Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

# Reading Comprehension using Dynamic Memory Networks

**Masterthesis**
at Language Technology Research Group
Prof. Dr. Chris Biemann

Department of Informatics
MIN-Faculty
Universität Hamburg

submitted by
**Ibrahim Dahmash**
Course of study: Intelligent Adaptive Systems
Matrikelnr.: 6756453
on
22.11.2018

Examiners: Prof. Dr. Chris Biemann
Dr. Alexander Panchenko

Advisor: M.Sc. Benjamin Milde

# Abstract

This thesis presents steps toward Question Answering (QA) task for Reading Comprehension (RC). RC is about reading a piece of text and understanding the meaning of its words. The ability to answer questions answered in a context is an essential factor for RC. Therefore, it forms the question answering task in order to generate answers to questions which might be answered in the given RC context. QA is increasingly present in the fields of Text Mining, Information Retrieval and Natural Language Processing (NLP). The aim is to develop the human-like ability to read some piece of text and then answer questions about it. In the traditional approaches, QA task was proposed based on a structured knowledge-base or information retrieval concepts. In recent years, deep learning techniques have had promising ways to deal with human-textual data in tasks such as QA.

As it is always the case with deep learning models, the more massive datasets used for training and evaluation, the more accurate the results we get. Our target is to explore and implement an end-to-end QA system on a manually generated large-scale dataset. One such dataset is Standford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) which consists of input-question-answer triplets. We propose to adopt the Dynamic Memory Network (DMN) approach introduced by Kumar et al. (2016) as the core model to be implemented in this thesis. Our improved DMN architecture will be the first attempt in SQuAD leaderboard using the DMN approach. We also re-implement Match-LSTM with Answer Pointer (Wang and Jiang, 2016) and the original Kumar et al. (2016) DMN models. The former two models, along with the baseline and our improved DMN models are trained and evaluated on SQuAD dataset. Finally, we show that our improved DMN model can compete with Match-LSTM and other models in the SQuAD leaderboard.

# Zusammenfassung

In dieser Abschlussarbeit werden Schritte zum Erstellen eines Question Answering (QA) Systems im Kontext für Leseverständnis dargelegt. In Leseverständnis geht es um das Lesen eines Textes und das gleichzeitige Verstehen der Bedeutung der Wörter. Die Fähigkeit Fragen in einem bestimmten Kontext beantworten zu können ist ein essenzieller Faktor für das Leseverständnis. Das Leseverständnis bildet daher das QA-System in einer Weise, um Antworten für Fragen zu generieren, welche eventuell im gegebenen Leseverständnis behandelt werden. QA ist immer gegenwärtiger in den Bereichen Textmining, Information-Retrieval und Natural Language Processing (NLP). Das Ziel ist es menschenähnliche Fähigkeit für das Lesen eines Textes und das anschließende Beantworten von Fragen zu diesem zu entwickeln. In traditionellen Ansätzen beruht das QA-System auf Konzepten von strukturierten Wissensdatenbanken oder Information-Retrieval. In den letzten Jahren haben Deep Learning Methoden erfolgreiche Ansätze gezeigt, um mit menschlichen Textdaten in Aufgaben wie QA umzugehen.

Sowie es immer der Fall mit Deep Learning Modellen ist, liefert die Verwendung von massiveren Datensätzen zum Training und Evaluieren genauere Ergebnisse. Unser Ziel ist es ein Ende-zu-Ende QA-System anhand eines manuell erzeugten, umfangreichen Datensatzes zu untersuchen und umzusetzen. Eine solcher Datensatz ist der Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016), welcher aus Input-Frage-Antwort Tripletts besteht. Wir schlagen vor, den Dynamic Memory Network (DMN) Ansatz von Kumar et al. (2016) als Kernmodell für diese Arbeit zu verwenden. Unsere verbesserte DMN Architektur wird der erste Ansatz in der SQuAD-Rangliste sein, welcher eine DMN Methode verwendet. Wir auch implementieren Match LSTM with Answer Pointer (Wang and Jiang, 2016) und die ursprünglichen Kumar et al. DMN Modelle neu. Diese werden zusammen mit der Baseline und unserern verbesserten DMN Modellen auf dem SQuAD-Datensatz trainiert und evaluiert. Abschließend zeigen wir, dass unser verbessertes DMN-Modell mit Match-LSTM und den anderen Modellen in der SQuAD-Rangliste konkurrieren kann.

# Acknowledgments

I would first like to thank my thesis advisor, M.Sc. Benjamin Milde. His office door was always open whenever I ran into a trouble spot or had a question about my research or writing. He always consistently wanted this thesis to be my own work and steered me in the right direction whenever he thought I needed it.

I would also like to thank Prof. Dr. Chris Biemann, the head of the Language Technology group at the University of Hamburg, for his efforts to make my work on this thesis as smooth as possible, beginning from the first brainstorming meeting until the end of the thesis.

Finally, I must express my appreciation and profound gratitude to my parents, siblings and friends for their endless support and continuous moral encouragement throughout my years of study here. You will remain the beginning and cornerstone of any achievement in my life, and I present this accomplishment to you as a sign of my thankfulness. A special thanks to my deceased father for the sacrifices he made for me to get the best possible education.

Your efforts have been greatly appreciated.

Thank you.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In the past, computers had superhuman capabilities to solve mathematical problems. Programs using data (as the input) run on computers to produce the output; they were programmed using traditional ways. However, concerning complex problems such as object recognition, driverless cars, stock-market prediction, memorise entire documents or movies in a single glance, and natural Language tasks to give people information on any subject were very difficult to be solved using the traditional programming. Recently, machine learning concepts made it easy to solve these complex problems. Machine learning is a field of Artificial intelligence (AI) to allow computers automatically learn by themselves using statistical techniques. Unlike the traditional programming, machine learning concepts allow the data (as the input) and the prior-known output to run on computers and then produce the program. Lastly, this program runs on computers with new data to produce the output. As computing systems are becoming more complex and the advances in computer technology increase, the AI field became indispensable to keep up with these developments. AI has experienced a dramatic evolution in our living applications. In the heart of these developments, the development of Natural Language Processing (NLP) field, which it is revisited again in this thesis.

Reading comprehension (RC) is not an easy approach to achieve because we have to combine natural language processing concepts and knowledge of the world. RC is about reading a piece of text and understanding the meaning of its words. RC should has the ability to answer questions answered in its context. Therefore, RC forms the Question Answering (QA) task. Recently, QA is becoming an essential task in Text mining and NLP fields. QA should automatically allow users to get answers to their questions which might be answered in the given RC context. Today, QA appears in search engines (Google), conversational phone interfaces (Wolfram Alpha, Apple Siri) and intelligent personal digital assistants (Microsoft Cortana). There are two types of datasets used in the reading comprehension approach: automatically and manually generated ones. Automatically generated datasets are artificially generated datasets and one of these datasets is the Cloze-style test. The cloze-style test is the best way to create reading skills test and typically used in the English language test in schools. The test contains complete the sentences task,

which missing words (deletion word) in those sentences have to be filled in. Another example of the artificially generated datasets is introduced by Weston et al. (2015) and it called Facebook bAbI dataset. In contrast to that, the manually generated datasets are closer to our QA system goals. They are about letting users ask questions about a whole paragraph and then receive the probable answers based on it. Usually, these manually generated datasets are small and insufficient for training deep learning models such as MCTest dataset (Matthew Richardson, 2013). SQuAD dataset solves this issue since it is one of the largest manually generated datasets. It contains contexts with over 100,000 question-answer pairs on more than 500 Wikipedia articles. SQuAD was recently introduced by Rajpurkar et al. (2016) as a large-scale dataset for the reading comprehension approach. SQuAD is now an invaluable dataset to train and evaluate deep neural network models for NLP tasks. Accordingly, we decided to explore the performance of different RNNs successors concerning the domain of QA using textual data on SQuAD dataset. Furthermore, we examine how well the semantics and syntactic of a given text can be understood and utilized by neural networks containing memory and attention mechanisms to answer questions posed in standard natural language by humans.

## 1.1 Motivation

Nowadays, challenges to keep up with the aspirations of people are getting bigger. Therefore, working on the field of automatic learning with neural networks and on large data (Big data) is rising. AI was mainly introduced to do what humans failed or could not to do. On the subject of this thesis, consider asking some humans to read and memorize an article on their working memory (brain memory), and then we will ask them to answer some questions about that article. Even though those humans are incredibly smart, they will not be able to answer most of those questions correctly. The reason is that we can not store everything in our working memory in a few moments. The solution to this difficulty is to allow as many glances as possible to read the article multiple time to be able to answer those questions. Another solution is about asking them to read questions first before reading the article. By this way, questions keywords will direct their attention when they read the article afterwards. That is how it works with humans. On the other hand, AI usually tries to simulate how our brains work in different tasks like this one. Fortunately, in neural networks, the problem mentioned beforehand has been solved by the same solutions. For the idea of the first solution which it allows as many glances as possible is called memory mechanism. Also, the idea of the second solution which is about using questions keywords to direct the attention of readers is called attention mechanism.

Currently, neural networks with memory and attention mechanisms exhibit precise reasoning capabilities required for QA task. In this light, we decided to explore and present the performance of the memory networks concerning the domain of the QA task. Following RNN, LSTM, GRU, and right up to MemNN, Dynamic

Memory Network (DMN) is becoming the state-of-the-art in this direction. DMN is another extended version of MemNN but with dynamic memory and attention mechanism usage. DMN is submitted for the first time by means of Kumar et al. (2016) which they implemented an end-to-end trainable network using artificially generated dataset called Facebook bAbI. This dataset is released by Weston et al. (2015) in Facebook AI research-lab. Kumar et al. DMN architecture has four modules: input, question, episodic memory and answer respectively. The input module encodes raw input text sequences (context) into distributed vector representations. The question module similarly encodes the question sentence. Then, input and question representations are fed into the episodic memory module. The memory module uses the attention mechanism to choose which sentences in the input representations are most important to focus on to answer questions. Then, it iteratively produces a memory vector space representing all the relevant information to the question. In the end, all the relevant information are used by the answer module to extract the answer. Throughout this thesis, we propose a new improved version of DMN architecture, and we called it improved DMN. Our improved DMN consists of some improvements in two modules: the input and episodic memory. In Chapter 6, we show our experiments and results of our improved DMN regarding QA task which it trained and evaluated on SQuAD dataset.

## 1.2 Research Questions

SQuAD dataset contains much more challenging questions whose correct answers can be any sequence of tokens from the given context. Moreover, SQuAD covers a diverse range of topics across a variety of domains. Unlike some other datasets whose questions and answers are created automatically and artificially, questions and answers in SQuAD dataset were generated by humans through crowd-sourcing. That makes SQuAD more realistic and reliable dataset for our QA task. In addition to, answers in SQuAD could consist of multiple words, not just a single word. As shown in Figure 1.1, there is an example of one context from the "Teacher" paragraph in SQuAD dataset. Each task in SQuAD consists of three-tuple (C, Q, ($A_s$, $A_e$)): $C$ refers to the context, $Q$ refers to questions and ($A_s$, $A_e$) refers to the extracted answer span to these questions. In SQuAD, the answer could be a single word or multiple words retrieved from a list of candidates of answers. The answer-span ($A_s$, $A_e$) is used to predict the probable answers in a variable length or range of two tokens, where $A_s$ refers to the answer start token and $A_e$ refers to the answer end token. One of the challenges in SQuAD is the lexical variation problem because of synonyms in both the context and questions are different. In order to solve this problem, we should use an external knowledge base to match synonyms in the context with those in questions. Another big challenge in SQuAD is the syntactic variation problem, which it compares the syntactic structure of the question and the context. We trained a word vector representation tool called Glove (Pennington et al., 2014) to solve these problems.

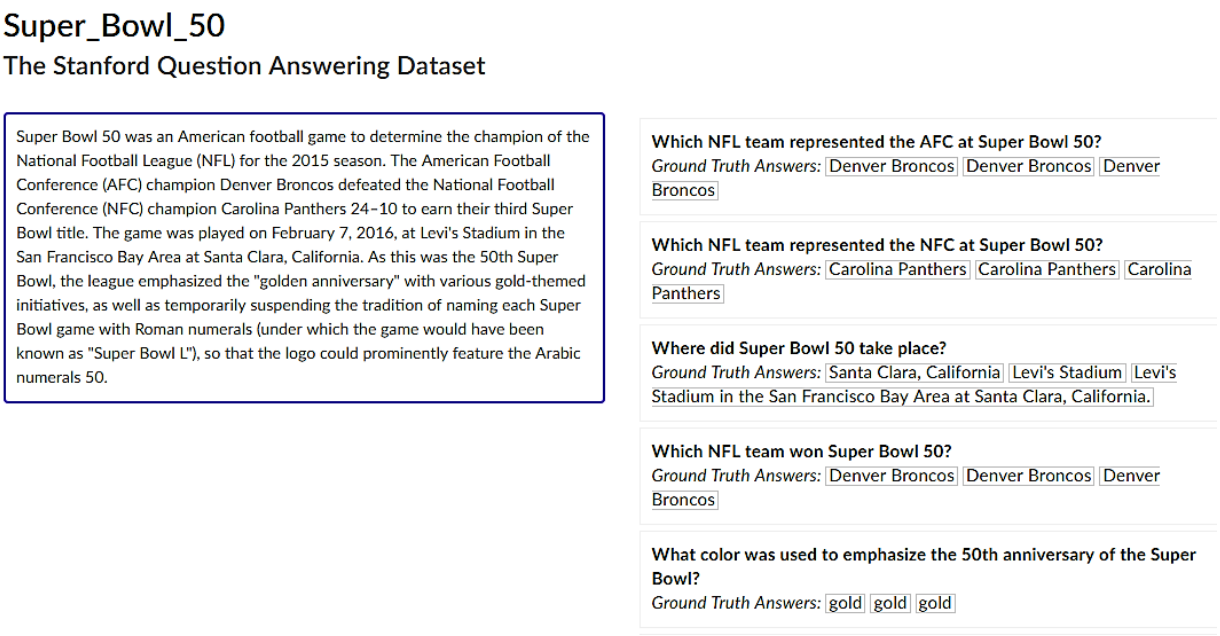


Figure 1.1: A training example from the SQuAD dataset, consisting of a context, questions and the ground truth and prediction answers (Rajpurkar et al., 2016).

Finally, Our Research Questions are:

1. **RQ-1**: How good is the performance of our improved DMN when evaluated on a large-scale dataset such SQuADv1.1?
2. **RQ-2**: How our improved DMN can compete others architectures in SQuAD leaderborad, especially: Match-LSTM with Answer Pointer (Wang and Jiang, 2016)?

In the following chapters, we are going to answer these questions.

## 1.3 Structure of Thesis

In this section, we provide the structure of the thesis as shown in Figure 1.2. By outlining our research questions, they will be solved in the following chapters as follows:

**Chapter 2** discusses the theoretical background for the domain of QA task and methods of neural networks. This chapter consists of two sections related to QA task. The first section presents the traditional QA task such as information extraction and retrieval methods with the knowledge base. The second section presents the current and recent QA task which implemented by neural systems. In the beginning, we give a brief theoretical background of the biological neural network. Afterwards, we talk about the so-called Artificial Neural Network (ANN) or Neural Network (NN) and its methods that were implemented in this thesis. Throughout this thesis, we tried many NN methods such as vanilla (RNN, LSTM, GRU), Bidirectional-LSTM, Bidirectional-GRU and MemNN in our implementations.

**Chapter 3** provides a quick feasible study of the most important researches and attempts regarding the domain of QA task. This chapter consists of two sections: Single-step models and Multi-step models. For the single-step models, the merging of the question and context occur only once before generating the final answer. In contrast, the multi-step models are closer to how humans think by reading the context more than once to answer the questions.

**Chapter 4** outlines and proposes the methods and technologies used to deal with SQuAD dataset in the domain of the QA task. This chapter consists of three sections: word embedding, attention mechanism, encoder-decoder architecture and approaches respectively. Firstly, we give a brief about the word embedding technique that is used to convert the raw data to high dimensional numerical vector space. Afterwards, we present the attention mechanism technique and how it recently became promising in most of the NLP tasks. Then, we propose the encoder-decoder architecture that has been implemented through all the implementations in this thesis. Finally, we propose the four approaches that were implemented and evaluated by SQuAD-v.1 dataset in this thesis. The four main approaches are vanilla LSTM as a baseline model, the re-implementation of match LSTM with answer pointer model by Wang and Jiang (2016) as a comparative model, Kumar et al. (2016) DMN model and our improved DMN model.

**Chapter 5** presents the data bank used for the evaluation in this thesis. This chapter consists of two sections: The first section is about SQuAD as our evaluation dataset. Following by the section that provides details of the preprocessing stage and the important configurations.

**Chapter 6** shows the final results of the four models. This chapter consists of two sections: Technical Description and Technical Results. For Technical Description

section, we cover the environment stuff information which was used in this thesis such as programming languages and the type of processing unit. Then, we provide all the results and tables of the scores achieved in the four models in the Technical Results section.

**Chapter 7** discusses the thesis approach, all the experiments implemented and the gap in results between the four models. In this chapter, we have two sections: conclusion and future work. For the conclusion section, we propose conclusions of the theoretical and technical stages at the same time as an open discussion. Moreover, we show the scientific contribution points of the thesis by summarizing the results achieved. For future work section, we share our future work from what we have done in this thesis and then discussing potential future works.

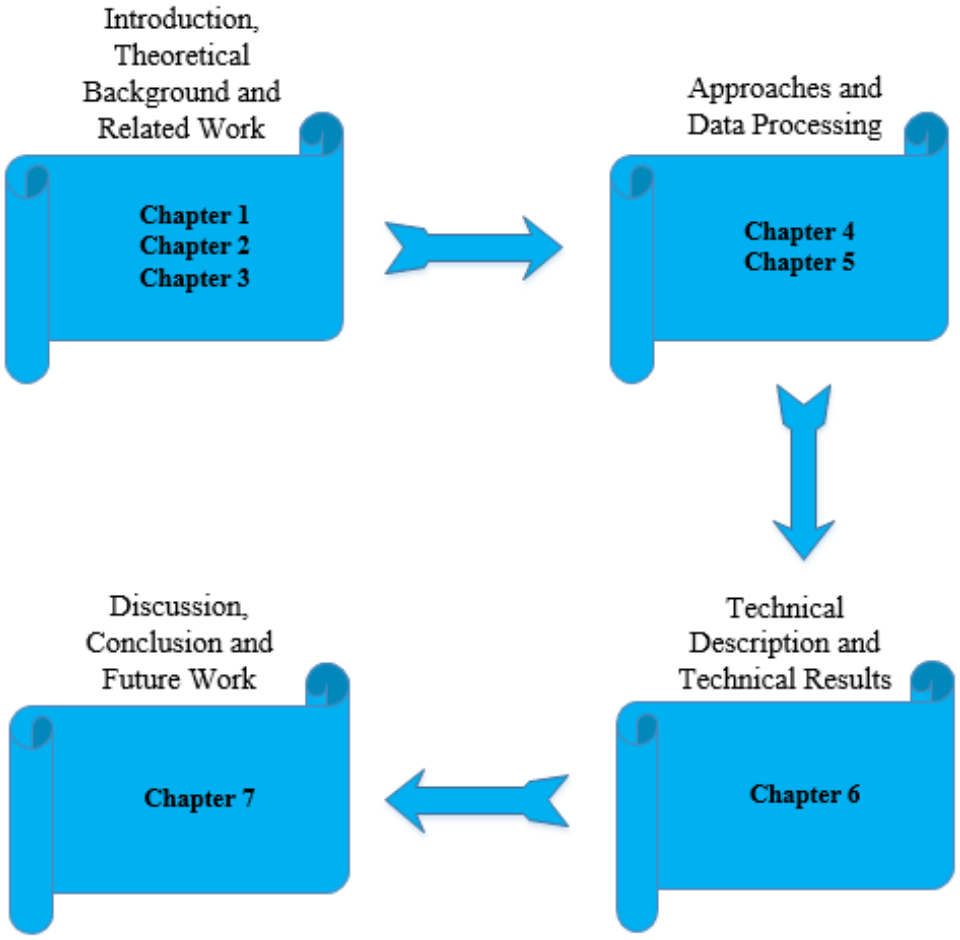


Figure 1.2: Structure of the thesis.

# Chapter 2

# Theoretical Background

## 2.1 Traditional Question Answering

First of all, have a minute and think about most humans problems in real life. They could be cast into a form of question answering. With the enormous revolution in modern technology, everything has to be automatic. Consequently, the target of our QA system is to answer questions posed in natural language by humans automatically. QA is becoming one of the most rising tasks in fields such as NLP and Text Mining. Nowadays, the QA task can work on any dimension of data and different semantic complexity levels. The simple level could be questioned about Name, Date, Time and currency about a few sentences. The complexity level could appear in open-ended questions with grammatical co-references or infinity answers such as "What is the meaning of life?". Therefore, the QA task has no limits by default. Thus far, QA is still an active research field looking for researchers.

QA system can be divided into non-factoid or factoid QA. In non-factoid QA system, users can ask the system about any question such as in Google search engine. Non-factoid QA system should automatically retrieve all probable answers by extracting causal relations from the knowledge base. The answer in non-factoid QA system could be more complex such as descriptions, opinions, articles, long story and so on, so forth. In contrast to that, factoid QA system allows users to ask about concise facts which might consist of person name, organization name, numeric expression, Date, Time and so on, so forth. Then, factoid QA system retrieves facts words in answers. Factoid QA system is considered as a subset of the non-factoid QA. Following questions are a simple example of factoid QA which questions can be answered with a short text expressing a personal name, temporal expression, numbers, currency or location based on a few sentences that lead to these answers:

- Where are the Pyramids located? Egypt.
- What currency is used in China? Yuan.

Last but not least, the traditional QA system first has been addressed using Information Extraction and Retrieval methods. After the significant progress of Artificial Neural Network (ANN), it motivated researchers to examine and implement QA system based on ANN.

### 2.1.1 Information Retrieval

Concerning the domain of the QA task, two modules could be used in the traditional QA: preliminary analysis of context text sequences (Information Extraction) and query processing (Information Retrieval) (Jurafsky and Martin, 2000). Information retrieval (IR) deals with the structured representation analysis which has been extracted from the so-called Information Extraction method. The primary task of IR is to retrieve the information that user needs as text sequences. IR is considered the low-level information extraction and one of the famous systems of IR is the Google search engine. Google engine allows a user to enter his input (Question, word or sentence) then the engine analyses the input to extract structured information. Then, this engine searches with this information to retrieve the answers back to the users using the IR method. As shown in Figure 2.1, a user searched "Where are the great pyramids located?". Then, the Google engine gave answers back to the user. However, the IR in the Google engine is a non-factoid QA system; it could consider as IR of complicated questions. This process depends on query based-summarization (Jurafsky and Martin, 2000) concept to retrieve all possible information to the user. Contrary to the less complicated system which is implemented by the factoid QA system based on IR, the factoid QA system retrieves a short answer based on facts. By searching for the relations between questions and context, the candidate sentences that might contain answers will be retrieved. Afterwards, by filtering these sentences, we can rank the candidate words (answers) from each sentence to retrieve the answer with the highest rank.

Information Extraction (IE) is a method for extracting the structured information from the unstructured one. IE is also used for understanding relevant parts of texts and try to gather all possible information to retrieve more optimal answers. Afterwards, the IR method can be used to develop an intelligent QA system by identifying the relations between words in question and context text sequences. Named Entity Recognition (NER) (Finkel et al., 2005) is used to classify each word in the context to a specific name entity such as Person, Organization, Location, Date, Time, Money, Profession and others. After linking all of these named entities, we can then extract the information that could help to answer questions, and most of these answers are named entities. The relations between named entities could be extracted by using Part Of Speech (POS) Tagging term (Toutanova et al., 2003) or sometimes called grammatical tagging. POS is used for mapping each word in the context to the corresponding POS tag such as VBD, VBG, VBZ, and others. POS plays a critical role in the grammatical tagging stage and how to apply the morphology (Lemma) term in a word. Morphology term is used for studying the meaning of each word and relationships between words in a sentence.

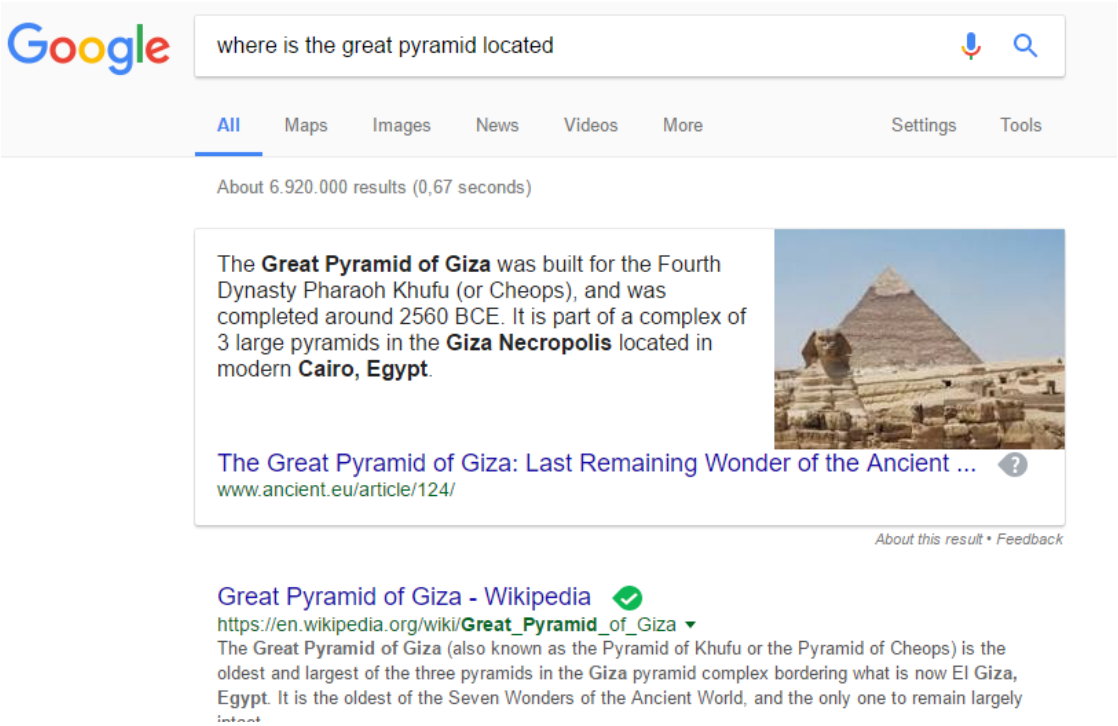


Figure 2.1: Example of Information Retrieval results using Google Engine.

After using morphology term, we should find the right format of the word such as: the word root (define the word tense: present, past and future), prefix (a, an, the, in, il, etc.), suffix (s,ed, st, er, not, etc.) and the stem of word. The stem of a word refers to a part of the word without any extra addition such as the stem of the word "Wait": "Waits", "Waited", "Waiting", "Waiter" or others.

Regarding traditional QA systems, Jurafsky and Martin (2000) proposed a QA system with the best architecture of information retrieval method. As shown in Figure 2.2, the architecture consists of documents and three modules: Question Processing, Passage Retrieval and Answer Processing, respectively. Documents are outside the architecture because these documents are considered as an external dataset. After attaching documents, users can ask some questions about the story of each document. The QA system takes these questions and starts the first module which is called Question Processing. This module consists of two steps: Query Formulation and Answer Type Detection. Query Formulation step is used for getting queries which contain all question keywords. However, Answer Type Detection step is used for extracting the answer type such as: Why, When, Who, Which, How, Where. Then, the QA system enters the second module which is called Passage Retrieval. This module consists of two steps: Document Retrieval and Passage Retrieval. Documents Retrieval step is used for mapping and indexing each document with the corresponding questions given by users. In the end, the QA system uses these indexed documents to retrieve answers back to the relevant documents. On the other hand, Passage Retrieval step is used for retrieving most likely pas-

sages that might lead to answer the question from each document. The top-ranked passages do not necessarily contain the answer to the question, because these documents have not an appropriate unit to rank. Therefore, Jurafsky and Martin have fixed rank scores for passages. After collecting queries from the first module and retrieving passages based on the highest scores from the second module, they fed into the Answer Processing module. In the end, the Answering Processing module is used for extracting the probable answers for each question from the final candidate's passages based on the fixed score-rank.

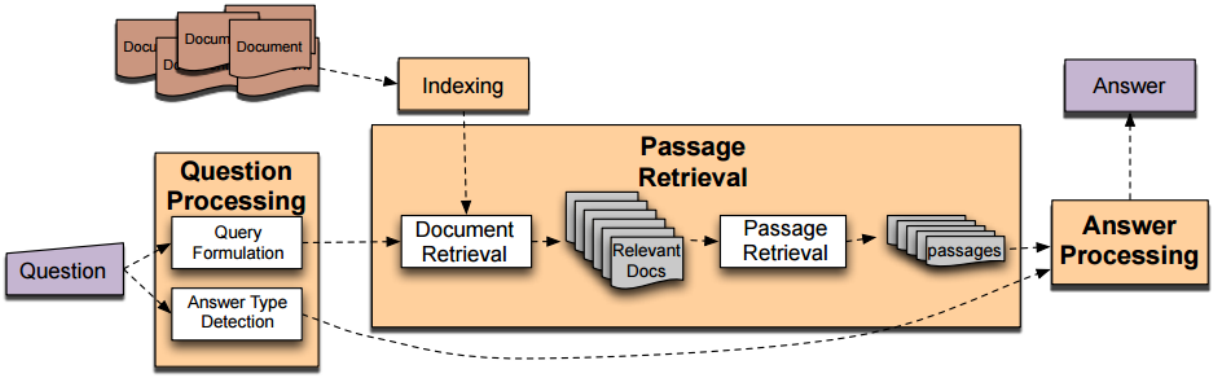


Figure 2.2: Question Answering System based on Information Retrieval (adapted from Jurafsky and Martin (2016).

Most of QA systems are focused on factoid QA system. Facebook bAbI (Weston et al., 2015) is one of the QA datasets often used in factoid QA system. As we mentioned earlier, the factoid QA can be answered with simple facts expressed in a short answer which extracted from the context given. In Facebook bAbI dataset, the answer should be a single word. As shown in Figure 2.3, there are different tasks on Facebook bAbI such as Single Supporting Fact and Two Supporting Facts. In the single supporting fact example, it means that the QA system needs to track only one fact to answer the question. As shown in the single supporting fact example, QA system needs to track the fact "Mary" to retrieve the location of Mary as the answer. However, in Two Supporting facts, QA system needs to track two facts to answer the question. As shown in the two supporting facts example, QA system needs first to track the fact "football" then found "Who picked up the football". Afterwards, QA should track the fact "John" to retrieve the location of John which it is the location of the fact "football". Supporting facts term refers to the position number of the sentence that might lead to answer the question. Thus, fact word refers to sentence and factoid term refers to the sentence id (if and only if, dataset provided the supporting facts). Supporting facts ids might be provided to pay attention in the sentences might lead to the answer.

**Task 1: Single Supporting Fact**
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

**Task 2: Two Supporting Facts**
John is in the playground.
John picked up the football.
Bob went to the kitchen.
Where is the football? A:playground

Figure 2.3: A training example from the Facebook bAbI dataset, consisting of a few sentences, question and the truth answer based on the supporting facts introduced by Weston et al. (2015).

On the other hand, the non-factoid QA system is considered as the narrative form of the QA system that needs more knowledge processing. Answers could be expressed in a long sentence without any supporting facts such as SQuAD dataset, as shown earlier in Figure 1.1.

## 2.2 Current Question Answering

Recently, most of QA systems were implemented based on Neural Networks. In this section, we present briefly the history of the neural systems starting from Biological Neural Networks right up Memory Neural Networks (MemNNs). Moreover, we present which neural networks work well with the sequential textual data. In this thesis, we present a variation of RNN generations such as Long Short-Term Memory (LSTM), Gate Recurrent Unit (GRU) and Memory Neural Network (MemNN).

### 2.2.1 Neural Systems

#### 2.2.1.1 Biological Neural Network

The brain is composed of approximately $10^{11}$ interlinked elements called neurons. They communicate along using electrical signals. As shown in Figure 2.4, neurons can receive, process or transmit the information in one direction. Like a thoroughfare used for transmitting information from point A to point B. Because of having different kinds of information, the brain activates diverse sets of neurons and routes from neuron A to neuron B. Each neuron has a body with branches called dendrites, which receive signals from other neurons through a kind of fiber pipeline called axon. Other neurons receive electrical signals through this axon. By this way, neurons can transmit electrical signals in the human nervous system. Electrochemical devices called synapses are responsible for transmitting electrical signals between neurons. Synapses are located at the meeting point between the axon of ingoing signals (Emitting neuron) and the dendrite of outgoing signals (Receiving neurons). When neurons send signals, it called pre-synaptic neurons and post-synaptic for receiving signals. For transmitting the ingoing electrical signals to other neurons, a gate should be opened on the dendrite of receiving neurons using neurotransmitters. This gate allows energetic ions to flow in or out through

the axon based on the voltage difference (activation level). These ions generate the activation level of the neuron, which is used to decide the potential of the neuron to send or receive these electrical signals. When the activation level of a neuron is larger than its threshold, the neuron become active (on-fire), and then it could propagate signals. By this way, neurons can communicate with each other for sending and receiving information in the brain. For more details can be taken from Chapter 3 in (Floreano and Mattiussi, 2008) book.

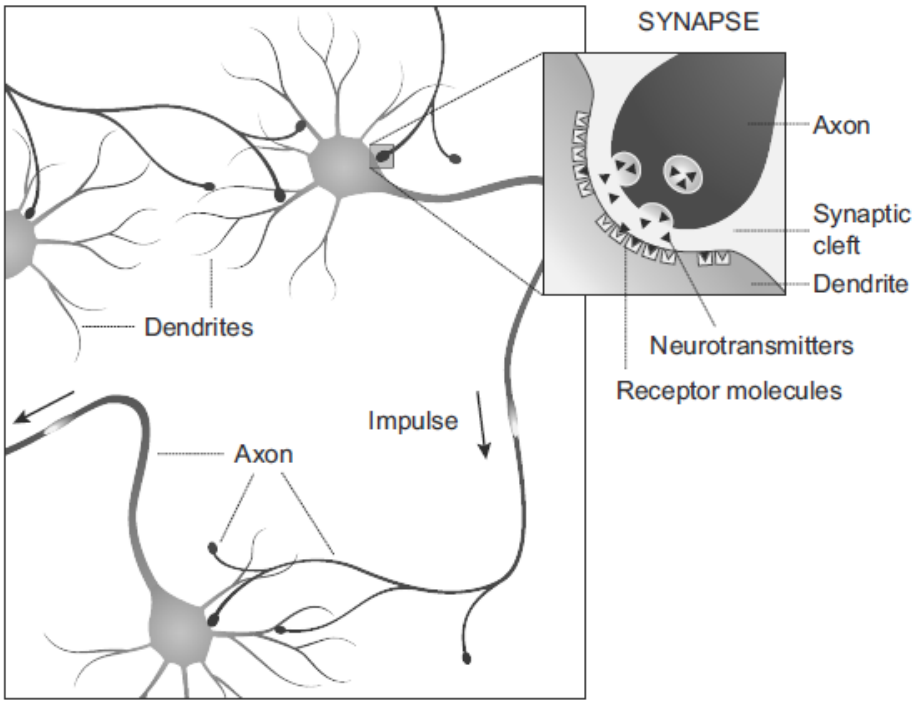


Figure 2.4: Biological Neural Network, credited to Floreano and Mattiussi (2008).

#### 2.2.1.2 Artificial Neural Networks (ANNs)

An Artificial Neuron Networks (ANNs) or sometimes called Neural Networks (NNs) are mathematical models inspired by the biological neural networks computations. They try to simulate the human nervous system for sending, receiving and processing information regarding the computer science field. However, ANNs are a bit different from their biological counterparts. Because, what works best in the brain, does not necessarily have to be best in artificial neural systems. As Floreano and Mattiussi (2008) proposed in his book Chapter 3, ANNs consist of several units called neurons, that communicate with each other. As we mentioned earlier, the neuron becomes active if its activation level is larger than its threshold. The threshold is the simple one in the activation functions family in neural systems. The threshold refers to a specific value that each neuron has to exceed it to be active then could be fired to the next stage. As shown in Figure 2.5, when we

process input neurons $X_n$ along with neurons weights $W_n$ in a neural network, the activation function is calculated by getting the summation of the input and its weight ($X_i$ * $W_i$) for each input unit. Afterwards, we could add the bias value [1] $b$ to the summation result $z$. Then, the new $Z$ fits into the activation function $g(Z)$. If the activation function used is the threshold, we should set the threshold value beforehand. Lastly, we use the result of the activation function to predict the output neuron $Y$. As shown in Figure 2.6, there are many other kinds of activation functions such as Sigmoid, Softmax, Tanh, and recently Rectified Linear Unit (ReLu).

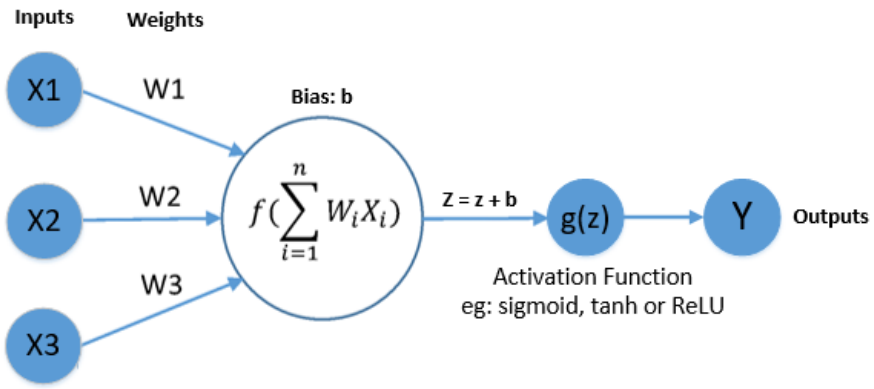


Figure 2.5: Example of the activation function in ANNs: Summation function $z$ sums three input with their weights; then by adding the bias value $b$, we can get the Big $Z$. Then we can calculate the activation function $g(Z)$ to extract the output $Y$.

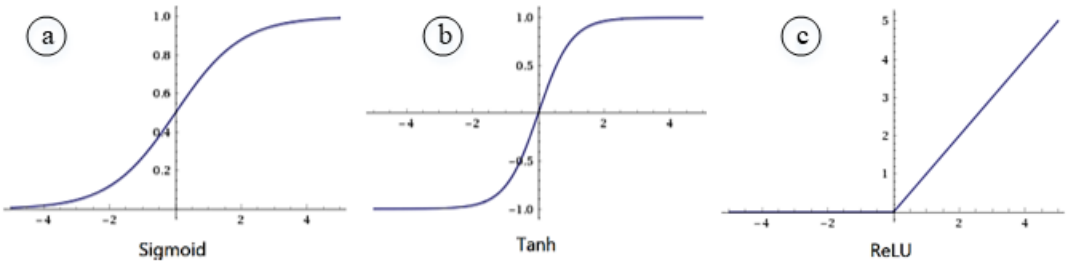


Figure 2.6: Common activation function types: a, sigmoid function that outputs values between (0 to 1); b, hyperbolic tangent or tanh function that outputs values between (-1, 1); c, rectifier or ReLU function that outputs x values in range of max(0, x).

[1] Bias: b is a constant value, it allows the function to move the line up or down to fit the prediction with the data better as defined in a linear function $(y = ax + b)$.

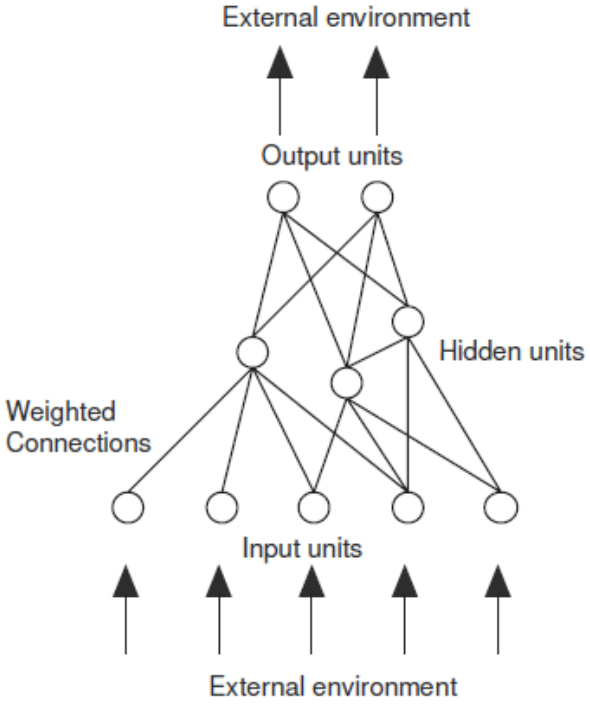


Figure 2.7: Generic neural network architecture, taken from Floreano and Mattiussi (2008).

As shown in Figure 2.7, the simplest Feed-forward ANN such as Multilayer Perceptron (MLP) network is composed of three layers: Input, Hidden and Output respectively. The data is flowing from the input layer to the output layer through the hidden layer in one direction. In the input layer, each unit could communicate with an external environment that provides the neural network with more features related to the inputs. The hidden layer is located in the middle between the input and output layers. Overall, the significant role of the hidden layer is extracting the most important features from the input data and then fits them into the output layer. This hidden layer could be more than one layer based on the problem complexity, and then it called Deep Neural Networks (DNNs) or most of the times called deep learning. So far, It is still hard to figure out how many hidden layers we should have to solve any problem in ANNs. However, researchers found that to solve a linear problem, there is no need to create a hidden layer, and the activation function will be implemented in the input layer. It means that to build ANNs, we have to know the complexity of the problem to decide the number of neurons should be created in each layer (Input, Hidden and Output). Afterwards, we can decide if we should use hidden layers and how many layers should we have? In the end, the output layer is used for generating the output to solve the problem. It predicts the result based on the collected information from the previous layers. After getting the final output, we collect the error cost between the predicted and the target outputs, and then we go through the back-propagation stage. This stage

is about updating the weights in the network and trained the network again until getting a small error.

#### 2.2.1.3 Recurrent Neural Networks (RNNs)

Recurrent Neural Network (RNNs) are ANNs models introduced by Elman (1991) to allow us working with sequences of data. The primary key of RNNs is to store the past information of the previous sequences and passes it to the next sequences. Recently, RNNs used remarkably in NLP tasks such as (language modeling, machine translation), Text mining and Image Captioning tasks. Usually, humans think by sequentially way (step by step), and their thoughts have persistence in the brain memory. It means that they use the past information in their brain memory in any task instead of thinking from scratch. In ANN, it works on all information independently, which it means that there are no relations between the input and the output data. The traditional ANNs show low performance in sequential tasks such in the NLP field. Unlike, RNNs use short loops (memories) to loop over the previous information and store this information for any incoming need to use. By this way, the output is becoming depended on the input data. RNNs architecture consists of three cells: input, recurrent, and output respectively. The input and output cells are working same as in the traditional ANNs. For recurrent cells, each cell extracts the important information from the input cells then passes them to the next cell and so on, so forth. RNNs are called recurrent because it use the same network for each input sequence. As we have mentioned earlier, the main point of using RNNs is that they have recurrent cells (short memories) between the input and the output cells. The recurrent cell is considered as a single hidden layer using (Sigmoid, Tanh, Softmax or ReLU) activation functions. As shown in Figure 2.8, a simple recurrent neural network that consists of input $x_t$, output $o_t$ and a loop $L$ between them. $L$ loop stores information from one recurrent cell and then passes it to the next cell in the next recurrent network. As shown in Figure 2.9, when we unrolled the loop, we found that RNNs look like multiple copies of the traditional ANNs.

These copies are linked as a sequence to sequence or linked-list until reach to the last recurrent cell. The recurrent cell is known as the hidden state of the memory that calculated based on the previous hidden stats. Because of that, RNNs are introduced to work with sequences of data such as speech, text or image sequences data. RNNs work successfully if the gap between sequences is small. For example, to predict the missing word in this text "I was sick. I went to missing-word."; RNNs will use an external knowledge base to predict that the missing word most probably is "Hospital". Because the gap between the relevant information "sick" and the target "missing-word" is small, there are no complex relations between sequences and RNNs do not need more past information. Unlike, if the gap is long, RNNs theoretically can work with long data sequences (long-term dependencies). However, if we use RNN to predict the missing word as we did in the former example, but using a long history like "I was sick. ... (other text sequences) ... I went

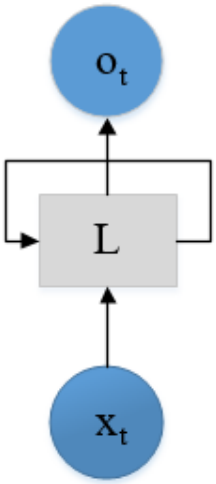


Figure 2.8: A simple Recurrent Neural Network with loops, drawn in the style of Olah (2015) in his article .

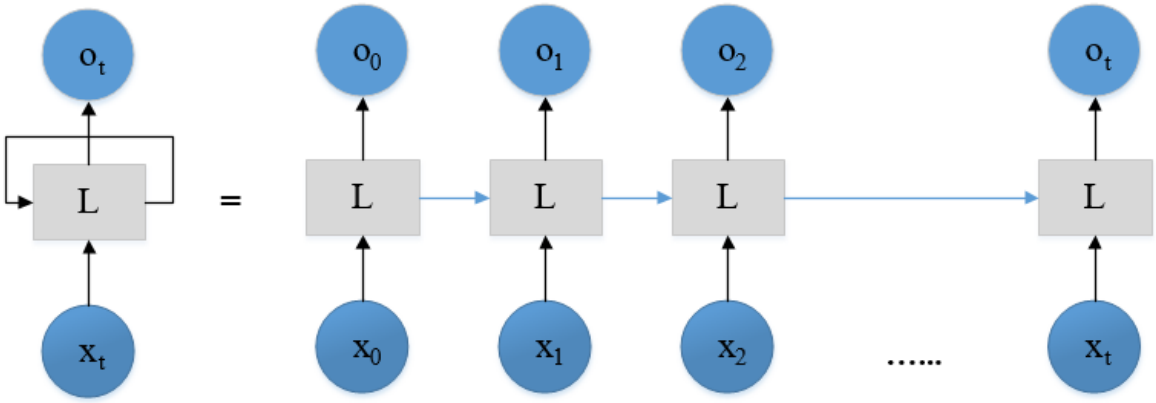


Figure 2.9: An unrolled Recurrent Neural Network, drawn in the style of Olah (2015) in his article.

to missing-word", it will fail to predict the missing word. At the last recurrent cell, it assumes that the missing word is probably the word of place or location based on remembering the previous words. Therefore, to predict this word, RNNs should look further back at all text sequences until finds sick word to predict that the word should be the place "Hospital". Unfortunately, in practice, RNNs show low performance for remembering the past information for long-term dependencies. The reason is that RNNs are suffered from the problem of vanishing gradient descent. In another meaning, RNNs cannot solve the problem of vanishing gradient. Because of that, if the gap between the relevant information and the target to predict is very large, RNNs will show low performance. This problem was introduced

and explored by Bengio et al. (1994). Hochreiter and Schmidhuber (1997) solved this problem by implementing Long Short-Term Memory (LSTM) as an improved version of RNN that able to work with long-term dependencies. LSTM combats the problem of vanishing gradient descent during the back-propagation stage by fixing gradients steep as enough as the network needs. Therefore, LSTM can work with the long-term dependencies.

#### 2.2.1.4 Long Short-Term Memory Networks (LSTMs)

LSTM is addressing the vanishing gradient problem. Thus, it can work with long-term dependencies by using the gating mechanism. In LSTM, retrieving the information of the previous input text sequences for a long time is now possible by learning the parameters for each gate. The major component of LSTM is the gating mechanism to add, update and output the information. The cell gate is a local memory that has all the information filtered by others gates used in LSTM and then passes the information to the next recurrent network. It is also responsible for passing the information without any change between the repeating recurrent networks using linear functions. As shown in Figure 2.10, LSTM has three main gates: forget gate $f_t$, input gate $i_t$ and output gate $o_t$.

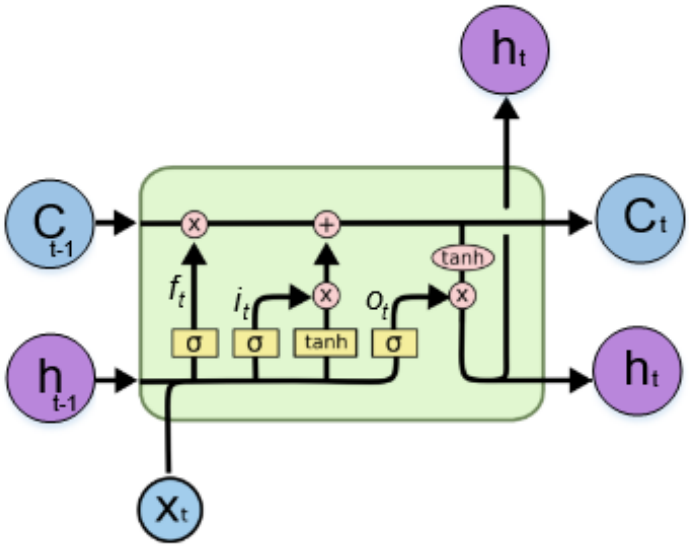


Figure 2.10: A simple Long Short-Term Memory Network, drawn in the style of Olah (2015) in his article.

In the beginning, the first gate is the forget gate $f_t$ which is used to ignore the less important information from the previous hidden state $h_{t-1}$ and keep the important information in the cell state $c_{t-1}$ as shown in Equation 2.1. The forget gate uses a Sigmoid function $\sigma$ that outputs the previous hidden state $h_{t-1}$ and input data $x_t$ between (0 and 1)[2]. As we mentioned earlier, in LSTM, the cell gate (local mem-

[2]The value of 0 means ignore this data (not important) and value of 1 means store this data in the cell gate (most important)

ory) updates its data using linear functions such as multiplication or addition. By calculating the forget gate $f_t$, then we multiply it by the previous cell gate ($f_t * c_{t-1}$), and then we wait for the new information to be added in the cell gate using the input gate. The second gate is used to filter the new information from the input data $x_t$ and add the new important information to the cell state $c_t$, and it is called input gate $i_t$. Using the input data $x_t$ along with the previous hidden state $h_t$, we can calculate the input gate. It uses a Sigmoid function $\sigma$ to output the information between (0 and 1) representing the new important information values, as shown in Equation 2.2. Then, it uses a Tanh function to create the candidate's vector $\tilde{c}$ contains the new important information from the input data that should be added in the cell gate as shown in Equation 2.3.

We can consider this candidates vector as new hidden states should be added in the cell gate. By using a Tanh function, the output of this function will be between (-1 and 1). Then, we can update the cell gate again by multiplying the input gate $i_t$ with the candidates $\tilde{c}$ like ($i_t * \tilde{c}$). The final cell gate $c_t$ can be now calculated by adding the updated cell gate from the forget and input gates as shown in Equation 2.4. The last gate is about extracting only the most important information that should be outputted to the final hidden state in this recurrent network, and then passes to the next network, and it is called the output gate $o_t$. This gate uses a Sigmoid function $\sigma$ to output the information between (0 and 1) representing the most important information should be outputted from the final cell gate $c_t$, as shown in Equation 2.5. Afterwards, it uses a Tanh function to output the important relevant information from the final cell gate $c_t$ between (-1 and 1) values. In the end, we multiply the output of the output gate and Tanh function ($o_t * \tanh(c_t)$) to output the final hidden state $h_t$ shown in Equation 2.6, and then passed to the next recurrent network.

$$f_t = \sigma^3(W_f{}^4 \odot {}^5[h_{t-1}, x_t] + b_f) \tag{2.1}$$

$$i_t = \sigma(W_i \odot [h_{t-1}, x_t] + b_i) \tag{2.2}$$

$$\tilde{c} = \tanh(W_c \odot [h_{t-1}, x_t] + b_c) \tag{2.3}$$

$$c_t = f_t * c_{t-1} + {}^6 i_t * \tilde{c} \tag{2.4}$$

$$o_t = \sigma(W_o \odot [h_{t-1}, x_t] + b_o) \tag{2.5}$$

$$h_t = o_t * \tanh(c_t) \tag{2.6}$$

The final hidden state of the final recurrent network contains the most important information might help to solve tasks such as QA task. Recently, LSTM shows a

[3] $\sigma$ is the sigmoid function that defined as $\sigma(t) = 1 \ / \ (1 + e^t)$)
[4] W is the weight value and it might be different for each gate.
[5] $\odot$ is the element-wise multiplication of vectors.
[6] $+$ is the element-wise addition of vectors.

great performance compared to the vanilla RNN in the long-term dependencies case[7] by solving the problem of vanishing gradient descent.

#### 2.2.1.5 Gate Recurrent Unit Network (GRU)

GRU is introduced by Chung et al. (2014) as another improved version of RNNs with some more differences. GRU is quite similar to LSTM performance and behavior. Unlike LSTM, GRU does not have the cell gate as a memory to store the relevant information. Moreover, GRU does not have the output gate as in LSTM, because it does not need another nonlinearity function to output the final state. As shown in Figure 2.11, GRU has two gates only: update gate and reset gate. The update gate $u_t$ keeps only the important information from the previous hidden state (recurrent network) or adds the new relevant information from the input data. GRU used the update gate for adding and updating information to the hidden state, instead of using separate gates such as forget and input gate in LSTM. On the other hand, reset gate $r_t$ is used for combining the new information from the input data $x_t$ to the previous hidden state, and then extracting the final hidden state of this network. In GRU, the cell state and the hidden state are merged into one state called hidden state $h_{t-1}$.

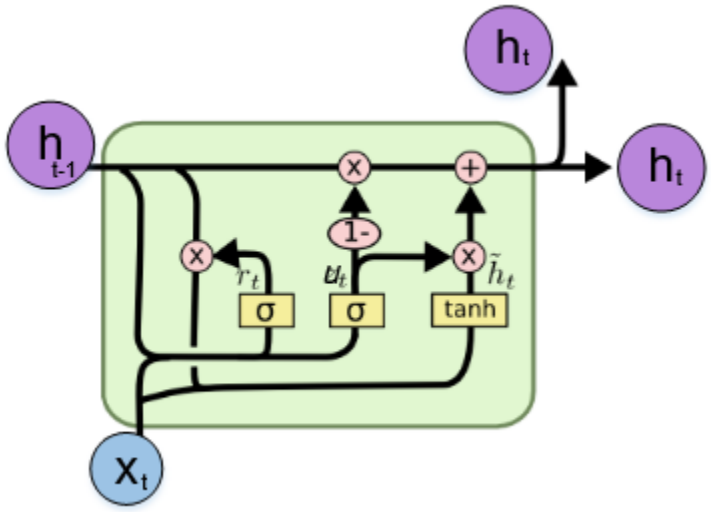


Figure 2.11: A simple Gate Recurrent Unit Network, drawn in the style of Olah (2015) in his article.

Firstly, the reset gate uses a Sigmoid function to get all values between (0, 1) using the input data $x_t$ along with the previous hidden state $h_{t-1}$ as shown in Equation

[7] If we fixed the input gate all to 1's, the forget gate all to 0's (you always ignore the previous information) and the output gate to all to 1's (you output the whole information). Therefore, we will get back to the standard RNN architecture. It means that we will add all information and output all possible information.

2.7. Following by the update gate, it uses a Sigmoid function to output values between (0, 1) to update the input data with the value one as shown in Equation 2.8. Afterwards, we use a Tanh function after multiply the reset gate and the previous hidden state like ($h_{t-1}$*$r_t$) to output the new candidates of the hidden state $\tilde{h}$ as shown in Equation 2.9. The last step is to calculate the final hidden state that will be passed to the next recurrent network. First, we multiply the information we decided to ignore it like (1 - $u_t$) by the previous hidden state $h_{t-1}$. At the same time, we also multiply the update gate $u_t$ and the final candidates of the hidden state $\tilde{h}$ to be sure that we extracted the most important information. In the end, we add both hidden state calculations to get the final hidden state as shown in Equation 2.10. In many tasks, both LSTM and GRU yield the same performance and behavior. GRU does not need long-sequences data to generalize the model. On the other hand, for a long-sequences data, LSTM might lead to better performance and results.

$$r_t = \sigma(W_e \odot [h_{t-1}, x_t]) \tag{2.7}$$

$$u_t = \sigma(W_u \odot [h_{t-1}, x_t]) \tag{2.8}$$

$$\tilde{h} = \tanh(W \odot [r_t \ * \ h_{t-1}, x_t]) \tag{2.9}$$

$$h_t = (1 - u_t) * h_{t-1} + u_t * \ \tilde{h} \tag{2.10}$$

#### 2.2.1.6 Memory Neural Networks (MemNNs)

Memory networks (MemNNs) are a newly proposed variant of RNNs with a global memory potential. MemNN was introduced by Weston et al. (2014) to have an external global memory. In LSTM, the cell gate is considered as local memory, and it addressed the shortcomings of RNN concerning vanishing gradient problem. However, with long input sequences, a bit long and compartmentalized memory became necessary. In order to solve this issue, we can add multiple LSTM and each cell gate as an independent memory in each LSTM. This solution is much computations expensive and shows low performance. Recently, MemNN allows us to read and write to its global memory for all text sequences. Moreover, we can increase the memory size. MemNN consists of a memory $m$[8] with many hops $k$ and four components. Each memory hop $k_t$ is considered as a dense vector.

MemNN has four main components: input, update (generalization), output and response respectively. In the input component, the raw input sequence (In our thesis, the input is text: sentences and words) is being encoded to convert the text to spare vector representation using the Bag[9]-of-words (BoW) model. BoW model is used for extracting information from the input text sequences based on the occurrence of words in each sentence. The update component in the MemNN looks like input gate in the LSTM that it is used for adding new input information given, and then

[8]Memory contains all the relevant information vector as an array of strings indexed based on the memory hop index $k_t$.

[9]It is called a "bag" of words, because this model ignores the ordering or structure of words, and this is the limitation of this model.

updating old memories from the previous component[10]. The update component called generalization step, because it could store and generalize its memories per-hop for many reasons such as reasoning over multiple sentences more than once to get the probable response (answer for the question in our case). Following by the output component, it is used for reading the memory hops $m_k$ that contain all the relevant information and outputting the new output based on a score function. The new output is produced based on the new input information from the input component with the updated memory per-hop $m_k$ from the update component. The score function determines the degree of matching the input information and the updated memory information. The last component is the response component that used for producing the final response based on the new output. We talk about a QA system in this thesis, however, in this case, the response component could use RNN or one of its classes (LSTM or GRU) to extract the answer to the question as multiple words. Last but not least, MemNN shows great performance in QA task. Recently, many models were implemented concerning the domain of the QA task based on MemNN architecture. In Chapter 4, we present the Dynamic memory network (DMN) which is another extended version of MemNN.

[10]If we have much data and the memory became full, you can replace or forget one of the memory hop $m_k$ like the forget gate in LSTM does.

# Chapter 3

# Related Work

Before DMN, Recurrent Neural Network (RNN) classes such as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) has been used for language systems such QA system. LSTM contains an external local memory cell to allow us to store the important information. Afterwards, Chung et al. (2014) was introduced Gate Recurrent Unit (GRU) as another improved version of RNN using the gate mechanism. GRU does the same behavior as LSTM with less one gate and fewer computations expensive (less memory state). Further development in this area led to Memory Network (MemNN) (Weston et al., 2014) which it is the recent class of RNN family but contains a global memory block. MemNN in the recent neural network experiments has achieved high performance in QA applications using Facebook bAbI dataset (Weston et al., 2015). However, in MemNN, the input module encodes sentences independently not in a sequence, and thus it shows low performance in other NLP tasks. Following the release of SQuAD (Rajpurkar et al., 2016), it is now possible to train end-to-end neural network models with large-scale QA dataset. We categorize these models into two broad groups based on how to produce the final answers. The QA models for encoding the question and context, and then extracting the useful information to answer the given question could be divided into two categories: single-step and multi-step models. In the single-step models, the merging of the question and context occur only once before generating the final answer. The single-step models mainly focus on the attention mechanism technique. Unlike the single-step models, the multi-step models are closer to how humans think by reading the context more than once to extract and retrieve the answer. This category mainly focuses on the memory mechanism technique along with the attention mechanism technique.

## 3.1 Single-Step Models

The most popular single-step models are mainly based on attention mechanism techniques. Wang and Jiang (2016) were implemented Match-LSTM with Answer Pointer model concerning the domain of QA task. Their model used their old Match-LSTM model (Wang and Jiang, 2015) to match the context and question

words by using the attention mechanism. They used a soft attention mechanism to pay attention on question keywords against the context. Then, they used another attention mechanism type called Pointer Networks (Vinyals et al., 2015) to extract all possible tokens from the context as the answer. Furthermore, Xiong et al. (2016b) have presented the Dynamic Co-attention Network (DCN) model. DCN consists of two encoders (Context, Question) and dynamic pointing decoder model. They used co-attention encoders for paying attention to the question against the context and the context against the question. Instead of using a simple calculation (such as summing or averaging) to combine individual attention into final attention. After they retrieved all question-context interactions, there might be several probable answer tokens in the context. Therefore, they used a dynamic pointing to generate the answer span by getting the prediction of ($A_s$, $A_e$) tokens. This technique is used to recover the maximum answer spans corresponding to incorrect answers. After each iteration, the dynamic pointing decoder updates its state based on the results of the co-attention encoder and the prediction of answer span and then produces a new prediction span and so on, so forth. Following, Cui et al. (2016) proposed the Attention-over-Attention Neural Networks (AoANN) model to match between question and context. They used AoANN for paying attention over the question against the context, and then they did another attention over the first attention to extract the most important information of each attention.

Later, Seo et al. (2016) proposed Bi-Directional Attention Flow (BIDAF) model to implement a question-ware and context representations. They proposed an updated attention mechanism that their attention layer did not summarize the question and context information into a fixed-vector as what Xiong et al. (2016b) did in their DCN model. Alternatively, they let the attention vector to go through the modeling layers. Then, in the end, they summarized all these attention layers, and thus they reduced any information loss. Recently, Kumar et al. (2016) proposed Dynamic Memory Network (DMN) model as an extended version of MemNN concerning the domain of QA. In their DMN model, they modified the attention mechanism to automatically decide which information in the input is most important to keep safe and which should ignore against the question. Kumar et al. calculated the gate attention scores then modified the GRU network by replacing the gate attention with the update gate in the GRU. They used their modified attention mechanism to extract the contextual vector, and then they updated the memory using vector. Then, they used the final memory state to generate the answers. Finally, again Wang et al. (2017) updated their model and proposed Gated Self-Matching model for QA task. They used gated attention based on GRU to obtain the degree of matching the question and context. Then, they used another attention mechanism called self-matching attention to matching the context again against itself to improve the contextual vector of the first attention. In the end, they used the pointer networks again to find all possible tokens positions in the context which they might be the answer or part of it. Then, they extracted the answer-span based on these positions.

## 3.2 Multi-Step Models

For the multi-step model, Weston et al. (2014) proposed MemNN, which they used a long-term memory component to store information and an inference component for reasoning over context. They used many memory-hops to allow reasoning over the context more than once to extract the probable answers. Later, Kumar et al. (2016) improved over MemNN by introducing an end-to-end trainable network called Dynamic Memory Networks (DMN). DMN has four modules: input, question, episodic memory and answer. In the episodic memory module, they used the memory mechanism after implementing the attention mechanism to update the memory per-hop with the most important information extracted from the attention mechanism. Then, they combined all memory states for each episode to generate the final memory state. Then, they used the final memory to extract the answer. By using memory mechanism, they allowed their QA system to read the context more than once to find the probable answer to the given question. Recently, Pan et al. (2017) proposed the Multi-layer Embedding with Memory Networks (MEMNN) based on MemNN. Instead of using single embedding layer for context and question, they used multi-layer of word embedding to extract the word-level and char-level embedding. Then, they almost used the same architecture of MemNN. However, they used the pointer networks to find all possible tokens potions in the context, and then generate the answer span.

Throughout this thesis, we examine and re-implement the Match-LSTM with Answer Pointer model which is introduced by Wang and Jiang (2016). This model consists of the encoder that encodes the context (input text sequences) and question and then merges both using Match-LSTM layer. On the other hand, the decoder extracts the optimal answers between the boundary answer span ($A_s$, $A_e$) tokens using Pointer Networks. Then, the output of these pointer networks is used to extract the probable answers after getting the hidden relationships between the context and question. Finally, in the following chapters, we propose the former model along with Kumar et al. (2016) DMN model and our improved DMN model.

# Chapter 4

# Approaches

In this chapter, we propose the four models which are implemented throughout the thesis. All implementations are trained and evaluated on SQuAD dataset that it will be described in detail in Chapter 5. We present these models in details as follows: baseline, Wang and Jiang (2015) Match-LSTM with Answer pointer, Kumar et al. (2016) DMN and our improved DMN. DMN approach was not evaluated on the SQuAD leaderboard so far, and therefore it would be the first try using the DMN architecture on SQuAD dataset. First of all, we describe the technique of word embedding methods which is about using word vector representations for processing any textual data. In this thesis, we used Glove (Pennington et al., 2014) as the word embedding method in all implementations with different dimension size. Last but not least, we describe the attention mechanism technique which is about determining the best locations of data to pay more attention and doing further analysis to solve the language tasks such as Machine Translation and Question Answering. In the QA task, these locations contain the most important information might use to answer questions by finding locations of the highest words probability related to the question.

## 4.1 Word Embedding

Word embedding has become the main first step towards implementing any NLP and text mining tasks. Word embedding technique encodes each word in the input text sequences into a unique integer. Then, it trains words against their neighbor words in the same sentence to get the relations between words in the vector. Word2vec is a word embedding technique and refers to word to vector representations. Word2vec is a small unsupervised neural network with two layers (Input and Output), but it is not a deep network. The input layer contains sequences of words, and the output layer produces a set of vectors space. As is known for deep neural networks, they can understand only a numerical format. Therefore, by using word embedding method such as Word2vec, we can map each word in a text to a numerical vector in a continuous space. Before, the one-hot encoding method is used for mapping words into a boolean vector with high dimension vector. Each

element in the one-hot vector maps to a unique word in the corresponding vocabulary. It means that the dimension of the one-hot vector equals the vocabulary size. Because of that, the dimension of the word embedding vector is lower than the dimension of the one-hot vector. The target of the word embedding technique is to collect vectors of similar words together in one low dimension vector space such as (give, live). The more large data we have, the more accurate guesses about the word meaning we get. The introduction of the word embedding technique is credited to Bengio et al. (2003). They have proposed a language model based on a probabilistic model over words and is divided into two parts. First, they mapped each word into a numerical vector by selecting the corresponding token in the vocabulary file. Second, they kept all vectors of words into one vector and then used a probabilistic model to get the final matrix of all words. Recently, most of word embedding techniques can use one of two architectures to produce representations of words as shown in Figure 4.1. Either using surrounding words representations to predict a target word and it is called Continuous Bag of Words (CBOW) Mikolov et al. (2013a) or using one word representations to predict the target surrounding words and it called Skip-gram Mikolov et al. (2013a). Mikolov et al. (2013b) again improved the performance of Bengio et al. (2003) model by proposed a model to map words into a high dimensional vector based on the Skip-gram model (surrounding words in the input texts). Recently, Skip-gram architecture became the core model of most word embedding techniques.

Mikolov et al. (2013b) model extracted the semantic and syntactic of words and then generated relationships between words. These relationships can be understood from the famous example of Word2vec: "word2vec(king) - word2vec(man) + word2vec(woman)" which yields the embedding vector of word2vec(queen). In this thesis, we trained the input text sequences using the Glove method which is introduced by Pennington et al. (2014) as a pre-trained word embedding technique. Glove also is a small unsupervised neural network with two layers (Input and Output). The input layer contains input text sequences, however the output layer processes these sequences word by word to produce the vector representations. In Glove version 1.2, we have four different pre-trained word vectors, and then we decided to work with the Glove package with the 6B vector that it contains (Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 300d vectors, 822 MB download)). Glove term stands for global vectors because these global vectors of words are extracted directly based on words co-occurrence in the data. Thus, we do not need to improve or learn these vectors again. Regarding any QA system, the raw data is passed as an input text sequences (sentence by sentence and word by word). After using the Glove method, we create the global vectors of words as embedding lookup-tables, and all words values often are in 0 values. Afterwards, these embedding vectors are fed into the network as input layer units.

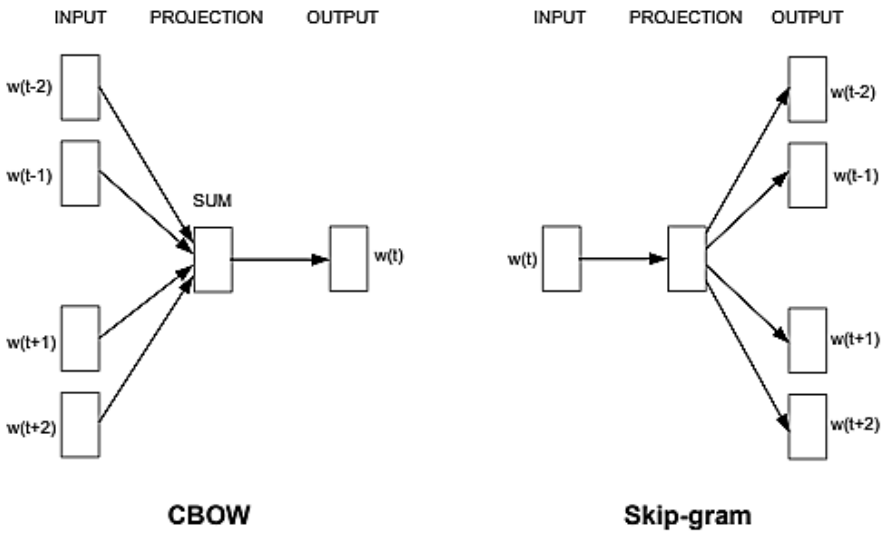


Figure 4.1: Continuous Bag-of-Word (CBOW) and Continuous Skip-gram, taken from Mikolov et al. (2013a).

## 4.2 Attention Mechanism

Attention mechanism is a technique to produce a vector representation about all information in the input text sequences. This vector is often a dense vector using Softmax function. Bahdanau et al. (2014) have proposed an attention mechanism for a machine translation task to improve the performance of its Encoder-Decoder architecture. Before attention mechanism, in machine translation task, we were reading the input text sequences and understanding the meaning of its words, and then encoded the information of these sequences into a fixed-length vector. Input text might be one sentence or more, and each sentence might contain several words. Thus, compressing all information into one fixed-length vector leads to losing information and low performance. Earlier, RNN had problems dealing with such long-term dependencies because of the vanishing gradient problem. Then, LSTM and GRU solved these problems using the gating mechanism potentials. However, applying LSTM or GRU in machine translation task with long sentences is still showing low performance without using attention mechanism techniques. By using encoder-decoder architecture, the encoder encodes the raw input text and then compresses all information into a fixed-length vector (the final hidden state). Afterwards, the decoder decodes the fixed-length vector by reversing the input sentence by sentence and word by word using the vocabulary file. Then, it produces the translation output words. This reversing process might work well with the language in the same structure (fixed word order) such as the translation from English to German or English to French or vice versa. Fixed word order term means that the position of the first word in English is most probably the same position of the first word in German.

For example:

- English: I am Ibrahim
- German: Ich bin Ibrahim

However, if we try to translate from English to Japanese, it definitively does not work well. The reason is that during the reversing process in the decoder, the position of the first word of English could be the position of the last word in Japanese.

Briefly, the issue with this encoder-decoder architecture without using attention mechanism is that after encoding the input text sequecnes, the network compresses all the relevant information (sentences) into a fixed-length vector. Bahdanau et al. (2014) solved this problem by allowing the decoder to pay attention over any part in the input text sequences. That means, they share all the final hidden states for each word in the input text rather than compress all information in a fixed-length vector (the final hidden state). As shown in Figure 4.2, they calculated the alignment method $a_t$ for each word $X_T$ in the input text by getting the correlation value. This value expresses how each hidden state of each word in the input text correlates with each word in the target text. Afterwards, they normalized these values often by means of a Softmax function to produce the probability scores, and sometimes they called attention weights. Lastly, they produced the contextual vector by getting the summation of normalized scores and the final hidden state for each word in the input text. In Bahdanau et al. attention mechanism, they allowed the decoder to output words based on the combination between all input words states (attention weights) as a contextual vector, not just the final hidden sate. Recently, attention has two different mechanisms: Soft attention and Hard attention. Bahdanau's attention is a soft attention mechanism. Soft attention is a differentiable mechanism and it often uses Softmax [1] to generate the attention weights (probability scores) over the input text sequences. In contrast to that, hard attention is a non-differentiable mechanism such as a binary way (0 or 1) or (ON or OFF) not in range like the soft attention technique. By this switch way, hard attention can determine which information to pay attention to it and ignore the rest. In the end, Kumar et al. (2016) have introduced a new attention mechanism based on the GRU gates, and they called modified AttGRU. In Section 4.4.3, we present this attention mechanism in details.

[1] Softmax function is a differentiable mechanism too, that it produces values between (0, 1).

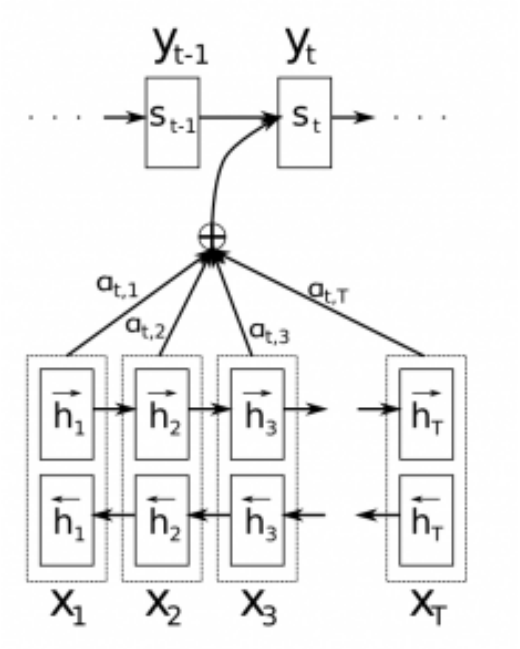


Figure 4.2: Bahdanau's Attention Mechanism (Bahdanau et al., 2014).

## 4.3 Encoder-Decoder Architecture

The encoder-decoder architecture for RNNs is the most standard and effective mapping models (Cho et al., 2014). It processes (encodes) a sequence as an input and generates (decodes) another sequence as an output. As shown in Figure 4.3, this is the generic architecture used in all implementations throughout this thesis. As we mentioned, we use SQuAD dataset as our evaluation dataset. It contains several paragraphs, and each paragraph consists of (Context, Question, Answer) triplets. We fed this raw input data into the word embedding technique such as Glove6B with 100 or 300 dimensions, and then we branch the embedding vector out into input and question word vector representations. Afterwards, we fed both word vector representations into the encoder. The encoder encodes the context and question vector representations (word by word) separately, and then it extracts the contextual vector. This vector contains all relevant words might lead to the answer. Finally, we fed this contextual vector into the decoder to decode the output and extract the final answers as a segment of text (span: $a_s$, $a_e$). Where, $a_s$ refers to the position of the start answer token and $a_e$ refers to the position of the end answer token. All the words between these two tokens in the original context are considered to be the answer.

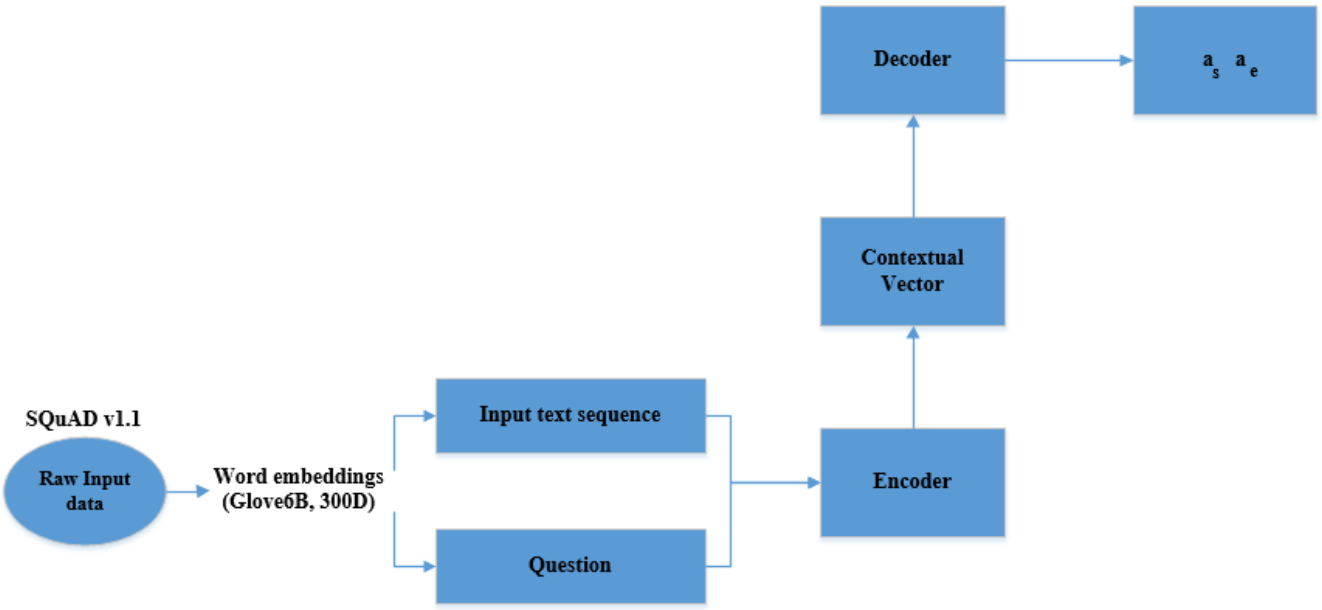


Figure 4.3: Our Generic encoder-decoder architecture for seq2seq model.

## 4.4 Approaches

### 4.4.1 Baseline Model

As a baseline model, we implement a simple seq2seq (vanilla) LSTM model with a simple attention mechanism, before going deeper into more complex models. According to Figure 4.3, we process the context and questions into two embedding layers (Glove6B for word-collections), then we pass the outputs of the embedding layers as inputs into the encoder that consists of two LSTM layers for each. Then we merge the two LSTM into the Bahdanau attention mechanism to generate the contextual vector (knowledge representation). In this model, we implement Bahdanau's attention on the question for each word against context to allow each word in the context has a hidden representation reflecting its relation to the question words. Afterwards, the decoder decodes the contextual vector using another LSTM to extract the output sequences. Finally, we generate the most probable answer (span: $a_s$, $a_e$) from the final output sequences.

### 4.4.2 Match LSTM with Answer Pointer Model

Wang and Jiang (2016) have proposed this model based on their Match-LSTM model used for predicting textual entailment (Wang and Jiang, 2015). Briefly, as shown in Figure 4.4, they used two LSTM to process the context or Passage $P$ and question $Q$. Then, they used a soft attention mechanism to calculate the alignment method of the question. As we described in Section 4.2, the alignment method extracts weight vector representations as attention weights. These vector representations contain the degree of matching information in each word in the context against the question words. After calculating the attention weights of the question, they combined it with the final hidden state of each word in the context to extract the contextual vector. Then, they fed this vector into LSTM to form their Match-LSTM layer.

Wang and Jiang also used the Ptr-Net model introduced by Vinyals et al. (2015) to generate the output from the matched input text sequences (context) using the vocabulary file. Ptr-Net used another attention mechanism as a pointer to select words positions from the input sequence. Then, Ptr-Net chooses the most probable answer tokens in the context after getting their hidden representations reflecting their relation to the question words. In the answer pointer layer, there are two answer models could be outputted. The first model is the sequence model, which it generates the final answer as a sequence of words from the whole sentence contains the matched answer. On the other hand, the second model is about producing the answer between two tokens (span: ($\text{answer}_{start}$, $\text{answer}_{end}$)), it called the boundary model. All the words between these two tokens in the original context are considered to be the answer. In practice, the advantage of this span prediction is that we can predict a very long answer spans without getting irrelevant information, in contrast to the sequence model. Last but not least, the boundary model is the model we examined and implemented in this thesis to be as a comparative model to our improved DMN. As shown in Figure 4.4, Wang et al. have proposed a model that consists of three layers:

1. LSTM preprocessing layer preprocesses the context and the question using different LSTM.

2. Match-LSTM layer tries to match each word in the context against the question words based on the contextual vector extracted by the attention mechanism.

3. Answer pointer layer uses the Pointer Network (Ptr-Net) mechanism to extract the answer by pointing the candidate words positions from the context. Then, It generates the answer in two ways: sequence or boundary model.

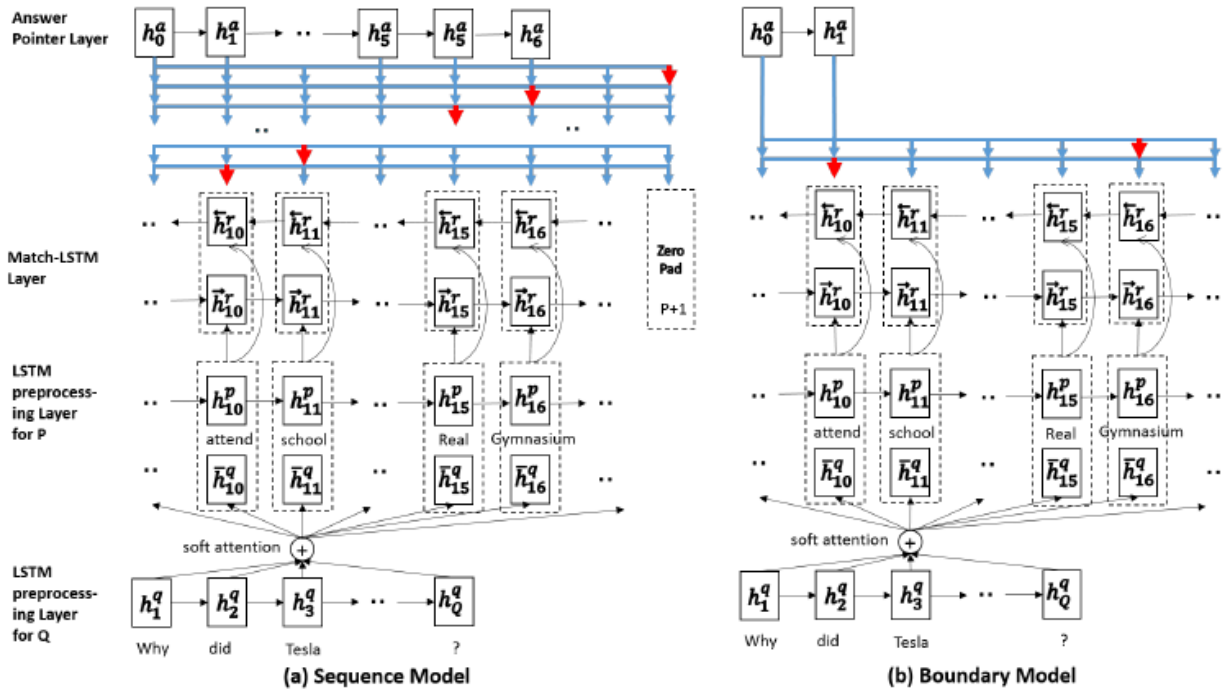


Figure 4.4: Match-LSTM with Answer Pointer model is credited to Wang and Jiang (2016). The final answer is selected either as: a, sequence (word by word) or b, the answer is selected as a segment of text called boundary of (answer$_{start}$, answer$_{end}$).

### 4.4.3 Kumar et al. Dynamic Memory Network model

The limitations of MemNN architecture motivated Kumar et al. (2016) to propose another extend version of MemNN with a new structure called Dynamic Memory Network (DMN). The shortcomings of the Bag-of-Words model which it ignores the ordering of words and losses the semantics of words are some of these limitations. In addition to that, computing the score for each memory is expensive, and when we have a huge memory, it will be a significant limitation. Kumar et al. DMN model addressed these limitations. DMN architecture consists of four modules: input, question, episodic memory and answer. As shown in Figure 4.5, the raw input data trained by one of pre-trained word embedding methods or using randomly high dimensional numbers in the range between (-1, 1). Afterwards, these pre-trained word vectors fed into the input and question modules. Then, the input and question vectors are encoded separately using one of the RNN classes to produce the final hidden representations for each of them. These representations contain all information extracted from the input and question modules. The episodic memory module consists of two mechanisms: Attention and Memory. This module uses the final representations outputted from the input and question modules through its mechanisms to extract the most important information might lead to answer the question. Finally, they used this information used by the answer module along with

the question vector representations to generate answers.

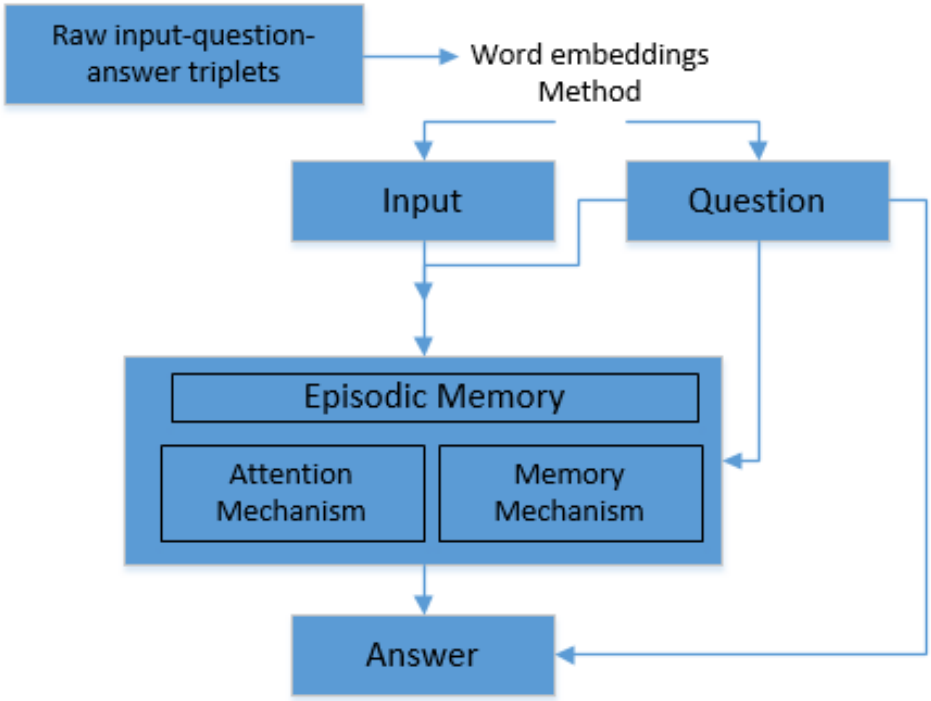


Figure 4.5: Generic Dynamic Memory Networks architecture introduced by Kumar et al. (2016).

**Input Module** The input module encodes input text sequences (sentence by sentence and word by word) using one of the RNN classes. This module encodes the sentence $S_I$ (word by word $w_1$,....,$w_I$) until reaching the end-of-sentence (EOS) token and then extracts the final hidden state at each word. If the input text sequences contain several sentences, the input module retrieves the final hidden representations at each of the EOS tokens. In the end, the final hidden representations fed into the episodic memory module and they called them fact representations $S_f$.

**Question Module** The question module encodes the question sentence $Q_I$ (word by word $w_1$,....,$w_I$) using one of the RNN classes. The question often is a single sentence. Then, it extracts the final hidden state $Q_t$ at each word. These final hidden representations along with the fact representations of the input module are fed into the episodic memory module. Additionally, the question module shares their final representations with the episodic memory module as well as the answer module.

**Episodic Memory Module** The episodic memory module consists of two components: the attention mechanism and memory mechanism as shown in Figure 4.5.

The role of this module is to collect the relevant information from the fact representations which might lead to answer the question given. For each episode, Kumar et al. used the attention mechanism to iterate over the fact representations $S_f$ to find the important information to focus on leading to the answer. They extract the contextual vector that contains all the relevant information about the context against the question. Then, the memory mechanism updates the memory using the contextual vector per episode and the previous memory state. They used the question representations to initiate the previous memory state at the first memory episode. After updating the memory at the last episode, they generate the final memory state which it contains the most important information to answer the question given. In the end, they forward the final memory state to the answer module to extract the probable answer.

**Answer Module** The answer module produces the final probable answers based on the final memory state of the episodic memory module along with the final hidden representations of the question module. For expecting a short answer (Single-word), we could use a simple function such as Softmax. On the other hand, for a long answer (Multiple-word), we could use one of the RNN classes such as LSTM or GRU.

As shown in Figure 4.6, this is an example of Kumar et al. DMN model using input sentences and a single sentence as a question from Facebook bAbI tasks (Weston et al., 2015). As we mentioned before, DMN architecture consists of four main modules, as well as the new module for the semantic memory. The semantic memory module contains the word embedding method such as Glove (Pennington et al., 2014) as pre-trained word vector representations. DMN works as follows: the input module, question module, episodic memory module and answer module respectively.

In this following paragraphs, we explain the functionality of each module according to Kumar et al. DMN implementation.

#### 4.4.3.1 Input Module

The input often is a list of sentences, and each sentence is a list of words. Kumar et al. used Glove vectors as word embedding method to get the word vector representations of the raw input sentences. As we mentioned in Chapter 2, the best way to encode the sequential input is via one of the RNN classes. Kumar et al. decided to use GRU in their DMN experiment. They used a single GRU to encode the input text sequences. The final hidden representations of the GRU is defined as in Equation 4.1:

$$h_t = GRU(x_t, h_{t-1}) \tag{4.1}$$

Where the $x_t$ is the input word at each time step $t$ and the $h_{t-1}$ is the previously hidden state that the GRU updates it at each time step until reaching to EOS

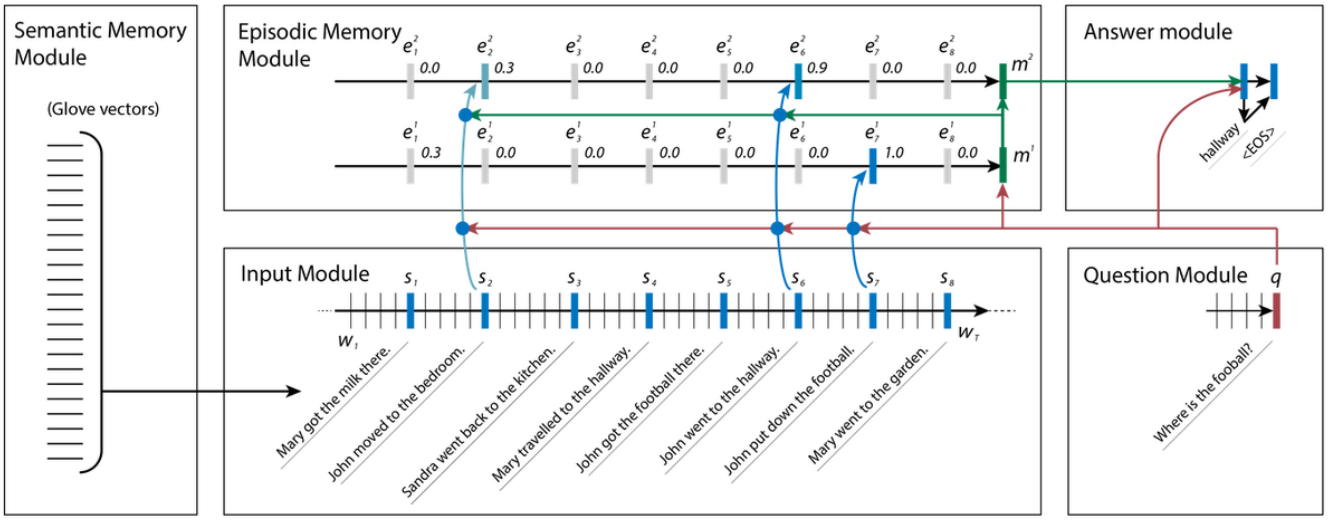


Figure 4.6: Real example of a list of sentences as the input and single sentence as the question from the Facebook bAbI tasks using Kumar et al. (2016) DMN.

token. As shown in Figure 4.6, the example shows that there are several sentences $S_t$ where $t = 8$. Therefore, The final hidden representations of GRU should be eight vector representations. Then, they fed the final hidden representations into the episodic memory module as the fact representations $S_f$.

#### 4.4.3.2 Question Module

Kumar et al. decided to use another GRU for the question module. Similar to what they did in the input module, they encoded the question sentence (word by word), and then they generated the final hidden representations. The final hidden representations $Q_t$ at each time step $t$ is defined as in Equation 4.2:

$$Q_t = GRU(x_t, Q_{t-1}) \tag{4.2}$$

Where the $x_t$ is the question word at each time step $t$ and the $Q_{t-1}$ is the previous hidden state for the question at each time step $t$. Last but not least, they fed the final question representations $Q_t$ along with the fact representation from the input module into the episodic memory module. They initiated the previous memory at the first memory episode by $Q_t$. In addition to, they fed the question representations into the answer module to extract the probable answers to this question.

#### 4.4.3.3 Episodic Memory Module

In this module, Kumar et al. tried to collect the relevant information from the fact representations along with the question representations and the previous memory state to answer the question given. By iterating over the fact representations, in each iteration which is called episode, they retrieved the relevant information that might lead to answer the question and stored it in the memory. They kept this information safe even if it does not answer the question in the first iteration; because it could help them to find more information might lead to the probable answer in the next iterations. Afterwards, they updated the memory by this information and then they produced the final memory state at the end of the last episode. This module mainly composed of two GRUs. The outer GRU is used to update the memory with the most important information and then generates the final memory state over a sequence of episodes. The inner GRU is used to form the contextual vector per episode $e^i$. This contextual vector contains the relevant information of the context against the question. In each iteration, they determined whether each of the fact representations $S_f$ is important [2] to lead to the answer using attention mechanism. In the inner GRU, they used a gate mechanism as their new attention mechanism technique. Then, they modified the update gate in the inner GRU with this gating mechanism. Their modified attention GRU (AttGRU) produced the contextual vector $e^i$ per episode. Finally, the outer GRU used this contextual vector along with the previous memory $m^{i-1}$ to update the current memory, and this is the process of the memory mechanism.

For a long story in the language tasks such as QA system, it shows high performance when we use multiple memory episodes (hops) to iterate over the input multiple time. The idea behind using multiple memory episodes is that sometimes the one-hop does not retrieve all the relevant information needed to answer the question given. As we mentioned in Section 1.1, humans can not store everything in their brain memory in a few moments. Because of that, they might need as many glances as possible to answer the question. These glances are the multiple memory episodes. During this process, humans used some keywords in the question to direct their attention to retrieve the relevant information, and this is the process of attention mechanism. Moreover, by using multiple episodes, we can also address transitive inference[3], and sentiment analysis[4] tasks. As shown in Figure 4.6, to answer the question, Kumar et al. needed two episodes to get the sentence that might lead to the answer. They retrieved the candidate sentence with the highest probability score, which generated by the attention mechanism. The question that was asked in the real example in Figure 4.6 is "Where is the football?". At the

[2]Important here means if the information in each fact (sentence) representations is relevant to the answer.

[3]Transitive inference is the ability to extract facts from premises, e.g. “Ibrahim is taller than Benjamin” and “Daniel is shorter than Benjamin”, then the fact is, “Ibrahim is taller than Daniel”.

[4]Sentiment Analysis is the ability to determine whether a piece of writing is positive, negative or neutral (e.g. to get the reader’s opinion).

first memory episode, the highest probability score is the seventh sentence "John put down the football" with (1.0) score. However, the question was asking about a location, not a person using (interrogative word: Where). Therefore, by using another memory episode, it let the attention mechanism iterated over the input once again using the information of the first episode. The candidate sentence of the first episode was "John put down the football"; thus, at the second memory episode, it should find the location where John was. After paying attention to sentences, the sixth sentence "John went to the hallway" has the highest score with (0.9). Finally, the memory mechanism updated its memory with the final new information, and then extracted the answer from the final memory which is "hallway".

**Attention Mechanism** Kumar et al. used a new type of attention mechanism called modified AttGRU using a gate mechanism. At each memory episode, they got the similarities ${z_i}^t$ between the fact representations $S_f$ as $\overleftrightarrow{F_i}$, question hidden representations $q$ and previous memory state $m^{i-1}$ as shown in Equation 4.3. Then, they used a score function $Z$ to produce the scalar scores based on the similarities results. This score function is a simple two Feed-Forward NN that it uses the Tanh function to get the output of the scalar score between (-1, 1) as shown in Equation 4.4. Then, it uses the Softmax function to normalize the scores and get the probability score for each sentence, and then the attention gate $g$ should be generated as shown in Equations 4.5 and 4.6.

$$ {z_t}^i = [\overleftrightarrow{F_i} \circ q \circ m^{i-1}; |\overleftrightarrow{F_i} - q|; |\overleftrightarrow{F_i} - m^{i-1}|] \tag{4.3} $$

$$ {Z_t}^i = W^{(2)} \tanh(W^{(1)} {z_t}^i + b^{(1)}) + b^{(2)} \tag{4.4} $$

$$ {g_i}^t = \text{softmax}({Z_t}^i) \tag{4.5} $$

$$ \text{softmax}({Z_t}^i) = \frac{\exp({Z_t}^i)}{\sum_{k=1}^{M_i} \exp({Z_t}^k)} \tag{4.6} $$

Afterwards, they extracted the contextual vector $e^i$ by replacing the update gate in the inner GRU with the attention gate that they generated ${g_i}^t$ at each memory episode. They called it Modified GRU which it weighted by the attention gate ${g_i}^t$ as shown in Equation 4.7. In the end, the final contextual vector is the final hidden state of the inner GRU $e^t = {h_i}^t$.

$$ {h_i}^t = {g_i}^t GRU(\overleftrightarrow{F_i}, {h^t}_{i-1}) + (1 - {g_i}^t) {h^t}_{i-1} \tag{4.7} $$

**Memory Mechanism** The memory mechanism is used for updating the memory per episode with the new candidate sentences. After getting the contextual vector $e^t$ per episode, they used it alongside the previous memory state $m^{t-1}$ to update the current memory. The final memory state is the final hidden state of the outer

GRU as shown in Equation 4.8. The initial state of this GRU is the initial state of the memory which is initialized to the question vector $m^0 = q$.

$$m^t = GRU(e^t, m^{t-1}) \tag{4.8}$$

Finally, the final memory state $m^{T_m}$ is then passed to the answer module to generate the answer, where $T_m$ is the index of memory episodes.

#### 4.4.3.4 Answer Module

In this module, they generated the answer from the final memory state $m^{T_m}$ outputted by the memory module. There are two ways to generate the answer, either to generate a short answer (Single-word) or long answer (Multiple words in a span: (answer$_{start}$, answer$_{end}$)). For the short answer, we can use the linear Softmax function for generating the answer as shown in Equation 4.9.

$$y_t = \text{softmax}(W^{(m)} m^{T_m}) \tag{4.9}$$

For the long answer, Kumar et al. used another GRU for generating a multiple word answer and then used the linear Softmax function as shown in Equation 4.10 and 4.11. The initial state of this GRU is the last memory state $m^{T_m}$. Moreover, at each time step $t$ the input of the GRU is the concatenation of the question vector $q$ and previously predicted output $y_{t-1}$, and the previous hidden state equals $a^{t-1}$.

$$a^t = \text{GRU}([y_{t-1}, q], a^{t-1}) \tag{4.10}$$

$$y_t = \text{softmax}(W^{(a)} a^t) \tag{4.11}$$

Kumar et al. DMN architecture have shown high performance compared with other memory networks using Facebook bAbI dataset. It is a small and artificial generated dataset. Kumar et al. did not evaluate their DMN model using a large-scale dataset such as SQuAD. After many tries over Kumar et al. DMN, Xiong et al. (2016a) improved the original DMN model. They used Facebook bAbI dataset as well to evaluate their model, and their improved model showed high performance using this dataset. Briefly, their improvements were in two modules: input and episodic memory. In the input module, they used a component called sentence reader to encode the words into a sentence using positional encoding schema. This schema used to save the order of the words in each sentence. Moreover, they used another component called input fusion layer to encode sentences by using a bi-directional GRU instead of a single GRU to get all the information from the sentences before and after. In the episodic memory module, they tried the ReLU function instead of the GRU to update the memory. They used ReLU to get untied memory update and process this step faster. After examining and re-implementing Kumar et al. DMN, we propose our improved DMN with some improvements over Kumar et al. DMN in Section 4.4.4. The most significant addition is that we evaluated our improved DMN on a large manually generated dataset such as SQuAD v1.1.

### 4.4.4 Our improved Dynamic Memory Network model

In this section, we present our improved DMN over Kumar et al. DMN architecture. In the beginning, the first simulation of the re-implementation of Kumar et al. DMN can be taken from Appendix A. Afterwards, we tried several improvements over this simulation until we reached to the most effective improvements to form our improved DMN. First of all, we had a comparison between the Long-Short Term Memory Network Hochreiter and Schmidhuber (1997) and Gate Recurrent Unit Network (GRU) Chung et al. (2014). We concluded that both performed similarly, but GRU is less computationally complicated and less expensive. However, LSTM showed good results with the long sentences but it is more computationally expensive. GRU is less computationally and less expensive since it consists of only two gates instead of three gates as in LSTM. Moreover, both LSTM and GRU models perform better than the standard RNN model because they are using a gating mechanism to combat the vanishing gradient problem Hochreiter and Schmidhuber (1997) in the standard RNN. Our improved DMN is evaluated using SQuAD v1.1 and it would be the first try using the DMN approach in SQuAD leaderboard. After many attempts and experiments, our improvements are in two modules: the input and episodic memory.

#### 4.4.4.1 Improved Input Module

In Kumar et al. DMN, they passed the input data sentence by sentence to the encoder without using any cleaning techniques. Afterwards, they used a single GRU to encode each sentence to generate its fact representations $\overleftrightarrow{\mathbf{s}_t}$ then combine all sentences to the final fact representations $\overleftrightarrow{S}$. When the single GRU encode a sentence, it cares only about the information of the previous sentences. Our improvements in this module are defined as follows:

1. Sentence cleaner (clean the encoding characters and remove stopwords).
2. Encoding schema to encode words into sentences.
3. Rather of using single GRU encoder, we get all relations between sentences using a Bi-directional GRU encoder.

We have the raw input data which it contains several paragraphs, and each paragraph contains several contexts with some questions and answers pairs. Each context contains several sentences and then we fed the input data (sentence by sentence) into the encoder as shown in Figure 4.7. Sentence cleaner is the first improvement in this module which is about cleaning the data before use it. We found that SQuAD dataset version 1.1 needs some encoding characters cleaning. The dataset is a JSON file and there are many different encoding characters. These differentiate made some issues such as in this sentence "champion Carolina Panthers 24\u201310", which this sentence should be "champion Carolina Panthers 24–10". Unfortunately, these issues affect the answer in the end. In addition to that, we found that many of predicted answers include stopwords. Because of that,

we decided to clean the data from stopwords using stopwords package from Natural Language Toolkit (NLTK)[5]libraries. We removed stopwords such as (a, an, the, which, like, that, is, are and others), as shown in Figure 4.8. The number of stopwords removed was around 57 million words. By removing these words from the raw data, the performance and results increased by 3% higher than without removing the stopwords. After getting the predication answer-span, we interpret the tokens from the original JSON file dataset to retrieve the stopwords again in the file answer text. As shown in Figure 4.7, now we can pass both the input text

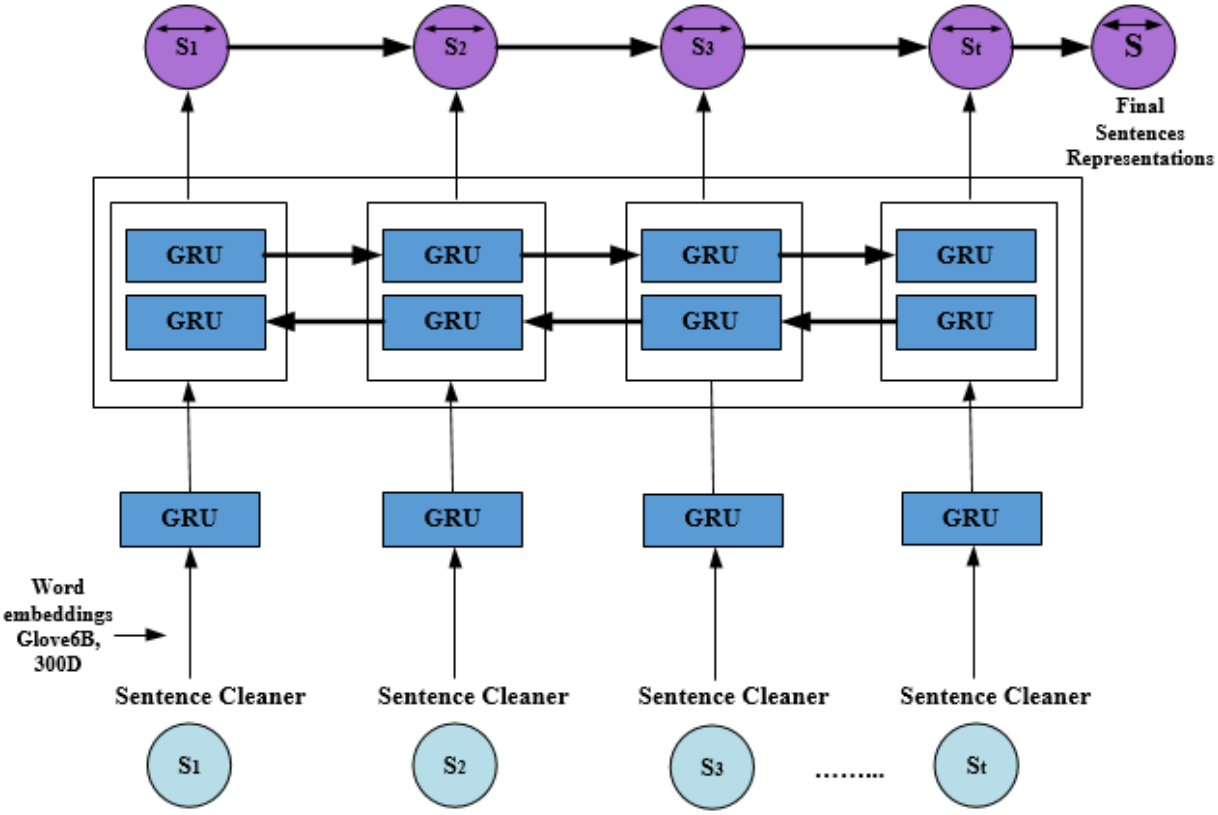


Figure 4.7: Our improvements in the input module, drawn in the style of Xiong et al. (2016a) in their paper.

sequences and question to the next stage. Before going to the next stage, we used Glove6B with 300 dimensions to generate the words vector representations for both (context and question). The next improvement is about using one of the encoding schemes such as Positional Encoding introduced by Sukhbaatar et al. (2015), GRU or LSTM. Using positional encoding schema is easier than GRU or LSTM but less accuracy with long sentences.

[5]NLTK is a set of libraries, used to build and implement symbolic and NLP programs in Python programming language Created by Guido van Rossum (1991).

```
import nltk
from nltk.corpus import stopwords
 set(stopwords.words('english'))
```

{'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during', 'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its', 'yours', 'such', 'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'from', 'him', 'each', 'the', 'themselves', 'until', 'below', 'are', 'we', 'these', 'your', 'his', 'through', 'don', 'nor', 'me', 'were', 'her', 'more', 'himself', 'this', 'down', 'should', 'our', 'their', 'while', 'above', 'both', 'up', 'to', 'ours', 'had', 'she', 'all', 'no', 'when', 'at', 'any', 'before', 'them', 'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does', 'yourselves', 'then', 'that', 'because', 'what', 'over', 'why', 'so', 'can', 'did', 'not', 'now', 'under', 'he', 'you', 'herself', 'has', 'just', 'where', 'too', 'only', 'myself', 'which', 'those', 'i', 'after', 'few', 'whom', 't', 'being', 'if', 'theirs', 'my', 'against', 'a', 'by', 'doing', 'it', 'how', 'further', 'was', 'here', 'than'}

Figure 4.8: Stop-word list from the NLTK Python package.

We decided to use a single GRU to save the position and order of words at each sentence as shown in Equation 4.12.

$$s_t = \sum_{i=1}^{M} GRU({x_t}^i, {w_{t-1}}^i) \tag{4.12}$$

Where $s_t$ is the collection of words after encoding them in each sentence. The $t$ is the index of sentence, $i$ is the index of word and $M$ is the number of words in each sentence. Then, we pass the sentences $s_t$ to the last improvement. We use a bi-directional GRU instead of a single GRU like Kumar et al. did. Bi-directional GRU allows us to retrieve all relations between sentences by getting all information about each sentence before and after. In the single GRU, we can only get the information about the sentence before. As shown in Figure 4.7, bi-directional arrows refer to forward and backward GRU cells. Then, we combine both sentences hidden representations to get the final bi-directional hidden representation as shown in Equations 4.13, 4.14 and 4.15.

$$\overrightarrow{\mathbf{s}_t} = GRU_{forward}(s_t, s_{t-1}) \tag{4.13}$$

$$\overleftarrow{\mathbf{s}_t} = GRU_{backward}(s_t, s_{t-1}) \tag{4.14}$$

$$\overleftrightarrow{\mathbf{s}_t} = \overrightarrow{\mathbf{s}_t} + \overleftarrow{\mathbf{s}_t} \tag{4.15}$$

In th end, we can extract the context vector $\overleftrightarrow{\mathbf{S}_t}$ by combining all the final hidden representations for each sentence $\overleftrightarrow{\mathbf{s}_t}$, and then we fed $\overleftrightarrow{\mathbf{S}_t}$ into the episodic memory module.

#### 4.4.4.2 Improved Episodic Memory Module

As shown in Figure 4.9, there are two components in the episodic memory module according to Kumar et al. DMN architecture. We have two improvements in this module to increase the performance and results. Our improvements are defined as follows:

1. Rather of using single AttGRU, we use attention Bi-directional GRU to get all relations between sentences after using the attention mechanism.

2. Rather using GRU or ReLU to update the memory, we use a single LSTM.

As shown in Figure 4.9, after generating the final sentences representations $\overleftrightarrow{\mathbf{S}_t}$, we iterate over these representations and extract the important information against the question using attention mechanism. As Kumar et al. did in this module in their DMN, we calculate the gate mechanism then we use our modified attention mechanism. Kumar et al. used AttGRU with a single GRU as attention mechanism. In our improvement, we use bi-directional GRU and replace the update gate in each GRU cell (forward and backward) with the gate mechanism that we calculated beforehand as shown in Equation 4.16. The reason of using the bi-directional GRU that we want to extract the contextual vectors $c^t$ per each memory episode $m^t$ based on all interactions between sentences after using attention mechanism as shown in Equations 4.17, 4.18 and 4.19. The advantage of this improvement that it might decrease the number of memory episodes.

In Kumar et al. example shown in Figure 4.6, they needed two memory episodes to get the answer. However, by letting each sentence get the information of sentences before and after, it could help to get the optimal probability score for each sentence using one-hop memory. Thereby, it decreases the number of memory episodes. When the number of memory episodes became smaller, it leads to higher performance. As we mentioned earlier, Kumar et al. used the memory mechanism to update the memory with the relevant information per episode. Then, the final memory state is fed into the answer module. In the light of that, the last improvement is about using LSTM instead of GRU or ReLU to update the memory. We have tried using GRU and ReLU as shown in Equations 4.20 and 4.21, taking into account the question vector is the initial memory $m^0 = q$. We found that ReLU shows higher performance and faster than GRU. However, we wanted to try using LSTM as shown in Equation 4.22. We found that LSTM and GRU are doing the same behavior, but LSTM is much better than GRU with the long sentences, and SQuAD dataset has very long sentences. Moreover, LSTM approximately showed the same performance as ReLU.

$$g_i{}^t = \text{softmax}(Z_t{}^i) \tag{4.16}$$

$$\overrightarrow{\mathbf{c}_i^t} = g_i{}^t GRU_{forward}(\overleftrightarrow{S_i}, c^t{}_{i-1}) + (1 - g_i{}^t)\, c^t{}_{i-1} \tag{4.17}$$

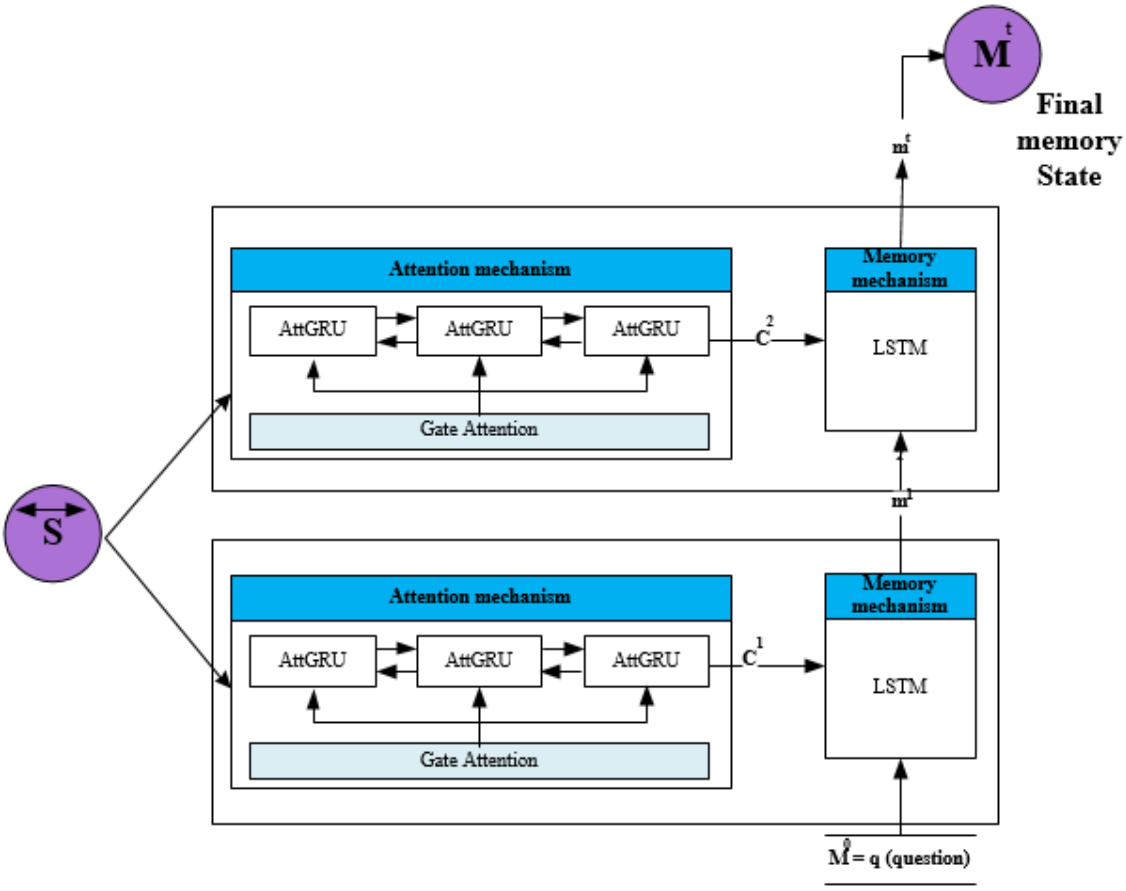


Figure 4.9: Our improvements in the episodic memory module, drawn in the style of Xiong et al. (2016a) in their paper.

$$\overleftarrow{\mathbf{c}_i}^t = {g_i}^t GRU_{backward}(\overleftrightarrow{S_i}, {c^t}_{i-1}) + (1 - {g_i}^t){c^t}_{i-1} \tag{4.18}$$

$$\overleftrightarrow{\mathbf{c}_i}^t = \overrightarrow{\mathbf{c}_i}^t + \overleftarrow{\mathbf{c}_i}^t \tag{4.19}$$

$$m^t = GRU(\overleftrightarrow{\mathbf{c}_i}^t, m^{t-1}) \tag{4.20}$$

Where the initial state of this GRU is the initial state of the memory initialized to the question vector $m^0 = q$.

$$m^t = ReLU(W^t[m^{t-1}; \overleftrightarrow{\mathbf{c}_i}^t; q] + b) \tag{4.21}$$

Where ; is the concatenation operator, $W^t$ the weight of each hidden unit and b is the bias (constant value).

$$m^t = LSTM(\overleftrightarrow{\mathbf{c}_i}^t, m^{t-1}) \tag{4.22}$$

Where the initial state of this LSTM is the initial state of the memory initialized to the question vector $m^0 = q$.

Last but not least, the final memory state $m^{T_m}$ is then fed to the answer module, where $T_m$ is the index of memory episodes. For the question and answer modules, we used the same structure in Kumar et al. DMN.

# Chapter 5

# Data and Processing

## 5.1 Stanford Question Answering Dataset

SQuAD dataset introduced by Rajpurkar et al. (2016). The paragraph in SQuAD version 1.1 came from 536 Wikipedia articles covering a wide range of topics. Each paragraph has several contexts, and each context had several sentences and associated with several questions. There are 23,215 contexts and 107,785 questions in total. SQuADv1.1 consists of training and testing datasets as JSON files. The first JSON file (32MB) for the training set contains around 87,599 question-answer pairs. The second JSON file (8MB) for the testing set contains around 10,570 question-answer pairs. The answer for each question is always a span of ($\text{answer}_{start}$, $\text{answer}_{end}$) tokens in the context. Regarding the training dataset, we split this dataset into train and validation sets according to the hold-out cross-validation technique in machine learning community of around 90%, 10% split. The validation set used to check the reliability of the training model as a process of the cross-validation and to avoid the overfitting problem of the training model. However, we use the testing data to evaluate our model.

The advantages of using SQuAD are that we do not need a knowledge base to implement QA system. In addition to that, SQuAD dataset is collected from several Wikipedia articles which it is an open domain dataset with the real-world complexity. In SQuAD dataset, we have three ground truth answers for each question about a context as shown in Figure 5.1. The first answer predicted by NNs models. The second and third answers predicted by humans. These truth answers are used for evaluating our our predicted answers. In the end, we have the official evaluation script available on SQuAD source to evaluate our prediction JSON file of our model against the other models in SQuAD leaderboard.

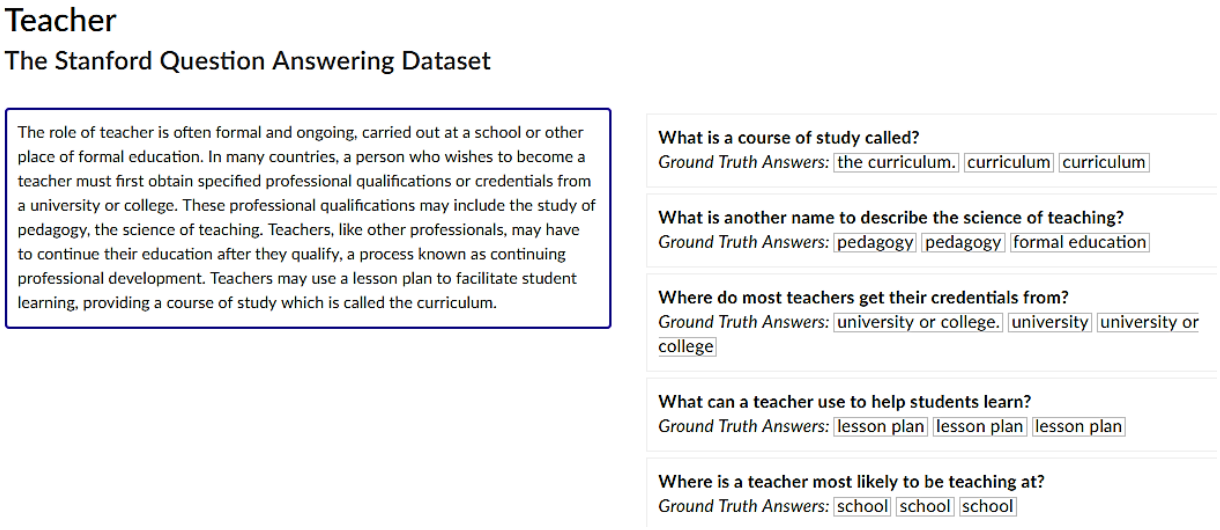


Figure 5.1: Another training example from the SQuAD dataset, consisting of a question, context and the ground truth and prediction answers (Rajpurkar et al., 2016).

## 5.2 Preprocessing and Configuration

Concerning reading comprehension approach, QA system should have the ability to read each word in each sentence in each context in each paragraph, and then understand the meaning of these words. Afterwards, it should extract the relevant information might lead to answer questions given; like humans do on any exam papers. Following sections, we present the preprocessing stage which it consists of four steps: tokenizing the data, create the vocabulary file of words, mask each word in the input data and the word embedding method such as Glove (Pennington et al. (2014)). Then, we present some of our important configurations which trained by our experiments.

### 5.2.1 Preprocessing

In the preprocessing stage, we convert the textual data into a numerical space. NNs understand only numbers, so we need to transform the data into real vector representations. Intuitively, we have different dimensions of input sequences for both (context and question), so we need to pad them into the same length by fixing 0 in the extension length.

#### 5.2.1.1 Tokenizing the data

After cleaning the SQuAD data as we described in Section 4.4.4 and Similar to any NLP tasks, firstly, we tokenize all data (context and questions). Tokenizing the data means split each sentence into several tokens. Each token has a word, and this word could be a text-word, a punctuation mark or a number. We use the

NLTK-tokenizer Stamatatos et al. (1999) package.

#### 5.2.1.2 Create the vocabulary

We create the vocabulary file by counting each word occurring in the whole dataset then we map these words to unique numbers. Let say that $W$ is the total number of words in the dataset and our vocabulary is $V$. Which V maps W $\rightarrow$ 0,1,2,3, ... *N-1*. Then, we encode each word to a unique number in our vocabulary. According to our experiments, we have around 8880555 unique words in the vocabulary file. For the empty word, we map it to 0 such as V(' ') = 0. Afterwards, we use this vocabulary file to convert the textual data into a natural number. Taking into account that, there is no maximum length for the number of sentences or the number of words in each sentence in both (context or question). Because of that, we create a dynamic vector space representation.

#### 5.2.1.3 Mask the input representations

As we mentioned earlier, the dimension of sentences (context and question) often are different in length. The dimension of the sentence means the number of words in each sentence. To solve this issue, we pad each input sequence to be in the same length by adding 0 value for each empty word extension. There are two ways to pad the sentence: pre-padding and post-padding. Pre-padding means that we will add 0 value to each word extension at the beginning of the sentence vector. Post-padding means that we will add the 0 value for each word extension at the end of the sentence vector. Padding step is very effective when we decide to do the batch processing. Batch processing is to split the training data into mini-batches to be able to train the whole data. If we do not want to pad the input sequence to reduce the huge waste of memory, we can use a single batch size to process sequence by sequence. With the single batch, we do not need to do the mask step. In our implementations, we used the whole SQuAD dataset which it is so large. Because of that, we need to do batch processing. However, we did pad the data sequences. Therefore, we should differentiate between the real words values and the empty word values. Because of that, we create the mask vector for the input text sequences and questions. For all recurrent operations in TensorFlow implementation (e.g. dynamic_rnn), they need a sequence_length, so we should mask the input sequences. As shown in Equation 5.1, we have the word with indexes $\{i, j\}$, which $i$ is the index of the word, and $j$ is the index of the sentence. If $W_{i,j}$ equals to 0 which it means that it is an empty word then, the mask of word $M_{i,j}$ sets to 0 otherwise 1 which it means that it is a real word.

$$M_{i,j} = \begin{cases} 1 & if \ W_{i,j} \neq 0 \\ 0 & if \ W_{i,j} = 0 \end{cases} \tag{5.1}$$

#### 5.2.1.4 The word embedding

After masking the input representations, we want to transfer the natural numbers to high dimensional vector representations. We trained the input sequences representations with the vocabulary file using word embeddings technique such as Glove6B with 300-dimensions (words are not randomly initialized) as described in Section 4.1. We utilized the pretrained GloVe vectors to obtain a word vector for each word in the context and question. All out-of-vocabulary words are mapped to the unknown token "UNK". Afterwards, we create the embedding vectors as the input of our neural networks.

### 5.2.2 Configuration

In this section, we present the important configurations that we used for our experiments. Before starting the training stage, we fix some configurations randomly. In the beginning, we use a random learning rate which is used for optimizing the loss value. NNs allow us to optimize the loss value by adjusting the weights of our model in the back-propagation stage. We start with a lower learning rate to go slower along the slope of gradient descent. Afterwards, we can also apply other sophisticated optimization algorithms like Adagrad (Duchi et al., 2011), Adadelta (Zeiler, 2012) or Stochastic Gradient Descent (Robbins and Monro, 1985) apart from our final choice of Adam (Kingma and Ba, 2015) for all implementations in this thesis. To apply Adam optimizer algorithm, we should optimize the gradient descent based on the learning rate (Pascanu et al., 2013).

First, we apply an exponential decay function to lower the learning rate and then get the decayed learning rate. Afterwards, we use a function to clip the global norm based on the decayed learning rate to fix the gradient values between two numbers. Before starting the training step, we need to fix the maximum gradient norm with a small value (in several experiments was 5). Gradient clipping function (Pascanu et al., 2013) became most common in RNNs implementations. Gradients are being calculated in the back-propagation stage through time to optimize the loss-error. These new gradients could vanish (much decreased) or explode (much increased). For vanishing gradients, they are multiplied by numbers less than one; thus the final error gradients are smaller than one. This problem is called "Vanishing gradients problem". On the other hand, for exploding gradients, they are multiplied by numbers larger than one; thus the final error gradients are larger than one. This problem is called "Exploding gradients problem". Therefore, the gradient clipping function is used for optimizing error gradients between two numbers to prevent them from getting larger or smaller than one. LSTMs and GRUs solved these problems using the gate mechanism.

In the training stage, we decide to use batch processing, because we have a huge dataset such SQuAD and it will be difficult to train the whole data using a single batch. Batch processing step splits the whole data into samples including input

and labeled output based on the batch size. We can now calculate the gradient for each sample value and each observation yielding different values. Then, we get the average of the gradient values over all samples. By this way, we can combat vanishing and exploding gradients problems.

In the end, we should fix the number of epochs that we are going to train the model based on it beforehand. Moreover, we should fix the number of the hidden layers. Furthermore, we should fix the number of the evaluation layers which are used to detect how much the training step in each epoch is good using the validation dataset. We present all configurations values used for implementing each model in the next Chapter 6.

# Chapter 6

# Experiments and Results

## 6.1 Technical Description

In this thesis, all implementations are conducted in Google's computational framework of TensorFlow Abadi et al. (2016) using *Python* version 3.6. Python is a high-level programming language for object-oriented, design pattern, functional programming or deep learning tasks. By the time, Python shows high performance in data sciences field, which it allows data scientists and developers to build and compile their theoretical ideas into code. Nowadays, Python becomes alongside R programming language the most powerful choices for machine learning tasks. Back to TenserFlow, we implemented our models in this thesis using TenserFlow version 1.7. The Tenserflow framework has two main steps: build the graph and run the session. In TenserFlow, we can save checkpoints for the best scores during the training stage per epoch. In the beginning, we ran our models locally[1] and approximately the epoch took around 42 mins. Because of lacking the local machine potentials, we used a graphics processing unit (GPU) as our cluster to run the whole dataset to speed up computations. We ran all experiments using a GeForce GTX TITAN X Pascal GPU. Last but not least, approximately every epoch in our models took around 20 mins, and we have logged all the results into a log folder in the GPU.

## 6.2 Technical Results

As we mentioned in Section 4.3, we implement the most effective sequence-to-sequence (seq2seq) architecture used for tasks with sequential data (Cho et al., 2014). First of all, we prepared our datasets before starting the training and evaluation stages. As described in Section 4.4.4, we cleaned SQuAD dataset from all the stop-words (the, a, an and others) and solved all encoding characters issues. As described in the previous Chapter 5, SQuADv1.1 provides us with 2 JSON files for the training and testing datasets. According to the hold-out cross validation

[1]Local machine with Intel(R) Core(TM) i5-3210M CPU@2.25GHz 2.50 GHz, 8 GB RAM on Windows 10 Pro.

theory, we have divided the training dataset into Train/Valid sets. The training set is used to train the model to learn various features and parameters to produce the prediction. The validation set is used to optimize the trained model in order to increase the accuracy of the model and avoid the overfitting problem. However, the testing dataset is used to run the optimized model and then predict the output in the real world. By this way, we can log and compare the error analysis for each model as long as we use the same train/valid/test sets. For evaluating our models, we use two metrics to evaluate models accuracy: Exact Match (EM) and F1 score (a softer metric). Exact match metric is used to measure the matching our predicted answer exactly with any one of the three ground truth answers. On the other hand, the F1 score metric is used to measure the average overlap between our prediction and ground truth answers. Then, we use the available evaluation script From SQuAD as the official evaluation after getting the prediction answers during the testing stage.

In the following sections, we present our results for each model. The first results will be the evaluation F1 score on the validation dataset after showing the training setup details based on the training dataset. The second results will be the evaluation results on the testing dataset. We present just two runs (the first and final run) and some of summaries of all runs results can be taken from Appendix B. Then, we show a comparison table between all models. In the end, we present the error analysis results between our improved DMN and Match-LSTM with Answer-Pointer model using random $\sim$ 100 inputs from the testing dataset.

### 6.2.1 Baseline Results

In this model, we use a unidirectional LSTM and it knows as (vanilla) LSTM as we mentioned in Section 4.4.1. We tried to figure out how the baseline model can process a broad context and answer questions about it. Moreover, we used the baseline to figure out if questions can be answered by a given context in straightforward way. The training stage took around 9 hours on the GPU for all epochs. All training details can be taken from Table 6.1 as a first run.

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Evaluation-size |
|---|---|---|---|---|---|
| 30 | 50 | 0.001 | 0.2 | 250 | 256 |

Table 6.1: Run1: training setup.

After the first training run, we got 49% F1 score as the best score on the validation dataset in th end of the last epoch. The final evaluation results using the testing dataset can be taken from Table 6.2.

There are various ways for improving the training model in neural networks and closing the gap between the validation and testing results. Apart from increasing the training data size because we used all the training dataset available on SQuADv1.1, varying the parameters (dropout-rate, L2 and learning rates), or

| Epochs-number | Dropout-rate | F1-score | EM-score |
|---|---|---|---|
| 0 | 1 | 28% | 22% |

Table 6.2: Run1: evaluation results on the testing dataset.

making the network architecture simply broader or deeper could help to avoid the overfitting problem. After doing some optimizations by varying the parameters with ADAM optimization, the final training run details can be taken from Table 6.3.

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Evaluation-size |
|---|---|---|---|---|---|
| 30 | 80 | 0.005 | 0.65 | 250 | 256 |

Table 6.3: Final Run: training setup.

After the final training run, we got 56% F1 as the best score on validation dataset. The final evaluation results using the testing dataset can be taken from Table 6.4 and all remaining settings stayed the same as in the first run.

| Epochs-number | Dropout-rate | F1-score | EM-score |
|---|---|---|---|
| 0 | 1 | 36% | 34% |

Table 6.4: Final Run: evaluation results on the testing dataset.

### 6.2.2 Match-LSTM with Answer Pointer Results

According to what we presented in Section 4.4.2, we re-implemented this model using the encoder-decoder architecture. The training stage took around 12 hours for all epochs on the GPU. As a first run, the training details can be taken from Table 6.5

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Evaluation-size |
|---|---|---|---|---|---|
| 30 | 80 | 0.001 | 0.2 | 200 | 100 |

Table 6.5: Run1: training setup.

After the first training run, we got 42% F1 as the best score on the validation dataset. The final evaluation results using the testing dataset can be taken from Table 6.6.

After optimizing this model, the final training run details can be taken from Table 6.7. After the final training run, we got 72% F1 as best score on the validation dataset. The final evaluation results using the testing dataset can be taken from Table 6.8 and all remaining settings stayed the same as in the first run.

| Epochs-number | Dropout-rate | F1-score | EM-score |
|---|---|---|---|
| 0 | 1 | 47% | 39% |

Table 6.6: Run1: evaluation results on the testing dataset.

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Evaluation-size |
|---|---|---|---|---|---|
| 50 | 120 | 0.005 | 0.8 | 200 | 100 |

Table 6.7: Run2: tuning 1th training run.

| Epochs-number | Dropout-rate | F1-score | EM-score |
|---|---|---|---|
| 0 | 1 | 67% | 52% |

Table 6.8: Run2: evaluation results on the testing dataset.

### 6.2.3 Kumar et al. Dynamic Memory Network Results

In this model, we re-implemented and trained Kumar et al. (2016) DMN using SQuAD dataset as we presented in Section 4.4.3. The training stage took around 18 hours for all epochs on the GPU. As a first run, the training details can be taken from Table 6.9.

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Attention-size | Memory-hops | Evaluation-size |
|---|---|---|---|---|---|---|---|
| 30 | 32 | 0.001 | 0.9 | 250 | 300 | 5 | 256 |

Table 6.9: Run1: training setup.

After the first training run, we got 61% F1 as the best score on the validation dataset. The final evaluation results of the first run using the testing dataset can be taken from Table 6.10.

After optimizing the first Kumar et al. DMN training run, the final training details can be taken from Table 6.11. After the final training run, we got 75% F1 as the best score on the validation dataset. The final evaluation results using the testing dataset can be taken from Table 6.12 and all remaining settings stayed the same as in the first run.

| Epochs-number | Dropout-rate | F1-score | EM-score |
|---|---|---|---|
| 0 | 1 | 28% | 26% |

Table 6.10: Run1: evaluation results on the testing dataset.

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Attention-size | Memory-hops | Evaluation-size |
|---|---|---|---|---|---|---|---|
| 30 | 80 | 0.005 | 0.6 | 250 | 300 | 5 | 256 |

Table 6.11: Run2: training setup.

| Epochs-number | Dropout-rate | F1-score | EM-score |
|---|---|---|---|
| 0 | 1 | 34% | 29% |

Table 6.12: Run2: evaluation results on the testing dataset.

### 6.2.4 Our Improved Dynamic Memory Network Results

In this section, we present of our improved DMN model results according to what we proposed in Section 4.4.4. We started the training stage with the same final settings implemented on the Kumar et al. DMN model in Table 6.11. We just changed the number of epochs, learning rate and the number of memory hops. In this section, we show the results after adding our improvements in the input module and episodic Memory module respectively.

**Results with Input Module Improvements:** The training details that used to train our improved DMN after adding our improvements in this module can be taken from Table 6.13. After the final training run, we got 89% F1 as the best score on the validation dataset. The final evaluation results using the testing dataset can be taken from Table 6.14.

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Attention-size | Memory-hop | Evaluation-size |
|---|---|---|---|---|---|---|---|
| 20 | 80 | 0.002 | 0.6 | 250 | 300 | 7 | 256 |

Table 6.13: Final run after input module improvements: training setup.

| Epochs-number | Dropout-rate | F1-score | EM-score |
|---|---|---|---|
| 0 | 1 | 47% | 36% |

Table 6.14: Final run after input module improvements: evaluation results on the testing dataset.

**Results with Episodic Memory Module Improvements:** After implementing and evaluating the input module improvements in our improved DMN model, we implemented and evaluated the improvements of this module. The training details that used to train the model after adding our improvements in this module can be taken from Table 6.15. After the final training run with our improved DMN, we got 86.64% F1 as best score on the validation dataset. The final evaluation results using the testing dataset after the final training run can be taken from Table 6.16 and all remaining settings stayed the same as in the first run.

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Attention-size | Memory-hop | Evaluation-size |
|---|---|---|---|---|---|---|---|
| 20 | 80 | 0.002 | 0.6 | 250 | 300 | 7 | 256 |

Table 6.15: Final run after episodic memory module improvements: training setup.

| Epochs-number | Dropout-rate | F1-score | EM-score |
|---|---|---|---|
| 0 | 1 | 59% | 45% |

Table 6.16: Final run after episodic memory module improvements: evaluation results on the testing dataset.

In the end, Table 6.17 shows the final results of all model collected in one table to present a clear comparison. We trained and evaluated all models on SQuAD dataset, which it means that we can check the error analysis results. Regarding hyper-parameters in this table, we can get that there are no many changes in values. We used Dropout value to avoid the overfitting problem which it determines which information to keep and which to drop.

In Table 6.17, we can conclude that:

- In the baseline model, SQuAD dataset is a bit complex to be solved using a simple model using RNNs classes without attention or memory mechanism.
- In the Match-LSTM with Answer Pointer model, we got these results after re-implementing this model from scratch by ourselves, which it was so close to their single model in SQuAD leaderboard.
- In Kumar-DMN, we can find that scores were not so good with the re-implementation of Kumar et al. (2016) DMN architecture, because SQuAD dataset is a bit more complex than Facebook bAbI dataset which it was introduced by Weston et al. (2015) as the evaluation dataset for the original

DMN. SQuAD dataset has long sentences and complex interactions between words and sentences; so we need to add some improvements to their DMN model to be able to retrieve probable answers.

- In our improved DMN, we can get that our improvements were effective and using less epochs. Moreover, we increased the memory-hops to get more information might lead to answer questions. Finally, our final scores are so close to the Match-LSTM with Answer Pointer model and much better than Kumar et al. DMN model.

| | Baseline | Match-LSTM | Kumar-DMN | Improved-DMN |
|---|---|---|---|---|
| Epochs number | 30 | 50 | 30 | 20 |
| Batch size | 80 | 120 | 80 | 80 |
| Learning rate | 0.005 | 0.005 | 0.005 | 0.002 |
| Dropout rate | 0.65 | 0.8 | 0.6 | 0.6 |
| Hidden units | 250 | 200 | 250 | 250 |
| Evaluation units | 256 | 100 | 256 | 256 |
| Attention units | 0 | 0 | 300 | 300 |
| Memory hops | 0 | 0 | 5 | 7 |
| F1 score | 36% | 67% | 34% | 59% |
| EM score | 34% | 52% | 29% | 45% |

Table 6.17: Comparison between the final training parameters and evaluation results for all models.

Recently, during the working on this thesis, Wang and Jiang made many improvements in their Match-LSTM with Answer Pointer model and did ensembles models from big companies with big teams. Match-LSTM with Answer Pointer model is becoming one of the top 5 models in SQuAD leaderboard. While that, the implementations of all these models in this thesis were implemented by a single person.

### 6.2.5 Error Analysis

In this section, we propose the error analysis between our improved DMN and Match-LSTM with Answer Pointer (Wang and Jiang, 2016) models using random $\sim$ 100 inputs from the testing dataset. In the previous sections, we proposed some of the error analysis results in each model using the validation dataset. For more details results, the tables and figures in Appendix B might be useful. However, we wanted also to test the error using random $\sim$ 100 contexts from the testing dataset. All hyper-parameters used in our improved DMN can be taken from Table 6.18. Figure 6.1 shows the evaluation results of our improved DMN model at the end of the best epoch. We ran this experiment with 30 epochs, but the best results were in the epoch 10. Moreover, the loss value became NAN it means that the loss value not a number. The reason is that there are large gradients throw the learning stage larger than 1. Because of that, we did early stopping in the epoch 10. From Figure 6.1, we can see that the results are so close to the ground truth answers and the results are good with long sentences. The best F1 score was 69.6% and the best EM score was 61.5%.

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Attention-size | Memory-hop | Evaluation-size |
|---|---|---|---|---|---|---|---|
| 30 | 80 | 0.001 | 0.8 | 250 | 300 | 7 | 256 |

Table 6.18: Our improved DMN hyper-parameters details and then evaluated on random $\sim$ 100 contexts from testing dataset.

```
Epoch 10 out of 30
Best F1 score so far: 64.56157366206995
EPOCH: ==> 10 (Avg Loss: [Train: 2.955233401120869][Test: 2.628620982170105 ])
---------- Evaluating on Test Set ------------
F1: 69.63431580971086, EM: 61.53846153846154, for 500 samples
Log samples:
Ground Truth: $ 5 million, Our Answer: $ 5 million
Ground Truth: 2015, Our Answer: 2015
Ground Truth: new england patriots, Our Answer: new england patriots
Ground Truth: cbs, Our Answer: cbs
Ground Truth: sun life stadium, Our Answer: sun life stadium , and the san francisco bay area 's levi 's stadium
Ground Truth: beyoncé and bruno mars, Our Answer: coldplay with special guest performers beyoncé and bruno mars
Ground Truth: arizona cardinals, Our Answer: new england patriots
Ground Truth: levi 's stadium ., Our Answer: sun life stadium , and the san francisco bay area 's levi 's stadium
Ground Truth: denver broncos, Our Answer: denver broncos defeated the national football conference ( nfc ) champion carolina panthers
Ground Truth: " golden anniversary ", Our Answer: " golden anniversary "
Ground Truth: new orleans ' mercedes-benz superdome , miami 's sun life stadium , and the san francisco bay area 's levi 's stadium,
Our Answer: new orleans ' mercedes-benz superdome , miami 's sun life stadium , and the san francisco bay area 's levi 's stadium
Ground Truth: sun life stadium, Our Answer: levi 's stadium
Ground Truth: three, Our Answer: three
Ground Truth: american football conference, Our Answer: american football conference
Ground Truth: super bowl xlvii, Our Answer: super bowl xlvii
loss_test: 2.628620982170105 f1_test: 69.63431580971086 em_test: 61.53846153846154

Final Best score F1 so far: 69.63431580971086
Final Best score EM so far: 61.53846153846154
```

Figure 6.1: Some of the real results after testing our improved model using $\sim$ 100 contexts.

The change of the loss in training and in testing ($\sim$ 100 contexts) can be taken from Figure 6.2 and 6.3.

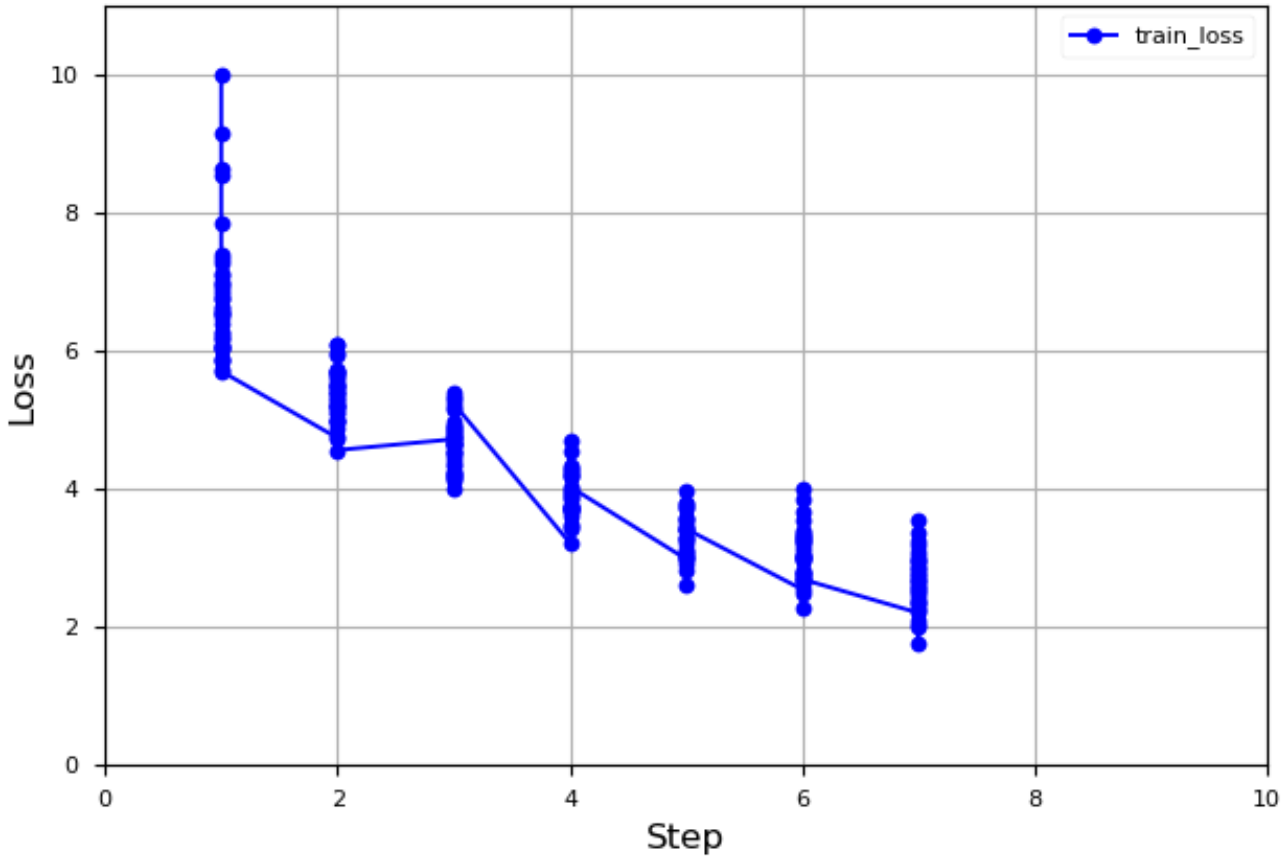


Figure 6.2: Plot of loss values in training. The x-axis depicts the step in epochs, and the y-axis is the loss value.

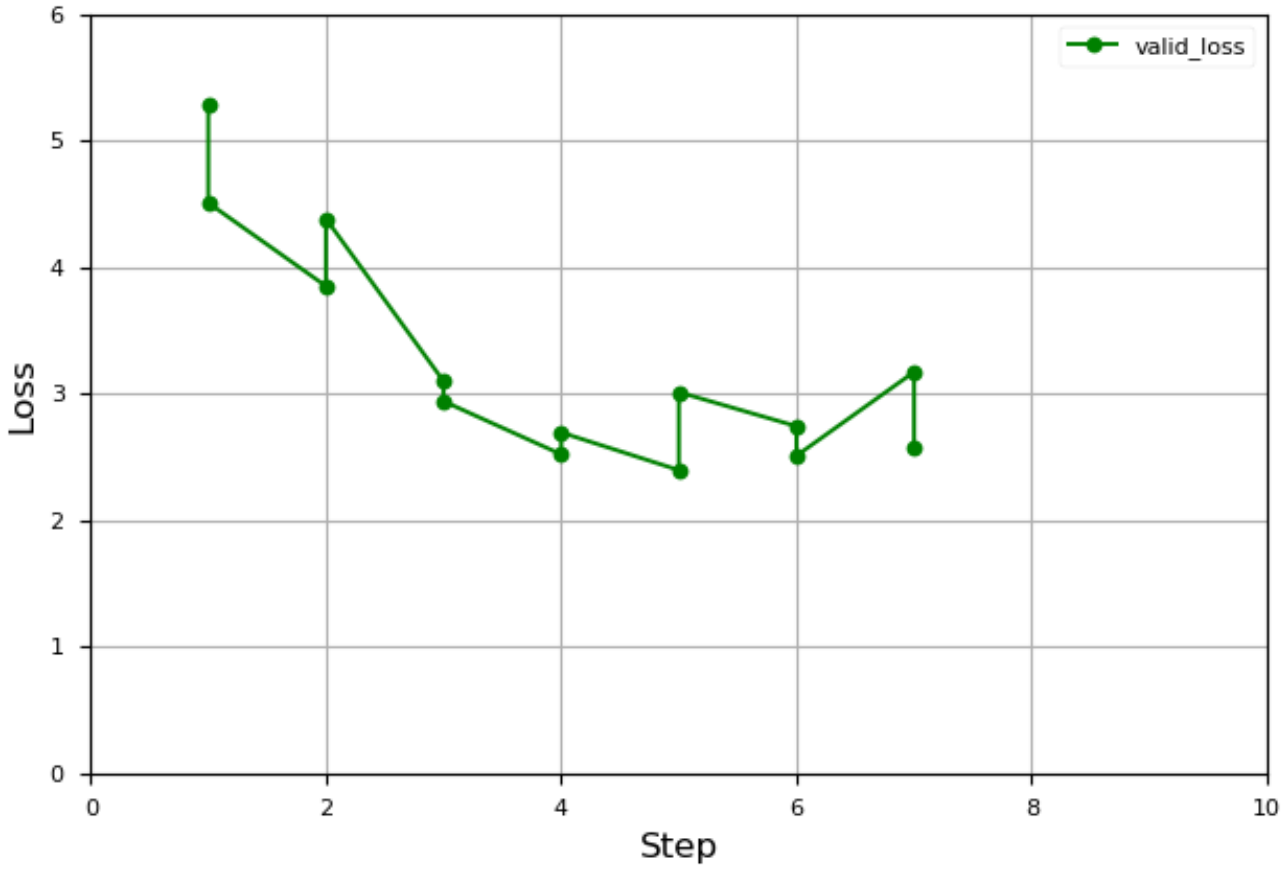


Figure 6.3: Plot of loss values in testing ($\sim$ 100 contexts). The x-axis depicts the step in epochs, and the y-axis is the loss value.

For Match-LSTM with Answer Pointer model, all hyper-parameters can be taken from Table 6.19.

| Epochs-number | Batch-size | Learning-rate | Dropout-rate | Hidden-units | Evaluation-size |
|---|---|---|---|---|---|
| 50 | 140 | 0.03 | 0.6 | 300 | 100 |

Table 6.19: Run2: tuning 1th training run.

Figure 6.4 shows the evaluation results of Match-LSTM with Answer Pointer model at the end of the best epoch. We ran this experiment with 50 epochs and the best results were in the epoch 35.

```
Epoch 35 out of 50
---------- Evaluating on Train Set ----------
loss_train: nan F1: 76.37506493807524, EM: 65.5, for 400 samples
Samples:
Ground Truth: 150,000, Our Answer: more than 150,000
Ground Truth: United Airlines Flight 175,
 Our Answer: American Airlines Flight 11 into the North Tower of the World Trade Center and United Airlines Flight 175
Ground Truth: 3-0, Our Answer: 3-0
Ground Truth: balloon , or airship , guns, Our Answer: balloon
Ground Truth: biodiversity, Our Answer: biodiversity
Ground Truth: written, Our Answer: written
Ground Truth: 1966, Our Answer: 1966
Ground Truth: outside of one 's region, Our Answer: family
Ground Truth: Slave dynasty of Delhi, Our Answer: The subsequent Slave dynasty of Delhi
Ground Truth: 9 meters, Our Answer: 10 km
---------- Evaluating on Val Set ------------
loss_val: 7.774 F1: 43.16862696672284, EM: 31.25, for 400 samples
Samples:
Ground Truth: Queen Victoria, Our Answer: Queen Victoria
Ground Truth: Migratory Bird Hunting Stamp Act, Our Answer: Bird Hunting Stamp Act
Ground Truth: the second half of the 15th century, Our Answer: second half of the 15th century
Ground Truth: " a universe with a god would be a completely different kind of universe from one without ,
 and it would be a scientific difference . ", Our Answer: the existence of God is an empirical question
Ground Truth: in a deal valued at $ 1 billion ( £640 million ) ., Our Answer: $ 1 billion
Ground Truth: YouTube Red, Our Answer: YouTube Kids
Ground Truth: unsatisfied, Our Answer: China 's income fell sharply during the wars as vast areas of farmland were destroyed
Ground Truth: 1169, Our Answer: 1169
Ground Truth: Post-punk, Our Answer: Post-punk

Final Best score F1 so far: 43.16862696672284
Final Best score EM so far: 31.25
```

Figure 6.4: Some of the real results after testing Match-LSTM with Answer Pointer model using $\sim$ 100 contexts.

In the end, the loss values in Match-LSTM with Answer Pointer model were always between (9.5 and 7.5) until the last epoch, and the best F1 score was around 43.2% and the best EM score was around 31%.

# Chapter 7

# Discussion

Throughout this thesis, we presented the Dynamic Memory Network (DMN) approach which it is the recent memory network class introduced by Kumar et al. (2016). Afterwards, we improved this architecture, and then we evaluated our improved DMN on a large-scale QA dataset called SQuAD. Kumar et al. were evaluated their DMN on a small QA dataset called Facebook bAbI dataset (Weston et al., 2015). The advantage of using SQuAD dataset for evaluating our models is that this dataset is a large-scale manually generated dataset. It consists of Wikipedia articles written by humans and questions about these articles are also created by humans. It means that it is an open domain dataset with the real-world complexity. Unlike, Facebook bAbI dataset which it is an artificially generated dataset like any English tests in schools. Therefore, SQuAD dataset is the perfect one for such QA task. In this thesis, we proposed the advantages of using memory and attention mechanisms in the QA task. By implementing different models, we noted that these advantages could appear on two challenges as follows:

- The first challenge is about working on long sentences in SQuAD dataset leading to low performance. To solve this problem, we used the attention mechanism that allows us to decide which parts in the input text sequences is most important to pay attention against the question keywords. Attention mechanism is used to retrieve all the relevant information might lead to answer the question, instead to compress all relevant information into a fixed-length vector.

- The second challenge is about working on the complex semantic relations between the sentences leading to low performance. As shown in the example that we proposed in Section 4.4.3, to answer the question in that example, we needed multiple memory-hops for getting all interactions or semantic relations between sentences to answer the question. To solve this problem, we used a memory mechanism by iterating over the sentences and getting all semantic relations. Then, we updated the global memory per-hop to extract the final memory state. It contains all relevant information might lead to answer the given question.

Kumar et al. DMN architecture is considered one of the flexible architecture for reading comprehension task. It consists of four separate modules that allow us and other researchers to understand and improve the process in any module in their architecture. The episodic memory module processes the output representations from the input and question modules by using the attention and memory mechanisms. In addition to that, we did not find a dynamic network using both of memory and attention mechanisms in SQuAD leaderboard. Therefore, Kumar et al. DMN architecture motivated us to improve their model and evaluate it on SQuADv1.1 dataset instead of Facebook bAbI dataset. By improving Kumar et al. DMN model and evaluating our improved model on SQuAD dataset, our improved DMN would be the first try of DMN approach in SQuAD. In the beginning, our brainstorming was focusing on what is the stat-of-the-art in SQuAD leaderboard. After researching and reading most of models in SQuAD, we found that Match LSTM with Answer-Pointer model is one of the state-of-the-art models and it is in a high rank in SQuAD leaderboard. Therefore, we decided to examine and re-implement this model to be as a comparative model to our improved DMN model.

In the following sections, all answers to any question in your mind might be found. By outlining our conclusions and scientific contributions, we propose answers to our research questions. In the end, we share our future work from what we have done.

## 7.1 Conclusion

We examined and re-implemented Kumar et al. (2016) DMN approach and evaluated it on the SQuADv1.1 dataset. Afterwards, we improved two major modules in Kumar et al. DMN architecture, and they were as follows: Input module and Episodic Memory module. Then, we evaluated our improved DMN on the SQuADv1.1 dataset to make a comparison between Kumar et al. and our improved DMN. From this comparison, we can know whether our improvements are useful and effective or they improved nothing in the final performance. For further evaluation, we re-implemented (Match LSTM with Answer Pointer) model credited to Wang and Jiang (2015) as a comparative model to our improved DMN. Then, we evaluated the former model using SQuADv1.1. From this comparison, we can know how can our improved DMN compete with other models in the SQuAD leaderboard such as Match-LSTM with Answer Pointer model.

Last but not least, we listed the results of the three models to show trade-offs between these models. Table 6.17 shows that our improved DMN outperformed the original DMN introduced by Kumar et al. (2016) using the large-scale dataset such as SQuAD. Consequently, our improvements increased the final scores by $\sim$ 13% high. Moreover, by comparing the results of Wang and Jiang Match-LSTM with Answer Pointer model and our improved DMN model, we found that the results of our model are so close to the latter model. Taking into account that

the SQuAD leaderboard was dramatically changing while i was working on this thesis. In addition to that, most of the top entries in the SQuAD leaderboard are ensembles models from different big companies by big teams, while i have a single model with single person efforts.

Here, we present our conclusions regarding our improvements over the original DMN approach, which they are in two modules as follows:

**Input module:**

- After cleaning encoding characters issues from the data and removing the stopwords according to the list of NLTK package, we found the final scores increased by 5% higher. It means that these improvements were so useful. Sometimes we could not retrieve the right answer because we could not interpret the word such as "24\u201310" in a sentence "champion Carolina Panthers 24\u201310". Consequently, after cleaning the data, we can interpret the right sentence which it should be "champion Carolina Panthers 24–10".

- On the other hand, by removing stopwords such as "which", "that", "the", like", "an", "a" and so on, it became easier to get a high exact match score than without removing stopwords. In the first experiments, we got several wrong answer words such as stopwords ("the the the"). Because of that, we decided to remove these stopwords to extract clean data to be fed into the encoder-decoder architecture. Moreover, these stopwords often were occurring a lot. The number of stopwords removed was around 57 million words. In our SQuAD dataset, for example, the word of "the" occurred more than 40000 times. This affected the vocabulary size and thus affected the word vector representations afterwards. However, after removing stopwords, we found that the EM score increased by 3% higher. Then, we could extract a meaningful answer even if it was not the optimal one not wrong answers such as "the the the".

- Regarding using Bi-directional GRU between the sentences, it showed an increase in the final scores because now we could get the important information from the sentence before and after not only the sentence before as the single GRU did. It means that we did not ignore any information and we get all interactions and relations between sentences.

**Episodic Memory module:**

- Using Bi-directional modified GRU with the attention mechanism also showed an increase in the final scores compared to Kumar et al. DMN. Because Kumar et al. used a single modified GRU with the attention mechanism to extract the context vector. As we mentioned in the Input Module, the unidirectional network cares about the sentence before only. However, we used

Bidirectional AttGRU or modified GRU to retrieve all interactions between sentences (before and after). By this way, we collected more useful information in the final context vector and then we updated the memory to extract the answer.

- As we mentioned earlier, we can update the memory by applying three kinds of NN layers: LSTM, GRU, ReLU. For GRU and LSTM, the final scores after updating the memory did not show many differences. However, we noted that LSTM is much better for a long sentence but took more time-consuming. Moreover, ReLU showed much better in the final scores and faster than using GRU or LSTM.

In the end, by evaluating our improved DMN, we could say that using the memory mechanism alongside the attention mechanism might solve different problems concerning NLP and Text Mining tasks. Furthermore, we could say that the DMN approach can show high performance in a large manually generated dataset such as SQuAD, as it showed high performance in Facebook bAbI dataset. The harmonious division of the DMN approach modules makes it so flexible framework to be used, and it might raise researchers to improve it regarding QA task in the future.

Finally, based on our conclusions, the answers to our research questions can be taken from Table 7.1.

| Results Summary | | |
|---|---|---|
| Question | Answer | Status |
| RQ 1 | - Our improved DMN showed final score 59%.<br>- Our improvements were so useful and effective.<br>- Our model outperformed Kumar et al. DMN on SQuADv1.1. | ✔ |
| RQ 2 | - By comparing our improved model with Match LSTM model, Our model was so close to the latter model and by improving it, it could compete other models in SQuAD leaderboard. | ✔ |

Table 7.1: Summary of answers to research questions.

As a summary:

- We conclude that our improved DMN can be an appropriate model to solve the question answering task for reading comprehension.
- We conclude that by merging the memory and attention mechanisms by the way that Kumar et al. introduced, is a promising way to solve complex NLP tasks.

## 7.2 Future Work

After submitting this thesis, some research questions have been raised and need to be answered. Due to the limited time-scope of the master thesis, we could not improve our improved DMN model scores more. The most important open question might be in our minds is "How high is the performance of our improved DMN against Kumar et al. DMN evaluated by Facebook bAbI dataset?". Unfortunately, we had not the time-space to explore and evaluate our improved DMN using Facebook bAbI dataset. However, we can give a temporary answer to this question by looking at the results shown in Table 6.17. From this table, we can get that our improved DMN model outperformed Kumar et al. DMN model evaluated by SQuAD dataset. As we mentioned earlier, that SQuAD dataset is one of the largest reading comprehension datasets and manually generated by humans with real-world complexity. In contrast, Facebook bAbI dataset is artificially generated dataset with short sentences. From here, we can conclude that our improved DMN might if not must outperform Kumar et al. DMN using Facebook bAbI dataset.
By the end of July 2018, Rajpurkar et al. (2018) proposed SQuADv2.0, that the significant difference to SQuADv1.1 is that SQuADv2.0 combines existing SQuADv1.1 data with over 50,000 unanswerable questions. The answer to each question does not necessarily be involved in the context. It is a big challenge task in NLP, and we plan to evaluate our improved DMN using SQuADv2.0 in the future. As investigated, we thought about doing fine-tuning of the word vector representations extracted from the word embedding method before using them. Moreover, we thought about exploring the results if we try to use the attention mechanism on both directions (question against context and context against question) to be as a Bi-directional attention mechanism. In the end, we would like to test our improved DMN model in the real-life systems such as "Detecting common opinions among online customer reviews" or "Finding alternative sources for answering the same question" and others.

# Appendix A

# Complete Simulation Figures

In Figure A.1, we show the first simulation of Kumar et al. (2016) DMN architecture and It was our proposal architecture.

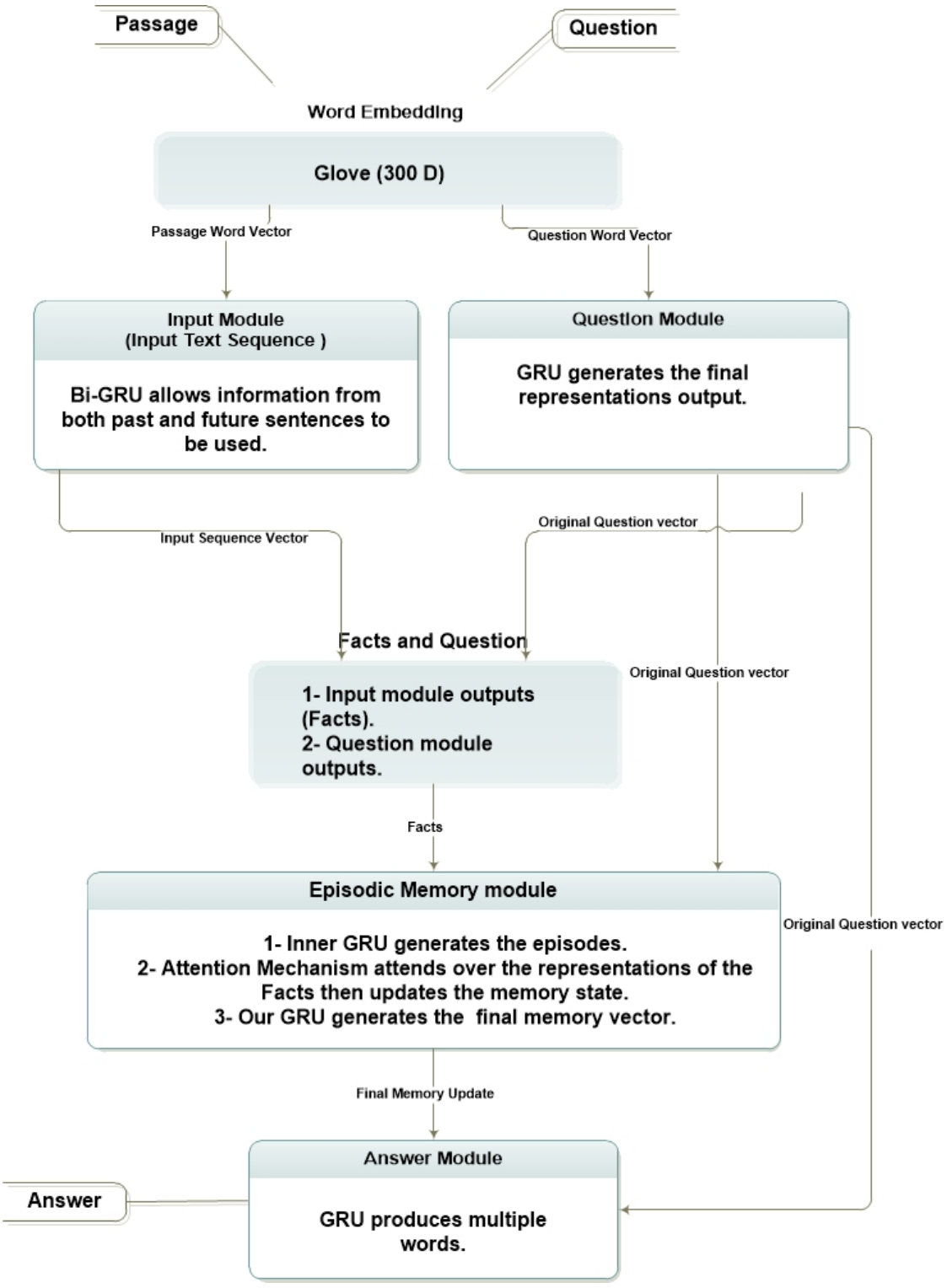


Figure A.1: Our first simulation of the re-implementation of Kumar et al. (2016) DMN architecture.

# Appendix B

# Additional Tables and Figures

In this chapter, all tables and figures that express our results after the final optimization of our improved DMN model are provided.

As shown in Figure B.1, we show the final results after using the validation set to evaluate the trained model. As we mentioned in Section 5.1, we split the training dataset into training and validation sets.

```
Epoch 30 out of 30
Best F1 score so far: 86.6687740184746
---------- Optimizing on Train Set ----------
---------- Validating using Valid Set ----------
EPOCH: ==> 30 (Avg Loss: [Train: 1.0212298391095127][Val: 1.1019878204052265 ])
---------- Evaluating on Val Set ------------
F1: 86.49628290384501, EM: 80.8107067205854, for 500 samples
Log samples:
Ground Truth: james watt, Our Answer: james watt
Ground Truth: remote sensing, Our Answer: remote sensing
Ground Truth: super bowl, Our Answer: super bowl
Ground Truth: acasta gneiss, Our Answer: acasta gneiss
Ground Truth: 2005, Our Answer: 2005
Ground Truth: outdated or only approproriate if herbal remedies were on offer to a large extent,
Our Answer: outdated or only approproriate if herbal remedies were on offer to a large extent
Ground Truth: sherwood boehlert, Our Answer: rep. joe barton
Ground Truth: polynomial-time reduction, Our Answer: polynomial-time reduction
Ground Truth: tate britain, Our Answer: tate britain
Ground Truth: 8, Our Answer: 8
EPOCH: %d ==> 30 ,loss_val: 1.1019878204052265 f1_val: 86.49628290384501 em_val: 80.8107067205854

Final Best score F1 so far: 86.6687740184746
Final Best score EM so far: 80.93587521663778
```

Figure B.1: The final results after evaluating our improved DMN using the validation set.

The change of the loss values in training and validation can be taken from Figures B.2 and B.3. For a clear comparison between the change of loss values, we combined the two figures into a single figure, and then we proposed Figures B.4 and B.5.

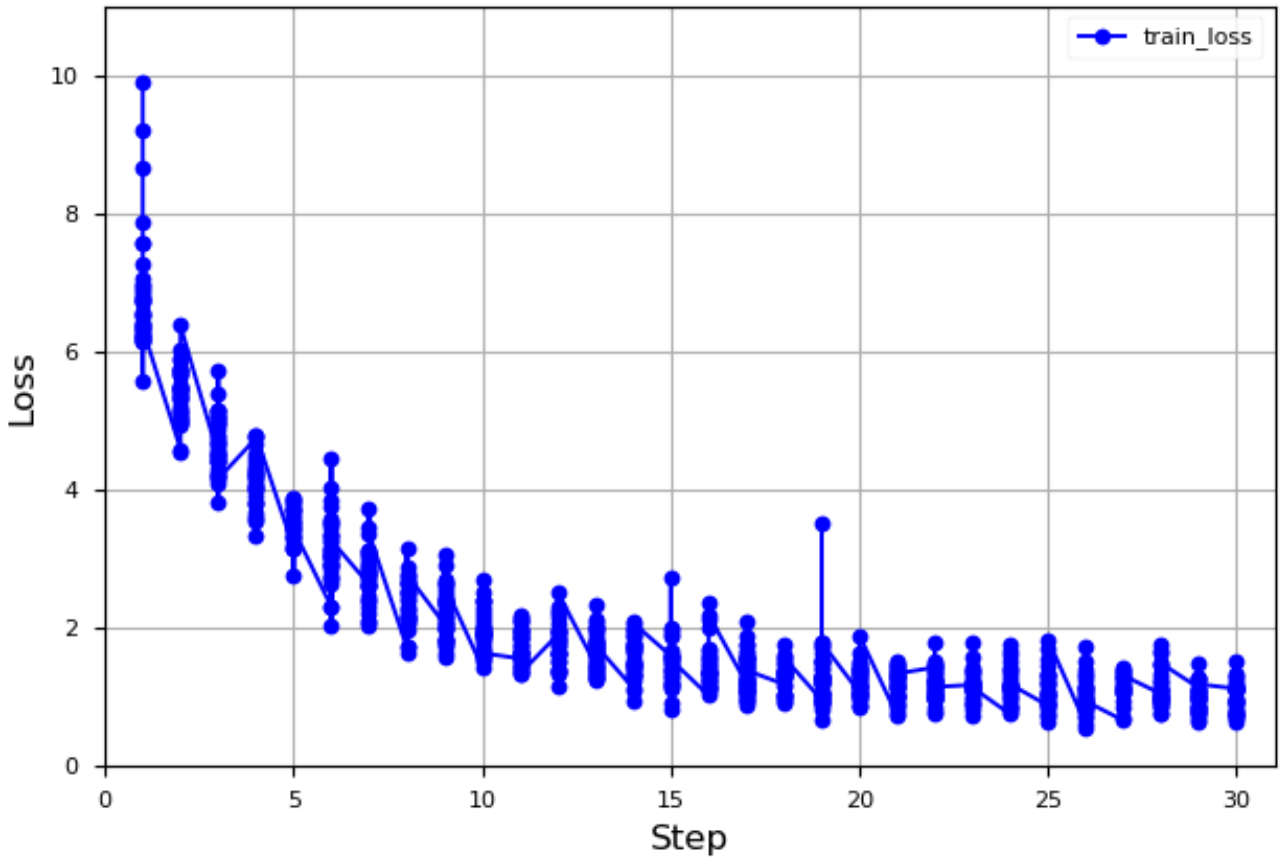


Figure B.2: Plot of loss values in training. The x-axis depicts the step in epochs, and the y-axis is the loss value.

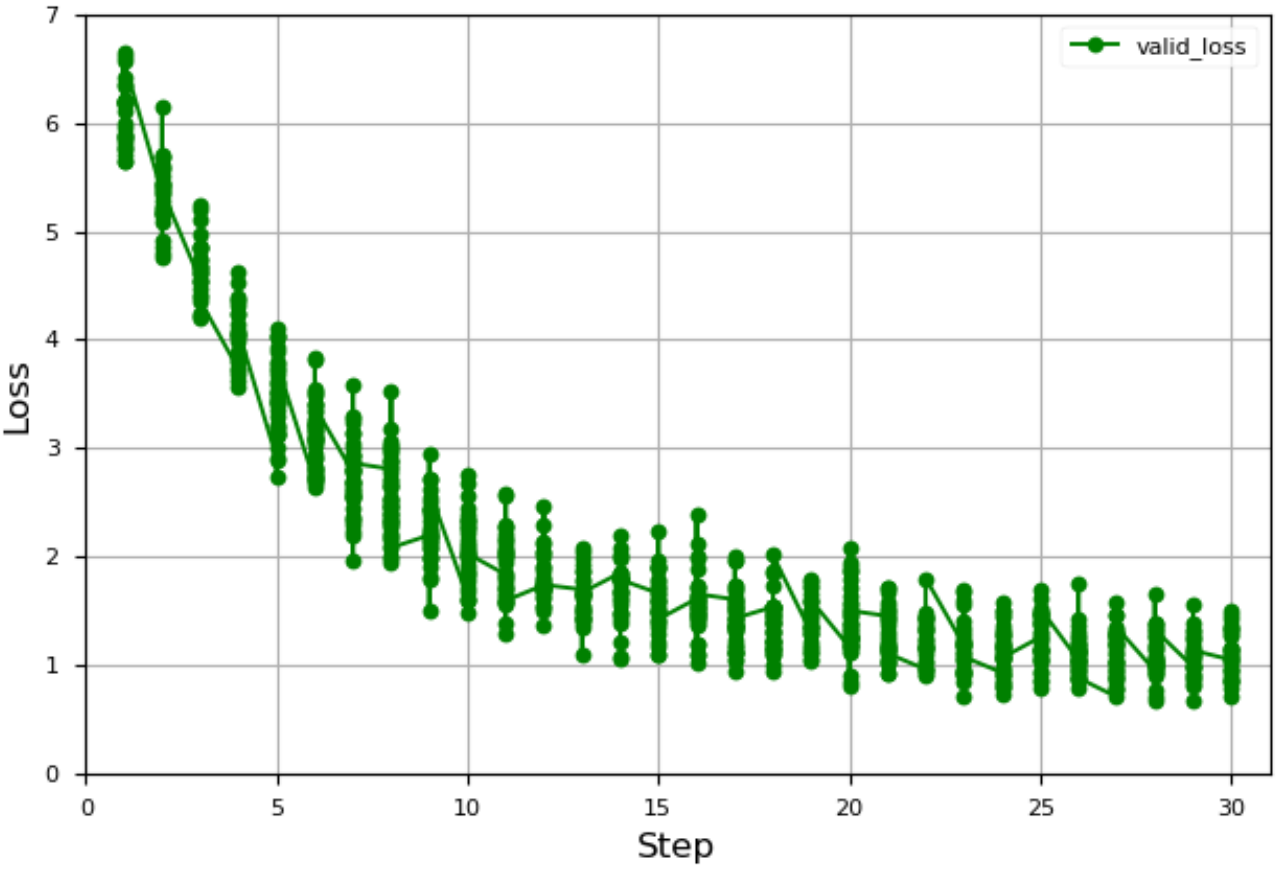


Figure B.3: Plot of loss values in validation set. The x-axis depicts the step in epochs, and the y-axis is the loss value.

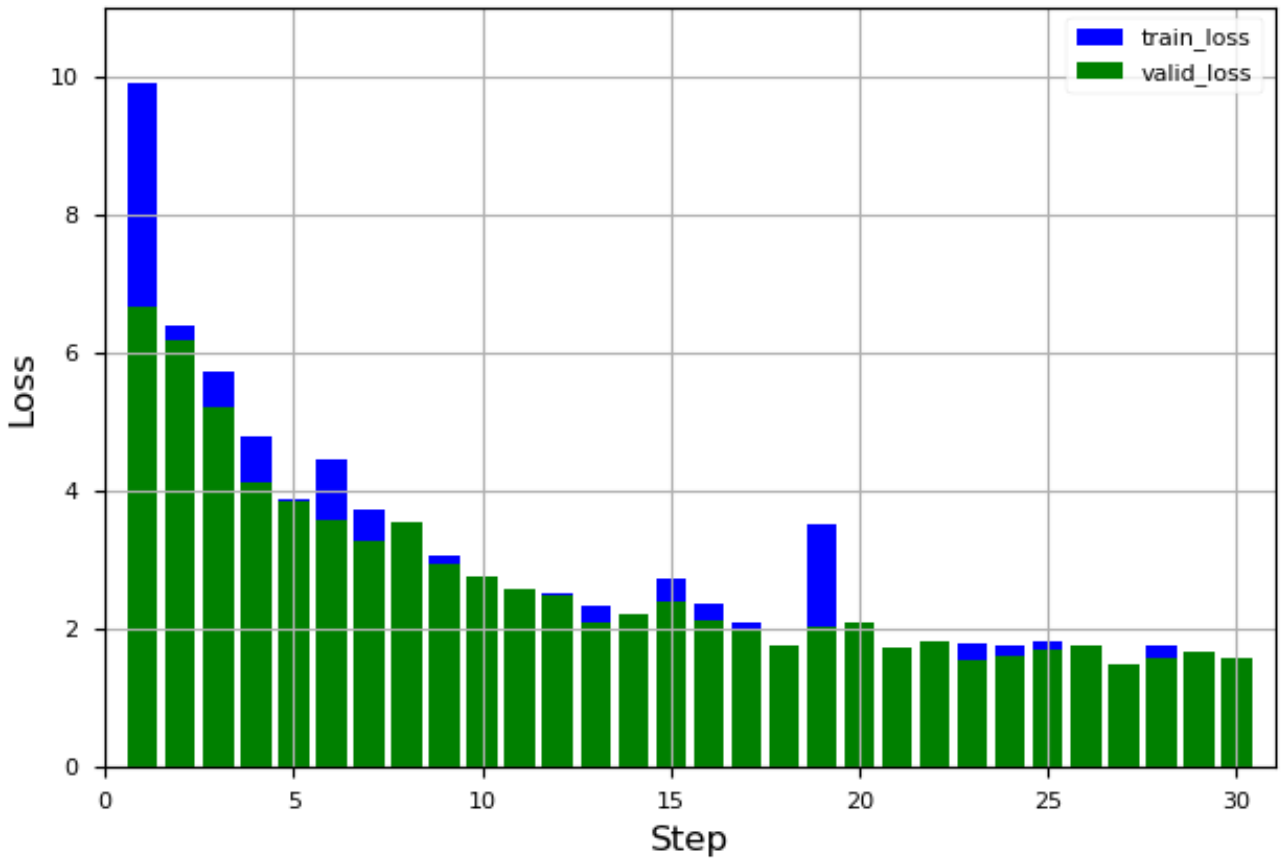


Figure B.4: Bar-plot of loss values in training and validation sets. The x-axis depicts the step in epochs, and the y-axis is the loss value.

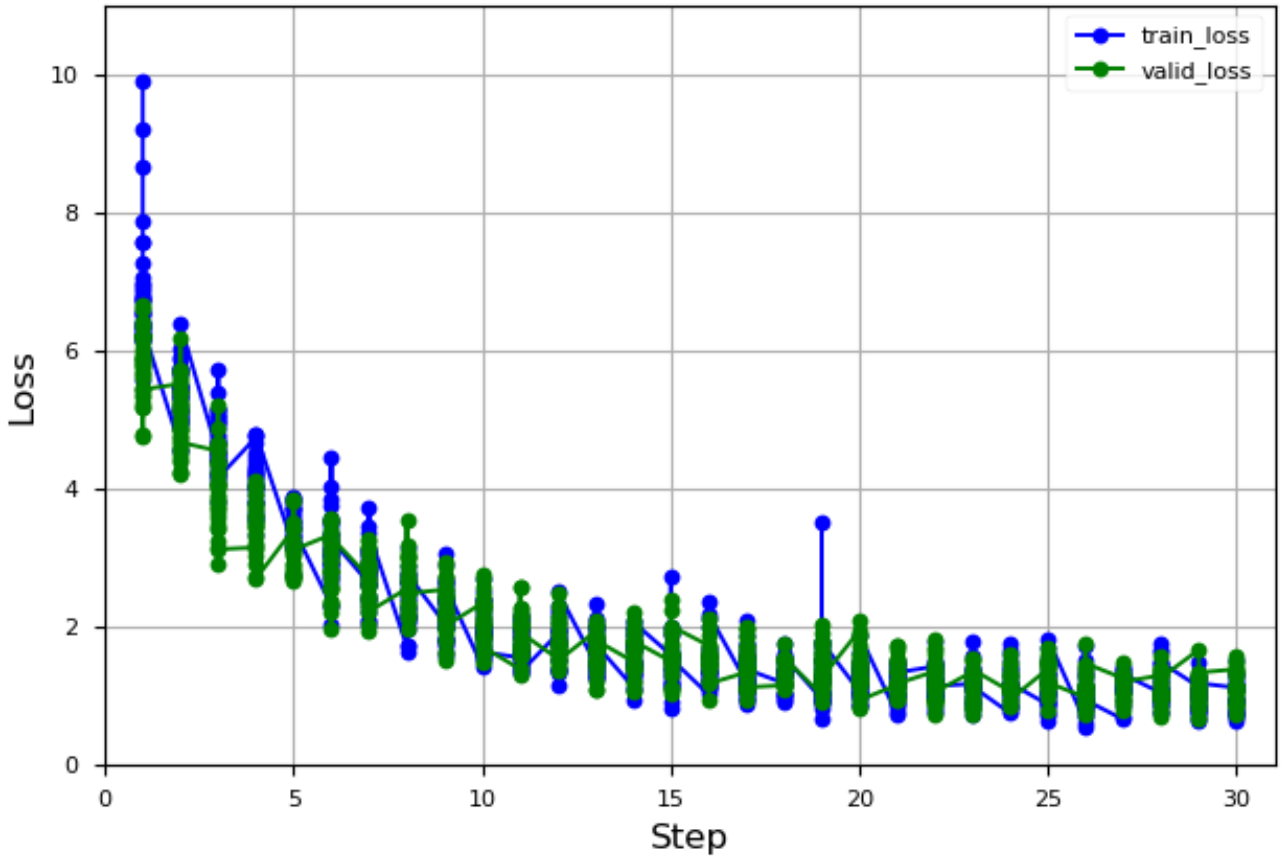


Figure B.5: Plot of loss values in training and validation sets. The x-axis depicts the step in epochs, and the y-axis is the loss value.

In Table B.1, we listed all results per epoch of our improved DMN model. First, we trained our model using the training set and then we validated it using the validation set.

| Epochs-number | Train Loss | Valid Loss | F1-score | EM-score |
|---|---|---|---|---|
| 1 | 6.79 | 6.15 | 32.02% | 22.7% |
| 2 | 5.41 | 5.31 | 42.80% | 32.83% |
| 3 | 4.69 | 4.64 | 51.60% | 41.64% |
| 4 | 4.13 | 4.02 | 60.12% | 49.57% |
| 5 | 3.87 | 3.46 | 64.48% | 55.17% |
| 6 | 3.42 | 3.04 | 69.83% | 60.10% |
| 7 | 2.68 | 2.76 | 73.20% | 63.67% |
| 8 | 2.42 | 2.49 | 75.23% | 66.11% |
| 9 | 2.18 | 2.21 | 78.10% | 69.71% |
| 10 | 1.97 | 2.09 | 78.79% | 70.30% |
| 11 | 1.84 | 1.94 | 80.14% | 71.94% |
| 12 | 1.72 | 1.84 | 79.98% | 72.27% |
| 13 | 1.65 | 1.68 | 81.86% | 74.70% |
| 14 | 1.75 | 1.66 | 82.96% | 75.92% |
| 15 | 1.48 | 1.58 | 82.93% | 75.95% |
| 16 | 1.44 | 1.54 | 83.21% | 76.22% |
| 17 | 1.39 | 1.42 | 84.36% | 77.07% |
| 18 | 1.31 | 1.39 | 84.15% | 77.74% |
| 19 | 1.26 | 1.35 | 84.55% | 77.97% |
| 20 | 1.24 | 1.31 | 85.19% | 78.70% |
| 21 | 1.21 | 1.29 | 85.42% | 80.19% |
| 22 | 1.18 | 1.25 | 84.58% | 78.41% |
| 23 | 1.16 | 1.19 | 85.75% | 79.50% |
| 24 | 1,16 | 1.17 | 85.64% | 79.33% |
| 25 | 1.14 | 1.18 | 85.80% | 79.64% |
| 26 | 1.07 | 1.15 | 86.02% | 79.85% |
| 27 | 1.05 | 1.14 | 86.66% | 80.93% |
| 28 | 1.04 | 1.10 | 86.65% | 80.80% |
| 29 | 1.03 | 1.08 | 86.47% | 80.71% |
| 30 | 1.02 | 1.10 | 86.64% | 80.81% |

Table B.1: Final results of our improved DMN evaluated on the validation set.

# Bibliography


Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: a system for large-scale machine learning. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, pages 265–283. USENIX Association in Savannah, GA, USA.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. pages 1360–1409. ICLR in San Diego, CA, USA.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. volume 3, pages 1137–1155. JMLR, Inc. and Microtome in USA.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. volume 5, pages 157–166. IEEE Press in Piscataway, NJ, USA.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. EMNLP Association in Doha, Qatar.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Cui, Y., Chen, Z., Wei, S., Wang, S., Liu, T., and Hu, G. (2016). Attention-over-attention neural networks for reading comprehension. *CoRR*, abs/1607.04423.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research, Inc. and Microtome in USA*, 12(Jul):2121–2159.

Elman, J. L. (1991). Distributed representations, simple recurrent networks and grammatical structure. *Machine learning - Connectionist approaches to language learning*, 7(2-3):195–225.

Finkel, J. R., Grenager, T., and Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics in Stroudsburg, PA, USA.

Floreano, D. and Mattiussi, C. (2008). *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press, ISBN: 0262062712, 9780262062718.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. volume 9, pages 1735–1780. MIT Press in Cambridge, MA, USA.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR in Upper Saddle River, NJ, USA, Upper Saddle River, NJ, USA, 1st edition.

Jurafsky, D. and Martin, J. H. (2016). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR in Upper Saddle River, NJ, USA, Upper Saddle River, NJ, USA, 3st edition.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Machine Learning, ICLR in San Diego, CA, USA*.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., and Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1378–1387, New York, New York, USA. PMLR in New York, New York, USA.

Matthew Richardson, Christopher J.C. Burges, E. R. (2013). Mctest: A challenge dataset for the open-domain machine comprehension of text. pages 193–203. Association for Computational Linguistics in Seattle, Washington, USA.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Olah, C. (2015). Understanding lstm networks. *http://colah.github.io/posts/2015-08-Understanding-LSTMs/*.

Pan, B., Li, H., Zhao, Z., Cao, B., Cai, D., and He, X. (2017). MEMEN: multi-layer embedding with memory networks for machine comprehension. *CoRR*, abs/1707.09098.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning in Atlanta, GA, USA*, pages 1310–1318. International Conference on Machine Learning in Atlanta, GA, USA.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543. EMNLP Association in Doha, Qatar.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*. ACL in Stroudsburg, PA, USA.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392. Association for Computational Linguistics in Austin, Texas.

Robbins, H. and Monro, S. (1985). A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer-Verlag New York, USA.

Seo, M. J., Kembhavi, A., Farhadi, A., and Hajishirzi, H. (2016). Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603.

Stamatatos, E., Fakotakis, N., and Kokkinakis, G. (1999). Automatic extraction of rules for sentence boundary disambiguation. In *Proceedings of the Workshop on Machine Learning in Human Language Technology*, pages 88–92. Springer-Verlag London, UK.

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448. Institute of Electrical and Electronics Engineers ( IEEE ) in Reston, Virginia, USA.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics in Stroudsburg, PA, USA.

Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700. NIPS in Montréal CANADA.

Wang, S. and Jiang, J. (2015). Learning natural language inference with lstm. In *Proceedings of NAACL-HLT*, pages 1442–1451. HLT-NAAC in San Diego, CA, USA.

Wang, S. and Jiang, J. (2016). Machine Comprehension Using Match-LSTM and Answer Pointer. pages 2–6. Singapore Management University in ICLR in New York, USA.

Wang, W., Yang, N., Wei, F., Chang, B., and Zhou, M. (2017). Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 189–198. Association for Computational Linguistics in Vancouver, Canada.

Weston, J., Bordes, A., Chopra, S., and Mikolov, T. (2015). Towards AI-Complete Question Answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698.

Weston, J., Chopra, S., and Bordes, A. (2014). Memory Networks. pages 1–7. Facebook AI Research in ICLR 2015 in New York, USA.

Xiong, C., Merity, S., and Socher, R. (2016a). Dynamic memory networks for visual and textual question answering. In *International conference on machine learning*, pages 2397–2406. JMLR.org in New York, NY, USA.

Xiong, C., Zhong, V., and Socher, R. (2016b). Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *Google Inc., USA 2New York University, USA*.

# Erklärung der Urheberschaft

Hiermit versichere ich an Eides statt, dass ich die vorliegende Masterthesis im Studiengang Intelligent Adaptive Systems selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

# Erklärung zur Veröffentlichung

Ich stimme der Einstellung der Masterthesis in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum Unterschrift