Question Answering as Text Metadata Extraction A Coattention-Based Approach

Felipe Kup
Stanford University
felipek@stanford.edu

Abstract

Most named entity recognition (NER) problems can be easily solved using current natural language processing (NLP) methods. A harder task involves choosing relevant entities among those identified based on their context. I tackle this problem using a question answering algorithm as an alternative tool for standard text metadata extraction tasks. To do that, I first train a model on the Stanford Question Answering Dataset (SQuAD). After trying multiple architectures, I opt for a model that uses a bidirectional LSTM encoder, a coattention layer and a fully-connected decoder. To illustrate the potential for such applications, I then propose a simple framework that involves using the trained QA model to extract a sequence of relevant (date, place) couples from Wikipedia biographies.

1 Introduction

Question answering (hereafter, QA) is a way of evaluating reading comprehension, a necessary skill in order to build more advanced NLP algorithms such as intelligent dialogue agents. QA has the advantage to be easier to evaluate than other tasks such as dialogue or summarization. However, many questions are only answerable with world knowledge. Therefore, in order to assess reading comprehension alone, a large branch of NLP research has focused on the a subset of QA known as closed QA, where questions are answerable based solely on the information contained in the corresponding passage, or context.

As a first part of this paper, I train a QA model on a large benchmark dataset (SQuAD), experimenting with different architectures and specifications. I then show that a QA model can a useful too for extracting metadata from raw text in a more end-to-end approach than standard NLP techniques.

2 Related work

2.1 Question answering literature

The main challenge within the field is to build an algorithm that is able to effectively answer questions based on a context. There is a very large literature on this particular problem. As an example of recent contribution, Weston et al. (2015) decompose QA tasks into different types of skills. This helps identifying the skills on which an algorithm performs more or less well, which allows for isolating specific aspects of QA where there is a larger margin for improvement. Examples include simple tasks such as yes/no questions, counting but also more complicated ones such as multiple supporting facts, simple negation or indefinite knowledge.

A second, important challenge in the closed QA research includes building a large enough dataset of (**q**uestion,**c**ontext,**a**nswer) triples, where the questions are answerable by humans and require only knowledge present in the context. For instance, one of the first well-known dataset that tackles most

CS230: Deep Learning, Winter 2018, Stanford University, CA.

of these issues is the MCTest (Richardson, 2013), which is a set of 660 (**q**,**c**,**a**) triples generated specifically in a way that requires little or no world knowledge, and "limited to those a young child would understand".

The existence of cleverly-designed, large benchmark datasets is a precondition for evaluating the performance of QA algorithms. Hence, these two strands of research go hand-in-hand. The SQuAD dataset represents one of the most recent and largest datasets that go in that direction.

2.2 SQuAD and state-of-the-art models

The Stanford Question Answering Dataset was introduced by Rajpurkar et al. (2016), and consists of 107,785 question-answer pairs based on context paragraphs, which were extracted from 536 high-quality Wikipedia articles. The dataset is thoroughly described and analyzed in section 3. The authors also build a strong logistic regression model, which achieves a 51% F1 score and a 40% EM (exact match[1]) score. According to the original paper, human performance is is 77.0% for the EM metric, and 86.8% for the F1 metric.

Since the first release of the dataset in 2016, there have been several models which achieved near-human performance, with some state-of-the-art models achieving superhuman performance according to the EM metric. For instance, my attention layer is inspired by the model architecture described in Xiong et al. (2016). The top 5 best-performing models are[2] all ensemble models. For instance, **r-net+ (ensemble)**, an ensemble, improved version of the original **r-net (single)** model (Natural Language Computing Group, 2017; Wang et al., 2017), achieves a score of 82.650% EM and 88.493% F1.

2.3 Generating trajectories from Wikipedia biographies

Wikipedia is arguably the most comprehensive source of information publicly available online. Nonetheless, it remains essentially an unstructured database in the form of text. There exist a variety of structured content datasets that store metadata extracted from Wikipedia articles. Some examples include Freebase[3](which has now migrated to Wikidata[4]) or DBpedia[5]. Using these databases, it is easy to compile a list of birth and death places for a number of selected individuals. However, the scope of these datasets remains limited to metadata that is relatively easy to extract. Birth or death places, for instance, are usually located at the very beginning of a biographical article's text (immediately after the person's name), as well as within the article's so-called Infobox.

These structured databases, however, do not contain information that requires more advanced extraction techniques, such as the list of all places visited by an individual. In the first attempt to build such a dataset, Gergaud et al. (2016) make use of all the GeoLinks[6] present in each article's main text to create a chronological, dated sequence of places visited by an individual - a "trajectory". They are able to compile a database of trajectories for 1,243,776 individuals and 7,184,575 locations associated with them throughout human history (3000BCE-2015AD).

Menini et al. (2017) make use of NLP modules to extract sequences of places explicitly associated with movements using biographies from 2,407 individuals (although they only identify movements for 1,283 of them). The model is able to pinpoint locations that are not associated to GeoLinks. However, it creates a false negative problem due to the limited list of movement frames selected by the authors. Moreover, according to the authors, the model is unable to deal with sentences with complex temporal structures. Finally, this approach is incomplete for extracting a full biographical trajectory, as it explicitly excludes places where individuals have lived if the sentence containing that information does not include a verb with movement.

---

[1] A metric that, according to the authors, "measures the percentage of predictions that match any of the ground truth answers exactly".

[2] As of today (March 22, 2018)

[3] `freebase.com`

[4] `https://dumps.wikimedia.org/wikidatawiki/`

[5] `http://wiki.dbpedia.org`

[6] GeoLinks are spatial hyperlinks present in an article, such as the name of a city, which includes geographical coordinates.

3 Data

My training data consists of 107,785 question-answer pairs based on 536 high-quality Wikipedia articles. The entire dataset was downloaded from the original paper's website (Rajpurkar et al., 2016), and was created using an crowdsourcing Internet marketplace (MTurk). Before feeding the data to the algorithm, I pre-process all examples by tokenizing and lowercasing them. Figure 2 displays two pre-processed question-answer pairs from the dataset, based on a same context.

The train/dev/test split is 80/10/10, which corresponds to around 80,000 and 10,000 examples for the training and dev sample. The test set is not available for download and is kept secret for evaluation purposes on the official leaderboard website (Rajpurkar et al., 2016).

Section A.1.2 in the Appendix displays plots of token and character length distribution for contexts, questions and answers in our training set. Those will be useful later when tuning hyperparameters, such as the maximum context length (see section 5).

4 Methods

4.1 Introduction

The model has three components: an RNN encoder, a coattention layer and an output layer. This is shown in detail in Fig. 12. For clarity, I follow the notation used in cs2 (2017-2018). Consider a training example consisting of one context, one question and one answer span (i.e. a couple of integers indicating the starting and word indices of the answer within the context). I encode each word in the context and in the question using g-dimensional Glove embeddings[7] downloaded from the original paper's dedicated website (Pennington et al., 2014). Therefore, each question and context can be represented by a sequence of g-dimensional embedding vectors

$$(q_1^*, ..., q_M^*), (c_1^*, ..., c_N^*) \in \mathbb{R}^g \tag{1}$$

where M and N are respectively the maximum context and question length, two hyperparameters. Notice that contexts with less than M words are still represented as M-by-g matrices, although the last rows of the matrix are padded with the same, g-dimensional representation of <pad>. Contexts with more than M words are truncated at the M-th word. The same applies to questions (N).

4.2 RNN encoder layer

First, the context and the question GloVe embeddings are fed into the same RNN encoder layer, which is a 1-layer bidirectional LSTM with hidden output dimension h. The context output forward and backward hidden states $(\vec{c_1}, ..., \vec{c_M}) \in \mathbb{R}^h$ and and $(\overleftarrow{c_1}, ..., \overleftarrow{c_M}) \in \mathbb{R}^h$ are concatenated into context hidden states

$$c_i = [\vec{c_i}, \overleftarrow{c_i}] \in \mathbb{R}^{2g},\ i = \{1, ..., N\} \tag{2}$$

and similarly, we obtain question hidden states

$$q_j = [\vec{q_j}, \overleftarrow{q_j}] \in \mathbb{R}^{2g}, \forall j = \{1, ..., M\} \tag{3}$$

4.3 Coattention layer

The question hidden states are then fed successively into a linear layer with a tanh activation function, obtaining projected question hidden states

$$q'_j = tanh(\mathbf{W}q_j + \mathbf{b}) \in \mathbb{R}^{2h},\ \forall j = \{1, ..., M\} \tag{4}$$

Successively, *sentinel* vectors c_0 and $q_0 \in \mathbb{R}^{2h}$ are appended at the end of the context hidden states and the projected question hidden states, giving us $\mathbf{c} \in \mathbb{R}^{(N+1)\times(2h)}$ and $\mathbf{q'} \in \mathbb{R}^{(M+1)\times(2h)}$. We then obtain the affinity matrix L by multiplying those two new matrices as follows:

$$\mathbf{L} = \mathbf{c}\mathbf{q'}^T,\ \in \mathbb{R}^{(N+1)\times(M+1)} \tag{5}$$

[7] in my implementation I chosse $g = 100$

Unlike the basic, one-way dot-product attention in which contexts attend to questions (Q2C), here we use this affinity matrix to compute a **two-way** attention (Q2C and QC2) between the questions and the answer - a concept pioneered by Seo et al. (2016):

$$\alpha^i = softmax(L_{i,:}) \in \mathbf{R}^{M+1} \tag{6}$$

$$\mathbf{a} = \alpha\ \mathbf{q}' \tag{7}$$

$$\beta^j = softmax(L_{:,j}) \in \mathbf{R}^{N+1} \tag{8}$$

$$\mathbf{b} = \beta\ \mathbf{c} \tag{9}$$

where equations 6 and 7 represent the Q2C attention and where equations 8 and 9 represent the C2Q attention. Here, however, there is an additional, *co-attention* computation, which involves multiplying the first N rows of the Q2C α matrix with the Q2C attention output matrix $\mathbf{b}$:

$$\mathbf{s} = \alpha_{\mathbf{1:N},:}\ \mathbf{b} \tag{10}$$

Finally, these second-level attention outputs s_i are then concatenated with the C2Q attention outputs a_i and fed into a bidirectional LSTM to generate coattention encodings

$$(u_i, ..., u_N) = BLSTM([s_1, a_1], ..., [s_N, a_N]) \in \mathbb{R}^{4h} \tag{11}$$

4.4 Output layer

Finally, these coattention encodings are then fed into a fully- connected layer with a ReLU activation function

$$b'_i = ReLU(\mathbf{W_{FC}}\ \mathbf{u_i} + \mathbf{v_{FC}}), \in \mathbf{R}^{2h} \tag{12}$$

followed by a linear layer whose output is finally fed to two softmax activation functions, one for the start and one for the end locations:

$$logits_i^{start} = \mathbf{w}_{start}^T b'_i + u_{start} \tag{13}$$

$$p^{start} = softmax(logits^{start}) \tag{14}$$

and similarly for the ending. The loss function is obtained by adding the cross-entropy for these start and end locations:

$$Loss = -log\ p^{start}(i_{start}) - log\ p^{end}(i_{end}) \tag{15}$$

where i_{start} and i_{end} are the gold start and end locations in the context.

5 Experiments and Results

My baseline model used a bidirectional GRU RNN encoder and a simple, dot-product attention layer, with learning rate 0.001, batch size 100 and context length $N = 600$. The training was performed on a AWS Deep Learning instance (single GPU, p2.xlarge). The Python (2) code was implemented in tensorflow 1.4.1 (ten, 2017).

Changing from a GRU to a LSTM encoder slightly improved performance (see Table 1 and Fig. 5). Given the limited time and resources (i.e. AWS credits) available, an essential part of the project involved improving training efficiency. One idea was to reduce the maximum context length size from 600 to 300, given that 300 is still between the 95th and 99th percentile in terms of context word length in our training sample (cf. 3c and Table 2). Indeed, this halved the running time without affect performance (cf.1 and 6). I also tried experimenting with different learning rates on a logarithmic scale (e.g. 0.01 and 0.0001). A learning rate of 0.01 improved training efficiency but at a high performance cost since the very beginning of training (cf. 7), whereas a learning rate of 0.0001 significantly reduced efficiency and I decided it was not worth experimenting it with 20,000 iterations.

Changing the architecture of the middle layer from a simple, dot-product attention to a coattention model dramatically increased performance (cf. Table 1 and Fig 8, above). Experimenting with mini batch sizes different from the baseline (=100) did not yield any improvement in performance: Fig. 9 shows a performance comparison for the first 7,000 iterations of models using a batch size of $2^5 = 64$ (blue) and $2^6 = 128$ (orange), as opposed to 100.

Model (hyperparameters)	Run time, h	EM train	F1 train	EM dev	F1 dev
Simple attention					
BGRU encoder, $N = 600$	6h 52 min	0.53	0.64	0.29	0.40
BLSTM encoder, $N = 600$	6h 56 min	0.59	0.69	0.31	0.42
BLSTM encoder, $N = 300$	**3h 37 min**	0.55	0.66	0.31	0.42
Co-attention					
BLSTM encoder, $N = 300$	8h 53 min	**0.77**	**0.88**	**0.49**	**0.64**

Table 1: Results for different architectures and specification (1 GPU, 20K iterations)

Instead of focusing on qualitatively analysing which types of questions the algorithm failed to answer correctly, I decided to use the QA model for an entirely different task: text metadata extraction.

6 Application: qualitative analysis

To illustrate how a QA method can successfully exploit such context, consider the case of the Wikipedia page of Ray Charles. Figures 10 and 11 display the trajectories outputted by two existing implementations of automatic extraction of Wikipedia biographical trajectories. First, we consider the algorithm used in Gergaud et al. (2016): it extracts a sequence of locations using only places mentioned in the article that have a Wikipedia Geolink. Moreover, the algorithm only extracts locations, but not dates. Finally, they assume that the links in the text appear in chronological order. Second, consider the extraction method used in Menini et al. (2017): it only extracts a (date, location) couple if it is accompanied by a verb of movement.

Now, consider our QA algorithm. It takes as input a context sentence from Wikipedia, which must contain: (i) a reference to the individual, (ii) a place and (iii) a date. These can be identified using standard NER algorithms. Then, based on the year identified in the sentence, it asks the following question: "where was {person name} in {year}?". Figure 1, below, shows two examples of locations that are not captured by either of the two previously mentioned algorithms, both because of the absence of a Wikipedia GeoLink as well as the absence of a verb indicating movement. However, they are accurately extracted using our QA algorithm.

---

by late 1961 , charles had expanded his small road ensemble to a full-scale big band , partly as a response to increasing royalties and touring fees , becoming one of the few black artists to cross over into mainstream pop with such a level of creative control . this success , however , came to a momentary halt during a concert tour in November 1961 , when a police search of charles 's hotel room in **indianapolis , indiana** , led to the discovery of heroin in the medicine cabinet .

where was charles in 1961 ?
indianapolis , indiana

---

charles had always played piano for other people , but he was keen to have his own band . he decided to leave florida for a large city , and , considering chicago and new york city too big , followed his friend gossie mckee to **seattle , washington** , in march 1948 , knowing that the biggest radio hits came from northern cities . here he met and befriended , under the tutelage of robert blackwell , a 15-year-old quincy Jones .

where was charles in 1948 ?
seattle , washington

Figure 1: Two context sentences from Ray Charles Wikipedia biography with the algorithm's predicted location and date of visit

7 Conclusion and future Work

This paper is an example of case in which a model trained at one particular task can excel at a very different, although related task. As a next step, tow possible directions can be considered. First, one focus on improving my QA model performance on the SQuAD dataset by experimenting with other

types of network architectures, as well as adding regularization (the model appears to overfit the training set after 10-15k iterations), trying an ensemble model, etc. Alternatively, one could consider that the performance of the model to be satisfactory, especially since the model was trained on "hard" questions, whereas the task of retrieving a relevant location is a relatively easier one. If I were to choose the second research avenue, my first step would be to compile a human-labeled dev set of (articles, trajectories), in order to further assess the performance of the QA module and determine the next steps for improving extraction. Then, depending on resource and time constraints, I would eventually compile a larger training set (hundreds, or maybe a thousand biographies), and consider implementing an RNN model, which would iterate across sentences in the article and where each RNN cell would be fed not only with a (context,question) pair, but also with the predicted (year, location) output of the previous cell. This could, for example, allow the model to discard anomalous dates (e.g. references to distant past or future events).

References

Tensorflow 1.4.1, December 2017. URL https://www.tensorflow.org.

Cs 224n default final project: Question answering, 2017-2018. URL http://web.stanford.edu/class/cs224n/default_project/default_project_v2.pdf.

Gergaud, Olivier, Laouénan, Morgane, and Wasmer, Etienne. A brief history of human time: Exploring a database of "notable people". Sciences Po Economics Discussion Papers 2016-03, Sciences Po Departement of Economics, 2016. URL https://EconPapers.repec.org/RePEc:spo:wpecon:info:hdl:2441/h4tv2ee028raq0ib4dabsqqei. See dedicated website: http://brief-history.eu (Accessed: 2018-03-22).

Menini, Stefano, Sprugnoli, Rachele, Moretti, Giovanni, Bignotti, Enrico, Tonelli, Sara, and Lepri, Bruno. "ramble on: Tracing movements of popular historical figures". *Proceedings of EACL 2017,Conference of the European Chapter of the Association for Computational Linguistics*, 2017. URL http://aclweb.org/anthology/E/E17/E17-3020.pdf. See dedicated website https://dh.fbk.eu/technologies/rambleon and online navigator http://dhlab.fbk.eu/rambleon/ ((Both accessed: 2018-03-22)).

Natural Language Computing Group, Microsoft Research Asia. R-net: Machine reading comprehension with self-matching networks. May 2017. URL https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf.

Pennington, Jeffrey, Socher, Richard, and Manning, Christopher D. Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–154, October 2014. URL https://nlp.stanford.edu/projects/glove/.

Rajpurkar, Pranav, Zhang, Jian, Lopyrev, Konstantin, and Liang, Percy. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL http://arxiv.org/abs/1606.05250. See dedicated website https://rajpurkar.github.io/SQuAD-explorer/ and blog post https://rajpurkar.github.io/mlx/qa-and-squad/ (Both accessed: 2018-03-22).

Richardson, Matthew. Mctest: A challenge dataset for the open-domain machine comprehension of text. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 193–203, October 2013. URL https://www.microsoft.com/en-us/research/publication/mctest-challenge-dataset-open-domain-machine-comprehension-text/.

Seo, Min Joon, Kembhavi, Aniruddha, Farhadi, Ali, and Hajishirzi, Hannaneh. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016. URL http://arxiv.org/abs/1611.01603.

Wang, Wenhui, Yang, Nan, Wei, Furu, Chang, Baobao, and Zhou, Ming. Gated self-matching networks for reading comprehension and question answering. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pp. 189–198, July 2017.

Weston, Jason, Bordes, Antoine, Chopra, Sumit, and Mikolov, Tomas. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015. URL `http://arxiv.org/abs/1502.05698`.

Xiong, Caiming, Zhong, Victor, and Socher, Richard. Dynamic coattention networks for question answering. 2016. URL `https://arxiv.org/pdf/1611.01604v4.pdf`.

A Appendix

A.1 Figures and tables

A.1.1 Dataset examples

---

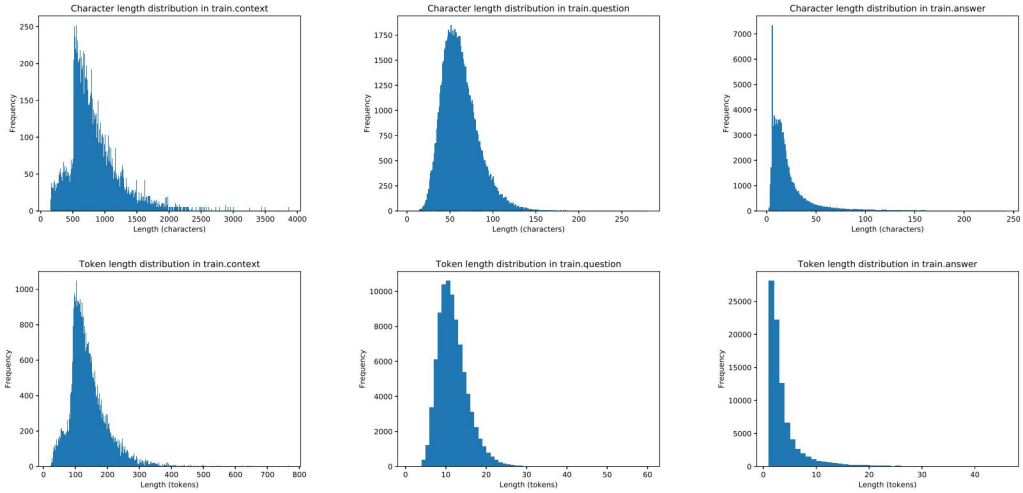
a deterministic turing machine is the most basic turing machine , which uses a fixed set of **rules** to determine its future actions . a probabilistic turing machine is a deterministic turing machine with an extra supply of random bits . the ability to make probabilistic decisions often helps algorithms solve problems more efficiently . algorithms that use random bits are called randomized algorithms . **a non-deterministic turing machine** is a deterministic turing machine with an added feature of non-determinism , which allows a turing machine to have multiple possible future actions from a given state . one way to view non-determinism is that the turing machine branches into many possible computational paths at each step , and if it solves the problem in any of these branches , it is said to have solved the problem . clearly , this model is not meant to be a physically realizable model , it is just a theoretically interesting abstract machine that gives rise to particularly interesting complexity classes . for examples , see non-deterministic algorithm .

what fixed set of factors determine the actions of a deterministic turing machine ?
rules

what type of turing machine is capable of multiple actions and extends into a variety of computational paths ?
a non-deterministic turing machine

Figure 2: Pre-processed examples from the dataset

A.1.2 Training set analysis: token and character length



(a) Context (b) Question (c) Answer

Figure 3: Character and token length distribution for training set contexts, questions and answers

	Contexts	Questions	Answers
90th percentile	211	16	7
95th percentile	245	18	11
99th percentile	325	23	21
Max	766	60	46

Table 2: Token length percentiles

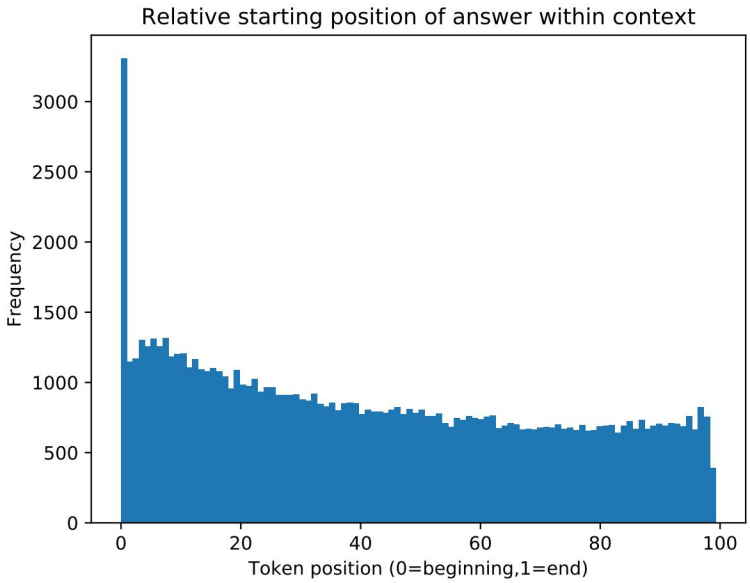


Figure 4: Relative position of first answer token within context

A.1.3 Experiments: Tensorboard (20k iterations)

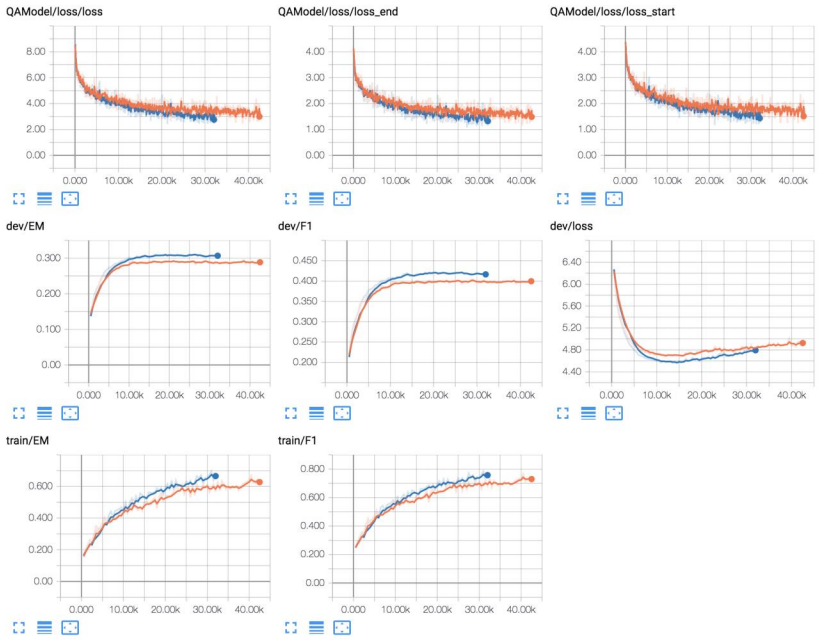


Figure 5: RNN encoder: bidirectional GRU (in orange) vs. bidirectional LSTM (in blue)

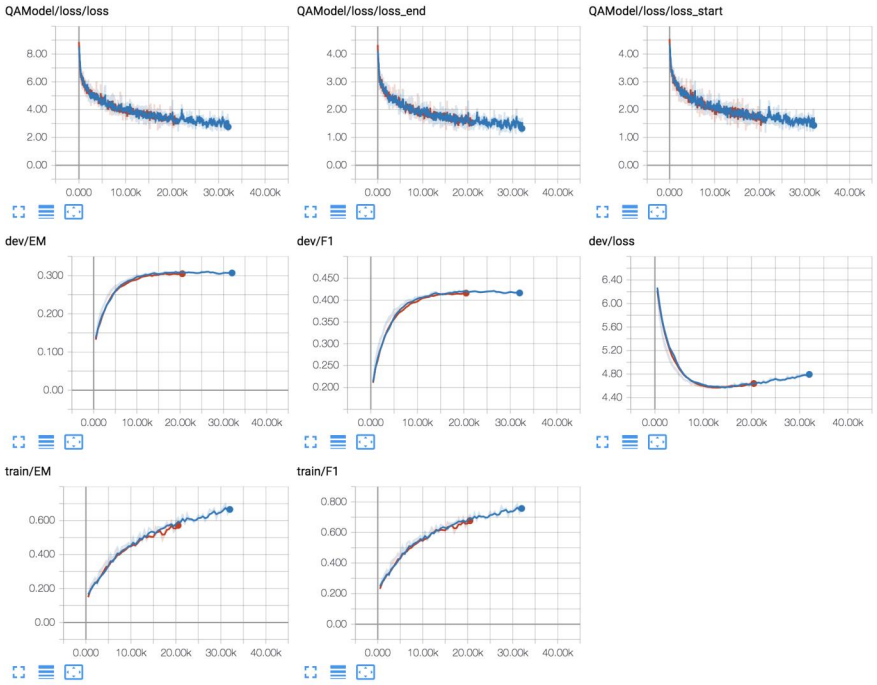


Figure 6: Bidirectional LSTM, N = 600 (in blue) vs. Bidirectional LSTM, N = 300 (in red). Note: the latter ran twice as fast

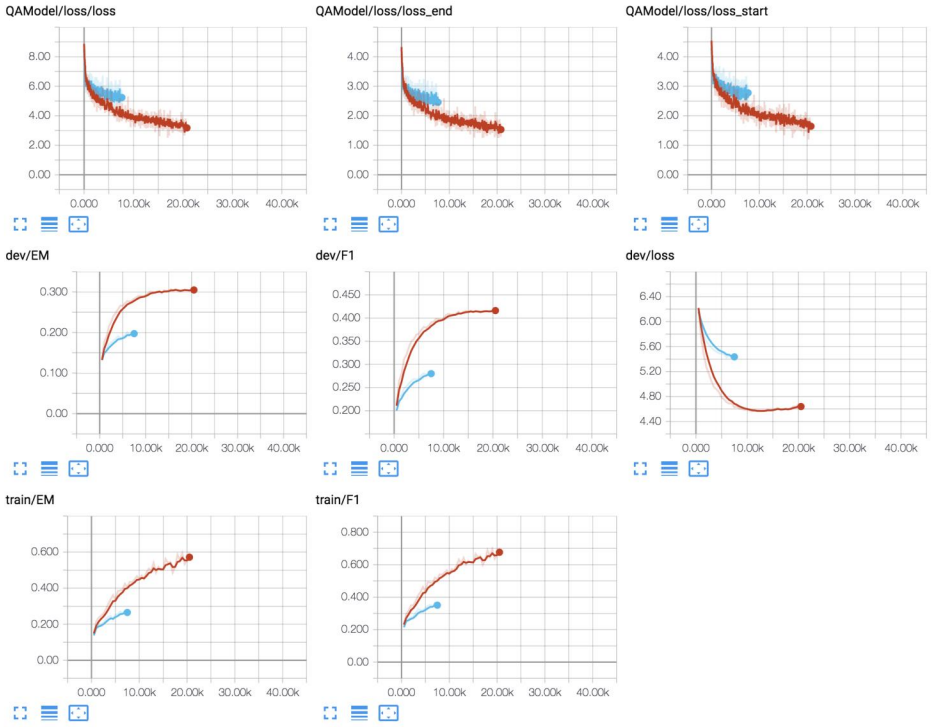


Figure 7: Experimenting with different constant learning rates did not yield any improvement in performance. For instance, here learning rate = 0.01 (light blue), as opposed to 0.001 in the baseline model (red).

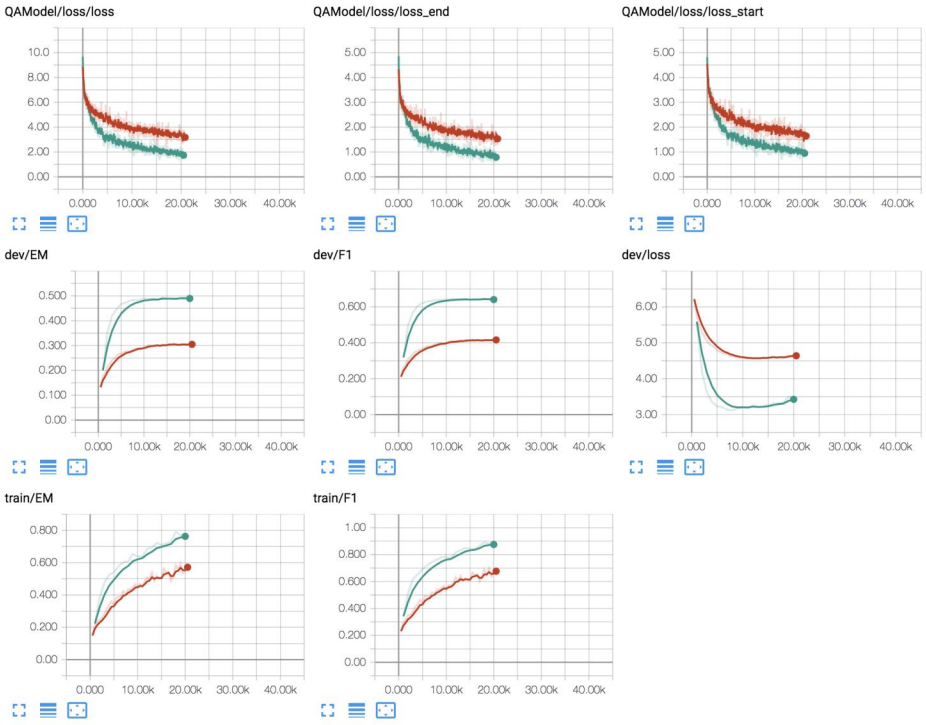


Figure 8: Changing the architecture of the attention layer from a basic dot-product attention to a coattention model drastically improves performance

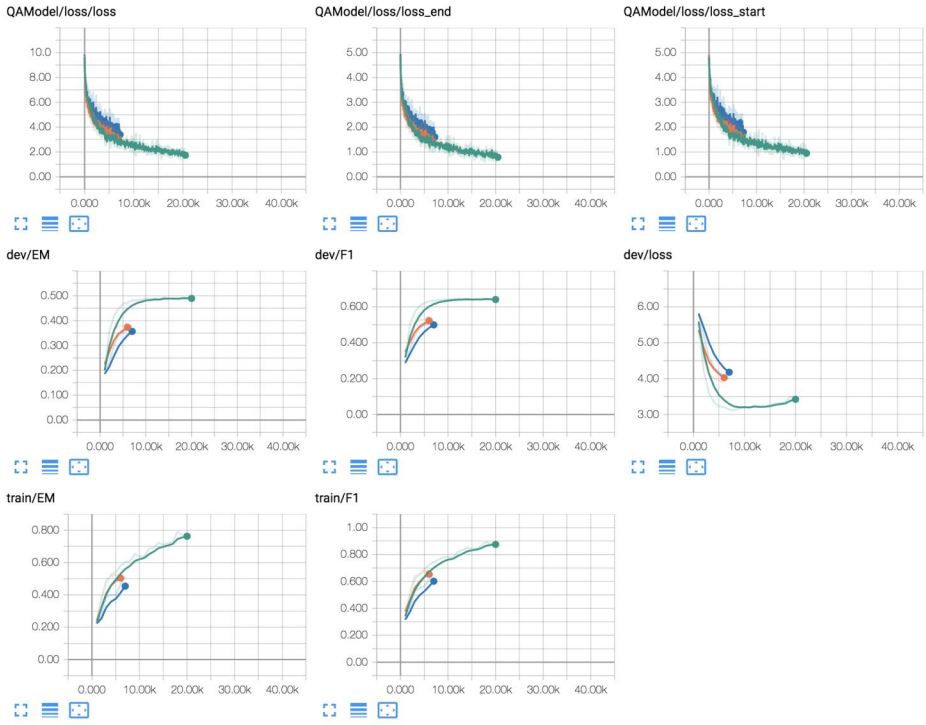


Figure 9: Experimenting with different batch sizes similar to the baseline (=100) did not yield any improvement in performance. For instance, here I try both $2^5 = 64$ (blue) and $2^6 = 128$ (orange), as opposed to 100 in the baseline model (green)

A.1.4 Generating biographical trajectories from Wikipedia articles: Ray Charles

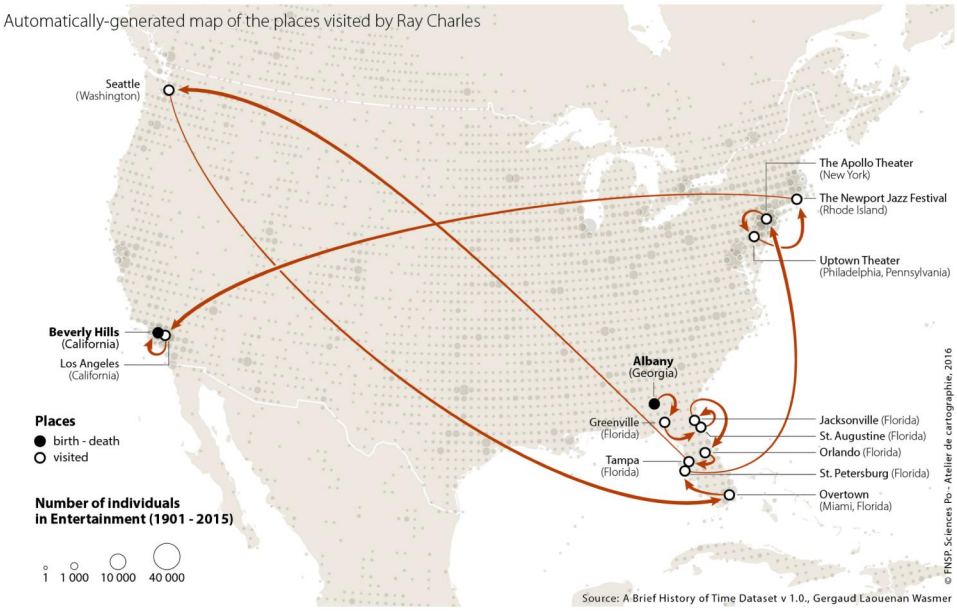


Figure 10: Trajectory obtained using the algorithm in Gergaud et al. (2016)

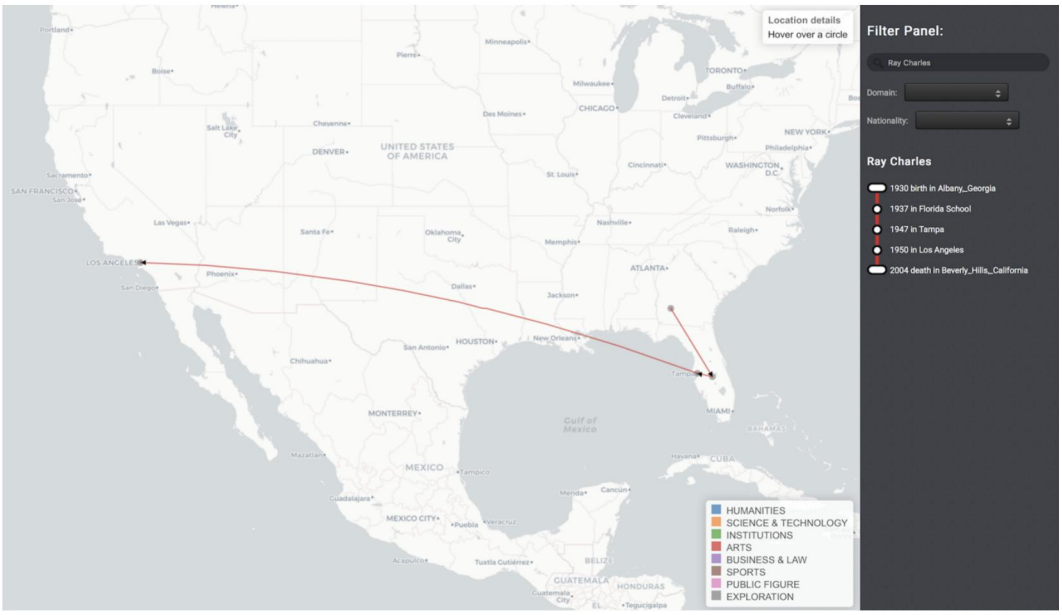


Figure 11: Trajectory obtained using the algorithm in Menini et al. (2017)

A.2 Model diagram

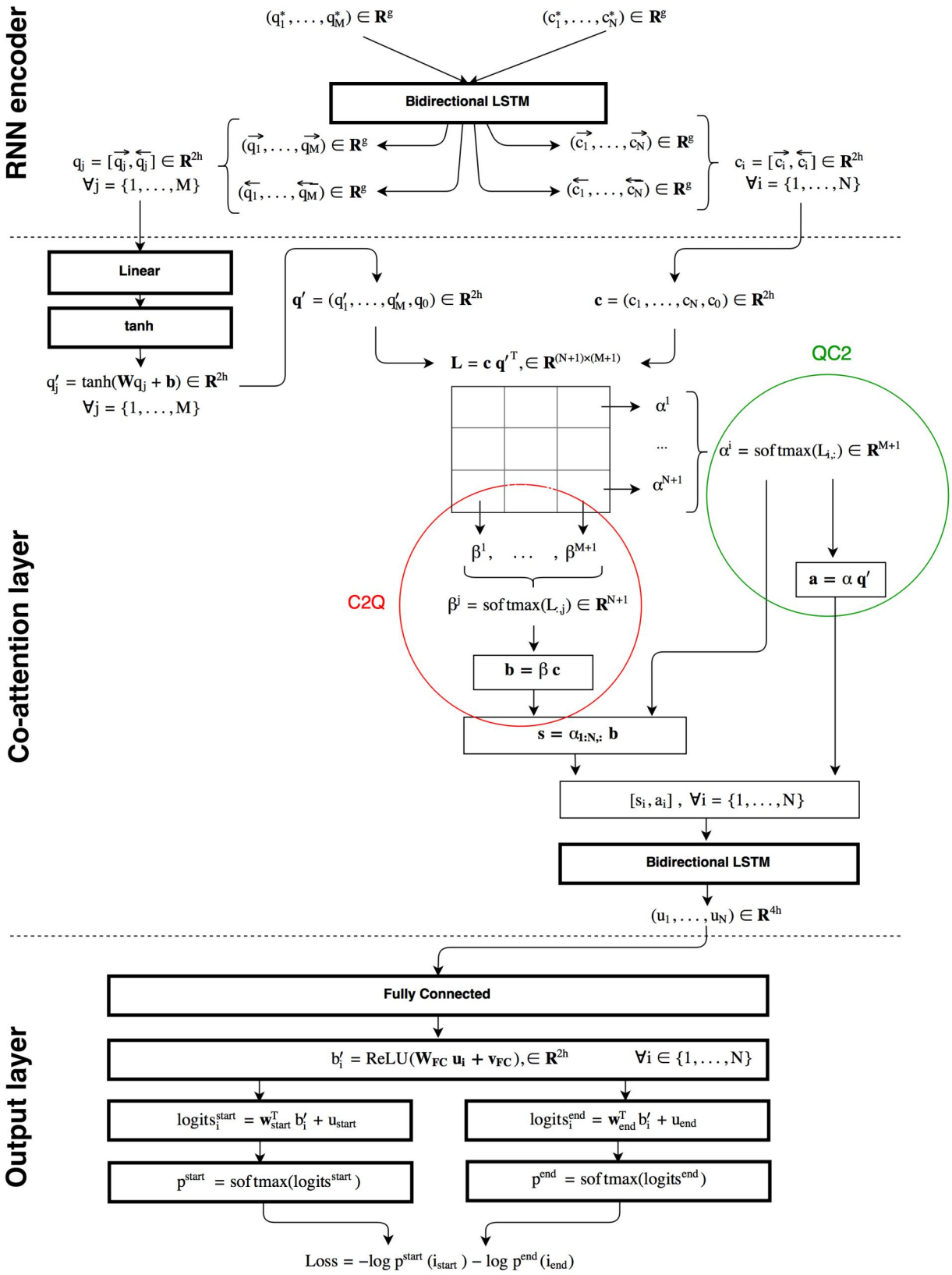


Figure 12: Diagram of the three-layer algorithm