Interactive Machine Comprehension with Information Seeking Agents

Xingdi Yuan[†*] **Jie Fu**[‡♠*] **Marc-Alexandre Côté**[†] **Yi Tay**[♢]
Christopher Pal[‡♠♡] **Adam Trischler**[†]
[†]Microsoft Research, Montréal [‡]Polytechnique Montréal
[♠] Mila [♢] Nanyang Technological University [♡] Canada CIFAR AI Chair
eric.yuan@microsoft.com jie.fu@polymtl.ca

Abstract

Existing machine reading comprehension (MRC) models do not scale effectively to real-world applications like web-level information retrieval and question answering (QA). We argue that this stems from the nature of MRC datasets: most of these are *static* environments wherein the supporting documents and all necessary information are fully observed. In this paper, we propose a simple method that reframes existing MRC datasets as *interactive*, partially observable environments. Specifically, we "occlude" the majority of a document's text and add context-sensitive commands that reveal "glimpses" of the hidden text to a model. We repurpose SQuAD and NewsQA as an initial case study, and then show how the interactive corpora can be used to train a model that seeks relevant information through sequential decision making. We believe that this setting can contribute in scaling models to web-level QA scenarios.[1]

1 Introduction

Many machine reading comprehension (MRC) datasets have been released in recent years (Rajpurkar et al., 2016; Trischler et al., 2016; Nguyen et al., 2016; Reddy et al., 2018; Yang et al., 2018) to benchmark a system's ability to understand and reason over natural language. Typically, these datasets require an MRC model to read through a document to answer a question about information contained therein.

The supporting document is, more often than not, static and fully observable. This raises concerns, since models may find answers simply through shallow pattern matching; e.g., syntactic similarity between the words in questions and documents. As pointed out by Sugawara et al. (2018), for questions starting with *when*, models tend to predict the **only** date/time answer in the supporting document. Such behavior limits the generality and usefulness of MRC models, and suggests that they do not learn a proper 'understanding' of the intended task. In this paper, to address this problem, we shift the focus of MRC data away from 'spoon-feeding' models with sufficient information in fully observable, static documents. Instead, we propose *interactive* versions of existing MRC tasks, whereby the information needed to answer a question must be gathered sequentially.

Question: What was the Harvard endowment total in 2011 ?	
o_1	Harvard has the largest university endowment in the world .
`next`	a_1
o_2	At the end of June 2009, it was worth \$25.7 billion, about 30% less than at the same time in 2008.
`Ctrl+F Harvard`	a_2
o_3	In December 2008, Harvard announced that its endowment had lost 22% from July to October 2008, necessitating budget cuts.
`Ctrl+F 2011`	a_3
o_4	As of September 2011 , it had nearly regained the loss suffered during the 2008 recession .
`Ctrl+F 2011`	a_4
o_5	It was worth **\$ 32 billion** in 2011 , up from \$ 28 billion in September 2010 and \$ 26 billion in 2009 .
`stop`	a_5
Prediction: `$ 32 billion`	

Table 1: Example of the interactive machine reading comprehension behavior.

The key idea behind our proposed interactive MRC (iMRC) is to restrict the document context that a model observes at one time. Concretely, we split a supporting document into its component sentences and withhold these sentences from the model. Given a question, the model must issue commands to observe sentences in the withheld set; we equip models with actions such as `Ctrl+F`

[*] Equal contribution.
[1]The dataset and implementation of our baseline agents are publicly available at https://github.com/xingdi-eric-yuan/imrc_public.

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 2325–2338
July 5 - 10, 2020.

to search for matches to a QUERY within *partially* observed documents. A model searches iteratively, conditioning each command on the input question and the sentences it has observed previously. Thus, our task requires models to 'feed themselves' rather than spoon-feeding them with information. This casts MRC as a sequential decision-making problem amenable to reinforcement learning (RL).

Our proposed approach lies outside of traditional QA work, the idea can be applied to almost all existing MRC datasets and models to study interactive information-seeking behavior. As a case study in this paper, we re-purpose two well known, related corpora with different difficulty levels for our iMRC task: SQuAD and NewsQA. Table 1 shows an example of a model performing interactive MRC on these datasets. Naturally, our reframing makes the MRC problem harder; however, we believe the added demands of iMRC more closely match web-level QA and may lead to deeper comprehension of documents' content.

The main contributions of this work are as follows:

1. We describe a method to make MRC datasets interactive and formulate the new task as an RL problem.

2. We develop a baseline agent that combines a top performing MRC model and two state-of-the-art RL optimization algorithms and test it on iMRC tasks.

3. We conduct experiments on several variants of iMRC and discuss the significant challenges posed by our setting.

2 Related Works

Skip-reading (Yu et al., 2017; Seo et al., 2017; Choi et al., 2017) is an existing setting in which MRC models read partial documents. Concretely, these methods assume that not all tokens in the input sequence are equally useful, and therefore learn to skip irrelevant tokens. Since skipping decisions are discrete, the models are often optimized by the REINFORCE algorithm (Williams, 1992). For example, the structural-jump-LSTM (Hansen et al., 2019) learns to skip and jump over chunks of text, whereas Han et al. (2019) designed a QA task where the model reads streaming data without knowing when the question will be provided. Skip-reading approaches are limited in that they only consider jumping *forward* over a few consecutive tokens. Based on the assumption that a single pass of reading may not provide sufficient information, multi-pass reading methods have also been studied (Sha et al., 2017; Shen et al., 2017).

Compared to skip-reading and multi-pass reading, our work enables an agent to jump through a document in a more dynamic manner, in some sense combining aspects of skip-reading and re-reading. Specifically, an agent can choose to read *forward*, *backward*, or to jump to an *arbitrary position* depending on the query. This also distinguishes the model we develop in this work from ReasoNet (Shen et al., 2017), a model that decides when to stop *forward* reading.

Recently, there has been various work on and around interactive environments. For instance, Nogueira and Cho (2016) proposed WebNav, a tool that automatically transforms a website into a goal-driven web navigation task. They train a neural agent to follow traces using supervised learning. Qi et al. (2019) proposed GoldEn Retriever, an iterative retrieve-and-read system that answers complex open-domain questions, which is also trained with supervised learning. Although an effective training strategy, supervised learning requires either human labeled or heuristically generated trajectories. However, there often exist multiple trajectories to solve each question, many of which may not be observed in the supervised data since it is difficult to exhaust all valid trajectories. Generalization can be limited when an agent is trained on such data.

Bachman et al. (2016) introduced a collection of synthetic tasks to train and test information-seeking capabilities in neural models. Narasimhan et al. (2016) proposed an information extraction system that acquires and incorporates external evidence to improve extraction accuracy in domains with limited data. Geva and Berant (2018) proposed a DQN-based agent that leverages the (tree) structure of documents and navigates across sentences and paragraphs. iMRC is distinct from this body of literature in that it does not depend on extra meta information to build tree structures, it is partially-observable, and its action space is as large as 200,000 (much larger than, e.g., the 5 query templates in (Narasimhan et al., 2016) and tree search in (Geva and Berant, 2018)). Our work is also inspired directly by QAit (Yuan et al., 2019), a set of interactive question answering tasks developed on text-based games. However, QAit is based on

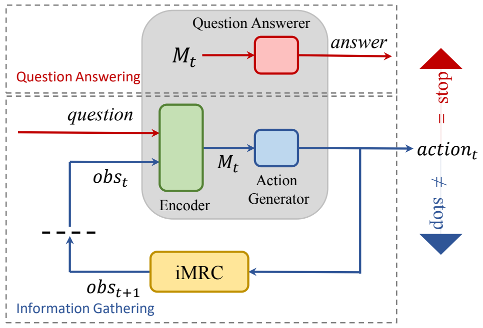

Figure 1: A demonstration of the proposed iMRC pipeline, in which the agent is illustrated as a shaded area. At a game step t, it encodes the question and text observation into hidden representations M_t. An action generator takes M_t as input to generate commands to interact with the environment. If the agent generates **stop** at this game step, M_t is used to answer question by a question answerer. Otherwise, the iMRC environment will provide new text observation in response to the generated action.

synthetic and templated language which might not require strong language understanding components. This work extends the principle of interactivity to the natural language setting, by leveraging existing MRC tasks already written in natural language.

Broadly speaking, our work is also linked to the query reformulation (QR) task in information retrieval literature (Nogueira and Cho, 2017). Specifically, QR aims to automatically *rewrite* a query so that it becomes more likely to retrieve relevant documents. Our task shares the spirit of iterative interaction between an agent (reformulator in QR) and an environment. However, the rewritten queries in QR tasks keep the semantic meaning of the original queries, whereas in our task, actions and queries across different game steps can change drastically — since our task requires an agent to learn a reasoning path (trajectory) towards answering a question, rather than to search the same concept repeatedly.

3 iMRC: Making MRC Interactive

The iSQuAD and iNewsQA datasets are based on SQuAD v1.1[2] (Rajpurkar et al., 2016) and NewsQA (Trischler et al., 2016). Both original datasets share similar properties. Specifically, each data-point consists of a tuple, $\{p, q, a\}$, where p represents a paragraph, q a question, and a is the answer. The answer is a word span defined by head and tail positions in p. NewsQA is more challenging because it has a larger vocabulary, more difficult questions, and longer source documents.

Every paragraph p is split into a list of sentences $\mathcal{S} = \{s_1, s_2, ..., s_n\}$, where n stands for number of sentences in p. At the start of a question answering episode, an agent observes the question q, but rather than observing the entire paragraph p, it sees only the first sentence s_1 while the rest is withheld. The agent must issue commands to reveal the hidden sentences progressively and thereby gather the information needed to answer q.

The agent should decide when to stop interacting and output an answer, but the number of interaction steps is limited.[3] Once the agent has exhausted its step budget, it is forced to answer the question.

3.1 Interactive MRC as a POMDP

As described in the previous section, we convert MRC tasks into sequential decision-making problems (which we will refer to as ***games***). These can be described naturally within the reinforcement learning (RL) framework. Formally, tasks in iMRC are partially observable Markov decision processes (POMDP) (Kaelbling et al., 1998). An iMRC data-point is a discrete-time POMDP defined by $(S, T, A, \Omega, O, R, \gamma)$, where $\gamma \in [0, 1]$ is the discount factor and the other elements are described in detail below.

Environment States (S): The environment state at game step t in the game is $s_t \in S$. It contains the environment's underlying conditions (e.g., the semantics and information contained in the document, which part of the document has been revealed so far), much of which is hidden from an agent, the agent can only estimate the state from its partial observations. When the agent issues an action a_t, the environment transitions to state s_{t+1} with probability $T(s_{t+1}|s_t, a_t)$. In this work, transition probabilities are either 0 or 1 (i.e., deterministic environment).

Actions (A): At each game step t, the agent issues an action $a_t \in A$. We will elaborate on the action space of iMRC in § 3.2 and § 3.3.

Observations (Ω): The text information perceived by the agent at a given game step t is the agent's observation, $o_t \in \Omega$, which depends on the environment state and the previous action with

[2]We choose SQuAD v1.1 because in this preliminary study, we focus on extractive question answering.

[3]We use 20 as the maximum number of steps, because information revealed by 20 interactions can cover a large portion of the text in either an iSQuAD or iNewsQA paragraph. A reasonable step budget also speeds up training.

probability $O(o_t|s_t)$. Again, observation probabilities are either 0 or 1 (i.e., noiseless observation).

Reward Function (R): Based on its actions, the agent receives rewards $r_t = R(s_t, a_t)$. Its objective is to maximize the expected discounted sum of rewards $E\left[\sum_t \gamma^t r_t\right]$.

3.2 Easy and Hard Modes

As a question answering dataset, we adopt the standard output format of extractive MRC tasks, where a system is required to point to a span within a given paragraph p as its prediction. However, we define two difficulty levels in iMRC, which are based on different action spaces and dynamics during the interactive information gathering phase.

Easy Mode: At a step t, an agent can issue one of the following four actions to interact with the (partially observable) paragraph p, where p consists of n sentences. Assume the agent's observation o_t corresponds to sentence s_k, where $1 \leq k \leq n$.

- `previous`: jump to $\begin{cases} s_n & \text{if } k = 1, \\ s_{k-1} & \text{otherwise;} \end{cases}$
- `next`: jump to $\begin{cases} s_1 & \text{if } k = n, \\ s_{k+1} & \text{otherwise;} \end{cases}$
- `Ctrl+F` QUERY: jump to the sentence that contains the next occurrence of QUERY;
- `stop`: terminate information gathering phase and ready to answer question.

Hard Mode: Only the `Ctrl+F` and `stop` commands are available (i.e., an agent is forced to generate QUERY to navigate the partially observable paragraph p).

3.3 QUERY Types

Given an objective (e.g., a question to answer), humans search by using both extractive and abstractive queries. For instance, when searching information about the actor "Dwayne Johnson", one may either type his name or "The Rock" in a search engine. We believe abstractive query searching requires a deeper understanding of the question, and some background knowledge (one cannot refer to "Dwayne Johnson" as the "The Rock" if they know nothing about his wrestling career).

Inspired by this observation, we study the following three settings, where in each, the QUERY is generated from different sources:

Dataset	iSQuAD	iNewsQA
#Training Games	82,441	92,550
Vocabulary Size	109,689	200,000
Avg. #Sentence / Document	5.1	29.5
Avg. Sentence Length	26.1	22.2
Avg. Question Length	11.3	7.6

Table 2: Statistics of iSQuAD and iNewsQA.

1. One token from the question: extractive QUERY generation with a relatively small action space.
2. One token from the union of the question and the current observation: still extractive QUERY generation, although in an intermediate level where the action space is larger.
3. One token from the dataset vocabulary: abstractive QUERY generation where the action space is huge (see Table 2 for statistics of iSQuAD and iNewsQA).

3.4 Evaluation Metric

Since iMRC involves both MRC and RL, we adopt evaluation metrics from both settings. First, as a question answering task, we use F_1 score to compare predicted answers against ground-truth, as in previous work. When there exist multiple ground-truth answers, we report the max F_1 score.

Second, mastering multiple games remains quite challenging for RL agents. Therefore, we evaluate an agent's performance during both its training and testing phases. Specifically, we report training curves and test results based on the best validation F_1 scores.

4 Baseline Agent

As a baseline agent, we adopt QA-DQN (Yuan et al., 2019), we modify it to enable extractive QUERY generation and question answering.

As illustrated in Figure 1, the baseline agent consists of three components: an encoder, an action generator, and a question answerer. More precisely, at a step t during the information gathering phase, the encoder reads observation string o_t and question string q to generate the attention aggregated hidden representations M_t. Using M_t, the action generator outputs commands (depending on the mode, as defined in § 3.2) to interact with iMRC. The information-gathering phase terminates whenever the generated command is `stop` or the agent has used up its move budget. The question answerer takes the hidden representation at the terminating step to generate head and tail pointers as its answer prediction.

4.1 Model Structure

In this section, we only describe the **difference** between the model our baseline agent uses and the original QA-DQN. We refer readers to (Yuan et al., 2019) for detailed information.

In the following subsections, we use "game step t" to denote the tth round of interaction between an agent with the iMRC environment.

4.1.1 Action Generator

Let $M_t \in \mathbb{R}^{L\times H}$ denote the output of the encoder, where L is the length of observation string and H is hidden size of the encoder representations.

The action generator takes M_t as input and generates rankings for all possible actions. As described in the previous section, a **`Ctrl+F`** command is composed of two tokens (the token "**`Ctrl+F`**" and the QUERY token). Therefore, the action generator consists of three multilayer perceptrons (MLPs):

$$\begin{aligned} R_t &= \mathrm{ReLU}(\mathrm{MLP}_{\text{shared}}(\mathrm{mean}(M_t))), \\ Q_{t,\text{action}} &= \mathrm{MLP}_{\text{action}}(R_t) \cdot M_{\text{mode}}, \\ Q_{t,\text{query}} &= \mathrm{MLP}_{\text{query}}(R_t) \cdot M_{\text{type}}. \end{aligned} \tag{1}$$

In which, $Q_{t,\text{action}}$ and $Q_{t,\text{query}}$ are Q-values of action token and QUERY token (when action token is "**`Ctrl+F`**"), respectively. M_{mode} is a mask, which masks the **`previous`** and **`next`** tokens in hard mode; M_{type} is another mask which depends on the current QUERY type (e.g., when QUERY is extracted from the question q, all tokens absent from q are masked out). Probability distributions of tokens are further computed by applying softmax on $Q_{t,\text{action}}$ and $Q_{t,\text{query}}$, respectively.

4.1.2 Question Answerer

Following QANet (Yu et al., 2018), we append two extra stacks of transformer blocks on top of the encoder to compute head and tail positions:

$$\begin{aligned} h_{\text{head}} &= \mathrm{ReLU}(\mathrm{MLP}_0([M_t; M_{\text{head}}])), \\ h_{\text{tail}} &= \mathrm{ReLU}(\mathrm{MLP}_1([M_t; M_{\text{tail}}])). \end{aligned} \tag{2}$$

In which, $[\cdot;\cdot]$ denotes vector concatenation, $M_{\text{head}} \in \mathbb{R}^{L\times H}$ and $M_{\text{tail}} \in \mathbb{R}^{L\times H}$ are the outputs of the two extra transformer stacks.

Similarly, probability distributions of head and tail pointers over observation string o_t can be computed by:

$$\begin{aligned} p_{\text{head}} &= \mathrm{softmax}(\mathrm{MLP}_2(h_{\text{head}})), \\ p_{\text{tail}} &= \mathrm{softmax}(\mathrm{MLP}_3(h_{\text{tail}})). \end{aligned} \tag{3}$$

4.2 Memory and Reward Shaping

4.2.1 Memory

In iMRC tasks, some questions may not be easily answerable by observing a single sentence. To overcome this limitation, we provide an explicit memory mechanism to our baseline agent to serve as an inductive bias. Specifically, we use a queue to store strings that have been observed recently. The queue has a limited number of slots (we use queues of size [1, 3, 5] in this work). This prevents the agent from issuing **`next`** commands until the environment is observed fully in memory, in which case our task degenerates to the standard MRC setting. We reset the memory slots episodically.

4.2.2 Reward Shaping

Because the question answerer in our agent is a pointing model, its performance relies heavily on whether the agent can find and stop at the sentence that contains the answer. In the same spirit as (Yuan et al., 2019), we also design a heuristic reward to guide agents to learn this behavior.

In particular, we assign a reward if the agent halts at game step k and the answer is a sub-string of o_k (if larger memory slots are used, we assign this reward if the answer is a sub-string of the memory at game step k). We denote this reward as the **sufficient information reward**, since, if an agent sees the answer, it should have a good chance of having gathered sufficient information for the question (although this is not guaranteed).

Note this sufficient information reward is part of the design of the baseline agent, whereas the question answering score is the only metric used to evaluate an agent's performance on the iMRC task.

4.3 Training Strategy

Since iMRC games are interactive environments and we have formulated the tasks as POMDPs (in § 3.1), it is natural to use RL algorithms to train the information gathering components of our agent. In this work, we study the performance of two widely used RL algorithms, one based on Q-Learning (DQN) and the other on Policy Gradients (A2C). When an agent has reached a sentence that contains sufficient information to answer the question, the task becomes a standard extractive QA setting, where an agent learns to point to a span from its observation. When this condition is met, it is also natural to adopt standard supervised learning methods to train the question answering component of our agent.

In this section, we describe the 3 training strategies mentioned above. We provide implementation details in Appendix B.

4.3.1 Advantage Actor-Critic (A2C)

Advantage actor-critic (A2C) was first proposed by Mnih et al. (2016). Compared to policy gradient computation in REINFORCE (Williams, 1992),

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t], \quad (4)$$

where the policy gradient $\nabla_\theta J(\theta)$ is updated by measuring the discounted future reward G_t from real sample trajectories, A2C utilizes the lower variance advantage function $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$ in place of G_t. The advantage $A(s_t, a_t)$ of taking action a_t at state s_t is defined as the value $Q(s_t, a_t)$ of taking a_t minus the average value $V(s_t)$ of all possible actions in state s_t.

In the agent, a critic updates the state-value function $V(s)$, whereas an actor updates the policy parameter θ for $\pi_\theta(a|s)$, in the direction suggested by the critic. Following common practice, we share parameters between actor and critic networks. Specifically, all parameters other than $\text{MLP}_{\text{action}}$ and $\text{MLP}_{\text{query}}$ (both defined in Eqn. 1) are shared between actor and critic.

4.3.2 Deep Q-Networks (DQN)

In Q-Learning (Watkins and Dayan, 1992; Mnih et al., 2015), given an interactive environment, an agent takes an action a_t in state s_t by consulting a state-action value estimator $Q(s, a)$; this value estimator estimates the action's expected long-term reward. Q-Learning helps the agent to learn an optimal value estimator. An agent starts from performing randomly and gradually updates its value estimator by interacting with the environment and propagating reward information. In our case, the estimated Q-value at game step t is simply the sum of Q-values of the action token and QUERY token as introduced in Eqn. 1:

$$Q_t = Q_{t,\text{action}} + Q_{t,\text{query}}. \quad (5)$$

In this work, we adopt the Rainbow algorithm (Hessel et al., 2017), which is a deep Q-network boosted by several extensions such as a prioritized replay buffer (Schaul et al., 2016). Rainbow exhibits state-of-the-art performance on several RL benchmark tasks (e.g., Atari games).

4.3.3 Negative Log-likelihood (NLL)

During information gathering phase, we use another replay buffer to store question answering transitions (observation string when interaction stops, question string, ground-truth answer) whenever the terminal observation string contains the ground-truth answer. We randomly sample mini-batches of such transitions to train the question answerer to minimize the negative log-likelihood loss.

5 Experimental Results

In this study, we focus on four main aspects:

1. difficulty levels (easy | hard mode);
2. strategies for generating QUERY (from question | question and observation | vocabulary);
3. sizes of the memory queue (1 | 3 | 5);
4. RL algorithms for the information gathering phase (A2C | DQN)

Regarding the four aspects, we report the baseline agent's training performance followed by its generalization performance on test data. We use DQN and A2C to refer to our baseline agent trained with DQN and A2C, respectively.

We set the maximum number of episodes (data points) to be 1 million, this is approximately 10 epochs in supervised learning tasks given the size of datasets. The agent may further improve after 1 million episodes, however we believe some meaningful and interesting trends can already be observed from the results. Besides, we hope to keep the wall clock time of the task reasonable[4] to encourage the community to work on this direction.

5.1 Mastering Training Games

It remains difficult for RL agents to master multiple games at the same time. In our case, each document-question pair can be considered a unique "game," and there are hundreds of thousands of

[4]Basic experiment setting (e.g., QUERY from question, single slot memory) take about a day on a single NVIDIA P100 GPU.

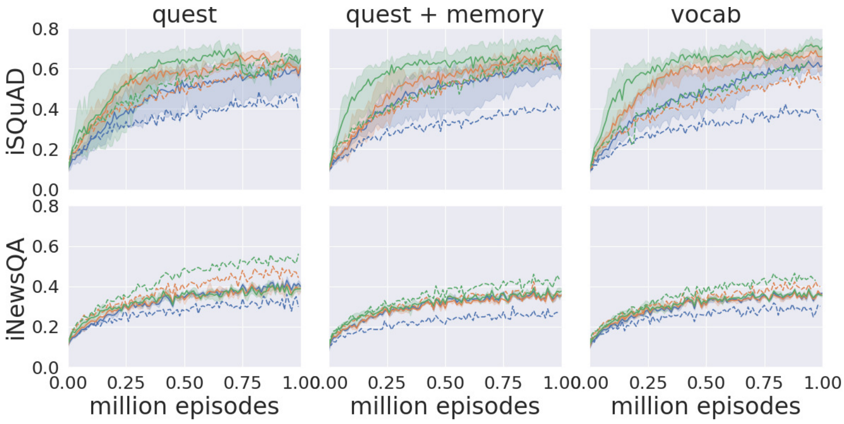

Figure 2: Training F_1 scores in **easy mode** with different QUERY types and memory sizes. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.

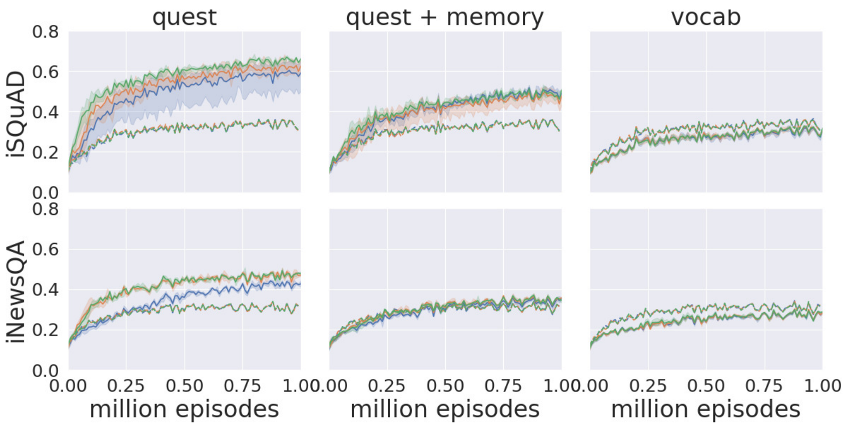

Figure 3: Training F_1 scores in **hard mode** with different QUERY types and memory sizes. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.

them. Therefore, as it is common practice in the RL literature, we study an agent's training curves.

Figure 2 and Figure 3 show the agent's training performance (in terms of F_1 score) in easy and hard mode, respectively. Due to the space limitations, we select several representative settings to discuss in this section. We provide the agent's training and validation curves for all experiments, and its sufficient information rewards (as defined in § 4.2.2) in Appendix A.

It is clear that our agent performs better on easy mode consistently across both datasets and all training strategies. This may due to the fact that the **`previous`** and **`next`** commands provide the agent an inefficient but guaranteed way to stumble on the sought-after sentence no matter the game. The **`Ctrl+F`** command matches human behavior more closely, but it is arguably more challenging (and interesting) for an RL agent to learn this behavior. RL agents may require extra effort and time to reach a desired state since they rely heavily on random exploration, and the **`Ctrl+F`** command leads to much larger action space to explore compared to commands such as **`next`**.

Related to action space size, we observe that the agent performs best when pointing to the QUERY tokens from the question, whereas it performs worst when generating QUERY tokens from the entire vocabulary. This is particularly clear in hard mode, where agents are forced to use the **`Ctrl+F`** command. As shown in Table 2, both datasets have a vocabulary size of more than 100k, whereas the average length of questions is around 10 tokens. This indicates the action space for generating QUERY from entire vocabulary is much larger. This again suggests that for moving toward a more realistic problem setting where action spaces are huge, methods with better sample efficiency are needed.

Experiments show that a larger memory queue almost always helps. Intuitively, with a memory mechanism (either explicit as in this work, or implicit as with a recurrent network aggregating rep-

Dataset		iSQuAD			iNewsQA		
Easy Mode							
QUERY	Agent	Mem=1	=3	=5	Mem=1	=3	=5
Question	A2C	0.245 (0.493)	0.357 (0.480)	0.386 (0.478)	0.210 (0.554)	0.316 (0.532)	0.333 (0.490)
	DQN	0.575 (0.770)	0.637 (0.738)	**0.666** (0.716)	0.330 (0.708)	0.326 (0.619)	**0.360** (0.620)
Question+Memory	A2C	0.221 (0.479)	0.484 (0.590)	0.409 (0.492)	0.199 (0.595)	0.233 (0.448)	0.253 (0.459)
	DQN	0.579 (0.784)	0.651 (0.734)	0.656 (0.706)	0.336 (0.715)	0.334 (0.626)	0.347 (0.596)
Vocabulary	A2C	0.223 (0.486)	0.314 (0.448)	0.309 (0.391)	0.192 (0.551)	0.224 (0.440)	0.224 (0.403)
	DQN	0.583 (0.774)	0.624 (0.738)	0.661 (0.731)	0.326 (0.715)	0.323 (0.590)	0.316 (0.593)
Hard Mode							
Question	A2C	0.147 (0.404)	0.162 (0.446)	0.158 (0.435)	0.166 (0.529)	0.160 (0.508)	0.164 (0.520)
	DQN	0.524 (0.766)	0.524 (0.740)	**0.551** (0.739)	0.352 (0.716)	**0.367** (0.632)	0.353 (0.613)
Question+Memory	A2C	0.160 (0.441)	0.150 (0.413)	0.156 (0.429)	0.163 (0.520)	0.160 (0.508)	0.164 (0.520)
	DQN	0.357 (0.749)	0.362 (0.729)	0.364 (0.733)	0.260 (0.692)	0.264 (0.645)	0.269 (0.620)
Vocabulary	A2C	0.161 (0.444)	0.163 (0.448)	0.160 (0.441)	0.160 (0.510)	0.167 (0.532)	0.162 (0.516)
	DQN	0.264 (0.728)	0.261 (0.719)	0.218 (0.713)	0.326 (0.694)	0.214 (0.680)	0.214 (0.680)

Table 3: Test F_1 scores in **black** and $F_{1\text{info}}$ scores (i.e., an agent's F_1 score iff sufficient information is in its observation when it terminates information gathering phase) in **blue**.

resentations over game steps), an agent renders the environment closer to fully observed by exploring and storing observations. Presumably, a larger memory could further improve an agent's performance; considering the average number of sentences in each iSQuAD game is 5, a memory with more than 5 slots defeats the purpose of our study of partially observable text environments.

We observe that DQN generally performs better on iSQuAD whereas A2C sometimes works better on the harder iNewsQA task. However, we observe huge gap between them on generalization performance, which we discuss in a later subsection.

Not surprisingly, our agent performs better in general on iSQuAD than on iNewsQA. As shown in Table 2, the average number of sentences per document in iNewsQA is about 6 times more than in iSQuAD. This is analogous to partially observable games with larger maps in the RL literature. We believe a better exploration (in our case, jumping) strategy that can decide where to explore next conditioned on what has already been seen may help agents to master such harder games.

5.2 Generalizing to Test Set

To study an agent's ability to generalize, we select the best performing checkpoint in each experimental setting on the validation set and report their test performance, as shown in Table 3. In addition, to support our claim that the more challenging part of iMRC tasks is information gathering rather than answering questions given sufficient information, we report the agents' F_1 scores when they have reached the piece of text that contains the answer, which we denote as $F_{1\text{info}}$.

From Table 3 (and validation curves provided in Appendix A) we observe trends that match with training curves. Due to the different sizes of action space, the baseline agents consistently performs better on the easy mode. For the same reason, the agent learns more efficiently when the QUERY token is extracted from the question. The best F_1 score on hard mode is comparable to and even slightly higher than in easy mode on iNewsQA, which suggests our baseline agent learns some relatively general trajectories in solving training games that generalize to unseen games.

It is also clear that during evaluation, a memory that stores experienced observations helps, since the agent almost always performs better with a memory size of 3 or 5 (when memory size is 1, each new observation overwrites the memory).

While performing comparably with DQN during training, the agent trained with A2C generalizes noticeably worse. We suspect this is caused by the fundamental difference between the ways DQN and A2C explore during training. Specifically, DQN relies on either ϵ-greedy or Noisy Net (Fortunato et al., 2017), both of which explicitly force an agent to experience different actions during training. In A2C, exploration is performed implicitly by sampling from a probability distribution over the action space; although entropy regularization is applied, good exploration is still not guaranteed (if there are peaks in the probability distribution). This again suggests the importance of a good exploration strategy in the iMRC tasks, as in all RL tasks.

Finally, we observe $F_{1\text{info}}$ scores are consistently higher than the overall F_1 scores, and they have less variance across different settings. This supports our hypothesis that information gathering plays an important role in solving iMRC tasks, whereas question answering given necessary information is relatively straightforward.

6 Discussion and Future Work

In this work, we propose and explore the direction of converting MRC datasets into interactive, partially observable environments. We believe information-seeking behavior is desirable for neural MRC systems when knowledge sources are partially observable and/or too large to encode in their entirety, where knowledge is by design easily accessible to humans through interaction. Our idea for reformulating existing MRC datasets as partially observable and interactive environments is straightforward and general. It is complementary to existing MRC dataset and models, meaning almost all MRC datasets can be used to study interactive, information-seeking behavior through similar modifications. We hypothesize that such behavior can, in turn, help in solving real-world MRC problems involving search. As a concrete example, in real world environments such as the Internet, different pieces of knowledge are interconnected by hyperlinks. We could equip the agent with an action to "click" a hyperlink, which returns another webpage as new observations, thus allowing it to navigate through a large number of web information to answer difficult questions.

iMRC is difficult and cannot yet be solved, however it clearly matches a human's information-seeking behavior compared to most static and fully-observable laboratory MRC benchmarks. It lies at the intersection of NLP and RL, which is arguably less studied in existing literature. For our baseline, we adopted off-the-shelf, top-performing MRC and RL methods, and applied a memory mechanism which serves as an inductive bias. Despite being necessary, our preliminary experiments do not seem sufficient. We encourage work on this task to determine what inductive biases, architectural components, or pretraining recipes are necessary or sufficient for MRC based on information-seeking.

Our proposed setup presently uses only a single word as QUERY in the **`Ctrl+F`** command in an abstractive manner. However, a host of other options could be considered in future work. For example, a multi-word QUERY with fuzzy matching is more realistic. It would also be interesting for an agent to generate a vector representation of the QUERY in some latent space and modify it during the dynamic reasoning process. This could further be used to retrieve different contents by comparing with pre-computed document representations (e.g., in an open-domain QA dataset), with such behavior tantamount to learning to do IR. This extends traditional query reformulation for open-domain QA by allowing to drastically change the queries without strictly keeping the semantic meaning of the original queries.

Acknowledgments

The authors thank Mehdi Fatemi, Peter Potash, Matthew Hausknecht, and Philip Bachman for insightful ideas and discussions. We also thank the anonymous ACL reviewers for their helpful feedback and suggestions.

References

Philip Bachman, Alessandro Sordoni, and Adam Trischler. 2016. Towards information-seeking agents. *arXiv preprint arXiv:1612.02605*.

Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. 2017. Coarse-to-fine question answering for long documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 209–220.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. 2017. Noisy networks for exploration. *CoRR*, abs/1706.10295.

Mor Geva and Jonathan Berant. 2018. Learning to search in long documents using document structure. *arXiv preprint arXiv:1806.03529*.

Moonsu Han, Minki Kang, Hyunwoo Jung, and Sung Ju Hwang. 2019. Episodic memory reader: Learning what to remember for question answering from streaming data. *arXiv preprint arXiv:1903.06164*.

Christian Hansen, Casper Hansen, Stephen Alstrup, Jakob Grue Simonsen, and Christina Lioma. 2019. Neural speed reading with structural-jump-lstm. *arXiv preprint arXiv:1904.00761*.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. 2017. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

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

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. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Karthik Narasimhan, Adam Yala, and Regina Barzilay. 2016. Improving information extraction by acquiring external evidence with reinforcement learning. *CoRR*, abs/1603.07954.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268.

Rodrigo Nogueira and Kyunghyun Cho. 2016. Webnav: A new large-scale task for natural language based sequential decision making. *CoRR*, abs/1602.02261.

Rodrigo Nogueira and Kyunghyun Cho. 2017. Task-oriented query reformulation with reinforcement learning. *CoRR*, abs/1704.04572.

Peng Qi, Xiaowen Lin, Leo Mehr, Zijian Wang, and Christopher D. Manning. 2019. Answering complex open-domain questions through iterative query generation. In *2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2018. Coqa: A conversational question answering challenge. *CoRR*, abs/1808.07042.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Neural speed reading via skim-rnn. *arXiv preprint arXiv:1711.02085*.

Lei Sha, Feng Qian, and Zhifang Sui. 2017. Will repeated reading benefit natural language understanding? In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 366–379. Springer.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2017. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1047–1055. ACM.

Saku Sugawara, Kentaro Inui, Satoshi Sekine, and Akiko Aizawa. 2018. What makes reading comprehension questions easier? *CoRR*, abs/1808.09384.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2016. Newsqa: A machine comprehension dataset. *CoRR*, abs/1611.09830.

Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning*, 8(3):279–292.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *CoRR*, abs/1809.09600.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. 2018. Qanet: Combining local convolution with global self-attention for reading comprehension. *CoRR*, abs/1804.09541.

Adams Wei Yu, Hongrae Lee, and Quoc V Le. 2017. Learning to skim text. *arXiv preprint arXiv:1704.06877*.

Xingdi Yuan, Marc-Alexandre Côté, Jie Fu, Zhouhan Lin, Christopher Pal, Yoshua Bengio, and Adam Trischler. 2019. Interactive language learning by question answering.

A Full Results

We show our experimental results (training and validation curves) in Figure 4,5,6,7,8,9,10,11.

B Implementation Details

In all experiments, we use *Adam* (Kingma and Ba, 2014) as the step rule for optimization, with the learning rate set to 0.00025. We clip gradient norm at 5.0. We initialize all word embeddings by the 300-dimensional fastText (Mikolov et al., 2018) word vectors trained on Common Crawl (600B tokens), they are fixed during training. We randomly initialize character embeddings by 200-dimensional vectors. In all transformer blocks, block size is 96.

Dimensionality of $\text{MLP}_{\text{shared}}$ in Eqn. 1 is $\mathbb{R}^{96\times150}$; dimensionality of $\text{MLP}_{\text{action}}$ is $\mathbb{R}^{150\times4}$ and $\mathbb{R}^{150\times2}$ in easy mode (4 actions are available) and hard mode (only 2 actions are available), respectively; dimensionality of $\text{MLP}_{\text{query}}$ is $\mathbb{R}^{150\times V}$ where V denotes vocabulary size of the dataset, as listed in Table 2.

Dimensionalities of MLP_0 and MLP_1 in Eqn. 2 are both $\mathbb{R}^{192\times150}$; dimensionalities of MLP_2 and MLP_3 in Eqn. 3 are both $\mathbb{R}^{150\times1}$.

During A2C training, we set the value loss coefficient to be 0.5, we use an entropy regularizer with coefficient of 0.01. We use a discount γ of 0.9 and mini-batch size of 20.

During DQN training, we use a mini-batch of size 20 and push all transitions (observation string, question string, generated command, reward) into a prioritized replay buffer of size 500,000. We do not compute losses directly using these transitions. After every 5 game steps, we sample a mini-batch of 64 transitions from the replay buffer, compute loss, and update the network. we use a discount γ of 0.9. For noisy nets, we use a σ_0 of 0.5. We update target network per 1000 episodes. For multi-step returns, we sample $n \sim \text{Uniform}[1, 2, 3]$.

When our agent terminates information gathering phase, we push the question answering transitions (observation string at this time, question string, ground-truth answer) into a question answering replay buffer. After every 5 game steps, we randomly sample a mini-batch of 64 such transitions from the question answering replay buffer and train the model using NLL loss.

For more detail please refer to our open-sourced code.

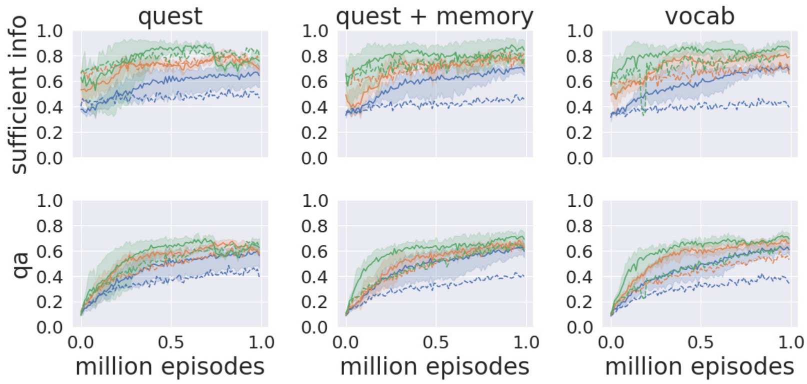

Figure 4: Training performance on iSQuAD, easy mode. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.

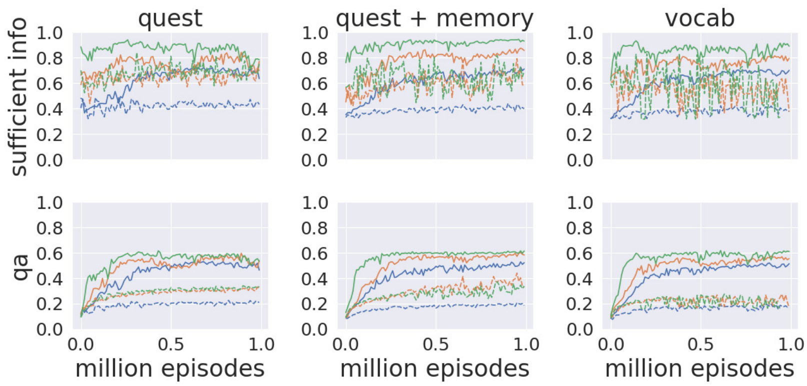

Figure 5: Validation performance on iSQuAD, easy mode. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.

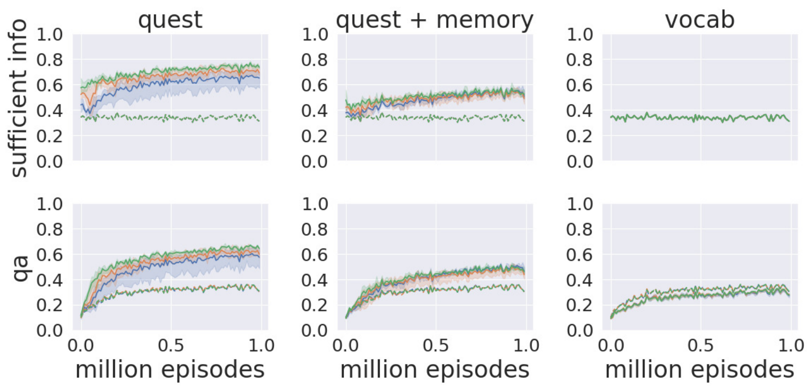

Figure 6: Training performance on iSQuAD, hard mode. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.

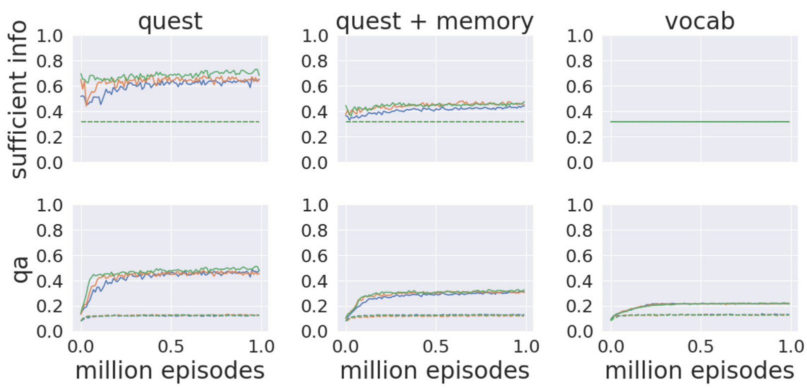

Figure 7: Validation performance on iSQuAD, hard mode. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.

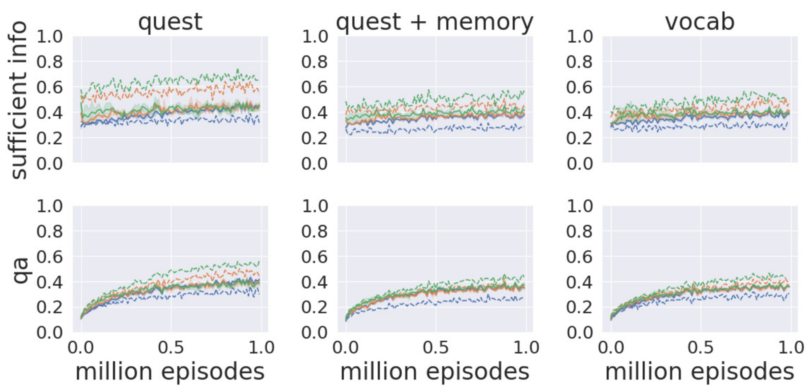

Figure 8: Training performance on iNewsQA, easy mode. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.

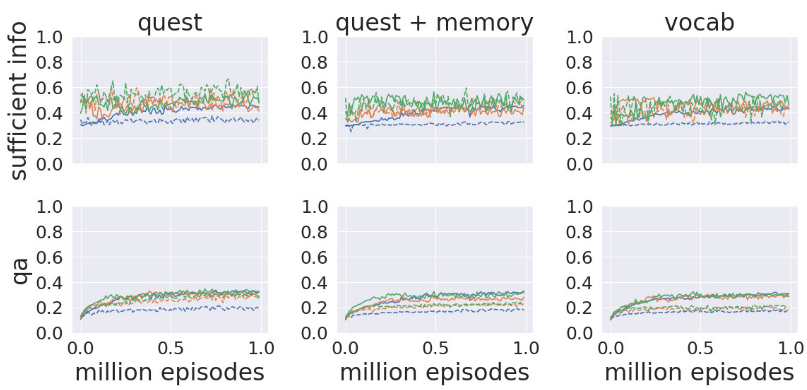

Figure 9: Validation performance on iNewsQA, easy mode. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.

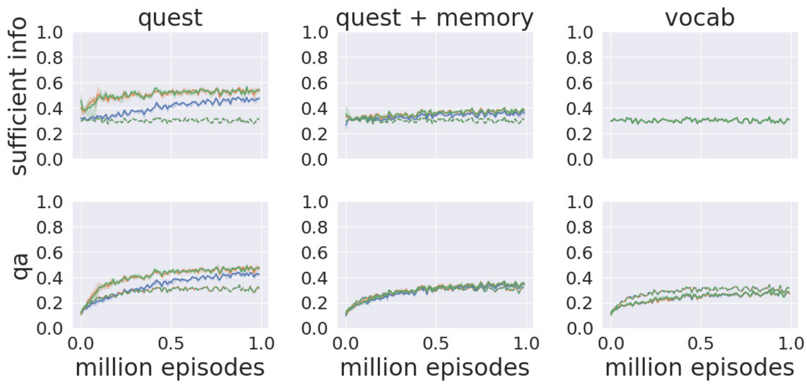

Figure 10: Training performance on iNewsQA, hard mode. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.

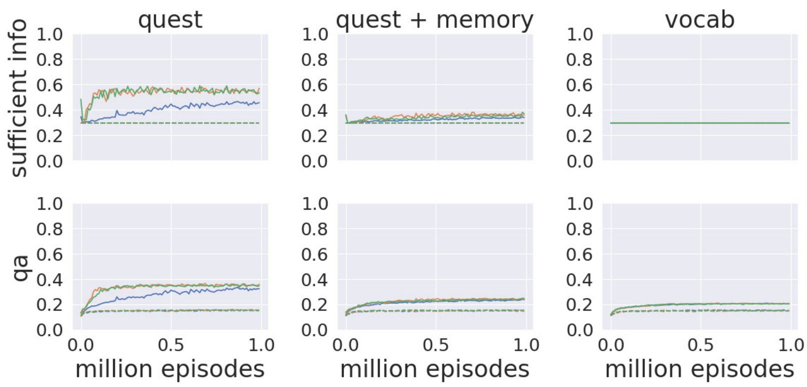

Figure 11: Validation performance on iNewsQA, hard mode. Solid line: DQN, dashed line: A2C; number of memory slots: 1, 3, 5.