DOM-Q-NET: GROUNDED RL ON STRUCTURED LANGUAGE

Anonymous authors
Paper under double-blind review

ABSTRACT

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 current deep reinforcement learning (RL) models due to large discrete action space and 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. It parametrizes Q functions with separate networks for different action categories, clicking DOM and typing a string input. DOM-Q-NET utilizes a graph neural network to represent the tree-structured HTML of a standard web page. We demonstrate the capabilities of our model on the WorldOfBits(MiniWoB) environment where we can match or outperform existing work without the use of expert demonstrations. Furthermore, we show 2x improvements in sample efficiency when training in the multi-task setting, allowing our model to transfer learned behaviours across tasks.

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 allow RL agents to learn the control policies from raw pixels without feature engineering by human experts. However, most of the deep RL methods focus on solving problems in either simulated physics environments where the inputs to the agents are joint angels and velocities, or simulated video games where the inputs are rendered graphics. Agents trained in such simulated environments have little knowledge about the rich semantics of the 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. Each action of navigation corresponds to interacting with DOMs from tree structured HTML. Tasks like finding a friend on a social network, clicking into a web page, and rating a place on Google maps can be framed as accessing the DOM and potentially modifying its value with user input.

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 state of the art performance with very few expert demonstrations, but their exploration policy requires constrained action sets, hand-crafted with expert knowledge in HTML.

In this work, our contribution is to propose a novel architecture, DOM-Q-NET, that parametrizes factorized Q functions for web navigation, which can be trained to match or outperform existing work on MiniWoB without using any expert demonstration. Graph neural network (Scarselli et al., 2009; Li et al., 2016) is used as the main backbone to provide three levels of state and action representations, which are selectively fed into three factorized Q networks.

Essentially, it uses neural message passing and readout of the node representations (Gilmer et al., 2017) from 'local' representations of the web pages to produce 'neighbor' and 'global' representations. Three representational modules are used by factorized Q networks that use three separate multilayer perceptrons(MLP) (Rumelhart et al., 1985) for parametrizing the factorized Q values of distinct action categories, "click", "type" and "mode". The entire architecture is fully differentiable, and all of its components are jointly trained.

Another contribution is that it is the first architecture to perform multitask learning of web navigation tasks, and demonstrate the transferability of learned behaviors on the web interface. We show that the multi-task agent achieves an average of 2x sample efficiency comparing to the single task agent.

2 BACKGROUND

2.1 REPRESENTING WEB PAGES USING DOMS

The Document Object Model (DOM) is a programming interface for HTML documents and it defines logical structure of such documents. DOMs are connected in a tree structure, and we frame web navigation as accessing a 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 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 $m_v^{t+1} = \sum_{\omega \in N(v)} M(h_v^t, h_w^t, e_{vw})$. 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 REINFORCEMENT LEARNING WITH GRAPH NEURAL NETWORKS

There have been works in robot locomotion that use graph neural networks (GNNs) to model the physical body (Wang et al., 2018; Hamrick et al., 2018). NerveNet demonstrates that policies learned with GNN transfers better to other learning tasks than policies learned with MLP (Wang et al., 2018). It uses GNNs to parametrize the entire policy whereas DOM-Q-NET uses GNNs to provide representational modules for factorized Q functions. Note that the graph structure of a robot is static whereas the graph structure of a web page can change at each time step. Locomotion based control tasks provide dense rewards whereas web navigation tasks are sparse reward problems with only 0/1 reward at the end of the episode. For web navigation, our model also needs to account for the dependency of actions on goal instructions.

2.5 PREVIOUS WORKS ON RL ON WEB INTERFACES

Shi et al. (2017) constructed benchmark tasks, Mini World of Bits (MiniWoB), that consist of many toy tasks of web navigation . This environment provides both the image and HTML of a web page. Their work showed benchmark results 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 attention between DOM elements and the goal, and workflow guided-exploration, which uses predefined formal language to constrain the action space of an agent at each state. They achieved new state of the art performance and sample efficiency of using demonstrations with the constrained action sets. Unlike these previous works, we aim to tackle web navigation tasks without any expert demonstration or prior knowledge.

3 NEURAL DOM Q NETWORK

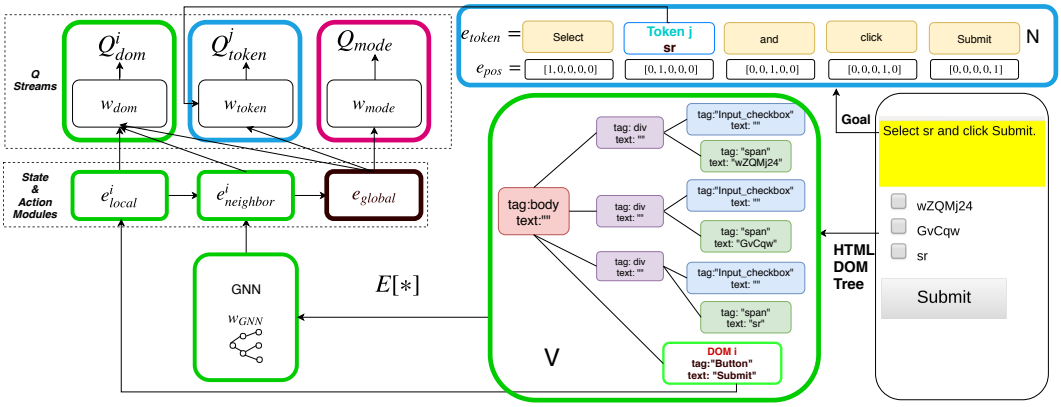


Figure 1: Given the web page on the right, its DOM tree representation is shown as a graph where each DOM represents a node from V. Different color indicates different tag attributes of DOMs. DOMs are set as local module $\boldsymbol{e}_{local}$, and also propagated by a GNN to produce neighbor module $\boldsymbol{e}_{neighbor}$. Global module $\boldsymbol{e}_{global}$ is aggregated from neighbor module. Then Q_{dom} network 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. Both Q_{token} and Q_{mode} streams only use the global module.

Consider the problem of navigating 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 that are presented to the agent in the form of a natural language sentence, e.g. "Select sr and click Submit" shown in figure 1 and "Use the textbox to enter Kanesha and press Search, then find and click the 9th search result" in figure 2. 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; and 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 require 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 checkboxes as well as choosing which text box to fill in the string input.

- 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 $a = (a_{dom}, a_{token}, a_{mode})$ though it will only use a_{token} when $a_{mode} = type$.

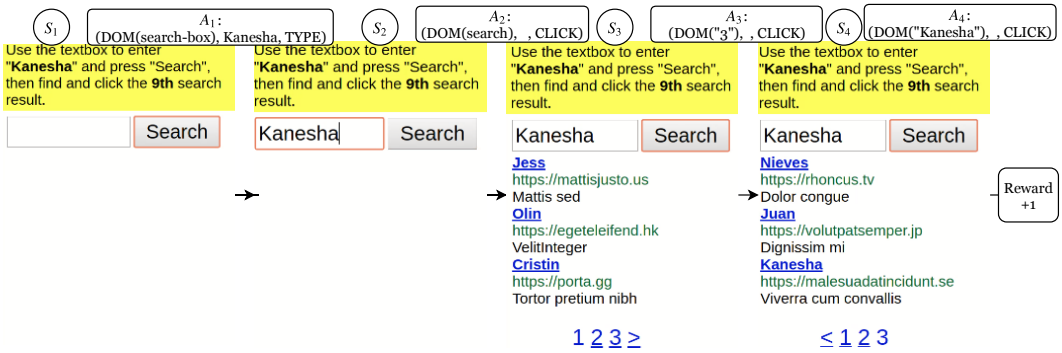


Figure 2: Example of a successful trajectory executed by DOM-Q-NET for Search-Engine task. S_i is the state, and $A_i = (a_{dom}, a_{token}, a_{mode})$ is a tuple of actions for the three distinct categories of actions at timestep i. DOM(x) represents the index of the corresponding element x in the web page.

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. For MiniWoB, this introduces an enormous action space with size $|V| \times |\mathcal{G}|$. The number of DOMs and goal tokens, $|V|$ and $|\mathcal{G}|$, can reach up to 60 and 18, and the total number of actions become over $1,000$ for some hard tasks.

To reduce the action space, we consider a factorized state-action value function where the action values 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 of its factorized Q networks. Explicitly modeling the HTML tree structure provides the 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 corresponding to node-level, and graph-level outputs of the GNN, which 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 use 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.

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

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 network (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 message passing. We apply T number of message passing steps to obtain the final neighbor embedding. We adopt GRU gated version (Li et al., 2016) for state updates between aggregation time steps.

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

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

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 feature vector of the entire HTML of the current web page after readout phase (Gilmer et al., 2017). It provides the high level state of the web page and is used by all factorized Q networks. We investigate two readout functions to obtain such global embedding with and without explicit goal information.

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

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

2) We use attention for the readout phase of the GNN where the attention query is the goal vector. This is in contrast to Velickovic et al. (2018) where the attention is used in message passing phase and the query is not a goal dependent description. 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 each concatenated representation before being maxpooled to obtain the goal vector $\boldsymbol{h}_{goal}$. Motivated by Vaswani et al. (2017), we use scaled dot product attention with local embeddings $\boldsymbol{e}^i_{local}$ as keys, and neighbor embeddings $\boldsymbol{e}^i_{neighbor}$ as values. 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", and show its simplified diagram in the appendix 6.2.

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

Note that the simplest method is to concatenate each node-level feature with goal vector though this will increase the size of the network, and still leads to the same performance. Further details and comparisons are shown in the appendix 6.3, 6.6.

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 w_{token} and w_{mode} respectively, with a shared $\boldsymbol{e}_{global}$ state module. Here, each $\boldsymbol{e}_{token}$ is fed into a MLP with w_{token} for computing the action value of its word token, 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 a graph neural network, and weights of the factorized Q-value function. The model parameters are updated by minimizing the mean squared TD error(Sutton, 1988):

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

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 evaluate 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, as defined in the appendix 6.4, are chosen from MiniWoB. Then we investigate the gain in sample efficiency with our model from multitask learning. We also perform an ablation study to justify the effectiveness of each representational module, followed by the comparison of gains in sample efficiency from goal attention in multitask and single task settings. Hyperparameters of the experiments are explained in the appendix 6.1, and we will release the code for reproducibility upon acceptance of the paper.

4.1 DOM-Q-NET BENCHMARK MINIWOB

We chooses an off-policy Q learning algorithm, using the four components of Rainbow DQN (Hessel et al., 2018), to train our agent because the web navigation tasks are sparse reward problems, and off-policy learning with a replay buffer is more efficient. Four components 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 require clicking DOM elements and typing strings, which is the same setting as Liu et al. (2018). The agent receives +1 reward if the task is completed correctly, and 0 reward otherwise. We perform $N = 3$ steps of neural message passing of GNN for all the tasks except "Social Media", for which we use $N = 7$ steps to address the large DOM space.

Evaluation metric: We plot the moving average of the rewards for last 100 episodes during the training. We follow the previous works (Shi et al., 2017; Liu et al., 2018), and report success rate, which is the percentage of test episodes that ends up with reward +1. Each reported success rate is based on the average of 4 different runs, and appendix 6.5 explains the experiment protocol.

Results: Figure 3 shows that DOM-Q-NET reaches 100% success rate for most of the tasks selected by Liu et al. (2018), except for "Click widget", "Social Media" and "Email inbox". Our model still reaches 86% success rate for "Social Media" task, and the use of goal attention enables the model to solve "Click widget" and "Social Media" with 100% success rate. We did not use any prior knowledge such as providing constraints on the action set during exploration (Liu et al., 2018) and using pre-defined fields of the goal or expert demonstrations. In particular, our model solves a long-horizon task "choose-date" that none of the previous works with demonstrations is able to solve. This task contains many similar actions but has a large action space. Even with imitation learning or guided exploration, the neural network needs to learn a representation that generalizes for unseen diverse set of DOM states and actions, which our model proves to do.

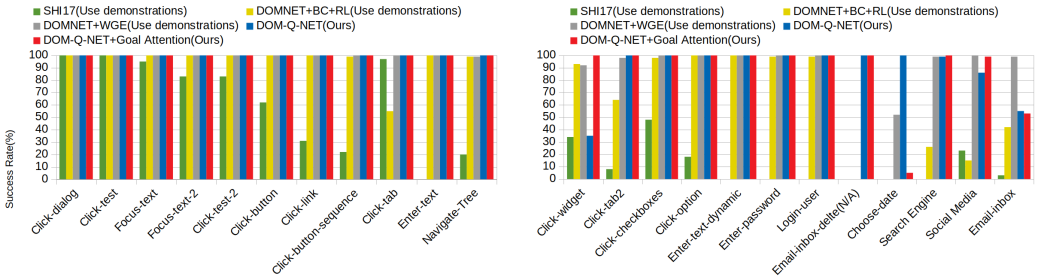


Figure 3: Comparison of DOM-Q-NET performance on MiniWoB with Shi et al. (2017); Liu et al. (2018)

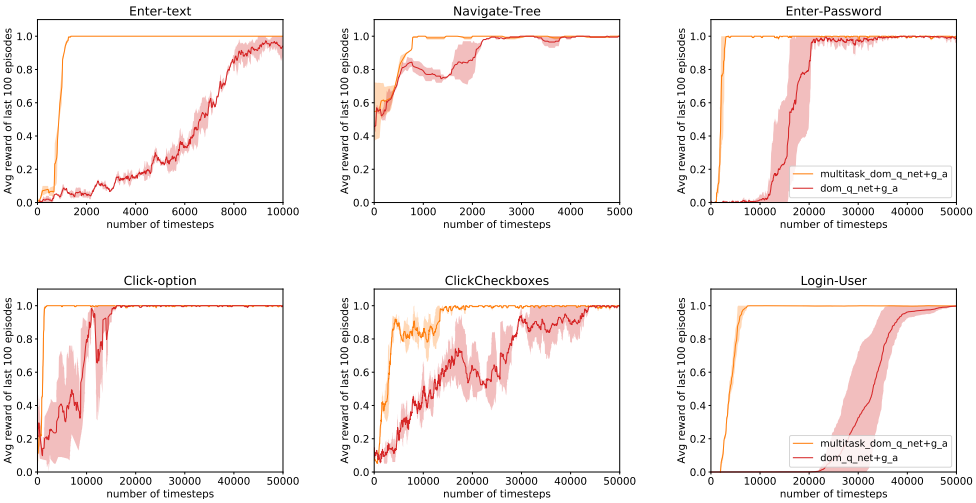


Figure 4: Multitask Comparisons: Multitask DOM-Q-NET with goal attention consistently has better sample efficiency among the shown tasks. g_a=goal attention.

4.2 MULTITASK

To evaluate the transferability of learned behaviours, we perform multitask learning of 9 tasks. In this setting, the agent makes one action for each of the 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. At 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 the total number of frames used for solving all the tasks against the sum of the number of frames used for solving each task for a single task agent.

Multitask agent solves the 9 tasks with about twice better sample efficiency, using approximately 63000 frames in total, whereas the single task agents use approximately a total of 127000 frames. Figure 4 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 6.6.1 shows the full comparisons.

Multitask learning is not applicable to the previous work using workflow-guided exploration(WGE) with DOMNET (Liu et al., 2018), because WGE needs structured key-value inputs as a result of

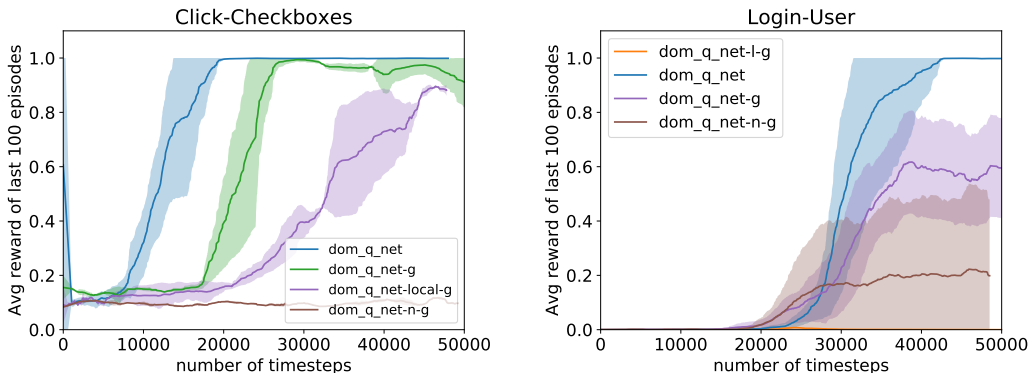


Figure 5: Ablation experiment of l=Local, n=Neighbor, g=Global modules. dom_q_net - g is the DOM-Q-NET without the global module. DOM-Q-NET -l - g is the DOM-Q-NET with only neighborhood module. dom_q_net-n-g is the DOM-Q-NET with only local module.

using the formal language for constraining actions. Since the embeddings of DOMNET for key-value inputs are fed into the network without being aggregated, the dimension of the weight matrices is task-dependent. As such, single task DOMNET with WGE cannot be extended to multitask learning.

4.3 Ablation Study on the DOM Representation Modules

We perform ablation experiments to justify the effectiveness of each module for Q_{dom} network. 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 5 shows the two tasks chosen, and the failure case for click-checkboxes task shows that DOM selection without neighborhood module will simply not work because many individual DOMs will have exactly the same representations without message passing phase. Liu et al. (2018) addressed this issue by hand-crafting the aggregation function. The faster convergence of DOM-Q-NET to optimal behaviour indicates the limitation of neighbor module and how global and local module provide shortcuts to highest and lowest levels of the representation for the DOM.

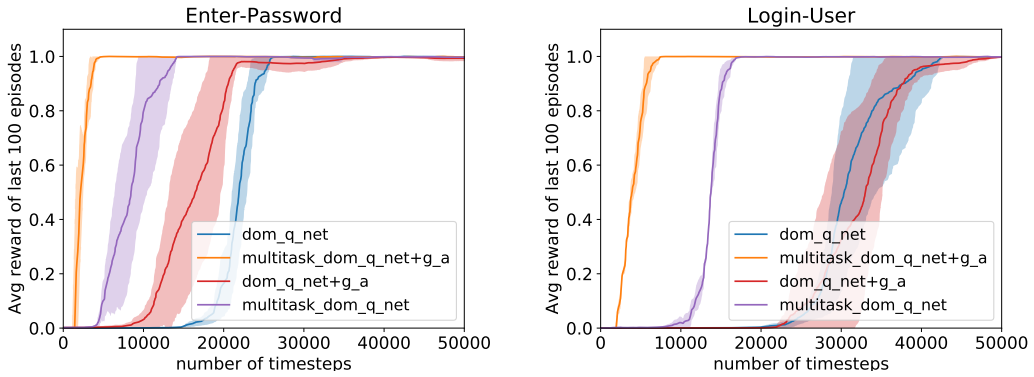


Figure 6: Effect of goal attention for single VS multi-task(g_a=goal attention)

4.4 Effectiveness of Goal Attention

Most of the MiniWoB tasks have only one type of expected control policy, such as "put a query word in the search and find the matched link", where the word token for query and link have alignments

with the DOM elements. Hence, our DOM-Q-NET solves most of the tasks without feeding the goal vector representation to each Q network, with exceptions like "click-widget". Appendix 6.6 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 6 shows that the gain in sample efficiency from using goal attention is considerable and much bigger in multitask than the gain in single task setting. This means that the agent successfully learns to pay attention to different parts of the DOM tree given different goal instructions when multitasking.

5 DISCUSSION

We propose a new architecture for parameterizing factorized Q functions using goal attention, local word embeddings and a graph neural network(GNN), and contributes to the formulation of web navigation with this model. Without any demonstration, it solves hard tasks with large action space, and transfers learned behaviours when multitasking, which are two important factors for web navigation. For future work, we investigate exploration strategies for tasks like "email-inbox" where the environment does not have a simple version of the task that the agent can use to generalize learned behaviours. Liu et al. (2018) demonstrated an interesting way to guide the exploration. Another problem is to reduce the computational cost of evaluating the Q values 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.

Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018.

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.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

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.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

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

Following hyperparameters are used throught all the experiments presented in this paper.

Table 1: Hyperparameters for training with Rainbow DQN (4 components)

Hyperparameter	Value
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
N steps (multi-step) bootstrap	8
Noisy Nets σ_0	0.5
Use DDQN	True
Easy Tasks: Number of steps for training	5000
Medium Tasks: Number of steps for training	50000
Hard Tasks: Number of steps for training	2000000

Table 2: Hyperparameters for DOM-Q-NET

Hyperparameter	Value
Vocabulary size: tag	80
Vocabulary size: text	400
Vocabulary size: class	80
Embedding dimension: tag	16
Embedding dimension: text	32
Embedding dimension: class	16
Dimension of Fully Connected(FC) layers	128
Number of FC layers for 3 factorized Q networks	2 each
Hidden Layer Activation	ReLU
Number of steps for neural message passing	3 (7 for social media task)
Max number of DOMS	160
Max number goal tokens	18
Out of Vocabulary Random vector generation	Choose-option, Click-Checkboxes

Table 3: Hyperparameters for Replay Buffer

Hyperparameter	Value
α: prioritization exponent	0.5
β for computing importance sampling weights	0
Single Task Buffer Size	15000
Multi Task Buffer Size	100000

6.2 GOAL ATTENTION OUTPUT MODEL

Figure 7 shows readout phase of the graph neural network using goal-attention. The graph-level feature vector, h_{global}, is computed by the weighted average of node-level representations processed with T steps of message passing, $\{h_1.....h_V\}$. The weights, $\{\alpha_1.....\alpha_V\}$, are computed with goal vector as query and node-level features without message passing as keys. For DOM-Q-NET, we use a scaled dot product attention (Vaswani et al., 2017) with local embeddings as keys and neighbor embeddings as values, as illustrated in 3.3.

We frame our goal-attention on the line of works for GNNs(Scarselli et al., 2009), and enable GNNs to be used as a parametrized state for goal-oriented RL.

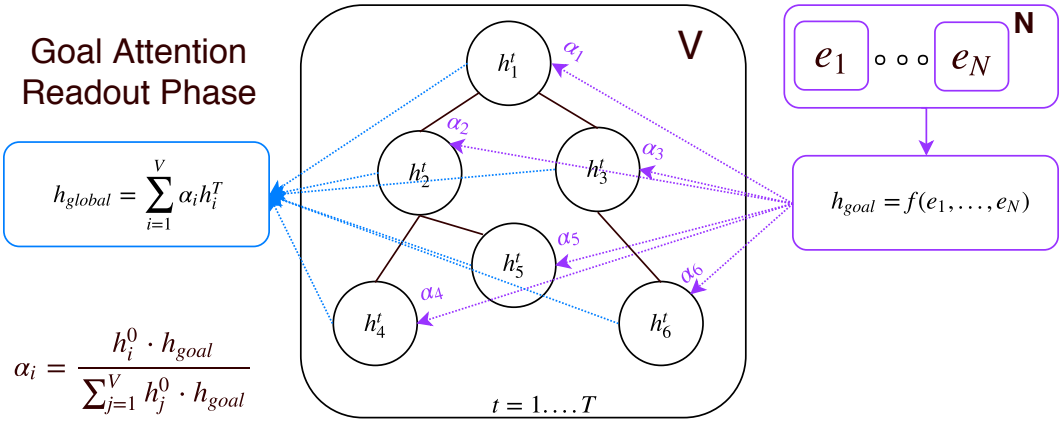


Figure 7: goal-attention

6.3 Goal Encoder

Three types of goal encoding module for global module are investigated.

1. Goal vector concatenation with node-level features
2. Goal Attention, as illustrated in 6.2
3. Both Goal vector concatenation and attention, as shown in figure8

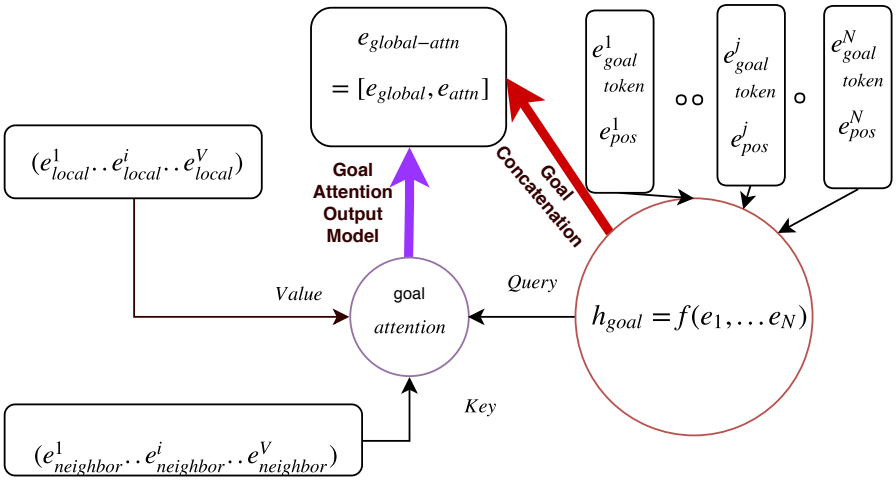


Figure 8: goal-encoder

Benchmark results for multitask and 23 tasks in the appendix 6.6 also compare the performances of using different goal encoding modules.

6.4 MiniWoB Tasks Difficulties Definition

- **Easy Task**: Any task solvable under 5000 timesteps by single task DOM-Q-NET {click-dialog, click-test, focus-text, focus-text-2, click-test-2, click-button, click-link, click-button-sequence, click-tab, click-tab-2, Navigate-tree}

- **Medium Task**: Any task solvable under 50000 timesteps by single task DOM-Q-NET {enter-text, click-widget, click-option, click-checkboxes, enter-text-dynamic, enter-password, login-user, email-inbox-delete}

- **Hard Task**: Any task solvable under 200000 timesteps by single task DOM-Q-NET, or any task for which the agent does not reach 100% success rate. {choose-date, search-engine, social-media, email-inbox}

6.5 EXPERIMENT PROTOCOL

We report the success rate of the 100 test episodes at the end of the training once the agent has converged to its highest performance. The final success rate reported in Figure 3 is based on the average of success rate from 4 different random seeds/runs. In detail, we evaluate the RL agent after training for a fixed number of frames depending on the difficulty of the task, as illustrated in the appendix 6.4. As shown in table 4, the results presented in this paper is based on a total of 536 experiments for the set of hyperparameters in table 1, 2, 3.

Table 4: Experiment statistics

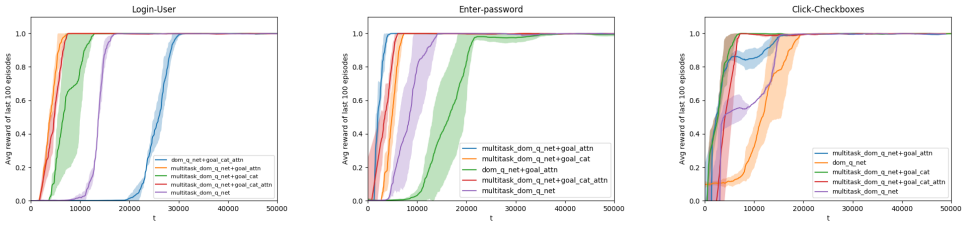
Number of tasks	23
Number of tasks concurrently running for multitask	9
Number of goal encoding modules compared	4
$N_1 = (23 + 9) * 4 = 128$	
Number of tasks for ablation study	2
Number of discounted models compared for ablation study	3
$N_2 = 2 * 3 = 6$	
Number of experiments for computing the average of a result	4
$N_{total} = (128 + 6) * 4 = 536$	

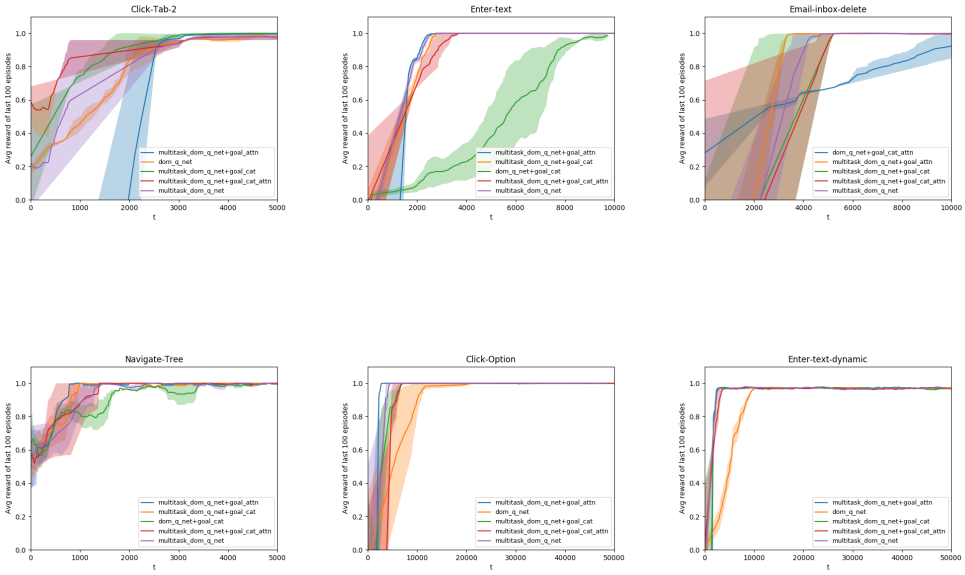
6.6 BENCHMARK RESULTS

We present the learning curves of both single and multitask agents that provided the results reported in this paper. Each plot is accompanied with the learning curves of a DOM-Q-NET agent with different goal encoding modules 6.3. X-axis and Y-axis represent the timestep and moving average of last 100 rewards respectively. For medium and hard tasks, we also show the fraction of transitions with positive/non-zero rewards in the replay buffer, and number of unique positive transitions sampled throughout the training. This is to demonstrate the sparsity of the reward for each task, and investigate whether the actual problem is due to the lack of exploration.
(Note that we are using multistep-bootstrap (Sutton, 1988) so some transitions that do not directly lead to the rewards are still analyzed as positive here)

6.6.1 MULTITASK (9 TASKS) RESULTS

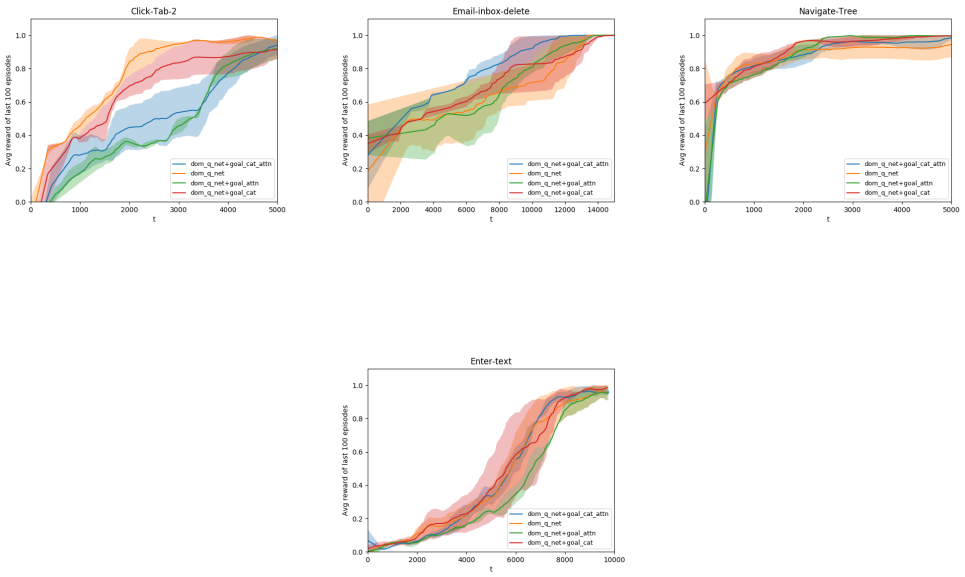
The following shows the results for 9 tasks used in Multitasking with different goal encoder modules.





6.6.2 Some Easy and Medium Tasks

The plots for very simple tasks with less than 1000 steps are omitted.



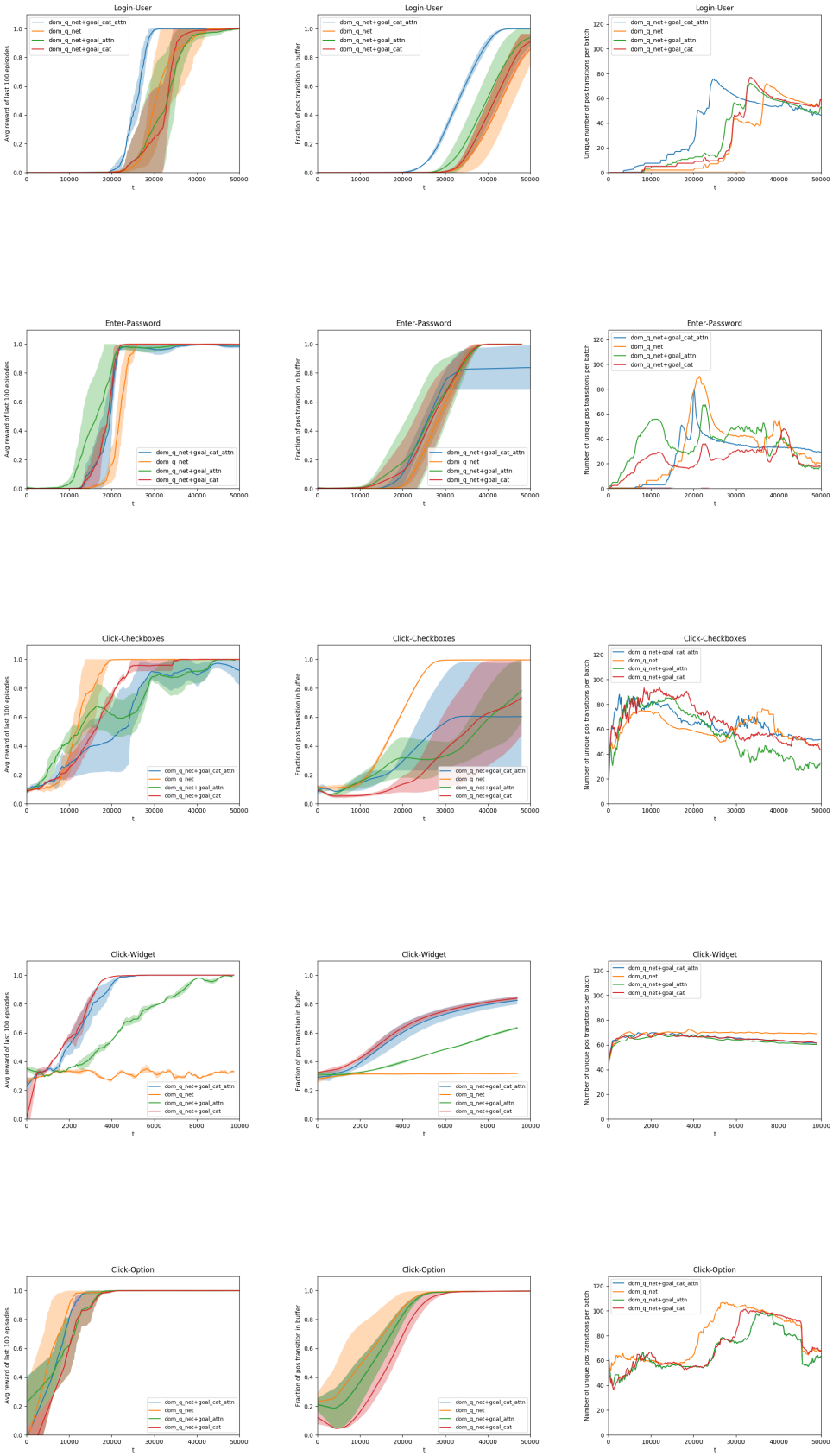
6.6.3 Medium Tasks with replay buffer information

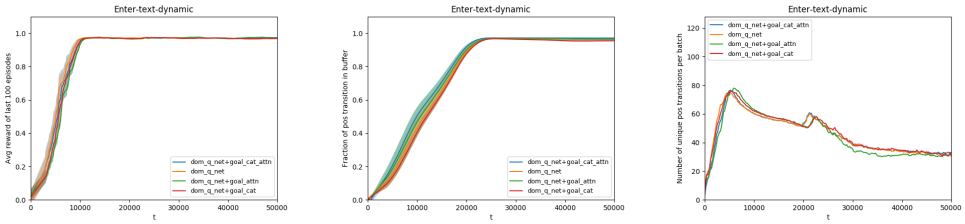
Benchmark DOM-Q-NET performance versus DOM-Q-NET + different goal encoding modules for global modules.

The plots on the left show average moving reward of last 100 episodes.

The plots on the center show fraction of postive transitions in replay buffer.

The plots on the right show unique number of postive transitions sampled at each sampling.


Login-User
Enter-Password
Click-Checkboxes
Click-Widget
Click-Option
Avg reward of last 100 episodes
Fraction of pos transition in buffer
Unique number of pos transitions per batch
Number of unique pos transitions per batch
t
dom_q_net+goal_cat_attn
dom_q_net
dom_q_net+goal_attn
dom_q_net+goal_cat

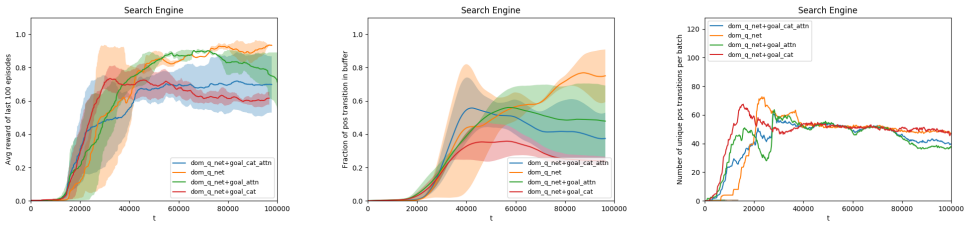


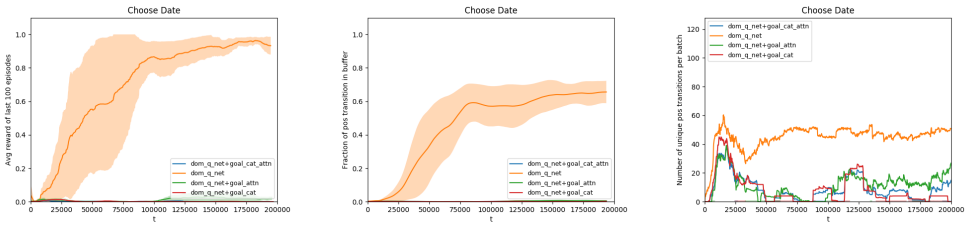
6.6.4 HARD TASKS

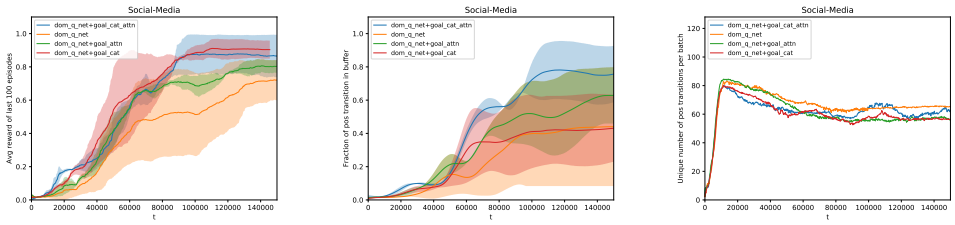
The plots on the left show average moving reward of last 100 episodes.

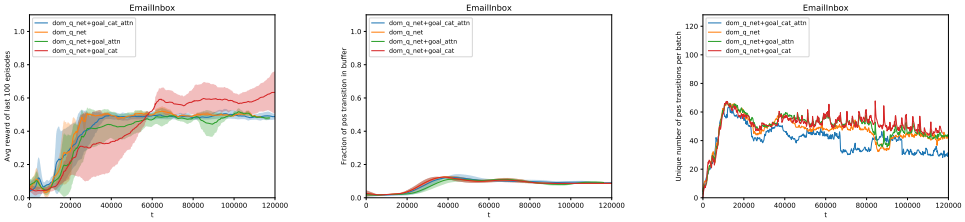
The plots on the center show fraction of postive transitions in replay buffer.

The plots on the right show unique number of postive transitions sampled at each sampling.








EmailInbox
dom_q_net+goal_cat_attn
dom_q_net
dom_q_net+goal_attn
dom_q_net+goal_cat
Avg reward of last 100 episodes
t
EmailInbox
dom_q_net+goal_cat_attn
dom_q_net
dom_q_net+goal_attn
dom_q_net+goal_cat
Fraction of pos transition in buffer
t
EmailInbox
dom_q_net+goal_cat_attn
dom_q_net
dom_q_net+goal_attn
dom_q_net+goal_cat
Unique number of pos transitions per batch
t