NEW YORK UNIVERSITY AT TREC 2018 COMPLEX ANSWER RETRIEVAL TRACK

Rodrigo Nogueira
New York University
rodrigonogueira@nyu.edu

Kyunghyun Cho
New York University
Facebook AI Research
CIFAR Global Scholar
kyunghyun.cho@nyu.edu

ABSTRACT

In this paper, we describe our submission to the TREC-CAR 2018. We use a method introduced by Nogueira et al. (2018) to efficiently learn diverse strategies in reinforcement learning for query reformulation and focus minimally on the ranking function. In this framework, an agent consists of multiple specialized sub-agents and a meta-agent that learns to aggregate the answers from sub-agents to produce a final answer. Sub-agents are trained on disjoint partitions of the training data, while the meta-agent is trained on the full training set. Our method makes learning faster, because it is highly parallelizable, and has better generalization performance than strong baselines, such as an ensemble of agents trained on the full data.

1 INTRODUCTION

In this work, we use a simple method introduced by Nogueira et al. (2018) to achieve efficient parallelized exploration of diverse query reformulation policies. We structure the agent into multiple *sub-agents*, which are trained on disjoint subsets of the training data. Sub-agents are co-ordinated by a meta-agent, called *aggregator*, that groups and scores answers from the sub-agents for each given input. Unlike sub-agents, the aggregator is a generalist since it learns a policy for the entire training set.

We argue that it is easier to train multiple sub-agents than a single generalist one since each sub-agent only needs to learn a policy that performs well for a subset of examples. Moreover, specializing agents on different partitions of the data encourages them to learn distinct policies, thus giving the aggregator the possibility to see answers from a population of diverse agents. Learning a single policy that results in an equally diverse strategy is more challenging.

Since each sub-agent is trained on a fraction of the data, and there is no communication between them, training can be done faster than training a single agent on the full data. Additionally, it is easier to parallelize than applying existing distributed algorithms such as asynchronous SGD or A3C (Mnih et al., 2016), as the sub-agents do not need to exchange weights or gradients. After training the sub-agents, only their actions need to be sent to the aggregator.

2 RELATED WORK

The approach used in this work is inspired by the mixture of experts, which was introduced more than two decades ago (Jacobs et al., 1991; Jordan & Jacobs, 1994) and has been a topic of intense study since then. The idea consists of training a set of agents, each specializing in some task or data. One or more gating mechanisms then select subsets of the agents that will handle a new input. Recently, Shazeer et al. (2017) revisited the idea and showed strong performances in the supervised learning tasks of language modeling and machine translation. Their method requires that output vectors of experts are exchanged between machines. Since these vectors can be large, the network bandwidth becomes a bottleneck. They used a variety of techniques to mitigate this problem. Anil et al. (2018) later proposed a method to further reduce communication overhead by only exchanging

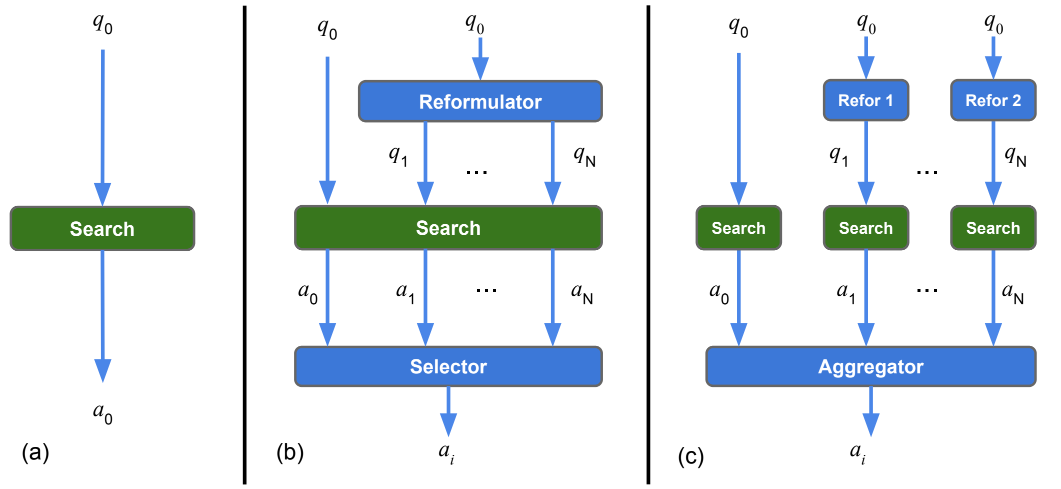


Figure 1: **a)** A vanilla search system. The query q_0 is given to the system which outputs a list of documents a_0. **b)** The search system with a reformulator. The reformulator queries the system with q_0 and its reformulations $\{q_1, ...q_N\}$ and receives back the lists of documents $\{a_0, ..., a_N\}$. A selector then decides the best list of documents a_i for q_0. **c)** The method proposed by Nogueira et al. (2018). The original query is reformulated multiple times by different reformulators. Reformulations are used to obtain documents from the search system, which are then sent to the aggregator, which merges and re-ranks the list of documents based on a learned weighted majority voting scheme. Reformulators are independently trained on disjoint partitions of the dataset thus increasing the variability of reformulations.

the probability distributions of the different agents. Our method, instead, requires only scalars (rewards) and short strings (original query, reformulations, and answers) to be exchanged. Therefore, the communication overhead is small.

Previous works used specialized agents to improve exploration in RL (Dayan & Hinton, 1993; Singh, 1992; Kaelbling et al., 1996). For instance, Stanton & Clune (2016) and Conti et al. (2017) use a population of agents to achieve a high diversity of strategies that leads to better generalization performance and faster convergence. Rusu et al. (2015) use experts to learn subtasks and later merge them into a single agent using distillation (Hinton et al., 2015).

The experiments are often carried out in simulated environments, such as robot control (Brockman et al., 2016) and video-games (Bellemare et al., 2013). In these environments, rewards are frequently available, the states have low diversity (e.g., same image background), and responses usually are fast (60 frames per second). We, instead, evaluate our approach on tasks whose inputs (queries) and states (documents and answers) are diverse because they are in natural language, and the environment responses are slow (0.5-5 seconds per query).

3 Method

3.1 Task

We describe the method using a generic end-to-end search task. The problem consists in learning to reformulate a query so that the underlying retrieval system can return a better list of documents.

Following Nogueira & Cho (2017) and Buck et al. (2018) we frame the task as a reinforcement learning problem, in which the query reformulation system is an RL-agent that interacts with an environment that provides answers and rewards. The goal of the agent is to generate reformulations such that the expected returned reward (i.e., correct answers) is maximized. The environment is treated as a black-box, i.e., the agent does not have direct access to any of its internal mechanisms. Figure 1-(b) illustrates this framework.

3.2 System

Figure 1-(c) illustrates the main method used in our submissions. An input query q_0 is given to the N sub-agents. A sub-agent is any system that accepts as input a query and returns a corresponding reformulation. Thus, sub-agents can be heterogeneous.

Here we train each sub-agent on a partition of the training set. The i-th agent queries the underlying search system with the reformulation q_i and receives a list of documents a_i. The set $\{(q_i, a_i)|0 \leq i \leq N\}$ is given to the aggregator, which then merges the lists of documents into a final list and re-rank it.

3.3 Sub-agents

The first step for training the new agent is to partition the training set. We randomly split it into equal-sized subsets. For an analysis of how other partitioning methods affect performance, see Nogueira et al. (2018). In our implementation, a sub-agent is a sequence-to-sequence model (Sutskever et al., 2014; Cho et al., 2014) trained on a partition of the dataset. It receives as an input the original query q_0 and outputs a list of reformulated queries (q_i) using beam search.

Each reformulation q_i is given to the same environment that returns a list of documents $(a_i^1, .., a_i^K)$ and a reward r_i. We then use REINFORCE (Williams, 1992) to train the sub-agent. At training time, instead of using beam search, we sample reformulations.

Note that we also add the identity agent (i.e., the reformulation is the original query) to the pool of sub-agents.

3.4 Meta-Agent: Aggregator

The aggregator receives as inputs q_0 and a list of candidate documents $(a_i^1, ..a_i^K)$ for each reformulation q_i. We first compute the set of unique documents a_j and two different scores for each query-document pair: the accumulated rank score s_j^A and the relevance score s_j^R.

The accumulated rank score is computed as $s_j^A = \sum_{i=1}^{N} \frac{1}{\text{rank}_{i,j}}$, where $\text{rank}_{i,j}$ is the rank of the j-th document when retrieved using q_i. The relevance score s_j^R is the prediction that the document a_j is relevant to query q_0. It is computed as:

$$s_j^R = \sigma(W_2 \text{ReLU}(W_1 z_j + b_1) + b_2), \tag{1}$$

where

$$z_j = f_{\text{CNN}}(q_0) || f_{\text{BOW}}(a_j) || f_{\text{CNN}}(q_0) - f_{\text{BOW}}(a_j) || f_{\text{CNN}}(q_0) \odot f_{\text{BOW}}(a_j), \tag{2}$$

$W_1 \in \mathbb{R}^{4D \times D}$ and $W_2 \in \mathbb{R}^{D \times 1}$ are weight matrices, $b_1 \in \mathbb{R}^D$ and $b_2 \in \mathbb{R}^1$ are biases. The symbol $||$ denotes the concatenation operation, σ is the sigmoid function, and ReLU is a Rectified Linear Unit function (Nair & Hinton, 2010). The function f_{CNN} is implemented as a CNN encoder[1] followed by average pooling over the sequence (Kim, 2014). The function f_{BOW} is the average word embeddings of the document. At test time, the top-K answers with respect to $s_j = s_j^A s_j^R$ are returned.

We train the aggregator with stochastic gradient descent (SGD) to minimize the cross-entropy loss:

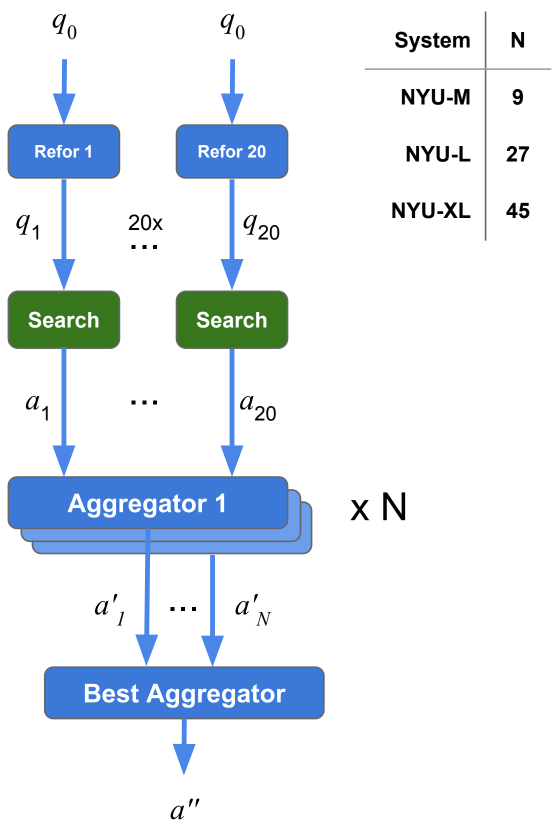
$$L = -\sum_{j \in J^*} \log(s_j^R) - \sum_{j \notin J^*} \log(1 - s_j^R), \tag{3}$$

where J^* is the set of indexes of the ground-truth documents. The architecture details and hyperparameters can be found in Appendix A.

4 Experiments

We now present experiments and results in the TREC-CAR task. In this task, the goal is to rewrite a query so that the number of relevant documents retrieved by a search engine increases.

[1]In the preliminary experiments, we found CNNs to work better than LSTMs (Hochreiter & Schmidhuber, 1997).



System	N
NYU-M	9
NYU-L	27
NYU-XL	45

Figure 2: Illustration of the three systems (NYU-M, NYU-L, NYU-XL) we submitted to TREC-CAR 2018. We first train 20 reformulators as in RL-20-Sub (Section 4.5). We then freeze them (i.e., no further training) and train N aggregators, where N varies depending on the system (see the table on the top-right). Each aggregator receives as input the same 20 lists of documents produced by the 20 reformulators. Each aggregator architecture is randomly chosen (Section 4.6), and they are trained independently (i.e., no communication). Finally, the N lists of documents (a'_1, ..., a'_N) produced by the N aggregators are merged and re-ranked by the aggregator that performed best out of the N aggregators on the development set.

4.1 Environment

The environment receives a query as an action, and it returns a list of documents as an observation/state and a reward computed using a list of ground truth documents. We use Lucene[2] in its default configuration[3] as our search engine. The input is a query, and the output is a ranked list of documents.

4.2 Dataset

Introduced by Dietz & Ben (2017), in the TREC-CAR dataset, the input query is the concatenation of a Wikipedia article title with the title of one of its section. The ground-truth documents are the paragraphs within that section. The corpus consists of all of the English Wikipedia paragraphs, except the abstracts. We used corpus v2.0, as it has a better paragraph parsing than the last year's v1.5. The released dataset has five predefined folds, and we use the first four as a training set (approx. 3M queries), and the remaining as a validation set (approx. 700k queries). The test set is the same used evaluate the submissions to TREC-CAR 2017 (approx. 2,250 queries), and the ground-truth documents are from the automatic annotations. We did not use the manual annotations because not all documents in the corpus were annotated for each test query (as this would be very

[2]https://lucene.apache.org/
[3]The ranking function is BM25.

	R@40	MAP	R-Prec	MRR	NDCG	Training (Days)	FLOPs ($\times 10^{18}$)
Lucene	25.7	9.4	8.3	17.7	15.4	N/A	
PRF	26.8	9.8	8.6	18.4	16.1	N/A	
RM3	28.0	10.2	9.0	19.2	16.8	N/A	
RL-RNN	29.8	10.8	9.4	20.3	17.8	10	2.3
RL-10-Ensemble	30.1	10.9	9.5	20.5	18.0	10	23.0
RL-RNN 20 Sampled + Aggregator	30.7	11.1	9.7	20.8	18.3	10	2.3
RL-10-Full	33.9	12.2	10.5	22.8	20.2	1	2.3
RL-10-Sub	34.9	12.3	10.6	23.2	20.5	1	2.3
RL-10-Sub (Pretrained)	35.1	12.5	10.8	23.5	20.8	$10^{\star}$+1	4.6
RL-10-Full (Extra Budget)	35.9	12.9	11.0	24.1	21.1	10	23
NYU-L (Aggregators' Ensemble)	37.7	14.3	12.6	25.8	23.0	12	23+$1.6^{\star\star}$

Table 1: Results on the test set of the TREC-CAR 2017 (Y1). *The weights of the agents are initialized from a single model pretrained for ten days on the full training set. **Each of the 27 aggregators costs 0.06×10^{18} FLOPs, so their total training cost is $27 \times 0.06 \times 10^{18} \approx 1.6 \times 10^{18}$ FLOPs.

System	MAP	R-Prec	NDCG
NYU-M	15.76	17.58	33.27
NYU-L	15.64	17.69	34.18
NYU-XL	20.64	22.04	45.22

Table 2: Official results of our three submissions to the TREC-CAR 2018 (Y2 test queries) when evaluated with manual annotations.

time-consuming), so systems that retrieve non-annotated but relevant documents would be unfairly penalized.

4.3 Reward

Since the main goal of query reformulation is to increase the proportion of relevant documents returned, we use recall as the reward: R@$K = \frac{|D_K \cap D^*|}{|D^*|}$, where D_K are the top-K retrieved documents and D^* are the relevant documents. We also experimented using as a reward other metrics such as NDCG, MAP, MRR, and R-Precision but these resulted in similar or slightly worse performance than Recall@40. Despite the agents optimizing for Recall, we report the results in MAP, NDCG, MRR, and R-Precision as these are more commonly used metrics in information retrieval.

4.4 Baselines

Lucene: We give the original query to Lucene and use the retrieved documents as results.

PRF: This is the pseudo relevance feedback method (Rocchio, 1971). We expand the original query with terms from the documents retrieved by the Lucene search engine using the original query. The top-N TF-IDF terms from each of the top-K retrieved documents are added to the original query, where N and K are selected by a grid search on the validation data.

Relevance Model (RM3): This is our implementation of the relevance model for query expansion (Lavrenko & Croft, 2001). The probability of adding a term t to the original query is given by:

$$P(t|q_0) = (1 - \lambda)P'(t|q_0) + \lambda \sum_{d \in D_0} P(d)P(t|d)P(q_0|d), \tag{4}$$

where $P(d)$ is the probability of retrieving the document d, assumed uniform over the set, $P(t|d)$ and $P(q_0|d)$ are the probabilities assigned by the language model obtained from d to t and q_0, respectively. $P'(t|q_0) = \frac{\text{tf}(t \in q)}{|q|}$, where tf$(t, d)$ is the term frequency of t in d. We set the interpolation parameter λ to 0.65, which was the best value found by a grid-search on the development set.

We use a Dirichlet smoothed language model (Zhai & Lafferty, 2001) to compute a language model from a document $d \in D_0$:

$$P(t|d) = \frac{\text{tf}(t, d) + uP(t|C)}{|d| + u}, \tag{5}$$

where u is a scalar constant ($u = 1500$ in our experiments), and $P(t|C)$ is the probability of t occurring in the entire corpus C.

We use the N terms with the highest $P(t|q_0)$ in an expanded query, where $N = 100$ was the best value found by a grid-search on the development set.

RL-RNN: This is the sequence-to-sequence model trained with reinforcement learning from Nogueira & Cho (2017). The reformulated query is formed by appending new terms to the original query. The terms are selected from the documents retrieved using the original query. The agent is trained from scratch.

RL-N-ENSEMBLE: We train N RL-RNN agents with different initial weights on the full training set. At test time, we average the probability distributions of all the N agents at each time step and select the token with the highest probability, as done by Sutskever et al. (2014).

4.5 BASE MODELS

We evaluate the following variants of the method proposed by Nogueira et al. (2018):

RL-N-FULL: We train N RL-RNN agents with different initial weights on the full training set. The answers are obtained using the best (greedy) reformulations of all the agents and are given to the aggregator.

RL-N-SUB: This agent is similar to RL-N-Full, but the multiple sub-agents are trained on random partitions of the dataset (see Figure 1-(c)).

RL-RNN SAMPLE + AGGREGATOR We sample K rewrites from a single reformulator trained on the full dataset. The K lists of ranked documents returned by the environment are then merged into a single list and re-ranked by the Aggregator.

4.6 SYSTEMS SUBMITTED TO TREC-CAR 2018

Our three submissions consist of 20 reformulators trained as in RL-20-Sub and an ensemble of N aggregators (Figure 2), where N is 9, 27, and 45 for NYU-M, NYU-L, and NYU-XL, respectively. Each architecture of the N aggregators is randomly chosen as follows: first, we sample the number of layers from {0, 1, 2, 3}. We then sample the number of hidden units for each layer from {256, 512, 1024}, such that a layer must have at most the same number of hidden units of the previous layer. A final layer with one hidden unit is always added so the model can output a score for each query-document pair (Equation 1, s_j^R). We train these N aggregators using as input the documents from 20 reformulators. The final list of documents (a'') is the result of merging and re-ranking the N list of documents (a'_1, ..., a'_N) using the aggregator that performed best on the development set.

4.7 RESULTS

The results using the TREC-CAR 2017 (Y1) test queries and ground-truth documents derived from the automatic annotations on the paragraph corpus v2.0 (2018/Y2 release) are shown in Table 1. We estimate the number of floating point operations used to train a model by multiplying the training time, the number of GPUs used, and 2.7 TFLOPS as an estimate of the single-precision floating-point of a K80 GPU.

Since the sub-agents are frozen during the training of the aggregator, we pre-compute all (q_0, q_i, a_i, r_i) tuples from the training set, thus avoiding sub-agent or environment calls. This reduces its training time to less than 6 hours (0.06×10^{18} FLOPs). Since this cost is negligible when compared to the sub-agents', we do not include it in the table.

The methods RL-10-{Sub, Full} have 20-60% relative performance improvement over the standard ensemble (RL-10-Ensemble) while training ten times faster. More interestingly, RL-10-Sub has a better performance than the single-agent version (RL-RNN), uses the same computational budget, and trains on a fraction of the time. Lastly, we found that RL-10-Sub (pretrained) has the best balance between performance and training cost across all datasets.

For more experiments regarding varying number of sub-agents, training stability, and the aggregator's contribution to the overall performance, see Nogueira et al. (2018).

Finally, we show on Table 2 the official results of our three TREC-CAR 2018 submissions when evaluated with manual annotations.

5 Conclusion

We evaluated a method to build a better query reformulation system by training multiple sub-agents on partitions of the data using reinforcement learning and a simple aggregator that learns to combine the answers of the multiple agents given a new query. We showed the effectiveness and efficiency of the approach on the TREC-CAR 2017 test set.

References

Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*, 2018.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Christian Buck, Jannis Bulian, Massimiliano Ciaramita, Andrea Gesmundo, Neil Houlsby, Wojciech Gajewski, and Wei Wang. Ask the right questions: Active question reformulation with reinforcement learning. In *Proceedings of ICLR*, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv preprint arXiv:1712.06560*, 2017.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

Laura Dietz and Gamari Ben. Trec car: A data set for complex answer retrieval. *http://trec-car.cs.unh.edu*, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

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

Victor Lavrenko and W Bruce Croft. Relevance based language models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 120–127. ACM, 2001.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Rodrigo Nogueira and Kyunghyun Cho. Task-oriented query reformulation with reinforcement learning. *arXiv preprint arXiv:1704.04572*, 2017.

Rodrigo Nogueira, Jannis Bulian, and Massimiliano Ciaramita. Learning to coordinate multiple reinforcement learning agents for diverse query reformulation. *arXiv preprint arXiv:1809.10658*, 2018.

Joseph John Rocchio. Relevance feedback in information retrieval. *The SMART retrieval system: experiments in automatic document processing*, pp. 313–323, 1971.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Satinder P Singh. Reinforcement learning with a hierarchy of abstract models. In *AAAI*, pp. 202–207, 1992.

Christopher Stanton and Jeff Clune. Curiosity search: producing generalists by encouraging individuals to continually explore and acquire skills throughout their lifetime. *PloS one*, 11(9):e0162235, 2016.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

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

Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 334–342. ACM, 2001.

Appendix A Hyperparameters

Sub-agents: We use mini-batches of size 256, ADAM (Kingma & Ba, 2014) as the optimizer, and learning rate of 10^{-4}.

Aggregator: The encoder f_{q_0} is a word-level two-layer CNN with filter sizes of 9 and 3, respectively, and 128 and 256 kernels, respectively. $D = 512$. No dropout is used. ADAM is the optimizer with learning rate of 10^{-4} and mini-batch of size 64. It is trained for 100 epochs.