Answering Questions Using Wikipedia: A Deep Reinforcement Learning Adventure

Victor Schmidt

Supervised by

Dr. Sebastian Riedel

Dr. Pontus Stenetorp

Johannes Welbl

This report is submitted as part requirement for the

MSc Degree in Machine Learning,

University College London.

September 5, 2017

Abstract

Question Answering is one of the most researched areas of Natural Language Processing. Current methods often rely on finding the answer within a provided context. But what if the answer is not there? On a subset of the Wikireading Dataset, we explore how Reinforcement Learning can be used to answer questions. We trained a Deep Reinforcement Relevance Network using Double-DQN with prioritized experience replay to explore improvements on a state of the art extractive model: FastQA. As these models depend exclusively on the context they are given, using a Reinforcement Learning approach to query new documents from the dataset allows the Agent to better answer modified Wikireading questions than FastQA on its own. The Agent does not answer questions itself, rather it uses two FastQA models (trained respectively on SQuAD and a custom subset of Wikireading) to propose candidate answers and it decides whether or not any of these is the true answer. If none of these seems to be the sought answer, the Agent can query the Environment for another Article providing a new context to the FastQA models, which can in turn propose new candidates. We show that such an Agent can learn how to provide more context to the extractive models, but also learn to circumvent the reward design to postpone failure. We explore various Agents, Learning Algorithms and Environment designs to try and improve on FastQA before suggesting further work to actually perform it.

Acknowledgements

I want to thank Dr. Riedel for accepting the project's idea and guiding me through its realization. A special gratitude to Dr. Pontus Stenetrop for the numerous meetings and for being an endless source of ideas. I want to express my warmest acknowledgement to Johannes Welbl, who has given me a lot of his time, suggestions and without whom this project would have been half of what it is.
Thank you to the Machine Reading group for the amazing course on Statistical Natural Language Processing and to my fellow students for their availability, support and cheerfulness.
Lastly, thank you to Dr. Khadir and Jonas Vetterle for helping me through the writing of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introductory Material

1.1 Motivation

Text is one of the earliest forms of knowledge sharing. As such, mankind has produced an enormous quantity of it and continues to do so. As a source of information, a natural problem is to find a given piece of information in a text, be it a book, a research article or an entire corpus as Wikipedia.
Natural Language Processing (NLP) is a field at the crossroads of Mathematics, Linguistics and Computer Science which focuses on trying to automate the understanding of the Human languages. Its applications are extremely wide, from automating translation [51] or finding patterns across languages [28] to word prediction [2] or document classification [55].
One of these tasks is to answer questions given a training corpus. This means that the NLP algorithm is asked to find a way to answer questions given only the information contained in a pre-defined knowledge base. This can be useful for a lot of applications as better Internet results (as Google's Knowledge Graph [47]), educational conversational agents [8] and modern personal assistants as Apple's Siri and Microsoft's Cortona [22]. Developing systems which can answer questions is also a way to work towards understanding human reasoning as a great variety of tasks can be formulated in a Question-Answer framework [53]. In this context we will focus on exploring the use of Reinforcement Learning techniques to answer questions using Wikipedia.

1.2 Question Answering: a review

Within the NLP field, Question Answering (QA) is part of a narrower family of tasks: Information Retrieval (IR). IR systems (IRS) seek to give users the specific content they want through *queries*, be it a human-formulated question or a convoluted SQL query.

1.2.1 Heuristics

These systems may be very simple, depending on the nature of the task: string matching can be a simple heuristic to retrieve information. For instance using a date-specific regular expression along with looking for the word ”When” can be a very efficient way to find dates. Table 1.1 shows an example from [52] illustrating this idea: the answer can easily be found by matching the words in the context with those in the question and by identifying the date-like numbers.
The algorithms used to find the information and then pass it along to the IR system are called Information Extraction (IE) algorithms. Their goal is to exploit the current *context* to find the relevant pieces of information the IR system needs. They analyze unstructured data as text whereas the IRS itself *queries* structured data.

Question: *When did building activity occur on St. Kazimierz Church?*

Context: Building activity occurred in numerous noble palaces and churches [...]. One of the best examples [..] are Krasinski Palace (1677-1683), Wilanow Palace (1677-1696) and St. Kazimierz Church (1688-1692).

Answer: 1688-1692

Table 1.1: Example of a QA setting which can be solved with simple heuristics

1.2.2 Parsing

Simple heuristics as the one described above do not take into account the order and role of words in sentences. An alternative approach is therefore to parse sentences: by analyzing the dependency between words and creating the associated graph, one

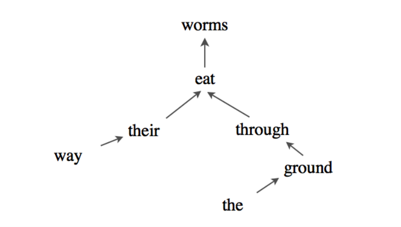
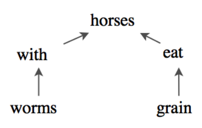
Question	**Answer 1**	**Answer 2**
What do worms eat?	*Worms eat grass.*	*Grass is eaten by worms*
What → eat → worms	grass → eat → worms	grass → eat → worms
Answer 3	**Answer 4**	**Answer 5**
Birds eat worms.	*Worms eat their way through the ground*	Horses with worms eat grain.
worms → eat → bird		

Figure 1.1: Dependency graphs for candidate answers from [31]. Answers 1 and 2 have a complete dependency agreement, 4 and 5 only partial and 3 has no agreement.

can hope to infer if a sentence answers a question. For instance (from [46]), considering the question ”*What do worms eat?*” and candidate answers, we can sort out answers by comparing dependency graphs as in figure 1.1: only those with a matching graph are kept [31], which are answers 1 and 2. Similarly, [**?**] parsed English sentences to mathematical equations by first simplifying the text and annotating operators. They then used heuristics to break the original sentences into simpler sentences which were, as a final step, then mapped to equations.
These approaches can however only be used on simple tasks, in particular contexts. They fail to generalize and need a lot of human intervention as annotations.

1.2.3 Supervised Learning

Finding the answer to a particular question can often be more challenging than using heuristics which are not scalable to a great number of different situations. One can not hope to create heuristics encapsulating several hundred thousands instances nor hand-designed parsers dealing with numerous structures. This is when Supervised Learning comes into play: for a particular task, one can *train* an IE system (IES). Several datasets exist to train and compare such systems like the Stanford Question

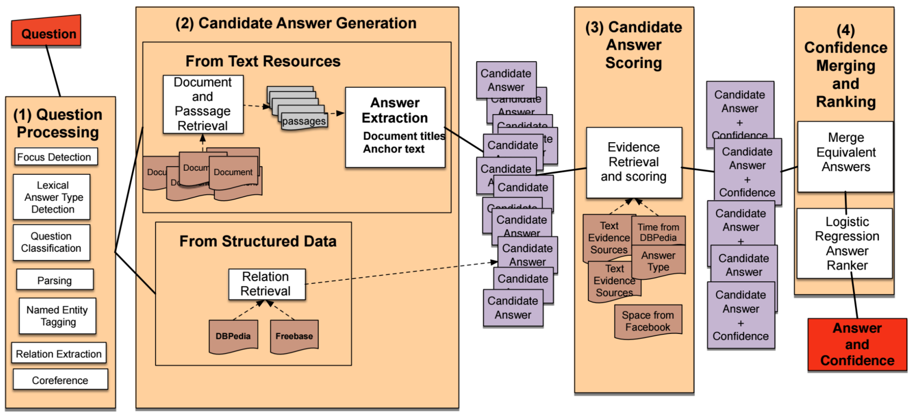


Figure 1.2: IMB's *Watson* procedure to answer a question, from [19]

Answering Dataset (SQuAD) [39] or Wikireading [13]. These datasets are designed to present the IES with a great number and variety of (Question, Context, Answer) tuples.

The Supervised Learning setting implies designing an IE model which *learns* from the dataset how to locate the answer to a question in a specific *context*. Even simple models as logistic regression perform relatively well: 51% F1 score on SQuAD [39]. A famous example of Question Answering system is IBM's *Watson* which won the *Jeopardy!* game in 2011. Its structure is detailed in figure 1.2.

On the SQuAD dataset, the current state of the art is achieved using complex structures as Bi-directional Recurrent Neural Networks (BRNN) which achieve 78.5% F1 score [52]. We will use this model as IES in our task. The model is called FastQA and computes an interaction between the representation it creates of the question and that of the current context, in order to infer the location of the answer. The basic idea is to create these representations with a BRNN, compute the "interaction" with a Fully-connected Neural Network (FNN) and output spans along with confidence scores. The spans represent the location of the potential answer, and the confidence score is the model's estimated probability of each span being the correct answer. We will investigate the model further in Chapter 3.

1.2.4 Reinforcement Learning

While the previously described models perform quite well (78.5% F1 score *vs* 86.8% for humans [39], [52]), they are, by design, incapable of answering question

Dataset	**Wikireading**	**Squad**
Example	**Context**: Angeles blancos is a Mexican telenovela produced by Carlos Sotomayor for Televisa in 1990. Jacqueline Andere, Rogelio Guerra and Alfonso Iturraldestar as the main [...] **Property**: original language of work **Answer**: Spanish	**Context**: In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity. The main forms [...] **Question**: What causes precipitation to fall? **Answer**: gravity
Details	Each instance is formed of a property for which we seek an answer and an associated document (Wikipedia article). There are 18.5 million instances and 884 different properties sought. Answers are not necessarily present in the document.	Given a context, several questions can be formulated but only one answer exists, and is always in the context. The dataset has 100,000+ instances.

Table 1.2: Datasets: SQuaD *vs* Wikireading

where the answer is not in the given context. For instance, inferring the gender of a person given their Wikipedia page is a simple task for humans: by the first name or the use of pronouns, it is straight-forward. So straight-forward that articles almost never specify the actual gender (*male* or *female*), crippling the previously described models (1.2.3) which need the answer to be actually stated.

This situation illustrates the need of external sources of information for the IR system to answer the end user. This can be done in several ways, depending on the task. If the information in the current context is only partial, multi-document extraction and event co-reference are of use [6].
If the query itself is the problem, a model can reformulate it in a way that is more understandable to the IES: [4] uses Reinforcement Learning (RL) to train a Ques-

tion reformulating agent. In a similar setting, [36] also trains an RL agent (with REINFORCE) to reformulate queries to a search engine, from "*What are the most promising directions to cure cancer and why?*" to "*cancer treatment state-of-the-art frontiers survey*" for instance.

If the information is present but written in a way which is hard for the IES to understand, [34] uses RL to look for other sources containing the same information, formulated in a different way. Table 1.3 from [34] shows how hard a formulation can be: for the IES to answer it has to count that *a couple and four children* makes 6 persons and that *murder-suicide* means that the latter subject of *suicide* is the actual killer sought. This approach is close to that of [36] with the difference that it uses prior information: [36] assumes open-domain whereas [34] uses pre-defined query templates.

Lastly, [21] uses RL in a resource-bounded environment, in a more structured setting, with the task of completing a knowledge base (MIT Faculty contact information) from the Web.

Questions	Who is the shooter?	How many people were killed?
Context	*A couple and four children found dead in their burning South Dakota home had been shot in an apparent murder-suicide, officials said Monday. [...] Scott Westerhuis's cause of death was "shotgun wound with manner of death as suspected suicide", it added in a statement.*	
Answer	6	Scott Westerhuis

Table 1.3: Situation when, while present, the information is unclear for an IES

1.3 Objectives

1.3.1 Outline

FastQA is an efficient model to answer questions and achieves the current state of the art performance on SQuAD. This dataset's focus is on the difficulty to find the answer to a question in the context. But the context always contains the answer, unlike in the Wikireading dataset. The goal of the present Thesis is to use Reinforcement Learning techniques to better answer questions with the FastQA model on a more challenging dataset where the answer may not be in the provided context: Wikireading.
Considering a subset of the Wikireading dataset, we will use a FastQA model alongside RL ideas from [21], [34] and [11] to create an Agent capable of improving on the sole FastQA model: it will be able to *decide* whether to trust the FastQA with its answer or to *query* another context for the former model to re-evaluate its decision.

More precisely the first goal is to design an Environment capable of *querying* an *adequate* subset of the Wikireading dataset. This means decisions have to be made regarding the kind of properties which can be interesting in our particular setup. Then the subset will have to be indexed to make it search-able.
Second, our key point of interest is that the answer may not be in the provided context. The Agent therefore has to find a way to know if the answer is there, and how to get a different document if not. It will have to use use insights from the FastQA model to *decide* whether or not the answer is present in the current article or if it has to be looked for in another one.
The final goal is to design a Learning Algorithm allowing the Agent to *improve* on the FastQA model. In other words, the Learning Algorithm should yield a behavior with greater reward for the Agent compared with the baseline policy of always *submitting* the FastQA's most likely answer.

1.3.2 Structure

In the following Chapter of the Thesis we will look into the theory used in this project, mainly in the perspective of using RL for NLP: we will investigate the foundations of Reinforcement Learning as Q-learning, how to use Deep Learning to approximate value functions and how to make the learning more potent and stable with Double Deep Q-Networks and Prioritized Experience Replay. As the FastQA model and the Agent utilize Deep Learning, we will see which improvements on Vanilla Fully-connected and Recurrent Neural Networks are implemented. Lastly we will delve into two major focuses of NLP: sentence representation and appropriate metrics.
In Chapter 3 we will focus on describing the task itself, its implementation and the Agent's design including the construction of the dataset, the Agent's inner structure and the Environment's scenario.
Lastly in Chapter 4 and 5, we will discuss the experiments run, analyze the results and criticize the choices made. We will see how using Reinforcement Learning could improve on the sole FastQA model for Question Answering if it were not for two major issues, and we will explore potential areas of amelioration for further work.

1.3.3 Vocabulary

Submit: Submit actions are those ending an episode. By sending a *submit* action to the Environment, the Agent means that the action sent *is* the answer according to it.
Query: Query actions are those asking the Environment for another context. By acting this way, the Agent means it does not trust the answer to be in the provided context and asks the Environment to *query* the indexed database for another context. How the Environment handles such a query will be discussed.
FastQA: Refers to the model described in [52]. Such models are not trained during the agent's learning process, they are used with fixed pre-trained weights.
Score: Unless specified otherwise, the score is the F1 score between a given sentence and a target sentence (*c.f.* Section 2.3.2).
Similarity: Implicitly the *cosine* similarity between a given sentence and a target

sentence (*c.f.* Section 2.3.2).

Hop: Describes the presence or absence of the expected answer in the document. If an instance's answer to its question is not in its context, the instance is said to contain a hop.

RL Dataset: Our task's dataset. It is extracted from Wikireading according to the criteria in 3 and split into training, validation and test sets.

SFastQA: FastQA model trained on SQuAD

WFastQA: FastQA model trained on a custom subset of Wikireading (*c.f.* 3.1.2)

Chapter 2

Background Theory

In this chapter, we will focus on understanding the theory which drives our implementation of the IRS. The reader is assumed to have basic knowledge in Supervised Learning (notions as overfitting), Deep Learning (vanilla Feedforward Neural Networks), and optimization (Gradient Descent). As mentioned, we will use a Reinforcement Learning (RL) approach so this chapter's first interest will be on RL. We will see that the Agent's design involves function approximation with Feedforward Neural Networks (FNNs) whose implementation is discussed in the second part. We will then investigate Recurrent Neural Networks (RNNs) and Long Short Term Memory Networks (LSTMs)s as improvements on those before we tackle two important subjects of NLP: sentence representations and metrics. Lastly we will summarize how the FastQA model works as it is a major component of the project.

2.1 Reinforcement Learning

Most content in this section is derived from [49], [50], [29] and [25].

2.1.1 Optimal behavior

Reinforcement Learning is a topic which aims at modeling decision processes. In other words, RL algorithm (Agents) try to learn a mapping from situations to actions within an environment. The formal mathematical context we will use in this section is that of Markov Decision Processes (MDPs).

Given that the Agent is in state $S_t \in \mathscr{S}$ and has a set of possible actions $\mathscr{A}$, the goal is to learn how to select action $a_t \in \mathscr{A}$ to maximize the expected *return* $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ (discounted sum of rewards) the agent will get using its current *policy*, from the current state. The policy itself is the mapping $\pi(a_t|s_t)$ from states to actions. When the Agent is in state S_t and performs action a_t it receives two pieces of information from the Environment: a reward r_{t+1} and the new state it is in, S_{t+1}. If the state reached is terminal (say at time T, meaning the Environment does not allow for another transition), the series $t = 0 \ldots T$ is called an episode.

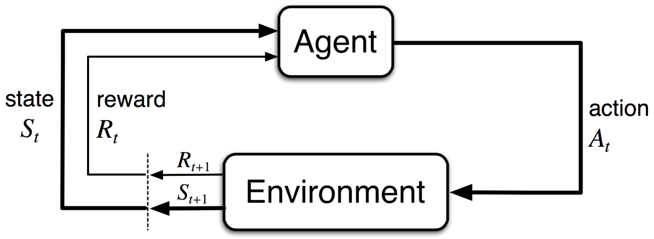


Figure 2.1: The agent - environment interaction in reinforcement learning from [49]

The expected return from state $S_t = s$ following policy π is called the *value function*: $V_\pi(s) = E[R_t|S_t = s]$. The MDP setting provides a probability transition table P which describes the probability of seeing state $S_{t+1} = s'$ after taking action a_t in state $S_t = s$. We can therefore recursively define V_π using the Bellman Expectation Equation for V_π:

$$V_\pi(s) = \sum_{a \in \mathscr{A}} \pi(s,a) \left(R_s^a + \gamma \sum_{s' \in \mathscr{S}} P_{ss'}^a V_\pi(s') \right) \tag{2.1}$$

One can also use the *action-value function* Q which computes the expected return of performing action a in state s. As V, Q can be defined recursively as defined in the Bellman Expectation Equation for Q_π:

$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in \mathscr{S}} P_{ss'}^a V_\pi(s') \tag{2.2}$$

$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in \mathscr{S}} P_{ss'}^a \sum_{a' \in \mathscr{A}} \pi(s',a') Q_\pi(s',a') \tag{2.3}$$

The *optimal* action value function is the maximum action value achievable by any policy for state s and action a: $Q_*(s,a) = \max_\pi Q_\pi(s,a)$, which also follows a Bellman Equation:

$$Q_*(s,a) = R_s^a + \gamma \sum_{s' \in \mathscr{S}} P_{ss'}^a \max_{a'} Q_*(s',a') \tag{2.4}$$

Given the action-value, one can derive a policy in various ways, two popular ones are:

$$\text{Epsilon-Greedy policy: } \pi(s) = \begin{cases} \text{argmax}_a\ Q(s,a) \text{ if } \varepsilon < \text{random}(0,1) \\ \text{random } a \in \mathscr{A} \text{ otherwise} \end{cases} \tag{2.5}$$

$$\text{Softmax policy: } \pi(s) \sim softmax(Q(s,a))_{a \in \mathscr{A}} \tag{2.6}$$

In our IRS setting, the goal will therefore be to learn the optimal Q function to be able to take actions: given the current information from the environment (state), should the Agent *submit* one of the FastQA's proposals or *query* a new context?

2.1.2 Deep Q-Networks

2.1.2.1 Q-Learning

As stated, the goal is to learn Q_* from interacting with the Environment. We however do not have access to it's Model P and can not solve directly (or iteratively) for Q in equation 2.4. A way to circumvent this issue is by using *Temporal Difference Learning* (TD-Learning).

TD-Learning is a way to learn the value function V directly from experience by updating the Agent's belief as it interacts with the Environment, at a learning rate of α:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s)) \tag{2.7}$$

More precisely, this update rule is called TD(0) as it only looks one step ahead; $r + \gamma V(s') - V(s)$ is called the TD-error. This error measures how wrong the current value function is given the knowledge of one transition's reward. Given this update rule, we can learn the Q value in a similar way and therefore learn a policy allowing the Agent to interact optimally with the Environment (using equations 2.5 or 2.6). Such an algorithm is called *Q-learning*.

Algorithm 1 Q-learning from [25]

```
Initialize $q$ arbitrarily, e.g., to 0 for all states
**for** each episode **do**
    Initialize state $s$ from Environment
    **while** state s is not terminal **do**
        $a \leftarrow$ action for $s$ derived by $Q$, e.g., $\varepsilon$-greedy
        take action $a$, observe $r$, $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$
        $s \leftarrow s'$
    **end while**
**end for**
```

2.1.2.2 Function Approximation

When $\mathscr{A} \times \mathscr{S}$ is too large, Q can not be stored in a tabular way as above, keeping track of each action-state pair's value. We therefore need to approximate Q, usually with a differentiable function, much like in the Supervised Learning setting in which one seeks to approximate the mapping from inputs to outputs. Let Q_θ be the approximating function, with parameters θ. The Q-learning algorithm's update rule is modified as follows:

$$Q_\theta(s,a) \leftarrow Q_\theta(s,a) + \alpha\Big(r + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)\Big)\nabla_\theta Q_\theta(s,a) \tag{2.8}$$

2.1.3 DQN

Deep Reinforcement Learning uses Neural Networks as function approximators. The learning process is however too unstable and two improvements to the Q-learning algorithm have to be made (from [30]):

1. *Replay buffers*: transitions are stored in buffers and learned as a sampled mini-batch instead of updating the Q-value online

2. *Target Network*: the TD-error is actually computed with a target network, which is updated from the actual one periodically. Having this delayed update allows for smaller over-estimation of the actual Q-value

This structure has been proven to be more stable and effective than earlier algorithms [30], [17] and is therefore the one we'll use to learn how to answer questions, though altered as described hereafter.

2.1.4 Improvements on DQN

Though more stable and potent than standard Q-learning with function approximation, the DQN algorithm can be improved further:

1. *Double Q-learning*: Van Hasselt *et. al* [50] introduced the concept of Double Q-learning and then applied it to DQN (Double DQN, D-DQN) . The idea is that using the same function approximator to both chose an action and compute its value tends to make the agent over-estimate the true action-value function. Using the online network to chose an action but the target network to evaluate it reduces this effect.

2. *Prioritized Experience Replay*: (D-)DQN uses experience replay buffers to train from past experience. A uniform sampling over those however neglects the fact that some of these transitions where more "instructive" than others, that is to say yielded smaller TD-Error. To solve for this, one can use Importance Sampling with weights directly linked to the transitions' TD-error [43].

2.2 Deep Learning

2.2.1 Fully Connected Networks

Even though the base unit of Neural Networks, the neuron, derives from the Perceptron algorithm from 1962 [42], their actual successes are recent and they are still being improved. In this section we will cover improvements on the vanilla FNNs as Deep RL relies (partly) on them:

1. *Dropout*: In order to limit overfitting, a proportion of each layer's neurons are de-activated (set to 0) so that neurons learn high level features and focus less on localities as they can't depend on their neighbors [14], [35].

2. *L2 regularization*: a second option to avoid overfitting is to modify the cost function by adding a penalty proportional to the weights' L2 norms. Large weights are more likely to be specific to a dataset and more costly at test-time if the "complex" hypothesis is wrong [35]. Alternatively, in a Bayesian maximum a posteriori view, the same effect can be achieved with a zero mean Gaussian prior over weights [7]: multiplying the expected cost of prediction by a prior that weights' norms are Gaussians with 0 mean and λ standard deviation is equivalent to adding the regularization term with a scaling factor of λ.

3. *Depth*: As a general heuristic depth allows for approximation of more com-

Algorithm 2 Double DQN with Prioritized Experience Replay, from [25]

```
1: Input: minibatch $k$, step-size $\eta$, replay period $K$ and size $N$, exponents $\alpha$ and
   $\beta$, budget $T$.
2: Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
3: Observe $S_0$ and choose $A_0 \sim \pi_\theta(S_0)$
4: for $t = 1$ to $T$ do
5:   Observe $S_t, R_t, \gamma_t$
6:   Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t =$
     $\max_{i<t} p_i$
7:   if $t \equiv 0 \bmod K$ then
8:     for $j = 1$ to $k$ do
9:       Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
10:      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
11:      Compute TD-error:
12:      $\delta_j = R_j + \gamma_j Q_{target}(S_j, argmax_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
13:      Update transition priority $p_j \leftarrow |\delta_j|$
14:      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
15:    end for
16:    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
17:    From time to time copy weights into target network $\theta_{target} \leftarrow \theta$
18:  end if
19:  Choose action $A_t \sim \pi_\theta(S_t)$
20: end for
```

plex functions (*wrt.* width). This is supported by [37] where they show such a result for piecewise-linear activation functions in FNNs.

4. *SELU*: the Scaled Exponential Linear Units induce self-normalizing properties in FNNs [24]. This is extremely useful for learning capacity as normalizing speeds up training, reduces overfitting and stabilizes learning [18] but SELUs are less computationally expensive than batch-normalization.

2.2.2 Recurrent Neural Networks

Feedforward Neural Networks are good at understanding underlying structures in static data. However they have a harder time inferring time dependencies such as two following video frames, time series, texts as sequences of words or characters. In this section we will investigate RNNs and the structure of FastQA.

RNNs use iterative function loops to store information between time steps of sequences. To put it another way, RNNs take the current data point as well as their previous output as input for the current time-step. The following figure shows an RNN unrolled in time to illustrate this.

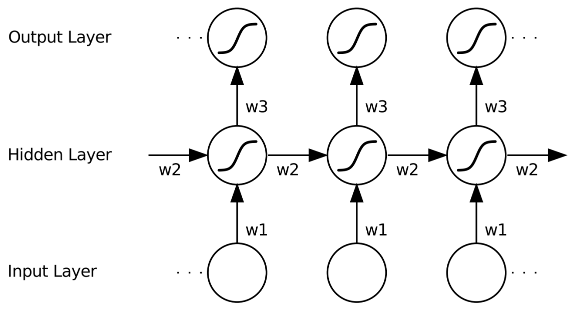


Figure 2.2: Illustration of an RNN unrolled in time, from [9].

An RNN's forward pass is almost the same as a feed-forward's: it computes a weighted sum of inputs and applies a non-linearity to it. Let us consider a sequence $X = \{x^1, \dots x^T\}$. Each x^t is n-dimensional and its coordinates are noted x_i^t. Let us denote $W = (w_{ih})_{i=1..n}^{h=1..m}$ the weight matrix of the RNN with m units. Lastly let s_h^t and a_h^t be the weighted sum of inputs and activation for the unit h at time-step t using the differentiable non-linearity σ:

$$s_h^t = \sum_{i=1}^{n} w_{ih} x_i^t + \sum_{j=1}^{m} w_{jh} a_h^{t-1} \tag{2.9}$$

$$a_h^t = \sigma(s_h^t) \tag{2.10}$$

For a classification task with C classes, then the output layer's c^{th} activation is

$$s_c^t = \sum_{i=1}^{m} w_{ic}^{output} a_i^t \tag{2.11}$$

$$a_c^t = \frac{exp(s_c^t)}{\sum_{i=1}^{C} exp(s_i^t)} \tag{2.12}$$

In this particular case the network outputs the following probabilities at the end of the sequence:

$$p(X) = p(x^1, \ldots, x^T) = p(x^1) \prod_{i=2}^{T} p(x^i | x^{i-1}, \ldots, x^1) \tag{2.13}$$

RNNs are trained in a similar way as FNNs, called Back Propagation Through Time. BPTT acts as the original Backpropagation algorithm, as if the RNN was unrolled through time forming a multi-layer FNN [48].

2.2.2.1 LSTMs

However vanilla RNNs are hard to train as the gradient often either vanishes or explodes [45] which makes them hard to use in practice [15]. The gradient issue is partially solved by the Long Short Term memory recurrent unit [16].

Using a vector notation, here is how the LSTM works: its output is h_t, f_t is called the forget gate, i_t the input gate, o_t the output gate, $\widetilde{c_t}$ is the candidate memory and c_t is the memory passed to the next time step:

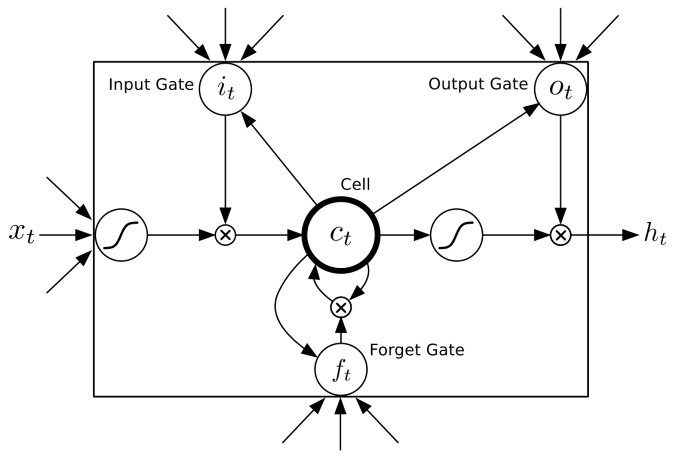


Figure 2.3: Illustration of the LSTM cell. For clarity, the links from h_{t-1} to the gates are omitted

$$f_t = \sigma\ (W_{fx}x_t + W_{fh}h_{t-1} + b_f) \tag{2.14}$$

$$i_t = \sigma\ (W_{ix}x_t + W_{ih}h_{t-1} + b_i) \tag{2.15}$$

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \tag{2.16}$$

$$\widetilde{c_t} = tanh(W_{jx}x_t + W_{jh}h_{t-1} + b_j) \tag{2.17}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c_t} \tag{2.18}$$

$$h_t = o_t \circ tanh(c_t) \tag{2.19}$$

The idea is that $\widetilde{c_t}$ is computed from the previous timestep's output and current input. Given this candidate memory for the LSTM, f_t and i_t control how much of the previous memory is kept and how much of the current candidate is added. Then o_t controls how much of the current memory is output by the layer to the next one.

This gating mechanism allows for a better memory, and prevents the gradient from vanishing (too quickly). To solve the exploding gradient issue, a simple heuristic is to clip its values [10].

2.3 Natural Language Processing

2.3.1 Sentence representation

To apply Machine Learning techniques and more especially Deep Learning techniques to Natural Language Processing, one needs to find a way to mathematically represent semantic objects as characters, words and sentences.

The most common way to vectorize words is to use pretrained embedding matrices as GloVe [38] or Word2vec [57]. These matrices associate known words (called *vocabulary*) to n dimensional vectors (usually n ranges between 50 and 300). They are trained in an unsupervised way, bringing words with a similar meaning closer in terms of cosine similarity (see section 2.3.2). This means that for instance the vector $v = v_{Paris} - v_{France} + v_{Italy}$ is close to the vector v_{Rome} illustrating that the training algorithms capture the underlying concept of capital.

Representing sentences however is a more tricky procedure. There are three main ways of doing so:

1. Compute an algebraic vector representation of the sentence from the individual word or character vectors as in [52]. For instance, a simple algebraic operation is to take the mean of the sentence's words' embedding vectors. This is called Continuous Bag of Words (CBoW, [1]); the mean is often preferable to the sum as it is not as sensitive to the sentence's length [5].

2. Run an RNN (possibly in both directions) across the sentence and compute the representation from the RNN's outputs (algebraically or with Attention for instance) [26], [54]

3. Run a Convolutional Neural Network across the sentence [56], [20]

In our task we will use a representation of the first form introduced by [5]. It is an improvement on the Continuous Bag of Words as described by [1] and will be presented in details in Section 3.3.2.

2.3.2 Metrics

Using appropriate metrics to evaluate a model is paramount. Using the wrong metric one could be misled into thinking they have achieved (or failed) a task: predicting always *True* in an unbalanced binary classification task could yield 90% accuracy if the dataset itself were made of 90% positive examples. It would not however prove any learning.
In this section we will focus on two metrics: *F1 score* and *Cosine Similarity*.

In Natural Language, *meaning* does not depend on the *exact* presence of all expected words, rather on the presence of at least some of these. This is the idea of the *F1 score*: computing a number between 0 and 1 illustrating how a given set is similar to an other, in terms of their elements. More precisely the $F1$ score is computed as follows:

$$Precision = \frac{T_p}{T_p + F_p} \tag{2.20}$$

$$Recall = \frac{T_p}{T_p + F_n} \tag{2.21}$$

$$F_1\ score = \frac{2\ Precision * Recall}{Precision + Recall} \tag{2.22}$$

Precision and Recall allow us to assess how much we can trust the model when it predicts the presence of certain words (Precision) and how much each word in the target sentence is correctly predicted (Recall). The F_1 score being the harmonic mean of the two, it aggregates these two concurrent metrics into a final score that is their weighted average. In our case we care as much about Precision as Recall. Otherwise one could have used the F_β score to weigh them differently [40].

The cosine similarity on the other hand is a purely mathematical concept and therefore depends on the word representation we use. It measures the *cosine* of the angle θ between two vectors u and v:

$$\cos(\theta) = \frac{\overrightarrow{u} \cdot \overrightarrow{v}}{\|\overrightarrow{u}\|_2 \cdot \|\overrightarrow{v}\|_2} = \frac{\sum_i u_i \cdot v_i}{\sqrt{\sum_i u_i^2} \cdot \sqrt{\sum_i v_i^2}} \tag{2.23}$$

To compute the cosine similarity between two sentences we will use the following sentence vector representation: counting the occurrences of words (u_i and v_i respectively) in each sentence, setting u_i (resp. v_i) to 0 if not present in the sentence v (resp u).
Once more we can see how important the choice of the metric is: in our context we can not use a cosine similarity based on the Matrix Embedding representation of words as for instance predicting *Rome* instead of *Paris* would yield a close cosine similarity (by construction) whereas it is a mistake in terms of *meaning*.

2.3.3 FastQA

The FastQA model uses ideas from sections 2.2 and 2.3. It is an *Extractive model* and as such its goal is to locate an answer to a question in a context.
To do so, it starts by *embedding* the words both in the question (which is a string we will call Q) and in the context (also a string, called X). Words in Q and in X are embedded independently, concatenating their vector representation from GloVe and from a custom-trained CNN (as in [44]). This embedding procedure results in the vectorized sequences $\widetilde{Q}$ and $\widetilde{X}$.
These are then encoded using a bidirectional LSTM network (two LSTM networks, going through the sequence in opposite directions) into $\widetilde{z}$ and H.
Finally the *Answer Layer* consists of an FNN and a Beam Search on possible locations. This final layer outputs spans (start and end indexes of the answer in the context) along with their probabilities.
In figure 2.4 from [52], the question is ”What is the term for a council of Mongol chiefs”, the context is ”[...] at a Khuruldai, a council of Mongol chiefs, [...]”, and the answer is ”Khuruldai”. We can see how this model depends exclusively on the context it is given and can only answer questions where the answer *is* written in this very context.

We will therefore build an RL Agent able to use this model's *extractive* capabilities and circumvent its weakness by querying *more* context from the dataset.

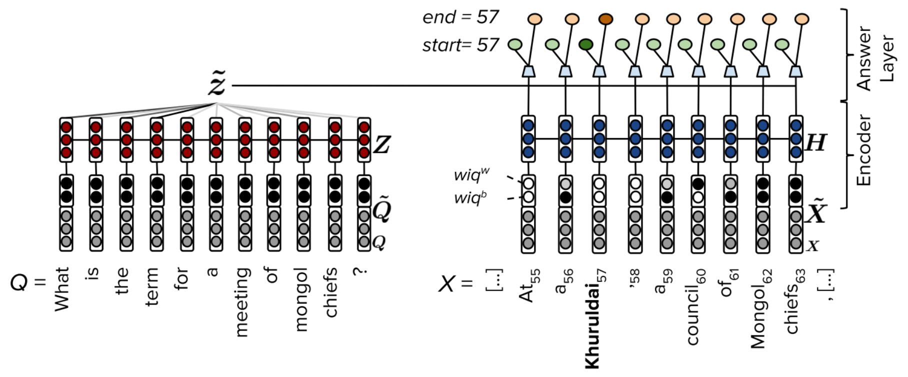

end = 57
start= 57
$\tilde{z}$
Z
$\tilde{Q}$
Q
H
$\tilde{X}$
x
wiq^w
wiq^b
Answer Layer
Encoder
Q = What is the term for a meeting of mongol chiefs ?
X = [...] At55 a56 Khuruldai57 ,58 a59 council60 of61 Mongol62 chiefs63 , [...]

Figure 2.4: Illustration of FastQA system on example question from SQuAD. wiq^w and wiq^b are features part of the word representations. See [52] for more details.

Chapter 3

Answering Questions

In this section we will delve into the specifics of the task's implementation. Firstly we will look at how the dataset has been designed. Secondly we will describe the Environment, its scenario and the type of information it gives to the Agent. Lastly we will focus on understanding the Agent's functional structure and how it computes the approximated Q-values.

3.1 Database Description

3.1.1 The Wikireading Dataset

As mentioned in Section 1.2.3 the Wikireading dataset is composed of more than 18 million instances, which have the following main fields[1]:

- *answer_location*: Word indexes in the document where any one token in the answer was found
- *answer_string_sequence*: list of words (as strings) in the answer
- *question_string_sequence*: list of words (as strings) in the question
- *string_sequence*: list of words (as strings) in the document (*i.e.* in the context)
- *key*: http link to the original Wikipedia article from which the context is extracted

[1]More details at: *github.com/google-research-datasets/wiki-reading*

The *question_string_sequence* can be any one of the 884 different kinds of properties sought for from the context. The first task regarding the dataset is to select a suitable subset compatible with our goal. Indeed this dataset is too large to be handled as is and properly indexed. We therefore focused on the following 9 properties (with sample answers):

- instance of: taxon, human, legislation, railway system...
- sex or gender: male, female, transgender male, transgender female
- country: France, Germany, United States of America, England...
- date of birth: 16 November 1988, 1893, August 1969, ...
- sport: taekwondo, auto racing, basketball, association football...
- color: color, black-and-white
- chromosome: chromosome 1, chromosome 2...
- melting point: +113, +198, +203 +205 (as *one* answer, both numbers)
- conferred by: Columbia University, Heinrich Himmler, PEN Oakland ...

The criteria used to select these categories were:

1. **Cardinality**: Number of instances in the category. We want to have a sufficient number of instances to learn from, and some variety in cardinalities to compare performance and see if the learned features can be transferred to low-cardinal categories.

2. **Proportion of question with hops**: we have to ensure that the dataset has various hop proportions as our very goal is to learn how to deal with these. Knowing whether the questions with hops are answerable at all (meaning the answer is originally present in Wikipedia) is still an open question. Questions with a *mentioned* answer are a subset of the questions *with hops*.

3. **Entropy of answers**: some questions have very few possible answers (as *sex or gender*) when others have a very large set of possible answers (as *date of birth*)

The entropy is computed as per [13]: it is the entropy $\hat{H}(c)$ of the answer distribution A in the dataset for a given category c, normalized to $[0,1]$ by dividing by the category's entropy under the uniform model [2]:

$$p_c^A(a) = \frac{count(a)}{\sum_{a \in A_c} count(a)} \tag{3.1}$$

$$H(A_c) = \sum_{a \in A_c} p_c^A(a) \cdot \log(p_c^A(a)) \tag{3.2}$$

$$\hat{H}(c) = \frac{H(A_c)}{\log |A_c|} \tag{3.3}$$

This way, an entropy value close to 0 means that the number of unique answers is very low compared to the total number of answers in this category. On the other hand, an entropy close to 1 means that every answer is different within the category. Table 3.1 summarizes these properties per category to illustrate the choice.

The Wikireading dataset is itself split into a training, a validation and a test set. Our final dataset, which we will refer to as the **RL Dataset**, is also split into a training, a validation and a test set, with respectively 250,000, 150,000 and 150,000 instances in the previously mentioned categories, with approximately the same properties as the original dataset. Each set in the RL Dataset comes from is counterpart in the original Wikireading Dataset.

3.1.2 FastQA and Wikireading

The original formulation of instances is not plainly compatible with the Question-Context format the FastQA model expects. As it finds similarities between the Question and the Context we need to adapt the way the former are formulated so that they are more *natural* and contain the subject of the category sought.

[2] The $-$ sign in the entropy formulation has not been forgotten: this definition of $\hat{H}(c)$ is equivalent, normalizing with $\log \frac{1}{|A_c|}$

Property	Cardinality	Hops			Entropy	Reason of choice
Category		**Ratio**	**With Hops**	**Without Hops**		
instance of	3 245 856	0.533	1 728 563	1 517 293	0.431	Cardinality, balanced hops, low entropy
sex or gender	1 143 126	0.966	1 104 711	38 415	0.189	Cardinality, Almost always with hops, very low entropy
country	986 587	0.161	159 142	827 445	0.536	Cardinality, very often no hop, moderate entropy
date of birth	953 481	0.049	46 414	907 067	0.936	Cardinality, almost never with a hop, very high entropy
sport	303 614	0.354	107 492	196 122	0.483	Cardinality, almost balanced hops, moderate entropy
color	9 821	0.589	5 783	4 038	0.292	Rare, almost balanced hops, low entropy
chromosome	9 519	0.779	7 414	2 105	0.805	Rare, mostly with hops, high entropy
melting point	420	0.983	413	7	0.984	Very Rare, almost only with hops, very high entropy
conferred by	427	0.016	7	420	0.958	Very Rare, almost only without hops, very high entropy

Table 3.1: Summary of each selected category's properties in the original Wikireading dataset

Key	Extracted Title	Category	Question	Answer
https://en.wikipedia.org/w/index.php?title=Jacques-Henri_Bernardin_de_Saint-Pierre&oldid=703750472	Jacques-Henri Bernardin de Saint-Pierre	sex or gender	What is the sex or gender of Jacques-Henri Bernardin de Saint-Pierre ?	Male
https://en.wikipedia.org/w/index.php?title=Center-South&oldid=691474932	Center South	country	What is the country of Center South ?	Russia
https://en.wikipedia.org/w/index.php?title=Image3D&oldid=691154592	Image3D	intstance of	What is Image3D an instance of ?	Business Enterprise

Table 3.2: Sample question reformulations from the Wikireading dataset

To achieve this, the title of the original Wikipedia article is extracted from the *key* and a proper question is formulated from the the title and the category as illustrated in table 3.2. What is more, the FastQA model goes over the whole context, which can reach more than 25,000 words in the Wikireading dataset. We therefore truncated contexts to the first 1,000 words for computational reasons.

As mentioned, the Agent will use pre-trained FastQA models to answer questions. We will use two pre-trained models: one trained on SQuAD and the other on Wikireading's training set. To achieve so we need to isolate answers *without* hops (*i.e.* the answer is somewhere in the context) and with a *complete* span. A span is said to be complete if it is a list of consecutive integers. The selection criteria for these instances from the original Wikireading dataset are therefore:

- *answer_location* is not empty
- *answer_location* contains consecutive indexes (there can not be a word not part of the answer in the span)
- all words in the previously defined spans are part of the actual answer (see 3.1.3 for why this is needed)
- the answer is fully present in at least one of the previously defined span
- the end index can not be larger than 999 (contexts are truncated to 1000)
- if there are multiple spans, select the one with smallest start index

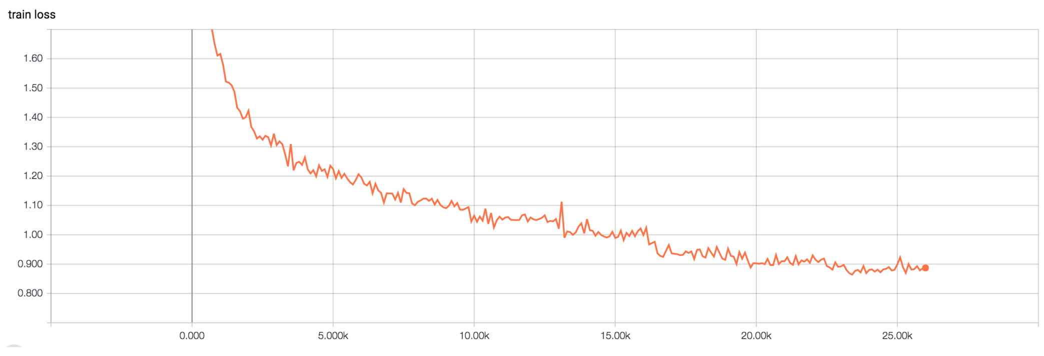
The resulting FastQA-compatible Wikireading training set contains 400,000 instances. The associated validation set contains 200,000 instances. The FastQA model trained on Wikireading (**WFastQA**) was trained using the same hyper-parameters as the FastQA model trained on SQuAD (**SFastQA**) and achieves a final F1 score of 71.2% on the validation set. As per [52], SFastQA achieves a 78.5% F1 score on the SQuAD validation set. We will use both these models to propose answers to the Agent.

3.1.3 Dataset Challenges

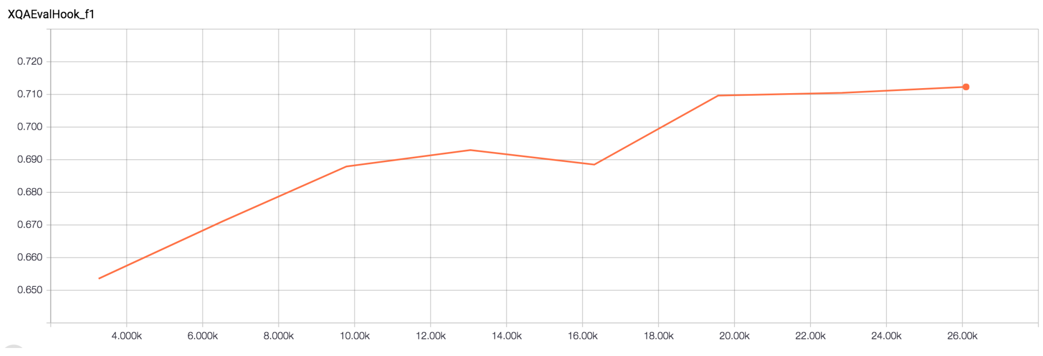
The original Wikireading Dataset presents two main challenges we have to deal with to reach an improvement on existing Question Answering techniques.
First, the *answer_location* field in the instances is extremely inconsistent. To see the kind of issues we have to face let us focus on the example in Table 3.3. We can see that even though every word in the answer is present in the *answer_location*, *(i)* it is not present in the form of a span, *(ii)* it includes a very common preposition (*of*) and *(iii)* even a word which is not in the true answer (*French*). We therefore can not expect an extractive model as FastQA to be able to properly answer such

an instance. It does not however mean that there isn't another article in the dataset mentioning the true answer, this is why such examples are not discarded, but this is a situation which is to be avoided during the training of WFastQA.

Second, when the answer is not in the document (*answer_location* is empty), there is no mention of where it actually is. There is no way to know in what document to look for the answer (*c.f.* Table 3.4). This means that when sampling the RL Dataset from the original dataset we have no guarantee that there *is* a way for the Agent to find the answer, as it may not at all be there. Some instances may therefore be unanswerable by design without external knowledge (such strategies are reviewed in Section 1.2.4). In absence of such an indication, we can not even measure the degree to which this is a problem. We will see in section 4 that while it does not affect the proof of concept, it has an important impact on the strategies learned by the Agents. Even if we did index the whole dataset, some instances would still be unanswerable as there are 555 words in the vocabulary of the answers which are not



(a) Train Loss



(b) Validation F1 score

Figure 3.1: WFastQA's training curves.

in the vocabulary of the articles: extractive models cannot be expected to answer questions with words which do not appear in contexts.

Question	What is Saint-Gonlay an instance of ?
Context	Saint-Gonlay (Breton: Sant-Gonlei) is a commune in the department of Ille-et-Vilaine in Brittany in the northwest of France. [...]
Answer Location	[12, 16, 27, 28, 74, 83, 90, 93, 101, 156, 164, 168, 176, 191, ...]
Answer Words	commune, of, of, France, of, of, of, of, of, of, French, of * 27
True Answer	Commune of France

Table 3.3: Example instance from Wikireading where the answer location is misleading

Question	When was David Womark born ?
Context	David Womark is an American film producer. With Ang Lee and Gil Netter, he was nominated for an Academy Award for his work on Life of Pi.
Answer Location	[]
True Answer	2000

Table 3.4: This instance's context is not shortened: it is the orginial one. It is also the only one mentioning *David Womark*. We can see how inferring the actual document containing the answer is not possible when creating the dataset.

3.2 Environment Design

In this section we will go over the various pieces of information the Agent receives from the Environment during an episode: State, Actions and Reward.

3.2.1 Indexed Database

The RL dataset is indexed into an Indexed Database (IDB) to make it search-able[3]. Along with the original RL Dataset/Wikireading fields, the IDB contains an *uuid* field. Universally Unique Identifiers (uuids) are randomly-generated 128-bits numbers used to identify objects. They are so random that one has to generate $1.03 \cdot 10^{14}$ uuids for there to be a probability 10^{-9} of collision (two identical uuids)[27]. There is therefore no need for a central index keeping track of previously generated uuids to ensure uniqueness. We use these uuids to sample episodes and avoid training on a question which has already been asked.

[3]See Whoosh, a Python search engine library http://whoosh.readthedocs.io

When the environment receives a *query* action, it searches the titles or contexts of all available articles for the words in the action. It returns the documents whose title or context is the closest to the action string according to the BM25F scoring algorithm [41]. The one actually used for a given timestep is the highest scoring one which has not yet been seen during the episode. If none is found or all have been seen, then the Environment returns the document whose context is closest to the current one.
All in all, the IDB stores every field in the dataset plus an uuid field, and indexes the titles, contexts and uuids.

3.2.2 Possible Actions

At every step of an episode, the Agent has to chose an action to perform. There are two kinds of actions: *query* actions and *submit* actions.
The *submit* actions come from WFastQA and SFastQA. Each of these outputs a list of 5 sentences from the context, along with their confidence score, which are candidate answers for the current question. To do so, the models are given the episode's question and the current context. The scores range from 0 to 1 with, in practice on the RL Dataset, few values larger than $0.5 \cdot 10^{-1}$. Also, as we force the models to output more than one possible answer confidence scores can be very low, as low as 10^{-20} or even closer to 0. To avoid this scaling problem which would prevent the agent from numerically differentiating between such values, we do not return the raw scores, rather their log, clipped to -6. By doing so we ensure a reasonable range of values and consider that the minimal possible confidence score is actually 10^{-6}.
If the Agent does not trust any of these 10 *submit* actions to be the answer to the episode's question, it can *query* the environment for another article. These *query* actions are the 5 noun phrases from the context with highest similarity with the question.
If, at a given state, the action received by the Environment is a *query* action, then the context the FastQA models read is not the one which was originally given with the question in the dataset, but the new one resulting from the query.

Overall the Agent can perform any one of 15 actions which entirely and solely depend on the (Question, Context) pair. Along with each of these action is associated a score, either the FastQA's score or the cosine similarity between the noun-phrase and the question.

3.2.3 End of an episode

An episode ends in one of the two following situations:

- The agent performs a *submit* action
- The episode reaches the maximum allowed length, which is set to 15 and noted T_{max}. By doing so we assume that no answer can be more than 15 articles away from its original instance's article.

When the episode is over, the Environment returns an *end* token to signal that it has to be reset and that the Agent can not perform any more action. The last returned state is said to be terminal and is noted T, with $T \leq T_{max}$.

3.2.4 Reward

We implemented four types of rewards, which have to be validated as this is a very important part of the Learning Algorithm's design. In the following definitions, a_t and a_{t-1} represent the current and previous actions. The answer being constant throughout an episode it is not indexed by time. The four reward types have the same structure: one common reward formulation for when the state is terminal, and 4 different rewards during the rest of the episodes.

- State is terminal, $t = T$:

$$\begin{cases} 2 * cosine(answer, a_T) \text{ if } score(answer, a_T) \; > \; 0 \\ -2 \text{ if } score(answer, a_T) \; \equiv \; 0 \text{ and } T \neq T_{max} \\ -5 \text{ otherwise} \end{cases} \tag{3.4}$$

- Otherwise:

1. **Instant cosine**:
$$\frac{1}{2}cosine(answer,\ a_t) - 0.5 \tag{3.5}$$

2. **Instant score**:
$$\frac{1}{2}score(answer,\ a_t) - 0.5 \tag{3.6}$$

3. **Improve cosine**:
$$\begin{cases} cosine(answer,\ a_t) - cosine(answer,\ a_{t-1}) \\ -0.25 \text{ if the above value is } 0 \end{cases} \tag{3.7}$$

4. **Improve score**:
$$\begin{cases} score(answer,\ a_t) - score(answer,\ a_{t-1}) \\ -0.25 \text{ if the above value is } 0 \end{cases} \tag{3.8}$$

The idea of the *terminal* case is that the agent can reach the end of an episode in two situations: by submitting or reaching the maximum possible number of steps. In the first case we encourage the Agent to submit at least partial answer by returning the *cosine*, which yields a less sparse reward than the *score*. We however want the action to have a positive *score* otherwise it means there really is little match and we want to penalize this behavior with a negative reward. If the Agent reaches the maximum timestep ($T = T_{max}$), it means it could not find the actual answer and performed no submission at all. We penalize this situation with a strong negative reward.

Regarding non-terminal states, *Instant* strategies measure the reward in terms of similarity or score with the answer. We want the Agent to go to the answer as soon as possible which explains why we scaled the reward to be a small negative number: *cosine* $\in\ [0,1]$[4] (as *score*) so the reward is in $[-0.5, 0]$. This way we encourage a smaller number of steps.

[4]Usually *cosine* $\in [-1;1]$ however the definition we use, based on counts, does not allow for negative numbers.

In the spirit of [34] *Improve* strategies on the other hand reward the Agent in terms of improvement (in score or similarity) compared with its previous attempt, with $r_0 = 0$. This situation however allows for a deviant strategy: if a query does not return a *new* context, then the score would be 0 and the *idle* strategy of always submitting the same query is not penalized, and even increases the return as the only non-zero reward is strongly discounted ($-5 * \gamma^{T_{max}}$ at worst). To avoid this we penalize non-improvements with a small negative reward.

3.2.5 Returned Values

In conclusion, the information returned by the Environment to the Agent after an action is composed of:

1. Actions
   - (a) 5 candidate answers from SFastQA (*submit*)
   - (b) 5 candidate answers from WFastQA (*submit*)
   - (c) 5 Noun Phrases from the context (*query*)
2. State
   - (a) The 10 *submit* scores (log of confidence)
   - (b) The 5 *query* scores (log of similarity with question)
   - (c) The current question
3. Reward from the previous action
4. An *end* token

3.2.6 Environment Scenario

The environment itself contains 2 models (SFastQA and WFastQA), a searcher for the Indexed Database and a parser to extract noun phrases. At the beginning of **training**, it initializes a list of known uuids: it is the list of instances which have already been trained on. It also initializes the models with their pre-trained weights. For the **first step** of an episode, it samples an unknown instance from the IDB, setting the current context, the episode-specific question and the episode-specific answer. The current context and the question are fed to the models which output candidate answers along with their scores. The noun-phrase parser gives the Environment the candidate queries along with their scores. The Environment returns these plus a *not_over* token to the Agent. Then it receives an **action** from the Agent and either terminates the episode (as per 3.2.3) or updates the current context from the *query* action and returns a new state. This process is illustrated in figure 3.2.

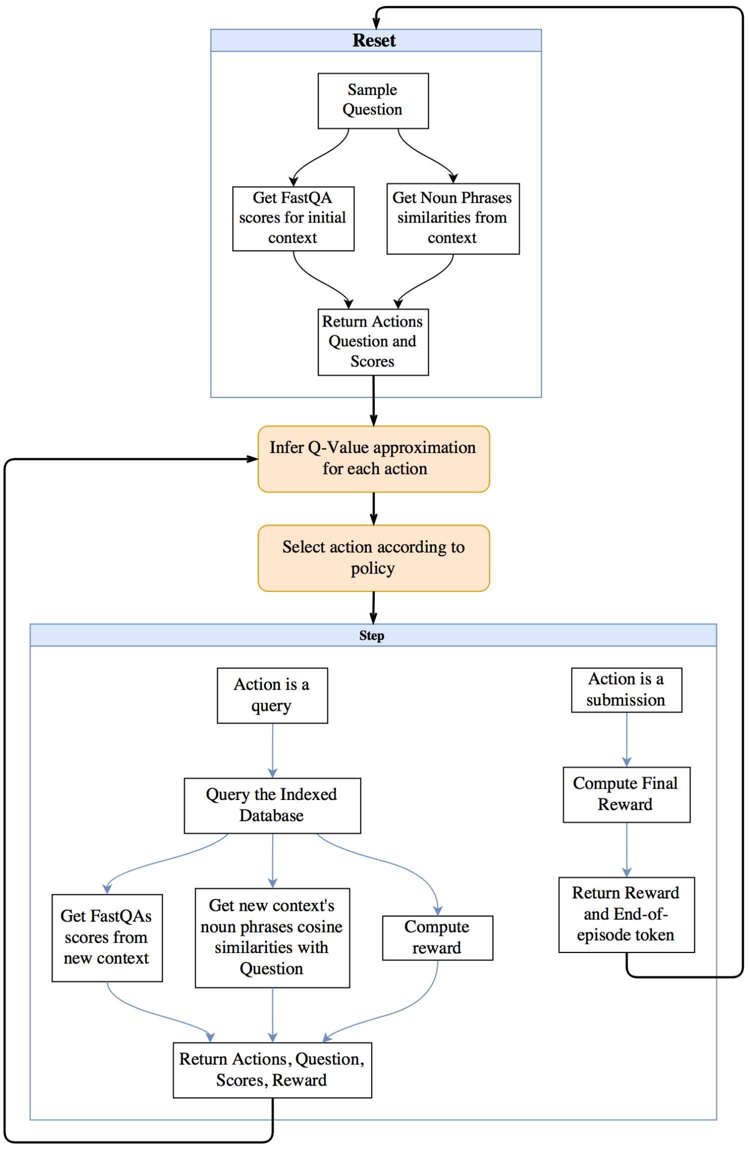

Reset
Sample Question
Get FastQA scores for initial context
Get Noun Phrases similarities from context
Return Actions Question and Scores
Infer Q-Value approximation for each action
Select action according to policy
Step
Action is a query
Query the Indexed Database
Get FastQAs scores from new context
Get new context's noun phrases cosine similarities with Question
Compute reward
Return Actions, Question, Scores, Reward
Action is a submission
Compute Final Reward
Return Reward and End-of-episode token

Figure 3.2: Environment scenario: in blue the two main environment functions, in orange the Agent's two main processing steps.

3.3 Agent

3.3.1 Deep Reinforcement Relevance Network

In our scenario, the set of actions the Agent can perform changes at every timestep of every episode. Plus, it is unbounded[5] due to the exponential complexity of word associations in a sentence. We cannot therefore implement a standard DQN network with one fixed output neuron per specific action (unlike in the Atari setting, in which they have 18 possible actions which are always the same [17]). To solve for this problem, we implemented a form of Deep Reinforcement Relevance Network (DRRNs). This type of network was introduced by He *et. al* [11] to solve for this issue of a Natural Language Action Space.

The idea is that instead of having a unique network mapping for each state-action pair, two different networks are built and the Q-value is computed by comparing the ”relevance” of each action considering the current question.

More precisely, we implement a FNN to represent the Question (the Question Network), a second one to represent the actions (the Action Network) and a third one (the Interaction Network) to approximate the Q-value from the two intermediate representations. Two different networks are desirable as the question and associated actions can be very different in nature (length, vocabulary, syntactic structure etc.).

3.3.2 From Inputs to Interaction Layer

When performing a gradient descent step, we need to have access to the state s_t, to the action a_t the agent took in s_t with a_t's score, the 15 actions $a_{t+1} \in \mathscr{A}_{t+1}$ and their respective scores, to a boolean specifying if s_{t+1} is a terminal state and the reward r_{t+1}.

Actions (a_t and all a_{t+1}) are fed to the Agent as lists of words. They are first transformed into lists of indexes from the embedding vocabulary and then embedded with the GloVe embedding matrix of dimension $n = 50$. As explained in section

[5]Not actually unbouned but in theory an action is a combination of words from contexts. Therefore with a vocabulary of size $\sim$ 10,000,000, the upper bound on the size of the action space grows as $10,000,000^L$, L being the size of a sentence.

2.3.1, we have to represent each action from the list of words it is composed of. We chose the improvement on CBoW described in [5], equation 20. Let $\mathbf{w}_i^a$ be the vector representation of the i^{th} word in action a and $\mathbf{v}^a$ its vector representation. We represent a's *collapsed* representation as:

$$\mathbf{v}_{mean}^a = \sum_{w_i^a \in a} \frac{\mathbf{w}_i^a}{length(a)} \tag{3.9}$$

$$\mathbf{v}_{max}^a = \text{argmax}_{w_i^a \in a} \, \|w_i^a\|_2 \tag{3.10}$$

$$\mathbf{v}^a = \left[\mathbf{v}_{mean}^a; \mathbf{v}_{max}^a\right] \in \Re^{2n} \tag{3.11}$$

The question's collapsed representation is computed the same way. After this operation, the collapsed representations are fed into the Question and Action Networks to yield $\mathbf{d}^a$ and $\mathbf{d}^q$, their *dense* representation.

3.3.3 Q-Value approximation

The Agent's Q-value approximation is computed from $\mathbf{d}^a$ and $\mathbf{d}^q$, using the Interaction network which has an output dimension of 1. As the Agent needs to take each action's score ϕ^a into consideration to compute it's associated Q-value, these need to be added to the dense representations. The input to the Interaction Network is therefore composed of:

1. $\mathbf{d}^a$
2. $\mathbf{d}^q$
3. Submit token $\mathbf{c}_s = \begin{cases} \phi^a \text{ if } a \text{ is a \textit{submit} action} \\ 0 \text{ otherwise} \end{cases}$
4. Query token $\mathbf{c}_q = \begin{cases} \phi^a \text{ if } a \text{ is a \textit{query} action} \\ 0 \text{ otherwise} \end{cases}$

Operations and details regarding intermediate tensor dimensions can be found in Appendix A, Figure A.2.

All in all, the DRRN outputs the Q-Value for each possible action given their collapsed representation, score and the current question as illustrated in figure 3.3.

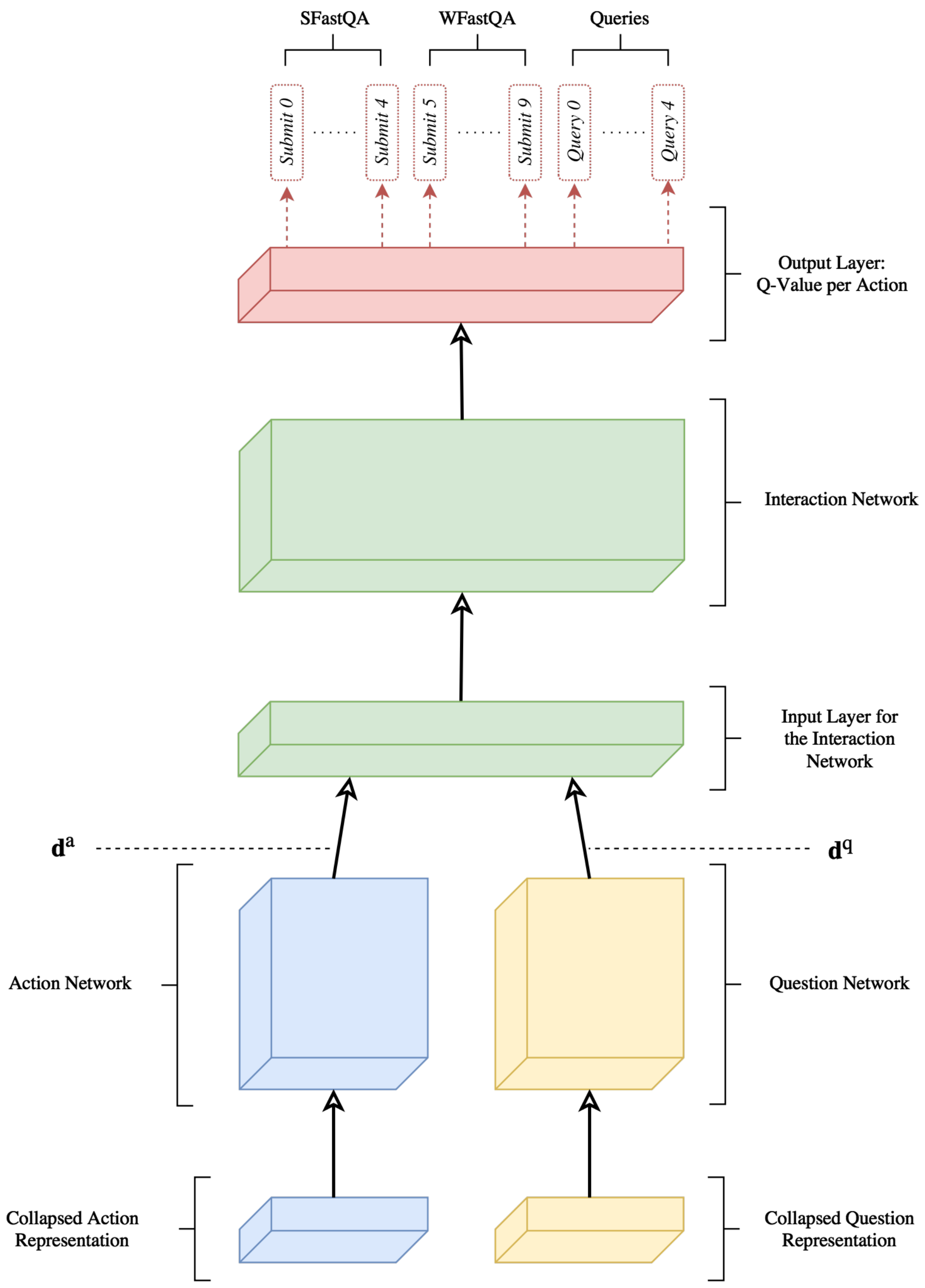


Figure 3.3: Illustration of the DRRN's structure: Question, Action and Interaction Networks. The score tokens are not represented here but are part of the Interaction Network's input layer

Chapter 4

Experiments

In this chapter we will go through the experiences run to evaluate the Agent and the Learning algorithm before we analyze the results: successes and weaknesses. We will see how the dataset issue mentioned in section 3.1.3 limits the overall performance but does not prevent the Agent from improving on the baselines. As a reminder, instances *with* hops are instances for which the answer to the question is not mentioned in the document.

4.1 Outline

All models in this Chapter were trained over training 50,000 steps. After an observation period of 3,000 steps, the training starts. Instance uuids are stored so that they are not repeated during the training procedure.

Models are usually compared by comparing their mean return and learning stability (standard deviation of returns). However the return depending on the reward design, the discount used and the validation set, we therefore compared the *improvement* of the models' mean return on baselines (section 4.2), for each reward type, rather than the raw return value.

Depending on the model architectures, we will mainly compare performance across these hyper parameters:

- reward design (section 3.2.4)
- base epsilon: initial epsilon value for the epsilon-greedy policy

- decay strategy: either exponential or linear decay of epsilon
- discount, in $\{0.5,\ 0.8,\ 1\}$
- batch size at train-time
- optimzier used
- validation set: either on any instance from the validation set or only instances *with* hops
- training set: either on any instance from the training set or only instances *with* hops

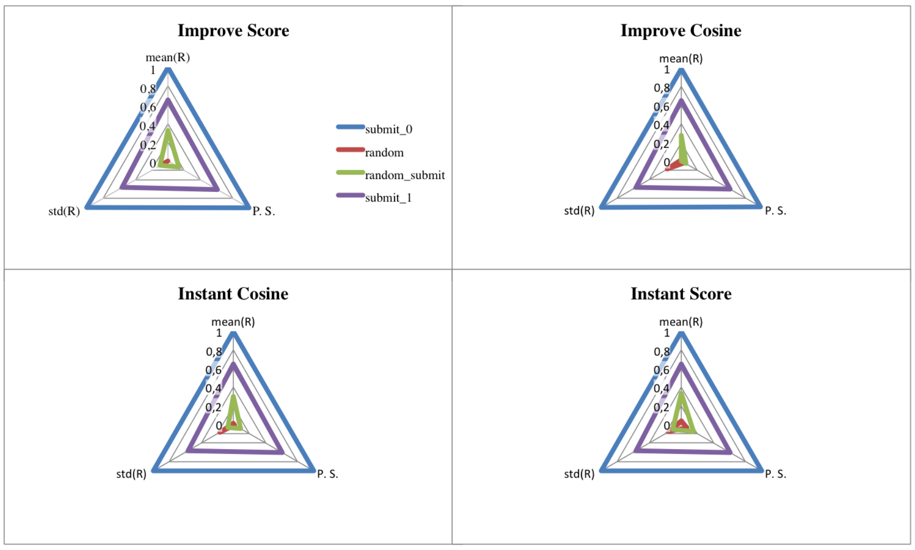
Unless specified otherwise, these experiments were parametrized using a random search as per [3].

4.2 Baseline

For each reward types, we ran the following baselines on the validation set, for 1000 episodes:

1. *random*: uniformly sample an action from the possible actions returned by the environment
2. *random submit*: uniformly sample one of the *submit* actions returned by the environment
3. *submit action i*: always submit the same action, $i \in \{0,\ 1,\ 5,\ 6\}$, that is to say always submit the most likely answer according to SFatsQA (0), always submit its second (1), always submit WFastQA's most likely answer (5), always submit its second (6).

As explained previously, the return also depends on the discount and the type of instances from the validation set selected. Each baseline policy was therefore also computed across these variables. Table 4.1 shows the mean return and its standard deviation for *submit*_0 which consists of always having SFastQA answer the



Values are normalized to [0,1] from Tables C.1.-C.4. SFastQA is better than WFastQA or random but also more variable.

Figure 4.1: Comparaison of 4 baseline policies for the 4 reward types.

question. Details for other policies can be found in Appendix C, Tables C.1.-C.4. Partial successes are the number of times a strictly positive reward was obtained. We can see from these baselines that unsurprisingly, they cannot answer at all instances with hops.

4.3 Models

In this section's tables, ”P. Success” and ”P. S.” stand for Partial Success, ”B. I.” stands for Baseline Improvement, ”Epsilon” stands for the epsilon-greedy policy's initial epsilon used before it is decayed, ”O. H.” stands for Only Hops. The return is noted $\boldsymbol{R}$. Until the last model, the Environment does not return unique contexts per episode: it returns the highest scoring one according to the query (which could be the same), and it definitely returns the same of none is found. Moreover, Environments in models 1 to 4 (included) query for articles' *titles* not *context*.

4.3.1 First model

The First model is very different from the final model described in Chapter 3. Here are its main differences:

1. Learning Algorithm:

Mean($\boldsymbol{R}$)	Partial Success	Reward Type	Discount	Std($\boldsymbol{R}$)	Only Use Hops
-1.025	277	instant_score	1.0	1.599	False
-1.025	277	improve_score	1.0	1.599	False
-1.025	277	instant_cosine	1.0	1.599	False
-1.031	274	improve_cosine	1.0	1.600	False
-1.034	269	improve_cosine	0.8	1.612	False
-1.041	271	instant_cosine	0.8	1.596	False
-1.048	267	instant_score	0.8	1.599	False
-1.048	267	improve_score	0.8	1.599	False
-1.061	266	instant_cosine	0.5	1.579	False
-1.068	267	improve_cosine	0.5	1.568	False
-1.097	259	improve_score	0.5	1.550	False
-1.097	259	instant_score	0.5	1.550	False
-1.984	4	improve_cosine	0.5	0.252	True
-1.988	3	improve_cosine	0.8	0.219	True
-1.996	1	instant_cosine	0.8	0.126	True
-1.996	1	improve_cosine	1.0	0.126	True
-2.000	0	instant_cosine	1.0	0.000	True
-2.000	0	instant_cosine	0.5	0.000	True
-2.000	0	instant_score	0.5	0.000	True
-2.000	0	improve_score	1.0	0.000	True
-2.000	0	instant_score	0.8	0.000	True
-2.000	0	improve_score	0.8	0.000	True
-2.000	0	instant_score	1.0	0.000	True
-2.000	0	improve_score	0.5	0.000	True

Table 4.1: SFastQA baseline, for each type of reward, validation set and discount.

   (a) standard Q-learning

   (b) no discount ($\gamma = 1$)

2. DRRN:

   (a) non-linear activation: ReLU (instead of SELU) for the Question and Action Network

   (b) no Interaction Network: Q-value represented by a dot-product as per [11]

3. Rewards:

   (a) no small (-0.25) negative intermediate reward for *improve* rewards

(b) no differentiated final reward between $T = T_{max}$ and $T \neq T_{max}$

(c) no instant_score reward: it was assumed to be too sparse to be tested at this point

4. Environment

(a) Articles are queried per their title (not context)

(b) A context may be returned several times within an episode

In these experiments, the *Adam* optimizer [23] was consistently used. As we can see this model cannot improve on baselines. Also, its return's standard deviation is very high, suggesting a very unstable result of the learning procedure.

B. I.	Mean($\boldsymbol{R}$)	P. Success	Std($\boldsymbol{R}$)	Reward Type	Batch Size	Epsilon	Decay Strategy
-12.3	-1.158	26.0	1.021	improve_cosine	64	0.3	linear
-18.7	-1.223	33.0	1.167	improve_cosine	64	0.4	linear
-47.9	-1.524	11.0	0.725	improve_cosine	256	0.3	linear
-49.7	-1.543	21.0	1.016	improve_cosine	256	0.4	exponential
-49.8	-1.536	15.0	0.790	improve_score	256	0.6	linear
-64.6	-1.687	16.0	0.885	improve_score	128	0.6	exponential
-75.5	-1.799	23.0	1.058	instant_cosine	128	0.7	linear
-85.5	-1.902	16.0	0.980	instant_cosine	512	0.3	linear
-93.1	-1.980	23.0	1.379	instant_cosine	128	0.5	linear
-93.1	-1.980	23.0	1.379	instant_cosine	128	0.5	linear

Table 4.2: First model results, Baseline Improvement in percentages

4.3.2 Second Model: Using SeLU, regularization and an interaction layer

To improve on the previous results and stabilize learning the model was modified with normalization (SELU) and regularization. The Q-value final representation was also changed from dot product to FNN to allow for more interaction between the Dense Question Representation and the Dense Action Representations. Discount and optimizer are the same as for the first model. Results can be seen in Table 4.3

As we can see, even if performance is not improved, it is in the same order of magnitude and the standard deviation is consistently lower.

B. I.	Mean($\boldsymbol{R}$)	P. Success	Std($\boldsymbol{R}$)	Reward Type	Batch Size	Epsilon	Decay Strategy
-13.9	-1.175	12.0	0.735	improve_cosine	64	0.7	linear
-53.7	-1.584	23.0	1.028	improve_cosine	512	0.5	linear
-58.2	-1.622	22.0	0.913	improve_score	128	0.7	linear
-58.7	-1.636	20.0	0.951	improve_cosine	64	0.5	linear
-64.0	-1.682	18.0	0.886	improve_score	128	0.5	exponential
-69.9	-1.742	14.0	0.750	improve_score	128	0.7	linear
-80.2	-1.847	7.0	0.554	improve_score	512	0.6	linear
-84.2	-1.889	16.0	0.896	instant_cosine	128	0.4	linear
-86.8	-1.915	16.0	0.946	instant_cosine	256	0.3	linear
-90.6	-1.954	14.0	0.937	instant_cosine	128	0.5	linear
-90.9	-1.958	14.0	0.912	improve_score	512	0.4	exponential
-91.0	-1.959	15.0	0.989	instant_cosine	128	0.4	linear
-134.6	-2.405	17.0	1.272	instant_cosine	512	0.6	linear

Table 4.3: Second model results, Baseline Improvement in percentages

4.3.3 Third model: DQN

To reach a better performance, we improved the learning algorithm: from standard Q-learning to DQN with a Target Network and replay memory. We also varied the optimizer used.

B. I.	Mean($\boldsymbol{R}$)	P. Success	Std($\boldsymbol{R}$)	Reward Type	Batch Size	Epsilon	Decay Strategy	Optimizer
-59.9	-1.640	20.0	0.917	improve_score	256	0.9	exponential	Adam
-67.7	-1.729	14.0	0.848	improve_cosine	256	0.6	linear	RMSProp
-72.7	-1.771	24.0	1.150	instant_cosine	256	0.7	linear	RMSProp
-93.8	-1.987	12.0	0.855	instant_cosine	64	0.5	linear	Adam
-97.6	-2.026	9.0	0.725	instant_cosine	64	0.6	exponential	Adam

Table 4.4: Third model results, Baseline Improvement in percentages

The results are however disappointing. DQN does not improve on Q-learning on the mean return nor its stability. Partial successes are however not that bad, at least not worst as is the return. The sample size is also notably smaller so further improvements on the sole learning algorithm were implemented.

4.3.4 Fourth model: Double-DQN with Prioritized Experience Replay

As pointed out by [50], DQN revealed to be still quite unstable and improvable: we therefore implemented Double-DQN and Prioritized Experience Replay.

Double DQN and Prioritized Experience Replay does help in comparison with DQN. Both the mean return and its standard deviation are significantly better but it

B. I.	Mean($\boldsymbol{R}$)	P. Success	Std($\boldsymbol{R}$)	Reward Type	Batch Size	Epsilon	Decay Strategy	Optimizer
-17.3	-1.209	9.0	0.610	improve_cosine	128	0.9	linear	RMSProp
-18.1	-1.211	10.0	0.645	improve_score	256	0.8	linear	RMSProp
-20.1	-1.231	12.0	0.657	improve_score	64	0.5	exponential	RMSProp
-66.1	-1.712	8.0	0.623	improve_cosine	512	0.6	exponential	Adam
-175.4	-2.824	9.0	1.119	instant_cosine	512	0.6	linear	Adam

Table 4.5: Fourth model results, Baseline Improvement in percentages

is surprisingly still not as good as Q-learning.

4.3.4.1 Using only hops

At this point a comparison is needed: even if these models don't improve on the baselines on average, can they do anything in the presence of hops: what if the models are trained and/or validated only on instances where the answer is *not* in the associated context? These results can be found in Table 4.6. For comparison purposes, we fixed all parameters but the training set: T.O.H. stands for Training Only on Hops: whether or not instances with the answer in the context were allowed at training time. Validation is made only on instances *with* hops for all models.

B. I.	Mean($\boldsymbol{R}$)	P. Success	Std($\boldsymbol{R}$)	Reward Type	Batch Size	Epsilon	Decay Strategy	T. O. H.
34.0	-1.317	1.0	0.366	improve_cosine	256	1.0	linear	False
33.9	-1.319	0.0	0.372	improve_cosine	256	1.0	linear	False
33.7	-1.323	1.0	0.372	improve_cosine	256	1.0	linear	True
33.6	-1.325	1.0	0.370	improve_cosine	256	1.0	linear	False
33.5	-1.327	1.0	0.367	improve_cosine	256	1.0	linear	True
33.4	-1.330	6.0	0.408	improve_cosine	256	1.0	linear	True
33.1	-1.335	3.0	0.383	improve_cosine	256	1.0	linear	True
33.1	-1.335	1.0	0.373	improve_cosine	256	1.0	linear	False
33.0	-1.338	1.0	0.363	improve_cosine	256	1.0	linear	True
32.9	-1.338	0.0	0.368	improve_cosine	256	1.0	linear	True
32.8	-1.341	0.0	0.377	improve_cosine	256	1.0	linear	True
32.4	-1.349	1.0	0.372	improve_cosine	256	1.0	linear	True
32.2	-1.353	2.0	0.384	improve_cosine	256	1.0	linear	True
31.9	-1.359	1.0	0.405	improve_cosine	256	1.0	linear	False
31.5	-1.367	0.0	0.360	improve_cosine	256	1.0	linear	False

Table 4.6: Fourth model results, always validating only on instances with hops

This experiment highlights that the Agent can improve on the baseline when only instances with hops are used. Training only on such instances or not does not seem to have a great impact. This is quite expected as on the one hand it gets more knowledge from a greater variety of instances but this knowledge does not help with the actual validation task. Learning is also much more stable, the policy learned seems to be much more consistent. Lastly we notice that results are overall very consistent between runs, invalidating the possibility of training on an outlier sample of the dataset.

However, looking at the decisions made, we can see what the Agent has actually learned: postpone the negative reward. The partial successes specifically show that improving on the mean return does not imply actually improving on the question answering task.

Training Step	Episode	Question	Answer	Time step	Action Taken
500	1	What is Aslan (band) an instance of ?	rock band	1	Query: an appearance
				2	Query: an Irish rock band
				3	Query: this band
				4	Submit: Kills
	2	What is G'day (album) an instance of ?	album	1	Submit: jazz
20,000	1	What is the sex or gender of Eva Zeikfalvy ?	female	1	Query: the country
				2	Query: a 1911 short silent comedy film
				3	Query: a 1911 short silent comedy film
				4	Query: Blanche Sweet
				5	Query: a 1911 short silent comedy film
				6	Query: a 1911 short silent comedy film
				7	Submit: Country Lovers
	2	In which country is Alva (Gotland) ?	Sweden	1	Query: Gotland Municipality
				2	Query: Gotland Municipality
				3	Submit: Sweden
50,000	1	When was Andy Propst born ?	7 March 1965	1	Query: Propst
				2	Query: Propst
				...	*idem*
				15	Query: Propst
	2	What is Gagan Biyani an instance of ?	human	1	Query: an Indian American serial entrepreneur
				2	Query: an Indian American serial entrepreneur
				...	*idem*
				15	Query: an Indian American serial entrepreneur

Table 4.7: The Agent learns to postpone its presumably unavoidable mistake

Table 4.7 shows this circumvention scheme by showing sample episodes from the validation procedure at different steps during the training. After only 500 steps, the Agent is not far from a random Agent. Even if its guesses seem related to the question, on average it is mostly out of luck. As the training goes on, the Agent learns to query more and more. Rare successes as the one showed above (Episode 2 of Training step 20,000) encourage it to keep submitting. By the end of the training, this effect has disappeared and the agent only does one thing: query the *query* action

B. I.	Mean($\boldsymbol{R}$)	Mean($\boldsymbol{R}$) O. H.	P. S.	P. S. O. H.	Std($\boldsymbol{R}$)	Std($\boldsymbol{R}$) O. H.	Reward Type	Batch Size
68.6	-0.565	-0.692	69.0	4.0	0.526	0.396	improve_cosine	64
67.6	-0.585	-0.700	62.0	4.0	0.543	0.411	improve_score	64
41.9	-1.055	-1.123	55.0	7.0	0.453	0.269	instant_cosine	64
-63.0	-3.261	-3.462	65.0	8.0	1.433	1.198	improve_score	256
-64.9	-3.291	-3.493	57.0	2.0	1.415	1.187	improve_cosine	256
-65.3	-3.307	-3.582	60.0	5.0	1.376	1.189	improve_score	256
-69.1	-3.374	-3.486	53.0	4.0	1.375	1.193	improve_cosine	256
-70.4	-3.402	-3.494	64.0	3.0	1.466	1.168	improve_cosine	256
-148.5	-4.971	-5.211	48.0	4.0	2.201	2.037	instant_cosine	256
-148.6	-4.971	-5.162	50.0	0.0	2.202	2.028	instant_cosine	256

Table 4.8: Fifth model results, always validating only on instances with hops

with highest score until the Environment stops the episode by itself. This may be explained as in section 3.2.4: the strategy of always querying and not answering (*i.e.* not *submitting*) in fact pays more than trying and failing.

4.3.5 Fifth model: Searching the context and Designing new rewards

To try and solve the previously described issue, we made two improvements to the Environment to encourage the Agent to learn the actual task:

1. **Context**: for previous models, the Environment would search the query only in the database's *title* index. Table 4.7 however suggests that after a query, the Environment does not find any other article (or at least none better than the original one) and therefore returns the same one. By searching the context however one can expect there to be more variety in the articles the Agent sees.

2. **New rewards**: up to now, rewards allowed for the circumvention scheme. We therefore differentiated $T = T_{max}$ and $T < T_{max}$, rescaling the former one to be even more penalizing. We also altered the *improve* strategies for them not to return 0 in case of *no* improvement. Finally the Instant Score reward type was introduced.

Unlike precedent models, this 5^{th} model was validated twice: once on the standard validation steps and once with the validation set using only instances with hops (O. H.).

Table 4.8 shows how the modifications increased the model's performance and shows a slightly higher rate of successes on the instances with hops. We therefore decided to explore further this model's abilities.

4.3.5.1 Grid Search

From previous explorations we selected a few parameters and their values to run a grid search on. According to Table 4.8, we set a consistent batch size of 64. The search parameters were *(i)* the reward's type, *(ii)* the optimizer used, *(iii)* the discount's value and *(iv)* the memory's size (in number of transitions stored).

Even though Table D.1 demonstrates improving results comparable to that of Table 4.8, looking at the actions taken shows how the Agent's behavior was little altered by this model's changes. More specifically, the policy of always querying is still important. The prevalent scenario (querying various actions and then submitting) may seem reasonable, however looking in details we see that on average unique actions represent only 30% of all actions taken in an episode. This means actions are still very redundant. Unlike model 4 however, the Agent still *submits* actions by the end of the training. More specifically it trusts a lot SFastQA with its ability to read *dates* and answer *date of birth* questions.
To compare this improvement due to the new design of the reward and the query of contexts rather than titles, we looked into the validation's specifics: *(i)* how many episodes with immediate *submit* action, *(ii)* how many episodes with only one type of *query* and no *submit*, *(iii)* how many episodes with several queries before a *submit* action and *(iv)* how many episodes with various *query* actions but no *submit*. For episodes of type *(iii)*, we also measured the average (over such episodes) ratio of unique actions, that is the number of different actions divided by the length of the episode.
Table 4.11 reveals that the 5^{th} model's changes produce improvements in the desired direction: more *submit* actions in general. It did not however solve the issue of always querying the same action without submitting or even changing the query.

Optimizer	**Mean(*R*)**	**Std(*R*)**	**Mean(*R*) O. H.**	**Std(*R*) O. H.**
Adam	-1.86	2.18	-1.99	2.26
RMSProp	-1.54	2.20	-1.65	2.21

Table 4.9: Comparison of optimizers' performances accross the Grid Search

4.3.6 Last Model: prevent redundant contexts

In the light of all the previous decisions, we decided to measure our model's performance when forced to use new contexts. In other words, the Environment is modified not to return twice the same context within one episode. If there is no such context, the closest one (according to the IDB) is returned. The agent and rewards are the same as for model 5.

As for the previous section, we ran a grid search on discounts and reward types. Table D.1 clearly shows that a memory of size 2000 is optimal and Table 4.9 shows that *RMSProp* outperforms *Adam* in our task so we did not include them in the search.

It seems as if there were few differences between this later model and the previous one in terms of return and performance against the baseline (Table 4.10). Once more it is necessary to look at decisions to see the actual behavior: Table 4.11 show that as expected, the Environment's constraint did force the model to submit more, and vary its queries. However we can also see that in the event of no more unknown context for a given query, even the search for other similar context has limits and returns the same context. This could be prevented by simply terminating the episode, but it does not change the dataset's issue, we therefore did not alter the Environment further.

B. I.	Mean(***R***)	Mean(***R***) O. H.	P. S.	P. S. O. H.	Std(***R***)	Std(***R***) O. H.	Reward Type	Discount
66.0	-0.611	-0.696	35.0	4.0	0.440	0.418	improve_cosine	0.5
40.7	-1.081	-1.119	22.0	5.0	0.301	0.261	instant_score	0.5
39.6	-1.095	-1.125	34.0	10.0	0.364	0.274	instant_cosine	0.5
29.5	-1.401	-1.514	42.0	6.0	0.463	0.208	improve_cosine	0.8
-15.7	-2.309	-2.399	38.0	7.0	0.513	0.236	instant_cosine	0.8
-17.8	-2.356	-2.399	28.0	3.0	0.413	0.200	instant_score	0.8
-152.7	-5.044	-5.170	26.0	3.0	2.738	2.587	improve_cosine	1.0
-260.2	-7.278	-7.204	36.0	6.0	3.994	3.868	instant_cosine	1.0
-263.0	-7.231	-7.383	24.0	4.0	3.949	3.932	instant_score	1.0

Table 4.10: Sixth model's grid search results, always validating only on instances with hops

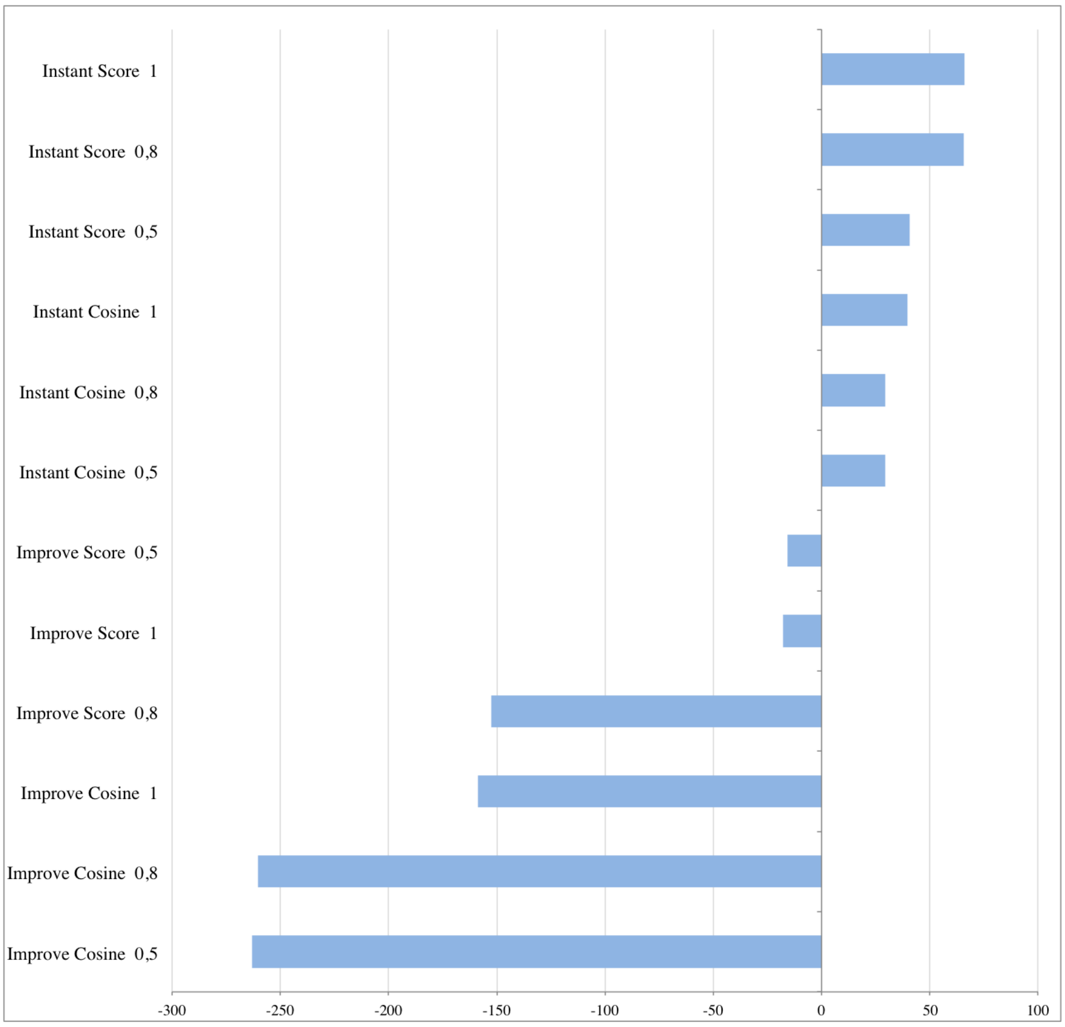


Figure 4.2: Sorted Baseline Improvements (in %) of the grid search on model 6's discount values and reward types.

Model	Submit	Unique Query	Query Then Submit		Various Queries, No Submit
			# of Episodes	Ratio of Unique Actions	
4	20	245	525	0.29	210
5	99	242	480	0.30	179
6	109	214	568	0.61	109

Table 4.11: Comparison between the 4th, 5th and 6th model's decisions

Chapter 5

General Conclusions

5.1 Findings and Discussion

Gradually experimenting and changing the model proved to be a valuable approach to understand the challenges of the task. If previous work had already shown that D-DQN was more potent than standard Q-learning, the design of the reward was a purely experimental procedure. Inspiration from other research was obviously needed but the thorough analysis of the models' actions was even more informative.

The overall improvement path we implemented is as follows:

1. Stabilize learning with **L2 regularization** and **SELU activations**
2. Increase interactions between Question and Action representations with an **Interaction Network** instead of a simple dot product
3. Improve the learning algorithm to DQN then to **D-DQN** with Prioritized Experience Replay
4. Limit the Circumvention Strategy of always querying until the Environment stops the episode by designing **new rewards** and searching the contexts (not the sole titles of documents)
5. Limit it even more by preventing the Environment from retrieving **redundant contexts** (twice the same context within one episode).

The final model's grid search found that the optimal set up is the following:

1. Search the context and prevent redundancy
2. Learn using D-DQN with Prioritized Experience Replay
3. Use the *improve_cosine* reward strategy
4. Linearly decay epsilon as the exponential decay stops exploration too early
5. Use the RMSProp optimizer
6. Save 2000 transitions in the memory
7. Discount the reward by 0.5
8. Compute the interaction between the Question and Action representations with an Interaction Network

A summary of the experiments run can be found in figure 5.1. The Instant Score reward type was not introduced before experimenting with the fifth model. We can see how successive modifications led to significant improvements. The main difference between models Five and Six being the Environment returning non-redundant rewards, performance is very similar yet much better than previous models as rewards were designed to prevent deviant behaviors.

All in all, we find that our setup allows the Agent to partially learn how to query the environment to eventually submit an Answer. This successful behavior is not however the prevailing one and the Agent queries more often than it submits. Situations where the Agent does not submit at all before the episode is terminated by the Environment are not negligible and we will investigate why hereafter.

Along whith the findings, the experiments also showed how qualitative analysis of decisions is paramount as the quantitative measure of the return can be biased by circumvention strategies. We hypothesize that these still happen in spite of the Environment's settings and reward design for two main reasons:

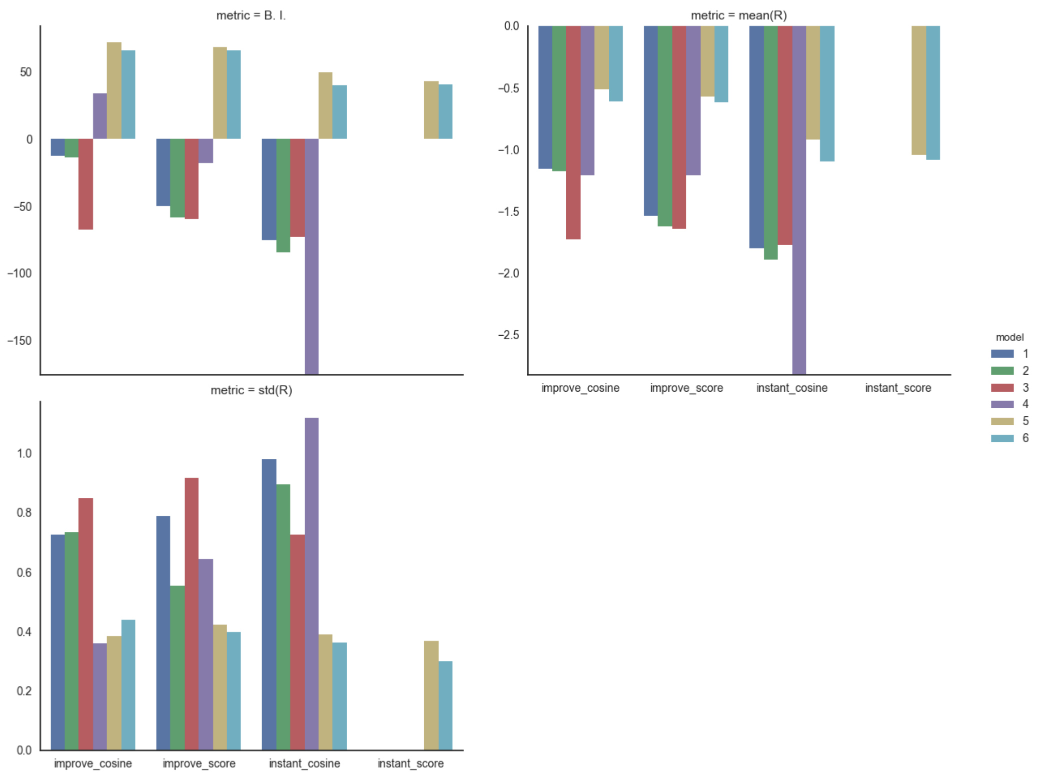


Figure 5.1: Baseline Improvement, Mean Return and Standard Deviation of Return for the best-achieving Agents, per model and reward types

1. **Dataset**: The issue mentioned in section 3.1.3 is a strong limitation to the models' learning possibilities. Not only don't we have a guarantee for the relevant context to be part of the dataset, we may also cripple the Environment's ability to retrieve meaningful intermediary context. Ideally, it should not be possible that the Environment does not find a meaningful document (*c.f.* Section 3.2.1), but our custom dataset's restrictions make it so that we have to work around it. Preventing redundant context is too restrictive as the dataset appears to be too small: it happens often that for a proper noun query there is only one meaningful context. In this case, querying contexts similar to the only one available but which has already been seen may also actually return no new context. This is why there is very little improvement between the last two models tried.

2. **No memory between timesteps**: The Interaction network is static: it has no way of "remembering" what actions it took previously. More specifically

it can be assumed that noun phrases are quite specific to a particular document and that therefore querying one of these will likely return the same context. As all components in the Environment are deterministic, the possible actions at timestep t+1 will be the same as those at timestep t if the context is the same. But as the Agent's networks are static (FNNs) they can not take into account the fact that they already chose a given action at the previous timestep. In turn, the Agent will therefore also act deterministically and statically, querying for the same action at timestep t+1 as the one it chose at timestep t. Preventing the Environment from returning twice the same context within an episode is a workaround but it increases the impact of issue 1.

What is more, looking in details into the *submit* actions taken, we realize that WFastQA is almost never used. Table 5.1 shows the differences between SFastQA's and WFastQA's inferences. It seems that WFastQA is actually quite bad at the task. One explanation for this phenomenon can be found in the analysis of vocabularies. GloVe's vocabulary and the Wikireading vocabulary are actually very dissimilar. This is due mainly to the fact that Wikipedia articles contain a lot of proper nouns and non-English words (mostly foreign names and locations) or even characters for languages with accents, different alphabets or no alphabet at all (as are French, Russian or Chinese). The GloVe vocabulary has 400 000 tokens, our subset of Wikireading has 9 727 963 of these but only 213 913 lie in their intersection. This means that a lot of what WFastQA reads from Wikireading instances (and especially the most important parts as a name or a location) is actually a meaningless Out Of Vocabulary token, making the task close to being impossible as is.

5.2 Further Research

The variation of the CBoW representation of sentences we use to compute the Collapsed representation of the question and actions is limited in many ways, including the fact that it does not take into account the order of words. Using a *Recurrent Neural Network as Sentence Representation Layer* would certainly create a more

Question	In which country is Bedesxan ?	
Answer	Azerbaijan	
Answer Location	[23]	
Context	Bedexan (also, Badasqan, Badashkhan, and Badashkhana) is a village and municipality in the Babek Rayon of Nakhchivan, Azerbaijan. It has a population of 274 .	
SFastQA	Azerbaijan	Confidence: 0.623
WFatsQA	the Babek Rayon of Nakhchivan	Confidence: 0.746
Question	In which country is Hutteldorfer Strasse (Vienna U-Bahn) ?	
Answer	Austria	
Answer Location	[]	
Context	Hutteldorfer Strasse is a station on Line U3 of the Vienna U-Bahn. It is located in the Penzing District. It opened in 1998.	
SFastQA	the Penzing District	Confidence: 0.207
WFastQA	is	Confidence: 0.362

Table 5.1: On two similar instances, with and without the answer in the context, we can see WFastQA's failure: it does not get the answer (1^{st} instance) and selects a worse candidate answer than SFatsQA (2^{nd} instance); yet both times it is more confident.

informative representation of these sentences. This would also allow one to add an Attention mechanism to the Interaction Network to focus on important similarities between each action and the question. The use of CNNs should also be explored.
As stated in Section 5.1 this Interaction Network ought to be able to know what action it has already taken. Once more, a sequential model as an *RNN as Interaction Network* instead of a FNN would allow the model to take into account its previous actions not to perform them again and again even though they are useless.
To solve for the WFastQA issue, one could train it without using pre-trained embeddings and have it learn the word representation from the Wikireading dataset which is large enough to learn in an unsupervised way task-specific word representation. However, even by doing so it seems unlikely that proper nouns appear often enough to yield a meaningful representations. This can be partially avoided however using Entity Anonymisation [12], [32].
Wikireading is a very large dataset with numerous kinds of questions. To answer questions, the Agent relies on 2 extractive models, FastQAs. If none of these can

actually answer the question when it is given the right context, the Agent cannot, by construction, do better. To improve the Agent's capabilities, one could design it as a Master algorithm in an *Ensemble method*: the Environment could be composed of (theoretically) any number and kind of IESs, the Agent would have to learn how to use them to get as shortly as possible to the Answer.
Lastly, we have kept the *query* actions' formulations very simple, selecting only noun phrases and ordering them by cosine similarity with the question. Improving this selection is likely to be another direction for amelioration using Part-of-Speech tagging [12] and Name Entity Recognition [33].

5.3 Conclusion

Extracting answers to questions from a given context is a complex task and a currently open research topic. Answering questions from a complete corpus is even harder and presents very different challenges. In this Thesis we put to use advances in Extractive Question Answering (EQA, with the FastQA model from [52]) to explore how Reinforcement Learning can collaborate with such models by providing them new contexts. On a subset of the Wikireading dataset [13], we trained a Deep Reinforcement Relevance Network as described in [11] to answer questions which had a major difficulty: half of the dataset's instances do not have their answer stated in the associated context. The Agent therefore has to *query* the Dataset for a new document before potentially *submitting* a FastQA model's answer.
We investigated the task by designing a complete set of baselines which varied in terms of policy, validation set and reward design mainly. Models with increasing degrees of complexity were implemented to both improve quantitatively on the baselines while maintaining a coherent qualitative behavior. The final Agent is trained using Double-DQN with Prioritized Experience replay and uses insights from two different FastQA models, pre-trained on SQuAD [39] and Wikireading.
A deviant strategy emerged during those experiments: the reward design allows for immobility, the Agent querying constantly for the same new context until the end of the episode. To account for this we designed a second set of rewards penalizing

such a behavior. While this change did produce encouraging results, limitations in the Dataset itself emerged, suggesting inherent challenges as the size and specificity of its vocabulary and the absence of ground truth locations for answers.

We show that our implementation can improve on the baselines but only rarely succeeds in querying *then* submitting the ground truth answer. We hypothesize two main explanations appart from the dataset: (*i*) the Agent's static internal state preventing it from remembering which actions it performed from timestep to timestep and (*ii*) the basic sentence representation we use for Questions and Actions (algebraic representation from word embeddings [5] in the spirit of CBoWs [1]).

From the present Thesis and related work it seems clear that Reinforcement Learning is a very potent domain to build on successes in EQA. More research is needed in this direction especially regarding the use of Entity Anonimization, Parts of Speech Tagging and Named-Entity Recognition to suggest better queries. Moreover, the setting we develop lands path to using Reinforcement Learning Agents to *exploit* EQA models in the manner of ensemble methods while allowing for *exploration* to provide these with new and adequate context if need be.

Appendix A

Q-value approximation : full procedure

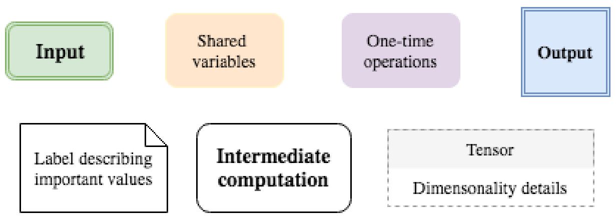


Figure A.1: Legend used in for figure A.3

The D-DQN algorithm uses a target network to evaluate $Q(s_{t+1}, a_{t+1})$ for each $a_{t+1} \in \mathscr{A}_{t+1}$. By ”Shared variables” we mean that the embedding matrix is actually shared as is by the Action and Question Networks, and the Action network itself is shared in the sense that its current weights are used to compute a_t's dense representation and are shared with the target Network. The Target Action network is updated periodically from the Action Network to represent all a_{t+1}.

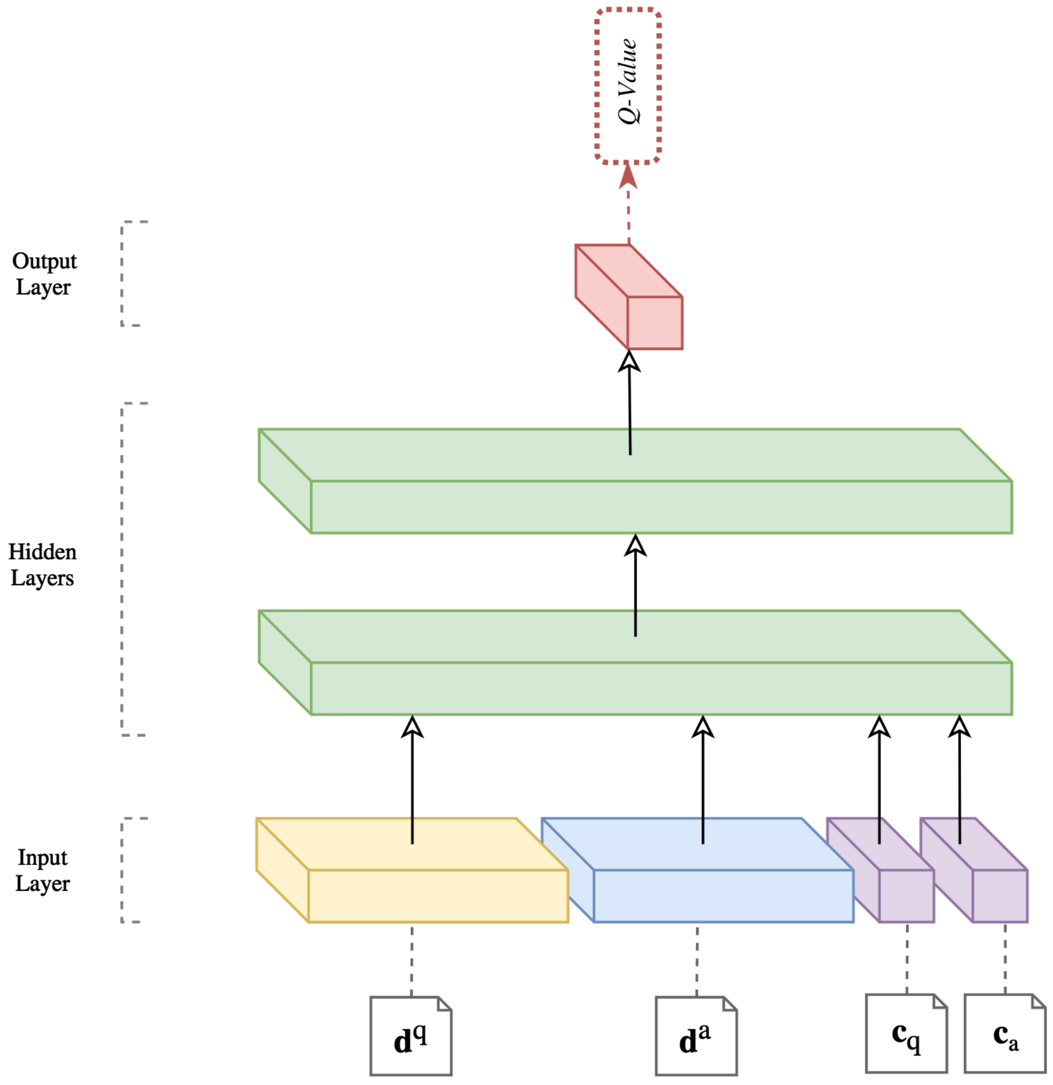


Figure A.2: Detailed Interaction Network

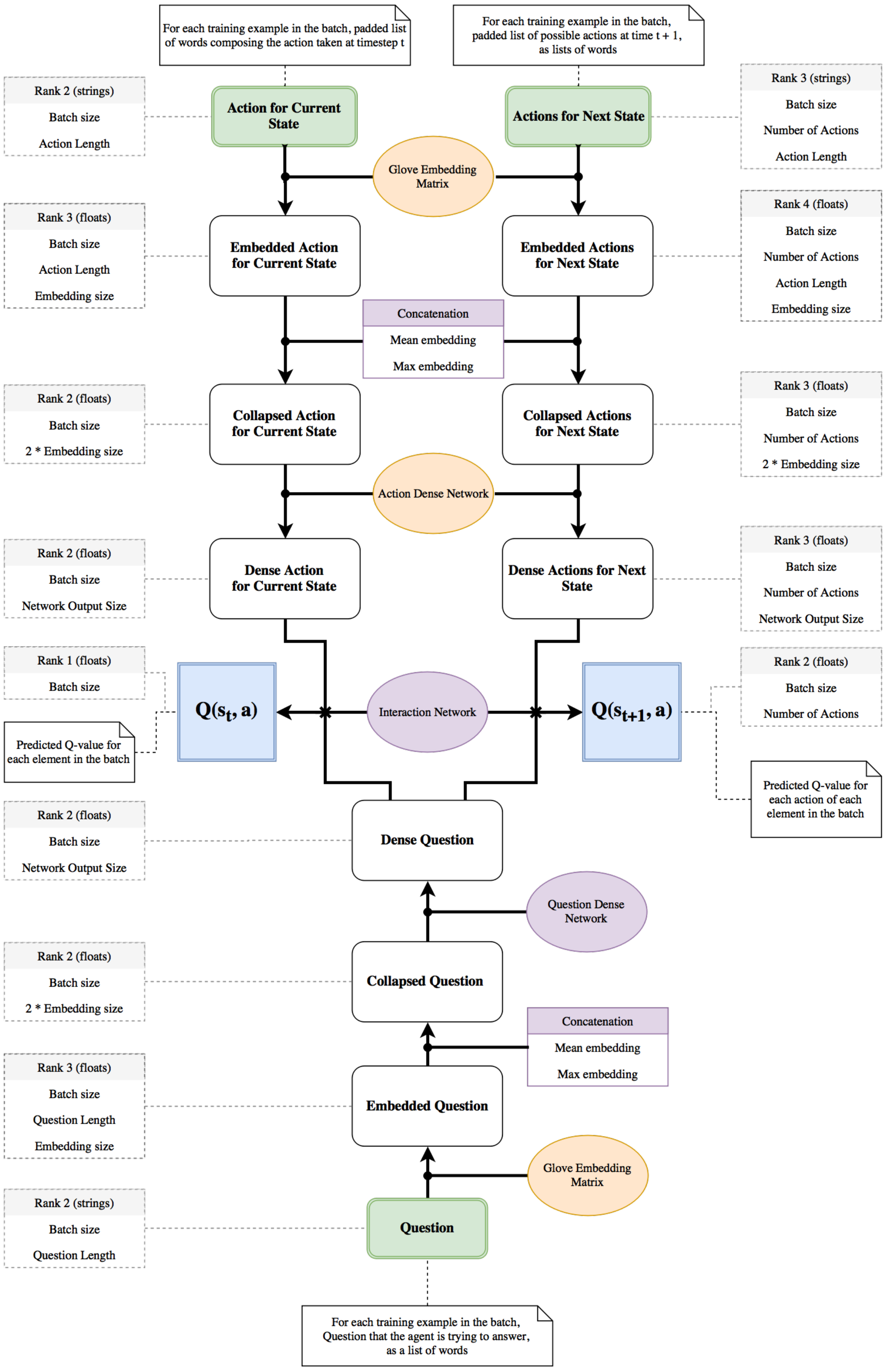


Figure A.3: Full process to compute $Q(s_t, a_t)$ and $Q(s_{t+1}, a_{t+1})$

Appendix B

Hyper-parameters

The following hyper-parameters were set as default and did not vary during the experiences discussed in Chapter 4. PER is short for Prioritized Experience Replay.

Question Network Layers	[64 64 32]
Action Network Layers	[64 64 32]
Interaction Network Layers	[64 32 1]
Learning rate	0.001
LR decay	0.97
Decay LR every (steps)	1000
Regularization	L2
Regularization parameter	0.0001
Softmax scaling parameter	1.0
Dropout (training)	0.8
PER initial Beta	0.4
PER initial max priority	0
PER epsilon	1e-6
PER alpha	0.8
Embedding Matrix	GloVe
Embedding Size	50
Validation every (steps)	500
Maximum episode length	15

Table B.1: Default hyper-parameters

Appendix C

Baselines

Policy	Mean($\boldsymbol{R}$)	Partial Success	Reward Type	Discount	Std($\boldsymbol{R}$)	Only Use Hops
random	-1.534	97	instant_score	0.5	0.923	False
random	-1.539	98	improve_score	0.5	0.916	False
random	-1.566	88	instant_cosine	0.5	0.882	False
random	-1.612	76	improve_cosine	0.5	0.824	False
random	-1.768	87	instant_cosine	0.8	0.908	False
random	-1.774	86	improve_score	0.8	0.888	False
random	-1.788	79	instant_score	0.8	0.883	False
random	-1.793	77	improve_cosine	0.8	0.876	False
random	-1.798	2	improve_cosine	0.5	0.324	True
random	-1.807	2	improve_score	0.5	0.293	True
random	-1.814	3	instant_cosine	0.5	0.312	True
random	-1.823	0	instant_score	0.5	0.279	True
random	-1.905	100	instant_score	1.0	1.056	False
random	-1.927	90	instant_cosine	1.0	1.067	False
random	-1.928	94	improve_cosine	1.0	1.069	False
random	-1.938	87	improve_score	1.0	1.011	False
random	-2.032	6	improve_score	0.8	0.187	True
random	-2.035	4	instant_cosine	0.8	0.141	True
random	-2.037	2	instant_score	0.8	0.140	True
random	-2.037	0	improve_cosine	0.8	0.065	True
random	-2.219	0	improve_cosine	1.0	0.421	True
random	-2.225	0	instant_score	1.0	0.395	True
random	-2.229	1	improve_score	1.0	0.447	True
random	-2.251	1	instant_cosine	1.0	0.458	True
random_submit	-1.587	125	instant_cosine	0.8	1.105	False
random_submit	-1.633	116	instant_score	1.0	1.027	False
random_submit	-1.640	113	improve_score	1.0	1.021	False
random_submit	-1.643	108	improve_cosine	0.8	1.038	False
random_submit	-1.663	104	instant_cosine	1.0	1.004	False
random_submit	-1.676	99	instant_cosine	0.5	0.989	False
random_submit	-1.679	98	instant_score	0.8	0.986	False
random_submit	-1.684	96	improve_score	0.8	0.983	False
random_submit	-1.687	98	improve_cosine	1.0	0.958	False
random_submit	-1.693	94	instant_score	0.5	0.964	False
random_submit	-1.702	92	improve_cosine	0.5	0.947	False
random_submit	-1.711	89	improve_score	0.5	0.937	False

Table C.1: Baselines

Policy	Mean($\boldsymbol{R}$)	Partial Success	Reward Type	Discount	Std($\boldsymbol{R}$)	Only Use Hops
random_submit	-1.992	2	instant_score	1.0	0.179	True
random_submit	-2.000	0	improve_score	0.5	0.000	True
random_submit	-2.000	0	improve_score	0.8	0.000	True
random_submit	-2.000	0	improve_score	1.0	0.000	True
random_submit	-2.000	0	instant_cosine	0.5	0.000	True
random_submit	-2.000	0	instant_cosine	0.8	0.000	True
random_submit	-2.000	0	instant_cosine	1.0	0.000	True
random_submit	-2.000	0	improve_cosine	0.5	0.000	True
random_submit	-2.000	0	improve_cosine	0.8	0.000	True
random_submit	-2.000	0	improve_cosine	1.0	0.000	True
random_submit	-2.000	0	instant_score	0.5	0.000	True
random_submit	-2.000	0	instant_score	0.8	0.000	True
submit_0	-1.025	277	improve_score	1.0	1.599	False
submit_0	-1.025	277	instant_cosine	1.0	1.599	False
submit_0	-1.025	277	instant_score	1.0	1.599	False
submit_0	-1.031	274	improve_cosine	1.0	1.600	False
submit_0	-1.034	269	improve_cosine	0.8	1.612	False
submit_0	-1.041	271	instant_cosine	0.8	1.596	False
submit_0	-1.048	267	improve_score	0.8	1.599	False
submit_0	-1.048	267	instant_score	0.8	1.599	False
submit_0	-1.061	266	instant_cosine	0.5	1.579	False
submit_0	-1.068	267	improve_cosine	0.5	1.568	False
submit_0	-1.097	259	improve_score	0.5	1.550	False
submit_0	-1.097	259	instant_score	0.5	1.550	False
submit_0	-1.984	4	improve_cosine	0.5	0.252	True
submit_0	-1.988	3	improve_cosine	0.8	0.219	True
submit_0	-1.996	1	instant_cosine	0.8	0.126	True
submit_0	-1.996	1	improve_cosine	1.0	0.126	True
submit_0	-2.000	0	improve_score	0.5	0.000	True
submit_0	-2.000	0	improve_score	0.8	0.000	True
submit_0	-2.000	0	improve_score	1.0	0.000	True
submit_0	-2.000	0	instant_cosine	0.5	0.000	True
submit_0	-2.000	0	instant_cosine	1.0	0.000	True
submit_0	-2.000	0	instant_score	0.5	0.000	True
submit_0	-2.000	0	instant_score	0.8	0.000	True
submit_0	-2.000	0	instant_score	1.0	0.000	True

Table C.2: Baselines

Policy	Mean($\boldsymbol{R}$)	Partial Success	Reward Type	Discount	Std($\boldsymbol{R}$)	Only Use Hops
submit_1	-1.240	234	improve_cosine	0.8	1.393	False
submit_1	-1.254	230	improve_score	0.8	1.383	False
submit_1	-1.254	230	instant_cosine	0.8	1.383	False
submit_1	-1.254	230	instant_score	0.8	1.383	False
submit_1	-1.269	230	improve_cosine	0.5	1.354	False
submit_1	-1.280	227	improve_score	0.5	1.346	False
submit_1	-1.280	227	instant_cosine	0.5	1.346	False
submit_1	-1.280	227	instant_score	0.5	1.346	False
submit_1	-1.287	220	improve_cosine	1.0	1.363	False
submit_1	-1.342	203	improve_score	1.0	1.322	False
submit_1	-1.342	203	instant_cosine	1.0	1.322	False
submit_1	-1.342	203	instant_score	1.0	1.322	False
submit_1	-1.993	2	improve_cosine	0.5	0.161	True
submit_1	-1.996	1	improve_cosine	1.0	0.126	True
submit_1	-1.997	1	improve_score	0.5	0.100	True
submit_1	-1.997	1	instant_cosine	0.5	0.100	True
submit_1	-1.997	1	instant_score	0.5	0.100	True
submit_1	-2.000	0	improve_score	0.8	0.000	True
submit_1	-2.000	0	improve_score	1.0	0.000	True
submit_1	-2.000	0	instant_cosine	0.8	0.000	True
submit_1	-2.000	0	instant_cosine	1.0	0.000	True
submit_1	-2.000	0	improve_cosine	0.8	0.000	True
submit_1	-2.000	0	instant_score	0.8	0.000	True
submit_1	-2.000	0	instant_score	1.0	0.000	True
submit_5	-1.970	9	improve_score	0.8	0.315	False
submit_5	-1.970	9	instant_cosine	0.8	0.315	False
submit_5	-1.970	9	improve_cosine	0.8	0.315	False
submit_5	-1.970	9	instant_score	0.8	0.315	False
submit_5	-1.974	8	improve_score	1.0	0.295	False
submit_5	-1.974	8	instant_cosine	1.0	0.295	False
submit_5	-1.974	8	instant_score	1.0	0.295	False
submit_5	-1.980	6	improve_cosine	1.0	0.262	False
submit_5	-1.984	5	improve_cosine	0.5	0.230	False
submit_5	-1.987	4	improve_score	0.5	0.204	False
submit_5	-1.987	4	instant_cosine	0.5	0.204	False
submit_5	-1.987	4	instant_score	0.5	0.204	False

Table C.3: Baselines

Policy	Mean($\boldsymbol{R}$)	Partial Success	Reward Type	Discount	Std($\boldsymbol{R}$)	Only Use Hops
submit_5	-2.000	0	improve_score	0.5	0.000	True
submit_5	-2.000	0	improve_score	0.8	0.000	True
submit_5	-2.000	0	improve_score	1.0	0.000	True
submit_5	-2.000	0	instant_cosine	0.5	0.000	True
submit_5	-2.000	0	instant_cosine	0.8	0.000	True
submit_5	-2.000	0	instant_cosine	1.0	0.000	True
submit_5	-2.000	0	improve_cosine	0.5	0.000	True
submit_5	-2.000	0	improve_cosine	0.8	0.000	True
submit_5	-2.000	0	improve_cosine	1.0	0.000	True
submit_5	-2.000	0	instant_score	0.5	0.000	True
submit_5	-2.000	0	instant_score	0.8	0.000	True
submit_5	-2.000	0	instant_score	1.0	0.000	True
submit_6	-1.946	17	improve_score	0.8	0.414	False
submit_6	-1.946	17	instant_cosine	0.8	0.414	False
submit_6	-1.946	17	instant_score	0.8	0.414	False
submit_6	-1.961	12	improve_cosine	0.8	0.353	False
submit_6	-1.962	12	improve_cosine	1.0	0.349	False
submit_6	-1.962	11	improve_cosine	0.5	0.362	False
submit_6	-1.962	11	improve_score	0.5	0.358	False
submit_6	-1.962	11	instant_cosine	0.5	0.358	False
submit_6	-1.962	11	instant_score	0.5	0.358	False
submit_6	-1.968	10	improve_score	1.0	0.317	False
submit_6	-1.968	10	instant_cosine	1.0	0.317	False
submit_6	-1.968	10	instant_score	1.0	0.317	False
submit_6	-2.000	0	improve_score	0.5	0.000	True
submit_6	-2.000	0	improve_score	0.8	0.000	True
submit_6	-2.000	0	improve_score	1.0	0.000	True
submit_6	-2.000	0	instant_cosine	0.5	0.000	True
submit_6	-2.000	0	instant_cosine	0.8	0.000	True
submit_6	-2.000	0	instant_cosine	1.0	0.000	True
submit_6	-2.000	0	improve_cosine	0.5	0.000	True
submit_6	-2.000	0	improve_cosine	0.8	0.000	True
submit_6	-2.000	0	improve_cosine	1.0	0.000	True
submit_6	-2.000	0	instant_score	0.5	0.000	True
submit_6	-2.000	0	instant_score	0.8	0.000	True
submit_6	-2.000	0	instant_score	1.0	0.000	True

Table C.4: Baselines

Appendix D

Fourth Model Grid Search

B. I.	Mean($\boldsymbol{R}$)	Mean($\boldsymbol{R}$) O. H.	P. S.	P. S. O. H.	Std($\boldsymbol{R}$)	Std($\boldsymbol{R}$) O. H.	Reward Type	Optimizer	Discount	Memory
42.6	-1.046	-1.138	61.0	2.0	0.422	0.301	instant_score	Adam	0.5	2000
42.6	-1.047	-1.129	63.0	6.0	0.490	0.275	instant_score	Adam	0.5	1000
41.5	-1.067	-1.131	59.0	2.0	0.491	0.276	instant_score	RMSProp	0.5	2000
41.5	-1.067	-1.135	46.0	2.0	0.367	0.279	instant_score	Adam	0.5	500
40.6	-1.082	-1.140	53.0	4.0	0.406	0.292	instant_score	RMSProp	0.5	500
40.2	-1.090	-1.145	51.0	4.0	0.429	0.291	instant_score	RMSProp	0.5	1000
-15.5	-2.311	-2.397	54.0	6.0	0.520	0.219	instant_score	Adam	0.8	1000
-16.1	-2.321	-2.396	45.0	4.0	0.505	0.220	instant_score	Adam	0.8	2000
-16.3	-2.326	-2.415	46.0	2.0	0.437	0.215	instant_score	RMSProp	0.8	500
-16.5	-2.330	-2.399	40.0	4.0	0.483	0.222	instant_score	RMSProp	0.8	2000
-16.9	-2.338	-2.406	43.0	5.0	0.469	0.209	instant_score	Adam	0.8	500
-17.0	-2.341	-2.396	43.0	4.0	0.437	0.224	instant_score	RMSProp	0.8	1000
-258.6	-7.142	-7.418	48.0	2.0	3.981	3.864	instant_score	RMSProp	1.0	1000
-264.5	-7.261	-7.478	42.0	4.0	4.038	3.854	instant_score	RMSProp	1.0	500
49.4	-0.919	-1.155	115.0	6.0	0.760	0.315	instant_cosine	Adam	0.5	2000
45.3	-0.993	-1.145	86.0	4.0	0.656	0.292	instant_cosine	RMSProp	0.5	1000
45.0	-0.998	-1.169	87.0	1.0	0.561	0.320	instant_cosine	RMSProp	0.5	500
44.8	-1.001	-1.130	71.0	3.0	0.591	0.274	instant_cosine	Adam	0.5	1000
41.0	-1.070	-1.119	42.0	2.0	0.426	0.258	instant_cosine	RMSProp	0.5	2000
40.6	-1.078	-1.118	56.0	2.0	0.389	0.263	instant_cosine	Adam	0.5	500
-15.0	-2.295	-2.388	50.0	6.0	0.521	0.224	instant_cosine	Adam	0.8	500
-15.3	-2.300	-2.398	54.0	6.0	0.522	0.231	instant_cosine	Adam	0.8	1000
-15.6	-2.308	-2.402	51.0	4.0	0.519	0.204	instant_cosine	RMSProp	0.8	2000
-16.0	-2.316	-2.409	55.0	7.0	0.473	0.226	instant_cosine	RMSProp	0.8	1000
-16.4	-2.323	-2.407	43.0	5.0	0.496	0.226	instant_cosine	Adam	0.8	2000
-17.3	-2.341	-2.398	39.0	4.0	0.421	0.215	instant_cosine	RMSProp	0.8	500
-243.5	-6.871	-7.090	45.0	3.0	3.903	3.855	instant_cosine	Adam	1.0	500
-245.5	-6.910	-7.658	46.0	2.0	3.869	3.806	instant_cosine	Adam	1.0	1000
-253.3	-7.066	-7.284	36.0	5.0	3.932	3.868	instant_cosine	RMSProp	1.0	1000
-257.9	-7.158	-7.595	43.0	5.0	3.990	3.808	instant_cosine	Adam	1.0	2000
-259.5	-7.288	-7.190	38.0	2.0	3.961	3.898	instant_cosine	RMSProp	1.0	2000
-260.8	-7.239	-7.216	33.0	3.0	3.988	3.813	instant_cosine	RMSProp	1.0	500
68.2	-0.574	-0.685	68.0	4.0	0.582	0.392	improve_score	RMSProp	0.5	2000
68.0	-0.578	-0.681	61.0	5.0	0.504	0.392	improve_score	Adam	0.5	1000
66.8	-0.600	-0.688	57.0	4.0	0.556	0.407	improve_score	Adam	0.5	500
66.7	-0.602	-0.696	64.0	1.0	0.555	0.401	improve_score	RMSProp	0.5	1000
66.4	-0.607	-0.685	76.0	3.0	0.478	0.396	improve_score	RMSProp	0.5	500
66.1	-0.612	-0.701	75.0	2.0	0.607	0.419	improve_score	Adam	0.5	2000
31.1	-1.378	-1.502	60.0	5.0	0.537	0.220	improve_score	Adam	0.8	1000
30.4	-1.393	-1.502	64.0	6.0	0.523	0.230	improve_score	RMSProp	0.8	1000
29.7	-1.406	-1.511	53.0	4.0	0.462	0.217	improve_score	Adam	0.8	500
29.6	-1.409	-1.501	53.0	2.0	0.423	0.198	improve_score	RMSProp	0.8	500
29.1	-1.417	-1.507	41.0	3.0	0.438	0.201	improve_score	RMSProp	0.8	2000
28.9	-1.422	-1.511	45.0	3.0	0.472	0.204	improve_score	Adam	0.8	2000
-143.5	-4.869	-5.148	46.0	2.0	2.741	2.589	improve_score	Adam	1.0	2000
-147.9	-4.958	-5.200	52.0	2.0	2.821	2.639	improve_score	RMSProp	1.0	500
-150.3	-5.006	-5.320	56.0	4.0	2.869	2.618	improve_score	RMSProp	1.0	1000
-151.0	-5.020	-5.305	39.0	2.0	2.710	2.600	improve_score	Adam	1.0	1000
-152.3	-5.047	-5.296	42.0	2.0	2.769	2.620	improve_score	RMSProp	1.0	2000
-154.9	-5.097	-5.193	49.0	9.0	2.769	2.626	improve_score	Adam	1.0	500
71.6	-0.511	-0.679	87.0	7.0	0.690	0.400	improve_cosine	RMSProp	0.5	2000
67.0	-0.594	-0.706	73.0	1.0	0.573	0.410	improve_cosine	Adam	0.5	500
66.9	-0.595	-0.688	42.0	3.0	0.433	0.397	improve_cosine	RMSProp	0.5	1000
66.4	-0.604	-0.677	44.0	2.0	0.443	0.387	improve_cosine	RMSProp	0.5	500
65.6	-0.619	-0.683	50.0	3.0	0.491	0.402	improve_cosine	Adam	0.5	2000
65.0	-0.629	-0.706	51.0	2.0	0.462	0.422	improve_cosine	Adam	0.5	1000
29.6	-1.399	-1.503	53.0	3.0	0.437	0.200	improve_cosine	Adam	0.8	2000
29.5	-1.402	-1.504	50.0	1.0	0.481	0.209	improve_cosine	RMSProp	0.8	1000
29.3	-1.406	-1.497	48.0	7.0	0.429	0.209	improve_cosine	RMSProp	0.8	2000
29.2	-1.408	-1.505	48.0	1.0	0.426	0.196	improve_cosine	Adam	0.8	500
28.4	-1.424	-1.501	49.0	6.0	0.416	0.229	improve_cosine	RMSProp	0.8	500
28.3	-1.424	-1.499	45.0	6.0	0.384	0.215	improve_cosine	Adam	0.8	1000
-150.4	-4.998	-5.315	44.0	3.0	2.765	2.625	improve_cosine	RMSProp	1.0	2000
-151.7	-5.023	-5.330	45.0	3.0	2.753	2.631	improve_cosine	Adam	1.0	2000
-154.2	-5.074	-5.135	44.0	6.0	2.752	2.613	improve_cosine	RMSProp	1.0	500
-155.4	-5.098	-5.127	47.0	6.0	2.783	2.633	improve_cosine	RMSProp	1.0	1000
-155.7	-5.105	-5.196	43.0	4.0	2.781	2.616	improve_cosine	Adam	1.0	500
-159.8	-5.186	-5.208	35.0	3.0	2.767	2.622	improve_cosine	Adam	1.0	1000

Table D.1: Fourth model grid search results, always validating only on instances with hops

Appendix E

Colophon

The main parts of the code including Agent and Environment design along with Learning Algorithm are available here. The full code for this project is available upon request: vsch@protonmail.com

Python	3.4.3
Tensorflow	1.1.0
Jack The Reader[1]	0.1.0
Whoosh	2.7.4
Spacy	1.9.0
Numpy	1.13.1
Pandas	0.20.2
Instance Type	c4.4xlarge
AMI	AWS Deep Learning
OS	Ubuntu 14.04.5
Lines	6851
Imports	203
Classes	17
Decorators	28
Properties	13
Functions	306
Conditions	704
Loops	494
With statements	100

Table E.1: Software used and code statistics.

[1]Jack the Reader is a knowledge base completion and question answering framework developed by the UCL Machine Reading group.

Bibliography

[1] Y. Adi, E. Kermany, Y. Belinkov, O. Lavi, and Y. Goldberg. Fine-grained Analysis of Sentence Embeddings Using Auxiliary Prediction Tasks. pages 1–13, 2016.

[2] H. Bast and I. Weber. Type Less, Find More: Fast Autocompletion Search with a Succinct Index. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 364–371, New York, NY, USA, 2006. ACM.

[3] J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *Journal ofMachine Learning Research*, 13:281–305, 2012.

[4] C. Buck, J. Bulian, M. Ciaramita, A. Gesmundo, N. Houlsby, W. Gajewski, and W. Wang. Ask the Right Questions: Active Question Reformulation with Reinforcement Learning. pages 1–15, 2017.

[5] Q. Chen, X. Zhu, Z. Ling, S. Wei, H. Jiang, and D. Inkpen. Enhanced LSTM for Natural Language Inference. (September), 2016.

[6] G. Durrett and D. Klein. A Joint Model for Entity Analysis : Coreference , Typing , and Linking. *Transactions of the Association for Computational Linguistics*, 2:477–490, 2014.

[7] T. Graepel. Deep Learning Lecture 2. *UCL MSc in Machine Learning*, 2017.

[8] A. C. Graesser, S. Lu, G. T. Jackson, H. H. Mitchell, M. Ventura, A. Olney, and M. M. Louwerse. AutoTutor: A tutor with dialogue in natural language.

Behavior Research Methods, Instruments, {&} Computers, 36(2):180–192, 2004.

[9] A. Graves. Supervised Sequence Labelling with Recurrent Neural Networks. *Image Rochester NY*, page 124, 2012.

[10] A. Graves. Generating Sequences with Recurrent Neural Networks. *Technical Reports*, pages 1–43, 2013.

[11] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf. Deep Reinforcement Learning with an Action Space Defined by Natural Language. *Acl*, pages 1–15, 2016.

[12] K. M. Hermann, T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching Machines to Read and Comprehend. pages 1–14, 2015.

[13] D. Hewlett, A. Lacoste, L. Jones, I. Polosukhin, A. Fandrianto, J. Han, M. Kelcey, and D. Berthelot. WikiReading: A Novel Large-scale Language Understanding Task over Wikipedia. pages 1535–1545, aug 2016.

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints*, pages 1–18, 2012.

[15] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Networks*, pages 237–243, 2001.

[16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1–32, 1997.

[17] I.-A. Hosu and T. Rebedea. Playing Atari Games with Deep Reinforcement Learning and Human Checkpoint Replay. page 6, 2016.

[18] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Arxiv*, 2015.

[19] D. Jurafsky. Question Answering. 2012.

[20] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A Convolutional Neural Network for Modelling Sentences. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, 2014.

[21] P. H. Kanani and A. K. McCallum. Selecting actions for resource-bounded information extraction using reinforcement learning. *Proceedings of the fifth ACM international conference on Web search and data mining WSDM 12*, page 253, 2012.

[22] O. Z. Khan and R. Sarikaya. Making Personal Digital Assistants Aware of What They Do Not Know. ISCA, 2016.

[23] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. pages 1–15, 2014.

[24] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-Normalizing Neural Networks. 2017.

[25] Y. Li. Deep Reinforcement Learning: An Overview. *arXiv:1701.07274 [cs]*, pages 1–66, 2017.

[26] M.-T. Luong, H. Pham, and C. D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *Emnlp*, (September):11, 2015.

[27] F. H. Mathis. A Generalized Birthday Problem. *SIAM Rev.*, 33(2):265–270, 1991.

[28] T. Mikolov, Q. V. Le, and I. Sutskever. Exploiting Similarities among Languages for Machine Translation. *CoRR*, abs/1309.4, 2013.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. pages 1–9, dec 2013.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, feb 2015.

[31] C. Monz. From document retrieval to question answering. 2003.

[32] A. Moull. Exploring applications of Machine Learning to the Law. 2017.

[33] D. Nadeau. A survey of named entity recognition and classification. *Linguisticae Investigationes*, (30):3–26., 2007.

[34] K. Narasimhan, A. Yala, and R. Barzilay. Improving Information Extraction by Acquiring External Evidence with Reinforcement Learning. 2016.

[35] M. A. Nielsen. Neural Networks and Deep Learning. 2015.

[36] R. Nogueira and K. Cho. Task-Oriented Query Reformulation with Reinforcement Learning. 2017.

[37] R. Pascanu, K. Cho, and Y. Bengio. On the Number of Linear Regions of Deep Neural Networks. *Nips*, pages 1–12, 2014.

[38] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[39] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. (ii), jun 2016.

[40] C. J. V. Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[41] S. Robertson and H. Zaragoza. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009.

[42] F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962.

[43] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized Experience Replay. *arXiv*, pages 1–23, 2015.

[44] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional Attention Flow for Machine Comprehension. pages 1–13, 2016.

[45] P. Simard, P. Frasconi, and Y. Bengio. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2), 1994.

[46] R. F. Simmons, S. Klein, and K. McConlogue. Indexing and dependency logic for answering English questions. *Journal of the Association for Information Science and Technology*, 15(3):196–204, 1964.

[47] A. Singhal. Introducing the Knowledge Graph: things, not strings, 2012.

[48] I. Sutskever. Training Recurrent neural Networks. *PhD thesis*, page 101, 2013.

[49] R. S. Sutton and A. G. Barto. Reinforcement Learning : An Introduction. 2012.

[50] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. *arXiv:1509.06461 [cs]*, (2):1–5, 2015.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. 2017.

[52] D. Weissenborn, G. Wiese, and L. Seiffe. Making Neural QA as Simple as Possible but not Simpler. 2017.

[53] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, and T. Mikolov. Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks. 2015.

[54] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu. Towards Universal Paraphrastic Sentence Embeddings. pages 1–19, 2015.

[55] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical Attention Networks for Document Classification. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[56] X. Zhang, J. Zhao, and Y. LeCun. Character-level Convolutional Networks for Text Classification. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pages 3057–3061, 2015.

[57] H. Zhao, Z. Lu, and P. Poupart. Efficient Estimation of Word Representations in Vector Space. *IJCAI International Joint Conference on Artificial Intelligence*, 2015-Janua:4069–4076, 2015.