VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Nguyen Minh Trang

ADVANCED DEEP LEARNING METHODS AND APPLICATIONS IN OPEN-DOMAIN QUESTION ANSWERING

MASTER THESIS

Major: Computer Science

HA NOI - 2019

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Nguyen Minh Trang

ADVANCED DEEP LEARNING METHODS AND APPLICATIONS IN OPEN-DOMAIN QUESTION ANSWERING

MASTER THESIS

Major: Computer Science

Supervisor: Assoc.Prof. Ha Quang Thuy

Ph.D. Nguyen Ba Dat

HA NOI - 2019

Abstract

Ever since the Internet has become ubiquitous, the amount of data accessible by information retrieval systems has increased exponentially. As for information consumers, being able to obtain a short and accurate answer for any query is one of the most desirable features. This motivation, along with the rise of deep learning, has led to a boom in open-domain Question Answering (QA) research. An open-domain QA system usually consists of two modules: retriever and reader. Each is developed to solve a particular task. While the problem of document comprehension has received multiple success with the help of large training corpora and the emergence of attention mechanism, the development of document retrieval in open-domain QA has not gain much progress. In this thesis, we propose a novel encoding method for learning question-aware self-attentive document representations. Then, these representations are utilized by applying pair-wise ranking approach to them. The resulting model is a Document Retriever, called QASA, which is then integrated with a machine reader to form a complete open-domain QA system. Our system is thoroughly evaluated using QUASAR-T dataset and shows surpassing results compared to other state-of-the-art methods.

***Keywords**: Open-domain Question Answering, Document Retrieval, Learning to Rank, Self-attention mechanism.*

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Assoc. Prof. Ha Quang Thuy for the continuous support of my Master study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

I would also like to thank my co-supervisor Ph.D. Nguyen Ba Dat who has not only provided me with valuable guidance but also generously funded my research.

My sincere thanks also goes to Assoc. Prof. Chng Eng-Siong and M.Sc. Vu Thi Ly for offering me the summer internship opportunities in NTU, Singapore and leading me working on diverse exciting projects.

I thank my fellow labmates in KTLab: M.Sc. Le Hoang Quynh, B.Sc. Can Duy Cat, B.Sc. Tran Van Lien for the stimulating discussions, and for all the fun we have had in the last two years.

Last but not the least, I would like to thank my parents for giving birth to me at the first place and supporting me spiritually throughout my life.

Declaration

I declare that the thesis has been composed by myself and that the work has not be submitted for any other degree or professional qualification. I confirm that the work submitted is my own, except where work which has formed part of jointly-authored publications has been included.

My contribution and those of the other authors to this work have been explicitly indicated below. I confirm that appropriate credit has been given within this thesis where reference has been made to the work of others. The work presented in Chapter 3 was previously published in Proceedings of the 3rd ICMLSC as "QASA: Advanced Document Retriever for Open Domain Question Answering by Learning to Rank Question-Aware Self-Attentive Document Representations" by Trang M. Nguyen (myself), Van-Lien Tran, Duy-Cat Can, Quang-Thuy Ha (my supervisor), Ly T. Vu, Eng-Siong Chng. This study was conceived by all of the authors. My contributions include: proposing the method, carrying out the experiments, and writing the paper.

Master student

Nguyen Minh Trang

Table of Contents

Acronyms

Adam	Adaptive Moment Estimation
AoA	Attention-over-Attention
BiDAF	Bi-directional Attention Flow
BiLSTM	Bi-directional Long Short-Term Memory
CBOW	Continuous Bag-Of-Words
EL	Embedding Layer
EM	Exact Match
GA	Gated-Attention
IR	Information Retrieval
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
QA	Question Answering
QASA	Question-Aware Self-Attentive
QEL	Question Encoding Layer
R3	Reinforced Ranker-Reader
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network

SGD	Stochastic Gradient Descent
TF-IDF	Term Frequency – Inverse Document Frequency
TREC	Text Retrieval Conference

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Open-domain Question Answering

We are living in the Information Age where many aspects of our lives are driven by information and technology. With the boom of the Internet few decades ago, there is now a colossal amount of data available and this number continues to grow exponentially. Obtaining all of these data is one thing, how to efficiently use and extract information from them is one of the most demanding requirements. Generally, the activity of acquiring useful information from a data collection is called Information Retrieval (IR). A search engine, such as Google or Bing, is a type of IR. Search engines are extensively used that it is hard to imagine our lives today without them. Despite their applicability, current search engines and similar IR systems can only produce a list of relevant documents with respect to the user's query. To find the exact answer needed, users still have to manually examine these documents. Because of this, although IR systems have been handy, retrieving desirable information is still a time consuming process.

Question Answering (QA) system is another type of IR that is more sophisticated than search engines in terms of being a natural forms of human computer interaction [27]. The users can express their information needs in natural language instead of a series of keywords as in search engines. Furthermore, instead of a list of documents, QA systems try to return the most concise and coherent answers possible. With the vast amount of data nowadays, QA systems can reduce countless effort in retrieving information. Depending on usage, there are two types of QA: closed-domain and open-domain. Unlike closed-domain QA, which is re-

stricted to a certain domain and requires manually constructed knowledge bases, open-domain QA aims to answer questions about basically anything. Hence, it mostly relies on world knowledge in the form of large unstructured corpora, e.g. Wikipedia, but databases are also used if needed. Figure 1.1 shows an overview of an open-domain QA system.

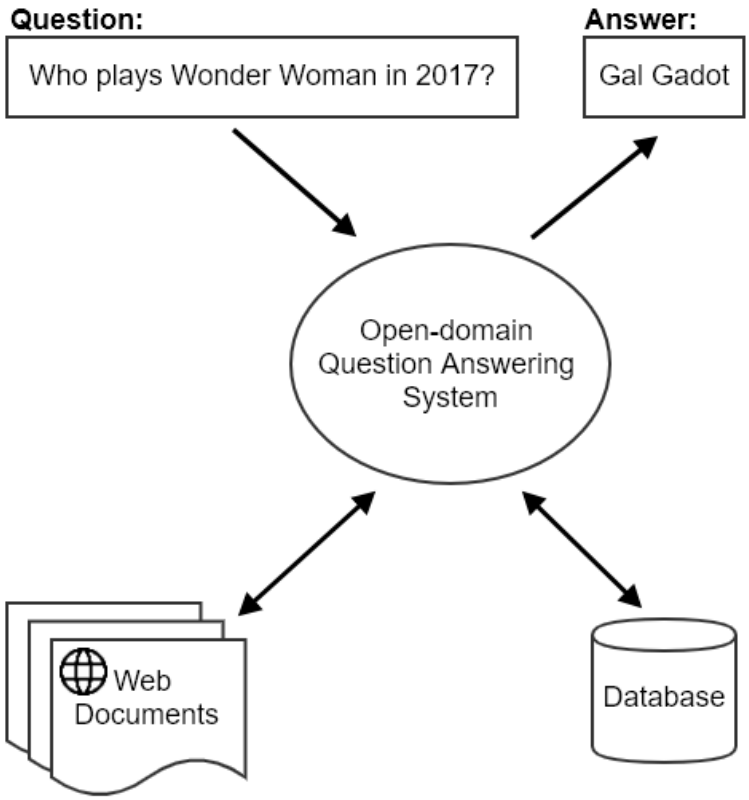

Figure 1.1: An overview of Open-domain Question Answering system.

The research about QA systems has a long history tracing back to the 1960s when Green et al. [20] first proposed BASEBALL. About a decade after that, Woods et al. [48] introduced LUNAR. Both of these systems are closed-domain and they use manually defined language patterns to transform the questions into structured database queries. Since then, knowledge bases and closed-domain QA systems had become dominant [27]. They allow users to ask questions about certain things but not all. Not until the beginning of this century that open-domain QA research has become popular with the launch of the annual Text Retrieval Conference (TREC) [44] started in 1999. Ever since, TREC competitions, especially the open-domain QA tracks, have progressed in size and complexity of the dataset provided, and evaluation strategies are improved. [36]. The attention is now shifting to open-domain QA and in recent years, the number of studies on the subject has increased exceedingly.

1.1.1 Problem Statement

In QA systems, the questions are natural language sentences and there are a many types of them based on their semantic categories such as factoid, list, causal, confirmation, hypothetical questions, etc. The most common ones that attract most studies in the literature are factoid questions which usually begin with Wh-interrogated words, i.e. What, When, Where, Who [27]. With open-domain QA, the questions are not restricted to any particular domain but the users can ask whatever they want. Answers to these questions are facts and they can simply be expressed in text format.

From an overview perspective, as presented in Figure 1.1, the input and output of an open-domain QA system are straightforward. The input is the question, which is unrestricted, and the output is the answer, both are coherent natural language sentences and presented by text sequences. The system can use resources from the web or available databases. Any system like this can be considered as an open-domain QA system. However, open-domain QA is usually broken down into smaller sub-tasks since being able to give concise answers to any questions is not trivial. Corresponding to each sub-task, there is a component dedicated to it. Typically, there are two sub-tasks: document retrieval and document comprehension (or machine comprehension). Accordingly, open-domain QA systems customarily comprise of two modules: a Document Retriever and a Document Reader. Seemingly, the Document Retriever handles the document retrieval task and the Document Reader deals with the machine comprehension task. The two modules can be integrated in a pipeline manner, e.g. [7, 46], to form a complete open-domain QA system. This architecture is depicted in Figure 1.2.

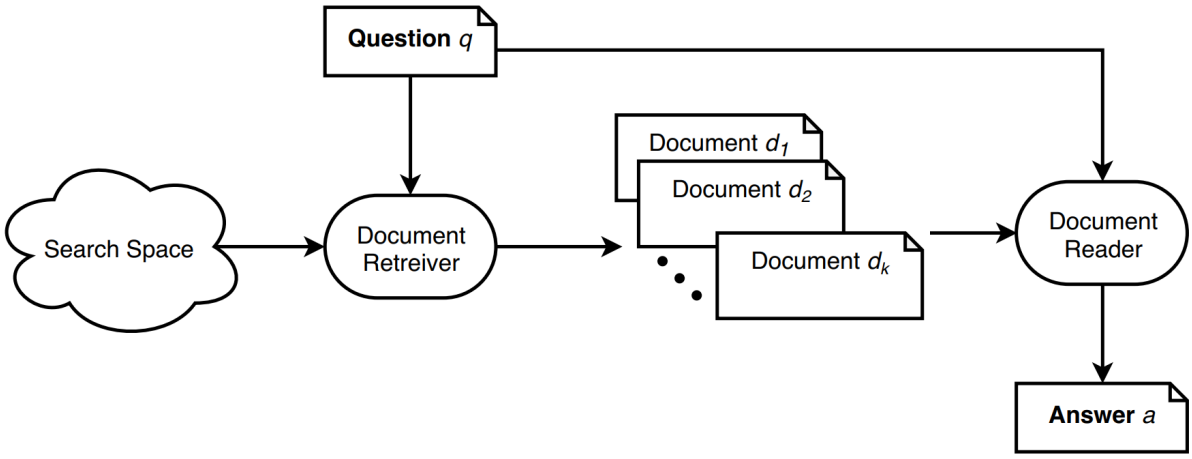

Figure 1.2: The pipeline architecture of an Open-domain QA system.

The input of the system is still a question, namely q, and the output is an answer a. Given q, the Document Retriever acquires top-k documents from a search space by ranking them based on their relevance to q. Since the requirement for open-domain systems is that they should be able to answer any question, the hypothetical search space is massive as it must contains the world knowledge. However, an unlimited search space is not practical, so, knowledge sources like the Internet, or specifically Wikipidia, are commonly used. In the document retrieval phase, a document is considered relevant to question q if it helps answer q correctly, meaning that it must at least contains the answer within its content. Nevertheless, containing the answer alone is not enough because the document returned should also be comprehensible by the Reader and consistent with the semantic of the question. The relevance score is quantifiable by the Retriever so that all the documents can be ranked using it. Let $\mathbb{D}$ represent all documents in the search space, the set of top-k highest-scored documents is:

$$\mathbb{D}^{\star} = \underset{X \in [\mathbb{D}]^k}{argmax} \left(\sum_{d \in X} f\left(d, q\right) \right) \tag{1.1}$$

where $f(\cdot)$ is the scoring function. After obtaining a workable list of documents, $\mathbb{D}^{\star}$, the Document Reader takes q and $\mathbb{D}^{\star}$ as input and produces an answer a which is a text span in some $d_j \in \mathbb{D}^{\star}$ that gives the maximum likelihood of satisfying the question q. Unlike the Retriever, the Reader only has to handle handful number of documents. Yet, it has to examine these documents more carefully because its ultimate goal is to pin point the exact answer span from the text body. This requires certain comprehending power of the Reader as well as the ability to reason and deduce.

1.1.2 Difficulties and Challenges

Open-domain Question Answering is a non-trivial problem with many difficulties and challenges. First of all, although the objective of an open-domain QA system is to give an answer to any question, it is unlikely that this ambition can truly be achieved. This is because not only our knowledge of the world is limited but also the knowledge accessible by IR systems is confined to the information they can process which means it must be digitized. The data can be in various formats such as text, videos, images, audio, etc [27]. Each format requires a different data processing approach. Despite the fact that the knowledge available is bounded,

considering the web alone, the amount of data obtainable is enormous. It poses a scaling problem to open-domain QA systems, especially their retrieval module, not to mention that contents from the Internet are constantly changing.

Since the number of documents in the search space is huge, the retrieving process needs to be fast. In favor of their speed, many Document Retrievers tend to make a trade-off with their accuracy. Therefore, these Retrievers are not sophisticated enough to select relevant documents, especially when they require sufficient comprehending power to understand. Another problem relating to this is that the answer might not be presented in the returned documents even though these documents are relevant to the question to some extent. This might be due to imprecise information since the data is from the web which is an unreliable source, or the Retriever does not understand the semantic of the question. An example of this type of problems is presented in Table 1.1. As can be seen from it, the retrieving model returns document (1) and (3) because it focuses on individual keywords, e.g. "diamond", "hardest gem", "after", etc. instead of interpreting the meaning of the question as a whole. Document (2), on the other hand, satisfies the semantic of the question but it exhibits wrong information.

Table 1.1: An example of problems encountered by the Document Retriever.

Question:	What is the second hardest gem after diamond?
Answer:	Sapphire
Documents:	(1) Diamond is a native crystalline carbon that is the hardest gem.
	(2) Corundum is the the main ingredient of ruby, is the second hardest material known after diamond.
	(3) After graphite, diamond is the second most stable form of carbon.

As mentioned, open-domain QA systems are usually designed in pipeline manner, an obvious problem is that they suffer cascading error where the Reader's performance depends on the Retriever's. Therefore, a poor Retriever can cause a serious bottleneck for the entire system.

1.2 Deep learning

In recent years, deep learning has become a trend in machine learning research due to its effectiveness in solving practical problems. Despite being newly and widely adopted, deep learning has a long history dating all the way back to the 1940s when Walter Pitts and Warren McCulloch introduced the first mathematical model of a neural network [33]. The reason that we see the swift advancement in deep learning only until recently is because of the colossal amount of training data made available by the Internet and the evolution of competent computer hardware and software infrastructure [17]. With the right conditions, deep learning has received multiple successes across disciplines such as computer vision, speech recognition, natural language processing, etc.

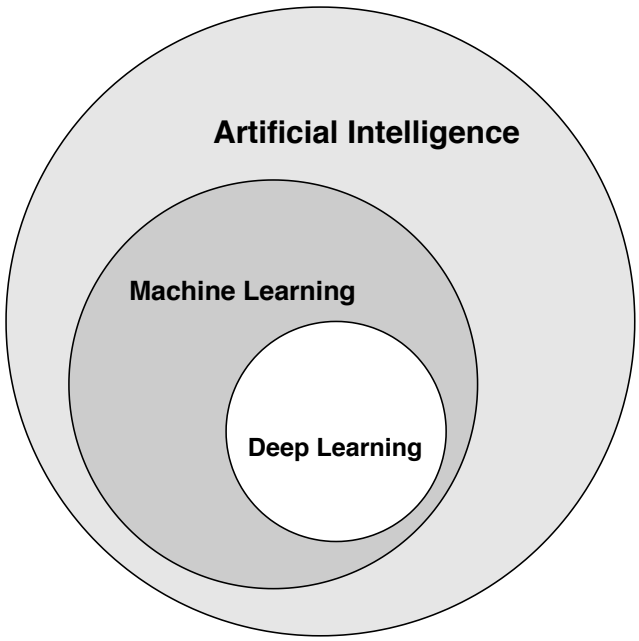

Figure 1.3: The relationship among three related disciplines.

For any machine learning system to work, the raw data needs to be processed and converted into feature vectors. This is the work of multiple feature extractors. However, traditional machine learning techniques are incapable of learning these extractors automatically that they usually require domain experts to carefully select what features might be useful [29]. This process is typically known as "feature engineering." Andrew Ng once said: "Coming up with features is difficult, time consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering."

Although deep learning is a stem of machine learning, as depicted by a Venn diagram in Figure 1.3, its approach is quite different from other machine learning methods. Not only does it require very little to no hand-designed features but also it can produce useful features automatically. The feature vectors can be considered as new representations of the input data. Hence, besides learning the computational models that actually solve the given tasks, deep learning is also representation-learning with multiple levels of abstractions [29]. More importantly, after being learned in one task, these representations can be reused efficiently by many different but similar tasks, which is called "transfer learning."

In machine learning as well as deep learning, supervised learning is the most common form and it is applicable to a wide range of applications. With supervised learning, each training instance contains the input data and its label, which is the desired output of the machine learning system given that input data. In the classification task, a label represents a class to which the data point belongs, therefore, the number of label values are finite. In other words, given the data $X = \{x_1, x_2, ..., x_n\}$ and the labels $Y = \{y_1, y_2, ..., y_n\}$, the set $T = \{(x_i, y_i) \mid x_i \in X, y_i \in Y, 1 \leq i \leq n\}$ is called the training dataset. For a deep learning model to learn from this data, a loss function needs to be defined beforehand to measure the error between the predicted labels and the ground-truth labels. The learning process is actually the process of tuning the parameters of the model to minimize the loss function. To do this, the most popular algorithm can be used is back-propagation [39], which calculates the gradient vector that indicates how the loss function changes with respect to the parameters. Then, the parameters can be updated accordingly.

A deep learning model, or a multi-layer neural network, can be used to represent a complex non-linear function $\mathrm{h}_{\mathbf{W}}(x)$ where x is the input data and $\mathbf{W}$ is the trainable parameters. Figure 1.4 shows a simple deep learning model that has one input layer, one hidden layer, and one output layer. Specifically, the input layer has four units that is x_1, x_2, x_3, x_4; the hidden layer has three units a_1, a_2, a_3; the output layer has two units y_1, y_2. This model belongs to a type of neural network called fully-connected feed-forward neural network since the connections between units do not form a cycle and each unit from the previous layer is connected to all units from the next layer [17]. It can be seen from Figure 1.4 that the output of the previous layer is the input of the following layer. Generally, the value of each unit of the k-th layer ($k \geq 2$, $k = 1$ indicates the input layer), given the input vector $\boldsymbol{a}^{k-1} = \left\{a_i^{k-1} \mid 1 \leq i \leq n\right\}$, n is the number of units in the $(k-1)$-th

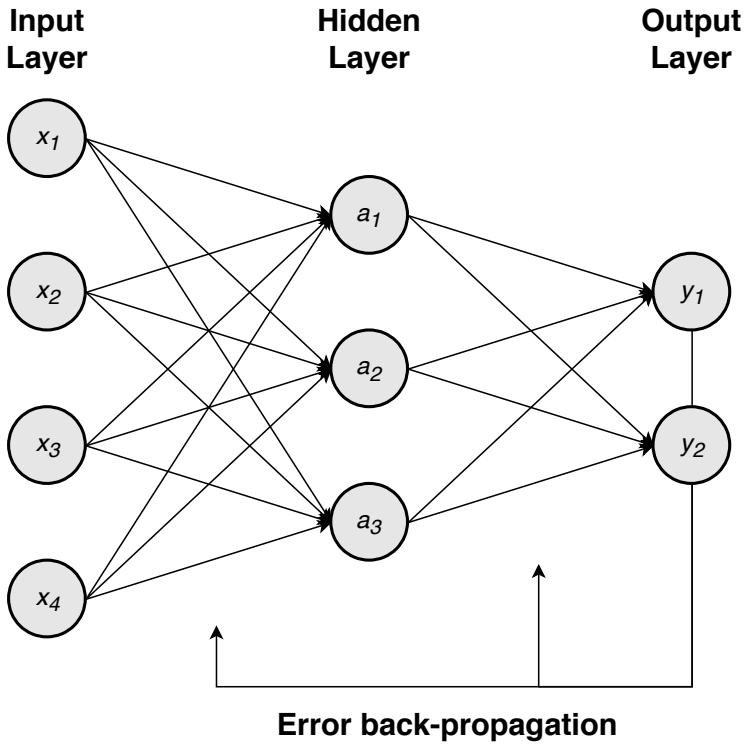

Figure 1.4: The architecture of a simple feed-forward neural network.

layer (including the bias), is calculated as follows:

$$a_j^k = \mathrm{g}\left(z_j^k\right) = \mathrm{g}\left(\sum_{i=1}^{n} w_{ji}^{k-1} a_i^{k-1}\right) \tag{1.2}$$

where $1 \leq j \leq m$, with m is the number of units in the k-th layer (not including the bias); w_{ji}^{k-1} is the weight value between the j-th unit of the k-th layer and the i-th unit of the $(k-1)$-th layer; $\mathrm{g}(x)$ is a non-linear activation function, e.g. sigmoid function. Vector $\boldsymbol{a}^k$ is then fed into the next layer as input (if it is not the output layer) and the process repeats. This process of calculating the output vector for each layer when the parameters are fixed is called forward-propagation. At the output layer, the predicted vector for the input data x, $\hat{\boldsymbol{y}} = \mathrm{h}_{\mathbf{W}}(x)$, is obtained.

1.3 Objectives and Thesis Outline

While there are numerous models proposed for dealing with machine comprehension task [9, 11, 41, 47], advanced document retrieval models in open-domain QA have not received much investigation even though the Retriever's performance is critical to the system. To promote the Retriever's development, Dhingra et al.

proposed QUASAR dataset [12] which encourages open-domain QA research to go beyond understanding a given document and be able to retrieve relevant documents from a large corpus provided only the question. Following this progression and the works in [7, 46], the thesis focus on building an advanced model for document retrieval and the contributions are as follow:

- The thesis proposes a method for learning question-aware self-attentive document encodings that, to the best of our knowledge, is the first to be applied in document retrieval.
- The Reader from DrQA [7] is utilized and combined with the Retriever to form a pipeline system for open-domain QA.
- The system is thoroughly evaluated on QUASAR-T dataset and achieves exceeding performance compared to other state-of-the-art methods.

The structure of the thesis includes:

Chapter 1: The thesis introduces Question Answering and focuses on Open-domain Question Answering systems as well as their difficulties and challenges. A brief introduction about Deep learning is presented and the objectives of the thesis are stated.

Chapter 2: Background knowledge and related work of the thesis are introduced. Various deep learning techniques that are directly used in this thesis are represented. This chapter also explains pairwise learning to rank approach and briefly goes through some notable related work in the literature.

Chapter 3: The proposed Retriever is demonstrated in detail with four main components: an Embedding Layer, a Question and Document Encoding Layer, and a Scoring Function. Then, an open-domain QA system is formed with our Retriever and the Reader from DrQA. The training procedures of these two models are described.

Chapter 4: The implementation of the models is discussed with detailed hyperparameter settings. The Retriever as well as the complete system are thoroughly evaluated using a standard dataset, QUASAR-T. Then, they are compared with baseline models, some of which are state-of-the-art, to demonstrate the strength of the system.

Conclusions: The summary of the thesis and future work.

Chapter 2

Background knowledge and Related work

2.1 Deep learning in Natural Language Processing

2.1.1 Distributed Representation

Unlike computer vision problems where they can take in raw images (basically tensors of numbers) as the input for the model, in natural language processing (NLP) problems, the input is usually a series of words/characters which is not a type of values that a deep learning model can work on directly. Therefore, a mapping technique is required to transform a word/character to its vector representation at the very first layer so that the model can understand.

Figure 2.1 depicts such mechanism which is commonly known as embedding look-up mechanism. The embedding matrix, which is a list of embedding vectors, can be initialized randomly or/and learned by some representation learning methods. If the embeddings are learned through some "fake" tasks before applying to the model, they are called pre-trained embeddings. Depends on the problem, the pre-trained embeddings can be fixed [24] or fine-tunned during training [28]. Whether it is word embedding or character embedding that we use, the look-up mechanism works the same. However, the impact that each type of embedding makes is quite different.

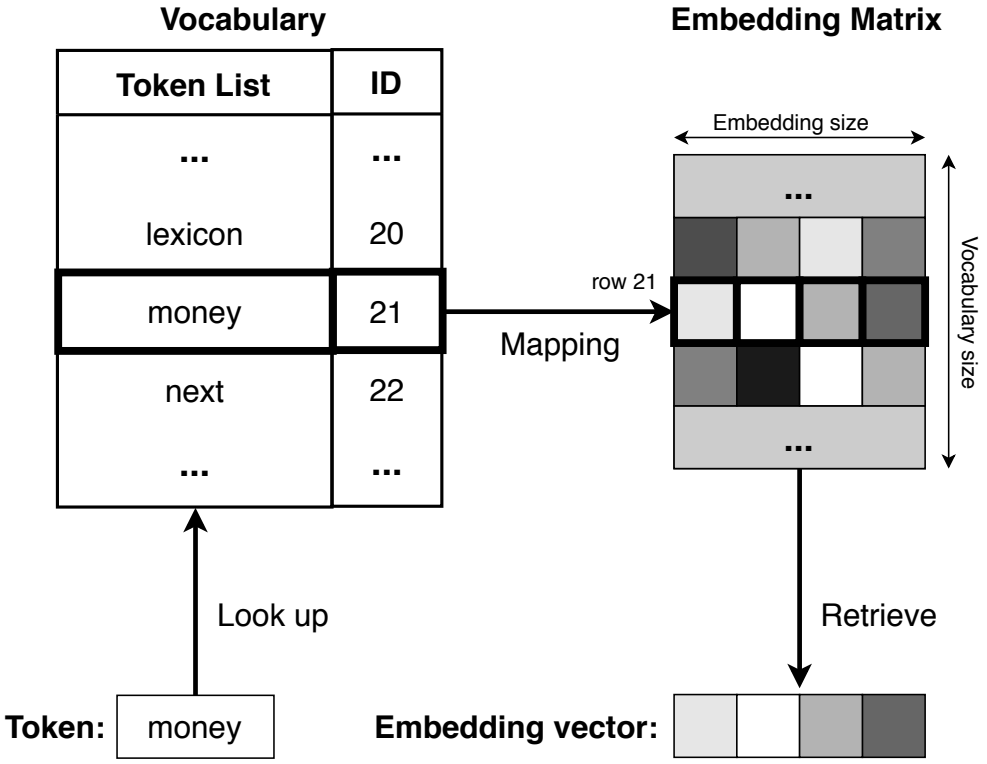

Figure 2.1: Embedding look-up mechanism.

2.1.1.1 Word Embedding

Word embedding is a distributional vector that is assigned to a word. The simplest way to acquire this vector is to create it randomly. Nonetheless, this would result in no meaningful representation that can aid the learning process. It is desirable to have word embeddings with ability to capture similarity between words [14], and there are several ways to achieve this.

According to [50], the use of word embeddings was popularized by the work in [35] and [34], where two famous models, continuous bag-of-words (CBOW) and skip-gram, are proposed, respectively. These models follow the distributional hypothesis which states that similar words tend to appear in similar context. With CBOW, the conditional probability of a word is computed given its surrounding words obtained by applying a sliding window of size k. For example, with $k = 2$, we calculate $\mathbf{P}(w_i \mid w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2})$. In this case, the context words are the input and the middle word is the output. Contrarily, the skip-gram model is basically the inverse of CBOW where the input are now a single word and the output are the context words. Generally, the original task is to obtain useful word embeddings, not to build a model that predicts words. So, what we care about are

the vector outputted by the hidden layer for each word in the vocabulary after the model is trained. Word embedding is widely used in the literature because of its efficiency. It is a fundamental layer in any deep learning model dealing with NLP problems as well as the contributing reason for many state-of-the-art results [50].

2.1.1.2 Character Embedding

Instead of capturing syntactic and semantic information like word embedding, character embedding models the morphological representation of words. Besides adding more useful linguistic information to the model, using character embedding has many benefits. For many languages (e.g. English, Vietnamese, etc), the character vocabulary size is much smaller than the word vocabulary size which results in much less embedding vectors needed to be learned. Since all words comprise of characters, character embedding is the natural choice for handling out-of-vocabulary problem that word embedding method usually suffers from even with large word vocabularies. Especially, when using character embedding with conjunction with word embedding, several methods show significant improvement [10, 13, 32]. Some other methods use only character embedding and still achieve positive results [6, 25].

2.1.2 Long Short-Term Memory network

For almost any NLP problems, the input is in the form of token stream (e.g. sentences, paragraphs). After mapping these tokens to their corresponding embedding vectors, we will have a list of such vectors where each vector is an input feature. If we apply a traditional fully-connected feed-forward neural network, each input feature would have a different set of parameters. It would be hard for the model to learn the position independent aspect of language [17]. For example, given two sentences “I need to find my key”, “My key is what I need to find” and a question “What do I need to find?”, we want the answer to be “my key” no matter where that phrase is in the sentence.

Recurrent neural network (RNN) is a type model that was born to deal with sequential data. RNN was made possible using the idea of parameter sharing across time steps. Besides the fact that the number of parameters can be reduced drastically, this helps RNNs generalize to process sequences of variable length

such as sentences even if they were not seen during training, which requires much less training data. More importantly, the statistical power of the model can be reused for each input feature.

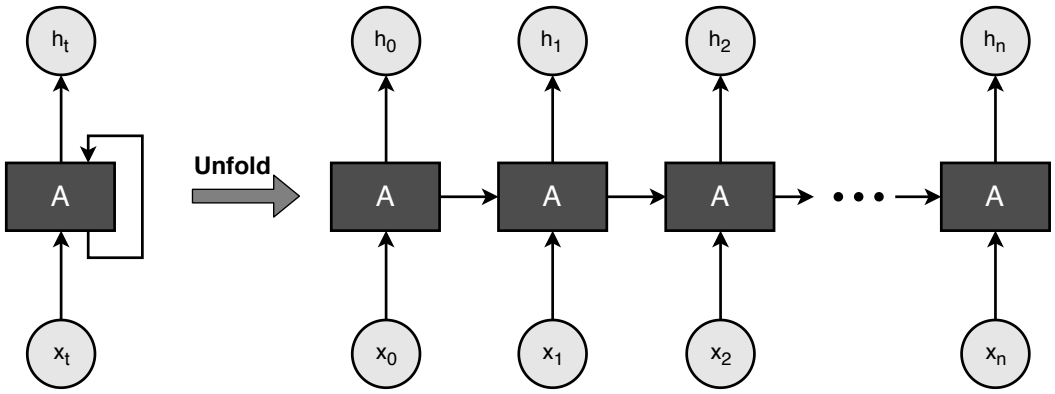

Figure 2.2: Recurrent Neural Network.

There are two ways to describe an RNN as depicted in Figure 2.2. The left diagram represents the actual implementation of the network at time step t, which contains the input $\boldsymbol{x}_t$, the output $\boldsymbol{h}_t$ and a function A that takes both the current input and the output of the previous step as arguments. It is worth nothing that there is only one function A with one set of parameters. We can see that all the information up to time step t is accumulated into $\boldsymbol{h}_t$. The right diagram is the unfolded version of the left diagram where all time steps are flatten out, each repeats the others in terms of computation except at a different time step, or state. The RNN shown in Figure 2.2 is one directional and the network state at time t is only affected by the states in the past. Sometimes, we want the output of the network $\boldsymbol{h}_t$ to depend on both the past and the future. In other words, $\boldsymbol{h}_t$ must take into account the information accumulated from both directions up to t. We can achieve this by reversing the original input sequence and apply another RNN to it. Then, the final output is a combination of these two RNNs' output. This network is called bi-directional recurrent neural network [17].

While it seems like RNN is the ideal model for NLP, in practice, the vanilla RNN is very hard to train due to the problem called vanishing/exploding gradient. Later, long short-term memory (LSTM) network [23] was proposed to combat the problem by introducing gating mechanism. The idea is to design self-loop paths that retain the gradient flow for long periods. To improve the idea even more, [15] proposed a weighted gating mechanism that can be learned rather than fixed. In the traditional RNN shown previously, function A is just a simple non-linear

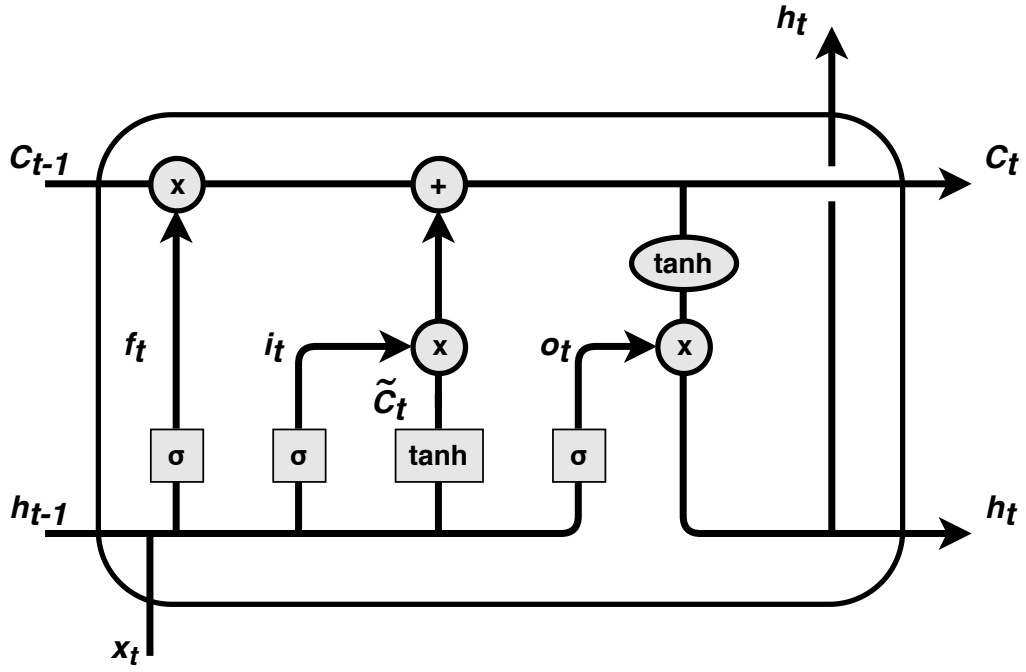

Figure 2.3: Long short-term memory cell.

transformation. In LSTM networks, A is replaced with a LSTM cell, which has an internal loop, as depicted in Figure 2.3. Thanks to this feature, LSTM networks can learn long-term dependencies much easier than the vanilla recurrent networks [17]. The operation visually represented in Figure 2.3 can be rewritten as formulas for a time step t as follows:

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{x}_t \mathbf{U}^i + \boldsymbol{h}_{t-1}\mathbf{W}^i\right) \tag{2.1}$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{x}_t \mathbf{U}^f + \boldsymbol{h}_{t-1}\mathbf{W}^f\right) \tag{2.2}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{x}_t \mathbf{U}^o + \boldsymbol{h}_{t-1}\mathbf{W}^o\right) \tag{2.3}$$

$$\tilde{\boldsymbol{c}}_t = \tanh\left(\boldsymbol{x}_t \mathbf{U}^g + \boldsymbol{h}_{t-1}\mathbf{W}^g\right) \tag{2.4}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{2.5}$$

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{c}_t\right) \odot \boldsymbol{o}_t \tag{2.6}$$

where $\mathbf{U}^i$, $\mathbf{W}^i$, $\mathbf{U}^f$, $\mathbf{W}^f$, $\mathbf{U}^o$, $\mathbf{W}^o$, $\mathbf{U}^g$, $\mathbf{W}^g$ are the parameters; $\odot$ is the element-wise multiplication; $\boldsymbol{i}_t$ represents the input gate that decides how much to take in new information; $\boldsymbol{f}_t$ is the forget gate which controls how much to forget the information stored in the cell; $\boldsymbol{o}_t$ is the output gate that regulates the information produced at time step t. Because of its robustness, LSTM has been widely and successfully adopted to solve various problems such as machine translation [3], speech recognition [19], image caption generation [49], and many others.

2.1.3 Attention Mechanism

2.1.3.1 General framework

For humans, attention mechanism is a solution to help us perceive information effectively, to select what matters and filter out what does not because our brains' processing power is way more limited than the amount of information we can (or cannot) absorb [2]. Although the way attention mechanism in machine learning works is far from how our brains function, there is a similarity in the abstract idea, which is the ability to focus on a particular part of the input. Hence, the term "attention" is borrowed.

Recurrent neural network, which was discussed previously, deals with sequential input. Sometimes, this input can get really long that can saturate the information overall to a point where it becomes useless or even misleading. This problem happens even with LSTM networks.

In the encoder-decoder architecture, which is a common solution for machine translation or text summarization problems, the encoder needs to learn to compress the input into a meaningful intermediate representation before feeding it to the decoder [50]. The longer the input sequence, the harder it is to encode it. This is where the attention mechanism comes into play. As in tasks like machine translation, each part of the output sequence is highly dependent on only some part of the input sequence, rarely all of it. With attention mechanism, this intuition can be achieved.

Attention mechanism was introduced by [3] for machine translation task. In the paper, the authors propose an extension to the traditional encoder-decoder architecture that enable it to perform (soft-)searches for relevant parts in the input sequence automatically. Their idea is demonstrated in Figure 2.4. Firstly, they define a conditional probability for each output [3]:

$$p\left(\boldsymbol{y}_t \mid \boldsymbol{y}_1, ..., \boldsymbol{y}_{t-1}, \mathbf{x}\right) = g(\boldsymbol{y}_{t-1}, \boldsymbol{s}_t, \boldsymbol{c}_t) \tag{2.7}$$

with $\boldsymbol{s}_t$ be the hidden state for time step t in the decoder:

$$\boldsymbol{s}_t = f\left(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_t\right) \tag{2.8}$$

and $\boldsymbol{c}_t$, which is called the context vector for time step t, be the weighted sum of

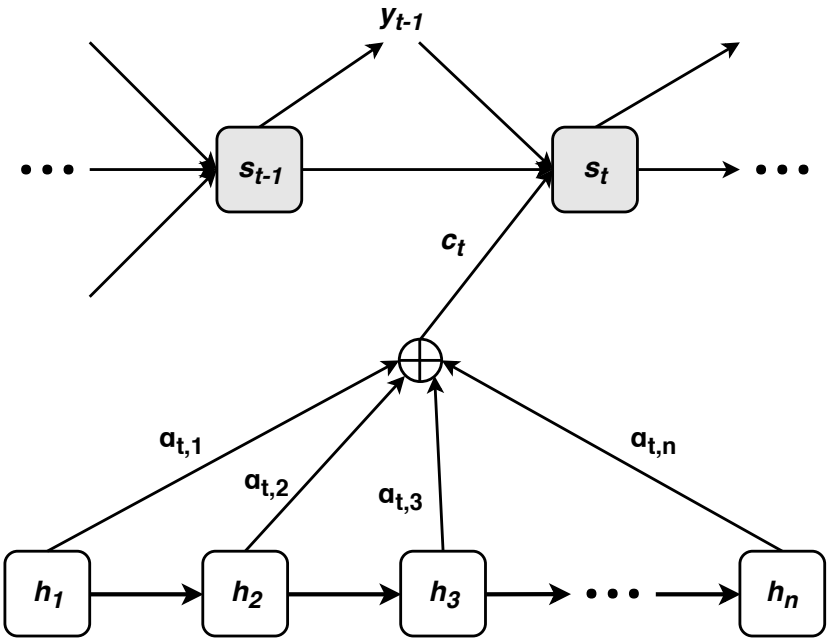

Figure 2.4: Attention mechanism in the encoder-decoder architecture.

all the hidden states in the encoder:

$$\boldsymbol{c}_t = \sum_{j=1}^{n} \alpha_{tj} \boldsymbol{h}_j \tag{2.9}$$

The weight for the hidden state $\boldsymbol{h}_j$ at time step t is calculated as follows:

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{n} \exp(e_{tk})} \tag{2.10}$$

$$e_{tj} = A(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j) \tag{2.11}$$

where A is the alignment model and implemented using feed-forward neural networks. With this attention mechanism, the encoder is relieved from trying to compress all the input information into a fixed-length vector. Instead, at each generation step in the decoder, the model learns to select the important parts of the input.

The attention mechanism proposed in [3] is considered the general attention framework. Although it is still an active field of research [50], attention mechanism has been shown to be highly effective and widely adopted not only in NLP but also in many other fields such as computer vision [49].

2.1.3.2 Self-attention mechanism

There are many variations of the attention mechanism that deviates from the general framework, one of which is self-attention mechanism proposed in [30]. The goal that their paper aims to achieve is to improve sentence embeddings as well as offer a way to interpret how these embeddings come to be. To obtain a fixed-size vector that represents an input sentence of variable lengths, the common methods are to take advantage of the last RNN hidden states or to use some pooling techniques over the hidden states. As mentioned before, it is hard for the RNN model to carry semantic information for too many time steps. Therefore, the authors introduce a self-attention mechanism that automatically learns which parts of the input sentence is semantically important to encode into its embedding using the input itself. After applying an RNN to the input sequence, we obtain the sequence of hidden states:

$$\mathbf{H} = \{\boldsymbol{h}_1, \boldsymbol{h}_2, ..., \boldsymbol{h}_n\} \tag{2.12}$$

which can be seen as a matrix $\mathbf{H} \in \mathbb{R}^{u \times n}$, u is the size of each hidden state and n is the length of the sequence. Then, the attention weights for $\mathbf{H}$ are calculated based on $\mathbf{H}$ itself, given the weight matrix $\mathbf{W} \in \mathbb{R}^{r \times u}$ and the weight vector $\boldsymbol{w} \in \mathbb{R}^{1 \times r}$, r is a hyperparameter:

$$\boldsymbol{\alpha} = \mathrm{softmax}\left(\boldsymbol{w} \tanh(\mathbf{WH})\right) \tag{2.13}$$

The embedding that represents an input sentence is the weighted sum of the hidden states:

$$\boldsymbol{e} = \sum_{i=1}^{n} \alpha_i \boldsymbol{h}_i \tag{2.14}$$

By thoroughly evaluating the proposed self-attention mechanism on three different tasks, [30] shows the effectiveness of the technique. Furthermore, this method can be apply for much longer input such as paragraphs or documents.

2.2 Employed Deep learning techniques

2.2.1 Rectified Linear Unit activation function

One of the main reasons why deep learning is such a powerful tool is that it can model highly complex non-linear functions by using non-linear activation func-

tions. Without these functions, a neural network, no matter how deep, is only a linear transformation from the input to the output, which is not very useful.

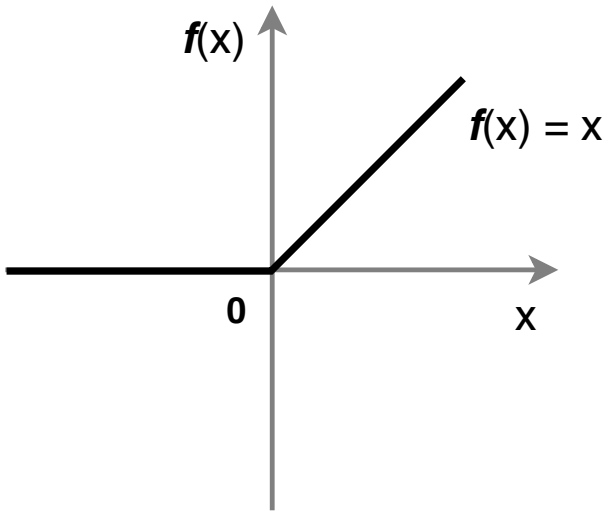

Figure 2.5: The Rectified Linear Unit function.

Rectified Linear Unit (ReLU) [37] is a common activation function due to its simplicity and nearly-instant calculating speed. Since it is unbounded, large values are not saturated (e.g. sigmoid function saturates large values to 1), which makes it efficient and become a recommended choice when building neural networks [14]. ReLU function is defined as [37]:

$$y = \max(0, x) \tag{2.15}$$

Figure 2.5 is the graphical representation of the Equation 2.15. One variant of ReLU that is also mentioned in [37] is Noisy Rectified Linear Unit (NReLU) with the formula as follows:

$$y = \max\left(0, x + \mathrm{N}\left(0, \sigma(x)\right)\right) \tag{2.16}$$

where $\mathrm{N}\left(0, \sigma(x)\right)$ is the Gaussian noise with zero mean and variance $\sigma(x)$.

2.2.2 Mini-batch gradient descent

As mentioned in 1.2, model training is just following a procedure to update the parameters so that the loss function can be minimized. It is basically an optimization problem. The engines of back-propagation are gradient descent and the chain

rule. Let $E(\mathbf{W})$ be the loss function with $\mathbf{W}$ represents real-valued parameters, the gradient descent algorithm can be described as follows:

1. Initialize $\mathbf{W} = \mathbf{W}_0$ randomly or selectively.

2. Update $\mathbf{W}$ until the loss value, $E(\mathbf{W})$, is acceptable using the following equation:

$$\mathbf{W}_k = \mathbf{W}_{k-1} - \epsilon \nabla_{\mathbf{W}} E(\mathbf{W}) \tag{2.17}$$

   where k is an iterator, $k \geq 1$; ϵ, which is a hyperparameter, is a scalar that determines the step size (or learning speed) when updating.

Batch gradient descent is simply using all the training examples when updating $\mathbf{W}$. One advantage of this method is that the direction towards an optimal point is stable since all the data is considered. However, with large training dataset, which is a common case nowadays, calculating gradient descent for the entire dataset is expensive or even infeasible, not to mention that we might not be able to load the dataset into the memory all at once. At the opposite end, we have stochastic gradient descent (SGD), where instead of using all examples, only one is used to update $\mathbf{W}$. Thus, SGD is much faster than batch gradient descent at each step, but it would take more steps before converging. Because SGD does not require all the data must be loaded altogether, it is suitable for big dataset and problems in which the model is constantly changing (e.g. online learning).

Inheriting the ideas from both batch gradient descent and SGD, mini-batch gradient descent divides the training data into small batches, each contains n data points, n is greater than 1 and much smaller than the total number of examples. One iteration through all the batches is called an epoch. Both SGD and mini-batch gradient descent require shuffling the data before training. The updating procedure is the same for mini-batch gradient descent except each batch is used at one step. For this reason, mini-batch gradient descent is faster than batch gradient descent and its converging direction is more stable than SGD.

2.2.3 Adaptive Moment Estimation optimizer

Adaptive Moment Estimation (Adam) [26] is one of the variants of gradient descent. Adam is an robust algorithm because it is easy to implement, does not requires much memory, only uses the first order derivative, scales well the data

and parameters. Unlike the vanilla gradient descent algorithm described in 2.2.2, which has a fixed learning speed ϵ, Adam uses adaptive procedure to calculate an appropriate learning rate for each parameter. The Adam algorithm is represented in Algorithm 2.1.

Algorithm 2.1: Adam algorithm [26].

Input : Step size α; exponential decay rates for moment estimates $\beta_1, \beta_2 \in [0, 1)$; stochastic objective function $f(\theta)$; the initial parameter vector θ_0.

Output: Resulting parameters θ_t

$m_0 \leftarrow 0$

$v_0 \leftarrow 0$

$t \leftarrow 0$

while θ_t *not converged* **do**

  $t \leftarrow t + 1$

  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$

  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$

  $\hat{m}_t \leftarrow m_t / \left(1 - \beta_1^t\right)$

  $\hat{v}_t \leftarrow v_t / \left(1 - \beta_2^t\right)$

  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / \left(\sqrt{\hat{v}_t} + \epsilon\right)$

end

return θ_t

According to [26], the hyperparameters α, β_1, β_2, ϵ have reasonable default settings as $0,001$, $0,9$, $0,999$, and 10^{-8} respectively. These values are advisable for many evaluated machine learning problems. In practice, Adam works quite well and it is more favorable than other optimizing methods.

2.2.4 Dropout

In many cases where there is insufficient amount of training data, a neural network with too many layers and parameters can "remember" all the training instances so that it can predict really well on the training dataset. However, when given new examples, the model performs poorly. This problem is called "overfitting" and it is detrimental to the model's performance because it makes the model much more

sensitive to noise and decrease the model's ability to generalize.

Dropout [43] is one of the techniques that can be used to mitigate overfitting problem. The core idea of Dropout is randomly and temporarily removing some units in the neural network along with their associate connections during the training phase. Every time dropout is applied, a new network is attained with less connections between layers than the original network. If the total number of units that a neural network has is n, then the number of possible new networks is 2^n. However, all the parameters are shared among these networks. According to [43], training a neural network with Dropout is equivalent with training 2^n smaller networks where not every network is guarantee to be trained.

Dropout can help reduce overfitting because it weakens the dependency of units, so called "co-adaptation" from the original paper. The units are forced to learn independently but can still cooperate with other random units. Dropout has been proven to be greatly effective when many proposed models for object classification, speech recognition, biomedical data analysis, etc. were significantly improved and even became state-of-the-art models.

2.2.5 Early Stopping

Besides Dropout, early stopping is another technique that can be used to restrain overfitting. The visual cue for overfitting is that when we plot out the training and validating loss over time, the model starts to overfit right after the point where the validating loss hits the global minimum while the training loss keeps decreasing. By observing this behaviour, the idea for early stopping strategy is quite simple: keeping track of the best version of the model parameters and revert to it when the training process stops improving for some time. The early stopping algorithm is formally represented in Algorithm 2.2.

Early stopping has many beneficial qualities compared too some other regularization techniques. As shown in Algorithm 2.2, it is fairly simple yet so effective. Moreover, it does not require changing the training process like some methods that modify the objective function. Instead, it is only an add-on that can work well with other strategies.

Algorithm 2.2: The early stopping algorithm [17].

Input : The number of training steps before evaluation n; the number of times willing to suffer lower validating error before giving up; the initial parameters θ_0

Output: Best parameters θ^*, best number of training steps i^*

$\theta, \theta^* \leftarrow \theta_0$
$i, j, i^* \leftarrow 0$
$v \leftarrow \infty$
while $j < p$ **do**
    Update θ for n steps.
    $i \leftarrow i + n$
    $v' \leftarrow \text{ValidationSetError}(\theta)$
    if $v' < v$ **then**
        $j \leftarrow 0$
        $\theta^* \leftarrow \theta$
        $i^* \leftarrow i$
        $v \leftarrow v'$
    else
        $j \leftarrow j + 1$
    end
end
return θ^*, i^*

2.3 Pairwise Learning to Rank approach

There are many problems in Information Retrieval can be regarded as ranking problems such as document retrieval, sentiment categorization, definition mapping, etc. Hence, ranking methods are the key to IR. They have been actively researched for decades with various algorithms have been proposed [31]. A research topic called "learning to rank" emerged which explores several ranking techniques using machine learning as the engine. Generally, learning to rank means building and training a ranking model using data with the objective is to sort a list of instances using some criteria such as the degree of relevance or importance. For the problem of document retrieval given a query, a common solution is to: (1) convert the query and documents into feature vectors, (2) use a similarity metric on these

vectors, and (3) sort the documents based on their scores [4]. Documents and queries can be in any type of formats, e.g. text, images, audio, web pages, etc. as long as they can be embedded into vector representations.

There are three approaches to learning to rank: pointwise, pairwise, and listwise approach. Each of them defines a different input/output space and use a different objective function [31]. Among them, pairwise approach is the most common one and will be discussed in more detail.

In pairwise methods, while training, the model takes in two documents as one training instance (instead of one as in pointwise or a list as in listwise) and outputs the corresponding scores for them. The prefer order of two documents depends on how the metric is defined but mostly, the document with the higher score is more preferred than the other one and it will be labeled as positive, thus, the other will be negative. As stated in [4], the ranking model is represented as a scoring function $f(q, d)$ with q and d are the embeddings of the query and the document (positive or negative), respectively. With the input tuple of $(q, d+, d-)$, the model needs to be selected so that $f(q, d+) > f(q, d-)$, meaning that the score for a positive document should be higher than the score for a negative document. This goal is the reason for the margin ranking loss function introduced in [21]:

$$J = \sum_{(q,d+,d-)\in\mathbb{D}} \max\left(0, \alpha - f(q, d+) + f(q, d-)\right) \tag{2.18}$$

with $\mathbb{D}$ is all the training tuples in the dataset; α is the margin value which enforces the score difference between positive and negative document. The model will then learn to differentiate the positive and negative document by at least α.

Pairwise learning to rank approach is not only used in NLP for problems like question answering [1, 5] but also used in computer vision, especially in face verification problem [40]. In this problem, instead of the margin ranking loss function, a different but similar loss function is used called triplet loss function:

$$J = \sum_{i}^{N} \max\left(0, \alpha + \left\|g\left(x_i^a\right) - g\left(x_i^p\right)\right\|_2^2 - \left\|g\left(x_i^a\right) - g\left(x_i^n\right)\right\|_2^2\right) \tag{2.19}$$

with N is the number of all training instances; α is still the margin value; $g(\cdot)$ is embedding function which learns to map the anchor image x_i^a, the positive image x_i^p, and the negative image x_i^n into the same vector space. Although their formulas look different, their ideas are the same. The margin ranking loss function can be considered as a more general case of the triplet loss function since function

$f(\cdot)$ models both the embedding function and the scoring metric. In the case of triplet loss function, the scoring metric uses Euclidean distance. The smaller the distance, the more prefer the object is.

When applying pairwise ranking, it is essential to select appropriate training instances. Because of how the loss function is defined, the model will try to learn so that $f(q, d+) > f(q, d-)$. If the training example $(q, d+, d-)$ has already satisfied this condition, it will not improve the model, only slow down the training process. Therefore, to speed up training, only training examples that can actually impact the learning process, i.e. $T = \{(q, d+, d-) \mid f(q, d+) - \alpha < f(q, d-)\}$, should be chosen.

2.4 Related work

Unlike closed-domain QA, which is restricted to a certain domain and requires manually constructed knowledge bases, open-domain QA aims to answer questions about basically anything [27]. Hence, it relies on world knowledge in the form of large corpora, e.g. Wikipedia. Many datasets have been proposed, such as SQuAD [38], WikiReading [22], or recently, QUASAR dataset [12], that facilitate the development of open-domain QA systems. The most well-known dataset is SQuAD which consists more than 100,000 questions derived from Wikipedia. It was proposed to help develop models that are capable of understanding and reasoning to answer open-domain questions correctly. Because the dataset has already provided the context document for each question and the answer is guaranteed to appear in the context, SQuAD is only used to train machine readers. However, since a complete open-domain QA system composes of a document retriever and a machine reader module, SQuAD or such dataset alone will not be enough to promote building an entire system without exploiting other sources. Having recognized the problem, Dhingra et al. present QUASAR dataset [12]. This dataset can be divided into two sub-datasets, each of which targets a different style of question answering. The QUASAR-S dataset has more than 37,000 fill-in-the-gap type of queries constructed using Stack Overflow as the source. Therefore, it can be considered as closed-domain dataset. On the other hand, the QUASAR-T dataset includes about 43,000 open-domain trivia questions gathered from various sources. It supports both the document retrieving and reading process by providing a list of documents associated with each question-answer pair.

The document retriever can be trained to rank these documents and return only some highest-scored ones.

Thanks to the advance of deep learning, especially, the emergence of attention mechanism there has been momentous progress in machine reading comprehension task. Wang et al. [47] propose a mechanism called Gated Attention-based recurrent networks which is employed prior to a self-matching layer which also uses attention to extract important evidence from the documents. Specifically, the model has four main parts which are a question/document encoding component, a gated-matching layer, a self-matching layer, and a pointer network. Their experiments done on the SQuAD dataset shows promising results since the model placed first on the official leaderboard of SQuAD. Another gated-attention is used in [11] where Dhingra et al. exploit a bi-directional Gated Recurrent Unit for question/document encodings at the beginning of each layer in their multi-hop architecture. Then, they apply a Gated-Attention module for each token of the sequence. Cui et al. [9] introduce Attention-over-Attention (AoA) reader in which another attention layer is introduced on top of document-level attention over individual query words. The model tries to solve cloze-style reading comprehension problem by take in account the interactive information between the query and the document. This work also shows that the query representation is essential and it requires more attention. After they obtain the contextual embeddings of the question and the document, a pair-wise matching matrix is calculated. Then, the AoA mechanism is applied in which the latter attention layer determines the importance of each previous individual attention. Their experiment shows exceeding results compared to various state-of-the-art systems. Later, Seo et al. [41] focus even further on the question-aware context representation by proposing the Bi-directional Attention Flow (BiDAF) network which is a hierarchical multi-stage architecture for document representation at various levels of abstraction. The model contains character-level, token-level, and contextual level information. The attention vector is calculated every time step, combining with the embeddings of the previous layers, flow through the model, hence, creating an attention flow. This method also produces state-of-the-art results for SQuAD dataset at the time of submission.

Besides the methods that are proposed to deal only with machine comprehension task as reviewed previously, there are some full open-domain QA systems that contains both a document retriever and a machine reader. One of the most well-known systems is DrQA [7]. In DrQA, the reader comprises of a paragraph encoding layer which is a multi-layer bi-directional long short-term mem-

ory (BiLSTM) applied on a selective feature set, a question encoding layer that learns a single vector representation of the question, and two classifiers trained independently for predicting the boundaries of the answer span. For fast retrieval speed, DrQA use a simple TF-IDF weighted bag-of-word vectors technique to select relevant documents. This in turn limits the retrieval performance and makes room for more improvement. This thesis utilizes the Reader from DrQA for the machine reading module and proposes a better document retrieval method. Hence, the detail of DrQA's Reader will be discussed in Chapter 3.

While in most open-domain QA systems, document retrieval and machine comprehension are treated as two separated tasks and trained independently, the system in [46], which is called R^3, is designed to have both of these modules integrated as one single model that can be trained in a joint manner. Another difference between [46] and many other recent open-domain QA papers is that instead of focusing only on the machine reader, the authors in [46] acknowledge the importance of the document retriever as well. They point out that the performance of the overall system depends a lot on the document retriever because with a poor retriever component, the reader cannot extract the correct answer afterward. As the name suggests, R^3 or Reinforced Ranker-Reader contains a Ranker (document retriever module) and a Reader (machine reader module), to which reinforcement learning technique is applied. They both use input produced by Match-LSTM architecture [45]. The Ranker is then trained with reinforcement learning to provide probability distribution of documents. The reward is how well the Reader performs on the top-ranked documents. This creates a link between two components and provides a signal to the Ranker so that it is aware of the end performance while still learning. Compared to the common ranking methods such as TF-IDF weighting scheme [7], the Ranker in [46] is more advanced and efficient. The Reader is trained using gradient descent to predict the boundary of the answer span in the documents. R^3 has state-of-the-art results in both document retrieval and machine comprehension task.

Although they have been shown to be highly efficient, in open-domain QA setting, these reading comprehension models depend heavily on document retrieval to acquire relevant documents. For example, the reading accuracy (exact match) of GA (Gated-Attention) model [11] on QUASAR-T test set is 60% but when considering the retriever's performance, the overall accuracy drops to 26.4% [12]. Therefore, the focus is now shifting to improving document retrieval process [7, 46].

Chapter 3

Material and Methods

As discussed in Chapter 1, the typical pipeline of an open-domain QA system consists of a Document Retriever, which handles the document retrieval task, and a Document Reader, which deals with the machine comprehension task. Following this framework, our system also comprises of those two modules with the main focus is on the Document Retriever. Concretely, the Document Retriever is an end-to-end deep learning model that can be divided further into four components: (1) an Embedding Layer for mapping each word in the questions and documents into a vector space, (2) a Question Encoding Layer and (3) a Document Encoding Layer that produce the final representations of the questions and documents, respectively, and (4) a neural-based Scoring Function for learning an effective similarity measurement between two fixed-size vectors. To exploit the power of our Document Retriever in an open-domain QA setting, we utilize the Document Reader from DrQA [7] for extracting the answer from retrieved documents.

3.1 Document Retriever

In several previous works [41, 42, 47] on machine comprehension task, the information from the question and document are fused together to form question-aware document representations. Although, the exact methods that were used in these research differ quite a lot, the core idea of using question-document combined signals is intuitive because neither the document nor the question alone would help finding the answer. Moreover, a document might contain lots of information and it would be redundant if all of these information is compressed into a fixed-size vec-

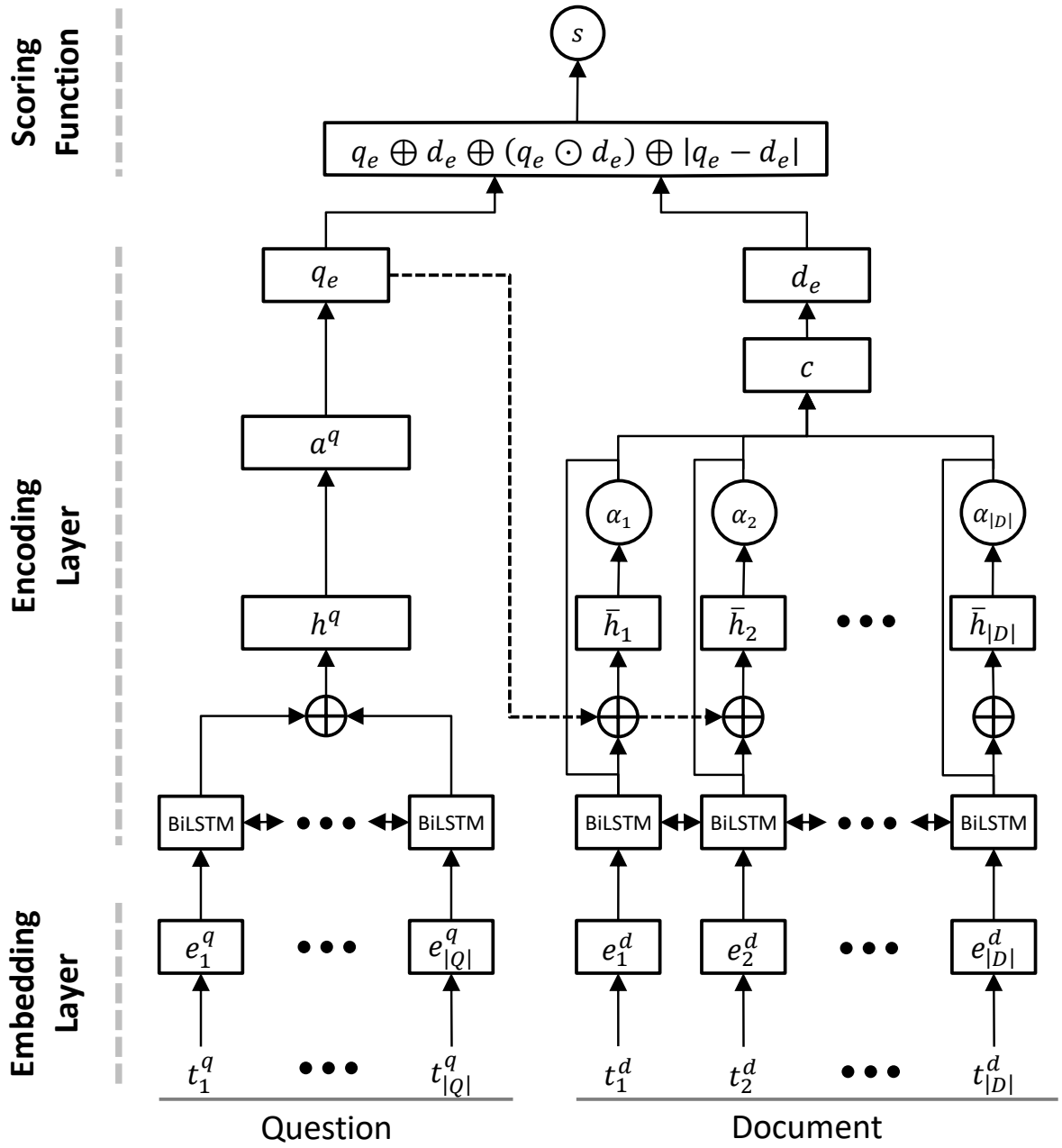

Figure 3.1: The architecture of the Document Retriever.

tor while only a small part of the document is enough to answer correctly. This is also the reason why attention mechanisms are widely adopted to encode the documents. Inheriting all of these ideas, the final document encoding of our model is produced by applying a self-attention mechanism that conditioned not only on the document itself but also on the final question encoding. We hypothesize that this question-aware self-attentive document encoding layer will learn better representations than the one which does not take the question information into account. We named our retriever QASA (short for Question-Aware Self-Attentive) to acknowledge the key ideas as well as the methods that have been applied.

Figure 3.1 shows the architecture (bottom-up) of the Document Retriever's network with one question and one document. In this case, the network will output a single score s for the document with respect to the question. With pairwise learning to rank approach mentioned in 2.3, the input comes in 3-tuple of a question and two documents. Hence, the same document branch of the network will be applied on both documents simultaneously and two independent scores will be produced while training. The following sections will explain each layer in greater detail.

3.1.1 Embedding Layer

An embedding layer (EL) is commonly used as the first layer in a deep learning model in order to solve various NLP problems [50]. It assigns a distributional vector to each token in the input sequence which can be further processed by subsequent layers. This layer can be considered as the first level of abstraction in the question/document representation learning process. Our model uses token-level and character-level embeddings to capture both semantic and morphological information of words. Although it is not necessary to have both type of embeddings, using them in combination has become a best practice since they compensate each other's weaknesses. In this low-level embedding layer, there is no underlying linguistic difference between the question and the document. Therefore, to maximize the representation power of this layer, the same parameters are used for both question and document. Figure 3.2 shows the architecture of EL applied to each token.

Pre-process. One mandatory step before converting the tokens into vectors is to extract all tokens from the raw documents in the first place. While the objective of this task is simple, there is no trivial way to do this with absolute accuracy. In written texts, the tokens are mixed with many ambiguous characters that required sufficient understanding of the language to decide where each token starts and ends. To simplify this problem, characters that are not word nor number characters are removed and the texts are converted to lowercase format. If the document contains an URL, that URL would become one lengthy token and non-informative. So, a simple template matching method is applied to find all URLs and replace them with the word "url". At this point, a document is still one string

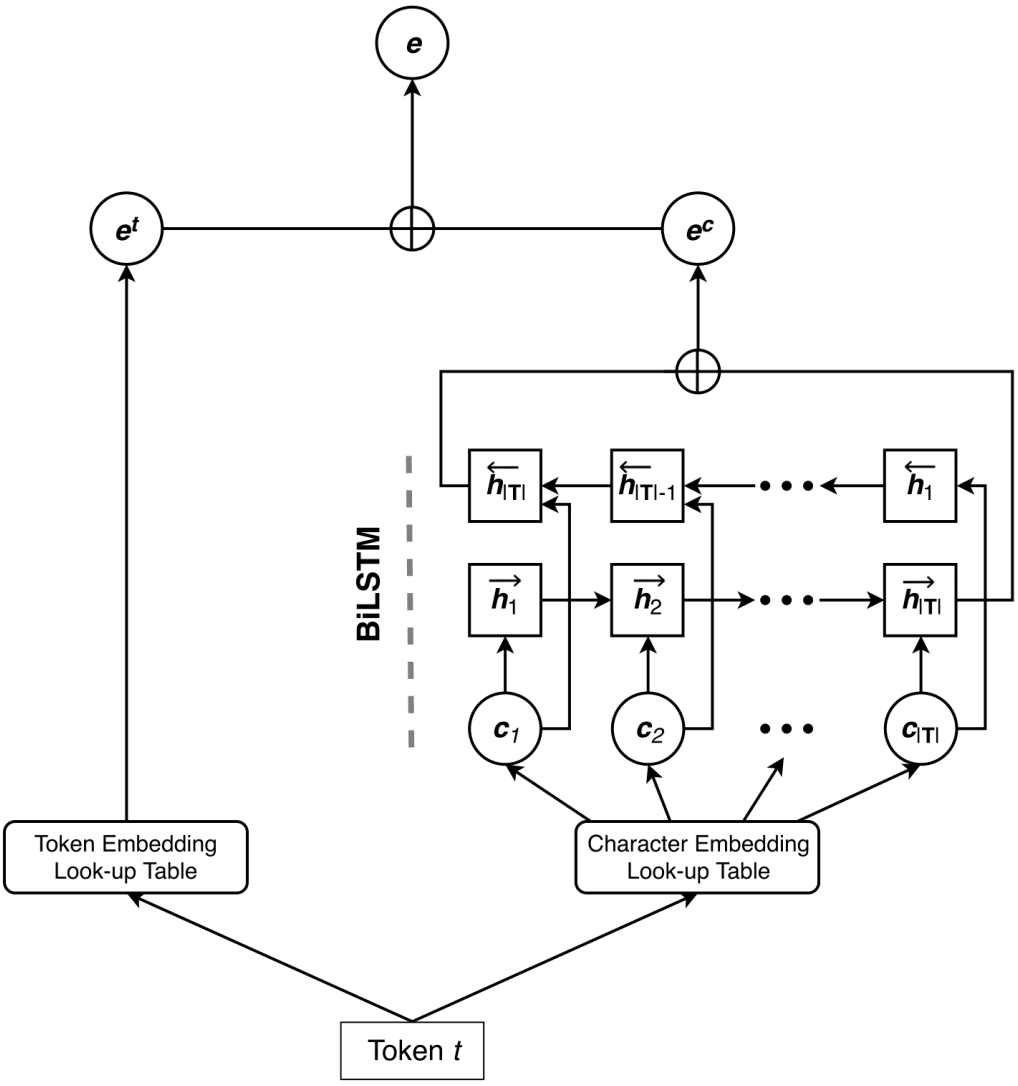

Figure 3.2: The architecture of the Embedding Layer.

of text. We use the best English tokenizer model from spaCy [1] to obtain a list of tokens from a document.

Token Embedding. After the pre-processing step, each token is mapped to its embedding by the mean of a look-up table. We use the pre-trained English word vectors from fastText [18], in which they employ CBOW [35] with position-weights. These vectors are trained on Common Crawl and Wikipedia and they have 300 dimensions. The vocabulary size is more than 2.5 million tokens. We choose not to tune these embeddings while training since doing so with a small dataset can actually disturb the overall structure of the trained vectors and pollute the general contextual representation of tokens.

[1] https://spacy.io/models/en#en_core_web_lg

Character Embedding. Although the vocabulary size of the token embeddings is fairly large (2.5 millions), there is no guarantee that all tokens that the model might encounter would fall under that set. This problem is known as out-of-vocabulary problem and character embedding has been applied to handle it. Different from the Token Embedding, we do not use any pre-trained Character Embedding. Since there are much less characters than tokens, the training data size does not need to be as large. After being pre-processed, the documents only contain word and number characters, hence, there are 36 characters in total. Besides, for characters, there is no semantic structure among them to preserve. So, it is best to learn their vector representations directly from the training dataset.

In this paper, let V_C be the character set, the character embedding matrix $\mathbf{C} \in \mathbb{R}^{|V_C| \times n}$ are first created randomly by Glorot initialization [16] and then fine-tuned as trainable parameters of the model. For each token t, using a look-up table, we can obtain a sequence of character embedding $\mathbf{T} = \left\{\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_{|\mathbf{T}|}\right\}$, $\boldsymbol{c}_i \in \mathbf{C}$. A single layer of bi-directional long short-term memory (BiLSTM) [23] is then applied on $\mathbf{T}$ to produce character-level embedding $\boldsymbol{e}^c$:

$$\boldsymbol{e}^c = \overrightarrow{\boldsymbol{h}}_{|\mathbf{T}|} \oplus \overleftarrow{\boldsymbol{h}}_{|\mathbf{T}|} \tag{3.1}$$

where $\oplus$ is the concatenating function; $\overrightarrow{\boldsymbol{h}}_{|\mathbf{T}|}$ and $\overleftarrow{\boldsymbol{h}}_{|\mathbf{T}|}$ are the last hidden states of the forward and backward direction, respectively.

Final Embedding. Given a sequence of tokens $P = \{t_i\}_{i=1}^{|P|}$, which is either a question or a document, the output of EL is a sequence of embeddings $\mathbf{E} = \{\boldsymbol{e}_i\}_{i=1}^{|P|}$, in which:

$$\boldsymbol{e}_i = \boldsymbol{e}_i^t \oplus \boldsymbol{e}_i^c \tag{3.2}$$

where $\boldsymbol{e}_i^t$ is the pre-trained token embedding, and $\boldsymbol{e}_i^c$ is the character embedding of t_i. The embedding matrix $\mathbf{E}$ now encompasses the input information of a sequence (e.g. question or document) and will be used as input for the following layers.

3.1.2 Question Encoding Layer

This layer aims to learn the fixed-size vector representation of the questions. Normally, the questions' lengths are much shorter than the documents'. Also, since the factoid questions represent the information needs, they are usually concise and explicit. Unlike the documents, each question only entails one specific topic that

is being asked about. For all of these reasons, it is needless to complicate the question encoding process with sophisticated mechanisms that might be prone to overfitting. To produce a single vector for a question, we apply one BiLSTM layer to $\mathbf{E}^q = \left\{\boldsymbol{e}_i^q\right\}_{i=1}^{|Q|}$, the output of EL, with $|Q|$ is the question's length:

$$\overrightarrow{\mathbf{H}}^q = \overrightarrow{\mathrm{LSTM}}\left(\mathbf{E}^q\right) = \left\{\overrightarrow{\boldsymbol{h}}_i^q\right\}_{i=1}^{|Q|} \tag{3.3}$$

$$\overleftarrow{\mathbf{H}}^q = \overleftarrow{\mathrm{LSTM}}\left(\mathbf{E}^q\right) = \left\{\overleftarrow{\boldsymbol{h}}_i^q\right\}_{i=1}^{|Q|} \tag{3.4}$$

This BiLSTM is used to model the contextual information. The last hidden states of the forward and backward LSTM are concatenated into one vector:

$$\boldsymbol{h}^q = \overrightarrow{\boldsymbol{h}}_{|Q|}^q \oplus \overleftarrow{\boldsymbol{h}}_{|Q|}^q \tag{3.5}$$

Then, two fully-connected layers are placed on top of this vector with the output of the first one is activated using ReLU function:

$$\boldsymbol{a}^q = \mathrm{ReLU}\left(\mathbf{W}^{(1)}\boldsymbol{h}^q + \boldsymbol{b}^{(1)}\right) \tag{3.6}$$

$$\boldsymbol{q} = \mathbf{W}^{(2)}\boldsymbol{a}^q + \boldsymbol{b}^{(2)} \tag{3.7}$$

where $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$, $\boldsymbol{b}^{(1)}$, and $\boldsymbol{b}^{(2)}$ are trainable weights and biases of the model. At this step, we obtain the final encoding, namely $\boldsymbol{q}$, of the input question.

3.1.3 Document Encoding Layer

First of all, to take advantage of the contextual representing power, we use the same BiLSTM from the Question Encoding Layer (QEL) along with its parameters:

$$\mathbf{H}^d = \overleftrightarrow{\mathrm{BiLSTM}}\left(\mathbf{E}^d\right) = \left\{\overrightarrow{\boldsymbol{h}}_i^d \oplus \overleftarrow{\boldsymbol{h}}_{|D|-i+1}^d\right\}_{i=1}^{|D|} = \left\{\boldsymbol{h}_i^d\right\}_{i=1}^{|D|} \tag{3.8}$$

However, different from its use in QEL where the last hidden states are combined, all hidden states of the BiLSTM in this layer are combined with the question encoding $\boldsymbol{q}$. For humans, in order to decide whether a document is relevant to a particular question, the meaning of that question should always be kept in mind while we read. Following this intuition, this layer utilizes $\boldsymbol{q}$, which embodied the general meaning of the question, to form the question-aware hidden states, $\overline{\mathbf{H}}$, by concatenating $\boldsymbol{q}$ with the hidden states outputted by a BiLSTM as follow:

$$\overline{\mathbf{H}} = \left\{\boldsymbol{h}_i^d \oplus \boldsymbol{q}\right\}_{i=1}^{|D|} = \left\{\overline{\boldsymbol{h}}_i\right\}_{i=1}^{|D|} \tag{3.9}$$

where $|D|$ is the number of tokens in the document; $\mathbf{E}^d = \left\{\boldsymbol{e}_i^d\right\}_{i=1}^{|D|}$ are the word embeddings from EL. By integrating the question encoding with the document's hidden states as in Eq. 3.9, the document encoding is conditioned on the question, such that it allows the model to produce distinctive representations of the same document depends on the question given.

As mentioned before, a document contains many sentences while frequently, for a factoid question, one sentence or even a part of a sentence is enough to produce the answer. In other words, many pieces of information can be extracted from a document. Some information might be useful for a particular question while others might not. It is favorable to only encode useful information that is most integral to the answer selection process. We achieve this by applying self-attentive network [8] to $\mathbf{H}^d$. It would make sense that the question signal is also used to decide which part of the document is most relevant. Therefore, the attention weights are calculated based on $\overline{\mathbf{H}}$ where the document and question's information are both available:

$$\boldsymbol{a}_i^d = \mathrm{ReLU}\left(\mathbf{W}\overline{\boldsymbol{h}}_i + \boldsymbol{b}\right) \tag{3.10}$$

$$\boldsymbol{u} = \left\{\boldsymbol{v}^\intercal \boldsymbol{a}_i^d + b\right\}_{i=1}^{|D|} \tag{3.11}$$

$$\boldsymbol{\alpha} = softmax\left(\boldsymbol{u}\right) \tag{3.12}$$

with $\mathbf{W} \in \mathbb{R}^{m\times(2z+l)}$, $\boldsymbol{v}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $b \in \mathbb{R}$ are parameters, z and m are the number of the hidden units in the BiLSTM and fully-connected layer respectively, l is the question encoding size. Final representation of the document $\boldsymbol{d}$ is the linear transformation of $\boldsymbol{c}$, which is the weighted sum of the hidden states $\mathbf{H}^d$:

$$\boldsymbol{c} = \sum_{i=1}^{|D|} \alpha_i \boldsymbol{h}_i^d \tag{3.13}$$

$$\boldsymbol{d} = \mathbf{W}\boldsymbol{c} + \boldsymbol{b} \tag{3.14}$$

with $\mathbf{W}$, $\boldsymbol{b}$ are learnable parameters, $\alpha_i \in \boldsymbol{\alpha}$ and $\boldsymbol{h}_i^d \in \mathbf{H}^d$. It is worth noting that $\boldsymbol{d}$ has the same size as $\boldsymbol{q}$, i.e. l, so that they can be in the same vector space.

3.1.4 Scoring Function

After obtaining two fixed-size vectors, $\boldsymbol{q}$ for the question and $\boldsymbol{d}$ for the document, their relevance is measured by a scoring function. Two most common measurements are Euclidean distance: $\sqrt{\sum_{i=1}^{l}\left(\boldsymbol{q}_i - \boldsymbol{d}_i\right)^2}$ and cosine similarity: $\frac{\boldsymbol{q}\cdot\boldsymbol{d}}{\|\boldsymbol{q}\|\|\boldsymbol{d}\|}$.

With Euclidean distance, the smaller the value the closer the vectors, hence, the more relevant. On the contrary, with cosine similarity, the cosine of the angle between two vectors is calculated, thus, the values are bounded between -1 and 1. The vectors are more similar if the value gets closer to 1. Although being effective in some problems, these functions are pre-defined and fixed which limits their power to measure the interaction of multi-dimensional vectors. We decide to let the model learn the scoring function itself so that this function would adapt to the vectors $\boldsymbol{q}$ and $\boldsymbol{d}$ outputted from previous layers.

Our scoring function is a neural network comprises of two feed-forward layers, which is similar to the idea of applying matching methods to extract relations between two vectors in [8]. Since our Document Retriever can be trained in an end-to-end fashion, the error is backpropagated through the scoring function and the encoding layers, which enables the model to learn better similarity measurement as well as question/document representations simultaneously.

Much the same as [8], our feature vector for the scoring function is also a concatenation of the question encoding $\boldsymbol{q}$, document encoding $\boldsymbol{d_e}$, their element-wise product, and their absolute element-wise difference. Especially, the last two features partially simulate how the cosine similarity and Euclidean distance are calculated. This helps the convergence process of the scoring function. Given the two encodings, their similarity score can be calculated as follow:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{d} \\ \boldsymbol{q} \odot \boldsymbol{d} \\ |\boldsymbol{q} - \boldsymbol{d}| \end{bmatrix} \tag{3.15}$$

$$\boldsymbol{a} = \mathrm{ReLU}(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}) \tag{3.16}$$

$$s = \boldsymbol{w} \cdot \boldsymbol{a} + b \tag{3.17}$$

where $\odot$ is the Hadamard product; $\mathbf{W} \in \mathbb{R}^{r \times 4l}$, $\boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^r$, and $b \in \mathbb{R}$ are trainable parameters of the network (l is the encoding size, r is the number of hidden units); scalar s is the similarity score between $\boldsymbol{q}$ and $\boldsymbol{d}$.

3.1.5 Training Process

The Scoring Function is the last layer of the network. In the forward pass, given a question and a document, the network will produce a score at the end. What

is expected of the model is that the relevant documents will receive higher scores than the irrelevant ones. For that purpose, we apply pairwise ranking approach as discussed in 2.3. Each training example is a 3-tuple of question, positive and negative document. The Document Encoding Layer is used twice in parallel to produce the encodings for the positive and negative document accordingly. Let S be the scoring function modeled by the Scoring Function, the Document Retriever model can be trained by minimizing the margin ranking loss [4]:

$$L = \max\left(0, 1 - S(\boldsymbol{q}, \boldsymbol{d}^+) + S(\boldsymbol{q}, \boldsymbol{d}^-)\right) \tag{3.18}$$

where $\boldsymbol{d}^+$ and $\boldsymbol{d}^-$ are the encodings of the positive and negative document, respectively. This loss function, the error rate is positive when $\boldsymbol{d}^+$ has lower score than $\boldsymbol{d}^-$. Otherwise, this value is 0 and the parameters of the model will not be updated. However, it does not make sense if the model stops learning when $S(\boldsymbol{q}, \boldsymbol{d}^+) = S(\boldsymbol{q}, \boldsymbol{d}^-)$. To prevent this situation, the margin value 1 is used to ensure that the model would still be improved upon when $S(\boldsymbol{q}, \boldsymbol{d}^+) - S(\boldsymbol{q}, \boldsymbol{d}^-) < 1$.

The entire network, from the Embedding Layer to the Scoring Function can be trained using backpropagation and mini-batch gradient descent. We use Adam optimizer to perform this training procedure. To reduce overfitting problem, Dropout and early stopping technique are also employed. The margin ranking loss function helps train the model to achieve adept question/document encodings and an effective scoring function at the same time by being a single objective that every parameters are tuned towards.

By using the margin ranking loss function, it is paramount to define what makes a document positive or negative in the first place. This depends largely on the training dataset. For the fact that we share the same problem as [46] where the ground-truth labels for the ranking task are not available, we employ their idea of *pseudo labels*. With the answers are the labels of the machine comprehension task, the documents at this step are labeled positive if they contain the exact match of the answer span.

How the training examples are selected is also crucial to the training process. For each question, there is usually a handful of positive documents while all the other ones are considered negative. It is infeasible to generate all possible training instances and train the model with them. This would only waste resources since the many of the negative documents are too easy to discriminate from the positive ones and will not contribute to the learning process. To reduce the number of negative documents, the simplest way is to randomly chose only n instances but it still

Algorithm 3.1: Pseudocode of the training procedure.

Input: Number of epochs with no improvement before stopping training (patience) p; Maximum number of negative documents n.

```
best_dev_acc ← 0
count_patience ← 0
while True do
    if count_patience == 0 then
        (Training examples generated by randomly selecting n negative
         documents, each is paired with all positive ones.)
    else
        (Training examples generated by selecting top-n highest-scored
         negative documents using the current saved model, each is paired with
         all positive ones.)
    end
    (Train the model with mini-batch gradient descent.)
    dev_acc ← (the accuracy on the development set)
    if dev_acc > best_dev_acc then
        (Save the current model.)
        count_patience ← 0
        best_dev_acc ← dev_acc
    else
        count_patience ← count_patience + 1
        if count_patience > p then
            break
        end
    end
end
```

does not guarantee that these instances are helpful. To effectively train the model, we need to provide examples that are hard enough by dynamically selecting the highest-scored negative documents. Nonetheless, this approach requires all the negative documents be processed with the latest set of parameters at every training step to find the top ones. While the model can be more capable, the training process will be slowed down dramatically.

Algorithm 3.1 demonstrate how the training procedure works. The early stopping mechanism is done by using the accuracy on the development set. To speed up the training process but still provide challenging examples for the model, we combine two negative sampling techniques: random and top-n. Normally, n random negative documents will be selected, so the sampling process is done quickly. However, when the model stops improving, current top-n highest-scored negative documents are used. This helps the optimizing process overcome local optima and keep improving since these are the most difficult, yet useful, training instances.

3.2 Document Reader

As briefly mentioned in 2.4, DrQA [7] is a popular open-domain QA and thoroughly evaluated with multiple standard datasets. Instead of using various knowledge sources as its previous works, DrQA only uses Wikipedia articles from which the answer to a given factoid question is selected. Moreover, it is designed with a clear pipeline approach which contains a Document Retriever and a Document Reader as typical. Because of this, it is easier for successive research to reuse and/or improve particular parts of the system. While the proposed Retriever is fairly simple, the Reader is a sufficiently complex and effective deep RNN trained for extracting answers span from a question and a list of documents. In order to focus on improving the document retrieval process and still have a complete open-domain QA system to evaluate the end performance, we utilize DrQA Reader and integrate it with the proposed Document Retriever. The following discusses DrQA Reader as well as the integration in a bit more detail.

3.2.1 DrQA Reader

After receiving a list of documents returned by the Retriever, the goal of the Reader is to predict the boundary of the text span within these documents that is most likely to be the answer to a given question. To tackle this, DrQA Reader comprises of three main modules: (1) Paragraph encoding, (2) Question encoding, and (3) Prediction.

Paragraph encoding. Since documents are usually lengthy which lessens the efficiency of RNNs, they are divided into n paragraphs. The paragraph encoding layer recognizes each paragraph $p = \{p_1, p_2, ..., p_m\}$, which is a sequence of m tokens, as one example and learns to convert it into a matrix representation with each row is the embedding of a token. Firstly, the authors construct a feature vector $\tilde{\boldsymbol{p}}_i$ for each token p_i by combining several information:

- The word embedding $f_e(p_i)$, which is taken from the pre-trained Glove word embeddings. Almost all these embeddings are kept fixed except 1000 most common question words such as "what", "when", "how", etc.
- The exact match indicator vector which contains three binary values signal whether p_i is in question q:

$$f_{em}(p_i) = \{\mathbb{I}(p_i \in q), \mathbb{I}(\text{lowercase}(p_i) \in q), \mathbb{I}(\text{lemma}(p_i) \in q)\}$$

- Some other token features include part-of-speech (POS) tag, named entity recognition (NER) tag and term frequency (TF) value:

$$f_t = \{\text{POS}(p_i), \text{NER}(p_i), \text{TF}(p_i)\}$$

- The aligned question embedding $f_a(p_i) = \sum_{j=1}^{l} a_{i,j} f_e(q_j)$, with q_j is one of l question words, $a_{i,j}$ is the attention score between p_i and q_j which is calculated as follow:

$$a_{i,j} = \frac{\exp\left(\alpha\left(f_e\left(p_i\right)\right) \cdot \alpha\left(f_e\left(q_j\right)\right)\right)}{\sum_k^l \exp\left(\alpha\left(f_e\left(p_i\right)\right) \cdot \alpha\left(f_e\left(q_k\right)\right)\right)}$$

  where $\alpha(\cdot)$ is a fully-connected feed-forward layer with ReLU as the activation function.

After obtaining a sequence of $\tilde{\boldsymbol{p}}_i$, a multi-layer bi-directional RNN is applied. The output of the paragraph encoding layer is:

$$\{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_m\} = \text{BiRNN}\left(\{\tilde{\boldsymbol{p}}_1, \tilde{\boldsymbol{p}}_2, ..., \tilde{\boldsymbol{p}}_m\}\right) \tag{3.19}$$

Question encoding. Instead of producing a sequence of vectors, each of which corresponds to a token, the question encoding outputs a single vector representation for the whole question q. The paper achieves this by employing an RNN

on the word embeddings of q to obtain $\{\boldsymbol{q}_1, ..., \boldsymbol{q}_l\} = \mathrm{RNN}\left(f_e\left(q_1\right), ..., f_e\left(q_l\right)\right)$. Then, the question embedding is $\boldsymbol{q} = \sum_j^l b_j \boldsymbol{q}_j$ with b_j determines how much of the corresponding word contributes to the final question vector:

$$b_j = \frac{\exp\left(\boldsymbol{w} \cdot \boldsymbol{q}_j\right)}{\sum_k^l \exp\left(\boldsymbol{w} \cdot \boldsymbol{q}_k\right)}$$

given the weight $\boldsymbol{w}$.

Prediction. At this phase, the authors build two independent classifiers, one for the answer span's start position and one for its end position:

$$P_{start}(i) \propto \exp\left(\boldsymbol{p}_i \mathbf{W}_s \boldsymbol{q}\right)$$

$$P_{end}(i) \propto \exp\left(\boldsymbol{p}_i \mathbf{W}_e \boldsymbol{q}\right)$$

The final answer prediction across all paragraphs is the sequence of tokens from position i to position j such that $P_{start}(i) \times P_{end}(j)$, $i \leq j \leq i + k$, is maximized, where k is the maximum answer's length allowed.

3.2.2 Training Process and Integrated System

The input for the Reader is a question, an answer, and a list of documents. A requirement from DrQA while training is that at least one document in the list must contain the exact match of the answer. This means that in the inference phase, DrQA always outputs an answer even when the answer is not available in the documents. How to prepare the list of documents for each training instance is also important. One way is to use all positive documents. By doing this, the model will learn to expect the answer to be presented in all of the provided documents which is not a case in realistic situation. Besides, when the system is integrated, the input documents of the Reader is the output of the Retriever, therefore, it does not guarantee that all returned documents are positive.

To simulate the inference phase of system while training the Reader, it would be best to present the model with the distribution of positive/negative documents produced by the Retriever. We achieve this by running the trained Document Retriever on the train dataset and then selecting 50 highest-scored documents. This means that there is a mix between positive and negative documents and we

find that this combination in the training data boosts the Reader's performance greatly.

After the Document Retriever and Document Reader are trained, the system is simply designed in a pipeline manner. In the QASA Retriever's running phase, for each question, all the documents in the database must be ranked by the network. This is bad for scaling or even impractical when the database gets extensive. One way to work around this is to use the QASA Retriever in conjunction with a simpler and faster retriever module. Even a method like filtering out all documents that do not have any overlap words with the question is able to reduce the number of documents drastically with minimal accuracy drop. This simple retriever acts as a loose filter and is applied before running the QASA Retriever.

Chapter 4

Experiments and Results

4.1 Tools and Environment

The Retriever is implemented using Python and TensorFlow[1]. TensorFlow is an end-to-end open source platform for machine learning which is developed by Google. It supports a comprehensive, flexible ecosystem of tools, libraries and community resources that has powered many state-of-the-art research in machine learning. The QASA Retriever's source code can be found at:

`https://github.com/trangnm58/QASA`

as well as its detailed instructions on how to train and use the model. To perform the experiments, the models are trained using the environment configuration presented in Table 4.1.

Table 4.1: Environment configuration.

Component	Specification	Quantity
CPU	Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz	2
RAM	16 GB DIMM ECC DDR4 @ 2400MHz	8
OS	Linux	-

[1]https://www.tensorflow.org/

4.2 Dataset

Both the Retriever and Reader are trained with the QUASAR-T dataset proposed by [12] using the official splits provided. This standard dataset consists of $43,012$ factoid questions obtained from numerous sources. Each question is associated with 100 pseudo-documents retrieved from ClueWeb09, a dataset that has about one billion web pages. The long documents contain no more than 2048 characters and the short ones contain no more than 200 characters. These documents have been filtered by a simple but fast retriever precedently and they now require a more sophisticated model to re-rank them efficiently. The answers for the given questions are free-form text spans, however, they are not guaranteed to appear in the documents which are challenging for both ranking and reading model. Figure 4.1 shows an example of a question associated with an answer and a list of pseudo-documents (contexts).

Question	Lockjaw is another name for which disease
Answer	**tetanus**
Contexts (partial)	As the infection progresses , muscle spasms in the jaw develop , hence the name lockjaw . The name comes from a common symptom of **tetanus** in which the jaw muscles become tight and rigid and a person is unable to open his or her mouth . **Tetanus** , commonly called lockjaw , is a bacterial disease that affects the nervous system .

Figure 4.1: Example of a question with its corresponding answer and contexts from QUASAR-T.

The statistics of QUASAR-T dataset are described in Table 4.2. As mentioned in 3.1.5, the dataset does not come with ground-truth labels for training the Retriever. Therefore, considering a question, if any document in the list of 100 pseudo-documents contains the exact answer within its text body, it is considered a positive document, otherwise, it's negative. Interestingly, there are instances in the dataset where none of their associated documents is positive. In these cases, the Retriever will always produce negative or unrelated documents. We call this type of instances is invalid. In Table 4.2, "Valid" indicates the number of instances in which the ground-truth answer is presented in at least one of the pseudo-documents. According to this, the upper bound for evaluating the performance of the retriever and the reader is the ratio between the number of valid

instances and the total number of instances. Particularly, for the test set, this upper bound is 77.37%.

Table 4.2: QUASAR-T statistics.

	Train	Validation	Test
Total	37,012	3,000	3,000
Valid	28,838	2,297	2,321

To evaluate the quality of QUASAR-T dataset, the authors from [12] employ several methods ranging from the simplest heuristics to state-of-the-art deep neural networks, and even acquire the output from human testers. According to their reports, the best model, which is BiDAF [41], achieves 28.5% while the human performance is 60.6%. It is worth noting that the human performance is still 16.77% lower than the upper bound calculated previously, which signifies the level of difficulty that the dataset presents.

As being an open-domain QA dataset, it is important for QUASAR-T to have questions about a variety of domains (e.g. music, science, food, etc.) Although the authors was unable to report a comprehensive categorization of the entire dataset, however, given 144 questions randomly selected from the development set, the annotators were able to categorize 214 genres of questions (one question can belong to multiple genres) and 122 entity-types of answers. The distribution the question genres and answer entity-types are demonstrated in Figure 4.2.

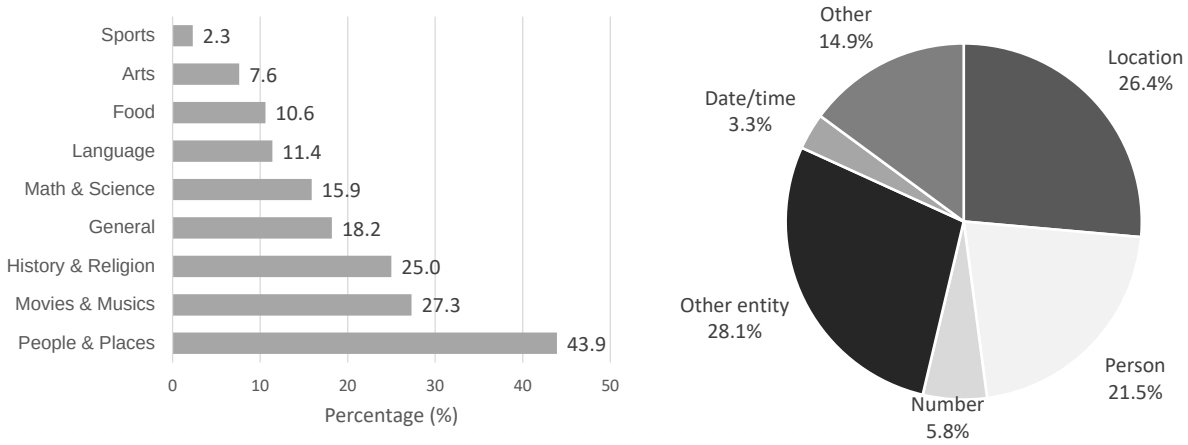

Figure 4.2: Distribution of question genres (left) and answer entity-types (right).

4.3 Baseline models

Our model is compared with four other proposed models that have results for the QUASAR-T dataset: GA [11], a reader that integrates a multi-hop architecture with attention mechanism for text comprehension; BiDAF [41], which uses bi-directional attention flow mechanism; R^3 [46], a novel Ranker-Reader system that is trained using reinforcement learning, and its simpler version, SR^2 [46], trained by combining two different objective functions from the ranker and reader. These models have been discussed briefly in 2.4.

GA and BiDAF are machine readers while R^3 and SR^2 are complete open-domain QA systems. Therefore, only R^3 and SR^2 have reported results for document retrieval task that can be compared with our model. These two models share the same Ranker (retriever) architecture; the only difference is that R^3 uses reinforcement learning to jointly train the Ranker and the Reader while SR^2 trains them separately just like our system. Their Ranker is also a deep learning model but it is very much different from ours. They deploy a variant of the Match-LSTM architecture [45] which produces the matching representations of the question and its N corresponding documents, denoted as $\mathbf{H}^{\text{Rank}} = \left\{\mathbf{H}_i^{\text{Rank}} \mid 1 < i < N\right\}$. Then, a standard max pooling technique is applied to each $\mathbf{H}_i^{\text{Rank}}$ to attain a vector $\boldsymbol{u}_i$. These vectors are concatenated together and non-linearly transformed into $\mathbf{C}$. The predicted probability of containing the answer for each document is an element of the vector γ, which is calculated by a normalization applied to $\mathbf{C}$. Based on γ, top-k documents is selected. Compared to our Retriever, the Ranker from [46] is much more complex with many deep layers and parameters; even the Match-LSTM layer alone is a convoluted network with six layers in total. This makes training the model more difficult since it requires a significant amount of time and resources. For the machine comprehension module, their Reader shares the same Match-LSTM layer with the Ranker and uses the outputted matching representations to compute the probability of the start and end position of the answer.

Besides comparing our system with other methods proposed in different papers, we also develop an internal baseline model to demonstrate the effectiveness of learning QASA document representations. In this model, we kickout the self-attention mechanism from the full model. That is, the Document Encoding Layer is constructed using the same architecture as the Question Encoding Layer. In subsequent section, this baseline model will be referred to as *kickout model*.

4.4 Experiments

4.4.1 Evaluation Metrics

To evaluate the Document Retriever and be comparable with other proposed methods, we employ top-k accuracy metric from [46]:

$$Top\text{-}k = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left(\exists d^{+} \in \mathbb{D}_i^{\star}\right) \tag{4.1}$$

which states that the top-k documents, $\mathbb{D}_i^{\star}$, for the i-th question are considered correctly retrieved if they include at least one positive document, d^{+}.

The performance of the Document Reader is also regarded as the performance of the overall system since it is the last module of the pipeline. To evaluate the Reader, two widely used metrics is applied, which are F1 and Exact Match (EM) [38]. Specifically, F1 measures the overlap between two bags of tokens that correspond to the ground-truth and predicted answer:

$$F1 = \frac{1}{N}\sum_{i=1}^{N}\frac{|\boldsymbol{g}_i \cap \boldsymbol{p}_i|}{|\boldsymbol{g}_i|} \tag{4.2}$$

where for the i-th example, $\boldsymbol{g}_i$ and $\boldsymbol{p}_i$ are sets of tokens in the ground-truth and the predicted answer, respectively. While F1 allows the predicted answers to match partially with the ground-truths, EM strictly compares the two strings to check whether they are equal or not:

$$EM = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(g_i = p_i) \tag{4.3}$$

where g_i and p_i are the text strings of the ground-truth and predicted answer of the i-th example, respectively.

4.4.2 Document Retriever

4.4.2.1 Hyperparameter Settings

There are many hyperparameters defined in order to train the QASA Retriever, all of which are listed in Table 4.3. Most of these hyperparameters are chosen based on the model's performance on the validation set.

Table 4.3: Hyperparameter Settings

Component	Hyperparameter	Setting
Embedding	Token embedding	300
	Character embedding	50
	Character BiLSTM units	50
Question Encoding	Encoding size	128
Document Encoding	Encoding size	128
	Fully-connected units	200
Scoring Function	Fully-connected units	50
Shared Layer	Contextual BiLSTM units	150
General	Batch size	32
	Optimizer	Adam
	Learning rate	0.001
	Random initializer	Glorot normal
	Dropout rate	0.5
	Top-*n* negative sampling	20

4.4.2.2 Results

The results for our Document Retriever is presented in Table 4.4 as it is compared with two other models that have results reported for the QUASAR-T dataset. As discussed, R^3 [46] jointly trains the document retrieval and answer extraction module simultaneously using reinforcement learning. By the mean of the rewarding scheme, their ranker can gain some insight into the reader's performance while being trained. This helps R^3 mitigate the cascading error problem that most pipeline systems with independently trained modules, like ours, suffer from and boosts its recall remarkably. As the result, their ranker has higher recall in top-1 and top-3 than QASA although being slightly lower in top-5. Another model from [46] is SR^2 which is a simpler variant of R^3. Because SR^2 is not benefited by joint learning, its ranker is more comparable to our model. To this end, QASA shows more favorable results where it achieves 3.87% and 1.53% higher than SR^2 ranker in top-1 and top-3 respectively.

When comparing with our *kickout model*, which only uses a feed-forward layer instead of self-attentive mechanism for document encoding, QASA also produces surpassing results among all top-*k* accuracy values. Concretely, by using

Table 4.4: Evaluation of retriever models on the QUASAR-T test set.

	Top-1	Top-3	Top-5
SR^2 ranker	28.80	46.40	54.90
R^3 ranker	**40.30**	**51.30**	54.50
QASA Retriever	32.67	47.93	**54.90**
Kickout model	32.43	46.57	53.20

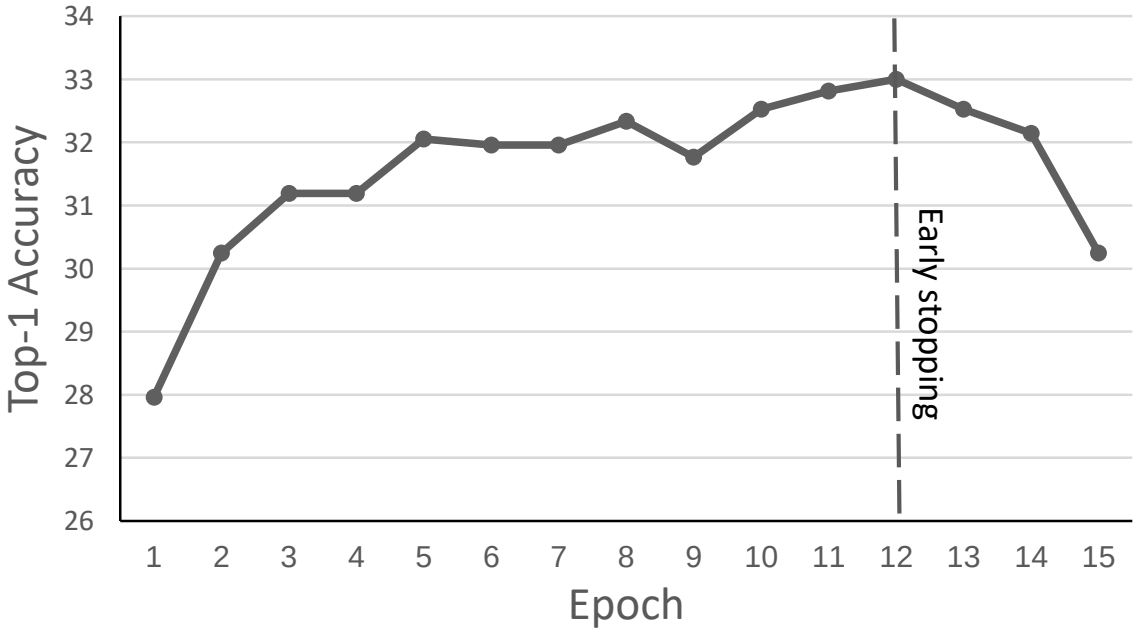

Figure 4.3: Top-1 accuracy on the validation dataset after each epoch.

the QASA document representation, the model gives an improvement of 1.7% in top-5. This results, in fact, have proven our hypothesis.

To analyse the results further, we plot a line chart, shown in Figure 4.3, represents the top-1 accuracy on the validation set evaluated after each epoch. Since the training process adopts Early Stopping technique, it waits for 3 epochs without any improvement until stopping. The best accuracy on the validation set is at the 12-th epoch, so the saved model at that epoch is considered the best model and it is evaluated on the test set for final results.

Figure 4.4 depicts another line chart that represents the training loss calculated at the end of each epoch. There are a few noticeable peaks in the diagram which are after the 4-th, 6-th, 9-th, and 13-th epoch. These peaks are correlated with the top-1 accuracy diagram shown in Figure 4.3. Referring back to the train-

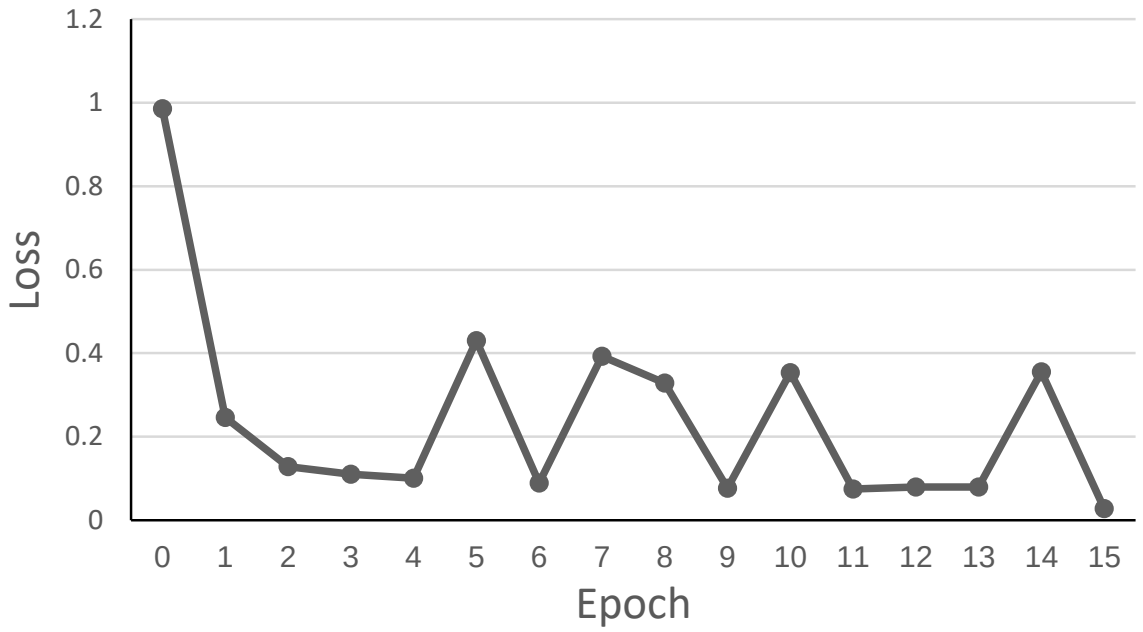

Figure 4.4: Loss diagram of the training dataset calculated after each epoch.

ing Algorithm 3.1, whenever the accuracy stops improving, the negative sampling technique switch from randomize approach to selecting top-*n* highest-scored negative documents using the latest model. As can be seen from Figure 4.3, the model does not improve at the 4-th, 6-th, 9-th, and 13-th epoch, same as listed previously. Since the negative documents sampled after these epochs are highest-scored, they present the hardest training examples for the Retriever. Consequently, the loss values calculated at the epochs after the corresponding listed epochs are peaks. Despite the fact the loss values increase, the accuracy rates also increase at these epochs which indicates that this negative sampling technique helps boosting the model's performance. Furthermore, it can be considered a training technique to get the optimization process out of local optima.

4.4.3 Overall system

The overall results of the proposed system are demonstrated in Table 4.5 along with several other open-domain QA systems. As can be seen from the table, QASA consistently offers better results than the *kickout model* when integrated with DrQA Reader, which proves once again the effectiveness of question-aware self-attentive mechanism. Specifically, QASA outperforms the *kickout model* by 1.68% in F1 and 2.13% in EM.

The results of BiDAF and GA model are presented in [12]. Since they are machine readers, in order to acquire the overall results of the system, they are integrated with a simple retriever. Despite being state-of-the-art machine comprehension models for their reported datasets, both BiDAF and GA give particularly poor results for the QUASAR-T dataset. This can demonstrate that the reader depends greatly on the retriever. Without a good enough retriever, the reader could become useless. When comparing with two systems from [46], our system excels both of them by a large margin, especially with R^3 (4.17% in F1 and 6.3% in EM) in spite of the fact that our Retriever and the Reader are trained independently.

Table 4.5: The overall performance of various open-domain QA systems.

	F1	EM
BiDAF	28.50	25.90
GA	26.40	26.40
SR^2	38.80	31.90
R^3	40.90	34.20
QASA Retriever + DrQA Reader	**45.07**	**40.50**
Kickout model + DrQA Reader	43.39	38.37

It is worth noting that the QUASAR-T dataset does not provide ground-truth for document retrieval, therefore, this module is evaluated using *pseudo labels*. A limitation of *pseudo labels* is that the positive documents are not guaranteed to be relevant to the question. For example, given the question "What is the smallest state in the US?", one of its positive documents is "1790, Rhode Island ratifies the United States Constitution and becomes the 13th US state" (it contains the answer, "Rhode Island"). However, this positive document does not help the reader since it is completely irrelevant. For the reader to extract the answer, not only the retrieved document must enclose the exact string but also it must convey information related to the query. For that reason, even though our Document Retriever has lower recall than R^3 ranker, its outputted documents are semantically similar to the question, thus, they are more useful to the Reader which results in a much higher performance of the overall system.

Conclusions

Following the work done in [7, 46], the thesis proposed an open-domain QA system that has two main components: a Document Retriever and a Document Reader. Specifically, the Document Retriever, called QASA, is an advanced deep ranking model which contains (1) an Embedding Layer, (2) a Question Encoding Layer, (3) a Document Encoding Layer, and (4) a neural Scoring Function. The thesis hypothesizes that in order to effective retrieve relevant documents, the Retriever must be able to comprehend the question and automatically focus on some important parts of the documents. Therefore, we proposed a deep neural network to obtain question-aware self-attentive document representations and then used pairwise learning to rank approach to train the model. A complete open-domain QA system is constructed in a pipeline manner combining the QASA Retriever with the Reader from DrQA. Having analyzed the results of QASA compared to the kickout model, we demonstrate the effectiveness of using question-aware self-attentive encodings for document retrieval in open-domain QA. We also show that the Retriever has a substantial contribution to the overall system and by improving the Retriever, we can extend the upper bound of machine reading module markedly.

Although the method shows promising results compared to several baseline models, some of which are even state-of-the-art, there are still many limitations that the model suffers such as the cascading error from the Retriever to the Reader. In the future, we will re-design the architecture so that the Retriever and the Reader can be jointly trained as in [46] and try to mitigate this cascading error problem. To evaluate the system even further, we will adopt more standard datasets such as SQuAD and TREC.

List of Publications

[1] **T. M. Nguyen**, Van-Lien Tran, Duy-Cat Can, Quang-Thuy Ha, Ly T. Vu, and Eng-Siong Chng, "QASA: Advanced Document Retriever for Open Domain Question Answering by Learning to Rank Question-Aware Self-Attentive Document Representations," in *Proceedings of the 3rd International Conference on Machine Learning and Soft Computing*, ACM, 2019, pp. 221-225.

References

[1] A. Agarwal, H. Raghavan, K. Subbian, P. Melville, R. D. Lawrence, D. C. Gondek, and J. Fan, "Learning to rank for robust question answering," in *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2012, pp. 833–842.

[2] J. R. Anderson, *Cognitive psychology and its implications*. Macmillan, 2005.

[3] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[4] B. Bai, J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger, "Learning to rank with (a lot of) word features," *Information retrieval*, vol. 13, no. 3, pp. 291–314, 2010.

[5] H. Bast and E. Haussmann, "More accurate question answering on freebase," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 2015, pp. 1431–1440.

[6] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[7] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading wikipedia to answer open-domain questions," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2017, pp. 1870–1879.

[8] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised learning of universal sentence representations from natural language inference data," in *Proceedings of the EMNLP*, 2017, pp. 670–680.

[9] Y. Cui, Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu, "Attention-over-attention neural networks for reading comprehension," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2017, pp. 593–602.

[10] T. H. Dang, H.-Q. Le, T. M. Nguyen, and S. T. Vu, "D3ner: biomedical named entity recognition using crf-bilstm improved with fine-tuned embeddings of various linguistic information," *Bioinformatics*, vol. 34, no. 20, pp. 3539–3546, 2018.

[11] B. Dhingra, H. Liu, Z. Yang, W. Cohen, and R. Salakhutdinov, "Gated-attention readers for text comprehension," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2017, pp. 1832–1846.

[12] B. Dhingra, K. Mazaitis, and W. W. Cohen, "Quasar: Datasets for question answering by search and reading," *arXiv preprint arXiv:1707.03904*, 2017.

[13] C. dos Santos and V. Guimarães, "Boosting named entity recognition with neural character embeddings," in *Proceedings of the Fifth Named Entity Workshop*, 2015, pp. 25–33.

[14] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

[15] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[16] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the AISTATS*, 2010, pp. 249–256.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[18] E. Grave *et al.*, "Learning word vectors for 157 languages," in *Proceedings of the LREC*, 2018.

[19] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[20] B. F. Green Jr, A. K. Wolf, C. Chomsky, and K. Laughery, "Baseball: an automatic question-answerer," in *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*. ACM, 1961, pp. 219–224.

[21] R. Herbrich, "Large margin rank boundaries for ordinal regression," *Advances in large margin classifiers*, pp. 115–132, 2000.

[22] 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," *arXiv preprint arXiv:1608.03542*, 2016.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[25] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush, "Character-aware neural language models," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[27] O. Kolomiyets and M.-F. Moens, "A survey on question answering technology from an information retrieval perspective," *Information Sciences*, vol. 181, no. 24, pp. 5412–5434, 2011.

[28] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," in *Proceedings of NAACL-HLT*, 2016, pp. 260–270.

[29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[30] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[31] T.-Y. Liu *et al.*, "Learning to rank for information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

[32] Y. Ma, E. Cambria, and S. Gao, "Label embedding for zero-shot fine-grained named entity typing," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016, pp. 171–180.

[33] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[34] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[36] A. Mishra and S. K. Jain, "A survey on question answering systems with classification," *Journal of King Saud University-Computer and Information Sciences*, vol. 28, no. 3, pp. 345–361, 2016.

[37] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[38] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[39] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[40] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[41] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension," in *Proceedings of ICLR*, 2017.

[42] Y. Shen, P.-S. Huang, J. Gao, and W. Chen, "Reasonet: Learning to stop reading in machine comprehension," in *Proceedings of the 23rd ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining. ACM, 2017, pp. 1047–1055.

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[44] E. M. Voorhees *et al.*, "The trec-8 question answering track report," in *Trec*, vol. 99. Citeseer, 1999, pp. 77–82.

[45] S. Wang and J. Jiang, "Learning natural language inference with lstm," in *Proceedings of NAACL-HLT*, 2016, pp. 1442–1451.

[46] S. Wang, M. Yu, X. Guo, Z. Wang, T. Klinger, W. Zhang, S. Chang, G. Tesauro, B. Zhou, and J. Jiang, "R3: Reinforced ranker-reader for open-domain question answering," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[47] W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou, "Gated self-matching networks for reading comprehension and question answering," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, vol. 1, 2017, pp. 189–198.

[48] W. A. Woods, R. M. Kaplan, B. Nash-Webber *et al.*, "The lunar sciences natural language information system: Final report," *BBN report*, vol. 2378, 1972.

[49] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[50] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.