforcement learning on DD track.

1. Introduction

TREC 2017 Dynamic Domain (DD) track simulates a professional search scenario where the search system needs to satisfy user's hierarchical information need in multiple rounds of search. This track target at search in within specific complex domains. User's information need are usually informational and composed of multiple aspects. Participant systems are expected to learn user's intent from previous interaction and find relevant documents with high efficiency.

In DD, a program called "Jig"[1] simulates user's response. At the beginning of each search topic, the participant systems receive the topic name (query) , a high-level description of user's information need. Each topic consists of several subtopics, but the name and the number of subtopics are not known to the participant systems. Then the participant systems need to find related documents in the next few rounds of retrieval. In each round, a participant system can return up to 5 documents to the simulated user (Jig). Jig responds with the relevance information regarding documents it receives. The feedback from Jig provides detailed relevance information, including graded relevance score regarding each subtopic for passages in the returned documents. Participant systems can then adjust its search algorithm based on the feedback, or choose to stop if they believe no relevant documents can be found in later rounds.

In 2017, DD track provides 60 search topics (queries) in New York Times Annotated Corpus [1]. Three metrics, i.e. Cube Test (CT) [2], Session-DCG (sDCG) [3] and Expected Utility (EU) [4], including their raw scores and normalized scores [5], are used for evaluation. These metrics models the outcome of search and the effort of user from different perspectives.

In our work, we model the dynamic search as a reinforcement learning problem. We employ a Deep Q-learning Network [9], which is successful in many reinforcement learning tasks, to optimize the search process. We show that Deep Reinforcement Learning has a great potential in dynamic search and it has hugely improves the performance of dynamic search system especially in terms of efficiency.

[1] https://github.com/trec-dd/trec-dd-jig

2. Reinforcement Learning Framework

The setting of DD shares many similarities with Reinforcement learning [6]. In DD track, the search system interacts with the simulated user and makes a series of decisions, such as whether to stop the search and how to rerank the documents. The search system is expected to make optimal decisions to maximize the evaluation scores. In reinforcement learning, an agent interacts with the environment and during every interaction, the agent need to take an action. The environment responds the action with a reward and the agent also need to update its internal state. The goal of reinforcement learning is to find the optimal policy that guides the agent to maximize the accumulated reward in the long term.

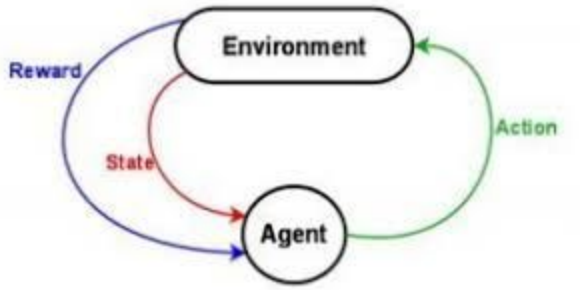

Figure 1. Framework of Reinforcement Learning

These similarities inspire us to model the dynamic search as a reinforcement learning problem. In this section, we will first review the basic concepts in reinforcement learning, then introduce one the most successful deep reinforcement learning framework, Deep Q-learning Network. Our approaches will be described in the following sections of this paper.

2.1. Markov Decision Process and Q learning

Markov Decision Process (MDP) is usually used for model a reinforcement learning problem. An MDP can be represented as a tuple $< S, A, T, R >$. The meaning of each symbol is described as follows:

S: the set of states (s). State is agent's belief about the its status in the environment.
A: the set of actions (a). Action is the agent's behavior under a given state. An action will result in the state transitions and bring in reward.
T: the transition function of the state. Transition function defines the state transition brought by the action of agent. That is, $T: S \times A \rightarrow S$, $s_{t+1} = T(s_t, a_t)$
R: the immediate reward function. The environment response the agent's actions with immediate rewards. Reward describes how "well" the action is. $R(s,a)$ is a scale value. The agent's ultimate goal is to maximize the accumulated reward in the long term.

While the reward function represents the immediate effects of an action, a value function describe the expected cumulative reward for the current state or action. The action value function (Q function) evaluates the expected long term reward in terms of an action in the given state. That is

$$Q(s_t, a_t) = E[r_{t+1} + \gamma * max_{a'} Q(T(s_t, a_t), a')]$$

Q learning is the approach that directly approximate the Q function, from which the optimal policy can be derived. It is proposed by Watkins and Dayan [14]. The Q function is updated in a step-by-step style. The updated rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \, max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

Where γ is the discounting factor and α is the learning rate.

2.2. Deep Q learning Network (DQN)

As the rise of Deep Learning, Mnih et al. propose DQN, where the Q(s,a) is estimated using a deep neural network [9]. DQN also introduces two import mechanism to stabilize the training and improve the performance.

The first mechanism is double Q learning. DQN contains two neural networks. The first one is the Q network while the second one is the target Q network. During training, The parameters of Q network are updated with Stochastic Gradient Descent (SGD). But the parameters of the target Q network will be kept frozen for a while and every after a given interval, the parameters of target Q network will be synchronized with those of Q network.

The second mechanism is experience. During each step, the DQN agent choose the action with ε-greedy algorithm. The newly chosen action will change the state and brings in the reward. All these transitions will stored in a experience buffer. During training, each time a batch of transitions are sampled from the experience buffer and used for training the Q network.

The training of Q network is based on the loss function shown below:

$$L(\theta) = \sum_{(s,\, a,\, s',\, r) \in batch} (r + \gamma \, max_{a'} \, Q(s', a' \mid \theta^{-}) - Q(s,\, a \mid \theta))^2$$

Where θ are the parameters of the Q network and θ^{-} are the parameters for the target Q network. The expected output for given state action pair (s, a) is its Q value, i.e. $r + \gamma \, max_{a'} \, Q(T(s,\, a), a')$.

DQN has made huge success in many reinforcement learning tasks, especially in Game Playing [9]. Inspired by the huge success of DQN and the similarity between Dynamic Search and Reinforcement Learning, we propose our RL-based solutions, which are detailed in the following sections.

3. DQN for Dynamic Search

We propose two frameworks that using DQN for dynamic search. Two frameworks have same definition of reward, similar actions and different compositions of states.

3.1. Reward for Dynamic Search

Finding the key results for each topic is crucial for maintaining high efficiency in dynamic search, for which we "highlight" the passages with the highest relevance scores. For every passage, its relevance score (rating) is redefined as: $r' = r$ iff. $r = 4$ else $r' = 0.1 * r$ where r is the original rating for the passage. The incremental of the gain of Cube Test [2] based on the redefined relevance score is the gain for the current step.

3.2. Framework 1

In this framework, we use state-of-the-art deep learning techniques to embed words and sentences. The embedding results make up the state representation of the DQN agent. The actions for the agent are reformulating queries, including adding term, removing term, reweighting terms and stop search, all of those are detailed in [11]. The whole framework is shown in Figure 2.

The state is defined as a tuple, $S =< Query, Relevant\ Passages, Iteration\ number >$. All the words are first embedded with word2vec [12]. Since the length of query and relevance passages may vary, we use Long Short Term Memory (LSTM) [13] to encode the query and passages respectively. The LSTM takes in as input the sequence of word vectors, which are obtained in the previous step. Then the final output of LSTMs and the iteration number are concatenated to form the state.

The state representation goes through the hidden layer and the output layer to produce the estimated long term reward for each action. Each unit in the output layer corresponds to a possible action. The agent will make decisions based on the output estimated cumulative reward.

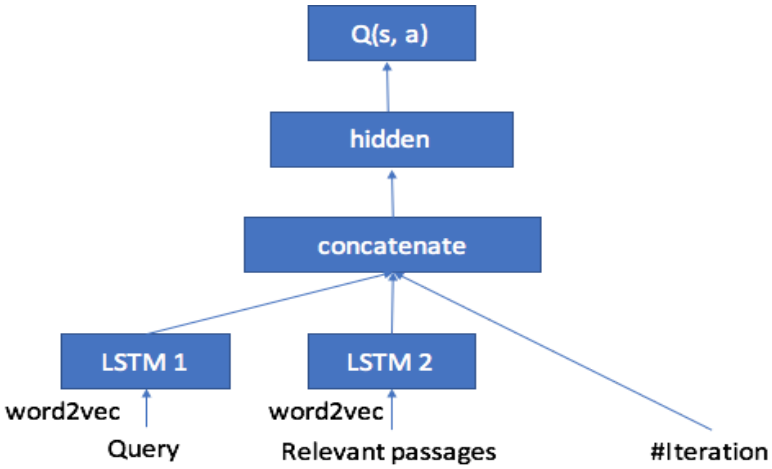

Figure 2. Framework 1

However, limited by the amount of available data in DD, we may have problem of training the LSTM, based on which we propose the second Framework, which uses simpler neural network and hand-crafted features.

3.3. Framework 2

In this framework, we handpick some features of the dynamic search process that we believe better represents the search status. Apart from all the actions defined in Framework 1, we also add another action, i.e. re-retrieval with the top 10 words with highest tf*idf values.

The state is defined as a tuple $S =< subtopic\ found\ flag, subtopic\ hit\ count, iteration\ number, miss\ count >$. The *subtopic found flag* is a boolean vector with a fixed length, where each entry indicates if the subtopic has been found or not. If an entry is 0, it means the subtopic has not been found so far or the subtopic may not even exist in current topic. The *subtopic hit count* is in the same length of *subtopic found flag*, and each entry is the number of documents have been found on the corresponding subtopic. *iteration number* is the same as in Framework 1. *miss count* is the number of iterations where no relevant documents are retrieved.

Different items in this tuple may go through different hidden layers. The final output layer is similar to that in Framework 1 where each unit corresponds with an action. The whole Framework is shown in Figure 3.

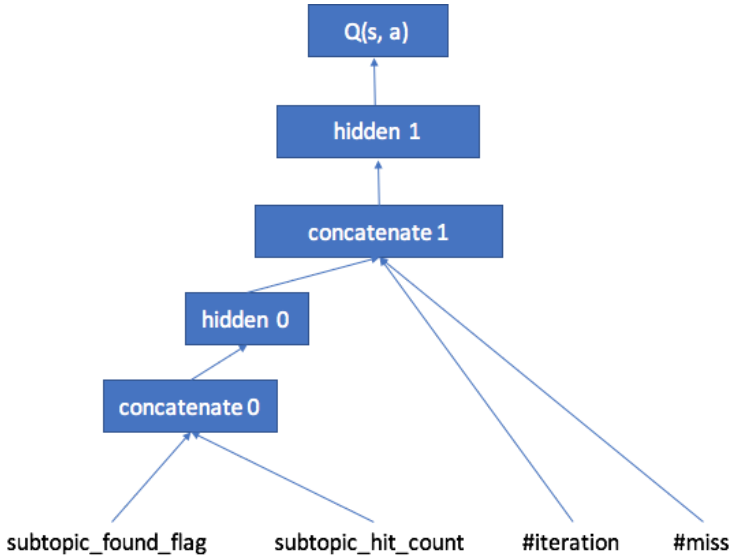

Figure 3. Framework 2

3.4. Implementation Details

K-fold Cross-validation: Since DD does not specify the training set and test set. We split the topics into 3 folds in equal sizes. Every time, the neural network is trained on 2 of them and tested on the remaining one.

DQN Parameters: The size of experience buffer is set to 10000, the learning rate of the neural network is set to 0.001, the discounting factor γ for MDP is set to 0.5. The target Q network and the Q network are synchronized every after 20 steps.

Other: We use galago[2] as our backend search engine and use the default structured query operator. We use gensim[3] to train the word2vec model and Keras[4] to build the deep neural network.

4. Results

We submitted 3 runs, *dqn_semantic_state*, *dqn_5_actions*, and *galago_baseline*. *dqn_semantic_state* uses Framework 1 and *dqn_5_actions* uses Framework 2. *galago_baseline* is the top 50 results of galago with no feedback information being used, which serves as the baseline for comparison.

The performance of three runs regarding Cube Test, session-DCG and Expected Utility and their normalized scores are shown in Figure 4 to Figure 9, more detailed results can be found in Table 10.

5. Discussion

Every metric reveals some different characteristics of these runs, which brings in very interesting discussions about our methods.

[2] https://www.lemurproject.org/galago.php
[3] https://radimrehurek.com/gensim/
[4] https://keras.io/

In terms of Cube Test, both frameworks surpass the baseline. Especially *dqn_semantic_state*, which uses Framework 1. It doubles the baseline in the end. It means that both frameworks improve the efficiency of the dynamic search system. The improvement on efficiency might come from more gaining of relevant information in given time. It may also come from early stopping on the topics where search system may not perform so well.

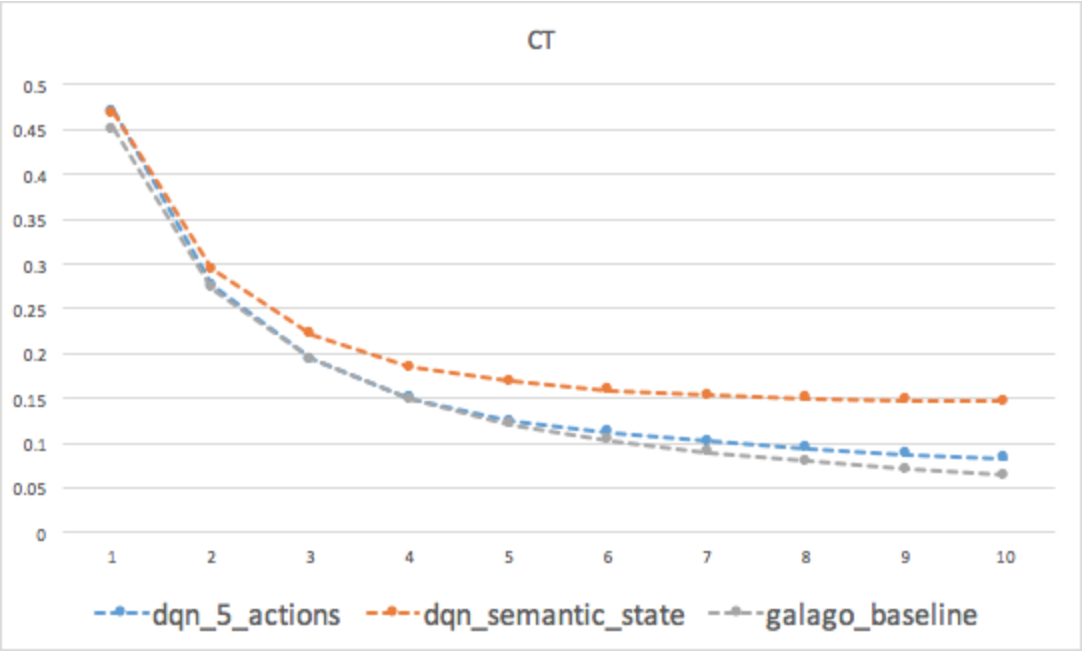

Figure 4. CT scores

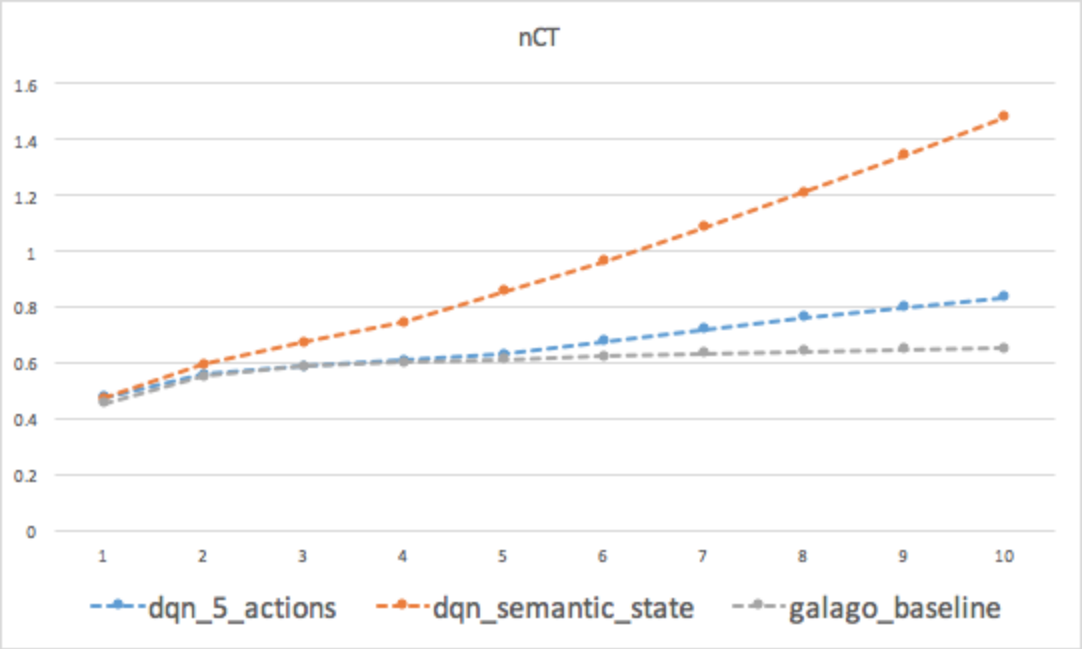

Figure 5. nCT scores

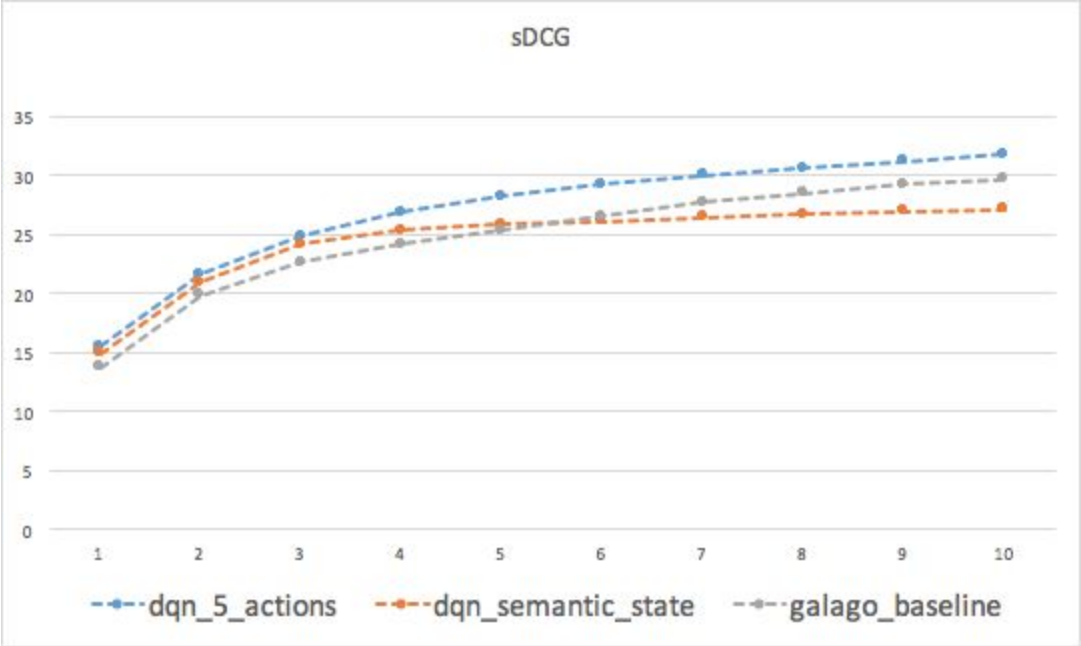

Figure 6. sDCG scores

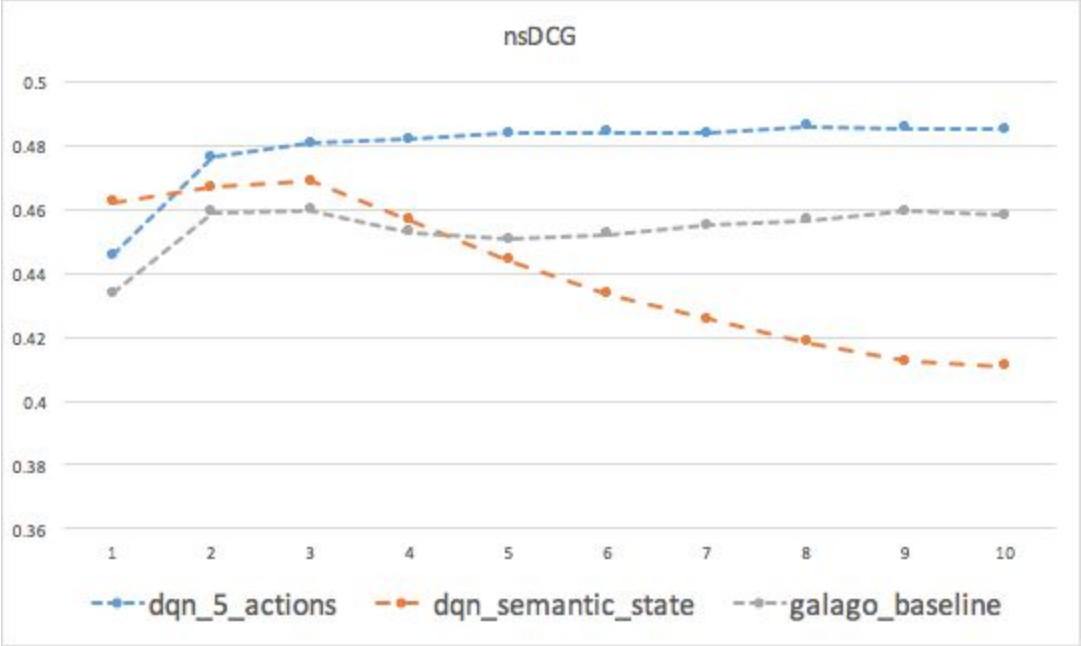

Figure 7. nsDCG scores

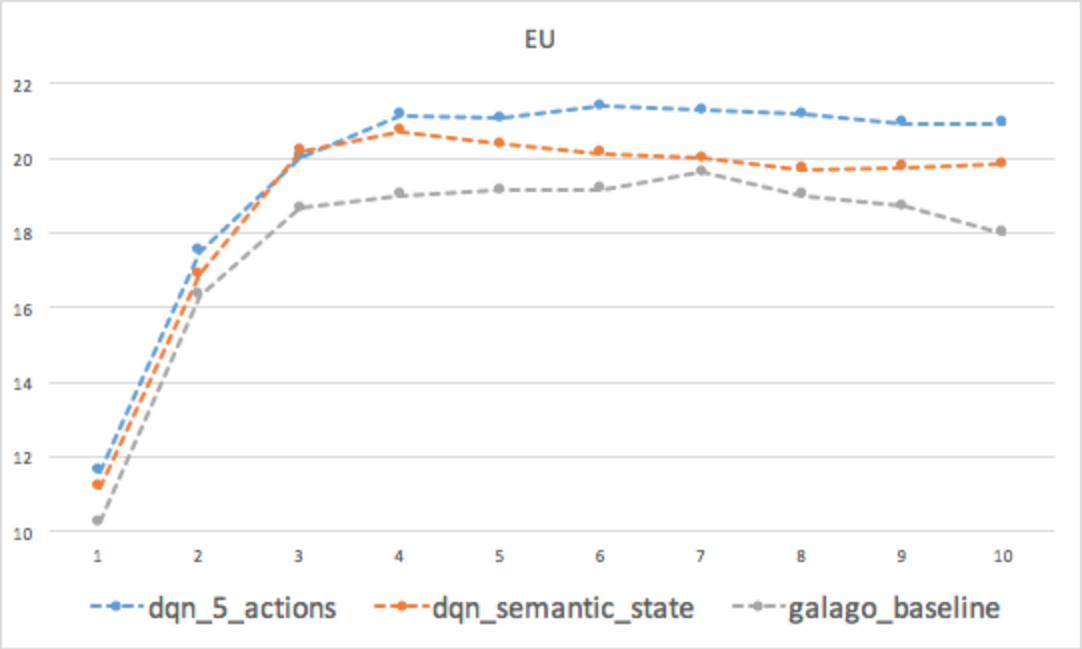

Figure 8. EU scores

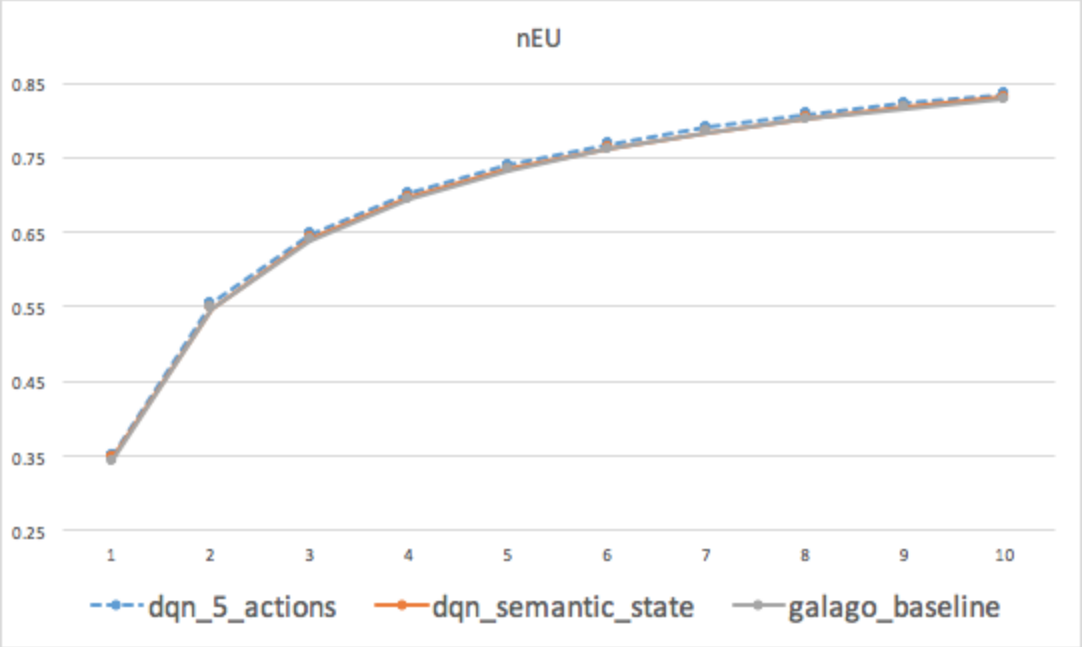

Figure 9. nEU scores

Session DCG gives more insight about the gaining of relevant information. It can be found that the session DCG score of *dqn_semantic_state* is below the baseline while *dqn_5_actions* is still better than the baseline. It can be inferred that the high performance of *dqn_semantic_state* in terms of Cube Test does not come from retrieving more relevant documents.

Expected Utility evaluates how well the search system balance the gaining of information and the effort of user. It is seen that both frameworks makes improvement over the baseline and *dqn_5_actions* is the best this time. It confirms again *dqn_semantic_state* achieve high performance in CT by early stopping while *dqn_5_actions* does find more relevant documents.

One of the major discoveries in our runs this year is the power of early stopping. A good stopping strategy can greatly improve the efficiency of search system and satisfies the user better. We also show the great potential of deep reinforcement learning in dynamic search. It found a stopping criterion that improve the efficiency this time. And it will be a much more interesting question that how to use it to find a good retrieval algorithm.

Acknowledgement

This research was supported by DARPA grant FA8750-14-2-0226 and NSF grant IIS-145374. Any opinions, ndings, conclusions, or recommendations expressed in this paper are of the authors, and do not necessarily reflect those of the sponsor.

Reference

[1] Sandhaus, Evan. "The new york times annotated corpus." Linguistic Data Consortium, Philadelphia 6, no. 12 (2008): e26752.

[2] Luo, Jiyun, Christopher Wing, Hui Yang, and Marti Hearst. "The water filling model and the cube test: multi-dimensional evaluation for professional search." In Proceedings of the 22nd ACM international conference on Information & Knowledge Management, pp. 709-714. ACM, 2013.

[3] Järvelin, Kalervo, Susan Price, Lois Delcambre, and Marianne Nielsen. "Discounted cumulated gain based evaluation of multiple-query IR sessions." Advances in Information Retrieval (2008): 4-15.

[4] Yang, Yiming, and Abhimanyu Lad. "Modeling expected utility of multi-session information distillation." In Conference on the Theory of Information Retrieval, pp. 164-175. Springer, Berlin, Heidelberg, 2009.

[5] Tang, Zhiwen, and Grace Hui Yang. “Investigating per Topic Upper Bound for Session Search Evaluation.” In Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval, pp. 185-192. ACM, 2017

[6] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1, no. 1. Cambridge: MIT press, 1998.

[7] Li, Yuxi. "Deep reinforcement learning: An overview." arXiv preprint arXiv:1701.07274 (2017).

[8] Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert et al. "Mastering the game of Go without human knowledge." Nature 550, no. 7676 (2017): 354-359.

[9] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning." Nature 518, no. 7540 (2015): 529-533.

[10] Thie, Paul R. Markov decision processes. Comap, Incorporated, 1983.

[11] Yang, Angela, and Grace Hui Yang. “A Contextual Bandit Approach to Dynamic Search.” In Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval, pp. 301-304. ACM, 2017

[12] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In Advances in neural information processing systems, pp. 3111-3119. 2013.

[13] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.

[14] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8, no. 3-4 (1992): 279-292.

Iteration	Run	CT	nCT	sDCG	nsDCG	EU	nEU
1	dqn_5_actions	0.4701114	0.4756687	15.4846276	0.4456217	11.6124865	0.3499484
	dqn_semantic_state	0.4683404	0.4742194	14.8987069	0.4620971	11.1862801	0.3473296
	galago_baseline	0.450791	0.4563555	13.7844676	0.4337153	10.248345	0.3431296
2	dqn_5_actions	0.2760114	0.5592316	21.6554912	0.4760824	17.4928539	0.5543615
	dqn_semantic_state	0.2938375	0.5938134	20.9093702	0.4666493	16.8901066	0.5473597
	galago_baseline	0.2722861	0.5507842	19.8558578	0.4589931	16.3226741	0.5476611
3	dqn_5_actions	0.1930831	0.5867918	24.8716411	0.4804548	20.0101208	0.6468404
	dqn_semantic_state	0.2217546	0.6720723	24.1273948	0.4687206	20.1680049	0.6419633
	galago_baseline	0.193695	0.5879702	22.6569676	0.4594243	18.6602872	0.639583
4	dqn_5_actions	0.1501568	0.6087354	26.8278014	0.4818966	21.1361679	0.7025435
	dqn_semantic_state	0.1846151	0.7457376	25.3275609	0.4564798	20.7058817	0.6966453
	galago_baseline	0.1484205	0.6008163	24.1543098	0.452551	18.9989242	0.6942974
5	dqn_5_actions	0.1242476	0.6294138	28.1966415	0.4836972	21.0581117	0.7395657
	dqn_semantic_state	0.169166	0.8536985	25.8106292	0.4440257	20.3719695	0.7338354
	galago_baseline	0.1211844	0.6132961	25.3420453	0.450617	19.1343022	0.7323573
6	dqn_5_actions	0.1114354	0.6770347	29.2426583	0.4840687	21.3944172	0.7682166
	dqn_semantic_state	0.1589661	0.9625451	26.1356308	0.4332171	20.1245082	0.7620878
	galago_baseline	0.102758	0.6240457	26.5015414	0.451918	19.1694004	0.7608299
7	dqn_5_actions	0.1014448	0.7187933	29.9681515	0.4836611	21.2873622	0.7899146
	dqn_semantic_state	0.1532954	1.0830598	26.4518386	0.425639	19.9831539	0.784234
	galago_baseline	0.0892321	0.6321475	27.6992079	0.4548045	19.6283697	0.7840226
8	dqn_5_actions	0.094049	0.7613281	30.6252363	0.4857005	21.1754035	0.8075685
	dqn_semantic_state	0.1496234	1.208232	26.6684806	0.4182104	19.6949104	0.8018887
	galago_baseline	0.0792522	0.6417305	28.4367084	0.4564056	18.9915583	0.8011377
9	dqn_5_actions	0.0877909	0.7992842	31.197493	0.4852854	20.9258899	0.8220898
	dqn_semantic_state	0.1475708	1.3405783	26.9146451	0.4124845	19.7538581	0.8171259
	galago_baseline	0.0710383	0.6471881	29.2236247	0.4592193	18.7172524	0.8159354
10	dqn_5_actions	0.082483	0.834239	31.7663899	0.4850046	20.9282911	0.8347376
	dqn_semantic_state	0.1464314	1.4784141	27.1194546	0.4107508	19.8255847	0.8302033
	galago_baseline	0.0643294	0.6512856	29.6418512	0.4581143	17.9727267	0.8280541

Table 1. evaluation results in first 10 iterations