DOM-Q-NET: GROUNDED RL ON STRUCTURED LANGUAGE

Anonymous authors
Paper under double-blind review

ABSTRACT

The World Wide Web is a rich repository of knowledge about the real world. The ability for agents to interact with the web would allow for significant improvements in knowledge understanding and representation learning. However, web navigation tasks are difficult for the current deep reinforcement learning (RL) models due to the large discrete action space and the varying number of actions between the states. In this work, we introduce DOM-Q-NET, a novel architecture for RL-based web navigation to address both of these problems. DOM-Q-NET utilizes a graph neural network to represent tree-structured HTML along with a shared state space across multiple tasks. We show 2x improvements in sample efficiency when training in the multi-task setting, allowing our model to transfer learned behaviours across tasks. Furthermore, we demonstrate the capabilities of our model on the WorldOfBits environments where we can match or outperform existing work without the use of expert demonstrations.

1 INTRODUCTION

Over the past years, deep reinforcement learning (RL) has shown huge success in solving tasks such as playing arcade games (Mnih et al., 2015) and manipulating robotic arms (Levine et al., 2016). Recent advances in neural networks allows RL agents to directly learn the control policies from raw pixels without feature engineering from human experts. However, most of the deep RL methods only focus on solving problems in either simulated physics environments where the inputs to the agents are joint angels and velocity, or simulated video games where the inputs are rendered graphics. Agents trained in these simulated environment have little knowledge about the rich semantics of the real world.

The World Wide Web (WWW) is a rich repository of knowledge about the real world. To navigate in this complex web environment, an agent needs to learn about the semantic meaning of text, images and the relationships between them. During navigation, each action corresponds to interacting with DOMs of tree structured HTML. Tasks like finding a friend on a social network, clicking into a web page on the WWW, and rating a place on Google maps can be considered as choosing a node from graphical structures and optionally feeding data to it.

Unlike Atari games, the difficulty of web tasks comes from their diversity, large action space and sparse reward signals. A common solution for the agent is to mimic the expert demonstration by imitation learning, and so efficiency of using demonstrations has been paid attention in previous works (Shi et al., 2017; Liu et al., 2018). However, the diverse set of tasks also include simple ones that agent can learn and generalize some actions to harder tasks. This capability is essential because the action space at each time step varies when visiting different web pages.

Two previous works (Shi et al., 2017; Liu et al., 2018) have evaluated RL-based web navigation by attempting to solve MiniWoB(Shi et al., 2017) benchmark tasks. Liu et al. (2018) achieved the state of the art performance with very few expert demonstrations, but their exploration policy requires action constraining set, hand-crafted with expert knowledge in HTML.

In this work, we focus on learning RL agents to navigate the web without human demonstration. Our contribution is to propose a modular Q network architecture with graph neural network (Scarselli et al., 2009; Li et al., 2016; Wang et al., 2018)as main backbone that can be trained to match or outperform existing work on MiniWoB without using any expert demonstrations. Essentially, it uses

message passing and readout of the node representations from 'local' representations of the web pages to produce 'neighbor' and 'global' representations. Three modules produce rich state and action representations fed into Q neural network streams. Different Q function streams for different action groups share those modules and can be jointly trained. Another contribution from our model is that it is the first architecture to demonstrate that learned behaviors on the web interface can be transferred, and jointly learned by performing multitask learning for web navigation tasks. We show that the multi-task agent achieves 2x sample efficiency comparing to single task training.

2 BACKGROUND

2.1 REPRESENTING WEB PAGES USING DOMS

We are interested in the web navigation setting where the agent can click and type. The Document Object Model (DOM) is a programming interface for HTML documents and it defines logical structure of such documents. Each DOM of HTML is connected in a tree structure, and so we can frame web navigation as accessing DOM and potentially modifying it by user input. As an element object, each DOM has a "tag" and attributes such as "class", "is focused", similar to objects in Object Oriented Programming. The values of attributes are used by browsers to render web page for users.

2.2 REINFORCEMENT LEARNING

In the traditional reinforcement learning setting, an agent interacts with an infinite-horizon, discounted Markov Decision Process (MDP) to maximize its total discounted future rewards. An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ where $\mathcal{S}$ and $\mathcal{A}$ are the state space and the action space respectively, $T(s'|s,a)$ is the transition probability of reaching state $s' \in \mathcal{S}$ by taking action $a \in \mathcal{A}$ from $s \in \mathcal{S}$, R is the immediate reward by the transition, and γ is a discount factor. Then we define the Q-value function for a tuple of actions to be $Q^\pi(s,a) = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s, a_0 = a]$, where T is the number of timesteps till termination. The formula represents the expected future discounted reward starting from state s, performing $\boldsymbol{a}$ and follow the policy until termination. The optimal Q-value function $Q^*(s,a) = max_\pi Q^\pi(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$ (Sutton & Barto, 1998) satisfies the Bellman optimality equation $Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')]$.

2.3 GRAPH NEURAL NETWORKS

For an undirected graph $G = (V, E)$, the Message Passing Neural Network (MPNN) framework (Gilmer et al., 2017) formulates two phases of forward pass to update the node-level feature representations h_v, where $v \in V$, and graph-level feature vector $\hat{y}$. Message passing phase updates hidden states of each node by applying a vertex update function U_t over current hidden state and message $h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$, where passed message m_v^{t+1} is computed as $m_v^{t+1} = \sum_{\omega \in N(v)} M(h_v^t, h_w^t, e_{vw})$. Here, N(v) denotes the neighbors of v in G, e_{vw} is an edge feature. This process runs for T timesteps. The readout phase uses readout function R, and computes graph level feature vector $\hat{y} = R(h_v^T | v \in G)$. Neural networks are used to parametrize each of the functions M_t, U_t and R_t.

2.4 PREVIOUS WORKS ON RL ON WEB INTERFACES

Shi et al. (2017) constructed benchmark tasks, Mini World of Bits, that consist of many toy tasks of web navigation . This environment supports providing both the visual and HTML representations of a web page. (Shi et al., 2017) showed a benchmark result that the agent with visual input does not solve most of the problems, even given the demonstrations. Then (Liu et al., 2018) proposed DOM-NET architecture that uses a series of attentions between DOM elements and goal, and workflow guided-exploration, which constrains the action space of an agent at each state. They achieved the new state of the art performance and sample efficiency by incorporating constrained action set handcrafted by a human expert. Unlike these previous works, we aim to tackle web navigation tasks without any human expert demonstration or prior knowledge.

3 NEURAL DOM Q NETWORK

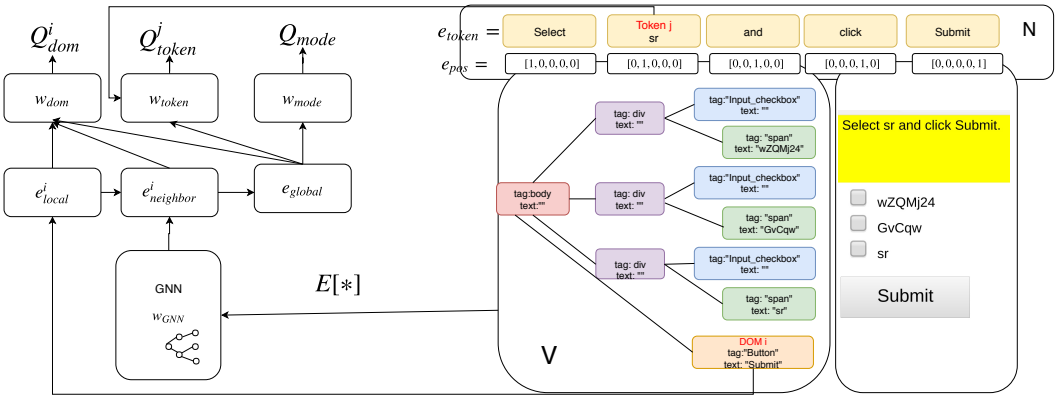


Figure 1: Given the web page on the right, its DOM tree representation is shown as a graph where each DOM represents a node from V. Different color indicates different tag attributes of DOMs. DOM are set as local module $\boldsymbol{e}_{local}$, and also propagated by a GNN to produce neighbor module $\boldsymbol{e}_{neighbor}$. Global module $\boldsymbol{e}_{global}$ is max-pooled from neighbor. Then Q_DOM stream uses all three modules. Here, Q value of "Submit button" is computed by this stream. Q value of 'sr' goal token is computed by Q_token stream

Consider the problem of learning an agent that navigates through multiple web pages or menus to locate a piece of information. Let V be the set of DOMs in the current web page. There are often multiple goals that can be achieved in the same web environment. We consider goals are presented to the agent in the form of a natural language sentence, e.g. "Select sr and click Submit". Let $\mathcal{G}$ represent the set of word tokens in the given goal sentence. The RL agent will only receive a reward if it successfully accomplishes the goal, i.e. it is a sparse reward problem. The primary means of navigation are through interaction with the buttons and the text fields on the web pages.

There are two major challenges in representing the state-action value function for web navigation: learning from the enormous action space; the number of actions can vary drastically between the states. We propose DOM-Q-NET to address both of the problems in the following.

3.1 ACTION SPACE FOR WEB NAVIGATION

Unlike typical RL tasks that requires choosing only one action a from an action space, $\mathcal{A}$, such as choosing one from all combinations of controller's joint movements for Atari game (Mnih et al., 2015), we frame acting on the web with three distinct categories of actions:

- DOM selection a_{dom} chooses a single DOM in the current web page, $a_{dom} \in V$. The DOM selection covers the typical interactive actions such as clicking buttons or check-boxes.
- Word token selection $a_{token} \in \mathcal{G}$ picks a work token from the given goal sentence to fill in the selected text box. The assumption that typed string comes from goal sentence aligns with previous work. Liu et al. (2018)
- Click or type mode $a_{mode} \in \{\text{click}, \text{type}\}$ action tells the environment whether the agent's intention is to "click" or "type" when interacting with the web page. We can represent a_{mode} as a binary action.

At each time step, the environment receives a tuple of actions, namely $a = (a_{dom}, a_{token}, a_{mode})$.

3.2 FACTORIZED Q FUNCTION

One way to represent the state-action value function is to consider all the permutations of a_{dom} and a_{token}. For example, (Mnih et al., 2015) used permutations to flatten out the combinations of joystick direction and clicking for atari games. However, in MiniWoB environment, this introduces

an enormous action space in $|V| \times |\mathcal{G}|$ number of states. The number of DOMs $|V|$ and $|\mathcal{G}|$ can reach up to 60 and 18 for most medium tasks and the total number actions can be over $1,000$ for some hard tasks.

To reduce the number of actions, we consider a factorized state-action value function where the action value of a_{dom} and a_{token} are **independent** to each other. Formally, we define the optimal Q-value function as the sum of the individual value functions of the three action categories:

$$Q^*(s,a) = Q^*(s, a_{dom}, a_{token}, a_{mode}) = Q^*(s, a_{dom}) + Q^*(s, a_{token}) + Q^*(s, a_{mode}). \quad (1)$$

Under the independence assumption, we can find the optimal policy by selecting the greedy actions w.r.t. each Q-value function individually. Therefore, the computation cost for the optimal action of the factorized Q function is linear in the number of DOM elements and the number of word tokens rather than quadratic.

$$a^* = \left(\arg\max_{a_{dom}} Q^*(s, a_{dom}), \arg\max_{a_{token}} Q^*(s, a_{token}), \arg\max_{a_{mode}} Q^*(s, a_{mode}) \right) \quad (2)$$

3.3 Learning state-action embeddings of web pages

Many web actions such as clicking different checkboxes and filling unseen type of forms share similar tag or class attributes. Our goal is to design a neural network architecture that can effectively capture such invariance for web pages and yet is flexible to deal with the varying number of DOM elements and goal tokens at different time steps. Furthermore, when locating a piece of information on the web, an agent needs to be aware of both the local information, e.g. the name of button and its surrounding texts, and the global information, e.g. the general theme, of the web page. The cue for click a particular button from the menu is likely scattered.

To address the above problem, we propose a graph neural network (GNN) based RL agent that computes the factorized Q-value for each DOM present in the current web page, called "DOM-Q-NET" shown in Figure 1. DOM-Q-NET uses additional information of tree structured DOM elements in HTML to guide the learning of state-action representations or embeddings $\boldsymbol{e}$, which is shared among all its factorized Q outputs. Explicitly models the HTML structures provide relational information among the DOM elements to the RL agents. Given a web page, DOM-Q-NET learns a concatenated embedding vector $\boldsymbol{e}^i = \left[\boldsymbol{e}^i_{local}, \boldsymbol{e}^i_{neighbor}, \boldsymbol{e}_{global}\right]$ from low level and high level modules correspond to node-level, and graph-level output of the GNN and are jointly shared and trained to represent the DOM $v^i \in V$:

Local Module $\boldsymbol{e}^i_{local}$ is simply the concatenation of each embedded attribute $\boldsymbol{e}_{Attr}$ of the DOM v^i, which includes the tag, class, focus, tampered and text information of the DOM element. In particular, we used a maximum of cosine distance between text and goal tokens to measure the 'soft' alignment of the DOM v^i with respect to the jth word token embedding $\boldsymbol{e}^j_{goal}$ in the goal. Liu et al. (2018) uses a direct alignment to obtain tokens that appear in goal but our method can detect synonyms that are not exactly matched.

$$\boldsymbol{e}^i_{local} = \left[\boldsymbol{e}^i_{Attr}, \max_j \left(cos(\boldsymbol{e}^i_{Attr}, \boldsymbol{e}^j_{goal}) \right) \right] \quad (3)$$

This module provides the action representation of clicking each DOM and is generally considered as the skip connection of GNN.

Neighbor Module $\boldsymbol{e}^i_{neighbor}$ computes the neighborhood context of the DOM v^i using a graph neural networks (GNN) with the weights $\boldsymbol{w}_{GNN}$ on the tree structure of the HTML page. Propagation model is a neural message passing system (Gilmer et al., 2017) between the nodes of the HTML of the web page. The initialization of the propagation is set to the local embedding. $\boldsymbol{m}^t$ is an intermediate state between propagation steps. We apply T number of propagation steps to obtain the final neigbhor embedding. We adopted GRU gated version (Li et al., 2016) for state updates between aggregation time steps.

$$\boldsymbol{m}^{i,t+1}_{neighbor} = \sum_{k \in N(i)} w_{GNN} \boldsymbol{e}^{k,t}_{neighbor}, \quad \boldsymbol{e}^{i,0}_{neighbor} = \boldsymbol{e}^i_{local}, \quad (4)$$

$$\boldsymbol{e}^{i,t+1}_{neighbor} = GRU(\boldsymbol{e}^{i,t}_{neighbor}, \boldsymbol{m}^{i,t}_{neighbor}), \quad \boldsymbol{e}^i_{neighbor} = \boldsymbol{e}^{i,T}_{neighbor} \quad (5)$$

Note that this module contains both action and state representation of the DOM, so the Q-value function can still be approximated with only this module.

Global Module $\boldsymbol{e}_{global}$ is the output model (Gilmer et al., 2017) of the HTML tree structure of the current web page. It provides the high level state of the web page and is used by all Q-value functions. We investigate two approaches to obtain global embedding with and without explicit goal information.

1) we use max-pooling to aggregate over all the DOM embeddings on the web page.

$$\boldsymbol{e}_{global} = maxpool\left(\left\{\left[\boldsymbol{e}^i_{local}, \boldsymbol{e}^i_{neighbor}\right] | v^i \in V\right\}\right) \tag{6}$$

2) Motivated by Vaswani et al. (2017), we use the goal information as an attention query when computing the output model of the GNN. Figure 1 shows each goal token e_{token} is concatenated with one-hot positional encoding vector e_{pos}. Position-wise feed-forward networks with ReLU activation is applied to the concatenated representation before being maxpooled to obtain the goal vector $\boldsymbol{h}_{goal}$. Note that $\boldsymbol{E}_{local}$ and $\boldsymbol{E}_{neighbor}$ are packed representations of $(\boldsymbol{e}^1_{local}, ..., \boldsymbol{e}^V_{local})$ and $(\boldsymbol{e}^1_{neighbor}, ..., \boldsymbol{e}^V_{neighbor})$ respectively, where $\boldsymbol{E}_{local} \in \mathbb{R}^{(V,d_k)}$, $\boldsymbol{E}_{neighbor} \in \mathbb{R}^{(V,d_k)}$ and d_k is the dimension of text token embedding. We will call this "goal attention"

$$\boldsymbol{e}_{attn} = \text{softmax}(\frac{\boldsymbol{h}_{goal}\boldsymbol{E}^T_{local}}{\sqrt{d_k}})\boldsymbol{E}_{neighbor}, \quad \boldsymbol{e}_{global_attn} = [\boldsymbol{e}_{global}, \boldsymbol{e}_{attn}] \tag{7}$$

Learning To obtain Q-value function Q^i_{dom} for the DOM v^i, the concatenation of the DOM embedding $\boldsymbol{e}^i = \left[\boldsymbol{e}^i_{local}, \boldsymbol{e}^i_{neighbor}, \boldsymbol{e}_{global}\right]$ of all levels are fed into a two layer MLP with weights w_{dom}, $Q^i_{dom} = MLP(\boldsymbol{e}^i; w_{dom})$. Similarly, the Q-value functions for the word token and the mode are computed using $\boldsymbol{e}^i_{neighbor}$ and $\boldsymbol{e}_{global}$ respectively, see Figure 1. All the model parameters including the embedding matrices are learned from scratch. Let $\theta = (E, w_{GNN}, w_{dom}, w_{token}, w_{mode})$ be the DOM-Q-NET model parameters including the embedding matrices, the weights of graph neural network, and weights of the factorized Q-value function. The model parameters are updated by minimizing the mean squared Bellman error:

$$\min_\theta \mathbb{E}_{(s,a,r,s')\sim\text{replay}}[\left(y^{DQN} - Q(s, a_{dom}; \theta) - Q(s, a_{token}; \theta) - Q(s, a_{mode}; \theta)\right)^2], \tag{8}$$

where the transition pairs (s, a, r, s') are sampled from the replay buffer and y^{DQN} is the factorized target Q-value with the target network parameters θ^- as in the standard DQN algorithm.

$$y^{DQN} = r + \gamma\left(\max_{a_{dom}} Q(s', a'_{dom}; \theta^-) + \max_{a_{token}} Q(s', a'_{token}; \theta^-) + \max_{a_{mode}} Q(s', a'_{mode}; \theta^-)\right) \tag{9}$$

.

4 EXPERIMENTS

In this section, we first evaluated the generalization capability of DOM-Q-NET for large action space by comparing it with previous works (Shi et al., 2017; Liu et al., 2018) that use expert demonstrations. Tasks with various difficulties are chosen from MiniWoB. In appendix, we define our criteria for difficulties of different tasks. We then investigated the gain in sample efficiency with DOM-Q-NET from multitask learning. We then performed an ablation study to justify the effectiveness of each representational module, followed by the comparison of gain in sample efficiency by goal attention in multitask and single task settings. Hyperparameters of the experiments are explained in Appendix, and we will release the code for reproducibility upon the acceptance of the paper.

4.1 DOM-Q-NET BENCHMARK MINIWOB

We used four components of Rainbow DQN (Hessel et al., 2018) to train our agent. They are DDQN (Van Hasselt et al., 2016), Prioritized replay (Schaul et al., 2016), Multi-step learning (Sutton, 1988) and NoisyNet (Fortunato et al., 2018). Selected tasks only require clicking on DOM elements and typing strings, the same setting as Liu et al. (2018). We performed $N = 3$ steps of propagation for all the tasks except "Social Media", for which we use $N = 7$ steps to address the large DOM space.

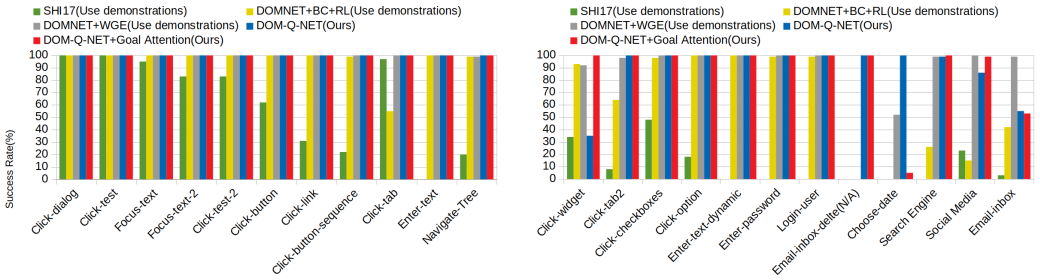


Figure 2: Comparison of DOM-Q-NET performance on MiniWoB with Shi et al. (2017); Liu et al. (2018)

Evaluation metric: we plot the moving average of the reward for last 100 episodes, and report the highest success rate for testing on 100 episodes.

Figure 2 shows that our model reached the same success rate for most of the tasks selected by Liu et al. (2018), except for "Click widget", "Social Media" and "Email inbox". We did not make any prior assumption such as providing constraints on the action set during exploration (Liu et al., 2018) or using pre-defined fields of the goal. In particular, our model solves a long-horizon task "choose-date" that none of the previous works with demonstrations are able to solve. This task contains many similar actions but has a large action space. Even with imitation learning or some guidance, the neural model needs to learn a representation that generalizes for unseen states and actions, which our model proves to be.

4.2 MULTITASK

To evaluate if our architecture can generalize over unseen or similar actions, we performed multitask learning of 9 tasks with medium difficulties. In this setting, the agent makes one action for each of M tasks at each time step and receives the next state and a reward for that task. All the transitions are stored in the same replay buffer. For each timestep, the network also updates its parameters M times. This means that the agent can make more updates for a particular task by sampling the corresponding transitions more frequently in earlier stages, and sample transitions for other tasks later. Hence, the evaluation criteria of sample efficiency for multitask versus non-multitask learning is to compare how many total number of frames the agent uses to solve all tasks versus the sum of number of the frames used for solving each task for single task agent. Multitask agent solved the 9 tasks with about twice better sample efficiency, using about 63000 frames in total whereas single task agents use about 127000 frames. Figure 3 shows the plots for 6 out of the 9 tasks. Particularly, "login-user" and "Click-Checkboxes" are solved with 40000 less frames using multitasking, but this gain is not as obvious when the task is relatively simpler, as in the case of "Navigate-Tree". This can be attributed to the capability of doing implicit curriculum learning by biased sampling of transitions for particular tasks at different training period with prioritized replay buffer. Appendix shows the full comparisons.

4.3 ABLATION STUDY ON THE DOM MODULES

Our Q learning setting is already suitable for generalizing over unseen actions, so we need to perform ablation experiments to justify the effectiveness each module. Three discounted versions that omit some modules for computing Q_{dom}, (a) $\boldsymbol{e}_{dom} = \boldsymbol{e}_{local}$, (b) $\boldsymbol{e}_{dom} = \boldsymbol{e}_{neighbor}$, (c)$\boldsymbol{e}_{dom} = [\boldsymbol{e}_{local}^T, \boldsymbol{e}_{neighbor}^T]^T$, are compared against DOM-Q-NET.

Figure 4 shows the three particular tasks chosen, and the failure case for click-checkboxes task shows that DOM selection without neighborhood module will simply not work because a lot of individual DOM will have exactly same representation without propagation steps. Liu et al. (2018) addressed this issue by hand-crafting the propagation function. The faster convergence of DOM-Q-NET to optimal behaviour indicates the limitation of neighbor module and how global and local module provide a shortcut to highest and lowest levels of representation of the web page.

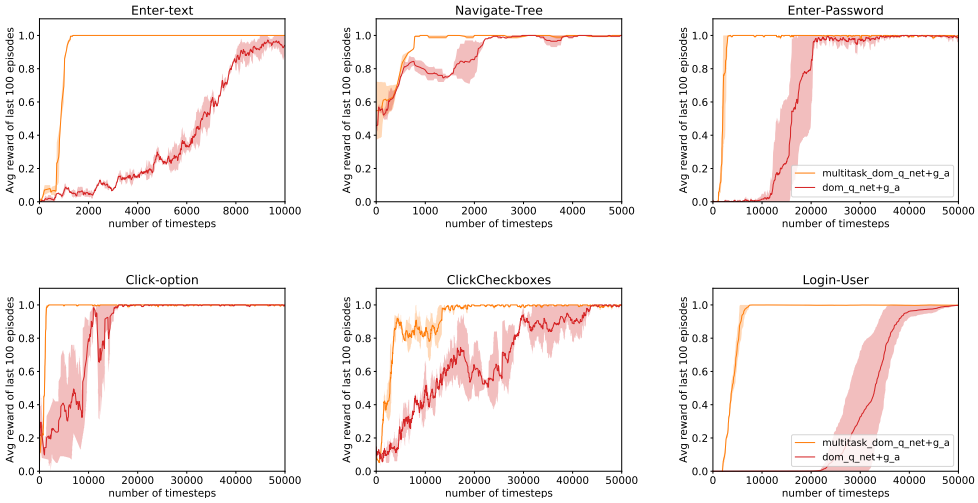


Figure 3: Multitask Comparisons: DOM-Q-NET with goal attention consistently has better sample efficiency among the shown tasks. g_a=goal attention.

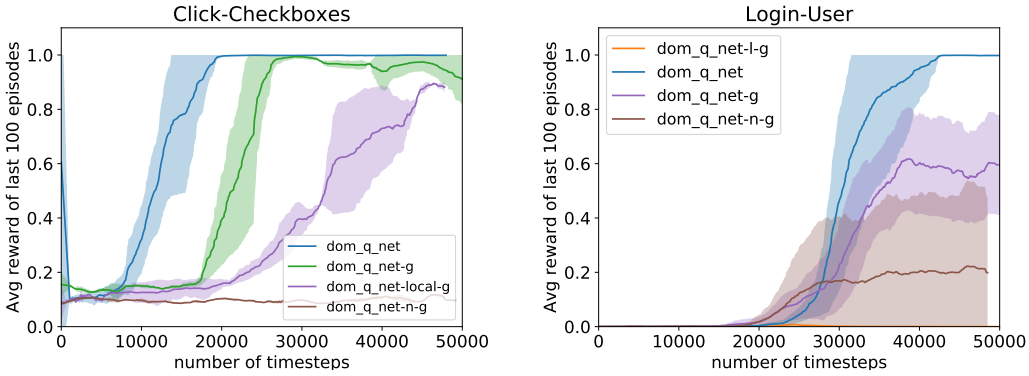


Figure 4: Ablation experiment of l=Local, n=Neighbor, g=Global modules. dom_q_net - g is the DOM-Q-NET without the global module. DOM-Q-NET -l - g is the DOM-Q-NET with only neighborhood module. dom_q_net-n-g is the DOM-Q-NET with only local module.

4.4 Effectiveness of Goal Attention

Most of the MiniWoB tasks contain only one type of expected control policy, such as "put a query word in search and find the specified link", where word token and link have alignments with the DOM elements. Hence, our DOM-Q-NET solved most of those tasks without feeding the goal vector representation to each stream, with exceptions like "click-widget". Appendix shows comparisons of DOM-Q-NET with different goal encoding methods including goal attention. The effect of goal attention is not obvious, as seen in some tasks. However, Figure 5 shows that the gain in sample efficiency from using goal attention is considerable and much bigger than the gain in single task setting.

5 Conclusion

We proposed a neural network architecture that generalizes well for large action space and multitask learning, which are two important factors to be applied in real web navigation settings. In future

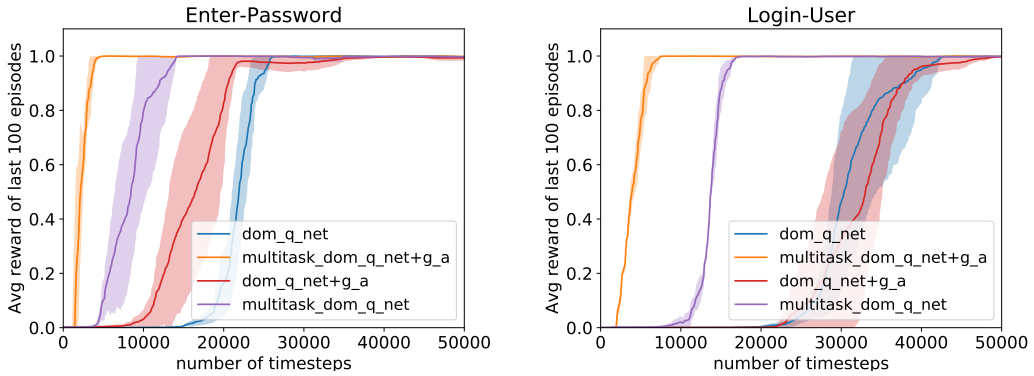


Figure 5: Effect of goal attention for single VS multi-task(g_a=goal attention)

work we will consider exploration in cases like the "email-inbox" task where the environment does not have a simpler task for the agent to learn generalizing actions. Liu et al. (2018) provided an interesting way to guide the exploration without overfitting to demonstration. We will also explore the computational cost of evaluating the Q value for each DOM element.

Finally, we intend on applying our methods to using search engines. Tasks like question answering could benefit from the ability of an agent to query search, navigate the results page and obtain relevant information for solving the desired goal. The ability to query and navigate search could also be used to bootstrap agents in realistic environments to obtain task-oriented knowledge and improve sample efficiency.

REFERENCES

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. In *ICLR*, 2018.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. 2017.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. In *JMLR*, 2016.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *ICLR*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. In *Nature*, 2015.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. In *IEEE Transactions on Neural Networks*, 2009.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR*, 2016.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *ICML*, 2017.

Richard S Sutton. Learning to predict by the methods of temporal differences. In *Machine learning*, 1988.

Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *ICLR*, 2018.

6 APPENDIX

6.1 HYPERPARAMETERS

Best DQN HyperParameters	
Optimization algorithm	Adam
Learning rate	0.00015
Batch Size	128
Discounted factor	0.99
DQN Target network update period	200 online network updates
Number of update per frame	1
Number of exploration steps	50
Easy Tasks Number of steps for training	5000
Medium Tasks Number of steps for training	50000
Hard Tasks Number of steps for training	2000000
Number of Multi-step bootstrap	8
Noisy Nets σ_0	0.5
Use DDQN	True

Best Network and Embedding HyperParameters	
Max tag vocabulary size	80
Max text vecabulary size	400
Max class Vocabulary size	80
E tag dimension	16
E text dimension	32
E class dimension	16
Fully connected layer dimension	128
Number of exploration steps	50
Number of propagation steps	3
Max number of DOMS	160
Max number goal tokens	18
Out of Vocabulary Random vector generation	Choose-option, Click-Checkboxes

Best Replay Buffer Hyperparameters	
Prioritized exponet	0.5
Single Task Buffer Size	15000
Multi Task Buffer Size	100000

6.2 GOAL ENCODER

Three type os goal encoding modules for global module

1. Goal vector concatenation
2. Goal Attention
3. Both Goal vector concatenation and attention

All the following cureves are comparing the effects of different encoding modules as well as our benchmark result for DOM-Q-NET

6.3 MINIWOB TASKS DIFFICULTIES DEFINITION

- Easy Task: Any task that can be solved under 5000 timesteps by single task DOM-Q-NET
- Medium: Any task that can be solved under 5000 0timesteps by single task DOM-Q-NET
- Hard: Any task that cannot be solved or requires 200000 timesteps to solve by single task DOM-Q-NET

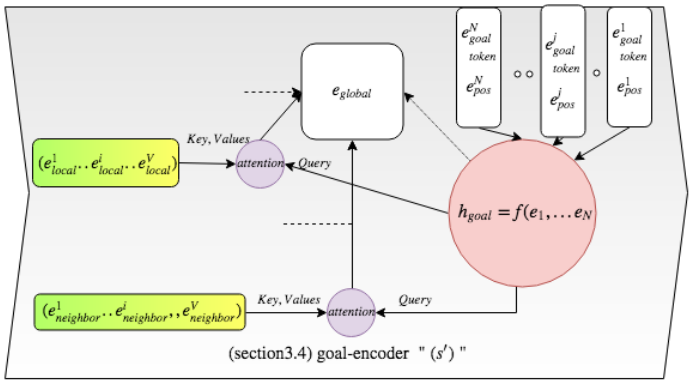
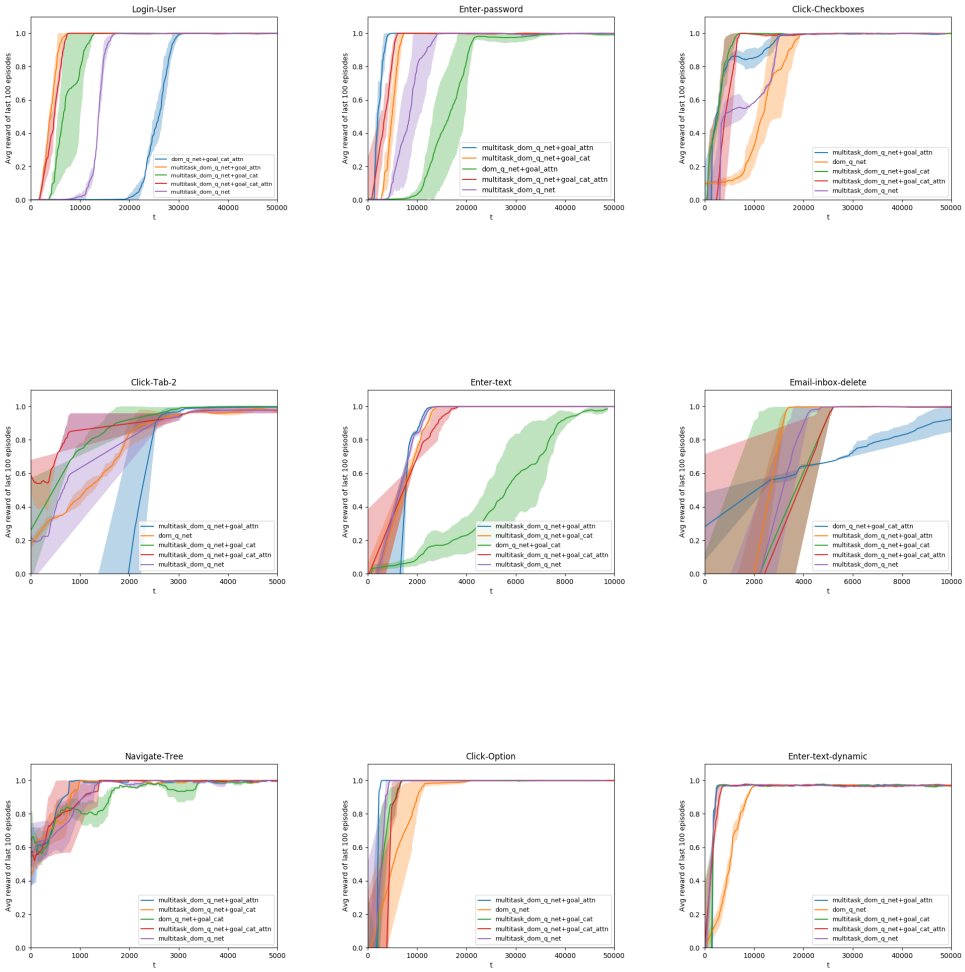


Figure 6: goal-encoder

6.4 MULTITASK (9 TASKS) RESULTS

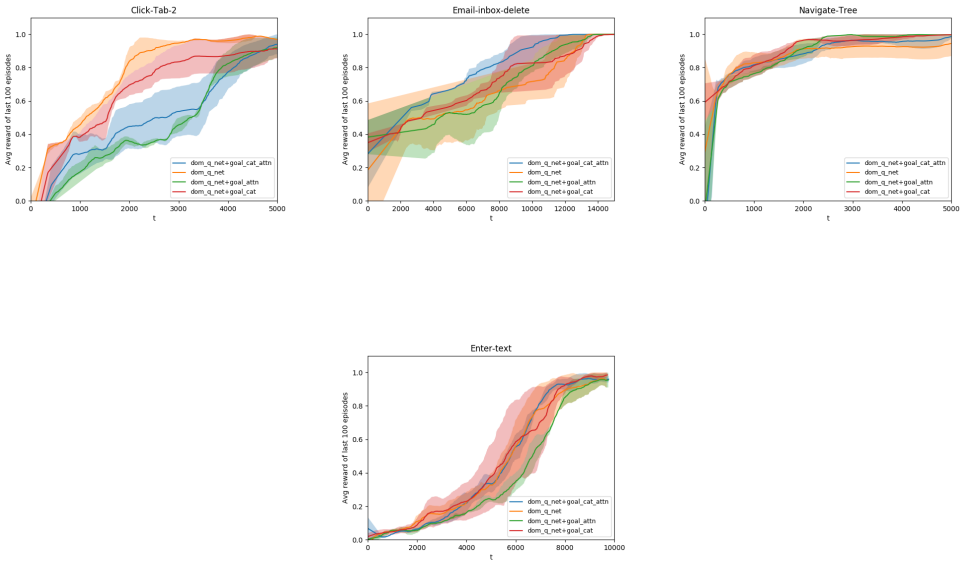
This fully shows the results for 9 selected medium difficulty Tasks for Multitasking with different goal encoder modules.



6.5 BENCHMARK RESULTS

6.5.1 EASY TASKS

Very simple tasks with less than 1000 steps are omitted.



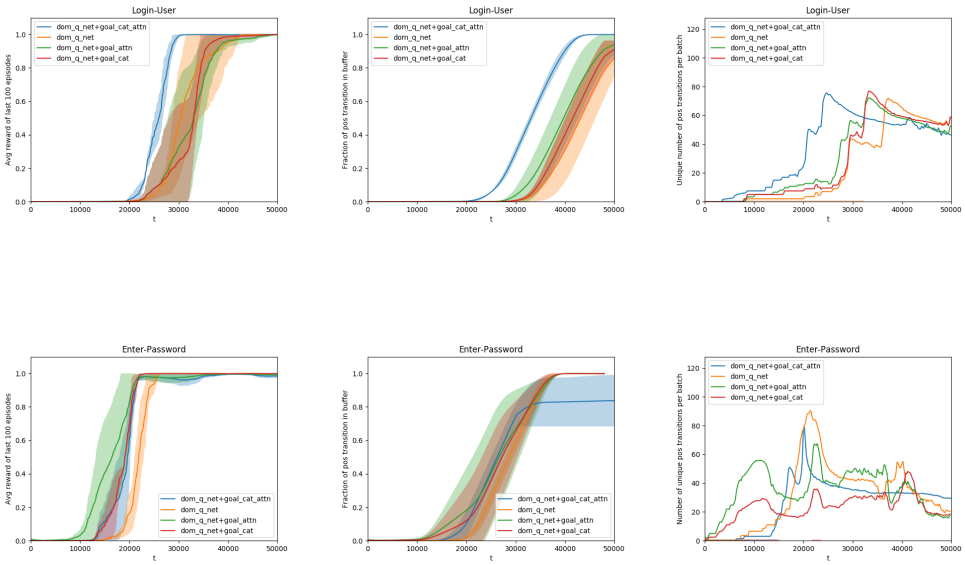
6.5.2 MEDIUM TASKS

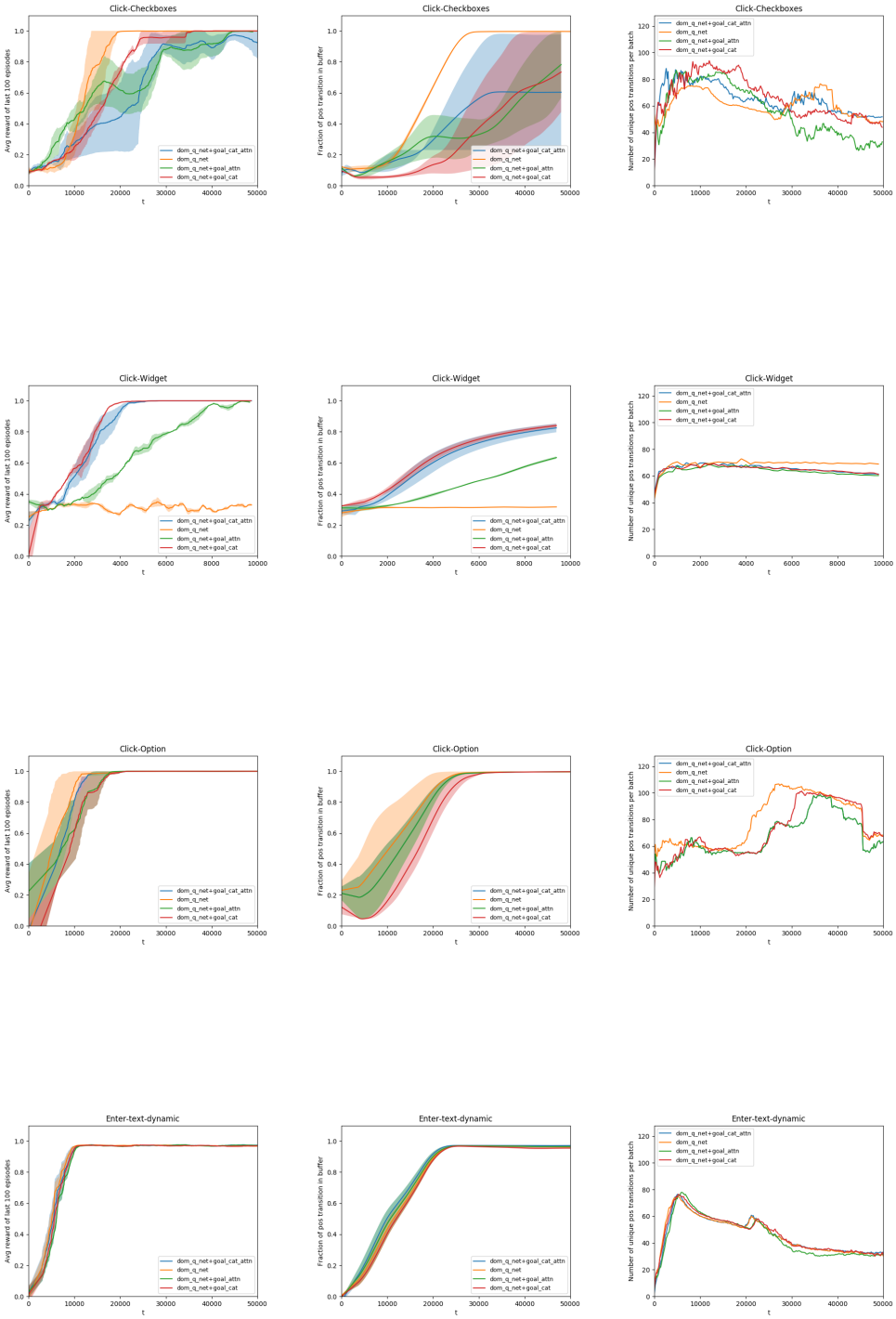
Benchmark DOM-Q-NET performance versus DOM-Q-NET + different goal encoding modules for global modules.

The plot on the left shows average moving reward of last 100 episodes.

The plot on the center shows fraction of postive transitions in replay buffer.

The plot on the center shows unique number of postive transitions sampled at each sampling.



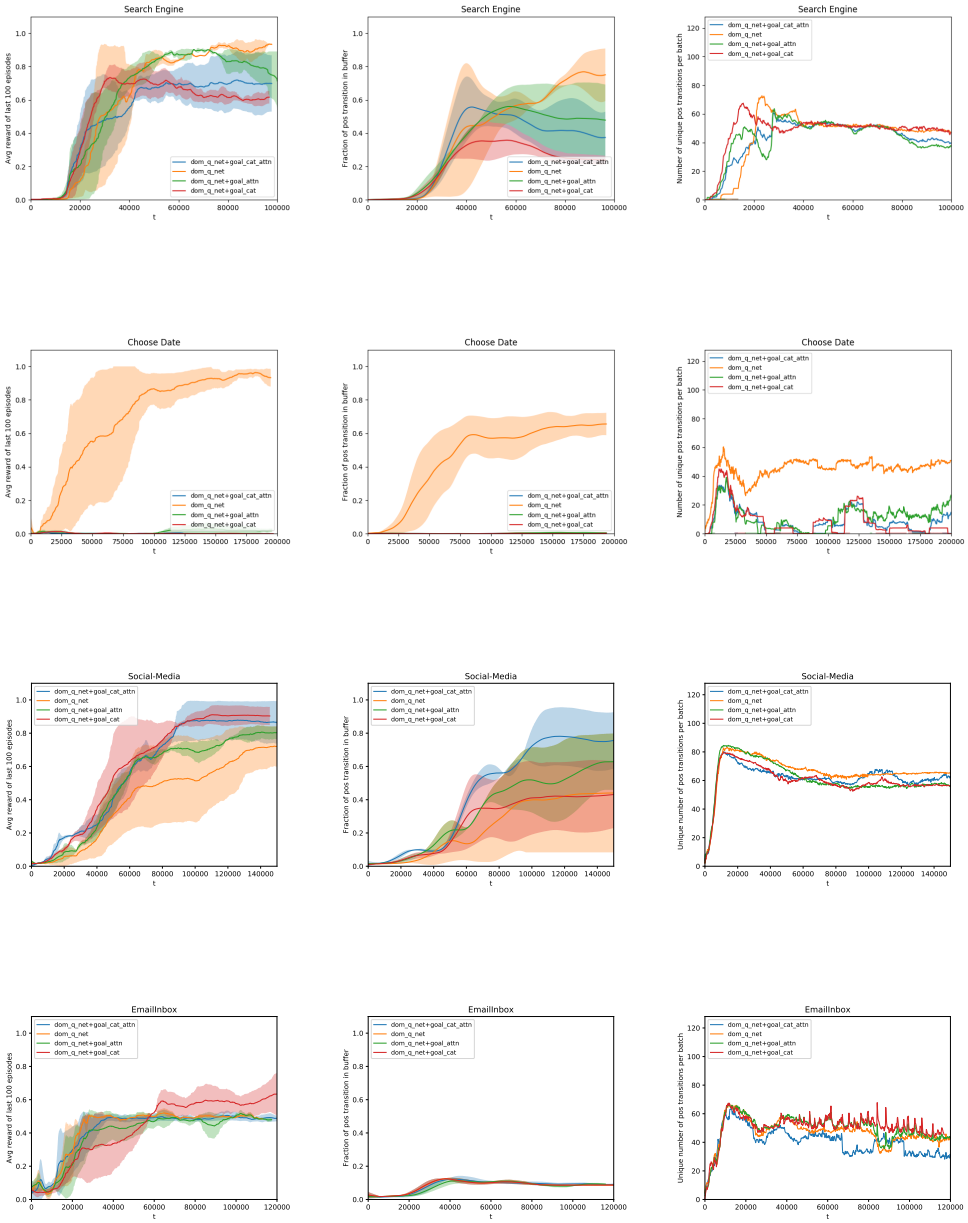


6.5.3 HARD TASKS

The plot on the left shows average moving reward of last 100 episodes.

The plot on the center shows fraction of postive transitions in replay buffer.

The plot on the center shows unique number of postive transitions sampled at each sampling.


Search Engine
Avg reward of last 100 episodes
Fraction of pos transition in buffer
Number of unique pos transitions per batch
Choose Date
Social-Media
Unique number of pos transitions per batch
EmailInbox
t
dom_q_net+goal_cat_attn
dom_q_net
dom_q_net+goal_attn
dom_q_net+goal_cat