EXPERIMENTS IN NEURAL QUESTION ANSWERING

RAFAT-AL-ISLAM
Bachelor of Science, Islamic University of Technology, 2013

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

EXPERIMENTS IN NEURAL QUESTION ANSWERING

RAFAT-AL-ISLAM

Date of Defence: December 13, 2017

Dr. Y. Chali Supervisor	Professor	Ph.D.
Dr. H. Cheng Committee Member	Associate Professor	Ph.D.
Dr. J. Zhang Committee Member	Associate Professor	Ph.D.
Dr. H. Kharaghani Chair, Thesis Examination Committee	Professor	Ph.D.

Dedication

I dedicate this thesis to my Father for being there for me and to my Mother for pushing me to achieve success that I had only dreamt of.

Abstract

In this thesis, we apply deep learning methods to tackle the tasks of finding duplicate questions, learning to rank the answers of a Multiple Choice Question (MCQ) and classifying the answers to a question coming from the context of a paragraph. We draw our attention toward the problems related to sentence-sentence similarity. We used siamese architecture for better representation of the question and answers. The basic aim of all the methods proposed in this thesis is to build the word embeddings of question and answers and feed them to a deep neural architecture. We used several such architecture like Long-Short term memory (LSTM) and Convolutional Neural Network (CNN). We have also implemented an attention mechanism to put more focus on the sentence-word relationship. Our goal was to extract a refined representation of the question and answers through different combination of these deep learning techniques. We generated a representation of a sentence according to the context of another sentence for solving our tasks. We provide some simple but efficient deep learning models to solve our tasks. As neural models are data driven, we train our model extensively by making pairs such as question-question and question-answer over a large-scale real-life dataset. We used three different datasets to solve our three different tasks. The Quora dataset of several question answer pairs was used for the task of finding duplicate question. The OpenTriviaQA question answering dataset for the ranking of multiple answers. Lastly, we use the SQuAD dataset for the answer classification of reading comprehension task. We evaluate our models based on metrics like accuracy, precision, recall, F1 scores. Our methods and experiments demonstrate some significant improvements over the state-of-the-art methods.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my supervisor Prof. Yllias Chali for the continuous support, invaluable advice and encouragement. His guidance helped me to explore research challenges and thinking about scientific problems profoundly. The door to Prof. Chali's office was always open whenever I ran into a trouble spot or had a question about my research or writing. I am really grateful to him.

I would like to thank my M.Sc. supervisory committee members Dr. Howard Cheng and Dr. John Zhang for their time and effort spent on thesis committee meetings. They provided very valuable feedback and helpful suggestions to my research.

I also must thank the University of Lethbridge for the financial and travel support. I am also thankful to Natural Sciences and Engineering Research Council (NSERC) of Canada discovery grant for providing me a TITAN X GPU machine to conduct my experiments without which it would not have been possible for me to carry out the whole work.

I am forever thankful to Hoimonti Rozario, you should know that your support, friendship and encouragement was worth more than I can express on paper.

I am also thankful to all my close friends, you people made my surrounding cheerful and lively.

Last but not the least, I would like to thank my family: my parents for their continuous and unparalleled love, help and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

"Sentence similarity" or "Paraphrase Detection" is the process of detecting similar or duplicate sentences not only by their word similarity but also by the meaning of their words. A "Question-Answering" system is enabled to extract information from a question in order to answer that question.

The internet has taken over in every aspect of human life. The internet now provides all sorts of information that a human need in his day to day life. An adult human spends around 10 hours a day on average browsing through an unlimited number of web-pages. These web pages contain limitless amounts of information. An online forum is a kind of web page, where people can start a conversation on any topic and people following the forum can join the discussion expressing their opinions, concerns and questions about the topic. Recently online forums are gaining an immense popularity as these web pages are starting to provide more and more useful information. People can ask questions expecting people to answer those questions. As more people join the forum, it becomes overloaded with too much information. Users of these forums are overwhelmed by the number of questions and answers of that are relevant and that are not. It has now become an utmost necessity to customize this information for the users to access this huge amount of information according to their need. People are reluctant to search thoroughly for information in a web page or forum, thus leading to more unnecessary questions whose answers already exist in the forum. Moreover people also write unnecessary answers to a question making the list of

answers a lengthy one. These redundancies in information cost valuable time of people. These concerns motivated and made us interested to initiate developing an automatic similar question detecting system. Such systems are designed to detect similar questions by extracting semantic relationships. We also kept our interest on finding relevant answers of a question by ranking them according to their relevance. These systems can help people save time by retrieving similar questions (if they already exist in the forum) and provide useful information by bringing out good answers from a long list of existing ones. For example if a person asks, *"What do you mean by force in physics?"* in a forum while in the same forum another question was asked by another person like *"Can someone explain force in terms of physics?"*. A good paraphrase detection system can classify these two questions marking them as similar and restricting the user posting a redundant question by bringing the answers of the questions that are already there in the forum. On the other hand if there are hundreds of answers for the same question a good ranking system can bring out the best answers to the user saving a lot of time.

1.2 Contributions of this Thesis

- We implemented four deep neural models to detect similar question using deep neural architecture. The models try to correctly identify duplicate questions and mark them as duplicate or not duplicate.

- We implemented three models to establish an answer ranking system based on Multiple choice Questions. The systems rank the answers of a question and tries to predict correct answer of the question. We used different deep neural architecture for implementing the models.

- We designed a new task for reading comprehension based question answering. The aim of the task is to classify an answer of a question based on a paragraph as correct or incorrect. We implemented a system for solving the task using deep neural

architectures.

1.3 Overview of The Organization

This thesis is organized as follows. In chapter 2, we describe some research works related to the tasks in the field of "Community Question Answering". We briefly explain the deep learning techniques. We mainly used these techniques for solving our tasks. In chapter 3, we describe our approaches for detecting question-question similarity and paraphrase detection. We present our models, explain the task, the performance of our models and the results. Chapter 4, contains the description of answer ranking task for a multiple choice question. It also has the neural model descriptions for solving the task, the performance of the models and the results. In chapter 5, we introduce a new task of classifying answers of reading comprehension based question-answer system. In this chapter, we describe our model, the dataset, the experimental setup, the performance of the model and the results. Chapter 6, draws the conclusion and introduces some new ideas and directions for future research work.

Chapter 2

Background

2.1 Community Question Answering : Overview

Online Question answering forums are gaining more and more popularity every day. These forums are contributing a lot to the community. Popularity of online forums involves people in specific topics that are open for discussion. Adding new posts and continuously combining them with good replies make these forums more active online. Subject matters of these posts eventually cover everything imaginable, reaching out to a massive range of participants. The main topics are opened as forum board for the discussion called "Thread", where people can answer the questions. The question thread starts to grow up, as more people join the thread. Some answers may appear in the thread which are vague and far away from the context of the question. The answers can be totally unrelated to the question of that thread. Nowadays people are posting more and more questions or starting a thread without even looking into previous discussions which leads to a problem of having one or more similar questions as a different thread. So people are answering the same question time to time, while some of them are not answered at all. Now, what if it is possible to make these forums automated? Finding the similar questions or answers of those particular questions which already exist in the forum can save a lot of time and effort for the individuals and the people answering the questions.

We address this problem as "Community Question Answering (CQA)". For some time, Natural Language Processing has been a prime tool for finding a better-optimized solution. Among many problems that have been addressed recently by CQA, **Question-Question**

Duplication and **Question-Answering (QA)** are the popular ones. "Question-Question Duplication" is a type of problem where the solution tries to find similarity between a pair of questions. The solution of the "Question-Answering" problem tell us whether the answer is relevant to the given question. There are other problems to address such as **Learning to Rank the Answers**, these systems allow us to find a correct answer of a **Multiple Choice Questions (MCQ)** by ranking the multiple answers.

2.2 Deep Neural Network

Deep Neural Network algorithms are mainly based on the functionality of the human brain. It tries ti replicate the learning process of the human brain. It has been referred as artificial neural networks (ANNs) by Goodfellow et al. (2016). The most common form of ANN is feed-forward network. The first layer in a feed forward network is the usually the input and last layer is the output. In between these two layers there may be one or more hidden layers. Now if the network has more than one hidden layer, it is called a deep neural network. Bengio (2009) represented some functions with a deep, hierarchical network establishing that it can be be more efficient compared to a shallow (one hidden layer) model.

2.3 Word Embedding

This section will provide an overview of the concept called word embedding. The word embedding method was first introduced by Yoshua Bengio in his work "A Neural Probabilistic Language Model" (Bengio et al., 2003). In simple words, word embedding is a process of converting a sentence into a vector by mapping words into real numbers. This method is widely used in natural language processing systems such as question answering, text summarization, paraphrase detection.

2.3.1 One Hot Vectors

One hot vector is a popular method for representing the words in a vector space model. It can be described as a sequence of 0's and 1's per vector representing the absence or presence of that word. The size of the one hot vector is equal to the number of words in the vocabulary. Suppose we have a vocabulary of three words : {similar, duplicate, complex }. For creating hot vectors for this vocabulary, each word vector will have a dimension of 3 and a single 1 at word index location:

similar = [1 0 0]

duplicate = [0 1 0]

complex = [0 0 1]

Cosine similarity[1] measures the similarity between two non-zero vectors to find the cosine angle between them. However, it fails to capture a meaningful representation between two similar sentences when they are expressed using different words. The same thing happens when we measure cosine similarities for one hot vector representation of a sentence. For instance, if we consider these two following news headlines:

- Obama speaks to the media in Illinois
- The President greets the press in Chicago

There are no common words except for the stop-words such as *the*, *in* etc. which are often filtered out before processing the data in NLP because they doesn't provide much for understating the semantic information in between these two sentences. If these sentences are converted into one hot vectors, their cosine distance will be maximum suggesting these sentences are not similar. In order to calculate semantic similarity precisely, we need more information about the structure and meaning of the sentences. These information can be learned from large amounts of data using machine learning models (Kusner et al., 2015).

[1] https://en.wikipedia.org/wiki/Cosine_similarity

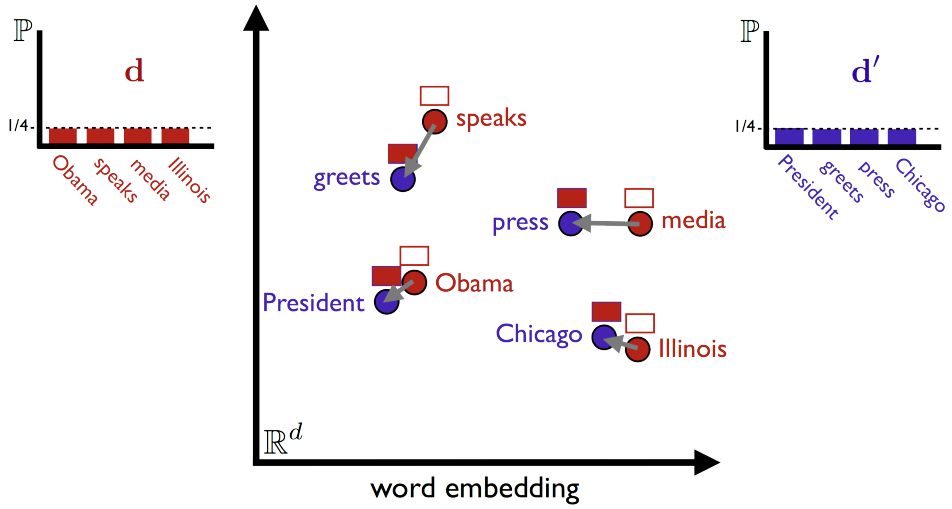

Figure 2.1: Visualization of word to word similarity of all non-stop words from both headlines are embedded into a word2vec space.

Figure 2.1 taken from (Kusner et al., 2015) visualizes the word to word similarity of the example headlines.

2.3.2 Word2Vec Embedding

Vector space models (VSMs) can represent sequence of words or documents over a distributed vector space. These models has been used since 1990 for computing continuous word representation, Latent Dirichlet Allocation (LDA) (Blei et al., 2003) and Latent Semantic Analysis (LSA) (Landauer et al., 1998). Bengio et al. (2003) introduced the concept of word embedding by developing a neural language model. This model is based on learning the joint probability $P(w_1,\ldots,w_T)$ of a sentence, where w_i is the i^{th} word in the sentence. The model assigns higher probabilities to the grammatical and meaningful sentences, and lower probabilities to meaningless sentence construction. For Example, when we are searching something in the internet form Fig 2.2[2], "Einstein won the nobel" the search engine would suggest the word "prize". The main reason behind this suggestion is that the probabilities of the word prize is higher among all other words and these probabil-

[2]Figure collected from http://book.paddlepaddle.org/04.word2vec/, this figure is under https://creativecommons.org/licenses/by-sa/4.0/

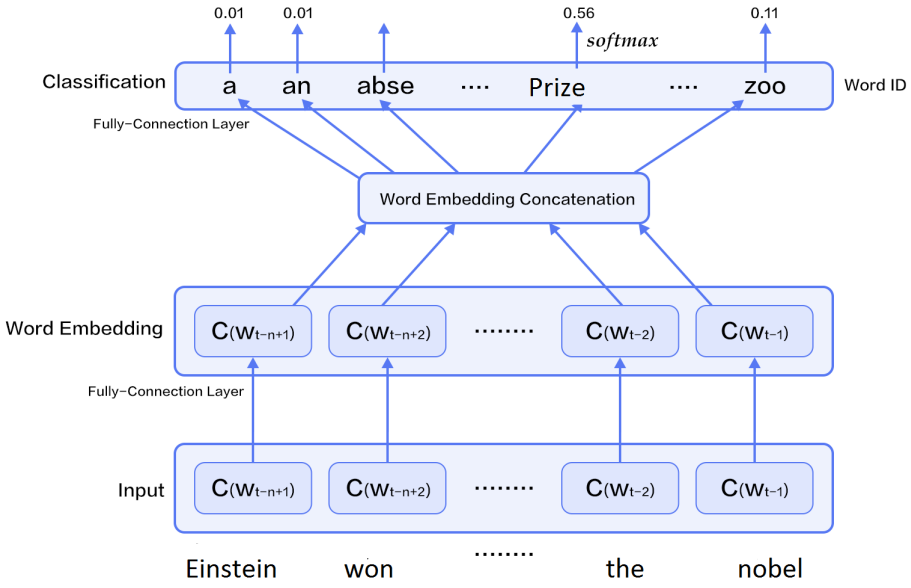

Figure 2.2: N-gram neural language model.

ities usually come from the language model that has been trained with all the words of the targeted vocabulary.

Word2vec (Mikolov et al., 2013a) is the most widely known method for learning word embeddings. It can learn word embeddings of a word from large number of texts to create high dimensional (50 to 300 dimensional) representations of that word in an unsupervised manner. The embeddings are formed in a distributed vector space where the words are placed according to their nearby points; that is the similar words are placed close together. We can say by changing a word with a synonym of that word does not change the meaning. For example “Define” and “Describe” will be placed closely, as they have the same meaning. This grouping of similar words helps us generalize a sentence and find similar types of sentences. Another feature of Word2vec is grouping of words of specific class or type (such as color, cities, countries etc.). This feature of word2vec helps us to find semantic relationships between words.

The example below is a demonstration of Figure 2.3 [3]

[3]Figure collected from (Mikolov et al., 2013a)

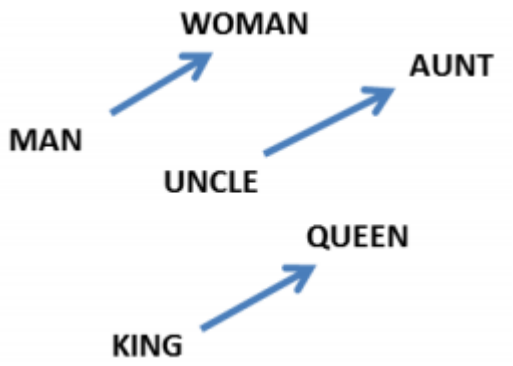

Figure 2.3: Word relationships from vector differences (Mikolov et al., 2013a).

$$W`(woman") - W("man") \simeq W("aunt") - W("uncle")$$

$$W("woman") - W("man") \simeq W("queen") - W("king")$$

Where W represents word Embedding. so according to the equations stated above [vector("woman") - vector("man")] results in a vector that is very close to [vector("aunt") - vector("uncle")]. Also [vector("woman") - vector("man")] results in a vector that is very close to [vector("queen") - vector("king")]. From these vectors we can infer that $woman - man + king = queen$

Further research in word2vec proved that more complex relationship can be extracted through word embeddings (Mikolov et al., 2013b) . Demonstration is shown in the Figure 2.4. Mikolov et al. (2013a) introduced two new architectures for learning word embeddings; Continuous bag-of-words model (CBOW) and Skip-gram model.

Continuous Bag-of-Words model (CBOW): While the previous language models could only predict a word based on the previous words of a sentence, CBOW model can predict and generate the current word based on N words in both before and after. The objective

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Figure 2.4: Complex word relationships pairs (Mikolov et al., 2013b)

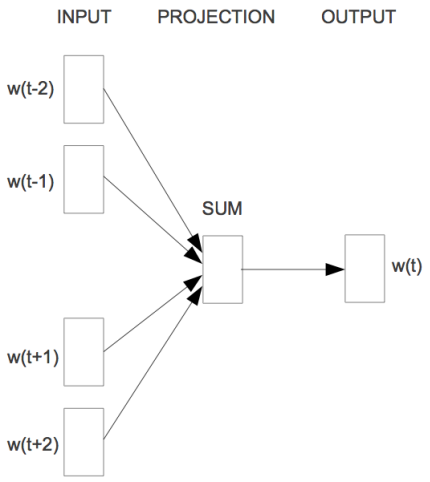

Figure 2.5: Continuous bag-of-words (Mikolov et al., 2013a)

function is given below:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} logP(w_t|w_{t-n}\ldots,w_{t-1},w_{t+1},\ldots w_{t+n})$$

Where J_θ is the objective function that gets n words of a sentence to around the target word to predict W_t at time step t. When n=2, the CBOW model is as shown in the Figure 2.5.

Skip-gram model: CBOW can be referred as a precognitive language model. CBOW model uses the surrounding words to predict the centre word, while skip-gram model uses the centre word $W(t)$ to predict its surrounding words. The objective functions is given

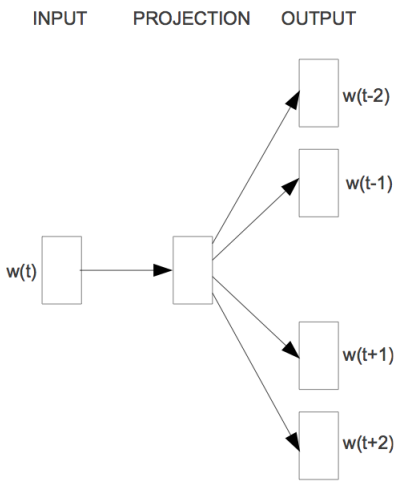

Figure 2.6: Skip-gram model (Mikolov et al., 2013a)

below:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \sum_{-n \leq j \leq n, \neq 0} logP(w_{t+j}|w_t)$$

Where J_θ is the objective function that gets W_t word and predict n words around it at time step t. When n=2, the model is as shown the Figure 2.6.

2.3.3 GloVe Embedding

In contrast to word2vec, GloVe[4] (Pennington et al., 2014) model takes advantage of the two well-known families of word vectors i.e., global matrix factorization methods (e.g., LSA (Landauer et al., 1998)) and local context window based methods (e.g. skip-gram (Mikolov et al., 2013a)). Word2Vec is based on probabilistic prediction, where GloVe can be referred as a "count-based" model. Count based models do dimensionality reduction for learning the vectors based on the co-occurrence counts of the word matrices. GloVe builds a co-occurrence matrix for the whole corpus first, then factorizes the matrix to yield a lower dimensions matrix to form word vectors and context vector. Word2vec model works with n-grams of words, trying to predict the n_{th} word from a given word vector.

[4]https://github.com/stanfordnlp/GloVe

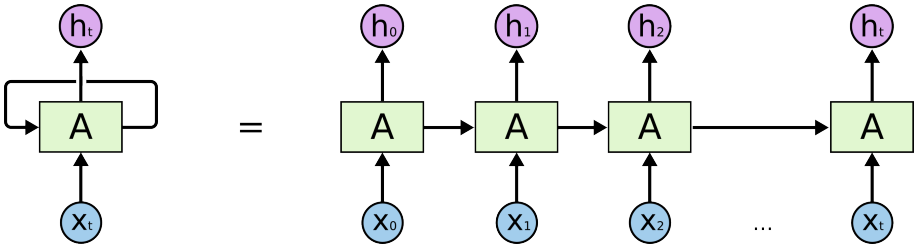

Figure 2.7: An unrolled recurrent neural network (Image is taken from the source: http://colah.github.io/posts/2015-08-Understanding-LSTMs)

2.4 Recurrent Neural Network (RNN)

For the case of a traditional neural network, we have to deal with the fact that all of the inputs and outputs are independent of each other. It has to start everything from new on each layer which is found to be a major shortcoming of a traditional neural network. Now for solving the tasks of Natural Language Processing (NLP), we need to know previous words in order to predict the next word of a sentence or find the context of a sentence. For doing so, each layer in the neural network must be dependent on the outcomes of its previous layer. This concept of using sequential information lead to the invention of Recurrent Neural Network (RNN). Recurrent neural network (RNN) has shown great success for solving the NLP tasks over the last few years. It helps in solving the classical NLP problems such as question answering, paraphrase detection and text generation.

As shown in Figure 2.7, it shows neural network A that takes a input x_t and produces the output of h_t. The loop as shown in figure allows to pass information to the next iteration. If we unfold an RNN at the t-th time step, we observe that it takes two inputs: the t-th input vector x_t (usually the embedded input word goes through an RNN as $e(x_t)$ at every time step) and the hidden state from the last time-step h_{t-1}. As it goes like a chain reaction all the previous hidden state helps in computing the current time-step h_t. This chain reaction continues until all the inputs of a sequence are processed. For the RNN function f, the function formulation is :

$$h_t = f(x_t, h_{t-1})$$

2.4.1 Long Short Term Memory (LSTM)

RNNs have an important property of connecting previous information to the present situation, acting like a chain of repeating modules of a neural network. We can just look into the present information for describing the previous information. If we consider a language model which predicts future words in a sentence based on its previous words, for example, "Einstein won Nobel ***prize***". For this case, we do not need to find the further context to predict the next word after nobel which is ***prize***. For this case the sentence length is small, all the relevant words that are required to understand the context of the sentence are close to each other. We did not have to go way back in the sentence to understand its meaning. On the other hand, there are some cases where the sentence is long and in order to find the relevant information the RNNs have to go way back to extract the necessary information, in such case it fails to learn the meaning of the sentence when the information gap grows large. For example "Bangladesh is an independent country, it got independence in the month of ***December***". Now to predict the last word of the given sentence RNNs must have the context of ***Bangladesh*** which is at the very beginning or far away from the last word. So it becomes hard for the RNNs in such cases to get hold of the context and learn necessary information[5].

This shortcoming of RNN to hold information for longer sequence is removed by a variation of RNN called Long Short-Term Memory networks in short **LSTM** (Hochreiter and Schmidhuber, 1997). It removes the long-distance dependency problem of RNN (Bengio et al., 1994). LSTM has proven to work extremely well in remembering information for longer sequences because of the explicit design for fulfilling this purpose. They also provide good results on various NLP related problems.

LSTM has the same repeated and chain-like structure like RNN. The main difference is the introduction of some gates inside each cell of LSTM. These gates perform special operations, which gives LSTM the ability to remember, forget or add information. So each LSTM cell has a memory cell c, an input gate i, a forget gate f and an output gate

[5]http://colah.github.io/posts/2015-08-Understanding-LSTMs/

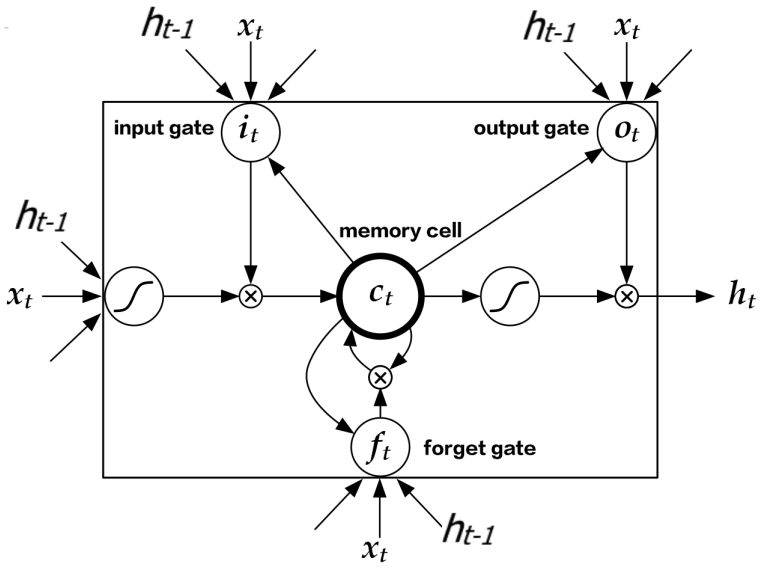

Figure 2.8: LSTM at time step t (Hochreiter and Schmidhuber, 1997)

o. These memory cells and gates provided LSTM the power to remove the "Long-term dependencies" problem. The LSTM function f is as follows

$$h_t = f(x_t, h_{t-1})$$

f contains following formulations from (Hochreiter and Schmidhuber, 1997),

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.1}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.2}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.3}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.4}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.5}$$

In the above equations, i_t, f_t, c_t, o_t stand for input gate, forget gate, memory cell and output gate respectively. W and b are model parameters, **tanh** is the hyperbolic tangent,

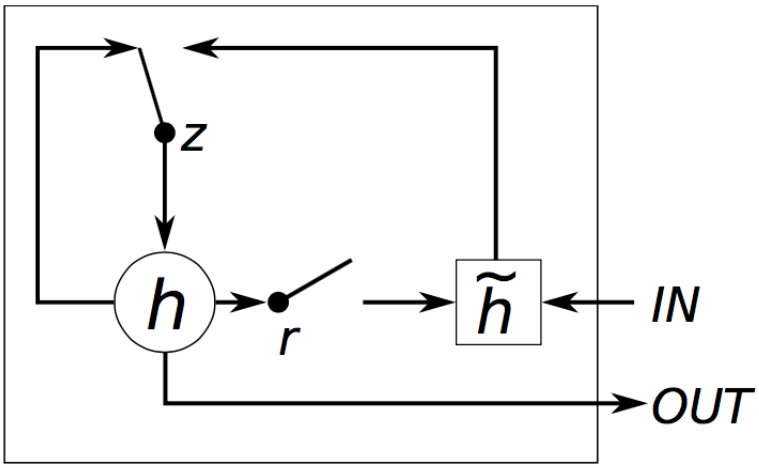

Figure 2.9: GRU Gating Mechanism (Chung et al., 2014)

and $\odot$ denotes an element-wise product operation as shown in Figure 2.8.

2.4.2 Gated Recurrent Unit (GRU)

GRU (Cho et al., 2014) is related to an LSTM it applies a different gating mechanism than the LSTM to prevent long-distance dependencies problem. This structure is relatively new, is less complex, trains faster, is computationally more efficient and performs better than a LSTM on less training data (Chung et al., 2014). GRU imitates LSTM for controlling the flow of information. GRU is structurally different than LSTM by combining forget gate and input gate into a single "update gate" and also by excluding the memory unit. GRU exposes the full hidden content without any control (Cho et al., 2014).

A GRU layer is quite similar to an LSTM layer, the following equations are for a single GRU layer (Cho et al., 2014):

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$

$$r = \sigma(x_t U^r + s_{t-1} W^r)$$

$$h = tanh(x_t U^h + (s_{t-1} \odot r) W^h)$$

$$s_t = (1-z) \odot h + z \odot s_{t-1}$$

In the above equations, a GRU has two gates, a reset gate r, and an update gate z. Intuitively, the reset gate determines how to combine the new input with the previous memory, and the update gate defines how much of the previous memory is kept as shown in Figure 2.9. For all recurrent units, the general formulation is,

$$h_t = Recurrent(x_t, h_{t-1})$$

where *Recurrent* is a unit which can be a simple RNN, GRU or LSTM.

2.5 Variations of Recurrent Neural Network (RNN)

2.5.1 Bi-directional RNNs

As we have already discussed about RNNs, they can summarize and control their previous inputs. However, they fail to predict further input tokens from the immediate next token. Nowadays the NLP tasks demand to predict more than just one word. Hence, the concept of sequential models that work in both direction to get the benefit of future encoded tokens as well as the history of previous states.

The bidirectional recurrent neural network (Bi-RNN/LSTM) (Schuster and Paliwal, 1997) became a very popular concept. This method was recently used in speech recognition (Graves et al., 2013) and in machine translation (Bahdanau et al., 2014). Bi-RNN can use any of the simple LSTM or GRU or RNN as the processing technique. Bi-RNN works in both forward and backward direction over the input sequence. The forward RNN or LSTM encodes the source sequence in its original order $(x_1, x_2, \ldots, x_T)$ from left-to-right to generate a sequence of hidden states $(\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_T})$. Encoding of the source sequence is done in reverse order from right to left $(x_T, x_{T-1}, \ldots, x_1)$ generating $(\overleftarrow{h_1}, \overleftarrow{h_2}, \ldots, \overleftarrow{h_T})$. Then combining both the outputs through concatenation we get a complete hidden state from the forward and backward RNN or LSTM, i.e., $h_i = \left[\overrightarrow{h_i^T}, \overleftarrow{h_i^T}\right]^T$. Thus the RNN or LSTM

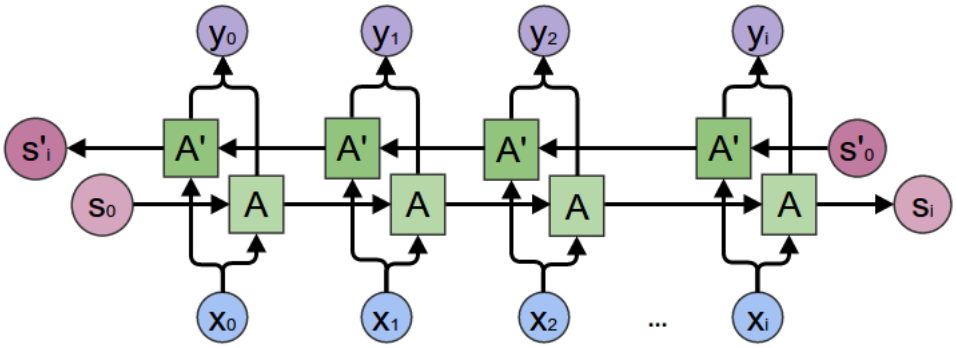

Figure 2.10: Bi-Directional Recurrent Neural Network (bi-RNN) (Image taken from the source http://colah.github.io/posts/2015-09-NN-Types-FP/)

layers at time-step t have the information of both previous and future words. Figure. 2.10[6] demonstrates the bidirectional recurrent neural networks

In Figure 2.10 S_0 and $S_0^{'}$ are the initial inputs for forward and backward RNN or LSTM network respectively, while S_i and $S_i^{'}$ are the outputs of the forward and backward RNN or LSTM network respectively passing over the sequence $(x_0, x_1, x_2, \ldots, x_i)$ and producing outputs $(y_0, y_1, y_2, \ldots, y_i)$ each time-step from each unit.

A much simpler demonstration of the Bi-RNN is provided in Figure 2.11.

2.5.2 Stacking multiple RNNs/LSTMs

Another variation of RNNs/LSTMs is stacking several RNNs or LSTMs or GRUs on top of each other. This architecture provides the model more power for processing. The linking between the stacked layers is straightforward as the output of the first layer acts as the input of the second layer and it goes on in a similar fashion until the final layer which becomes the desired output. For example, in a 3-layer stacked RNN, the calculation at time step t would look as follows:

$$h_{1,t} = \mathbf{LSTM_1}(x_t, h_{1,t-1})$$

$$h_{2,t} = \mathbf{LSTM_2}(h_{1,t}, h_{2,t-1})$$

[6]Image taken from the source http://colah.github.io/posts/2015-09-NN-Types-FP/

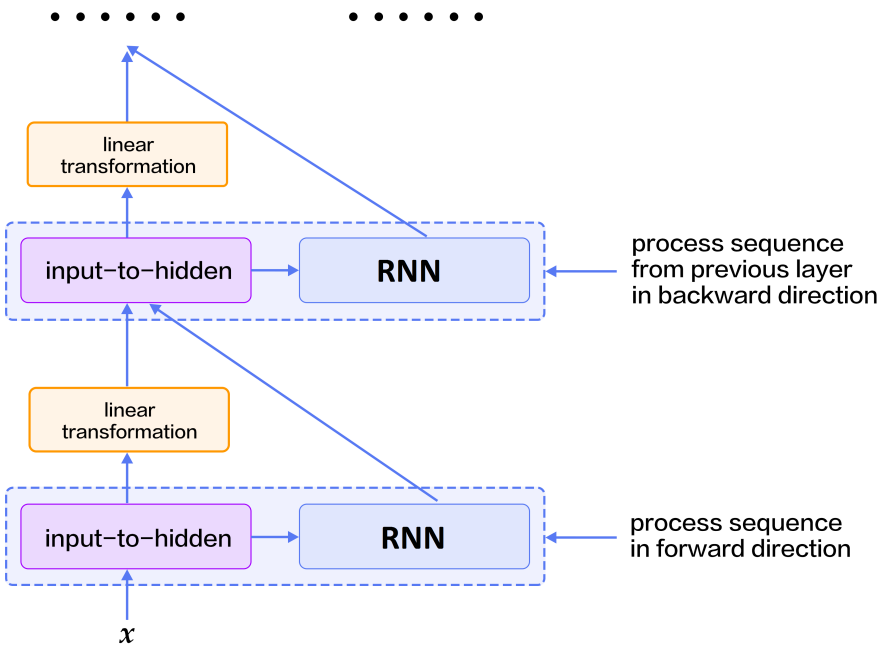

Figure 2.11: Bi-Directional Recurrent Neural Network (bi-RNN) (Schuster and Paliwal, 1997)

$$h_{3,t} = \mathbf{LSTM_3}(h_{2,t}, h_{3,t-1})$$

where $h_{n,t}$ is the hidden state for the nth layer at time step t of the RNN. Similarly, we could substitute LSTMs with RNNs, GRUs, or any other recurrent unit (Neubig, 2017).

Figure 2.12 shows an example of stacked RNNs.

2.6 Convolutional Neural Networks (CNN)

Over the past few years, Convolutional neural networks (CNN) has shown promising result on varieties of pattern recognition problems, i.e computer vision, natural language processing, voice recognition. CNN was notably first mentioned by LeCun et al. (1999). From then on-wards there were several modifications and several layers have been added to improve the performance of the CNN model.

A convolution [7] is an integral that expresses the amount of overlap of one function g

[7] http://mathworld.wolfram.com/Convolution.html

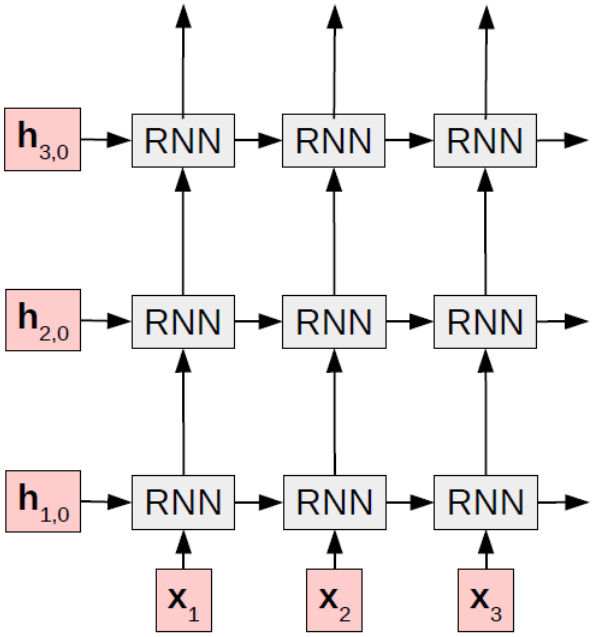

Figure 2.12: 3-layer stacked RNN (Neubig, 2017)

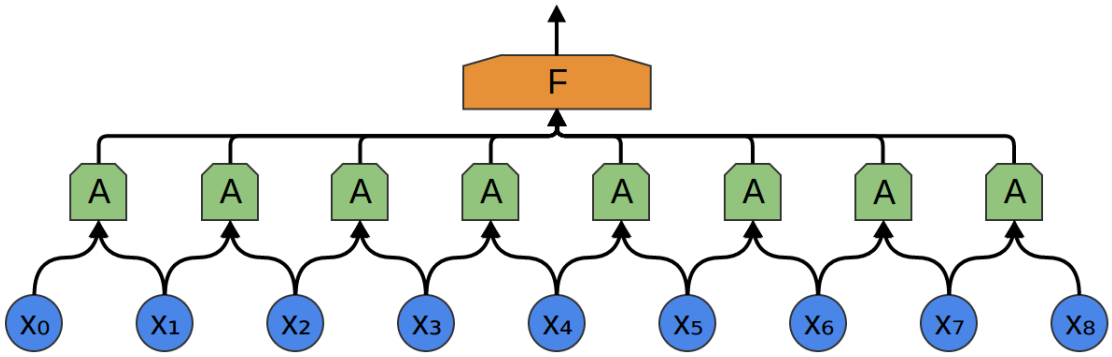

Figure 2.13: Convolutional Neural Networks (CNN) weight distribution (Image taken from the source http://colah.github.io/posts/2014-07-Conv-Nets-Modular/#fn1)

as it is shifted over another function f. In simple words, convolutions can be thought as a sliding window, which can pick or count necessary information at each window size. CNNs is formed of many layers of convolution with some activation function. In a fully connected neural network we connect each input neuron to each output neuron in the next layer. While CNN uses convolutions over the input in order to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer can apply different filters, and combine their results. Figure 2.13[8] explains the local input connections to the convolution layer and weight distribution of phrases.

In Figure 2.13 the sequence of inputs $(x_0, x_1, x_2, \ldots, x_8)$ and producing weights taking 2 word token as phrases. All the outcomes A are then combined together to produce a final outcome F.

2.6.1 Pooling

Nowadays often a pooling is added on top of a convolutional layer. A convolutional layer generates outputs of larger dimensions, so downsampling makes it easier to reduce the complexity. Pooling layer provides an output matrix of fixed size which is normally necessary for classification. Pooling layer also performs dimension reduction of the matrix keeping the necessary information by filtering. It filters the data following a stride of a given length. There are several types of pooling layers such as maxpooling, average pooling , L-2-norm pooling. Among these types of pooling, maxpooling is the most popular and widely used one. Maxpooling divides the outputs of a convolutional layer into sub region and pooling out maximum value of that sub region thus downsampling the output and also reducing the dimension. Figure 2.14[9] shows an simple example of maxpooling with a filter of $2 * 2$ and a stride of 2. It divides a matrix of $4 * 4$ to 4 sub regions. Then it picks the maximum value in the sub regions downsapmling it into a $2 * 2$ matix.

[8] Image taken from the source http://colah.github.io/posts/2014-07-Conv-Nets-Modular/#fn1

[9] Image taken from the source https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

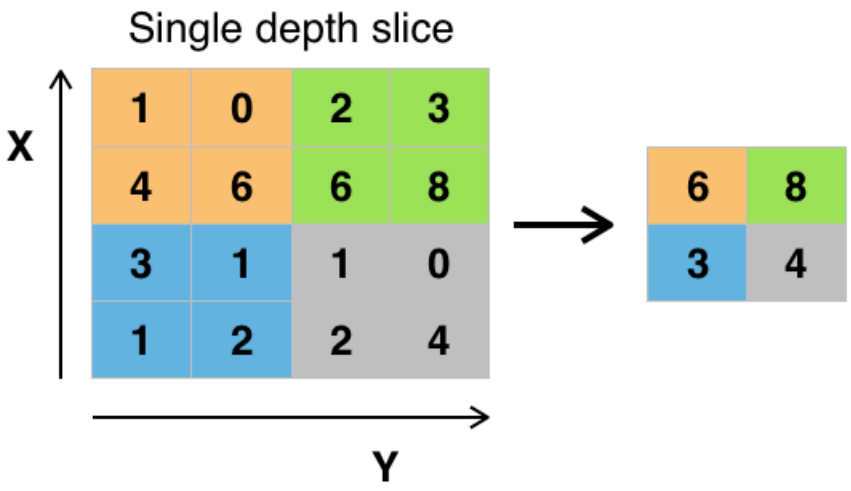

Figure 2.14: Maxpool with a $2 * 2$ filter and stride of 2

2.7 Encoder

Encoder computes a representation of a given sequence of word vectors. This representation can be termed as sentence vector. In this step the word vectors are passed as an input and it tries to represent the context for each of the token of the word vector into a matrix format. We can use RNN, LSTM, GRU, CNN architectures described in the earlier sections for encoding.

Any sequence of words encoded via RNN: This can be described mathematically as:

$$h_i = \varnothing_\theta (x_i, h_{i-1})$$

where, h_0 is a zero vector, $\varnothing_\theta$ is a non-linear activation function (e.g. sigmoid, ReLU, tanh), and $\mathbf{h} = \{h_1, \ldots, h_N\}$ is the sequential encoding of the first N words from the input source sequence. After the continuous vector x_N of the last word is read, the RNNs internal state h_n represents a summary of the whole source sentence.

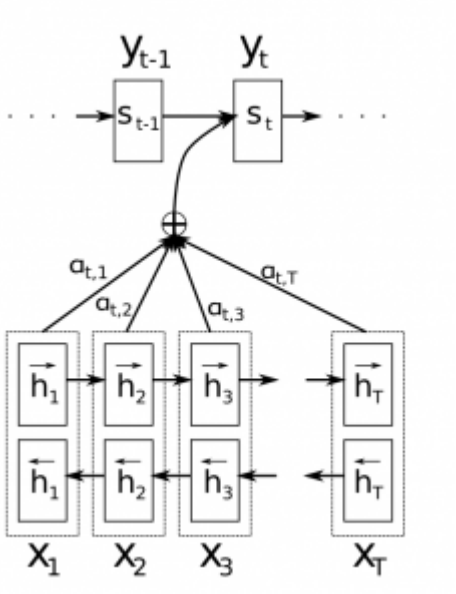

Figure 2.15: Attention Model (Bahdanau et al., 2014)

2.8 Attention Mechanism

In simple words, an attention mechanism provides a technique to pay attention on important information while removing focus from not so important informations. Formally, attention mechanism produces a single vector after an encoding step reducing the matrix representation. Attention mechanism captures the context of the vectors, keeping in mind that we do not lose information while reducing matrix dimensions. While encoding long sentences trying to extract syntactic and semantic information can lead to "vanishing gradient problem".

Vanishing gradient problem[10] arises during the training phase of a neural network that uses gradient descent techniques. We lose some information while processing longer sequence of words during longer training phase. The gradients for some tokens is too small which is hard to represent with numerical value. These tokens may hold important information about the meaning of the sentence. On the other hand, there may be some words that may not hold relevant meaningful information but has higher gradient value. So paying more attention to some part of the source sentences with such small values and put less

[10]https://www.quora.com/What-is-the-vanishing-gradient-problem

focus on unimportant words is beneficial to generate output at each step.

From the above discussion we get the idea that encoding all the information (syntactic and semantic) for a sentence with a fixed dimensional vector representation is unreasonable. Therefore, Bahdanau et al. (2014) proposed an attention mechanism in Neural Machine Translation (NMT) as shown Figure 2.15, that can decode various part of the context sequence and point out the difficulties faced during feature learning of long sentences. An attention mechanism allows us to "attend" different part of the sentence at each time step for generating output. Attention for x_{i+1} is computed as:

$$x_{i+1} = \phi_\theta (c_i, x_i)$$

At each time step, instead of using a fixed content which is the last state of the encoder, a unique context vector c_i is used to generate word y_i. The context vector c_i can also be referred to as the weighted sum of the hidden states of the LSTM which is (h_j). The strength of attentions is a weight denoted as a_{ij} for the i^{th} word in the target sentence to j^{th} word in the source sentence.

$$c_i = \sum_{j=1}^{N} a_{ij} h_j$$

$$a_i = [a_{i1}, a_{i2}, \ldots, a_{iN}]$$

$$a_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{N} exp(e_{ik})}$$

$$e_{ij} = align(x_i, h_j)$$

Where, $align$ calculates the fitness between the i-th word in the target language sentence and the j-th word in the source sentence. This is known as an alignment model.

Hard alignment is a traditional alignment model that focuses on one or more precisely chosen words of the target language, which categorically corresponds to each word of the

target sentence. In contrast to hard alignment, **soft alignment** is used for any word which has semantic or syntactic relation to any word of the target language. The strength of that relation or attention can be real numbers computed by the alignment model.

2.9 Siamese Neural Network

Siamese Neural Network is a special class of neural network that has two or more similar sub-networks that learn to differentiate between two inputs. Each of the sub-networks is a neural network that has identical configurations (i.e, parameters, weights etc). Siamese network is often called mirror networks because the sub-networks are mirror images of each other. Siamese network was introduced by Bromley et al. (1993) for solving the signature verification problem. The architecture learns the similarity information by learning the non-linear metrics. Though originally used for signature verification, this architecture performs really well in identifying identical images (Chopra et al., 2005). They used siamese architecture for face verification. Siamese Neural Network is also used for solving tasks in NLP like question answering, paraphrase detection, ranking the answers. Das et al. (2016) and Tan et al. (2015) used siamese architecture for similar question detection and answer selection respectively. Figure 2.16 shows an example for siamese network for question answering.

2.10 Softmax

Softmax (Memisevic et al., 2010) function can calculate the probabilities distribution of the event over n different events. This function will calculate the probabilities of each target class over all possible target classes. Later the calculated probabilities is used for determining the target class for the given inputs. The main advantage of using Softmax is the output probabilities range. The range is from 0 to 1, and the sum of all the probabilities will be equal to one. If the softmax function used for multi-classification model it returns the probabilities of each class and the target class will have the high probability.

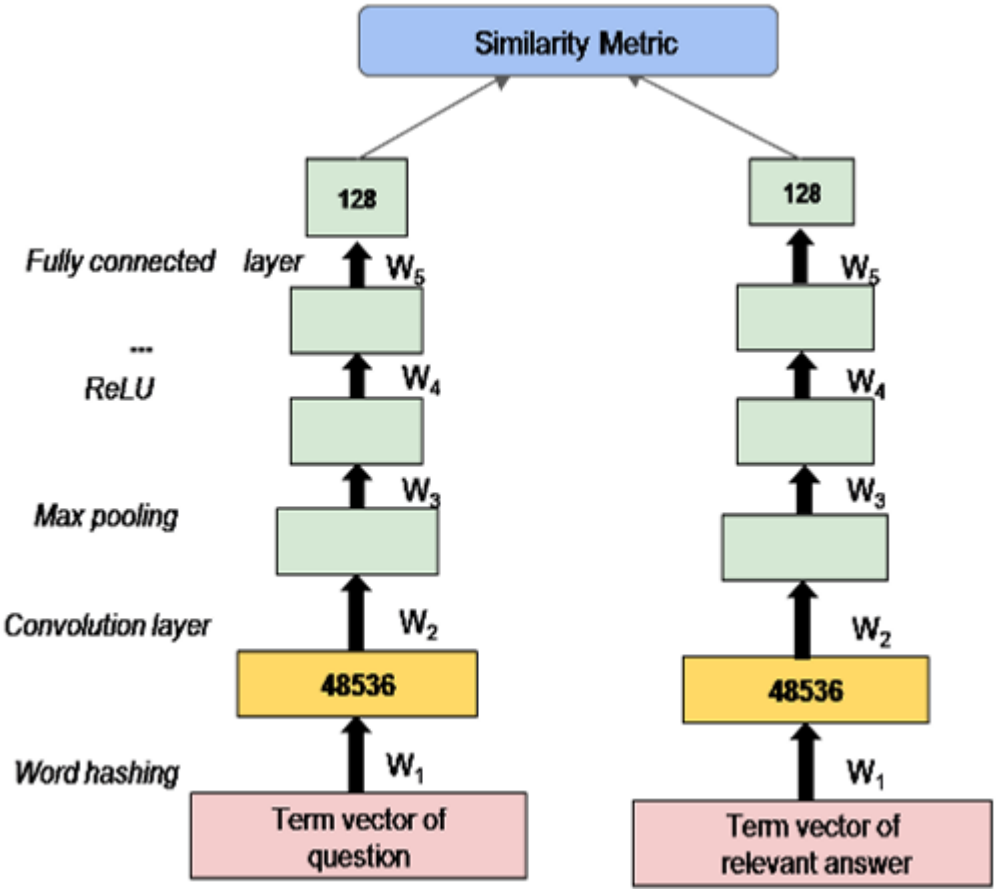

Figure 2.16: Siamese Network for question answering (Das et al., 2016)

The formula computes the exponential (e-power) of the given input value and the sum of exponential values of all the values in the inputs. Then the ratio of the exponential of the input value and the sum of exponential values is the output of the softmax function. The softmax equation is given below

$$F(X_i) = \frac{exp(X_i)}{\sum_{j=1}^{N} exp(X_j)}$$

Where $F(X_i)$ is the softmax function and X_i are the input parameters, N is the number of inputs.

2.11 Evaluation Metrics

Predictive models work on predefined guidelines producing specific and well targeted outputs. Evaluation metrics help us explain and measure the performance of the predictive models. We evaluate our results based on some widely used metrics.

2.11.1 Confusion Matrix

Confusion Matrix is build on the results of a set of test data of which the truth values or ground truth are known. It describes the performance of a classification model. Confusion matrix is also known as error matrix[11]. A binary classification model can have output of 1 (True) and 0 (false). So confusion matrix for a binary classifier can have four possible types of outcome based on the actual value which is the ground truth and the predicted value. For a binary classifier confusion matrix reports the following outcomes. Figure 2.17[12] shows confusion matrix for a binary classifier.

- **True Positives (TP)** denotes the class of results where both the actual and the predicted value is true (1).

[11] https://en.wikipedia.org/wiki/Confusion_matrix

[12] Image taken from the source:https://docs.wso2.com/download/attachments/47520050/Binary_Classification_Matrix_Definit

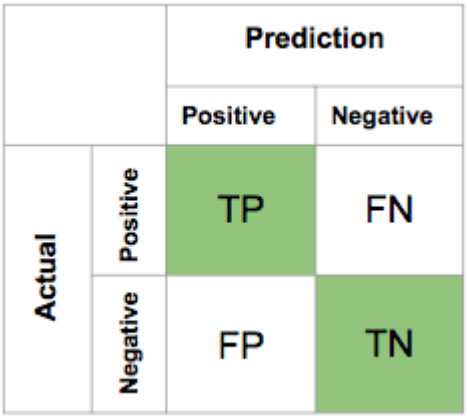

Figure 2.17: Confusion Matrix for a binary classifier

- **True Negatives (TN)** denotes the class of results where both the actual and the predicted value is false (0).

- **False Positives (FP)** denotes the class of results where the model has predicted True (1) but the ground truth is false (0). This is known as **Type I error**.

- **False Negatives (FN)** denotes the class of results where the model has predicted false (0) but the actual value is true (1). This is known as **Type II error**.

2.11.2 Metrics

We can generate some metrics based on the outcome of confusion matrix. These metrics actually describes how well the model has performed. We can evaluate our models based on these metrics.

- **Accuracy** describes the correctness of the model by calculating the ratio between total correct prediction and total number of prediction. It tells us how close is the predicted value compared to the actual value. The formula is as follows:

$$\frac{true positives + true negatives}{total\ number\ of\ predictions}$$

- **Precision** describes the correctness of predictions of a certain class (in this case pos-

itive class) by calculating the ratio between the true positive and total prediction positive values. It is also called the Positive Predictive Value (PPV).The formula is as follows:

$$\frac{true\ positives}{true\ positives + false\ positives}$$

- **Recall** calculates the proportion of actual number of true cases that were predicted correctly. The formula is as follows:

$$\frac{true\ positives}{true\ positives + false\ negatives}$$

- **F1 Score** establishes balance between precision and recall. Formula for calculating F1 score is given below:

$$\frac{2 * (precision\ *\ recall)}{(precision\ +\ recall)}$$

2.12 Related Works

2.12.1 Question-Question Duplication

Over the past few years, many question-question duplication or paraphrase detection methods have been developed. Most of the methods use the machine learning, graph-based or deep learning techniques.

The most recent work on paraphrase detection is based on the bilateral multi-perspective matching of the sentence vectors (Wang et al., 2017). In this paper, the authors proposed five paraphrase detection systems that detect paraphrases between two sentences by encoding the sentences into context representing sentence vectors . Deep neural networks are used as an encoder for encoding the sentences. These encoded vectors are then matched against each other to form the multi-perspectives. The first two systems mentioned in the paper are named “Multi-Perspective-CNN” and “Multi-Perspective-LSTM”. These models use CNN and bi-directional LSTM for representing the context of the sentences. Each of the context vector generated from the context representation layer is matched against all other context

vectors coming from same layer to form the perspectives. The two out coming vectors from the matching layer is merged together and sent to the prediction layer for detecting the paraphrases.

"Siamese-CNN" and **"Siamese-LSTM"** models mentioned by Wang et al. (2017) use convolutional neural network and uni-directional LSTM to form the encoder respectively by following the siamese architecture. They measure the cosine similarity between the vectors obtained form the encoder for detecting the paraphrases.

Another model mentioned in the same paper (Wang et al., 2017) is a re-implementation of the "L.D.C" model proposed by the (Wang et al., 2016b). In this model, two questions are individually passed through a bi-directional LSTM model to obtain two separate matching vectors. These two vectors are merged together into a single vector of fixed dimension. Then this result vector is used for detecting the paraphrases in the prediction layer over the probabilistic distribution of the output obtained from the context of the given input questions. These five models became our baseline models for the task of question-question duplication.

Clustering is a process of making a cluster of short text sentences based on the nearest centroid representation. Wang et al. (2016a) proposed a system that implemented clustering method and concatenation of vectors using deep neural architecture for detecting similar sentences. The clustering process is implemented using k-means algorithm. The cluster is formed after getting the representations from the current neural network to find sentence similarity. Their clustering process works well over short text and factoid questions.

Attention mechanism is the process of providing an extra attention over a specific word or phrase. Recently this mechanism was used in many natural language processing tasks to obtain better results specially in the tasks of finding text similarity. This mechanism helps to capture the meaning of longer sentences. Bahdanau et al. (2014) introduced an attention mechanism based on the soft alignment of the tokens. Alignment is the process of computing gradient value of corresponding matching tokens between two sentences. In

order to provide importance or attention to words, the model assigns a probabilistic value or weight to the target word depending on the previous and future word. Attention mechanism removed the complexities of remembering all the information of a long source sentences.

Parikh et al. (2016) proposed a method that uses token alignment between the source sentences for finding sentence similarity. The authors followed three steps; attend, compare and aggregate to finally predict the sentence similarity. The input representation of the sentences is passed through an attention layer. Authors proposed a variation of neural attention model initially implemented by Bahdanau et al. (2014). The sentence representation vectors are broken into vectors of sub-phrases after the tokens of these two sentences are soft aligned according to the attention model. These sub-phrases vectors with attention are compared with sub-phrases vectors produced without attention. Lastly, an aggregation layer is used to combine the set of vectors with and without attention to predict the similarity.

Another variation of attention mechanism is based on hierarchical structure. In the hierarchial structure a document is organized and ranked according to their hierarchy. The ranking starts from word level followed by sentence level and ending at document level. Yang et al. (2016) introduced an attention mechanism implementing word level and sentence level attention following hierarchical structure. The main intuition behind this hierarchical structure is to capture important information over the whole document as the documents hold more information. This attention mechanism works well for document classification.

Rocktäschel et al. (2015) provided a reasoning based model to find the semantic similarity between two sentences. Inference techniques are used to achieve the semantic similarity. This method can also be described as a word by word matching technique providing attention over the important words.

2.12.2 Question Answering (QA)

Systems that are enabled to seek similar answers, verify the correctness of the answer of a question and also find related answers for a given question can be termed as question-

answering systems. Since online forums are growing at an exponential rate, it is tough to moderate all these forums manually. A lot of questions are being posted every day which need to be answered. Along with the questions, a large number of answers are also being posted without moderation which are sometimes vague and loosely connected to the question. Some notable works related to question-answering systems are mentioned in this section.

Bian et al. (2008) described that common people use complex questions to find answers instead of using the normal factoid based question. There are some translation based models like **TransLM** (Xue et al., 2008) which can find a best answer from a list of answers. The best answer is obtained by going through all the answers of that question and selecting one answer that produces the maximum probability for generating that question. Though mainly applied for question generation, this model can also be applied for answer generation too. The score is determined by calculating the average of the two conditional probabilities. Machine translation is being used for the smooth generation of words. The probability of generating the best answer is

$$a^* = argmax_{a \in A}\, P_{TransLM}(q|a)$$

Where q and a represents the questions and answers respectively. a^* is maximum score of the joint probability.

Recently word embedding based methods got the popularity that involve the computation of real valued embedding of questions and answers. The embedding can be the average or the sum of the word vectors according to Mikolov et al. (2013b). Word embedding techniques are described in the later section of this chapter.

Das et al. (2016) proposed a model named "Siamese Community Question Answering" for finding the similar sentences. The model is based on siamese convolutional neural network structure that can learn similarities between the question-answer pairs. Authors

have worked with labeled data, passing it into a convolutional layer along with max pooling. Max-pooling layer with small non-overlapping grids finds the important local features form a fixed length vector. A non-linear activation function Rectified Linear Unit (ReLU) is used as a preferred backdrop. The semantic distances between the relevant and irrelevant answers are measured using the cosine similarity. Their loss function is as follows:

$$L = \sum_{(Q_i,\, A_i \,\in\, C \,\cup\, C')} loss(Q_i, A_i)$$

where C represents the relevant question-answer pairs and C' represents the irrelevant question-answer pairs. The function $loss(Q_i, A_i)$ determines the semenatic difference between the questions and answers Q_i, A_i respectively. Stochastic Gradient Descent (SGD) was applied to update the network sub-parameters.

Recently some models have been developed for predicting an answer of a question from a reading comprehension. These systems use a paragraph and the questions from the context of the paragraph as input. The task is to find the answer for the given question or to find the correctness of the answer. Models like "Multi-layer Embedding with Memory Network(MEMEN)" (Pan et al., 2017) enables a machine for reading comprehension. Authors used skip-gram model to obtain the syntactic and semantic information of the sentences. Memory function of the model is provided using the bi-directional LSTM. For scoring, "Query-Based Similarity" and "Context-Based Similarity" is used along with the softmax classifier and the Gated Recurrent Unit(GRU) as loss function .

Nowadays question-answering models can handle both complex questions as well as the factoid questions. Factoid questions can be answered with a yes or no type answer. Tan et al. (2015) proposed some deep learning models for non-factoid answer selection following the siamese architecture for question and answers. The models are formed by the combination of long short-term memory, convolutional neural network along with pooling layer and attention mechanism. Cosine similarity is measured between the question and the

answers to find best possible answer. Their models can capture long-range dependencies of the sentences as authors used attention mechanism on top of the LSTM layer. The success of these models led many researchers to work with the combination of siamese LSTM and CNN architecture.

2.12.3 Question Answer Ranking

As we have discussed earlier, with the increase of answers in the online forums, the users have to spend a good amount of time to find some relevant answers. For finding relevant answers among numerous posted answers has itself become a task. The users would like to search for the most voted or liked answers in the question-answer thread. Often the most voted answer is not relevant or not answering the question properly as people sometimes like to vote a funny or sarcastic comment which is unrelated to the question. So ranking or sorting relevant answers among many answers according to the relevancy of a given question has become important for saving time and effort.

A global thread level inference model that can classify or rank the answers was introduced by (Joty et al., 2015). Two classifiers named as "Good vs Bad" and "Same vs Different" are used for classifying the comments. "Good vs Bad" classifier obtains the output of another classifier (Same vs Different) in order to predict whether the comment is good or bad. Each of the comment is passed through these classifiers to obtain separate scores. Then two inference techniques "Graph Partition Approach" and "Integer Linear Programming Approach" are used to optimize the result in order to provide a ranked relevant answer list.

Tymoshenko et al. (2016) implemented a ranking method for non-factoid answers using machine learning techniques such as convolution tree kernels (CTKs) and convolutional neural networks (CNNs). CTK method is based on tagging the parts of speech (POS) of a sentence starting from the root level forming a tree like structure. With the help of these technique the authors were able to form a relationship between the questions and

the answers. Tree blocks with the same POS tag are filtered out for finding patterns and similarity between the sentences. They also implemented a neural network model using convolutional layer with max-pooling for predicting the sentence similarity. In the paper, they provided comparisons for both the approaches for finding sentence similarity.

A word embedding based CNN model (Feng et al., 2015) generates a question and answer vector embeddings using a CNN network following a siamese architecture. Word embeddings are learned from pre-trained Word2Vec model. A convolutional layer is used followed by a 1-max pooling layer. The model was trained with a loss function known as max-margin which can be described as follows:

$$Loss = \sum_{(Q,A)\in Q_n} \sum_{(Q,A')Q'_n} max(0, \gamma - s(Q,A) + s(Q,A'))$$

Where Q represents the question and A represents the answers. Q_n is the set of questions with correct answers and Q'_n is set of questions with incorrect answers. γ is the margin threshold and s is a scoring function.

Severyn and Moschitti (2015) proposed a model that can learn an optimal representation of the text pairs for ranking sentences with short text. CNN is used for extracting patterns. Activation unit is applied that allow the system to learn non-linear decision boundaries with each convolutional layer. Rectified linear unit (ReLU) function as the activation function which can be defined simply as $max(0,x)$. This activation function ensures that the outcome is always positive. This activation function affects the convergence rate and the quality of the solution. A pooling layer is used to reduce the dimension of the representation obtained from the convolutional layer. Answer classification is done by a softmax classifier to produce an optimized list of relevant answers.

2.13 Summary

In this chapter, we presented the necessary background information and recent related researches in community question answering. All the necessary research terms such as

word embedding, Recurrent Neural Network (RNN), Encoder framework, Attention mechanism, Siamese network is properly described with a good understanding. Our proposed models are heavily dependent on these concepts. The terms explained in this chapter is from the computational linguists perspective. From the next chapters, we will start introducing our proposed task models.

Chapter 3

Question-Question Duplication

3.1 Introduction

In this chapter, we aim to develop a system that can identify similar questions and mark them as duplicate or not-duplicate. We implemented some models using deep neural architecture that allows us to differentiate between a pair of questions. We conducted our experiments on Quora Dataset of Question Pairs released in 2016. Our methods demonstrate a better performance in terms of accuracy, precision, recall and F1 scores over the current methods and models.

3.2 Model Approaches

This section describes our four models that we have implemented for finding duplicate questions. Our first model uses only long short-term memory for solving the task. After that we introduce a model using an attention (ATTN) mechanism with long-short term memory. Our third model is formed of Bi-directional long short-term Memory (LSTM) with an attention mechanism. Our fourth model is a combination of Bi-directional long short-term memory (LSTM) with attention and convolutional neural network (CNN). All our models are using similar siamese network structure (Mueller and Thyagarajan, 2016) and inspired by the models proposed by (Tan et al., 2015) and (Weston et al., 2015)

3.2.1 Question-Question LSTM

We implemented a simple LSTM based model in Figure 3.1 that produces distributed weight representations for question 1 and question 2 independently. We use separate encoders for processing the questions following a siamese architecture. The encoder in this case is a simple uni-directional LSTM that learns the hidden annotations h_t at time t as processing the input from left to right is as follows:

$$h_t = f(x_t, h_{t-1})$$

Where, the $h_t \in \mathbb{R}^n$ encoding all of the contents seen so far at time t which is computed from h_{t-1} and x_t, where $x_t \in \mathbb{R}^m$ is the m-dimensional embedding of the current word x_t.

It generates an output h_t when t^{th} unit along with the previous output h_{t-1} are passed to the current input x_t. When calculating h_t it uses i_t, f_t, c_t, o_t input gate, forget gate, memory cell and output gate respectively to achieve a good performance over longer sequences. The forward propagation of LSTM is as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{3.1}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{3.2}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{3.3}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{3.4}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.5}$$

In the above equations, i_t, f_t, c_t, o_t stand for input gate, forget gate, memory cell and output gate respectively. W_x, W_c ,W_f, $W \in \mathbb{R}^{n \times m}$ are weight matrices, n is the number of hidden units, $\sigma()$ is the sigmoid function, **tanh** is the hyperbolic tangent and $\odot$ denotes an element-wise multiplication.

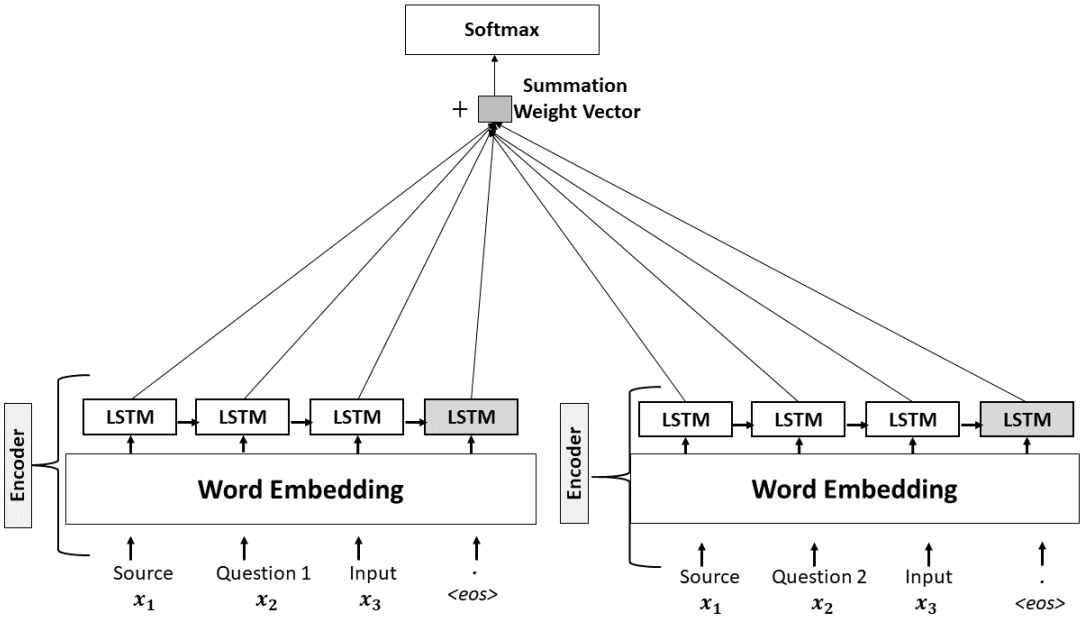

Figure 3.1: Long Short Term Memory (LSTM) model

As shown in figure 3.1 LSTM processes the word embeddings generated from the questions. The LSTMs from both the encoders produces separate vector outcomes. These two vectors are then merged together to form a final representation vector. This final representation vector is passed through a **softmax** classifier to classify between the duplicate and not-duplicate questions. **Categorical crossentropy** has been used as the loss function. The following equation calculates the loss function between the predicted value and actual value which is the ground truth.

$$H(p,q) = -\sum_{x} p(x)\log(q(x))$$

Where $p(x)$ denotes the actual value and $q(x)$ denotes the predicted value of the input pair of question in either 0 or 1 and x is the number of outcomes or predictions.

3.2.2 Question-Question LSTM with Attention

In this section we improved our the model described in section 3.2.1 by adding an attention layer. This model imitates the same siamese structure described in the earlier

model with the exception of attention layer on top of the LSTM layer. The encoder learns the hidden representations in the same way as described in the section 3.2.1. We now pass these two output vectors coming form the encoder in to an attention mechanism layer.

Attention layer adds an extra weight to the vectors. At each time step t, a context vector c_t is generated by the LSTM hidden state (h_i) from the encoder. The weight α_{ij} is the strength of the attention i^{th} word in the target vector to the j^{th} word in the source vector. The general idea behind the attention vector is that it tells us how much focus should be given to a particular source word at a particular time step of the encoder. Larger the value of α_{ij}, more focus will be given to a particular word while forming the next vector representation.

$$c_t = \sum_{i=1}^{N} a_{ij}\, h_t$$

$$a_{ij} = \frac{exp(h_t^{Q_1}.\, h_i^{Q_2})}{\sum_i exp(h_t^{Q_1}.\, h_i^{Q_2})}$$

Where, $h_t^{Q_1}$ and $h_t^{Q_2}$ are the LSTMs hidden states at each time step t coming from the question1 encoder and question2 encoder respectively. We use the (.) *dot* attention mechanism of (Luong et al., 2015) due to its efficiency and which is simple to implement. The dot attention mechanism is actually the dot product between two hidden vectors.

Then we merge the encoder outcome of question 1 with the outcome of the attention layer to form the final representation. We use the same softmax classifier used in the previous model to mark the questions as duplicate or not-duplicate. We tried to minimize the loss function using the categorical crossentropy loss function.

3.2.3 Question-Question Bi-directional LSTM-Stacking with Attention

In this model we used bi-directional RNN (Bi-RNN) (Graves et al., 2013) as encoder for learning the hidden states but for our case we replaced the RNN with a LSTM makiing it a bi-directional LSTM (Bi-LSTM). Simple uni-directional LSTM that learns the hidden

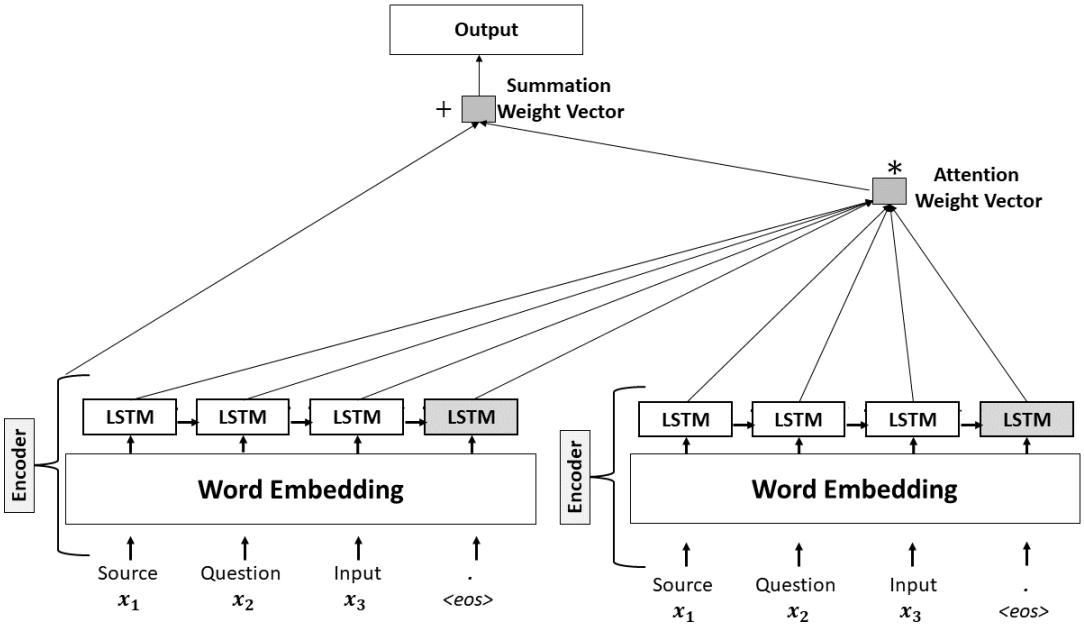

Figure 3.2: Question-Question LSTM with Attention model

annotations h_t at time t as processing the input from left to right is as follows:

$$h_t = f(x_t, h_{t-1})$$

Where, $h_t \in \mathbb{R}^n$ encodes all contents seen so far at time t which is computed from h_{t-1} and x_t, where $x_t \in \mathbb{R}^m$ is the m-dimensional embedding of the current word x_t.

Now we implemented Bi-LSTM that processes each questions word embedding representations in both forward and backward direction calculating the hidden annotations h_t at each time step t. For each position t, we merge the hidden states by concatenation to achieve the final hidden state:

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t}$$

Where operator $\oplus$ indicates concatenation and $\overrightarrow{h_t}$, $\overleftarrow{h_t}$ represents the forward and backward representation respectively. Forward representation $\overrightarrow{h_t}$ is calculated from $\overrightarrow{h_t} = LSTM(x_t, \overrightarrow{h_{t-1}})$ and backward representation $\overleftarrow{h_t}$ is calculated from $\overleftarrow{h_t} = LSTM(x_t, \overleftarrow{h_{t-1}})$.

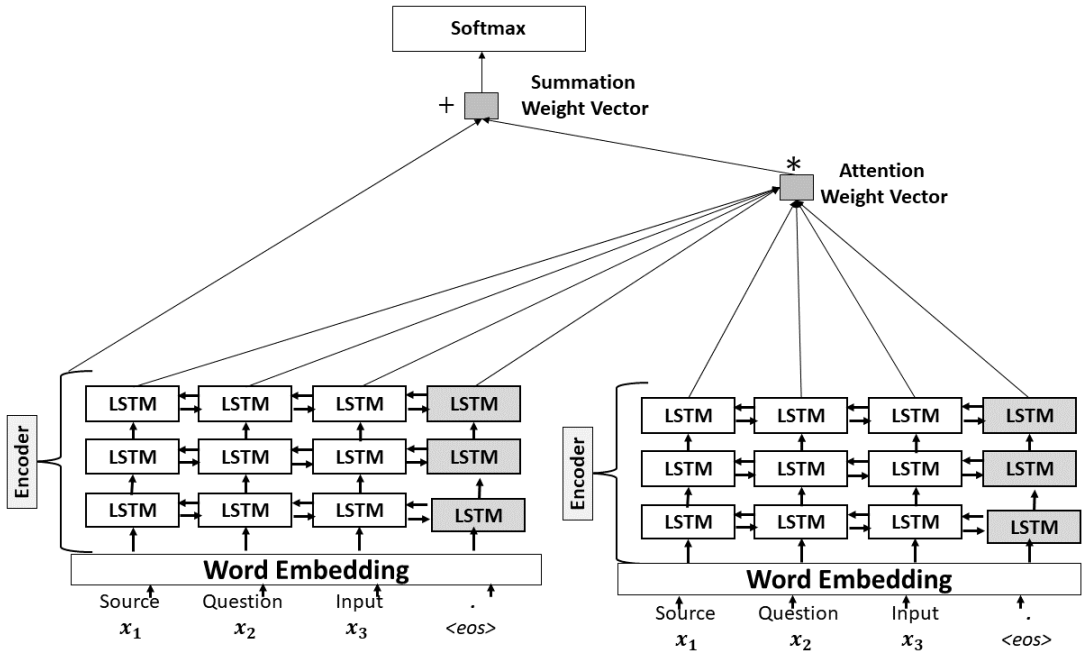

Figure 3.3: Question-Question Bi-directional LSTM-Stacking with Attention

In this model we implemented a 3-layer Bi-LSTM following (Luong et al., 2015) by stacking 3 Bi-LSTM layers on top of each other. A 3-layer stacked LSTM, the calculation at time step t would be as follows:

$$h_{1,t} = \mathbf{LSTM_1}(x_t,\ h_{1,t-1})$$

$$h_{2,t} = \mathbf{LSTM_2}(h_{1,t},\ h_{2,t-1})$$

$$h_{3,t} = \mathbf{LSTM_3}(h_{2,t},\ h_{3,t-1})$$

Then we passed the encoder outputs to an attention layer (same attention layer mechanism described in the previous model in section 3.2.2) to obtain an attention representation vector. We also follow the same merging technique of the previous model to merge the outcome vector of question 1 encoder with the attention representation vector to form the final representation. We use softmax classifier for identifying the classes between duplicate and not-duplicate. In this model we use categorical crossentropy as the loss function. Figure 3.3 shows the model structure.

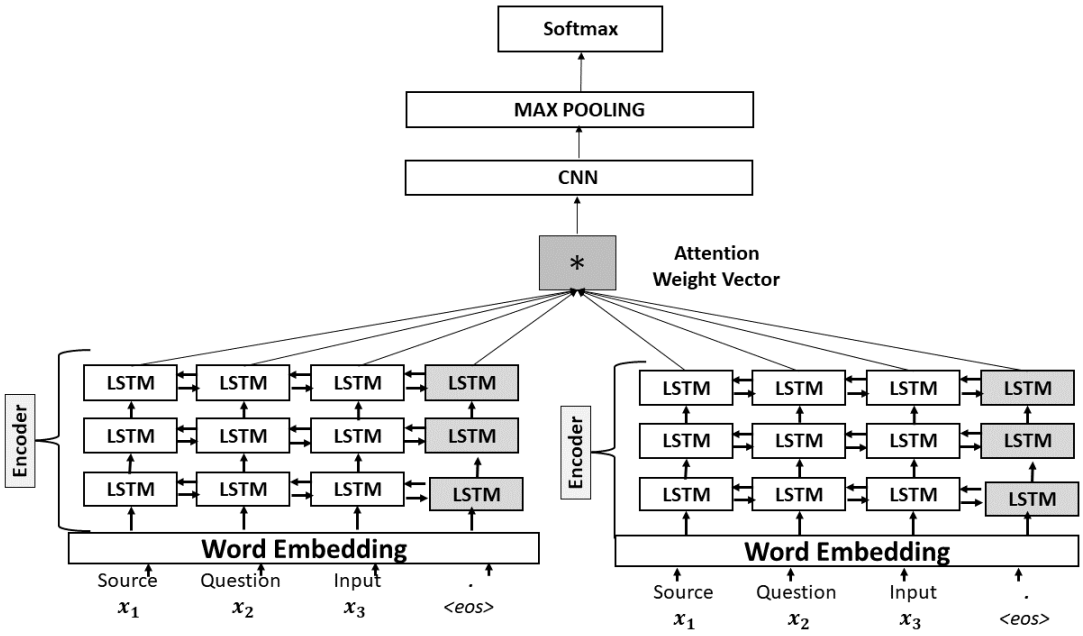

Figure 3.4: Question-Question Bi-directional LSTM/attention/CNN with Maxpooling

3.2.4 Question-Question Bi-Directional LSTM-ATTN-CNN with maxpooling

In this model we use the same bi-directional LSTM (Bi-LSTM) described in section 3.2.3 with an attention mechanism described in section 3.2.1. We follow siamese architecture for implementing the encoder for the questions. Then we pass the attention representation vector to a convolution neural network (CNN) layer. We applied a one dimensional convolution layer with a filter size of 32 along with a stride size of 1. On top of the CNN layer, we use a max-pooling layer of 2*2 to reduce the matrix dimension generated from the convolution layer. Max-pooling layer is used to extract the crucial local features to form a fixed length feature vectors. Then we use a softmax classifier to classify the questions into duplicate or not-duplicate class. Categorical crossentropy loss function is being used to minimize the loss function. Figure 3.5 shows the model structure.

id	qid1	qid2	question1	question2	is_duplicate
61	123	124	Is it normal to have a dark ring around the iris of my eye?	What causes a dark ring around the iris? How should	0
62	125	126	How is the new Harry Potter book 'Harry Potter and the Cu	How bad is the new book by J.K Rowling?	1
63	127	128	Why do I always get depressed?	Why do I always get depressed in the evening?	0
64	129	130	Where can I find a European family office database?	Where do I find a U.S. family office database?	0
65	131	132	What is Java programming? How To Learn Java Programmi	How do I learn a computer language like java?	1
66	133	134	What is the best book ever made?	What is the most important book you have ever read	1
67	135	136	Can we ever store energy produced in lightning?	Is it possible to store the energy of lightning?	1
68	137	138	What is your review of Performance Testing?	What is performance testing?	0
69	139	140	At what cost does so much privacy as in Germany come? W	Are there any people who genuinely enjoy salad with	0
70	141	142	What are the types of immunity?	What are the different types of immunity in our body	0

Figure 3.5: Overview of the Quora Dataset of Questions Pairs

3.3 Task Description

The main goal of this task is to find out identical questions pairs. Our system classifies the questions into two categories duplicate and not-duplicate. We formalize our task of finding identical or duplicate questions as follows: Given a two Question set $Q_1 = (q_{11}, q_{12}, q_{13}, \ldots, q_{1n})$ and $Q_2 = (q_{21}, q_{22}, q_{23}, \ldots, q_{2n})$, our models takes a question from Q_1 and Q_2 to make pair (q_{1i}, q_{2i}) as input, and tries to find out whether the questions are duplicate or not-duplicate.

3.4 Dataset Description

For conducting our experiments we used the Quora dataset[13]. This dataset originally released by Quora has 400k pairs of questions each marked with a ground truth of either 1 or 0. Where 1 denotes duplicate pair of questions and 0 denotes the opposite. Each pair of questions has a unique id named as id. Each question in the question pair also has a unique id named as qid1 and qid2. This dataset contains 149,306 positive examples (i.e, semantically duplicate questions) and 255,045 negative examples

3.4.1 Input and Output

We provide pair of questions (q_{1i}, q_{2i}) as inputs to the deep neural models and obtain either **0** or **1** as outputs.Obtaining an output **1** makes the question pair as identical questions and **0** makes the pair non-identical.

[13]https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

Examples from obtained results:

Question 1: what can an introvert do to become an extrovert & vice-versa?

Question 2: how do i stop being an introvert and become an extrovert?

Output: 1

Ground Truth: 1

Question 1: why was parks and recreation a success?

Question 2: why is parks and recreations so boring?

Output: 0

Ground Truth: 0

Question 1: how do you compare and contrast acids and bases?

Question 2: how do you compare and contrast the properties of acids and bases?

Output: 1

Ground Truth: 1

3.5 Experimantal Setup

3.5.1 Pre-processing

We begin our prepossessing by reading question 1, question 2 and is_duplicate column from the file. Then we perform a series of pre-processing tasks which includes tokenization, removal of stop-words, filtering punctuation marks and Lemmatization. For performing these tasks, we utilized the **NLTK** toolkit[14] for processing the questions to make our dataset clean and noise free. Noise in our case is ungrammatical and meaningless sentences. We now describe each of the preprocessing steps.

[14]http://www.nltk.org/

Tokenization

The tokenization process mainly takes a whole sentence stream as input, splitting it into small tokens. A token is a word or any punctuation marks. A tokenizer splits a sentennce or paragrapgh into tokens when it finds a space or full-stop sign in between the words. The tokenization process can be divided into two types: sentence level tokenization and word level tokenization.

- **Sentence tokenization** splits a paragraph into a list of sentences. For example for a given paragraph, the output will be the following:

  Input Text: "What we can do to help the environment is, 1. Reuse, renew and recycle. This means that we should try to save everything we use like papers, cans, water bottles and old books by putting them in recycling bins which are all around Canada. Use of toxic materials such as plastic and others which are hard to recycle are one of the reasons for global warming. We need educate our fellow people on the importance of recycling and reducing the usage of plastic. Certain plastics have also been introduced which can be recycled."

  Sentence Tokenization Output: ['What we can do to help the environment is, 1. Reuse, renew and recycle', This means that we should try to save everything we use like papers, cans, water bottles and old books by putting them in recycling bins which are all around Canada' , 'Use of toxic materials such as plastic and others which are hard to recycle are one of the reasons for global warming.', 'We need educate our fellow people on the importance of recycling and reducing the usage of plastic', 'Certain plastics have also been introduced which can be recycled.']

- **Word tokenization** processes a sentence by splitting it into a list of individual words. For example for a given sentence , the output will be the following:

  Input Sentence: "What we can do to help the environment is, 1. Reuse, renew and recycle."

Word Tokenization Output: [‘What’ , ‘we’ , ‘can’ , ‘do’ , ‘to’ , ‘help’ , ‘the’ , ‘environment’ , ‘is’ , ‘1’, ‘,’ , ‘Reuse’ , ‘renew.’ , ‘and’ , ‘recycle’ , ‘.’]

Removing punctuation

For removing punctuations and other symbols we used the keras preprocessing[15] library. This library removes known punctuation signs and symbols. We used the text_to_word_sequence function of this library to perform this task.

Removal of stopwords

Stopwords can be referred as common words that reoccur in the sentences but does not contribute to the meaning of the sentences. NLTK has stopword corpus list of 128 words for English languages. The reason behind removing the stopwords is to focus more on the important words and to reduce the length of the sentences. We used a stop word list of 571 words to make our sentences more topic-related. The list of stopwords is shown in Appendix A. For example for a given sentence, the output of stopword removal will be the following:

Input Sentence: “a photon leaves the sun, hits a leaf, hits my retina. is the photon that hit my retina exactly the same photon from the sun?.”

Sentence after filtering stopword: [’photon’, ’leaves’, ’sun’, ’hits’, ’leaf’, ’retina’, ’photon’, ’hit’, ’exactly’, ’photon’, ’sun’, ’?’]

Lemmatization

Lemmatization takes a tokenized form of a sentence as input and groups together the different inflected form of a token to its common base form. Words in English language can appear in several inflected forms depending on the context. To give an example the verb “pass” can appear as, “passed” , “passing” , “passes”. The base form, “pass”, that one might look up in a dictionary, is called the lemma for the word[16] .

[15]https://keras.io/preprocessing/text/
[16]https://en.wikipedia.org/wiki/Lemmatisation

Input Sentence: “has a fan made music video ever been released as the official music video for a song.”

Sentence after Lemmatization:[’has’, ’a’, ’fan’, ’make’, ’music’, ’video’, ’ever’, ’be’, ’release’, ’as’, ’the’, ’official’, ’music’, ’video’, ’for’, ’a’, ’song’]

3.5.2 Generate Word Embeddings

After performing the above steps we now generate the word embedding for each of the clean tokens using the Word2vec (Mikolov et al., 2013a) algorithm discussed earlier in section 2.3.2. We used the gensim library[17] for generating word2vec word embeddings.

We also used the Glove word embedding of Pennington et al. (2014), which is pre-trained with 840B tokens, 2.2M vocab, and has dimensions of 300 for a better representation.

After loading the glove pre-trained embeddings we found the representations for each word in the vocabulary. For this task our vocabulary size is 201,104, indicating that we have this number of unique words in our dataset.

3.5.3 Conversions of Sentences to Vectors

The next step is to convert the sentences into vectors. To do this, it is required to find all the unique vocabularies in the dataset and assign a unique index to each word. This method is known as word to index. Now using those indices from the word to index method, we replace the words in the questions according to their corresponding index thus converting the sentences into word vectors. The same indices are used while reverting back to human readable sentences from the sentence vectors. Consider the following sentences, the sentence vectors will be the following:

Input Sentence: “How can I convert raw file to JPEG in photo in a MacBook?”

[17]https://radimrehurek.com/gensim/models/word2vec.html

Vector outcome of the sentence: [5, 14, 6, 734, 2277, 1397, 8, 14239, 9, 841, 9, 7, 7475]

Input Sentence: "How do you convert raw file to JPEG?"

Vector outcome of the sentence: [5, 11, 16, 734, 2277, 1397, 8, 34184]

3.5.4 Padding Vectors

All the sentences are not of equal lengths. So when the sentences are converted into vectors they are not equal too. In order to have all the vectors of same length we determine a sequence length. If the vectors are smaller than the sequence length then the remaining length is filled up with zeroes either in the front of the sentence or at the back of the sentence. On the other hand if the vector size exceeds the sequence length the rest of the vector is ignored after that length. The sequence length is determined by calculating the average length of the sentences for the whole dataset adding some λ (where λ = 10, 20, 30...). The reason for choosing a sequence length is to reduce the training time. In most of the cases, the sentence length is a lot less than the chosen sequence length. So to avoid processing longer sequence of zeroes we truncate some of the bigger sentences. The resulting vectors are passed into the neural models as input. Considering the following vector, the outcome is shown after padding the sequence of vectors with a sequence length of 30.

Input Vector: [5, 14, 6, 734, 2277, 1397, 8, 14239, 9, 841, 9, 7, 7475]

Vector outcome after Post Padding: [5 14 6 734 2277 1397 8 14239 9 841 9 7 7475 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

3.6 Training

We train all our models mentioned in this chapter with same parameters. We used **ADAM** (Kingma and Ba, 2014b) optimizer with a learning rate of **0.001**. We used post method for padding the vectors. We tried to minimize the loss function using the **categorical crossentropy** discussed earlier in this chapter in section 3.2.1. We experimented on

the sequence length using different maximum sequence length of the vectors. We split the dataset into two portions for training and testing, where 90% data were placed in training set and the rest 10% were kept for testing.

We set the hidden size of the LSTM and the Bi-Directional LSTM layers as 64. We processed the questions by batch of 32 questions pairs each time for training the network. Typically training is faster with mini-batches as the weights are updated after each propagation. But one of the disadvantages the smaller batches is that provide less accurate estimate of the gradient. The whole training process was run for 20 epochs. Epochs are one forward pass and one backward pass on all the training examples. Finally the training was run on **TITAN X GPU** machine with 12 gigabyte of memory.

3.7 Evaluation

We compare the results in terms of **Accuracy**, **Precision**, **Recall**, and **F1 score** described in Section 2.10. We also compare our system against recent works based on the quora dataset. We performed the Evaluation on 2500 question pairs randomly chosen from the dataset.

While running our experiments on the models we observed a similar pattern for the convergence. The accuracy after the 3rd epoch increased sharply comparing to the previous epochs and became flat over the next 2 or 3 epochs. We observed the similar situation in the case of the training loss too. The loss at the beginning of the training was high but it quickly came down and flattened after 2 or 3 epochs.

The training time for each model was about 2 days on average for each of the models. The models that include stacked Bi-directional LSTM took the longest time to train and the model with only LSTM mentioned in section 3.2.1 took the least time to complete the training. This allows us to conclude that more complex the models become more time it need to finish the training. A reason behind that the unidirectional LSTM only processes

one input sequence in one direction only. In the contrast the bi-directional LSTM has to handle the reverse sequence of the input sequence thus increasing the training time.

3.7.1 Baseline Systems

We compared our results against the results mentioned in the paper (Wang et al., 2017). Our Bi-directional LSTM-Stacking with Attention model outperforms all the models in term of accuracy. Our models outperform the models based on Simaese-CNN, Multi-Perspective-CNN, Siamese-LSTM, Multi-Perspective-LSTM, and L.D.C. These models are described in related works in section 2.2.1

3.7.2 Results

The table 3.1 shows various models and their accuracy. The results are sorted according to the accuracy of the models and the names that are in bold highlighting are the models we implemented. From the results of the table 3.1 we observed that the complex models performed better than the simpler models. The model **Bi-directional LSTM-Stacking with Attention** performed best among all other models. This model outperform the state of the art model **BiPMP** (Wang et al., 2017) by a little margin. The other models **Bi-Directional LSTM-ATTN-CNN with maxpooling** and **LSTM with Attention** also perform better than most of the models as shown in table 3.1.

We also show a comparisons between the models we implemented in this Chapter in terms of various metrics. From the results in table 3.2 we can come to the conclusion that **Bi-directional LSTM-Stacking with Attention** performed better than all of the other models. The hybrid models gains 7% performance gain in overall than the basic model in accuracy. Attention over the word vectors is also playing a key factor for improving the performance.

Table 3.1: Performance based on Accuracy for Question Duplication on the Quora dataset.

Models	Accuracy (%)
Siamese-CNN (Wang et al., 2017)	79.60
Multi-Perspective-CNN (Wang et al., 2017)	81.38
LSTM	**81.2**
Siamese-LSTM (Wang et al., 2017)	82.58
Multi-Perspective-LSTM (Wang et al., 2017)	83.21
Bi-Directional LSTM-ATTN-CNN with maxpooling	**85**
L.D.C. (Wang et al., 2016b)	85.55
LSTM with Attention	**87**
BiMPM (Wang et al., 2017)	88.17
Bi-directional LSTM-Stacking with Attention	**88.8**

Table 3.2: Performance Comparisons of the implemented Models

	LSTM	**Bi-Directional LSTM-ATTN-CNN with maxpooling**	**LSTM with Attention**	**Bi-Directional LSTM-Stacking with Attention**
Accuracy	81.2	85	87	**88.8**
Precision	77.7	84.2	85	**85.8**
Recall	79.4	76.4	81.4	**86.2**
F1 score	78.7	86	83.1	**86**

3.8 Summary

In this chapter we implemented some Deep learning methods, outperforming some other models for the task of finding similar questions. The hybrid models are performing better than the basic models. Attention over the word vectors is a key factor for improving the performance.

There are a lot of other possible models that can be formed in extensions of these models like reverse the training questions pairs, **Gradient Recurrent Unit (GRU)** can be introduced instead of LSTM. We also think that training on better machine may provide a extra performance boost. Overall Attention combined with the bi-directional LSTM is clearly a powerful tool and has a promising future for its extension.

Chapter 4

Ranking Answers of Multiple Choice Question

4.1 Introduction

Nowadays online communities are more vocal than ever before. The online forums have a large number of active users. Still sometimes a question starves for an answer, it also happens that a question thread is overwhelmed with answers. In such a situation it is difficult to find relevant answers for the question. So it has become a priority for the researchers in CQA to address this situation. In order to find a solution, there must be a system to sort through the answers according to their relevance to the question.

In this chapter we develop a system that can rank between the answers of Multiple Choice Question(MCQ). Multiple Choice Question is a very popular format of questionnaire, where each question has several possible answers. Among these answers one of them is the correct one. Our system aim to predict the correct answer of a question among several incorrect answers of that question. We approach to solve this task by implementing some deep neural architectures. We rank among the answers of the question to find the best possible answer.

4.2 Model Approaches

In this section we describe our approaches to obtain a better solution of the problem. We used several deep learning techniques in this chapter like Long short-term memory (LSTM), bi-directional LSTM, convolutional neural network (CNN) and attention mechanism all of

which was described in Sections 2.4, 2.5, 2.6, 2.8 respectively. we propose three models in this chapter, a LSTM based model with attention. After that we introduce CNN along with LSTM and attention. Finally we propose a novel model that also uses LSTM, attention and CNN. The difference between the models will be described in later sections of this chapter. All the models implemented in this chapter are inspired by the siamese network structure from Das et al. (2016) and the models proposed by Tan et al. (2015) and Parikh et al. (2016)

4.2.1 MCQ Long Short-Term Memory with Attention

We developed a model with LSTM following a siamese architecture where there is a dedicated uni-directional LSTM for the question and the answer word embeddings. Then the outcome of the LSTM layer is passed through the attention layer to provide the soft attention. After that we merged the outcome of the attention layer with the question-LSTM encoder output for comparison by summation. We used the aggregate method of comparing two vectors by summation before passing it to the classifier (Parikh et al., 2016). Two vectors V_1 and V_2 represent the outcome of the question-LSTM and the attention layer respectively. Each vector has the same length, which is predefined while padding the vectors. The summation is as follows:

$$V_1 = \sum_{i=1}^{Q_n} V_{1,i}$$

$$V_2 = \sum_{i=1}^{Attention_n} V_{2,i}$$

Finally we used a softmax classifier to classify the answers of the given question in two classes correct (1) and incorrect (0). We have used **Categorical Crossentropy** as the loss functions. The following equation calculates the functions loss between the predicted value and the actual value which is the ground truth.

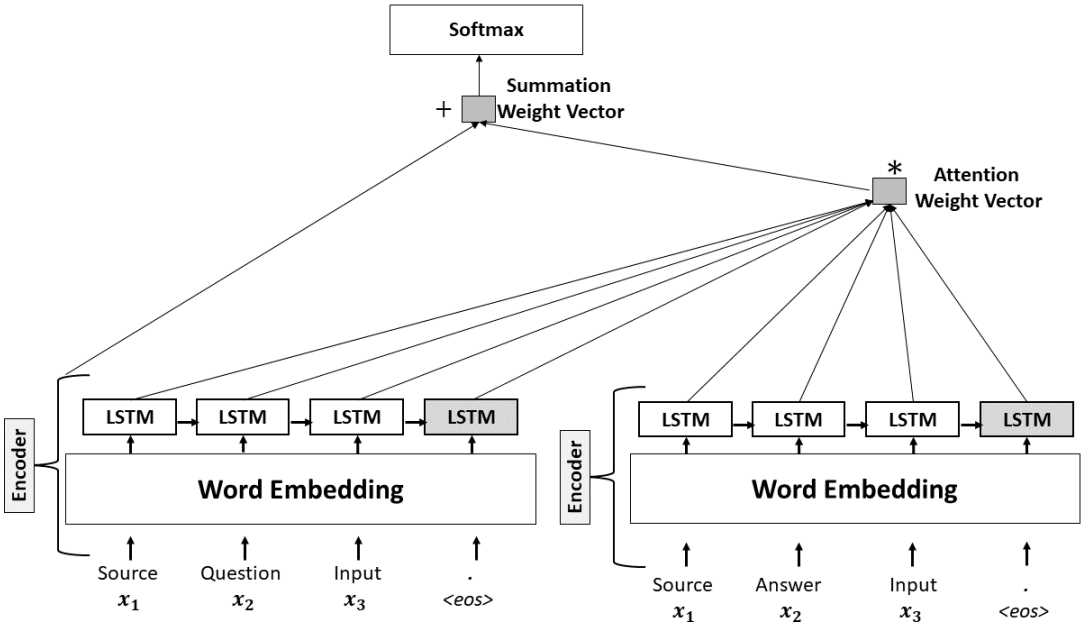

Figure 4.1: MCQ LSTM with Attention model

$$H(p,q) = -\sum_{x} p(x)\log(q(x))$$

Where $p(x)$ denotes the actual value and $q(x)$ denotes the predicted value and x is the number of outcomes or predictions. The predicted class Y is the scoring function which is measured from the probabilities of all the answers of the multiple choice question by the following formula.

$$Y = argmax_i(y_i)$$

where y_i is a list of the probabilities for the answers for a given question. Figure 4.1 illustrates the model.

4.2.2 MCQ LSTM/CNN/Attention with Dense layer

In this section we discuss another model, the encoder receives word embeddings of the questions and answers through separate uni-directional LSTM, CNN and a max-pooling layer following the siamese architecture. The CNN structure is inspired from (Feng et al., 2015). Here the CNN layer takes advantage of the local interactions of the question and

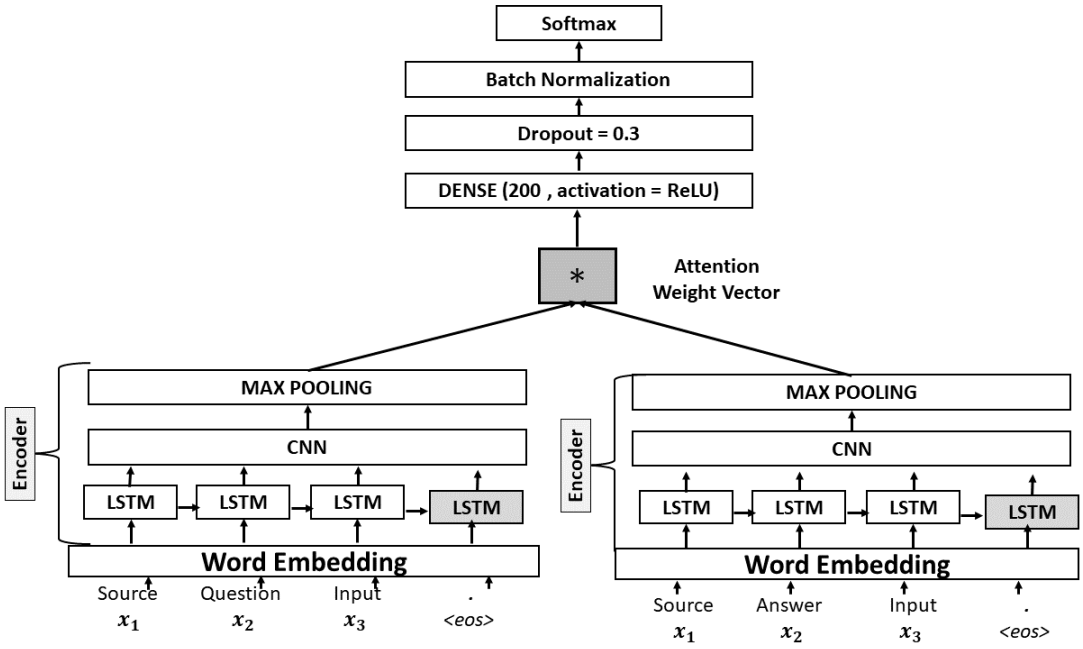

Figure 4.2: MCQ LSTM/CNN/Attention with Dense layer

answer inputs separately within its predefined filter size. It is very much typical to use a pooling layer along with a CNN layer. We combine the output of this two separate chains with an attention layer. After that we pass the outcome of this layer into a dense layer with an activation function of Rectified Linear Unit (ReLU), which always gives a positive output in between $max(0,x)$. We also perform a dropout operation as we do not want our model to overfit during the initial epochs and normalize each batch of vector representation before passing these vectors to the classifier. Figure 4.2 illustrates the model described above.

4.2.3 MCQ Bi-directional LSTM/Attention CNN with Max-pooling

We build a model using bi-directional LSTM following the siamese architecture to form separate question and answer vector representations. Then we pass these vectors to an attention layer. The attended vectors are then passed into a CNN layer. We then perform max pooling operation over the outcome of the CNN layer before passing it to a softmax classifier. Figure 4.3 illustrates the model described above.

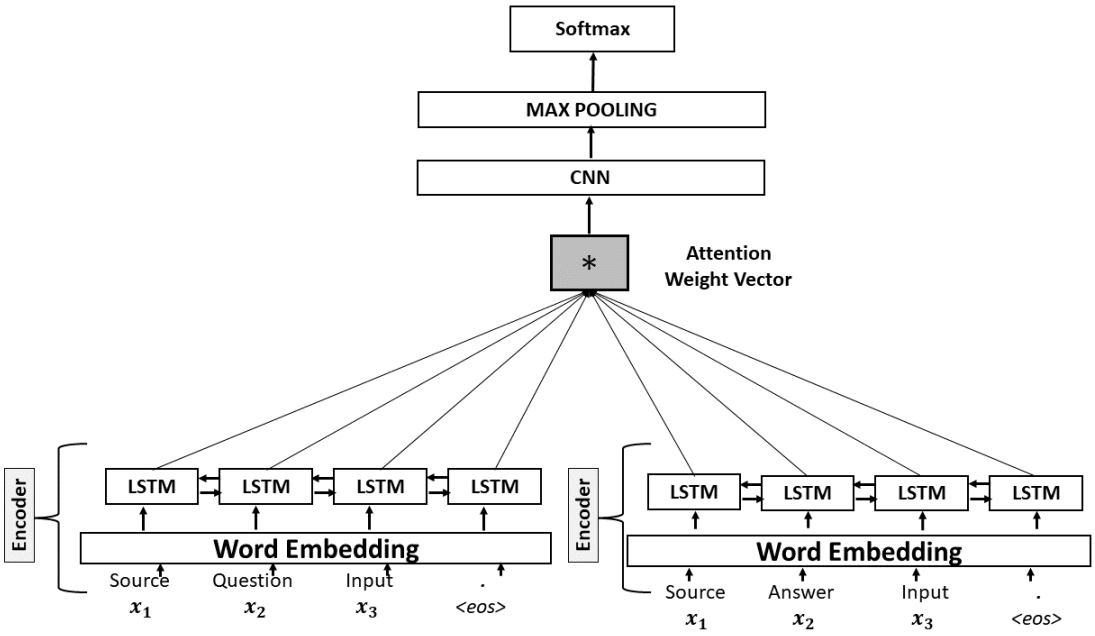

Figure 4.3: MCQ Bi-directional LSTM/Attention CNN with Max-pooling

4.3 Task Description

In simple words the task is to find the correct answer of a Multiple Choice Question (MCQ). We now formalize our task, given a set of question Q = $(q_1, q_2, q_3, \ldots, q_n)$, where each question has an answer set A = $(a_1, a_2, a_3, \ldots, a_m)$. Our model takes a question (q_i) and its answers $(a_1, a_2, a_3, \ldots, a_m)$ to form pairs like (q_i, a_1), (q_i, a_2), (q_i, a_3),..., (q_i, a_m). These question-answer pairs are the inputs for our system. The system will pick the best possible answer from these list of answers. We can also think of this task as ranking between the answers in the process of predicting the best possible answer according to their relevancy.

4.4 Dataset Description

Experiments were conducted on a look alike dataset launched by kaggle for the competition named "Is your model smarter than an 8th grader?"[18]. The dataset we worked on is OpenTriviaQA[19] which has similar format of questions about different trivia's. For our experiment we picked sports and television trivia's, each having 2840 and 5230 multiple

[18]https://www.kaggle.com/c/the-allen-ai-science-challenge
[19]https://github.com/uberspot/OpenTriviaQA

Table 4.1: Data format of OpenTriviaQA

#Q A question until the newline
∧ The text of the correct answer
A multiple choice answer 1
B multiple choice answer 2
C multiple choice answer 3
D multiple choice answer 4

choice questions respectively. Each of the questions have a four possible answers among them one is a correct answer. For each question, correct answer is mentioned that we can use as the ground truth. The format of data is given in table 4.1 as taken form the source. Table 4.1 shows the data format and for better understanding we put some examples from the dataset.

#Q In which US state is the active Mount Rainier volcano located?

∧ Washington

A Michigan

B Ohio

C Washington

D New York

#Q What is the capital and largest city of Hawaii, the 50th US state?

∧ Honolulu

A Little Rock

B Dover

C Frankfort

D Honolulu

#Q Which continents are entirely in the Southern Hemisphere?

∧ Australia and Antarctica

A South America, Africa, and Australia

B Australia and Antarctica

C South America, Australia and Antarctica

D South America and Australia

4.5 Experimental Setup

4.5.1 Pre-processing

All the Pre-processing operation performed on this dataset is similar to the dataset pre-processing done in the section 3.5. Instead of Quora we are processing OpenTriviaQA dataset.

We perform word tokenization operation on the questions and the answers. We also removed the stopwords along with the punctuations. Then we generate the word embeddings of the tokens using the glove embedding pre-trained with 840B tokens. The pre-trained model has 2.2M unique vocabulary and the dimension of 300d that allow us to find better word embedding representation for all the tokens in our dataset (Pennington et al., 2014). Then we convert each of the questions and their answers into vectors by word2index method. We have 17908 unique vocabulary in our dataset. Then pad the vectors with post zeroes with pre-defined λ. As we have four choices for each question, we make four pair of question and answer for the same question for training. Consider a sample question,

Question: what did walter morrison invent in 1948? Answers 'the frisbee', 'the microwave', 'doppler radar', 'the solar battery'.

The pair of question and answer will be as follows:

- (what did walter morrison invent in 1948 , the frisbee)
- (what did walter morrison invent in 1948 , the microwave)
- (what did walter morrison invent in 1948 , the doppler radar)

- (what did walter morrison invent in 1948 , the the solar battery)

Each of the question has unique id for identification, for all the question we mark the pair with the correct answer as 1 and all other pairs of the corresponding questions as 0. For this question the correct answer is "the frisbee". So this pair will have ground truth of 1 and all other pairs will be marked as 0. We split our dataset into training set and test set. We split 90 percent of data for training and 10 percent for testing.

4.6 Training

While training the models we used different parameters for the models. Only the best performing parameters are mentioned. We used **ADAM** (Kingma and Ba, 2014b) optimizer with a learning rate of **0.001**. The smaller learning rate helps the neural model to converge slowly and accurately. We used **categorical crossentropy** as loss function. We set the maximum length of questions and answers to 40.

We used filter size of 32, stride of 5. Larger filter size allows us to capture more information as the covolutional layer become wide. The stride size allows us to move over the words by making a window, thus forming the concept of N-grams, where N is the size of the stride in CNN.

We set the hidden size of the LSTM and Bi-directional LSTM as 64. Each time we pass a batch of 32 pairs of question and answer to the models for training. We continue training for 20 epochs. Epochs are one forward pass and one backward pass on all the training example. Finally the training was run on TITAN X GPU machine with 12 gigabyte of memory

4.7 Evaluation

We evaluated our models in terms of **Accuracy** described in section 2.10. We also compare our result against the results of a data science competition held by kaggle named "Is your model smarter than an 8th grader?". The reason behind using this competition result,

is the similarity in the dataset. We performed the test on 815 MCQ question randomly chosen from both the chosen trivias.

We observe a similar pattern of convergence for all the models. Initially, the model started to train with a higher loss and lower accuracy. But there were noticeable amount of changes in the loss and accuracy after the first few epochs. The loss of the models decreased sharply and the accuracy increased. One more thing we observe is that, after 5 or 6 epochs the loss and accuracy difference from the last epoch was barely distinguishable.

All the model on average took 2 days to train. Among them model MCQ LSTM/CNN/Attention with Dense layer mentioned in section 4.2.2 was the fastest and MCQ Bi-directional LSTM/Attention CNN with Max-pooling mentioned in 4.2.3 took the longest to complete the training. The one reason behind is that unidirectional LSTM only handles one input sequence as it is. On the other hand Bi-directional LSTM has to also handle the reverse sequence of the input sequence thus increasing the training time. CNN algorithm benefits form the CuDNN and CuMEM, which are GPU-accelerated libraries for deep neural network providing highly tuned parameters for faster training.

4.8 Results

In this section we compare our results with the leader-board of a data science competition arranged by kaggle named “Is your model smarter than an 8th grader?”. This competition released a similar type dataset and evaluated the submitted systems based on accuracy. We mention the top 3 systems from the leaderboard[20]. Team Cardal won the competition with an accuracy of 59%. Our all models performed better than the top 3 leading systems of this competition. The bold names in the table indicates our models. Our model MCQ Bi-directional LSTM/Attention CNN with Max-pooling showed the best result among all of the models with an accuracy of 61.5%. Again we can conclude that complex models performs better.

[20]https://www.kaggle.com/c/the-allen-ai-science-challenge/leaderboard

Table 4.2: Performance Comparisons of the implemented Models

Team names and our models	**Accuracy (%)**
Team Alejandro Mosquera	58.2
Team poweredByTalkwalker	58.3
Team Cardal	59.3
MCQ LSTM with attention	**59.4**
MCQ LSTM/CNN/Attention with Dense layer	**60.6**
MCQ Bi-directional LSTM/Attention CNN with Max-pooling	**61.5**

4.9 Summary

In this chapter we implemented some Deep learning methods for solving our answer ranking task. Our models perform better than the mentioned models. Once again the hybrid models performed better than the less complex methods. The approach of combining LSTM, attention and CNN is new.

There are a lot of other possible models that can be formed in extensions of these models **Gradient Recurrent Unit (GRU)** can be introduced instead of LSTM. We can try to achieve better results by using the stacked architectures.

Chapter 5

Answer Classification for Comprehension Question Answering

5.1 Introduction

Recently systems that does the question answering for a reading comprehension has emerged to be a hot topic in the natural language processing researcher community. Systems are developed that can read a paragraph, process and understand it. Then they answer a question based on the paragraph. Most of the recent works on such systems are mainly focused on generating an answer to a question based on information gained from a paragraph. Now the question arises is the predicted answer is a correct one.

In this chapter we are focused to develop a system that predicts whether the answer of the question from a paragraph is correct or incorrect. For implementing this system we used deep learning techniques.

5.2 Overview of the Model

In this section we describe our model that takes a paragraph , a question and an answer to predict whether it is correct or incorrect. In the following section we describe our encoder, the attention mechanism and vector summation method.

We now formalize the task of classifying the answer of the question for a reading comprehension as follows: Given a paragraph P = $(p_1, p_2, p_3, \ldots, p_n)$, question Q = $(q_1, q_2, q_3, \ldots, q_n)$, answers A = $(a_1, a_2, a_3, \ldots, a_n)$, our model takes a triple of paragraph, question and answer (p_i, q_i, a_i) as inputs and tries to predict whether the answer a_i is correct (1) or incorrect (0).

5.2.1 Encoder

The encoder is our case is a bi-directional LSTM (bi-LSTM) (Schuster and Paliwal, 1997) which performs better for long sequences. Simple uni-directional LSTM learns the hidden annotations h_t at time t as processing the input from left to right is as follows:

$$h_t = f(x_t, h_{t-1})$$

Where, the $h_t \in \mathbb{R}^n$ is encoding all the contents seen so far at time t which is computed from h_{t-1} and x_t, where $x_t \in \mathbb{R}^m$ is the m-dimensional embedding of the current word x_t.

It generates an output h_t when t^{th} unit along with the previous output h_{t-1} is passed to the current input x_t. When calculating h_t it uses i_t, f_t, c_t, o_t input gate, forget gate, memory cell and output gate respectively to achieve a good performance over longer sequences. The forward propagation of the LSTM is as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{5.1}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{5.2}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{5.3}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{5.4}$$

$$h_t = o_t \odot \tanh(c_t) \tag{5.5}$$

In the above equations, i_t, f_t, c_t, o_t stand for input gate, forget gate, memory cell and output gate respectively. W_x, W_c ,W_f, $W \in \mathbb{R}^{n \times m}$ are weight matrices, n is the number of hidden units, $\sigma()$ is the sigmoid function. **tanh** is the hyperbolic tangent and $\odot$ denotes an element-wise multiplication.

Conventional RNNs normally process a sequence of words from start to end gradually building the hidden state of each word from its previous words. The hidden state should

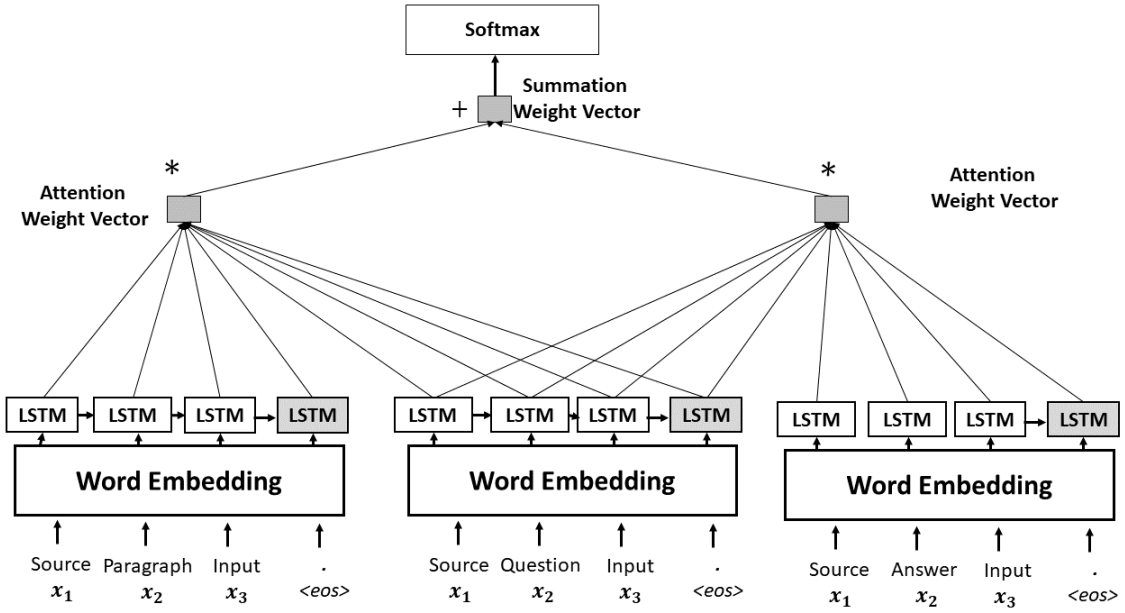

Figure 5.1: Reading comprehension with Bi-directional LSTM and attention

consider the following word as well. Hence, we implement bi-directional RNN (Bi-RNN) (Schuster and Paliwal, 1997) for learning the hidden state, but for our case we use bi-directional LSTM that performs better than uni-directional LSTM.

As shown in Figure 5.1, Bi-LSTM processes the source paragraph, question and answer word embedding representation in both forward and backward direction, while calculating the hidden layers with LSTM to obtain the forward and backward hidden states. Now for each position t, we merge the hidden states by concatenation to achieve the final hidden state:

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t}$$

Where operator $\oplus$ indicates concatenation and $\overrightarrow{h_t}$, $\overleftarrow{h_t}$ represents the forward and backward representation respectively. Forward representation $\overrightarrow{h_t}$ is calculated from $\overrightarrow{h_t} = LSTM(x_t, \overrightarrow{h_{t-1}})$ and backward representation $\overleftarrow{h_t}$ is calculated from $\overleftarrow{h_t} = LSTM(x_t, \overleftarrow{h_{t+1}})$

5.2.2 Attender

Now at each time step t, a context vector c_t is generated by an LSTM hidden state (h_i) from the encoder. This context vector c_t is a weighted sum of the LSTM hidden state. Along with this we add weight to the context vector, this weight is an attention vector. The general idea of the attention vector is how much attention we should provide to a particular word at the particular time step. The weight α_{ti} is the strength of the attention i^{th} word in the target vector to the i^{th} word in the source sentence. Larger the value of α_{ti}, more stronger the attention will be given to a particular word while forming the next vector representation. On this model we applied attention to the context vectors coming from the paragraph and the question LSTMs, thus forming a target attended vector focusing on the words of the paragraph and the question. We also apply attention over the outcome of the question and the answer LSTMs. This helps us to form word relationships between the paragraph and the question and between the question and the answer as well.

5.2.3 Aggregate

After applying attention, we merge the attentive paragraph-question vector and question-answer vector to form the final representation. Now the question becomes the bridge to form word relationship between paragraph and the answer. Two attentive vectors v_1 and v_2 over which we perform aggregation over each set by summation as described by Parikh et al. (2016). The summation is as follows:

$$v_1 = \sum_{i=1}^{Attention_{pq}} v_{1,i}$$

$$v_2 = \sum_{i=1}^{Attention_{qa}} v_{2,i}$$

pq represents number of paragraph-question attention vectors and *qa* represents the number of question-answer attention vectors.

Finally we pass the summation vector to the softmax classifier for prediction between

two classes correct (0) and incorrect (1). We use categorical crossentropy as our loss function.

5.3 Dataset Description

We train our model over the Stanford Question Answering Dataset (SQuAD)[21]. This dataset contains around 90K question-answer pairs along with a paragraph. The dataset is made by the crowdwokers from various Wikipedia articles. The questions and answers are generated from those articles (Rajpurkar et al., 2016). Table 5.1 illustrate the SQuAD dataset format with a simple passage. The example segment in table 5.1 is taken from Rajpurkar et al. (2016). In Table 5.1 the first segment of text is a paragraph, the following next segment consists of a pair of question and answer based on the context of the paragraph.

5.4 Experimental Setup

5.4.1 Dataset Processing

We prepare our dataset for training by marking the ground truth for the answers of a question. We mark the questions with correct answers as 1. We also add some false answers with the questions to generate some negative data. We pick some answers randomly and assign them to some randomly picked questions from the dataset. We keep track of the negative answers by marking them as 0. Then we shuffle our combined dataset with the passage, question, answers pairs so that positive and negative answers get mixed. Then we follow the prepossessing procedure such tokenization, word embedding, converting source sentences to vector and padding the vector operation on the dataset as mentioned in section 3.5. For our dataset we did word level tokenization. We used glove embedding (Pennington et al., 2014), a pre-trained model trained with 840B tokens among 2.2M unique vocabulary to find the word embedding representation for our tokens. We chose the dimension of word

[21] https://rajpurkar.github.io/SQuAD-explorer/

Table 5.1: Format of question-answer pair along with the paragraph in SQuAD dataset

In meteorology, precipitation is any product
of the condensation of atmospheric water vapor
that falls under **gravity**. The main forms of precipitation
include drizzle, rain, sleet, snow, **graupel**
and hail... Precipitation forms as smaller
droplets coalesce via collision with other rain
drops or ice crystals **within a cloud**. Short, intense
periods of rain in scattered locations are
called showers.

What causes precipitation to fall?
gravity

What is another main form of precipitation besides
drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals
to form precipitation?
within a cloud

Table 5.2: Summary of the questions and answers of SQuAD dataset

# of total triplets of paragraph , question and answers	# of triplets having correct answer	# of triplets having incorrect answer	# of triplets for training set	# of triplets for test set
87599	60599	27000	78839	8760

embedding to be 300. We set the length for padding the paragraph, question and answer to 200, 20, 20 respectively. Any words beyond the respective numbers will be ignored. Each training triplets consists of a paragraph, a question and an answer along with ground truth of either 1 or 0. We split the dataset into training and test set. The training set consists of 90 percent of the data while the test set contains the other 10 percent. Both the sets obtained through random split and are indeoendent of each other. We have 196133 unique vocabulary in this dataset. Table 5.2 gives an overview about the dataset for the training and the test sets.

5.5 Training

We used bi-directional LSTMs of size 64. We use the same size of LSTMs for paragraphs, questions and answers. 300 dimensional word embedding are passed through the LSTMs as input. The overall learning rate of the model was **0.01**. We chose **ADAM**(Kingma and Ba, 2014a) as the optimizer over the stochastic gradient descent (SGD) because it can maintain a learning rate over each parameter improving the performance dealing with the problems of sparse gradients. This is very much helpful in our case, because it converges faster than the SGD and maintains same learning rate through out the training phase. We continue our training for 20 epochs. Epochs are one forward pass and one backward pass on all the training examples. We train our model with a mini-batch size of 80. We did not use any dropout for this model. Finally the training was run on TITAN X GPU machine with 12 gigabytes of memory. The training objective of our model is to minimize the loss over the function H by reducing the difference between the actual and the predicted value. We apply categorical crossentropy as our loss function.

$$H(p,q) = -\sum_{x} p(x)\log(q(x))$$

Where $p(x)$ denotes the actual value and $q(x)$ denotes the predicted value and x is the number of outcomes or predictions.

5.6 Evaluation

We evaluate our model automatically using the metrics described in section 2.10.2 such as Accuracy, Precison, Recall and F1 scores. We performed our prediction on the test set which consists of 8760 paragraph, question and answer triplets. The model started training with a large loss value but over the period of training it reduced significantly. The best minimum loss obtained was 0.08 at the 7th epoch. The model started to converge from the 5th epoch as the loss difference was very small from the last epoch. The loss difference flattened out after the 10th epoch.

5.7 Baseline System

Recent works with the SQUAD dataset is mainly focused on generating the answers of the questions. The task we designed evaluates the answer of a comprehension based question and it is using SQUAD dataset. This task of classification of the answers of a question from the reading comprehension is equally important and essential as the comprehension based answer generating task. We used a system developed by Horbach et al. (2013) that classifies the answer of a question based on reading comprehension as our baseline system. Authors used the German CREG corpus (Ott et al., 2012) which contains short answers to the questions. They combined an answer based classifier with textual features and a text based classifier for performing the classification task of the answers. N-gram features is used to find the alignment order of the tokens. Timbl toolkit (Daelemans et al., 2009) is used for classification of the answers.

Table 5.3: Performance comparison with some classification systems

Systems	**Accuracy**
text-based and answer-based model combined (Horbach et al., 2013)	72.2
Siamese-CNN (Wang et al., 2017)	79.60
Multi-Perspective-CNN (Wang et al., 2017)	81.38
Bi-directional LSTM and attention	**81.0**
LSTM	81.2
Siamese-LSTM (Wang et al., 2017)	82.58
Multi-Perspective-LSTM	83.21
Bi-Directional LSTM-ATTN-CNN with maxpooling	85
L.D.C. (Wang et al., 2017)	85.55
LSTM with Attention	87
BiMPM (Wang et al., 2017)	88.17
Bi-directional LSTM-Stacking with Attention	88.8

Table 5.4: Performance of the model on the evaluation metrics

	Accuracy	**Precison**	**Recall**	**F1 Scores**
Bi-directional LSTM and attention	81	81.2	94	87.1

5.8 Results

The model we implemented is for a classification task that can be used to classify the generated answers. So this system can be used to measure the performance of the answer generating systems that are based on reading comprehensions. We measure our model in terms of the popular classification metrics such as the accuracy, the precision, the recall and the F1 score. Table 5.2 shows that our neural network based model performed better than the word by word matching based text models in terms of accuracy. We also compare our results with the models named Simaese-CNN, Multi-Perspective-CNN, Siamese-LSTM, Multi-Perspective-LSTM, and L.D.C proposed by Wang et al. (2017). We also compare this system with models we implemented in chapter 3. The system in bold letters is the system we implemented for solving our reading comprehension answer classification task.

5.8.1 Performance

The output generated by our model is very promising. Table 5.3 show the performance of the model in terms of the accuracy, the precision, the recall and the F1-score. The results shows that it is predicting accurate results in identifying correct and incorrect answers. The value of the precision column indicates low false positive rate. It does not identify wrong answers as correct answers. Higher recall value suggests that our system makes less mistakes in classifying the answers of a question. It can correctly predict that is the answer right or wrong. The number of predictions in which the actual answer is correct but system predicted false is less as the recall value is high. Finally F1 score is the average of the recall and the precision in which we consider the false positives and the false negatives. Higher F1 score establishes that our system makes less mistakes.

5.9 Summary

From the results obtained we can conclude that the model performs better in terms of the recall and the F1 score for this dataset. This method can be used further for evaluating the answer generating systems.

In this chapter, We designed a neural model that can process a paragraph, a question, and an answers establishing word relationship between them and finally classify the answer as correct or incorrect. Our proposed model is simple yet effective solution to a classification problem.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

In this thesis, we have developed some models for solving some well known problems of "Community Question Answering". We implemented four different deep neural models to find similarity between questions. Our model looks for the semantic similarity between the questions. Our model **Bi-directional LSTM-Stacking with Attention** performs outstandingly to acheive a better result than other existing models. We further extended our research towards ranking answers. Our three models are efficient to rank answers of a multiple choice question. The models can predict correct answer form a possible list of choices. The model **MCQ Bi-directional LSTM/Attention CNN with Max-pooling** provides the best outcome on the dataset outperforming the winner of the data science competition arranged by The Allen Institute for Artificial Intelligence. Next we moved our focus to more recent hot topics in the field of CQA which is **Reading Comprehension Question Answering**. We observed that there is a lot of work focused on generating answers for a question based on a paragraph. We changed the a track and focused more on classifying the answer rather than generating one. We created a new task **Answer Classification for Comprehensive Question Answering**, where we enabled a system to predict whether the provided answer is correct or incorrect. Our model **bi-directional LSTM with attention** obtained a good results which in future can be used as a baseline for reading comprehension classification tasks. Our models were based on the state-of-the-art deep neural algorithms such as Long-Short Term Memory (LSTM), Convolutional Neural Network (CNN), At-

tention mechanism, Siamese network. We performed our experiments on Quora dataset for finding the question similarity. We used OpenTriviaQA multiple choice question answering dataset for the ranking task. Finally we used a very recently launched and popular dataset for reading comprehension named Stanford Question Answering Dataset (SQuAD) for the reading comprehension answer classification task. For all the tasks we used popular evaluation metrics like accuracy, precision, recall and F1 scores. Our Experiments demonstrates significant improvement over some state-of-the-art methods.

6.2 Future Work

The result obtained from our experiments on different tasks with different deep neural models on finding semantic similarity, ranking the answers and reading comprehension are effective, still it is possible to improve in a number of different ways:

- We will focus on changing the core algorithms to learn better representation and try to manipulate the hidden layers of deep neural networks.
- We also extend our experiments on different sequence length, optimizer, learning rate, loss function and classifier.
- We have plans to propose a better ranking method dealing with multi-class answers rather than just true and false.
- We intend to modify our model with attention to hierarchical attention networks (Yang et al., 2016) and test our models with the existing models.
- We will try to develop some models for reading comprehension that can generate and also predict the correctness of the generated answer.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR* abs/1409.0473. http://arxiv.org/abs/1409.0473.

Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.* 5(2):157–166. https://doi.org/10.1109/72.279181.

Yoshua Bengio. 2009. Learning deep architectures for ai. *Found. Trends Mach. Learn.* 2(1):1–127. https://doi.org/10.1561/2200000006.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.* 3:1137–1155. http://dl.acm.org/citation.cfm?id=944919.944966.

Jiang Bian, Yandong Liu, Eugene Agichtein, and Hongyuan Zha. 2008. Finding the right facts in the crowd: Factoid question answering over social media. In *Proceedings of the 17th International Conference on World Wide Web*. ACM, New York, NY, USA, WWW '08, pages 467–476. https://doi.org/10.1145/1367497.1367561.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.* 3:993–1022. http://dl.acm.org/citation.cfm?id=944919.944937.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "siamese" time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, NIPS'93, pages 737–744. http://dl.acm.org/citation.cfm?id=2987189.2987282.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 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)*. Association for Computational Linguistics, Doha, Qatar, pages 1724–1734. http://www.aclweb.org/anthology/D14-1179.

S. Chopra, R. Hadsell, and Y. LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. volume 1, pages 539–546 vol. 1. https://doi.org/10.1109/CVPR.2005.202.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR* abs/1412.3555. http://arxiv.org/abs/1412.3555.

Walter Daelemans, Jakub Zavrel, Ko Sloot, and Antal Van Den Bosch. 2009. TiMBL: Tilburg Memory-Based Learner, version 6.2, Reference Guide. ILK Technical Report 09-01. http://ilk.kub.nl/ ilk/papers/ilk9803.ps.gz.

Arpita Das, Harish Yenala, Manoj Kumar Chinnakotla, and Manish Shrivastava. 2016. Together we stand: Siamese networks for similar question retrieval. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. http://aclweb.org/anthology/P/P16/P16-1036.pdf.

Minwei Feng, Bing Xiang, Michael R. Glass, Lidan Wang, and Bowen Zhou. 2015. Applying deep learning to answer selection: A study and an open task. *CoRR* abs/1508.01585. http://arxiv.org/abs/1508.01585.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press.

Alex Graves, Navdeep Jaitly, and Abdel rahman Mohamed. 2013. Hybrid speech recognition with deep bidirectional lstm. In *ASRU*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Andrea Horbach, Alexis Palmer, and Manfred Pinkal. 2013. Using the text to evaluate short answers for reading comprehension exercises. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*. Association for Computational Linguistics, Atlanta, Georgia, USA, pages 286–295. http://www.aclweb.org/anthology/S13-1041.

Shafiq Joty, Alberto Barrón-Cedeño, Giovanni Da San Martino, Simone Filice, Lluís Màrquez, Alessandro Moschitti, and Preslav Nakov. 2015. Global thread-level inference for comment classification in community question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 573–578. http://aclweb.org/anthology/D15-1068.

Diederik Kingma and Jimmy Ba. 2014a. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Diederik P. Kingma and Jimmy Ba. 2014b. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980. http://arxiv.org/abs/1412.6980.

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. 2015. From word embeddings to document distances. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. JMLR.org, ICML'15, pages 957–966. http://dl.acm.org/citation.cfm?id=3045118.3045221.

Thomas K Landauer, Peter W Foltz, and Darrell Laham. 1998. An introduction to latent semantic analysis. *Discourse processes* 25(2-3):259–284.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*. Springer-Verlag, London, UK, UK, pages 319–. http://dl.acm.org/citation.cfm?id=646469.691875.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Lisbon, Portugal, pages 1412–1421.

Roland Memisevic, Christopher Zach, Marc Pollefeys, and Geoffrey E Hinton. 2010. Gated softmax classification. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, Curran Associates, Inc., pages 1603–1611. http://papers.nips.cc/paper/3895-gated-softmax-classification.pdf.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*. Curran Associates Inc., USA, NIPS'13, pages 3111–3119. http://dl.acm.org/citation.cfm?id=2999792.2999959.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. *CoRR* abs/1310.4546. http://arxiv.org/abs/1310.4546.

Jonas Mueller and Aditya Thyagarajan. 2016. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press, AAAI'16, pages 2786–2792. http://dl.acm.org/citation.cfm?id=3016100.3016291.

Graham Neubig. 2017. Neural machine translation and sequence-to-sequence models: A tutorial. *CoRR* abs/1703.01619. http://arxiv.org/abs/1703.01619.

Niels Ott, Ramon Ziai, and Detmar Meurers. 2012. Creation and analysis of a reading comprehension exercise corpus. *Multilingual corpora and multilingual corpus analysis* 14:47.

Boyuan Pan, Hao Li, Zhou Zhao, Bin Cao, Deng Cai, and Xiaofei He. 2017. MEMEN: multi-layer embedding with memory networks for machine comprehension. *CoRR* abs/1707.09098. http://arxiv.org/abs/1707.09098.

Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. *CoRR* abs/1606.01933. http://arxiv.org/abs/1606.01933.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. *CoRR* abs/1606.05250. http://arxiv.org/abs/1606.05250.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, and Phil Blunsom. 2015. Reasoning about entailment with neural attention. *CoRR* abs/1509.06664. http://arxiv.org/abs/1509.06664.

M. Schuster and K.K. Paliwal. 1997. Bidirectional recurrent neural networks. *Trans. Sig. Proc.* 45(11):2673–2681. https://doi.org/10.1109/78.650093.

Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, USA, SIGIR '15, pages 373–382. https://doi.org/10.1145/2766462.2767738.

Ming Tan, Bing Xiang, and Bowen Zhou. 2015. Lstm-based deep learning models for non-factoid answer selection. *CoRR* abs/1511.04108. http://arxiv.org/abs/1511.04108.

Kateryna Tymoshenko, Daniele Bonadiman, and Alessandro Moschitti. 2016. Learning to rank non-factoid answers: Comment selection in web forums. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, New York, NY, USA, CIKM '16, pages 2049–2052. https://doi.org/10.1145/2983323.2983906.

Zhiguo Wang, Wael Hamza, and Radu Florian. 2017. Bilateral multi-perspective matching for natural language sentences. *CoRR* abs/1702.03814. http://arxiv.org/abs/1702.03814.

Zhiguo Wang, Haitao Mi, and Abraham Ittycheriah. 2016a. Semi-supervised clustering for short text via deep representation learning. *CoRR* abs/1602.06797. http://arxiv.org/abs/1602.06797.

Zhiguo Wang, Haitao Mi, and Abraham Ittycheriah. 2016b. Sentence similarity learning by lexical decomposition and composition. *CoRR* abs/1602.07019. http://arxiv.org/abs/1602.07019.

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

Xiaobing Xue, Jiwoon Jeon, and W. Bruce Croft. 2008. Retrieval models for question and answer archives. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, USA, SIGIR '08, pages 475–482. https://doi.org/10.1145/1390334.1390416.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. 2016. Hierarchical attention networks for document classification. In *HLT-NAACL*.

Appendix A

Smart Stopwords List

Table A.1: Smart Stopwords List

a	contain	hers	nine	some	very
a's	containing	herself	no	somebody	via
able	contains	hi	nobody	somehow	viz
about	corresponding	him	non	someone	vs
above	could	himself	none	something	w
according	couldn't	his	noone	sometime	want
accordingly	course	hither	nor	sometimes	wants
across	currently	hopefully	normally	somewhat	was
actually	d	how	not	somewhere	wasn't
after	definitely	howbeit	nothing	soon	way
afterwards	described	however	novel	sorry	we
again	despite	i	now	specified	we'd
against	did	i'd	nowhere	specify	we'll
ain't	didn't	i'll	o	specifying	we're
all	different	i'm	obviously	still	we've
allow	do	i've	of	sub	welcome
allows	does	ie	off	such	well
almost	doesn't	if	often	sup	went
alone	doing	ignored	oh	sure	were
along	don't	immediate	ok	t	weren't
already	done	in	okay	t's	what
also	down	inasmuch	old	take	what's
although	downwards	inc	on	taken	whatever
always	during	indeed	once	tell	when
am	e	indicate	one	tends	whence
among	each	indicated	ones	th	whenever
amongst	edu	indicates	only	than	where
an	eg	inner	onto	thank	where's
and	eight	insofar	or	thanks	whereafter
another	either	instead	other	thanx	whereas
any	else	into	others	that	whereby
anybody	elsewhere	inward	otherwise	that's	wherein
anyhow	enough	is	ought	thats	whereupon
anyone	entirely	isn't	our	the	wherever

anything	especially	it	ours	their	whether
anyway	et	it'd	ourselves	theirs	which
anyways	etc	it'll	out	them	while
anywhere	even	it's	outside	themselves	whither
apart	ever	its	over	then	who
appear	every	itself	overall	thence	who's
appreciate	everybody	j	own	there	whoever
appropriate	everyone	just	p	there's	whole
are	everything	k	particular	thereafter	whom
aren't	everywhere	keep	particularly	thereby	whose
around	ex	keeps	per	therefore	why
as	exactly	kept	perhaps	therein	will
aside	example	know	placed	theres	willing
ask	except	knows	please	thereupon	wish
asking	f	known	plus	these	with
associated	far	l	possible	they	within
at	few	last	presumably	they'd	without
available	fifth	lately	probably	they'll	won't
away	first	later	provides	they're	wonder
awfully	five	latter	q	they've	would
b	followed	latterly	que	think	would
be	following	least	quite	third	wouldn't
became	follows	less	qv	this	x
because	for	lest	r	thorough	y
become	former	let	rather	thoroughly	yes
becomes	formerly	let's	rd	those	yet
becoming	forth	like	re	though	you
been	four	liked	really	three	you'd
before	from	likely	reasonably	through	you'll
beforehand	further	little	regarding	throughout	you're
behind	furthermore	look	regardless	thru	you've

being	g	looking	regards	thus	your
believe	get	looks	relatively	to	yours
below	gets	ltd	respectively	together	yourself
beside	getting	m	right	too	yourselves
besides	given	mainly	s	took	z
best	gives	many	said	toward	zero
better	go	may	same	towards	
between	goes	maybe	saw	tried	
beyond	going	me	say	tries	
both	gone	mean	saying	truly	
brief	got	meanwhile	says	try	
but	gotten	merely	second	trying	
by	greetings	might	secondly	twice	
c	h	more	see	two	
c'mon	had	moreover	seeing	u	
c's	hadn't	most	seem	un	
came	happens	mostly	seemed	under	
can	hardly	much	seeming	unfortunately	
can't	has	must	seems	unless	
cannot	hasn't	my	seen	unlikely	
cant	have	myself	self	until	
cause	haven't	n	selves	unto	
causes	having	name	sensible	up	
certain	he	namely	sent	upon	
certainly	he's	nd	serious	us	
changes	hello	near	seriously	use	
clearly	help	nearly	seven	used	
co	hence	necessary	several	useful	
com	her	need	shall	uses	
come	here	needs	she	using	
comes	here's	neither	should	usually	
concerning	hereafter	never	shouldn't	uucp	
consequently	hereby	nevertheless	since	v	
consider	herein	new	six	value	
considering	hereupon	next	so	various	

Appendix B

Summaries of the models (System generated) for Question Question Duplication task

In the following we provide system generated model summaries of the model described for the task question-question duplication in chapter 3.

Question-Question LSTM Model Summary

```
_________________
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================
input_1 (InputLayer)         (None, 30)                0
_____________________________________________________________________________
input_2 (InputLayer)         (None, 30)                0
_____________________________________________________________________________
embedding_1 (Embedding)      (None, 30, 300)           60330000    input_1[0][0]
_____________________________________________________________________________
embedding_2 (Embedding)      (None, 30, 300)           60330000    input_2[0][0]
_____________________________________________________________________________
lstm_1 (LSTM)                (None, 30, 64)            93440       embedding_1[0][0]
_____________________________________________________________________________
lstm_2 (LSTM)                (None, 30, 64)            93440       embedding_2[0][0]
_____________________________________________________________________________
add_1 (Add)                  (None, 30, 64)            0           lstm_1[0][0]
                                                                   lstm_2[0][0]
_____________________________________________________________________________
flatten_1 (Flatten)          (None, 1920)              0           add_1[0][0]
_____________________________________________________________________________
dense_1 (Dense)              (None, 2)                 3842        flatten_1[0][0]
==================================================================================
Total params: 120,850,722
Trainable params: 190,722
Non-trainable params: 120,660,000
```

Question-Question LSTM with Attention Model Summary

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_1 (InputLayer)         (None, 30)                0
_____________________________________________________________________________________________
input_2 (InputLayer)         (None, 30)                0
_____________________________________________________________________________________________
embedding_1 (Embedding)      (None, 30, 300)           60330000    input_1[0][0]
_____________________________________________________________________________________________
embedding_2 (Embedding)      (None, 30, 300)           60330000    input_2[0][0]
_____________________________________________________________________________________________
lstm_1 (LSTM)                (None, 30, 64)            93440       embedding_1[0][0]
_____________________________________________________________________________________________
lstm_2 (LSTM)                (None, 30, 64)            93440       embedding_2[0][0]
_____________________________________________________________________________________________
dot_1 (Dot)                  (None, 30, 30)            0           lstm_1[0][0]
                                                                   lstm_2[0][0]
```

Layer (type)	Output Shape	Param #	Connected to
flatten_1 (Flatten)	(None, 900)	0	dot_1[0][0]
dense_1 (Dense)	(None, 1920)	1729920	flatten_1[0][0]
reshape_1 (Reshape)	(None, 30, 64)	0	dense_1[0][0]
add_1 (Add)	(None, 30, 64)	0	lstm_1[0][0] reshape_1[0][0]
flatten_2 (Flatten)	(None, 1920)	0	add_1[0][0]
dense_2 (Dense)	(None, 2)	3842	flatten_2[0][0]

Total params: 122,580,642
Trainable params: 1,920,642
Non-trainable params: 120,660,000

Question-Question Bi-directional LSTM-Stacking with Attention Model Summary

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 30)	0	
input_2 (InputLayer)	(None, 30)	0	
embedding_1 (Embedding)	(None, 30, 300)	60330000	input_1[0][0]
embedding_2 (Embedding)	(None, 30, 300)	60330000	input_2[0][0]
bidirectional_1 (Bidirectional)	(None, 30, 64)	186880	embedding_1[0][0]
bidirectional_4 (Bidirectional)	(None, 30, 64)	186880	embedding_2[0][0]
bidirectional_2 (Bidirectional)	(None, 30, 64)	66048	bidirectional_1[0][0]
bidirectional_5 (Bidirectional)	(None, 30, 64)	66048	bidirectional_4[0][0]
bidirectional_3 (Bidirectional)	(None, 30, 64)	66048	bidirectional_2[0][0]
bidirectional_6 (Bidirectional)	(None, 30, 64)	66048	bidirectional_5[0][0]
dot_1 (Dot)	(None, 30, 30)	0	bidirectional_3[0][0] bidirectional_6[0][0]
flatten_1 (Flatten)	(None, 900)	0	dot_1[0][0]
dense_1 (Dense)	(None, 1920)	1729920	flatten_1[0][0]
reshape_1 (Reshape)	(None, 30, 64)	0	dense_1[0][0]
add_1 (Add)	(None, 30, 64)	0	bidirectional_3[0][0] reshape_1[0][0]
flatten_2 (Flatten)	(None, 1920)	0	add_1[0][0]
dense_2 (Dense)	(None, 2)	3842	flatten_2[0][0]

Total params: 123,031,714
Trainable params: 2,371,714
Non-trainable params: 120,660,000

Question-Question Bi-Directional LSTM-ATTN-CNN with maxpooling Model Summary

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 40)	0	
input_2 (InputLayer)	(None, 40)	0	
embedding_1 (Embedding)	(None, 40, 300)	60330000	input_1[0][0]
embedding_2 (Embedding)	(None, 40, 300)	60330000	input_2[0][0]
bidirectional_1 (Bidirectional)	(None, 40, 64)	186880	embedding_1[0][0]

bidirectional_4 (Bidirectional)	(None, 40, 64)	186880	embedding_2[0][0]
bidirectional_2 (Bidirectional)	(None, 40, 64)	66048	bidirectional_1[0][0]
bidirectional_5 (Bidirectional)	(None, 40, 64)	66048	bidirectional_4[0][0]
bidirectional_3 (Bidirectional)	(None, 40, 64)	66048	bidirectional_2[0][0]
bidirectional_6 (Bidirectional)	(None, 40, 64)	66048	bidirectional_5[0][0]
dot_1 (Dot)	(None, 40, 40)	0	bidirectional_3[0][0] bidirectional_6[0][0]
conv1d_1 (Conv1D)	(None, 11, 32)	38432	dot_1[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 1, 32)	0	conv1d_1[0][0]
dropout_1 (Dropout)	(None, 1, 32)	0	max_pooling1d_1[0][0]
flatten_1 (Flatten)	(None, 32)	0	dropout_1[0][0]
dense_1 (Dense)	(None, 2)	66	flatten_1[0][0]

Total params: 121,336,450
Trainable params: 676,450
Non-trainable params: 120,660,000

Appendix C

Summaries of the models (System generated) for Ranking Answers of MCQ

In the following we provide system generated model summaries of the model described for the task question-question duplication in chapter 4.

MCQ Long Short-Term Memory with Attention

```
Layer (type)                 Output Shape          Param #     Connected to
==================================================================================
input_1 (InputLayer)         (None, 50)            0
__________________________________________________________________________________
input_2 (InputLayer)         (None, 50)            0
__________________________________________________________________________________
embedding_1 (Embedding)      (None, 50, 300)       8201100     input_1[0][0]
__________________________________________________________________________________
embedding_2 (Embedding)      (None, 50, 300)       8201100     input_2[0][0]
__________________________________________________________________________________
lstm_1 (LSTM)                (None, 50, 64)        93440       embedding_1[0][0]
__________________________________________________________________________________
lstm_2 (LSTM)                (None, 50, 64)        93440       embedding_2[0][0]
__________________________________________________________________________________
conv1d_2 (Conv1D)            (None, 46, 32)        10272       lstm_2[0][0]
__________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 23, 32)     0           conv1d_1[0][0]
__________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 23, 32)     0           conv1d_2[0][0]
__________________________________________________________________________________
dropout_1 (Dropout)          (None, 23, 32)        0           max_pooling1d_1[0][0]
__________________________________________________________________________________
dropout_2 (Dropout)          (None, 23, 32)        0           max_pooling1d_2[0][0]
__________________________________________________________________________________
add_1 (Add)                  (None, 23, 32)        0           dropout_1[0][0]
                                                               dropout_2[0][0]
__________________________________________________________________________________
flatten_1 (Flatten)          (None, 736)           0           add_1[0][0]]
__________________________________________________________________________________
dense_1 (Dense)              (None, 2)             402        flatten_1[0][0]    batch_normalization_4[0][0]
==================================================================================
Total params: 16,881,226
Trainable params: 477,426
Non-trainable params: 16,403,800
```

MCQ LSTM/CNN/Attention with Dense layer

```
Layer (type)                 Output Shape          Param #     Connected to
==================================================================================================
input_3 (InputLayer)         (None, 50)            0
__________________________________________________________________________________________________
input_4 (InputLayer)         (None, 50)            0
```

embedding_3 (Embedding)	(None, 50, 300)	8201100	input_3[0][0]
embedding_4 (Embedding)	(None, 50, 300)	8201100	input_4[0][0]
bidirectional_1 (Bidirectional)	(None, 50, 64)	186880	embedding_3[0][0]
bidirectional_2 (Bidirectional)	(None, 50, 64)	186880	embedding_4[0][0]
dot_1 (Dot)	(None, 50, 50)	0	bidirectional_1[0][0] bidirectional_2[0][0]
conv1d_3 (Conv1D)	(None, 46, 32)	8032	dot_1[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 23, 32)	0	conv1d_3[0][0]
dropout_7 (Dropout)	(None, 23, 32)	0	max_pooling1d_3[0][0]
flatten_2 (Flatten)	(None, 736)	0	dropout_7[0][0]
dense_6 (Dense)	(None, 2)	1474	flatten_2[0][0]

Total params: 16,785,466
Trainable params: 383,266
Non-trainable params: 16,402,200

MCQ LSTM/CNN/Attention with Dense layer

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 50)	0	
input_4 (InputLayer)	(None, 50)	0	
embedding_3 (Embedding)	(None, 50, 300)	8201100	input_3[0][0]
embedding_4 (Embedding)	(None, 50, 300)	8201100	input_4[0][0]
bidirectional_1 (Bidirectional)	(None, 50, 64)	186880	embedding_3[0][0]
bidirectional_2 (Bidirectional)	(None, 50, 64)	186880	embedding_4[0][0]
dot_1 (Dot)	(None, 50, 50)	0	bidirectional_1[0][0] bidirectional_2[0][0]
conv1d_3 (Conv1D)	(None, 46, 32)	8032	dot_1[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 23, 32)	0	conv1d_3[0][0]
dropout_7 (Dropout)	(None, 23, 32)	0	max_pooling1d_3[0][0]
flatten_2 (Flatten)	(None, 736)	0	dropout_7[0][0]
dense_6 (Dense)	(None, 2)	1474	flatten_2[0][0]

Total params: 16,785,466
Trainable params: 383,266
Non-trainable params: 16,402,200

Appendix D

Summaries of the models (System generated) for Answer Classification for Comprehensive Question Answering task

In the following we provide system generated model summary of the model described for the task question-question duplication in chapter 5.

Bi-directional LSTM with attention

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 20)	0	
embedding_2 (Embedding)	(None, 20, 300)	57100800	input_2[0][0]
input_1 (InputLayer)	(None, 200)	0	
bidirectional_2 (Bidirectional)	(None, 20, 64)	186880	embedding_2[0][0]
embedding_1 (Embedding)	(None, 200, 300)	57100800	input_1[0][0]
dropout_2 (Dropout)	(None, 20, 64)	0	bidirectional_2[0][0]
bidirectional_1 (Bidirectional)	(None, 200, 64)	186880	embedding_1[0][0]
bidirectional_3 (Bidirectional)	(None, 20, 64)	66048	dropout_2[0][0]
dropout_1 (Dropout)	(None, 200, 64)	0	bidirectional_1[0][0]
dropout_3 (Dropout)	(None, 20, 64)	0	bidirectional_3[0][0]
dot_1 (Dot)	(None, 200, 20)	0	dropout_1[0][0] dropout_2[0][0]
dot_2 (Dot)	(None, 20, 20)	0	dropout_2[0][0] dropout_3[0][0]
flatten_1 (Flatten)	(None, 4000)	0	dot_1[0][0]
flatten_2 (Flatten)	(None, 400)	0	dot_2[0][0]
dense_1 (Dense)	(None, 6400)	25606400	flatten_1[0][0]
dense_2 (Dense)	(None, 6400)	2566400	flatten_2[0][0]
reshape_1 (Reshape)	(None, 100, 64)	0	dense_1[0][0]
reshape_2 (Reshape)	(None, 100, 64)	0	dense_2[0][0]
add_1 (Add)	(None, 100, 64)	0	reshape_1[0][0] reshape_2[0][0]

```
____________________________________________________________________________________________________
flatten_3 (Flatten)              (None, 6400)          0           add_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 2)             12802       flatten_3[0][0]
====================================================================================================
Total params: 142,827,010
Trainable params: 28,625,410
Non-trainable params: 114,201,600
____________________________________________________________________________________________________
```