1 INTRODUCTION

During this project we set to take on the foundations laid down in Geva et al. (2018), which introduced a novel framework to navigate document structure for the reading comprehension task using deep reinforcement learning and improve upon it. The goal of the Reading Comprehension (RC) task is to enable machines to understand documents, and answer questions about them. Previous work on the RC task for long documents relied on scanning with a RNN entire documents token-by-token which was time consuming even with a basic model. The Information Retrieval (IR) approach is to retrieve excerpts from the document and over these excerpts run an RNN, this approach is problematic since as document length increases, the efficiency of one-shot IR methods decreases and thousands of tokens are retrieved.

In contrast, the work done in Geva et al. (2018) introduces a new approach, in the article documents were represented as trees, and a reinforcement learning agent learns to combine tokens from the document tree with feedback from a more expressive answer extraction model to navigate within the tree. The authors showed that it is possible to train an RL agent to successfully navigate a document while consuming a fraction of the tokens that a standard RC model uses.

We follow up on this approach and further study the methods proposed. We accomplish this by redesigning of the code base used to allow use of any general actor-critic reinforcement learning framework for the TRIVIAQA-NoP dataset. We showcase our results using the IMPALA framework and the V-trace algorithm introduced in Espelholt et al. (2018), and denote our model IMPALA Question Answer Navigator (IQAN)\textsuperscript{2}.

In order to highlight the impact the new architecture has on training, we modify the original environment to use a sparse reward function, remove the replay buffer and do not employ tree sampling, despite those changes, IQAN is able to converge and reaches a new SOTA on the TRIVIAQA-NoP environment set in a fraction of the time it took DocQN, and using a fraction of the transitions DocQN used. We conclude that this is because of the off-policy nature of DocQN, and the fact that at in point DocQN either updates or performs steps in the environment, but not both. We circumvent this issue with the IMPALA architecture which updates and acts concurrently. V-trace is a general off-policy learning algorithm, this work uses it mostly in an on-policy fashion, which allows us to optimize our policy much more efficiently, since using on-policy methods allows the model to discover “mistakes” much faster. On the TRIVIAQA-NoP dataset, we improve by 3% against the DocQN model and by 6.2% against the DQN baseline, since many components were omitted, this suggests future improvements are highly likely.

2 BACKGROUND

2.1 Markov Decision Process

An infinite horizon Markov Decision Process (MDP) is a tuple \((S,A,R,P,\gamma)\) where \(S\) is the state space, \(A\) is the action space \(R\) is the reward function, \(P\) is the stochastic transition function and \(\gamma \in (0,1]\) is the discount factor. The environment derived from an MDP is one where at each step an agent takes an action w.r.t to a state and receives a reward and the next current state. A deterministic policy is a function to the actions given a state, and a stochastic policy is a distribution on the actions given a state. For a policy \(\pi\), we define the Value function for every state as \(V^\pi(s) = E^\pi[\sum_{t \ge 0} \gamma^t \cdot r_t]\) where \(r_t = R(s_t, a_t)\) and \(a_t \sim \pi(.|s_t)\), and the Q function as \(Q^\pi(s,a) = R(s,a) + \gamma V^\pi(s')\). The primary goal of such model is to find a policy \(\pi\) that maximizes the sum of future discounted rewards.
2.2 On policy vs Off policy

Within the context of RL, a strictly off-policy learning algorithm, given episodes/trajectories of the form $s_1, a_1, r_1, \ldots, s_T, a_T, r_T$ generated using a behavior policy $\mu$, the goal is to learn the optimal value function and in turn the optimal policy $\pi^*$, i.e., it has no access to the environment except for “logs” in the form of episodes that it receives. An on-policy learning algorithm on the other hand, is able to take actions in the environment and see the effect of the actions it took. This allows a much more informed learning process, since the model is able to explore the state space much more effectively, and correct inaccurate estimations quickly.

3 RELATED WORK

DocQN and TRIVIAQA\textsubscript{Anop}

To solve the RC task, Geva et al. (2018) used a variant of the TriviaQA dataset Joshi et al. (2017), which contains question-answer pairs, along with a small set of documents that in almost all cases contain the answer. Because TriviaQA is biased towards answers appearing at the beginning of the document, Geva et al. (2018) introduced a new modified version of TriviaQA called TriviaQA-NoP, where the preface section is removed from every document. The training data was comprised of question-document-answer triplets, and the training was done with the Deep Q-Network (DQN) algorithm. During training, the agent observes only a small portion of the local text to decide on the next action, which can be either movement to another node, answering the question with a more computationally expensive RC model, or terminating navigation. Doing so, the agent uses only a small part of the document at each navigation step. The main motivation behind this model is that a deep model for RC is hard to train, and inference is relatively computationally intensive. The advanced RC model we used is RaSoR introduced in Lee et al. (2016), which was pretrained on TriviaQA and achieved 53.4% F1. In order to decrease the training time, Geva et al. (2018) saved the 100 top predictions for every document, the predictions were then stored in a file, and a separate remote procedure call (RPC) server was run to allow the model to query for predictions for the “answer” action. The model employed a deterministic policy and exploration was done greedily, and uses scheduling to decrease $\epsilon$ through time. The primary model DocQN used tree sampling techniques and a standard DQN model was trained to act as a baseline.

Other work

As new advancements in the field emerge, the use of RL for Natural Language Processing (NLP) has increased for many common tasks for example: Machine Translation (Grissom II et al. 2014) Language Modeling (Fedus et al. 2018) and Summarization (Paulus et al. 2017). For the task of question answering for long documents, similarly to us (Choi et al. 2016) used a combination of two models, a coarse model and a fine model, unlike Choi et al. 2016 we evaluate on a much broader set of questions. Most recently, (Wang et al. 2017) introduced an end-to-end model that trains RL agents for both the answer extraction task and the ranking task. Unlike (Wang et al. 2017), we use the information about the structure of document trees which has the potential for consuming exponentially less tokens.

Deep architecture for NLP applications are known to be hard to train, therefore, our choice to use the IMPALA architecture is due to the fact that is the previous state-of-the-art distributed architecture for RL, A3C (Volodymyr Mnih, Badia, et al. 2016) was not scalable enough to be used for our needs, since the mechanism where the actors calculate the loss and pass it to the learn resulted in major policy-lag. Unlike A3C, actors in IMPALA pass entire trajectories to the Learner. Another recent distributed architecture is Ape-X (Horgan et al. 2018), which allows the use of the Prioritized Replay Buffer in a distributed setting, which seemed like a natural continuation for DocQN. At first we attempted to combine IMPALA and Ape-X, but due to time constraints we weren’t able to do so.

4 PROBLEM SETUP

The TriviaQA-NoP environment set introduced in Geva et al. (2018) is as follows: given a question-document-answer triplets $\{(q, d, a_i)\}_{i=1}^N$, we define an environment as $(q, d, a)$, for the node $u$ in the document tree $d$, a state is of the form $s^u = (q, o, \phi_n, \phi_z)$ where $q$ is the question, $o$ is the concatenated path from the title down to the sentence, $\phi_n$ are the navigation features and $\phi_z$ are the current answer prediction features from RaSoR. Every node $u$ corresponds to a structural element and is labeled with text $(u)$ i.e the title of the document, the name of a section or the text in a paragraph. In addition, the index of every non-sentence tree node $u$ is defined by the linear order of the text and is denoted as $n(u)$, and the index of every sentence node is the index of its direct ancestor. A navigation example can be seen in Figure 7.

The possible actions are answering the question...
with a RC model, stopping navigation, along with the movement actions in Figure 2. While for legal moves the environment allows the agent to move deterministically, if the agent chooses an action that is not legal, e.g. moving right when he is currently in the rightmost node, for those cases, we do not allow movement. Therefore, since the environment in our case is almost always deterministic, and the state space is exponential, we still benefit from the formalism of an MDP.

### Reward functions

The reward in the environment is based on whether the agent locates a node that contains the answer, and can be seen in Figure 1. The reward function $R_0$ is the one used in Geva et al. (2018), and is based on the normalized line distance from the closest answer index, we believe this function may allow false positives to occur, since for a long documents, the agent can stop in a different section from an answer node and still receive a positive reward. The first reward function we choose to use was $R_1$, which we believe is much less noisy than $R_0$, since $R_1$ is much more sparse. Since the model can still stop at the paragraph level and still receive positive reward, we created $R_2$ which also requires that the model will stop on a sentence node, this contains more tokens and therefore more expressive and informative, the downside is that it results in more sparseness. We use $R_0$ in an experiment to investigate the influence the number of actors used effects convergence rate.

### 5 METHOD

The network we used is as seen in Figure 6.

\begin{align*}
R_0 &= \begin{cases} 
2 & a = \text{STOP}, |n(u) - n(u^*)| = 0 \\
1 - \frac{|n(u) - n(u^*)|}{\max_n(n(u))} & a = \text{STOP}, |n(u) - n(u^*)| > 0 \\
-0.06 & a = \text{ANSWER} \\
-0.02 & o.w
\end{cases} \\
R_1 &= \begin{cases} 
1 & a = \text{STOP}, |n(u) - n(u^*)| = 0 \\
0 & o.w
\end{cases} \\
R_2 &= \begin{cases} 
1 & a = \text{STOP}, |n(u) - n(u^*)| = 0 \\
0 & o.w
\end{cases}
\end{align*}

Figure 1: The reward functions in used in this work. Note: $u^*$ is one of the golden answers.

We leverage the recent advancements in distributed RL, as shown in Espeholt et al. 2018, the high throughput enabled by IMPALA results in order of magnitude faster learning.

The main difference between our work and Geva et al. (2018) are the fact that we use the V-trace actor-critic algorithm as described in Espeholt et al. 2018. Note that although V-trace is a general off-policy learning algorithm, when comparing to DocQN we use it in a an on-policy fashion and since in the on-policy case V-trace reduces to TD(0) n-step return, which is known to improve convergence time (Sutton et al. 1998). In contrast with DocQN, to encourage exploration, we use a stochastic log-linear policy, which is a much more natural from of exploration. The log-linear policy also removes the need for additional hyper-parameters for scheduling ε-greedy exploration. Also, note that the state space used in this work is identical to the one used in Geva et al. (2018).

### 5.1 IMPALA

#### 5.1.1 Loss function

The Q function has the regression loss w.r.t the V-trace target, the policy gradient update uses a REINFORCE update with the V-trace target and the value function as baseline, and as in A3C (Volodymyr Mnih, Badia, et al. 2016), we add an entropy bonus to avoid premature convergence, each of those is scaled by their appropriate hyper-parameter $\beta$.

#### 5.1.2 V-trace target

Unlike other actor-critic architectures such as A3C (Volodymyr Mnih, Badia, et al. 2016), in which actors pass the loss, which causes a policy lag, actors in IMPALA pass the trajectories to a learner. Along with the logits of the executing policy, the logits serve as a way to indicate to the learner how much the actor was sure of the action he took. The learner in part calculates the logits for his own estimation of the state, then the ratio of the probabilities $\frac{q(a_t | s_t)}{p(a_t | s_t)}$ is calculated and if it is smaller then...
a truncation hyperparameter $\tilde{\rho}$, this indicates that the discrepancy between the two is high, and therefore we would like to give less significance to this estimation. The importance weight is then defined as $\rho_t = \min(\tilde{\rho}, \frac{1}{\pi(a_t|s_t)}, \text{for the action taken.}

The V-trace is calculated with backwards induction, as in Algorithm 1.

Algorithm 1 V-trace - backwards view

Given a trajectory $s_1, a_1, r_1, ... s_T, a_T, r_T$
for $t = T, ..., 1$
\[ \delta_t V = \rho_t \cdot (r_t + \gamma V(s_{t+1}) - V(s_t)) \]
\[ v_t = V(s_t) + \delta_t V + \gamma \rho_t \cdot (v_{t+1} - V(s_{t+1})) \]

Algorithm 2 IMPALA Learner

Initialize policy and q-value parameters $\omega, \theta$
Initialize episode queue $Q$, batch size $B$, epoch number $M$
for $t = 1, ..., M$
\[ s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1} \leftarrow Q.\text{enqueue}(B) \]
for $i = 1, ..., T$ accumulate gradients
\[ \theta = \theta + \beta_1 \cdot (v_i - V_\theta(s_i)) \nabla_\theta V_\theta(s_i) \]
\[ \omega = \omega + \beta_2 \cdot \rho_i \nabla_\omega \log \pi_\omega(a_i|s_i)(r_i + \gamma v_{i+1} - V_\theta(s_i)) \]
\[ \omega = \omega - \beta_3 \cdot \nabla_\omega H[\pi_\omega(|s_i)] \]

Algorithm 3 IMPALA Actor

Acquire reference to q-value network parameters $\theta$
Acquire reference to episode queue $Q$
while True
Sample start state $s_0$, initialize transition list $L$
for $t = 1, ..., T_{\text{max}}$
\[ q_t \leftarrow Q_\theta(|s_t|) \]
sample action $a_t \sim \text{softmax}(q_t)$
\[ r_t, s_{t+1} \leftarrow R(s_t, a_t), P(s_t, a_t) \]
\[ L.\text{add}(s_t, a_t, r_t, q_t) \]
if $a_t = \text{STOP}$
break
Q.$\text{enqueue}(L)$

5.1.3 The omission of the Prioritized Replay Buffer

In our testing we found that using a buffer as in the original DocQN is not beneficial for the following reasons: (a) With a large enough number of actors we can have a throughput high enough to update the network with just the transitions from the trajectory queue while keeping high GPU utilization. (b) With the increased throughput transitions inside the replay buffer became stale very quickly. (c) The replay buffer became a bottleneck since the number of transition insertion per second

\[ \text{Algorithm 3 IMPALA Actor} \]

| FAO1 = \{q \mid \text{fao}(q) < 3\} |
| FAO2 = \{q \mid 3 \leq \text{fao}(q) < 22\} |
| FAO3 = \{q \mid 22 < \text{fao}(q)\} |

Table 1: FAO partition.

of around 17 was not scalable to a large number of actors.

6 RESULTS

We compare our work with Geva et al. (2018) in the x-axis with respect to several statistics: the first being the number of times we update the function, the second with respect to how many total transitions we updated the parameters up to that point, and finally w.r.t to wall clock time.

6.1 First Answer Occurrence

The index of the first answer occurrence is a good indicator of the difficulty of the question, we differentiate between examples in the dataset using this feature and partition the dataset into 3 parts as in Table 1. We will discuss levels of converges with respect to this partition.

6.2 METRICS

6.2.1 Navigation Performance

We define a successful navigation when the model reaches a paragraph containing one of the golden answers. The navigation accuracy of them model is then the percentage of successful navigations.

6.2.2 Summary collection

In order to collect summaries, we use a modified actor algorithm where periodically we sample a batch from the train set and the development set and perform inference using a greedy policy, since the the log-linear policy, is primary for the benefit of exploration and the policy gradient loss.

At first we used a mechanism where every actor once every 3 episodes collects summaries about a sample, and once 400 samples have been summarized save those to the file. This can cause a bias towards a shorter trajectories since in a fixed amount of time given a fixed amount of actors more short trajectories will appear, but since the model performs better on shorter trajectories, we found our results to be skewed. Notice that the same bias appears in the network update, currently, the model slowly learns that stopping early is not beneficial for examples in FAO2 and FAO3. To combat this

\[ ^3 \text{IMPALA used another truncation hyperparameter } \tilde{\rho} \text{ to control the rate of convergence, we chose to use } \tilde{\rho} = \tilde{\rho} = 1, \text{ as suggested by Espeholt et al. 2018 to work best.} \]

\[ ^4 \text{a possible solution to this issue is elaborated on in Section 8.} \]
Table 2: Navigation Accuracy on the Development set

<table>
<thead>
<tr>
<th></th>
<th>DQN</th>
<th>DocQN</th>
<th>IQAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>27.2%</td>
<td>30.4%</td>
<td>33.4%</td>
</tr>
<tr>
<td>FAO1</td>
<td>96.2%</td>
<td>85.8%</td>
<td>93.6%</td>
</tr>
<tr>
<td>FAO2</td>
<td>16%</td>
<td>27.5%</td>
<td>29.9%</td>
</tr>
<tr>
<td>FAO3</td>
<td>0.2%</td>
<td>2.9%</td>
<td>4.1%</td>
</tr>
</tbody>
</table>

Bias, we switched to a mechanism where the learner sends a signal to all the actors to begin collecting summaries and pauses. Once all the actors are finished collecting summaries, the learner resumes.

6.3 Performance

6.3.1 Experiment 1

In this experiment we use IQAN with 8 actors and the $R_1$ reward function, and compare it against DQN and DocQN.

**Analysis**

Using IQAN we get a speedup of 5x against DocQN and use 15x less transitions then DocQN. We conclude that this is since the Deep-Q-Network algorithm (Volodymyr Mnih, Kavukcuoglu, et al. (2015)) that DocQN is based on is in essence off-policy, since during a single step of the DocQN, the algorithm either allows the agent to take an action or to sample an episode using tree distribution methods. Either way, DocQN then performs a parameter update, this means that the proportion of time the agent acts relative to the time the network is being updated is very low. Also note that IQAN manages to keep performance high on FAO1 while improving on FAO2 and FAO3, in contrast, DQN achieves 96.2% on FAO1 while performing poorly on the rest of the dataset. IQAN also explores more frequently with an average path length of 19. Finally, we improve by 3% against the DocQN model and by 6.2% against DQN.

**Notes**

(a) IQAN has a constant batch size of 16 trajectories, each can contain up to 16 transition, which effectively means it has a variable transition batch size between [16, 256], where DocQN used a constant 32 transition batch size. (b) During our testing we found that $R_1$ didn’t always converge. This is part of the reason we used $R_2$ in Experiment 2, since the signals from the sentence tokens resulted in a higher convergences rate. (c) The maximum episode length $T_{max}$ for IQAN was 16 and 30 for DocQN. (d) Since IQAN here is on-policy, this effectively means that V-trace is reduced to n-step TD(0). (e) For DocQN and DQN the runtime was estimated.

Figure 3: Experiment 1 - Accuracy on the Development set for IQAN and DocQN.

<table>
<thead>
<tr>
<th></th>
<th>DQN</th>
<th>DocQN</th>
<th>IQAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg. Path length</td>
<td>7.7</td>
<td>15.2</td>
<td>19</td>
</tr>
<tr>
<td>range</td>
<td>1.36</td>
<td>3.100</td>
<td>4.100</td>
</tr>
<tr>
<td>avg. Line distance</td>
<td>5.51</td>
<td>5.23</td>
<td>5.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stopping node distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
</tr>
<tr>
<td>headline</td>
</tr>
<tr>
<td>paragraph</td>
</tr>
<tr>
<td>sentence</td>
</tr>
</tbody>
</table>

Table 3: Navigation metrics on the Development set

6.3.2 Experiment 2

In this experiment, we compare the effect a variable number of actors has on IMPALA. We noticed that in a constant amount of time more actors translate into faster convergence, though like most concurrent applications the speedup is not linear in the amount of actors. Note: the $R_2$ reward function was used.

**Analysis**

When we start to converge towards FAO2, our performance in FAO1 decreases, as well as a consistent 100% on FAO1, and a consistent 0% on FAO3, this suggests a training schedule will be beneficial. Q-difference which is defined as $	ext{avg}_{a' \in A} Q(s, a') - Q(s, a)$, can be interpreted as the uncertainty the model exhibits with respect to taking action $a$.
Figure 4: Experiment 2 - Each run was allotted 4 days. Note: We used exponential smoothing with a factor of 0.8 since the data was very noise. The original data is in the appendix.

given state $s$. For example, 8-actors exhibits two noticeable increases, we notice the second one at around 80k, combined with the increase in the final line number average, this change is followed by a stabilization of episode length, and by a rise in performance in FAO2. For 4-actors, we can see that at 120k there is also an increase in the final line number average, this suggests that the 4-actors run was also starting to increase performance w.r.t FAO2. We also notice that the model learns to quickly locate the node containing a golden answer, but uses all allotted steps for an episode. Given that, it does slowly learns how to use less steps to reach the answer as performance does not degrade significantly until the FAO2 performance spike. Each run was comprised of 50% CPU actors with a policy-lag of 20, i.e we copy the network parameters from the learner every 20 update steps, the other 50% were on-policy actors that used the learner policy.

Note that in Experiment 2, due to memory constraints, we had to use a batch size of 8 since we found we cannot use a batch of size 16 as in Experiment 1. To compensate, we increased $T_{max}$ to 30. Unfortunately, Since Experiment 1 had only on-policy actors and Experiment 2 included both off-policy and on-policy actors, this means that we can’t compare the two IQAN reward functions or the effects of the V-trace target, since there are too many different parameters between the experiments. In the future we would like to evaluate $R_1$ and $R_2$ against each other, and evaluate the effects of V-trace as we increase policy-lag.

Figure 5: Experiment 2 - Navigation statistics

7 DISCUSSION

The unique aspect of this environment set for the RC problem, is that while in the inference stage, other models need to linearly scan the entire document, this setting allows us in a best case scenario to scan the document while consuming logarithmically less tokens.

We attribute our fast convergence to the high throughput we are able to achieve due to the IMPALA architecture and the on-policy nature of our model, and despite the sparse reward function which further demonstrates the impact the IMPALA architecture has on training.

Although most of the work done in this project is can be classified as mostly software engineering, we believe this work sets a strong foundation for this type of research to grow.

8 FUTURE WORK

We can describe a few directions this line of research can go:

1. Incorporating previously visited states

We might want to increase the expressiveness of the model by adding a sort of history for the navigation via an LSTM. The model can learn to recognize when it “went” in a circle, this will allow it to learn how to avoid them much more quickly. Also, the model could infer relationships about title tokens, for example in wikipedia, most countries share similar section names such as geography, economy, demographics. This will allow the model to learn the
structure of similar documents that may share the same format. We could also incorporate additional actions to allow the model to return to a previously visited node instead of preforming a circle.

Training wise:

2. Pre-training

use pre-training for the ANSWER action - in our current model, we give a reward of 0 of using the answer action, we think this causes the part of the network to under-train. A trained model that is forced to use the answer action in regular intervals will be able to train those parts of the network much better. Using these weights in a new network will allow us to train a new model much faster.

3. Curriculum Learning

Since we cannot converge towards FAO3 without passing through FAO2 and FAO1, Curriculum learning (Bengio et al. 2009) is a perfect fit for our needs. We can use the FAO partition to create a training scheme where we first train the model with examples where the FAO is low, then begin increasing the difficulty. This approach will allow us to slowly require the model to generalize better.

References


Appendix A

Figure 6: The network used.

Figure 7: Navigation example using DocQN

Figure 8: Figure 4 - original data