Analysis of State of the Art Deep Learning based Techniques for Medical Natural Answer Generation

Submitted May 2020, in partial fulfillment of
the conditions for the award of the degree **BSc Computer Science with Artificial Intelligence.**

Xuanming Zhang
16521855

Supervised by Dr. Zheng Lu

School of Computer Science
University of Nottingham Ningbo China

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature Xuanming Zhang

Date 18th / May / 2020

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.

Abstract

Deep Learning based techniques have been widely adopted in Natural Language Processing (NLP) tasks. Such tasks include text generation, which aims to generate the target sequence given the input source sequence. Specifically, answer generation can be deemed as a type of text generation, where the target answer is generated in response to the input source question. Recently, several well-performing Deep Learning based approaches have been successfully applied to answer generation in many real world application domains (e.g. Community Question-Answer) and achieved state-of-the-art results on several benchmark datasets. Nevertheless, due to inaccessibility and challenges in acquiring large-scale training data in medical domain, few experiments has been conducted to reveal the power of these state-of-the-art Deep Learning based techniques on medical domain Question-Answer (QA) tasks. In our work, we specifically probed QA in medical domain, especially in the context of medical natural answer generation, the goal of which is to generate natural answers given medical related queries. Moreover, we researched on the existing data corpus for training a medical QA system, among which a newly developed medical QA dataset (i.e. *MedQuAD*[1]) was selected for model training and testing. This dataset consists of thousands of Question-Answer pairs. In addition, we explored several state-of-the-art Deep Learning based approaches, including sequence-to-sequence modelling [68], adopting pre-trained encoders and decoders [13, 65, 77, 85] and etc., and experimented with them on *MedQuAD*. In the end, we conducted in-depth analysis based on the evaluation results of these chosen state-of-the-art deep learning based systems on *MedQuAD*.

Acknowledgements

We thank **Dr. Zheng Lu** for supervising our project and offering variety of inspiring ideas to this work. Also, special thanks should go to Dr. Yue Li, who originally created this practical Latex format and eases the process of editing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Automatic question answering (QA) system has received a lot of attentions from Natural Language Processing (NLP) community. In retrospect, automatic QA has made huge progress with the help of several open-domain and machine comprehension systems built on the basis of large-scale annotated datasets [76, 15]. Moreover, in the light of the breakthrough of deep learning techniques applied at NLP, QA systems has obtained greater popularity in the past couple of years [61]. Even though there are a lot of well-performing deep learning based QA systems in many real world application domains (e.g. Community QA and Open Domain QA), particular obstacles and challenges persist in the medical domain [61]. First of all, due to the fact that highly precise answers are required in medical QA systems, expert and professional knowledge needs to be incorporated from different resources [61]. However, due to insufficient number of professionals in medicine and restricted access to adequate medical resources for some people [21], it is quite difficult to build up such medical QA systems. Another obstacle is that the volume of reliable fine-grained medical QA pairs for training the system is restricted since it is time-consuming for medical experts to conduct the labeling process [61]. Therefore, few well-performing deep learning based medical QA models can be trained because of the insufficiency of the training data (i.e. QA pairs) [12]. Regardless of these difficulties, automatic medical QA systems is urgently needed because these virtual health consultancy can effectively

address the problems raised by patients, through generating immediate and accurate answers in response to the medical queries[21]. Therefore, great efforts have been made in order to overcome the existing challenges in medical domain QA. For example, to tackle the problem of data insufficiency for training medical QA models, [43] and [56] provides practical insights to enrich the training QA pairs so that the size of the training data can be expanded. Specifically, [43] proposed a QA corpus for Electronic Medical Records (EMR) related research (i.e. emrQA corpus). Besides, a Chinese medical QA data corpus was developed by [21] to facilitate the research in Chinese medical domain. With the help of these emergent medical QA dataset containing large-scale Q-A pairs, further advance in the development of automatic medical QA systems can be achieved.

As indicated in [61], deep learning based techniques have been applied to multiple NLP tasks, including text generation (i.e. generating the target texts given the source texts), and have achieved state-of-the-art results. However, to the best of my knowledge, few attempts has been made to apply these state-of-the-art systems on medical answer generation, where answers can be generated given medical related queries. The motivation of the project is therefore to 1) search for the emergent medical QA pairs dataset; 2) apply state-of-the-art deep learning based techniques on the medical dataset and 3) analyze the performances of these state-of-the-art techniques for medical answer generation.

1.2 Aims and Objectives

As indicated in the motivation section, several recently emergent medical QA corpus come into rescue for training and testing a deep learning based system. However, not all of the corpus can be selected as the training data, since the structures of each of the dataset deviate from each other. For instance, QA pairs proposed in [2] contain, for each question, a long answer (consists of more than one paragraph), while in [43] the answers are much shorter, with only one sentence or paragraph. Thus, the aims and objectives of the project are specified as follows:

- Compare and contrast different data corpus and select the most suitable data corpus for training the deep learning system.
- Research for state-of-the-art deep learning techniques for answer generation in any domain (not specifically in medical domain).
- Apply the selected state-of-the-art deep learning based approaches to the chosen data corpus (i.e. train and test the systems on the chosen dataset).
- Conduct analysis and evaluations on the performance results of the chosen state-of-the-art deep learning systems for medical answer generation.

Chapter 2

Background and Related Work

2.1 History of Question and Answering Systems

In retrospect, the study of QA system was initiated by [18] in 1961, when the first QA system was developed in the context of baseball games. The emergence of such system allowed users to query any information related to a baseball game, such as the date and venue of a game. This triggered the insights that users can ask questions in the platform and expect precise and accurate answers generated from the machine system in a highly automated manner without having to wait for the reply from a human.

LUNAR system was another well-known work in QA field [80]. It was created in 1971 following the Apollo moon mission. The intention of the system was to enable lunar geologists to access more conveniently the data for the chemical analysis on lunar rock and its soil composition. Also, some other early QA system (such as SYNTHEX) intended to accomplish the same aim of retrieving the answer for a given question expressed in natural language [46]. Over the past a few decades, QA systems have evolved massively and they have been improved to be equipped with the structure formatting the foundation of modern QA systems.

2.2 Different Varieties of QA researches

Throughout the past few decades, research in QA has evolved into two different branches, domain restricted QA and open domain QA.

Specifically, since the early stages of of AI, researches in QA have created systems that can answer questions with the help of knowledge base, which encodes the knowledge in the database as an information resource [38]. These systems can answer questions related to the information initially encoded in the database. Obviously, these systems can only provide answers for the querying information in a restricted domain. According to [38], the benefit of adopting knowledge-based approaches in domain restricted QA research is to have a conceptual model of the application domain that can enable the use of advanced techniques (e.g. theorem proving) so that more complicated information requests can be addressed.

On the contrary, with the surge of activity in the area of Information Retrieval (IR) [76], some of the QA researches have put great emphasize on locating text excerpts that may potentially contain the answer within large sets of documents. As a result, these QA systems can handle questions independent of the application domain (i.e. open domain QA). In the context of open domain QA, the overall pipeline was proposed in [60] and is shown in Figure 2.1 below.

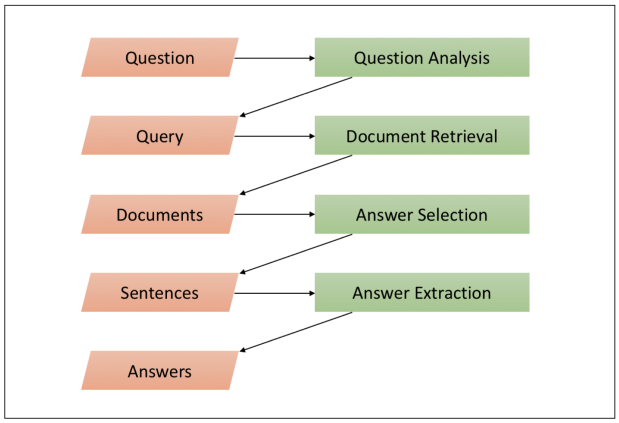

Figure 2.1: Question Answering Pipeline for Open Domain Systems

2.3 Major Components in QA systems

In general, Question Answering (QA) serves as a research field that fuses different, but relevant, research areas, such as Information Retrieval (IR), Information Extraction (IE) and Natural Language Processing (NLP) [3]. Even though details of different QA systems vary from each other, a classical modern QA system consists of three principle modules: Question Processing, Document Processing and Answer Processing [3]. Specifically, these three major modules establish the foundation of the general modern QA system architecture. The overall structure of the QA architecture is demonstrated in Figure 2.2.

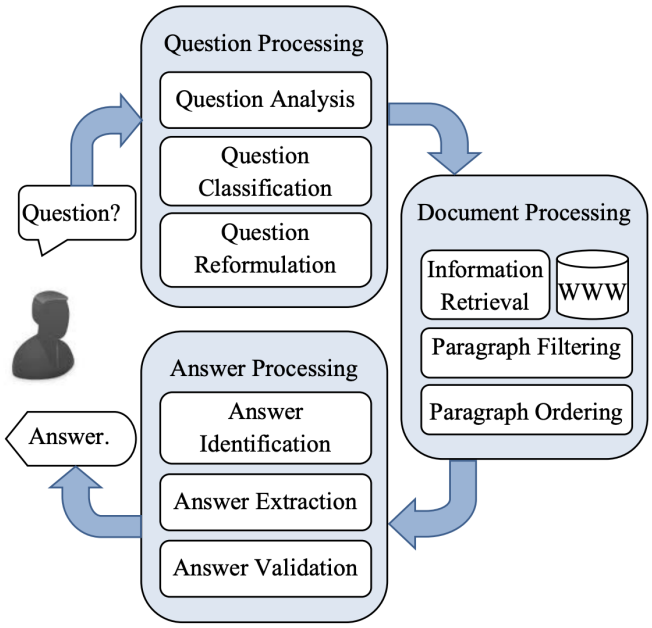

Figure 2.2: Question Answering System Architecture

The general functions for each of the module are illustrated as follows [3]. Question Processing module, taking the query information as the input, performs analysis on the question and then processes the question through the usage of some representation of the required information. As for Document Processing module, relying on the information retrieval sub-systems, it creates a collection of candidate ordered documents (or paragraphs) that may potentially contain the correct answers. Making reference to these collection of paragraphs, Answer Processing module will identify, extract and validate the potential answers to the given questions.

Given the topic of this work (i.e. answer generation), components related to *Answers* will attract more attentions. Specifically, special emphasize will be put on the following answer-related components of QA systems: 1) Answer Extraction; 2) Answer Selection and 3) Answer Generation. Moreover, the existing methodologies applied on each of the component will be discussed.

2.4 Answer-related Components in QA systems

As mentioned before, three answer-related components in the spectrum of QA systems will be covered.

2.4.1 Answer Extraction

In [55], a model for locating answers through exploiting surface text information employing manually crafted surface patterns was proposed. However, the hand-crafted patterns suffered from the poor recall. Therefore, many researchers devised to obtain text patterns automatically [82]. Meanwhile, in order to enhance the patterns, [48] proposed an approach to gather long-distance dependencies by exploiting linguistic structures. In contrast, rather than employing surface text information using patterns, other researchers exploited the named-entity methods to locate the answer [63].

2.4.2 Answer Selection

In the context of open domain QA, answer selection is a significant task. In particular, answer selection is the task to identify the correct answer from a collection of candidate answers [31]. As demonstrated in Figure 2.1, *Answer Selection* is applied after *Document Retrieval* phase such that sentences which get selected are the most relevant to the questions within the spectrum of the retrieved documents.

Also, it is well noted that previous work for answer selection largely relied on feature engineering, linguistic techniques or external resources [78, 23]. Recently, as the surge

of deep learning techniques, many deep learning based methods have been presented for the problem [6, 73]. As a result, the deep learning based approaches outperform the traditional methods.

2.4.3 Answer Generation

Natural answer generation (NAG) obtained greater popularity in real-world Knowledge Base QA systems [79]. Current NAG applications are able to automatically generate natural language response using structured KB. The aims of knowledge base QA systems center around providing exact answers using "pre-seen" information in knowledge bases, while the objective of general QA system is to provide answers for questions expressed in natural language [15].

Previous focus of Knowledge Base QA system was to analyze questions and then select candidate words, phrases or knowledge bases (KB) to return back to the users [11, 74]. In other words, the responses were usually factoid and these answers are usually in the forms of short spans of text (e.g. named entities, dates, etc.) [42]. Nevertheless, users prefer answers correctly replied in a more natural way [22]. Compared with systems providing only factoid answers, QA systems generating natural answers are user-friendlier and more desirable [79]. To produce natural answers, dialogue systems, for example, deal with this issue by employing the template-based answer generation [42]. However, these systems, although appropriate for a domain restricted task, are too restricted for domain independent systems (e.g. open domain QA) [42]. Therefore, for the purpose of presenting the users with a user-friendlier experience, [42] devised a full-length answer generator that can be deployed in most QA systems. In particular, the system can generate a full-length answer given a question along with the extracted answer (regardless of being factoid or not) without having to rely on the documents, where the answers got extracted.

In [22], "natural answer" is defined as the natural response in daily conversations for answering factual questions. These questions are usually expressed in the form of complete

natural language sentences. Thus, the NAG system needs to 1) parse the questions properly; 2) retrieve relevant documents from Knowledge Base and 3) generate a correct and natural response. Previous methods deployed message-response patterns to generate natural answers [37]. Nevertheless, such method suffered from pattern engineering, in which case manual annotations are so costly for training data. Also, according to [22], pattern engineering tend to have low coverage that is difficult to deal with varieties of linguistic patterns in distinct domains. Instead, [22] presented a paradigm trying to consider QA in an end-to-end framework. The outcomes of this design is that the complex QA process (i.e. Question Analyzing, Document Retrieval from Knowledge Base and Natural Answers Generation) can be resolved altogether.

According to [79], NAG serves as an end-to-end Knowledge Base Question Answering system, which is trained directly on question-answer pairs. These pairs are easier to collect from the internet. For the purpose of generating natural answers, NAG adopts a sequence-to-sequence (Seq2Seq) learning framework [68]. Besides, NAG also deploys the copy mechanism [19] and KB retrieval mechanism. With the help of these practical approaches, NAG can be employed at different domains.

2.5 Medical Domain Natural Answer Generation

With the surge of techniques applying deep learning approaches in Natural Language Processing (NLP), QA has attracted great attentions from NLP communities. For the past a few decades, the domain of research in QA tasks has been expanded. Among these domains, medical QA serves as one of the most arresting NLP applications [61]. In the real-world application of medical domain QA system, patients, for example, may post questions regarding his or her health status onto an online QA platform, where the natural answers can be generated with response to the patients' queries. This is applicable and valuable to use especially in a couple of rural areas, where medical resources are limited and visiting doctors on site is less affordable [61].

Nevertheless, with the outburst of medical information sources, locating the most related document resources for generating accurate answers to certain medical related questions is demanding and time-consuming [30]. Given a medical related question expressed in natural language, a robust medical QA system is required to automatically discover the most relevant document resources and generate the correct answers. In [5], generating a precise and natural answers (i.e. summary) to the corresponding biomedical questions serves as the main focus of the BioASQ challenge, which is a large-scale QA challenge in the domain of medication.

Traditional methods for generating answers to medical related questions center around two branches: extractive and abstractive [30]. The former technique (i.e. extractive) is able to generate grammatically correct answers (or summaries) by grouping up relevant sentences. Nevertheless, given the nature of the extractive method, issues related to readability and coherence always occur [34]. Specifically, answers, generated using extractive method, patch the most relevant text units together from different documents. Hence, they are often not readable. To overcome this drawback, [34] presented to conduct the re-ordering of the selected sentences and to incorporate words to sentences to formulate a more coherent answer summary. As for the abstractive method for generating ideal answers to medical related questions, it extracts related data information from the original document. This information is then used to generate natural answer and summary [30]. Even though abstractive techniques are more concise, the generated answers sometimes suffer from grammar error [30]. Therefore, [30] proved that the combination of both extractive and abstractive approaches can bring about more robust performances on the natural answer generation task.

2.6 Existing Data Corpus for Training a Medical QA Systems

In general, QA datasets are categorized into two classses: machine comprehension (MC) deploying unstructured texts, and Knowledge Bases QA[43]. Specifically, given a referenced document, the objective of MC systems is to return answers to any questions related to the document. Advances have been made in crowd-sourcing as well as search engines such that it brought about a surge of large-scale MC datasets for factoid QA [54]. On the contrary, Knowledge Base QA datasets are literally restricted by the specification of annotated question and logical form (LF) pairs [9]. In particular, the LFs are utilized to obtain answers from a schema [7].

In the remaining part of this section, several existing medical related QA datasets for training a robust medical QA system will be discussed.

2.6.1 emrQA: A Large Corpus for QA on Electronic Medical Records

In retrospect, automatic QA has made huge progress with the help of several open-domain and machine comprehension systems built on the basis of large-scale annotated datasets [76, 15]. Nevertheless, few automatic QA-related research has been conducted in the clinical domain, where physicians need to query information about a patient from his or her electrical medical records (EMR) for supporting the clinical decision making [12], due to insufficiency of large-scale clinical QA related datasets [43]. In most cases, these EMRs are semi-structured and they contain free-text medical notes. As a result, it may take so much effort for physicians to seek for appropriate answers to questions related to the medical record of a certain patient. Also, it is highly probable that physicians are not able to extract the expected answers they want from the EMRs [71]. Moreover, in the past, because of the lack of large-scale clinical QA datasets, there is no general well-performing system to answer questions asked by physicians on a patient's EMR [53]. Fortunately, in

2018, a recent QA corpus for EMR-related research comes into rescue. In [43], emrQA was created as a large-scale dataset for training a clinical QA system. It was built by employing a novel generation framework requiring minimum expert involvement. Specifically, existing annotations for other clinical NLP tasks (i2b2 datasets[1]) were re-purposed so that the annotations serve as credible "experts" in generating answers in response to patient specific queries [20]. According to [62], creating intelligent and precise models is really significant in medical domain. Hence, [43] issued large-scale gold annotations (i.e. questions, logical forms, and answers) for the purpose of providing supervision to the learning process. Also, similar to [67], logical forms, generated in emrQA from a structured Knowledge Base, were used to collect answers and allowed users to train well-performing models, which can justify the answers using corresponding logical forms. In the end, the first publicly available patient-specific electronic medical records QA dataset was proposed. Note that the dataset contains 400,000 QA pairs along with 1 million question-logical form pairs.

2.6.2 ChiMed: A Chinese Medical Corpus for Question Answering

Usually, models trained in a general domain are able to be migrated to be trained in an emergent target domain. However, often there is great decline with regard to the performances of these models owing to the domain mismatch [72]. As mentioned in the previous sections, one of the alternatives to boost the performances of these QA systems is to train the models using large-scale domain-specific datasets. Nevertheless, such datasets are difficult to retrieve, especially in medical domain [72]. Specifically, with the emergence of online forums for medical related question and answering services, requirements from users to ask for their corresponding health concerns are increasing. Therefore, it is tempted to build up datasets using data from these forums. One of the major concerns for this temptation is that the quality of the derived datasets may not be guaranteed due to the occurrences of some noise in the forum.

[1] https://www.i2b2.org/NLP/DataSets/

Hence, [72] built up a Chinese medical QA dataset (i.e. ChiMed) through crawling medical data from a credible Chinese online medical discussion forum, where the questions from the users are resolved by authorised physicians. In ChiMed, Questions and Answers are paired with each other (i.e. QA pairs). Besides, the corpus also contain the key information of the questions (e.g. titles, key phrases and etc.), the name of the authorised physicians answering the questions and etc [72]. The statistics for ChiMed is shown in the Figure 2.3.

# of As per Q	# of Qs	% of Qs
1	5,517	11.8
2	39,098	83.7
≥3	2,116	4.5
Total	46,731	100.0

Figure 2.3: Statistics for ChiMed Data Corpus

As shown in the table, there are 46731 questions in the ChiMed, each of which can have one or more answers. Indicated in [72], ChiMed can be applied in many NLP applications, including text classification, text summarization and question answering (QA). In the task of medical domain QA, given a biomedical related question, the answer is generated naturally as the response.

2.6.3 MedQuAD

To conduct QA research in clinical and medical domain, [1] initiated and built up the MedQuAD data collection, which contains 47,457 Q-A pairs from accredited medical platforms. Specifically, according to [1], data from MedQuAD corpus is obtained by crawling the website from the National Institutes of Health [2]. Specific topics (such as the name of a disease) along with their synonyms are described in each web page.

In particular, the Q-A pairs are generated automatically by using hard-coded patterns for different websites, which were constructed in [1], according to the textual structure and

[2] www.nih.gov

the organizations of each web page (e.g. the section titles and etc.). Moreover, for each question in the corpus, [1] annotated the associated topic (i.e. the focus) along with the type of the question, specified using the pre-defined patterns.

In [1], different categories for question types annotated in MedQuAD is proposed. Concretely, there are 16 types for Diseases, 10 types for Drugs and one type (i.e Information) for the other named entities (e.g. Procedures, Medical exams and Medical Treatments). The aforementioned question types are further described in details as follows.

- **Diseases**: there are 16 types of questions related to diseases, including Information, Research (or Clinical Trial), Causes, Treatment, Prevention, Diagnosis (Exams and Tests), Prognosis, Complications, Symptoms, Inheritance, Susceptibility, Genetic changes, Frequency, Considerations, Contact a medical professional, Support Groups. The demonstrated questions are shown below:
  - What research (or clinical trial) is being done for DISEASE?
  - What is the outlook for DISEASE?
  - Who is at risk DISEASE?
- **Drugs**: there are 20 types of questions related to drugs, including Information, Interaction with medications, Interaction with food, Interaction with herbs and supplements, Important warning, Special instructions, Brand names, How does it work, How effective is it, Indication, Contraindication, Learn more, Side effects, Emergency or overdose, Severe reaction, Forget a dose, Dietary, Why get vaccinated, Storage and disposal, Usage, Dose. The example questions are demonstrated as follows:
  - Are there interactions between DRUG and herbs and supplements?
  - What important warning or information should I know about DRUG?
  - What to do in case of a severe reaction to DRUG?

- **Other medical entities: Procedure, Exam, Treatments**: there is only one type for this category, Information. The example questions is shown below:
  - What is Coronary Artery Bypass Surgery?
  - What are Liver Function Tests?

The example below shows the Q-A pairs crawled from the accredited medical resources.


```
-<Document id="0000065" source="ADAM" url="https://www.nlm.nih.gov/medlineplus/ency/article/000321.htm">
  <Focus>Acromegaly</Focus>
 -<FocusAnnotations>
    <Category>Disease</Category>
   -<UMLS>
    -<CUIs>
      <CUI>C0001206</CUI>
     </CUIs>
    -<SemanticTypes>
      <SemanticType>T047</SemanticType>
     </SemanticTypes>
     <SemanticGroup>Disorders</SemanticGroup>
   </UMLS>
  -<Synonyms>
     <Synonym>Somatotroph adenoma</Synonym>
     <Synonym>Growth hormone excess</Synonym>
     <Synonym>Pituitary giant (in childhood)</Synonym>
   </Synonyms>
 </FocusAnnotations>
-<QAPairs>
  -<QAPair pid="1">
     <Question qid="0000065-1" qtype="information">What is (are) Acromegaly ?</Question>
    -<Answer>
       Acromegaly is a condition in which there is too much growth hormone in the body.
     </Answer>
   </QAPair>
  -<QAPair pid="2">
     <Question qid="0000065-2" qtype="causes">What causes Acromegaly ?</Question>
    -<Answer>
       Acromegaly is a rare condition. It is caused when the pituitary gland makes too much growth hormone. The pituitary gland is a pea-sized endocrine gland located at the base of the brain. It controls, makes, and releases several hormones, including growth hormone. Usually a noncancerous (benign) tumor of the pituitary gland causes the gland to release too much growth hormone. In children, too much growth hormone causes gigantism rather than acromegaly.
     </Answer>
   </QAPair>
  -<QAPair pid="3">
     <Question qid="0000065-3" qtype="symptoms">What are the symptoms of Acromegaly ?</Question>
    -<Answer>
       Symptoms of acromegaly may include any of the following: - Body odor - Carpal tunnel syndrome - Decreased muscle strength (weakness) - Decreased peripheral vision - Easy fatigue - Excessive height (when excess growth hormone production begins in childhood) - Excessive sweating - Headache - Hoarseness - Joint pain, limited joint movement, swelling of the bony areas around a joint - Large bones of the face - Large feet (change in shoe size), large hands (change in ring or glove size) - Large glands in the skin (sebaceous glands) - Large jaw (prognathism) and tongue (macroglossia) - Sleep apnea - Thickening of the skin, skin tags - Widely spaced teeth - Widened fingers or toes, with swelling, redness, and pain Other symptoms that may occur with this disease: - Colon polyps - Excess hair growth in females (hirsutism) - Type 2 diabetes - Weight gain (unintentional)
     </Answer>
   </QAPair>
  -<QAPair pid="4">
     <Question qid="0000065-4" qtype="exams and tests">How to diagnose Acromegaly ?</Question>
    -<Answer>
       The health care provider will perform a physical exam and ask about your symptoms. The following tests may be ordered to confirm diagnosis of acromegaly: - Blood glucose - Growth hormone - High insulin-like growth factor 1 (IGF-1) level - Spine x-ray - MRI of the brain, including thepituitary gland - Echocardiogram - Prolactin
     </Answer>
   </QAPair>
  -<QAPair pid="5">
     <Question qid="0000065-5" qtype="treatment">What are the treatments for Acromegaly ?</Question>
    -<Answer>
       Surgery to remove the pituitary tumor that is causing this condition often corrects the abnormal growth hormone. Sometimes the tumor is too large to remove completely. People who do not respond to surgery may have radiation of the pituitary gland. Medications are used after surgery. Some patients are treated with medicines instead of surgery. After treatment, you will need to see your health care provider regularly to make sure that the pituitary gland is working normally. Yearly evaluations are recommended.
```


Figure 2.4: Sample Question Answer pairs in MedQuAD

2.7 Related Task Study: Text Generation

Text Generation covers a broad collection of NLP tasks, which aim to generate natural language given input data and representations [27]. In this respect, Natural Answer

Generation (NAG) is similar to Text Generation in that the goal of NAG is to generate natural answers from input question sentences. Therefore, we can approach NAG using alike methodologies adopted in Text Generation. With the help of modern advanced deep learning techniques, rapid progress has been made in many of sub-tasks of this active area. In particular, the methods used in two sub-tasks of Text Generation, namely machine translation [8] and text summarization [26], will be discussed in detail.

2.7.1 Machine Translation

In general, the job of a Machine Translation (MT) system is to generate in one language the meaning expressed by the text in another language [8]. Initially, MT was approached using a statistical method, where the goal is to maximize the probability *P(Source|Target)*. Note that *Source* and *Target* denotes the source sentence and target sentence, respectively. When deep learning is applied in MT, instead of using the traditional statistical method, Neural Machine Translation (NMT) becomes a better and reasonable choice given its power at producing more accurate translations [64]. In particular, RNNs and LSTMs [25] have been adopted in NMT to train the network to convert the source sentence expressed in one language to target sentence expressed in another language, without having to use large database of rules [64]. Besides, in NMT, *Word Alignment* is also incorporated into the network so that the generated target translation is more coherent and precise [84].

Moreover, with the prevalence of Sequence-to-Sequence (Seq2Seq) models [68], NMT has achieved significant performance boost, outperforming classic phrase-based [10] systems. Also, the encoder-decoder paradigm has proved to be effective when interacting through a soft-attention mechanism [4]. Much more recently, a Transformer model [75] was proposed, on the basis of a self-attention mechanism [45] and feed-forward connections. This Transformer model further advances the field of NMT with respect to its quality of translation and convergence speed. Note that the training data for NMT systems is in the form of several source-target pairs, which aligns with the idea to have Question-Answer pairs for training NAG system.

2.7.2 Text Summarization

As mentioned in section 2.5, extractive and abstractive methods are two classic approaches for medical answer generation [30]. Similarly, to approach Text Summarization, extractive and abstractive methods are also applicable. Specifically, extractive Text Summarization (eTS) aims to generate a brief summary via the extraction of proper collection of sentence clips from original one or more documents [14], whereas abstractive Text Summarization (aTS) generates new summarization sentences that may not present in the original text [47]. Due to this distinction, we will put emphasize on aTS because it internally aligns better with the goal of NAG, which is to generate the target answers that may not exist in the original source questions.

There are several effective deep learning based approaches for aTS. For instance, neural network models proposed in [39], which adopt the encoder-decoder framework associated with attention mechanism [75], can generate decently good abstractive summaries. In particular, the input sequences of these summarization systems consist of just one or two sentences, in which case NAG also applies [3]. Besides, pre-trained language models (e.g. BERT [13]) has also proven to be effective for aTS, where BERT is used as the pre-trained encoders in the network [36]. In this way, semantic representations of the input sentences can be captured and facilitate the generation of summaries with good quality [36].

2.8 Evaluation Metrics: BLEU and ROUGE Scores

In order to properly evaluate the performance of NAG models, inspired by Machine Translation (MT) and Text Summarization (TS), we explored two metrics: BLEU score and ROUGE score.

[3]Note that the question sentences from existing Q-A pairs in medical QA dataset are all very short

2.8.1 BLEU score

BLEU stands for **Bi**-**L**ingual **E**valuation **Understudy**, which was initially proposed in [44]. Originally, BLEU was initiated as an automatic evaluation metric for MT systems so that expensive human evaluations can be avoided [44]. Specifically, in traditional evaluation of MT systems, BLEU can be measured by comparing n-grams of the candidate translation with the n-grams of the human gold reference translation and ultimately count the number of matches [44]. Therefore, the more the matches are between candidates and references, the better the candidate translation is.

In a stricter setting for calculating BLEU scores, a reference word is supposed to be considered as exhausted after a matching candidate token is found [44]. Following this intuition, the calculated BLEU score is represented as the *modified n-gram precision* [44], where one would counts up the total number of n-grams that appear in both candidate and reference translations and then divides by the total number of n-grams in the candidate translation[4]. Ultimately, given the generated candidates and the corresponding gold references, four different BLEU scores (i.e. BLEU-1, BLEU-2, BLEU-3 and BLEU-4) can be calculated, reflecting the average BLEU precision results in terms of uni-grams, bi-grams, tri-grams and 4-grams, respectively. The utility we adopted for calculating BLEU scores is `fairseq-score`, which is a built-in utility program in `fairseq`[5] library.

Moreover, the expected candidate sentences should be neither too short or too long [44]. In other words, the generation system should issue penalties on both too-short and too-long candidates. To some extent, applying *modified n-gram precision* has already penalized longer candidates, since the more the spurious words are in the candidates that do not appear in the references, the lower the precision score will be. As for candidates that are too short, a brevity penalty (BP) is introduced so that candidate sentences must match the reference sentences with respect to length, word choice and word order [44]. Note that BP is also calculated using `fairseq-score`. Specifically, when BP equals to 1, we would

[4]Note that uni-grams, bi-grams, tri-grams and 4-grams are counted separately
[5]https://fairseq.readthedocs.io/en/latest/index.html

have a perfect match between candidates and references in length. Reversely, a zero BP indicates that the generated candidates are way too short.

2.8.2 ROUGE score

Recall-**O**riented **U**nderstudy for **G**isting **E**valuation (i.e. ROUGE) serves as an automatic measurement for the quality of the system-generated summaries, referring to other standard summaries created by humans [35]. Specifically, *ROUGE-N* (i.e. N-gram co-occurrence statistics) and *ROUGE-L* (i.e. Longest common subsequence) were devised as two primary metrics for ROUGE measurements [35].

In general, *ROUGE-N* measures the recall between a candidate summary and one or more gold standard reference summaries [35]. It was computed by 1) counting up the number of matched n-grams between the candidate summary and references and 2) dividing the number by the total number of n-grams in the reference summaries. It is obvious to notice that *ROUGE-N* is actually a recall-based measurement, while *BLEU*[44] is precision-based.

As for ROUGE-L, at the sentence level, the length of the Longest Common Subsequence (LCS) between two summary sequences determines the similarity between them [35]. The idea is that the longer the LCS is, the more identical the two summaries are to each other. The way to calculate ROUGE-L is to divide the length of LCS by the length of the reference summary [35], reflecting the recall-oriented intuition. The closer ROUGE-L is to 1, the more similar the candidate is to the reference.

To calculate both ROUGE-N and ROUGE-L in a single run, the utility we can use is the Python `rouge` libary [6], which will conduct the sentence-level ROUGE calculation and output the averaged result over all the predicted and gold reference sentences. Specifically, the result will show *ROUGE-1*, *ROUGE-2* and *ROUGE-L* scores.

[6] https://pypi.org/project/rouge/

Chapter 3

State-of-the-Art Deep Learning Techniques for Natural Answer Generation

Deep learning techniques have been greatly deployed in NLP community, given the fact deep learning methods are able to produce state-of-the-art results by employing multiple hidden processing layers for the purpose of learning and capturing the hidden hierarchical representations of text information[83]. Specifically, deep learning based methods have been proven to be successful when applied in the context of text generation (e.g. machine translation, text summarization, answer generation and etc.)[81]. Conceptually, when applied in text generation tasks, deep learning based models are trained to be end-to-end, consisting of an encoder and a decoder [81]. The encoder takes in charge of producing hidden representations of the source text information, while the decoder is able to generate the corresponding targets given the input source [68].

In this work, we shed all lights on the application of deep learning in the task of Natural Answer Generation (NAG). Specifically in this section, we detail some of the state-of-the-art deep learning based approaches for NAG. Note that the following techniques may not directly address the problem of NAG, but their model architectures can also be applicable

for training NAG systems.

3.1 LSTM-Seq2Seq: Sequence to Sequence Learning with LSTM as Encoder and Decoder

With the empirical success of Deep Neural Networks (DNNs), whenever large-scale training data is available, DNNs can produce excellent results on distinct learning tasks [69]. However, given an input sequence, DNNs fail to map from the input sequence to an output sequence [69]. Therefore, an end-to-end network structure is proposed in [69] in order to address the problem of sequence learning. In particular, a multilayered Long Short Term Memory (LSTM) [25] is adopted as the encoder, which can learn the representations from the inputs by mapping the input sequence into a vector with fixed dimensions. Besides, another similar structured LSTM is used as the decoder of the model, generating the target sequence from the input representation vector. Essentially, the idea is to 1)enable the LSTM encoder to read in the input sequence, one timestep at a time, in order to acquire the vector representation that is fixed-dimensional and 2) utilize another LSTM decoder to generate the output sequence from the learned encoded vector [69]. The model structure is demonstrated in Figure 3.1, where the input sequence is *"ABC"* and the output sequence is *"WXYZ"*.

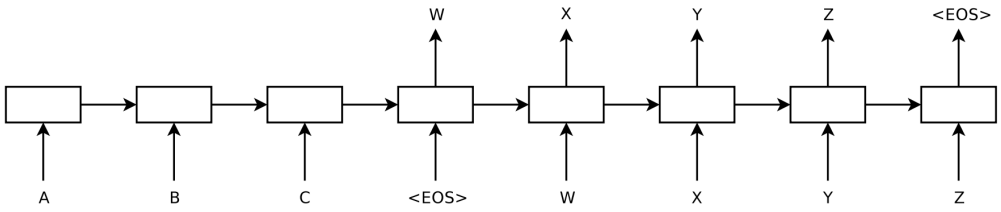

Figure 3.1: Model architecture for sequence to sequence learning

In addition, a novel attention mechanism, introduced in [17] and further refined in [4], is also incorporated in [69] so that the sequence neural network can learn to put emphasize on different part in the input in order to generate correct output.

The model is evaluated on WMT-14 English-French Machine Translation dataset. The

performance was close to state-of-the-art methods on the same test dataset.

3.2 BertSum: Text Summarization with Pre-trained Encoders

Recent improvements for the performance of a variety of NLP systems attribute to the latest emergence of pre-trained language models. **B**idirectional **E**ncoder **R**epresentations from **T**ransformers (BERT) [13], for example, exemplifies the strengths of pre-trained encoders in lots of NLP tasks (e.g. text generation). In order to fully reveal the effectiveness of BERT in text generation, [36] proposed a general pipeline and framework to apply BERT in the context of text summarization as a showcase (note that text summarization is also a sub-branch of text generation [27]). In this section, we focus on the proposed framework by [36] for applying BERT into text summarization, as we can later re-use the same framework for NAG.

In [36], an innovative BERT-based document-level encoder is introduced. By deploying this encoder to a document consisting of several sentences, the semantics of this document can be learned and expressed and the representations for all the sentences from the document can also be obtained. In particular, two classes of summarization algorithm are examined: extractive summarization and abstractive summarization. Given the nature of the NAG systems, which, given a source sentence, will normally generate some new phrases or use words that may not exist in the original text, we focus more on the abstractive summarization [47].

Combining the representations of both words and sentences into a large Transformer [75], BERT is able to be pre-trained on a large-scale data corpus, with masked language modelling as an unsupervised objective [13]. Furthermore, it can also be fine-tuned on some downstream low-resource (i.e. resources containing limited amount of training data) tasks. After the pre-training stage, the models are usually deployed as encoders for some lan-

guage understanding problems [13]. Specifically for the abstractive summarization model in [36], an encoder-decoder framework is adopted, incorporating both the pre-trained BERT encoder and a Transformer decoder that is randomly initialized [75]. Besides, due to the fact that the BERT encoder is pre-trained and the Transformer decoder is trained from scratch, a novel fine-tuning schedule is devised to break apart the optimizers of the encoder and the decoder. The model architectures for both the original BERT and BERT for Summarization in [36] are illustrated in Figure 3.2. As you can visualize in this figure, the input document consisting of several sentences is represented on the top sequence. Then, following three types of embeddings for each token will be summed up in order to be fed into the multiple bidirectional transformer layers, which will eventually generate contextual representations for each token. Compared to the original BERT, BERT for summarization embeds several *CLS* icon to capture sentence representations.

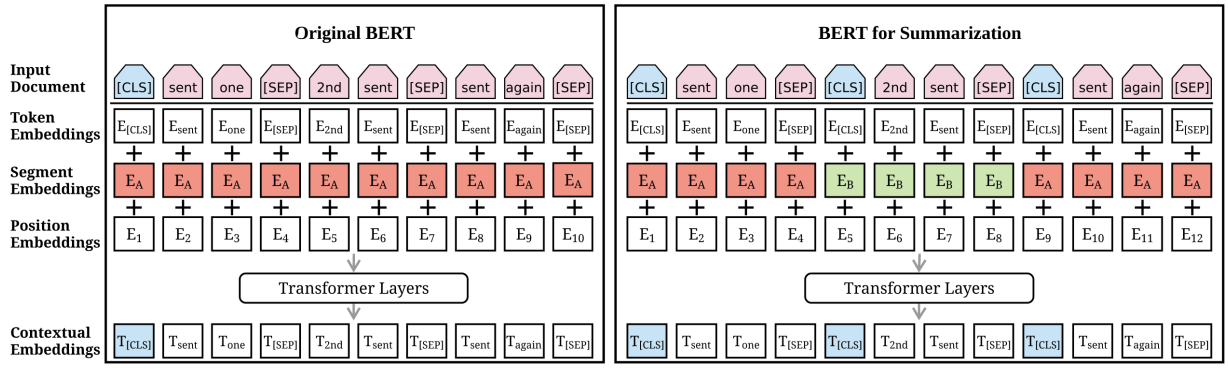

Figure 3.2: The Architectures for original BERT and BERT for Summarization

The proposed model in [36] is evaluated on three benchmark datasets: CNN/DailyMail news highlights dataset [24], New York Times corpus [57] and XSum [40]. On all of the three datasets, the model achieves state-of-the-art results.

3.3 MASS: Masked Sequence to Sequence Pre-training

Transferring knowledge learnt from adequate-resource pre-training tasks to relatively inadequate-resource downstream tasks has proven to be successful to produce state-of-the-art results in many language understanding tasks [65]. For instance, BERT (Bidirectional

Encoder Representations from Transformers)[13] was proposed to reveal the strengths of pre-training and fine-tuning in NLP tasks and has achieved remarkably good performance.

Stimulated by the great success of BERT, [65] initiated **MA**sked **S**equence to **S**equence pretraining (MASS) for language generation on the basis of the end-to-end encoder-decoder neural network framework. In general, MASS reformulate fragments of a sentence from the remaining part of the sentences (unmasked) by employing the encoder-decoder framework [65]. Specifically, the encoder of MASS takes as input the sequence consisting of several continuing randomly masked fragments, while the decoder attempts to reconstruct these masked fragments conditioned on the encoder representations. As a result, by jointly pre-training the encoder and decoder, MASS is capable of developing the ability to extract meaningful representations from training instances and conduct language modelling on the basis of these representations. Further fine-tuning on various inadequate-resources text generation tasks (e.g. machine translation, text summarization and natural answer generation) can be conducted in order to achieve state-of-the-art results.

Compared to BERT, which only pre-trains the encoder, MASS is considerably carefully structured to jointly pre-train both the encoder and decoder [65]. The steps for the joint pre-training are illustrated as follows (Note that the encoder-decoder framework for MASS is also shown in Figure 3.3). First of all, given a masked input sequence, MASS forces the encoder to interpret the meaning of the unmasked tokens (i.e. x1, x2, x7 and x8) through predicting the masked fragments of the sentence on the encoder side. In this way, the masked tokens in the decoder side can be predicted precisely. Secondly, through masking the input tokens in the decoder side that are not masked in the encoder side (i.e. x1, x2, x7 and x8), MASS forces the decoder to predict the next token by relying more on the representation of the source instead of the previous tokens in the target. Following the aforementioned two steps, joint pre-training between encoder and decoder in MASS can be facilitated. Also, according to [65], MASS jointly pre-trains encoder and decoder using only labeled data, and thus can be deployed to most language generation tasks (in our

case: NAG tasks). In addition, MASS also incorporates attention mechanism[4] between encoder and decoder so that when predicting the current token in the decoder side, it knows which source representation to focus on.

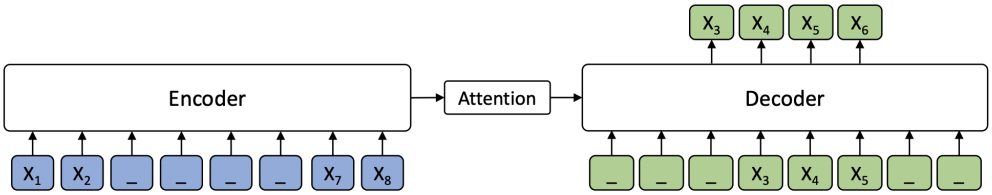

Figure 3.3: The encoder-decoder framework for MASS

Also, it is important to note that MASS adopts Transformer [75] as the backbone for the sequence-to-sequence learning and is pre-trained on the WMT monolingual corpus [1], which is a large-scale rich-resource language corpus originally designed for machine translation. After further fine-tuning on three low-resource text generation tasks (i.e. machine translation, text summarization and conversational response generation), MASS outperforms baselines, which are without pre-training or with different pre-training techniques, and achieves state-of-the-art results.

3.4 PoDA: Denoising based Sequence-to-Sequence Pre-training

In [77], **P**re-training **o**f **D**enoising **A**utoencoders (PoDA) is introduced as a new sequence-to-sequence (Seq2Seq) pre-training technique. Similar to MASS [65], PoDA is able to learn representations appropriate for the tasks of text generation by going through the pre-training step. There are a variety of distinct pre-training approaches, including BERT (encoder-only pre-training) [13] and OpenAI GPT (decoder-only pre-training) [50]. However, like neither of them, PoDA jointly pre-trains both the encoder and decoder via *denoising* the noise-corrupted texts [77]. Also, when fine-tuning PoDA on other downstream text generation tasks (e.g. NAG), the network architecture in the pre-training stage can remain unaltered, which is also the advantage of this approach. Unlike MASS,

[1] Available from here: http://www.statmt.org/wmt16/translation-task.html

which employs only Transformer as the backbone architecture for pre-training and fine-tuning, PoDA embeds both Transformer and pointer-generator networks [58] into the system. On one hand, as indicated in [75], Transformer is conductive to long-distance dependency modeling and is proven to be empirically successful. On the other hand, pointer-generator network adopts copying mechanism [19], which is also effective to the majority of text generation tasks.

The major feature of PoDA lies in the deployment of *Noising and Denoising* components in the network. Similar to denoising autoencoders [33], the noising component introduces noises into a given input word sequence, while the denoising component attempts to remove the noises and recover the original word sequence via a seq2seq model. Specifically, given an input word sequence S, three noising functions are adopted: tokens in S can be 1) randomly shuffled; 2) deleted or 3) replaced by other tokens [77].

The overall PoDA architecture is illustrated in Figure 3.4, where an example is provided suggesting how the model is pre-trained through denoising the noise-corrupted sequence. As you can visualize in Figure 3.4, the original sequence is *"The fox jumps over the lazy dog"*. Subsequently, the three aforementioned noising functions are applied at this input sequence, resulting in the corrupted sequence *"fox fly over the dog lazy."*. Then, the denoising component will attempt to maximize the conditional probability of recovering the original sequence conditioned on having the corrupted sequence. Note that *"bos"* tag stands for the beginning of the sequence, which is a special padding symbol. As a matter of fact, the objective of the denoising part (i.e. maximizing the conditional probability of recovering the original sequence given the corresponding corrupted sequence) indicates that PoDA is able to incorporate both encoder-only pre-training and decoder-only pre-training techniques. Resembling BERT, when computing the loss function, PoDA also adds a mask to the target sequence. Additionally, a small amount of positions where the tokens are not corrupted are kept, in order to have the model learn to copy from the original input when necessary.

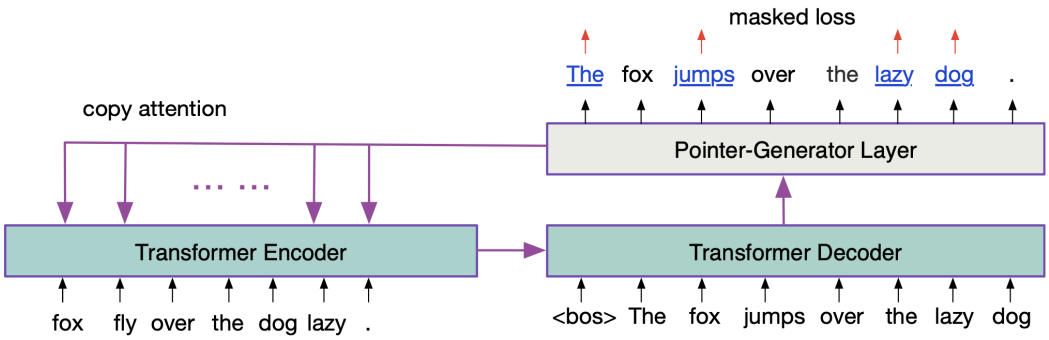

Figure 3.4: Model architecture for PoDA

Also, it is well noted that PoDA is pre-trained on two large-scale adequate-resource text corpora: English Wikipedia [2] and the Billion Word Benchmark [3]. After fine-tuning on two text generation tasks (i.e. abstractive summarization and grammatical error correction), PoDA is able to achieve the state-of-the-art results.

3.5 DIALOGPT : Large-Scale Generative Pre-training

Large-scale rich-resource pre-training based on Transformer has brought about massive advances in NLP community [51, 52]. OpenAI GPT [50] exemplifies that by pre-training on large-scale adequate-resource datasets, Transformer-based models are capable of capturing long-term dependencies within the texts as well as decoding and generating sequences that aligns with human expressions (i.e. similar to what human would express). Such generated results are coherent in the sense that they are fluent and rich in content. Nevertheless, challenges still exists for these Transformer-based models in terms of generating neural response, consisting of natural-looking textual responses that are different from any training instance and are responsive to the prompt so that they can deal with informal or noisy queries frequently occurred in human conversations in the form of textual chat [85].

For better generating neural responses, DIALOGPT is proposed, extending the nice prop-

[2] Available from here: https://dumps.wikimedia.org/
[3] Available from here: http://www.statmt.org/lm-benchmark/

erties of OpenAI GPT [85]. Similar to OpenAI GPT, DIALOGPT is carefully designed to be structred as an autoregressive model [85]. The model structure of DIALOGPT is based on multi-layer Transformer, which enables the model to preserve the long-term dependency information across time and improves the consistency of content in the texts [50]. Specifically, DIALOGPT is trained on the large-scale conversation pairs retrieved from Reddit dialogue comment chains, which spans from 2005 until 2017. Consequently, this allows DIALOGPT to acquire a finer-grained joint probability distribution of *P(Target, Source)* within the dialogue flow in the sense that the trained model is capable of better characterizing language data distribution of human.

In addition, a scoring function of maximum mutual information (MMI) [32] is also implemented in [85] in order to deal with the problem of generating tedious and non-comprehensive responses. Ultimately, the pre-trained model was evaluated on two large dataset: a benchmark dataset (DSTC-7) and a 6K dataset for testing extracted from Reddit comment streams. In both automatic and human evaluations, DIALOGPT achieves state-of-the-art testing results.

Chapter 4

Implementation

As illustrated in the last chapter, the characteristics and nature of five state-of-the-art deep learning based techniques for Natural Answer Generation (NAG) were revealed. In order to experiment with all the techniques, we dive into the implementation details for how to use their systems. In this chapter, the code implementation of the five techniques will be discussed.

4.1 LSTM-Seq2Seq

This sequence-to-sequence (Seq2Seq) based system is implemented using tensorflow[1]. Specifically, in the code base, *task summarization* is experimented for the purpose of showing how the model architecture works. Essentially, the system is built on the basis of a encoder-decoder model with attention mechanism. Moreover, GloVe [49] is deployed for pre-training word vectors and initializing word embeddings, which enables better word representations in forms of fixed-dimensional vectors to be fed into the LSTM encoder. As demonstrated in section 3.1, both the encoder and decoder is implemented using LSTM cells. For the encoder side, LSTM cells are used together with *stack_bidirectional_dynamic_rnn*. As for the decoder, the basic LSTM decoder is used during training phase, while the beam search decoder is employed during testing and inferring. Note that the deployment of the beam search enables the decoder of the model

[1]Code implementation is here: https://github.com/dongjun-Lee/text-summarization-tensorflow

to output the best combination of words for the generation task. The general pipeline for training and testing this Seq2Seq model will be illustrated in details.

4.1.1 Data Preparation

The program for data preparation is *prep_data.py*, which is located in the project root directory. There are two alternatives for the stage of preparing data for training: 1) with GloVe pre-trained word embeddings; 2) without GloVe pre-trained word embeddings. Running the following command on the terminal allows us to 1) download the GloVe pre-trained embeddings and 2) create a data directory *sumdata* storing the training and testing data, which is in the forms of source-target pairs. Note that the downloaded GloVe data is also stored in the project root directory.

```
$ python prep_data.py --glove
```

4.1.2 Training

The script for model training is *train.py*. Similar to *prep_data.py*, we can voluntarily choose to use GloVe for training. The training command is shown as follows:

```
$ python train.py --glove
```

Additionally, several hyper-parameters (e.g. number of hidden layers, beam width for beam search, embedding size, learning rate and etc.) can be configured in order to customize our model training configuration. Deleting the *–glove* flag will allow us to train the model without the help of GloVe pre-trained word embeddings.

4.1.3 Testing

The testing script for model evaluation is encoded in *test.py*. Given the source testing data, the target data can be generated automatically via running the following command:

```
$ python test.py
```

Note that the generated sentences are stored in a *result.txt* file.

4.1.4 Testing the pre-trained model

According to the project specification, a model[2] has already been trained on the Gigaword dataset [3]. Therefore, we can directly first download the trained model and run it on the test dataset to evaluate the model performance. The command to do so is as follows (after you download the trained model):

```
$ unzip pre_trained.zip
$ python test.py
```

4.2 BertSum

Analogous to Seq2Seq with LSTM encoder and decoder, the implementation of the Pre-trained encoder (e.g. BERT) based system also showcases the effectiveness of the model on *text summarization* task, where the system is named as *BertSum*[4]. Specifically, *BertSum* is built on the basis of PyTorch, OpenNMT[29], and the *"bert-base-uncased"*[5] version of BERT. Several additional packages are needed, including pytorch_transformers, tensorboardX, and multiprocess. The general schedule for training *BertSum* will be illustrated in the following sub-sections.

4.2.1 Data Preparation for Benchmark Datasets

According to [36], *BertSum* is trained and evaluated on three benchmark dataset, including CNN/DailyMail [24] and XSum [40]. In particular, for replicating the model performance mentioned in [36] on the two benchmark datasets, the data preparation steps are discussed respectively. For XSum, the pre-processed data can be directly downloaded [6]. As for CNN/DailyMail, two options are offered: 1) directly download the processed data [7]; 2) process the data by ourselves. Note that all the downloaded BERT data should be

[2]The model can be downloaded from here: https://drive.google.com/file/d/1V8pS1eoiv51wfiVp2rOB7IvJ5PeQs2n-/view
[3]Available from here: https://catalog.ldc.upenn.edu/LDC2012T21
[4]Code available from here: https://github.com/nlpyang/PreSumm
[5]Available from here: https://git.io/fhbJQ
[6]Available from here: https://drive.google.com/file/d/1BWBN1coTWGBqrWoOfRc5dhojPHhatbYs/view
[7]Available from here: https://drive.google.com/file/d/1DN7ClZCCXsk2KegmC6t4ClBwtAf5galI/view

stored into *bert_data* directory.

The processing of the raw data in CNN/DailyMail essentially performs the following three actions: 1) tokenize the raw data using Stanford CoreNLP toolkit [8] and split the raw file into several separate sentences; 2) format the sentences into JSON and store them into *json_data* directory; 3) generate the binary files given the JSON files. Note that in step 2, the source and target texts are tokenized with the subwords tokenizer of BERT. Besides, after step 3, the generated binary files are going to be directly used in model training step.

4.2.2 Model Training

As explained in section 3.2, we put more emphasize on the abstractive summarization setting. Specifically, a dropout with probability 0.1 was adopted, which is positioned before all linear layers. Also, a smoothing factor of 0.1 was applied for label smoothing [70]. Besides, the Transformer-based decoder contains 768 hidden units and for all feed-forward layers the hidden size is 2048. In [36], the models were trained for 200,000 epochs.

In addition to BertSum, a baseline system, consisting of Transformer-based encoder and decoder, can also be trained by setting the *-encoder* flag to be "baseline" (Note that the default setting for *-encoder* is "bert").

4.2.3 Model Evaluation and Testing

During decoding, beam search was adopted with the beam width of 5. There are two modes for evaluating the model: validate and test. Setting *-mode* to "validate" enables the program to inspect the model directory and evaluate the model for each saved checkpoints, while "test" mode needs to be associated with *-test_from* flag, suggesting the checkpoint you want to use. It is worth noticing that out-of-vocabulary words were rarely observed because of the deployment of BERT's subwords tokenizer.

[8] Available from here: https://stanfordnlp.github.io/CoreNLP/

4.3 MASS

The implementations details about MASS can be divided into two stages: pre-training and fine-tuning [9]. For pre-training, Transformer [75] is used as the basic model architecture, consisting of 6-layer encoder and decoder with the hidden embedding size being 1024 and feed-forward filter size being 4096. For text summarization task, which is what we are interested, MASS is pre-trained on a large-scale English monolingual data consisting of several source and target sentence pairs. Besides, regarding the optimization method for pre-training, Adam optimizer [28] is adopted with learning rate being 10^{-4}.

After pre-training, MASS can be further fine-tuned on other low-resource text generation tasks. In [65], fine-tuning on text-summarization is showcased. Specifically, the pre-trained model is fine-tuned on different scales of training data, which is from Gigaword corpus (i.e. a benchmark text summarization data corpus) [16]. Gigaword corpus contains 3.8M English article-title pairs, in which [65] takes the article as the input to the encoder (i.e. source) and the title as the input to the decoder (i.e. target).

The code implementation for MASS is based on PyTorch and Fairseq [41], which is an extensible toolkit for sequence modelling. In the code base of MASS for text-summarization [10], pipelines for both pre-training and fine-tuning are provided. In general, the pre-training stage involves four steps: 1) collect the data corpus; 2) Tokenize the corpus using wordpiece vocabulary (Note that pytorch_transformers is needed for generating the tokenized data); 3) Binarize the data for pre-training using "fairseq-preprocess"; 4) pre-train the model via "fairseq-train" command together with several pre-set hyperparameters (e.g. learning rate, optimizer, dropout and etc.). As for fine-tuning, the data needs to be tokenized following the tokenization step in pre-training stage. Also, the wordpiece-level data is generated given the tokenized corpus. Moreover, all the source data will be suffixed with `src`, while the targets will be suffixed with `tgt`. Like pre-training, "fairseq-

[9] Code available from here: https://github.com/microsoft/MASS
[10] Available from here: https://github.com/microsoft/MASS/tree/master/MASS-summarization

preprocess" will generate the required binarized data for the actual model training in fine-tuning step. Then, "fairseq-train" is called with other hyperparameters (e.g. learning rate, max-source-positions and etc.). After training, the best checkpoint will be store in `checkpoint_best.pt` file, which will be used for inference stage together with "fairseq-generate" command. Note that beam search is also deployed with beam width being 5.

4.4 PoDA

The system for **P**re-training **o**f **D**enoising **A**uto-encoders (PoDA) is implemented using PyTorch [11]. Like MASS, fairseq [41] is also deployed for model training and data processing. Similar to MASS, the procedures for pre-training and fine-tuning PoDA model are illustrated respectively. Specifically, PoDA is pre-trained on two rich-resource data corpus: 1) full data in English Wikipedia [12] and 2) the Billion Word Benchmark [13]. Both datasets were pre-processed before the actual pre-training. The network was pre-trained on 4 GPUs with synchronization and data parallelism. In particular, on a single GPU, each batch contains at most 3000 tokens. The model is pre-trained for 14 epochs, with each epoch taking almost two days, on the two datasets for their entirety. In general, PoDA consists of a 6-layer encoder, a 6-layer decoder and a single pointer-generator layer, with in total 97 millions parameters. Also, for each positionwise feed-forward layer, the hidden size is 4096. There are 8 heads used for multi-head attention layers. For vocabulary, top 50k most frequently occurred words were selected, whose embedding dimension were 512. Nesterov Accelerated Gradient optimizer [66] was adopted, with the initial learning rate set to 0.002. Additionally, dropout is set to 0.2 for all self-attention layers, feedforward layers and input embedding layers. The Transformer implementation from *fairseq* was employed to structure the encoder and decoder.

As for fine-tuning PoDA on text generation tasks, the overall network structure from

[11] Code available from here: https://github.com/yuantiku/PoDA
[12] Available from here: https://dumps.wikimedia.org/
[13] Available from here: http://www.statmt.org/lm-benchmark/

pre-training remains unchanged and is re-used, given that both stages (pre-training and fine-tuning) take as input a source sequence of tokens and generate a target sequence as the output. Specifically, at fine-tuning stage, the network is initialized using the parameters trained after pre-training. Besides, the hyper-parameters, scheduling for learning-rate change, early stopping, and gradient clipping are also exactly the same as pre-training. The only difference lies in the data pre-processing step, which is data-specific. The objective function used for fine-tuning is the word-level negative log-likelihood. After each training epoch, the validation loss is computed and the learning rate is halved no matter when the validation loss stops decreasing. Note that the training procedure is terminated once the learning rate is below 10^{-4}. During inference, standard beam search is adopted on the basis of the length-normalized log-likelihood.

In the code implementation, the steps for running the system are illustrated. At first, the input dataset will be binarized and pre-processed. Then, the pre-trained model is loaded and fine-tuned on the chosen dataset. Subsequently, beam search decoding can be performed and the model can be finally evaluated using several matrix, including ROUGE score and BLEU score. Note that the script for pre-training PoDA is not provided.

4.5 DIALOGPT

DIALOGPT is trained based on the architecture used in OpenAI GPT-2 [51], where the generic Transformer model [75] and a stack of masked multi-head self-attention layers are adopted. Specifically, inspired by OpenAI GPT-2 [51], the Transformer architecture of DIALOGPT contains 12-to-48 layer along with layer normalization. Also, since model depth is modified, a separate initialization scheme was devised to account for the depth modification. Besides, Byte-Pair Encodings (BPE) [59] was deployed to tokenize the data corpus. Moreover, DIALOGPT is implemented on the basis of an open source repository: PyTorch-transformer [14]. The code base of DIALOGPT can be found in here: *https://github.com/microsoft/DialoGPT*.

[14] Available from here: https://github.com/huggingface/ pytorch- transformers

The model for DIALOGPT was trained under three different size of parameter setting: 117M, 345M and 762M. In particular, the model details can be observed from Figure 4.1. Note that "B" denotes the batch size for each GPU. The vocabulary employed by DIALOGPT contains more than 50000 entries (50257 in total). Also, 16 Nvida machines were used to train DIALOGPT. Noam was chosen as the learning rate scheduler with 16000 warm-up steps, where the learning rate was chosen on the basis of validation loss. The training procedure stops when no progress is made on validation loss. Due to the difference in parameter size, the small and medium models were trained for 5 epochs, while the large model is only trained for at most 3 epochs.

Model	Layers	D_{emb}	B
117M	12	768	128
345M	24	1024	64
762M	36	1280	32

Figure 4.1: Model specification for DIALOGPT

In the code base of DIALOGPT, pipelines for fine-tuning DIALOGPT using the pre-trained models (small, medium and large) were specified. To begin with, the pre-trained/fine-tuned models need to be downloaded into the project directory. Note that it also allows to download the training Reddit data. Before the start of training, the input data needs to be preprocessed (using `prepare4db.sh`) and converted into the correct format that the following training script can recognize. Subsequently, `prepro.py` is called to further pre-process the input data. The training script is encoded in `LSP_train.py` file. By tweaking arguments passed to the training script, the GPU training can be configured automatically (i.e. training using single GPU or multiple GPUs). The training step can also be further tweaked by adjusting the hyperparameters, including *max_seq_length*, learning rate, optimization steps and etc. Also note that currently the decoding script is not provided due to the problem of toxic generation.

Chapter 5

Experimental Details and Evaluation

As indicated in Chapter 3, five state-of-the-art deep learning based techniques that can be adopted in Natural Answer Generation (NAG) were introduced in details. In this chapter, we experiment with these approaches and specifically applied them to medical NAG task using our own dataset, namely *MedQuAD* [1]. This data corpus initially contains 47,457 Question-Answer pairs crawled from 12 accredited medical service websites [1].

In particular, the rest of this chapter will be organized as follows. In section 5.1, the pre-processing step for the raw data, encrypted in XML format, will be discussed. Then in section 5.2, details about how we train/evaluate these state-of-the-art systems on *MedQuAD* are going to be covered. Lastly, in section 5.3, we will analyze the experimental evaluation results for all systems.

5.1 Pre-processing Raw Data

The raw data of *MedQuAD* is directly downloaded from its GitHub project page [2]. In total, there are 12 directories containing 47,457 Question-Answer pairs crawled from 12 NIH websites (e.g. Cancer.gov, GARD, MedlinePlus Health Topics and etc.). Besides, the set of directories spans over 37 question types (e.g. Treatment, Diagnosis, Information).

[1] https://chiqa.nlm.nih.gov
[2] Available from here: github.com/abachaa/MedQuAD

Within each directory, the files containing Q-A pairs are all encoded in XML format, with several useful annotations to identify the correct matching Q-A pair. However, due to the copyright issue from *MedlinePlus*[3], the answers from 3 subsets (i.e. 10_MPlus_ADAM_QA, 11_MPlusDrugs_QA and 12_MPlusHerbsSupplements_QA) were removed, in which case these three sets of Q-A pairs will not be used in our experiments. As a result, after filtering out the unavailable Q-A pairs, we end up with a dataset containing 12,159 Q-A pairs. Note that even though it is indeed a significant shrink from the original *MedQuAD* dataset, we can still achieve relatively reasonable results adopting the state-of-the-art systems.

The structure for each XML file was demonstrated in Figure 2.4. In order to extract the corresponding Q-A pairs from these XML files, we wrote a script to parse all the files using *ElementTree* module [4]. The annotations we put emphasize on in the XML files are: *QAPairs*, *QAPair*, *Question* and *Answer*. In particular, since each *QAPair* annotation is associated with a `pid`, it can be used to identify each Q-A pair. Specifically, for each extracted Q-A pair from an XML file, a dictionary is created, with the key set to be the combination of the name of the XML file and the `pid` for each *QAPair* and value set to be an object consisting of *Qtype*, *Question* and *Answer*. Ultimately, we exported a JSON file (i.e. MedQuAD.all.json) containing all 12,159 Q-A pairs we need for the experiments. As an example, one instance of Q-A pair stored in the exported JSON file is shown in Figure 5.1. In this way, each Q-A pair can be queried with a unique id.


```
"0000001_1.xml_4": {
    "Qtype": "outlook",
    "Question": "What is the outlook for Adult Acute Lymphoblastic Leukemia ?",
    "Answer": "Certain factors affect prognosis (chance of recovery) and treatment options. The prognosis (chance of recovery) and treatment options depend on the following:
},
```


Figure 5.1: Sample Q-A pair in JSON format

[3]https://medlineplus.gov/copyright.html
[4]https://docs.python.org/2/library/xml.etree.elementtree.html

5.2 Training and Evaluating State-of-the-Art Systems on MedQuAD

In this section, we will discuss the steps for training and evaluating the five state-of-the-art deep learning based systems on our pre-processed *MedQuAD* dataset stored in JSON format. We performed a generic train-valid-test split on the data, with the number of Q-A pairs for each set to be 8170, 2043 and 1946, respectively.

5.2.1 LSTM-Seq2Seq

[Data Preparation]

LSTM-Seq2Seq is built on top of a basic sequence-to-sequence(seq2seq) model, with multilayered LSTMs as both the encoder and decoder. For each Q-A pair, the "Question" in *MedQuAD* can be treated as the source sequence, while the corresponding "Answer" is regarded as the gold target sequence. In the data directory of LSTM-Seq2Seq, namely `sumdata`, a sub-directory `train` is created, containing all the alignment source and target files for train, valid and test set. Specifically, the source files are post-fixed with `.src`, while the target files use `tgt` as the postfix. For example, the 8170 training instances are stored in two alignment files: `train.src` and `train.tgt`. The `train.src` contains all the "Question" sentences, with each sentence per line, while the `train.tgt` contains the corresponding "Answer" sentences, with each answer per line. Note that the questions and answers need to be aligned in the two files. Same logic applies to `valid.src`, `valid.tgt`, `test.src` and `test.tgt`. We wrote the script for generating these alignment files, namely `alignment.py`, in *LSTM-Seq2Seq* directory.

[Model Training]

The training script is encoded in `train.py`. Specifically, we train the LSTM-Seq2Seq model with two different configurations: 1) with GloVe [49] and 2) without GloVe. Note that the GloVe pre-trained word embeddings is downloaded using the following command: `python prep_data.py --glove`. For both configurations, the hyper-parameters setting

remains the same and is shown in Table 5.1.

Hyperparameters	Value Setting
# of hidden nodes	150
# of encoder layers	6
# of decoder layers	6
beam width	10
word embedding size	300
learning rate	1e-3
batch size	64
# of training epochs	100
Maximum src length	50
Maximum tgt length	100

Table 5.1: Hyperparameters for Training LSTM-Seq2Seq

In particular, we name the training configuration with GloVe involved as **LSTM-Seq2Seq-glove**, while the others without GloVe is named **LSTM-Seq2Seq-no-glove**. The training script `train.py` is called together with `--glove` flag for **LSTM-Seq2Seq-glove**. On the contrary, directly running `train.py` will establish the word embeddings from scratch and train the network without the help of GloVe. Besides, as indicated in section 4.1.4, a trained LSTM-Seq2Seq model can be directly downloaded and named as **LSTM-Seq2Seq-trained**, which is regarded as the third configuration for LSTM-Seq2Seq network.

[**Model Evaluation**]

In order to evaluate the model performance of the aforementioned three LSTM-Seq2Seq configurations, namely **LSTM-Seq2Seq-glove**, **LSTM-Seq2Seq-no-glove** and **LSTM-Seq2Seq-trained**, the testing script `test.py` is called to generate the predicted answers given the questions in `test.src` file. For each configuration, using the model trained from training set, a `result.txt` file is generated containing system predictions. Given the prediction file (i.e. result.txt) and gold target sentences (i.e. test.tgt), BLEU and ROUGE score can be respectively computed. Specifically, the evaluation results are shown in Table 5.2. Note that the generated results are stored in `evaluation` directory.

Evaluation Metrics	LSTM-Seq2Seq-glove	LSTM-Seq2Seq-no-glove	LSTM-Seq2Seq-trained
BLEU-1	52.0	52.5	14.2
BLEU-2	25.5	25.8	1.5
BLEU-3	16.4	16.6	0.1
BLEU-4	10.6	10.7	0.0
BP	0.02	0.019	0.000
syslen	91825	91262	10423
reflen	452516	452516	452516
ROUGE-1	0.23	0.23	0.01
ROUGE-2	0.11	0.11	0.00
ROUGE-L	0.29	0.29	0.02

Table 5.2: Evaluation results of LSTM-Seq2Seq

5.2.2 BertSum

As introduced in section 3.2, BertSum is implemented to exemplify the power of BERT [13] on Text Generation, specifically on Text Summarization. As we indicated in the background study, Natural Answer Generation (NAG) is identical to Text Summarization, where "Question" can be used as source and "Answer" can be regarded as target. We reused the network structure in BertSum and trained/evaluated the model on our *MedQuAD* dataset. Similar to LSTM-Seq2Seq, we also had three experiment configurations for BertSum: 1) **BertSum-trained**: a trained BertSum model based on XSum dataset [40]; 2) **BertSum-transformer**: a re-trained BertSum model adopting Transformers [75] as both the encoder and decoder; 3) **BertSum-bert**: a re-trained BertSum model deploying BERT [13] as the pre-trained encoder. We will discuss the pipeline for training/evaluating the three aforementioned model configurations for BertSum in detail.

[Data Preparation]

BertSum-trained First, we downloaded and unzip the pre-processed data directory (i.e. `bert_data_xsum_final.zip`) [5]. This directory contains several `.pt` files, which were further put into the `bert_data` directory inside the project root directory. Next, we created another directory `raw_data`, containing `test.src` and `test.tgt`. These two alignment files were created using the same approach as in LSTM-Seq2Seq.

[5] Available from here: https://drive.google.com/file/d/1BWBN1coTWGBqrWoOfRc5dhojPHhatbYs/view

BertSum-transformer & BertSum-bert The data preparation steps for these two model configurations are the same. First of all, we generated three raw data files (i.e. `train.raw`, `valid.raw` and `test.raw`) based on the train-valid-test split of MedQuAD data. To do it, the script `generate-raw.py` was created. Specifically, each line of these files contain one Q-A pair, separated by a 'tab' key. These raw data were stored in `MedQuAD` directory. Subsequently, for each of the raw data file in `MedQuAD`, the corresponding JSON file was created, where each source (i.e. question) and target (i.e. answer) sentence was stored as a list of tokenized words. The sample JSON format is demonstrated in Figure 5.2. Note that we tokenized the raw data using *Stanford CoreNLP* tokenizer [6]. The script for generating the JSON files is `generate-json.py` and all of the generated JSON files (i.e. `train.json`, `valid.json` and `test.json`) were stored in `json_data` directory. The last step in data preparation for **BertSum-transformer** and **BertSum-bert** is to generate the binary files for model training, given the previosly generated JSON files. In particular, we used the script `preprocess.py` in the source code directory (i.e.src) [7] together with several arguments. The complete command is shown below. The resulting binary files were stored in `bert_data` directory.

```
$ Python /src/preprocess.py -mode format_to_bert
-raw_path ../json_data -save_path ../bert_data -lower -n_cpus 3
-log_file ../logs/preprocess-bert.log -min_src_nsents 1
```

[6] https://nlp.stanford.edu/software/tokenizer.shtml
[7] Code available from here: https://github.com/nlpyang/PreSumm

```
"src": [
    [
        "What",
        "is",
        "-LRB-",
        "are",
        "-RRB-",
        "HIV/AIDS",
        "?"
    ]
],
"tgt": [
    [
        "HIV",
        "stands",
        "for",
        "human",
        "immunodeficiency",
        "virus",
        ".",
        "It",
        "kills",
        "or",
        "damages",
        "the",
        "body",
        "'s",
        "immune",
        "system",
        "cells",
        ".",
```


Figure 5.2: Sample tokenized src-tgt pair in JSON format

[Model Training]

Since **BertSum-trained** is a trained model, there is no need to re-train it again. The detailed steps for training **BertSum-transformer** and **BertSum-bert** are the main focus in this section.

BertSum-transformer The training script we used is the one included in `src` directory, namely `train.py`. Specifically, the hyper-parameter setting is shown in Table 5.3. Note that `-encoder baseline` is passed as an argument to the training script, resulting in training in transformer settings for both encoder and decoder. Also, during training, the model checkpoints were saved every 2000 steps. In other words, 5 checkpoints (since we trained for 10000 steps) for **BertSum-transformer** were saved in `model` directory.

Hyperparameters	Value Setting
batch size	300
learning rate	0.05
save checkpoints steps	2000
train steps	10000
warmup steps	8000
maximum output length	512
# of hidden nodes in encoder/decoder	512
# of layers in encoder/decoder	6
dropout rate in encoder	0.1

Table 5.3: Hyperparameters for Training BertSum-transformer

BertSum-bert Similar to training BertSum-transformer, the same script `train.py` was used. In particular, the hyper-parameter setting is illustrated in Table 5.4. Analogously, 5 checkpoints were saved into `model` directory.

Hyperparameters	Value Setting
batch size	140
bert encoder learning rate	0.002
decoder leanring rate	0.2
save checkpoints steps	2000
train steps	10000
bert warmup steps	20000
decoder warmup steps	10000
maximum output length	512
dropout rate in decoder	0.2

Table 5.4: Hyperparameters for Training BertSum-bert

[**Model Validation**]

After training, setting `-mode` to `validate` along with `-test_all` flag enables the system to load all saved checkpoints and evaluate them on the validation set (i.e. valid.src and valid.tgt). For both **BertSum-transformer** and **BertSum-bert**, the best checkpoint was the one saved after 10000 steps (i.e. model_step-10000.pt). This will be used to ultimately evaluate the two models on the test data.

[Model Evaluation]

In this section, all of the model configurations for BertSum were evaluated, namely **BertSum-trained**, **BertSum-transformer** and **BertSum-bert**. Specifically, we set `-mode` to be `test` and the generated results were stored in a `results` directory. For **BertSum-trained**, the tested checkpoint was `model_step-30000.pt`, which was directly downloaded from the GitHub webpage [8]. The testing script generated a `.candidate` file in the `results` directory, which contained the predictions given source questions from test set. We used `test.tgt` along with the generated `.candidate` file to calculate the BLEU and ROUGE scores. The evaluation results were shown in Table 5.5.

Evaluation Metrics	BertSum-transformer	BertSum-bert	BertSum-trained
BLEU-1	58.9	59.2	40.9
BLEU-2	34.3	35.7	5.2
BLEU-3	26.6	28.3	0.4
BLEU-4	23.1	24.8	0.1
BP	0.13	0.097	0.000
syslen	148433	135181	33274
reflen	452516	452516	452516
ROUGE-1	0.34	0.32	0.08
ROUGE-2	0.19	0.19	0.01
ROUGE-L	0.36	0.35	0.08

Table 5.5: Evaluation results of BertSum

5.2.3 MASS

We experimented on two configurations for MASS, namely **MASS-with-pretrained** and **MASS-without-pretrained**. Specifically, the pre-trained model by [65] was used in **MASS-with-pretrained**, while **MASS-without-pretrained** was trained from scratch.

[Data Preparation and Preprocessing]

MASS-with-pretrained Similar to the data preparation step in LSTM-Seq2Seq, alignment files for train, validation and test set were created using `alignment.py` script. Then, we followed the tokenization step used in MASS-SUM [9], where `encode.py` was called. The

[8] https://github.com/nlpyang/PreSumm
[9] https://github.com/microsoft/MASS/tree/master/MASS-summarization

resulting tokenized corpus is stored in `tokenzied` directory. Subsequently, the pre-trained MASS model was downloaded [10] and unzipped to the project root directory. Next, we used `fairseq-preprocess` together with several arguments to generate the binarized files for the tokenized MedQuAD data, which will be used in training/fine-tuning. The command to do so is demonstrated as follows. Note that `dict.txt` was initially included in the directory containing the pre-trained model (i.e. mass-base-uncased). The preprocessed data was saved in `processed` directory. Also note that `--thresholdtgt 0` and `--thresholdsrc 0` ensured that no token was marked as *UNK*.

```
$ fairseq-preprocess \
 --user-dir ./MASS-summarization/mass --task masked\_s2s \
 --source-lang src --target-lang tgt \
 --trainpref ./tokenized/train --validpref ./tokenized/valid
 --testpref ./tokenized/test \
 --destdir ./processed --srcdict dict.txt --tgtdict dict.txt \
 --workers 20 \
 --thresholdtgt 0 --thresholdsrc 0 \
```

MASS-without-pretrained Similar to BertSum, we generated three raw data files, namely `train.raw`, `valid.raw` and `test.raw`, by using `generate-raw.py`. These raw data files were stored in `raw_data` directory. We then followed the same tokenization step used in **MASS-with-pretrained** to tokenize the three data files and store them in the `tokenized-without-pretrained` directory. Subsequently, the binary files were created based on the tokenzied data using `fairseq-preprocess`. The complete commond we used is also shown below. The resulting binary files were stored in `processed-without-pretrained` directory.

```
$ fairseq-preprocess \
--user-dir ./MASS-summarization/mass --only-source --task masked_s2s \
--trainpref tokenized-without-pretrained/train.txt \
```

[10] https://modelrelease.blob.core.windows.net/mass/mass-base-uncased.tar.gz

```
--validpref tokenized-without-pretrained/valid.txt \
--testpref tokenized-without-pretrained/test.txt \
--destdir processed-without-pretrained
--srcdict dict.txt --workers 60
```

[Model Training/Fine-tuning]

MASS-with-pretrained Given the pre-trained model, stored in `mass-base-uncased`, we fine-tuned the model on our binarized *MedQuAD* dataset, stored in `processed` directory. The `fairseq-train` was called along with other hyper-parameters for fine-tuning. The values for all hyper-parameters are shown in Table 5.6. The optimizer we chose was *Adam*. Besides, we used *inverse_sqrt* as the learning rate scheduler. The complete training script was included in `mass-ft.pbs` in the project directory.

Hyperparameters	Value Setting
adam betas	(0.9, 0.98)
maximum learning rate	0.0005
minimum learning rate	1e-09
warmup updates	4000
maximum training epoch	100
maximum source length	512
maximum target length	512

Table 5.6: Hyperparameters for Fine-tuning MASS-with-pretrained

MASS-without-pretrained We also used `fairseq-train` for training **MASS-without-pretrained**. The required binary files were from **processed-without-pretrained**, which was created during data processing. The hyperparameters are shown in Table 5.7. In this case, we adopted *polynomial_decay* as the learning rate scheduler. In particular, `mass-pretrain.pbs` shows the complete command used for training.

[Model Evaluation]

To evaluate both model configurations, `fairseq-generate` was used to generate the predicted answers given the questions from test set. Specifically, we selected the best checkpoint (i.e. checkpoint_best.pt) from the trained model directory. The script for generating

Hyperparameters	Value Setting
tokens per sample	512
adam betas	(0.9, 0.98)
maximum learning rate	0.0005
warmup updates	10000
total # of updates	650
dropout rate	0.1
attention dropout	0.1

Table 5.7: Hyperparameters for Training MASS-without-pretrained

the hypothesis answers was encoded in `mass-eval.pbs`. We set the beam width to be 5. In particular, `fairseq-generate` creates a file containing all the predicted results. We calculated the BLEU and ROUGE scores based on the hypothesis results and the gold answers in `test.tgt`. The detailed results were shown in Table 5.8.

Evaluation Metrics	MASS-with-pretrained	MASS-without-pretrained
BLEU-1	53.8	31.3
BLEU-2	30.2	11.7
BLEU-3	23.1	6.3
BLEU-4	19.7	4.1
BP	0.434	0.952
syslen	153795	269008
reflen	282322	282322
ROUGE-1	0.38	0.29
ROUGE-2	0.21	0.11
ROUGE-L	0.39	0.29

Table 5.8: Evaluation results of MASS

5.2.4 PoDA

Unlike previously trained models, we experimented PoDA on our *MedQuAD* dataset with only one configuration. In particular, we fine-tuned PoDA's pre-trained model on *MedQuAD* and tested its result.

[Data Preparation]

Like LSTM-Seq2Seq, alignment files for train, validation and test set were generated using `alignment.py`. Then, we tokenized the alignment files using *Stanford CoreNLP* tokenizer, in which case all the tokenzied files were stored in `MedQuAD-tokenized` directory. Then,

we pre-processed and binarized the tokenized corpus via `preprocess.py` inside `fairseq` directory. Note that we did not use `fairseq-preprocess` in PoDA model training. The script we used is shown as follows. Note that `da-pretrained` is the directory containing the pre-trained PoDA model and the resulting dictionary files from the model (i.e. dict.src.txt and dict.trg.txt). The binary files were stored in `/MedQuAD-tokenized/bin` directory.

```
$ processed_dir='./MedQuAD-tokenized/'
$ dict_size=50000
$ python3 -u ./fairseq/preprocess.py \
 --source-lang src --target-lang trg \
 --trainpref $processed_dir/train \
 --validpref $processed_dir/valid \
 --testpref $processed_dir/test \
 --padding-factor 0 \
 --srcdict ./da-pretrained/dict.src.txt \
 --tgtdict ./da-pretrained/dict.trg.txt \
 --destdir $processed_dir/bin \
 --thresholdtgt 0 \
 --thresholdsrc 0 \
 --nwordssrc $dict_size \
 --nwordstgt $dict_size \
```

[Model Training/Fine-tuning]

Having downloaded the pre-trained PoDA model, stored in `da-pretrained` directory, we fine-tuned it on our binarized *MedQuAD* data. Specifically, the script `train.py` under `fairseq` directory was used along with other tunable hyper-parameters, which are shown in Table 5.9. The complete training script was encoded in `poda-ft.pbs`.

Hyperparameters	Value Setting
maximum epochs	100
batch size	64
warmup updates	4000
maximum learning rate	0.0005
minimum learning rate	1e-5
dropout rate	0.2
relu dropout rate	0.2
attention dropout rate	0.2
encoder/decoder word embedding dimension	512
maximum source length	512
maximum target length	512

Table 5.9: Hyperparameters for Fine-tuning PoDA

[**Model Evaluation**]

Similar to MASS, we used a script (i.e. /fairseq/generate.py) to generate the predicted answers given the source questions from test set. Specifically, `-mas_tokens` was set to 3000 to enable the model to generate longer answers. At the same time, `-unkpen`, known as *UNK* penalty, was set to 100 to discourage the model to generate unknown tokens. The evaluation results are shown in Table 5.10. Similar to other model evaluations, the BLEU and ROUGE scores were calculated referring to the system generated file (i.e. gen.sys) and the gold target file (trg.ref).

Evaluation Metrics	PoDA
BLEU-1	57.8
BLEU-2	29.7
BLEU-3	20.4
BLEU-4	15.9
BP	0.009
syslen	64797
reflen	370386
ROUGE-1	0.22
ROUGE-2	0.11
ROUGE-L	0.25

Table 5.10: Evaluation results of PoDA

5.2.5 DIALOGPT

We directly used the fine-tuned model of DIALOGPT from its GitHub webpage [11]. Specifically, we adopted the large fine-tuned DIALOGPT model (approximately 762M) and directly run the model on our *MedQuAD* test data (containing 1946 Q-A pairs). Following the usage information about the fine-tuned model [12], we wrote a script (i.e. run.py) to feed our source question sentence from `test.src` to the large fine-tuned model. The generated response was recorded in a `runlog.txt` file. Subsequently, we calculated the BLEU and ROUGE score using the generated `runlog.txt` and the file containing the gold answers `test.tgt`. The detailed results are shown in Table 5.11.

Evaluation Metrics	DIALOGPT-trained
BLEU-1	21.2
BLEU-2	2.1
BLEU-3	0.5
BLEU-4	0.1
BP	0.0
syslen	22223
reflen	452516
ROUGE-1	0.04
ROUGE-2	0.00
ROUGE-L	0.04

Table 5.11: Evaluation results of DIALOGPT

5.3 Analyzing Evaluation Results

Through the experiments we conducted based on *MedQuAD* dataset, the performances of different model configurations of the state-of-the-art deep learning based systems vary a lot. In general, we categorized these tested model configurations into four classes: 1) adopting a pre-trained model and fine-tuning on *MedQuAD*; 2) pre-training from scratch and fine-tuning on *MedQuAD*; 3) training from scratch without pre-training or using any pre-trained models; 4) testing a trained/fine-tuned model directly on our *MedQuAD* test set. The specific categorization for all the tested models is demonstrated in Table 5.12.

[11] https://github.com/microsoft/DialoGPT
[12] https://huggingface.co/microsoft/DialoGPT-large

In this section, we will analyze the evaluation results of these tested model configurations from several distinct perspectives, with the assistance of their specified categories.

Category	Model Configurations
pre-trained+fine-tuning	MASS-with-pretrained, PoDA, LSTM-Seq2Seq-glove
pre-training+fine-tuning	BertSum-bert, BertSum-transformer
no pre-training	LSTM-Seq2Seq-no-glove, MASS-without-pretrained
trained model	LSTM-Seq2Seq-trained, BertSum-trained, DIALOGPT-trained

Table 5.12: Model Configurations Classification

5.3.1 Comparison among "pre-trained+fine-tuning" Models

The three model configurations, namely **MASS-with-pretrained**, **PoDA** and **LSTM-Seq2Seq-glove**, adopted pre-trained models and were further fine-tuned over the *MedQuAD* dataset. Specifically, **MASS-with-pretrained** employed a model that is pre-trained on a large-scale English monolingual data; **PoDA** is pre-trained on two rich-resource data corpus and **LSTM-Seq2Seq-glove** employed GloVe [49] pre-trained word embeddings for training. The evaluation results for these three models on *MedQuAD* test set is synthesized into Table 5.13 shown below. What is clearly observed in the table is that **MASS-with-pretrained** outperformed the other two models on almost every evaluation metric, except for *BLEU-1*. Note that there is a significant difference in terms of the Brevity Penalty (BP), indicating that **MASS-with-pretrained** tends to generate longer answers than the others. A few sample generated answers from the three models are demonstrated in Table 5.14. As we closely observed these sample generated answers, several crucial patterns can be identified. First of all, **MASS-with-pretrained** model tends to generate longer answers that are grammatically correct and structurally coherent. The answers seem to address directly to the question, even though occasionally, spurious repeated words may be generated, such as "intellectual disability" in the answer to the second question. Besides, noisy answers may be generated that are irrelevant to the question. As for **PoDA**, it tends to generate answers following a general template (i.e. Key Points etc.). Nevertheless, **PoDA** fails to differentiate between different subjects in the questions (e.g. HIV/AIDS or Hydranencephaly). The generated answers contain almost

similar contents except for the queried disease. Also note that **PoDA** tends to generate shorter answers, within which repeated answers may be also generated. With respect to **LSTM-Seq2Seq-glove**, even though it tends to generate relatively long responses, they are not structurally as coherent as **MASS-with-pretrained** in that different sentences are not explicitly separated by periods. Thus, the generated answers may not be readable. Considering all these performance disparities among the three pre-trained based models, **MASS-with-pretrained** is the most suitable technique for medical natural answer generation on *MedQuAD* dataset.

Evaluation Metrics	MASS-with-pretrained	PoDA	LSTM-Seq2Seq-glove
BLEU-1	53.8	**57.8**	52.0
BLEU-2	**30.2**	29.7	25.5
BLEU-3	**23.1**	20.4	16.4
BLEU-4	**19.7**	15.9	10.6
BP	**0.434**	0.009	0.02
syslen	**153795**	64797	91825
reflen	282322	370386	452516
ROUGE-1	**0.38**	0.22	0.23
ROUGE-2	**0.21**	0.11	0.11
ROUGE-L	**0.39**	0.25	0.29

Table 5.13: Evaluation results for "pre-trained+fine-tuning" models

5.3.2 Comparison among "pre-training+fine-tuning" Models

This involves **BertSum-bert** and **BertSum-transformer**. As introduced in section 3.2, both models pre-train a document encoder to capture the representations of the sentences in the *MedQuAD* dataset. Then, the pre-trained sentence representations will be treated as the input vectors to the encoder-decoder network, where the encoder is pre-trained based on BERT [13] and the Transformer-based decoder is trained from scratch. The evaluation results for these two models are demonstrated in Table 5.15. In general, **BertSum-bert** outperforms **BertSum-transformer** in terms of all BLEU measurements. This is likely due to the fact that fine-tuning the parameters in pre-trained BERT encoder over *MedQuAD* produces the answers that have better precision than training the Transformer-based encoder and decoder all together from scratch. Besides, both models generate short answers with respect to the reference length (i.e. reflen). The reason

Question	**What is (are) HIV/AIDS ?**
MASS	there is no cure for hiv / aids, nor is there a standard course of treatment. treatment is based on the signs and symptoms present in each person. for example, if you have hiv, your doctor may prescribe medicines to treat the infection. you may need to take medicine to treat it. nih : national institute of diabetes and digestive and kidney diseases.
PoDA	Key Points - HIV/AIDS is a disease in which malignant blood cancer cells form in the tissues of the brain. - HIV/AIDS is a disease in which malignant blood cancer cells form in the tissues of the brain.
LSTM Seq2Seq	Key Points - Avoiding risk factors and increasing protective may help prevent lung cancer The following are for : Cigarette , cigar pipe smoking Secondhand smoke Family history HIV infection Environmental Beta carotene supplements increase the of gum disease oropharyngeal Not Quitting Lower exposure to workplace
Question	**What is (are) Hydranencephaly ?**
MASS	hydranencephaly is a rare neurological disorder characterized by abnormal development of the brain and spinal cord (central nervous system). it may occur in children between the ages of 5 and 15 years it is most common in children younger than 5 years of age. it usually occurs in children under the age of 5 years. the disorder may be associated with seizures, seizures , intellectual disability, and intellectual disability. the cause of the disorder is unknown. treatment is symptomatic and supportive.
PoDA	Key Points - Hydranencephaly is a disease in which malignant blood cancer cells form in the tissues of the brain. - Hydranencephaly is a disease in which malignant blood cancer cells form in the tissues of the brain.
LSTM Seq2Seq	Hydranencephaly is a rare condition in which the brain's cerebral hemispheres are absent and replaced by sacs filled with cerebrospinal fluid (CSF) # Affected infants generally die within first 6 months of age remains until period residual microcephaly children The most common symptoms , childhood development may be delayed mid-adulthood Progression varies widely based on signs present each person

Table 5.14: Sample generations from "pre-trained+fine-tuning" models

may be that during training, the maximum output length for both models were set to 512, which was shorter than the actual maximum target length in *MedQuAD* dataset. The statistics for *MedQuAD* in terms of the total/average/maximum/minimum sentence length for both questions and answers are shown in Table 5.16. Moreover, **BertSum-transformer** has slightly better ROUGE evaluations, indicating better alignments with respect to reference answers.

5.3.3 Effects of Adopting Pre-trained Models for Fine-tuning

We will compare and contrast the performances between **MASS-with-pretrained** and **MASS-without-pretrained** in order to reveal the effectiveness of adopting pre-trained models for training/fine-tuning. In the experiment for **MASS-without-pretrained**, as

Evaluation Metrics	BertSum-transformer	BertSum-bert
BLEU-1	58.9	**59.2**
BLEU-2	34.3	**35.7**
BLEU-3	26.6	**28.3**
BLEU-4	23.1	**24.8**
BP	**0.13**	0.097
syslen	**148433**	135181
reflen	452516	452516
ROUGE-1	**0.34**	0.32
ROUGE-2	0.19	0.19
ROUGE-L	**0.36**	0.35

Table 5.15: Evaluation results of BertSum "pre-training+fine-tuning" models

Category	Questions	Answers
total # of length	107798	2794665
avg # of length	9	230
min # of length	4	1
max # of length	34	4801

Table 5.16: Statistics of sentence length in MedQuAD

illustrated in section 5.2.3, no pre-trained model was used during training. The evaluation results are re-demonstrated in Table 5.17, with additional highlights for the specific values. It is noticeable that **MASS-with-pretrained** significantly outperforms **MASS-without-pretrained** in terms of both BLEU and ROUGE measurements, indicating better quality for the generated answers. However, the latter has a higher BP, suggesting a better alignment in length with the references. For the purpose of demonstration, sample generated answers by the two models are illustrated in Table 5.18. Even though **MASS-without-pretrained** tends to generate longer answers in order to align with the reference in length, the resulting texts are sometimes not readable and contain a lot of spurious words that may not be in the references. As illustrated, **MASS-with-pretrained** still persists to produce natural answer, which is grammatically correct and structurally coherent, exemplifying the power of adopting pre-trained models in natural answer generation.

Evaluation Metrics	MASS-with-pretrained	MASS-without-pretrained
BLEU-1	**53.8**	31.3
BLEU-2	**30.2**	11.7
BLEU-3	**23.1**	6.3
BLEU-4	**19.7**	4.1
BP	0.434	**0.952**
syslen	153795	**269008**
reflen	282322	282322
ROUGE-1	**0.38**	0.29
ROUGE-2	**0.21**	0.11
ROUGE-L	**0.39**	0.29

Table 5.17: Evaluation results of MASS with highlights

Question	**What is (are) Colorectal Cancer ?**
MASS with pretrained	key points - colorectal cancer is a disease in which malignant (cancer) cells form in the tissues of the colon. - tests that examine the colon are used to detect (find) , diagnose , and stage colon cancer. - certain factors affect prognosis (chance of recovery) and treatment options.
MASS without pretrained	what are the treatments for colorectal cancer ? ? there are the signs and symptoms ? the national cancer institute ? the diagnosis and treatment options are therapies ? the standard treatment are used to study , theraalalal cancer is ? ? the treatment are the standard standard treatment ? the currently used to prevent colorectectalal colorectomy ? the cancer ? the following are the national institute of colorectableal cancer and the national hormone ? what are medications used to reduce the risk factors ? the risk , and whether or dietary dietary changes , such as carcinoma? the coloral cancer has just been diagnosed ? the the following pdqs ? ? colorectss ? the possible side effects ? the studies have shown to be treated ? the use the color color coloralal or the the national color color , or the national national cancer ? there is (colorectedsss , color color ? the size , and the size size of the size and location of
Question	**What are the treatments for Pigmented villonodular synovitis ?**
MASS with pretrained	these resources address the diagnosis or management of pigmented villonodular synovitis: - american cancer society: how are skin lesions diagnosed ? - genetic testing registry : skin lesions these resources from medlineplus offer information about the diagnosis and management of various health conditions: - diagnostic tests - drug therapy - surgery and rehabilitation - genetic counseling - palliative care
MASS without pretrained	what are the treatments for villonodular synovitis pigmentary retinitis pigmentitis pigmentosa ? there is no standard treatment for vitiligo ? there are the treatment options for vitelliform macular degeneration ? these resources address the diagnosis or management of vitreoretinal pigmented vitrectitis pigmentation ? how might vitreodystrophizen pigmentary synoviramate synthes ? treatment options are the currently there is currently no effective treatment for pigmentary disorders ? treatment treatment options that can be treated ? the treatment planitis pigmentoscopolysynostosis pigmentary pigmentary ? treatment depends on the specific treatment options options for pigmentation disorders and pigmentary degenerative disorders ? therapies may be used to diagnose pigmentation group of pigmented pigmentation pigmentation. treatment options may be required to remove the pigmentation of pigmentation with vi vi ? the best treatment options with vitreous pigmentation and pigmentation

Table 5.18: Sample generations from MASS models

5.3.4 Effects of Adopting GloVe Pre-trained Word Embeddings

During experimenting with **LSTM-Seq2Seq** models, we explored the effectiveness of employing pre-trained word embeddings (i.e. GloVe) for generating natural answers. Specif-

ically, we trained and evaluated two models: **LSTM-Seq2Seq-no-glove** and **LSTM-Seq2Seq-glove**. We did not adopt GloVe in training **LSTM-Seq2Seq-no-glove** model, instead the word embeddings were learned from scratch. The evaluation results are presented in Table 5.19 for your reference. In terms of BLEU scores, the performance of **LSTM-Seq2Seq-no-glove** was slightly better. The reason may be that *MedQuAD* contains some rare words and their corresponding word embeddings were learned from scratch. Therefore, during training, the model can better capture the underline representations of those "difficult" words. Moreover, **LSTM-Seq2Seq-glove** generated slightly longer answers. In general, the difference in performance for these two models is trivial, suggesting that GloVe pre-trained word embeddings bring little performance boost for generating natural answers given medical related queries.

Evaluation Metrics	LSTM-Seq2Seq-glove	LSTM-Seq2Seq-no-glove
BLEU-1	52.0	**52.5**
BLEU-2	25.5	**25.8**
BLEU-3	16.4	**16.6**
BLEU-4	10.6	**10.7**
BP	**0.02**	0.019
syslen	**91825**	91262
reflen	452516	452516
ROUGE-1	0.23	0.23
ROUGE-2	0.11	0.11
ROUGE-L	0.29	0.29

Table 5.19: Evaluation results of LSTM-Seq2Seq models

5.3.5 Comparison among Trained Models

Instead of training or fine-tuning on the train/validation set of **MedQuAD**, the trained models (i.e. **LSTM-Seq2Seq-trained**, **BertSum-trained** and **DIALOGPT-trained**) were directly tested on the test set of *MedQuAD*. The evaluation results are gleaned from previous sections and are shown in Table 5.20 below. Doubtlessly, the BLEU and ROUGE scores for all trained models should be really low, since they were not trained/fine-tuned on our *MedQuAD* corpus. In other words, these bad BLEU and ROUGE scores indicates the lack of contextual information and representations from *MedQuAD*. A few generated

sample answers are demonstrated in Table 5.21 below. It can be inferred from the sample answers that **LSTM-Seq2Seq-trained** is not able to generate any useful tokens if the words in the question are never seen before. Instead, the *UNK* symbol is generated and the whole sentence is not readable. **BertSum-trained** and **DIALOGPT-trained** tends to generate natural answers, even though the answer itself did not directly address the question. Besides, **BertSum-trained** tends to refer to some political excerpts and announcements as the response. Interestingly, for DIALOGPT, conversational responses were generated, inferring that DIALOGPT might be useful in a multi-turn dialogue settings. In a nut shell, these three trained models can not be directly used to address medical related queries, even if they can generate natural answers.

Evaluation Metrics	LSTM-Seq2Seq-trained	BertSum-trained	DIALOGPT-trained
BLEU-1	14.2	**40.9**	21.2
BLEU-2	1.5	**5.2**	2.1
BLEU-3	0.1	0.4	**0.5**
BLEU-4	0.0	**0.1**	**0.1**
BP	0.0	0.0	0.0
syslen	10423	**33274**	22223
reflen	452516	452516	452516
ROUGE-1	0.01	**0.08**	0.04
ROUGE-2	0.00	**0.01**	0.00
ROUGE-L	0.02	**0.08**	0.04

Table 5.20: Evaluation results for "trained models"

Question	**How to prevent Urinary Tract Infection In Adults ?**
LSTM Seq2Seq	*UNK* to richer
BertSum	health officials in the uk have announced a series of measures to prevent urinary tract infections.
DIALO-GPT	What is this?
Question	**Who is at risk for Omsk Hemorrhagic Fever (OHF) ?**
LSTM Seq2Seq	*UNK* depends on law-making scheme
BertSum	all 23,000 people in north-eastern nigeria have been warned to stay at risk of a deadly illness.
DIALO-GPT	I'm not sure if you're being sarcastic or not, but I'm not sure if you're being sarcastic.
Question	**How to prevent Asthma ?**
LSTM Seq2Seq	*UNK* to form
BertSum	health secretary jeremy corbyn has announced a series of measures to reduce the risk of asthma.
DIALO-GPT	I'm allergic to asthma medication.

Table 5.21: Sample generations from "trained models"

Chapter 6

Summary and Reflections

6.1 Project Management

The project was developed in a test-driven and agile manner. The advantage of adopting test-driven development methodology is its ability to painlessly update the system programs to address unforeseen variables so that the risks of system crash are largely minimized. Also, the agile methodology is beneficial because of its characteristics of iterative planning, making the implementation easy to compile if unexpected changes occur. This is crucial in a research project given that the direction of the project is mostly governed by the findings of the research.

In particular, the initial objective of our work (i.e. build up a deep learning based system for medical answer generation) was altered in the presence of several unpredictable factors. For instance, it was hard to tweak the structure of MASS [65] for the purpose of boosting its performance on *MedQuAD* data so that we could build our system based on the tweaked system. Instead, we experimented with all the state-of-the-art deep learning based systems on the *MedQuAD*, conducted a detailed analysis based on their performance and evaluated their power in generating natural answers given medical related questions.

6.2 Future Work and Reflections

Based on the analysis of the performances of the state-of-the-art deep learning based techniques over *MedQuAD*, future work of this topic (i.e. Medical Natural Answer Generation) will center around developing more robust deep learning based approaches. Specifically, we will 1) crawl the filtered-out *MedQuAD* data that was removed due to copyright issue so that we can conduct the experiments using more Q-A pairs; 2) utilize the question type in *MedQuAD* for some other topics, such as medical question classification and 3) fine-tune DIALOGPT according to the instructions so that a more comprehensive analysis can be performed; 4) based on the analysis, we will develop our own answer generation system that can overcome the restrictions (e.g. output length, repeated words and etc.) of the current state-of-the-art systems.

Bibliography

[1] Abacha, A. B., and Demner-Fushman, D. A question-entailment approach to question answering. *arXiv preprint arXiv:1901.08079* (2019).

[2] Abacha, A. B., Shivade, C., and Demner-Fushman, D. Overview of the mediqa 2019 shared task on textual inference, question entailment and question answering. In *Proceedings of the 18th BioNLP Workshop and Shared Task* (2019), pp. 370–379.

[3] Allam, A. M. N., and Haggag, M. H. The question answering systems: A survey. *International Journal of Research and Reviews in Information Sciences (IJRRIS) 2*, 3 (2012).

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

[5] Balikas, G., Kosmopoulos, A., Krithara, A., Paliouras, G., and Kakadiaris, I. Results of the bioasq tasks of the question answering lab at clef 2015.

[6] Bian, W., Li, S., Yang, Z., Chen, G., and Lin, Z. A compare-aggregate model with dynamic-clip attention for answer selection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (2017), ACM, pp. 1987–1990.

[7] Bordes, A., Usunier, N., Chopra, S., and Weston, J. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075* (2015).

[8] Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J., Mercer, R. L., and Roossin, P. S. A statistical approach to machine translation. *Computational linguistics 16*, 2 (1990), 79–85.

[9] Cai, Q., and Yates, A. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2013), pp. 423–433.

[10] Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Parmar, N., Schuster, M., Chen, Z., et al. The best of both worlds: Combining recent advances in neural machine translation. *arXiv preprint arXiv:1804.09849* (2018).

[11] Cui, W., Xiao, Y., Wang, H., Song, Y., Hwang, S.-w., and Wang, W. Kbqa: learning question answering over qa corpora and knowledge bases. *Proceedings of the VLDB Endowment 10*, 5 (2017), 565–576.

[12] Demner-Fushman, D., Chapman, W. W., and McDonald, C. J. What can natural language processing do for clinical decision support? *Journal of biomedical informatics 42*, 5 (2009), 760–772.

[13] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[14] Duraiswamy, K. An approach for text summarization using deep learning algorithm.

[15] Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., Lally, A., Murdock, J. W., Nyberg, E., Prager, J., et al. Building watson: An overview of the deepqa project. *AI magazine 31*, 3 (2010), 59–79.

[16] Graff, D., Kong, J., Chen, K., and Maeda, K. English gigaword. *Linguistic Data Consortium, Philadelphia 4*, 1 (2003), 34.

[17] Graves, A. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

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

[19] Gu, J., Lu, Z., Li, H., and Li, V. O. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393* (2016).

[20] Guo, Y., Gaizauskas, R., Roberts, I., Demetriou, G., Hepple, M., et al. Identifying personal health information using support vector machines. In *i2b2 workshop on challenges in natural language processing for clinical data* (2006), Citeseer, pp. 10–11.

[21] He, J., Fu, M., and Tu, M. Applying deep matching networks to chinese medical question answering: a study and a dataset. *BMC medical informatics and decision making 19*, 2 (2019), 52.

[22] He, S., Liu, C., Liu, K., and Zhao, J. Generating natural answers by incorporating copying and retrieving mechanisms in sequence-to-sequence learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2017), pp. 199–208.

[23] Heilman, M., and Smith, N. A. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics* (2010), Association for Computational Linguistics, pp. 1011–1019.

[24] Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. Teaching machines to read and comprehend. In *Advances in neural information processing systems* (2015), pp. 1693–1701.

[25] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[26] Hovy, E., and Lin, C.-Y. Automated text summarization and the summarist system. In *Proceedings of a workshop on held at Baltimore, Maryland: October 13-15, 1998* (1998), Association for Computational Linguistics, pp. 197–214.

[27] Hu, Z., Shi, H., Tan, B., Wang, W., Yang, Z., Zhao, T., He, J., Qin, L., Wang, D., Ma, X., et al. Texar: A modularized, versatile, and extensible toolkit for text generation. *arXiv preprint arXiv:1809.00794* (2018).

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

[29] Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810* (2017).

[30] Kumar, A. N., Kesavamoorthy, H., Das, M., Kalwad, P., Chandu, K., Mitamura, T., and Nyberg, E. Ontology-based retrieval & neural approaches for bioasq ideal answer generation. In *Proceedings of the 6th BioASQ Workshop A challenge on large-scale biomedical semantic indexing and question answering* (2018), pp. 79–89.

[31] Lai, T., Bui, T., and Li, S. A review on deep learning techniques applied to answer selection. In *Proceedings of the 27th International Conference on Computational Linguistics* (2018), pp. 2132–2144.

[32] Li, J., Galley, M., Brockett, C., Gao, J., and Dolan, B. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055* (2015).

[33] Li, J., Luong, M.-T., and Jurafsky, D. A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057* (2015).

[34] Li, Y., Gekakis, N., Wu, Q., Li, B., Chandu, K., and Nyberg, E. Extraction meets abstraction: Ideal answer generation for biomedical questions. In *Proceedings of the 6th BioASQ Workshop A challenge on large-scale biomedical semantic indexing and question answering* (2018), pp. 57–65.

[35] Lin, C.-Y. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out* (Barcelona, Spain, July 2004), Association for Computational Linguistics, pp. 74–81.

[36] Liu, Y., and Lapata, M. Text summarization with pretrained encoders. *arXiv preprint arXiv:1908.08345* (2019).

[37] McTear, M., Callejas, Z., and Griol, D. The conversational interface: Talking to smart devices: Springer international publishing.

[38] Mollá, D., and Vicedo, J. L. Question answering in restricted domains: An overview. *Computational Linguistics 33*, 1 (2007), 41–61.

[39] Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023* (2016).

[40] Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745* (2018).

[41] Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038* (2019).

[42] Pal, V., Shrivastava, M., and Bhat, I. Answering naturally: Factoid to full length answer generation. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization* (2019), pp. 1–9.

[43] Pampari, A., Raghavan, P., Liang, J., and Peng, J. emrqa: A large corpus for question answering on electronic medical records. *arXiv preprint arXiv:1809.00732* (2018).

[44] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics* (2002), Association for Computational Linguistics, pp. 311–318.

[45] Parikh, A. P., Täckström, O., Das, D., and Uszkoreit, J. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933* (2016).

[46] Paris, C. L. Towards more graceful interaction: a survey of question-answering programs.

[47] Paulus, R., Xiong, C., and Socher, R. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304* (2017).

[48] Peng, F., Weischedel, R., Licuanan, A., and Xu, J. Combining deep linguistics analysis and surface pattern learning: A hybrid approach to chinese definitional question answering. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing* (2005), Association for Computational Linguistics, pp. 307–314.

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

[50] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding with unsupervised learning. *Technical report, OpenAI* (2018).

[51] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog 1*, 8 (2019), 9.

[52] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683* (2019).

[53] Raghavan, P., and Patwardhan, S. Question answering on electronic medical records. In *CRI* (2016).

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

[55] Ravichandran, D., and Hovy, E. Learning surface text patterns for a question answering system. In *Proceedings of the 40th annual meeting on association for computational linguistics* (2002), Association for Computational Linguistics, pp. 41–47.

[56] Roberts, K., Demner-Fushman, D., Voorhees, E. M., Hersh, W. R., Bedrick, S., Lazar, A. J., and Pant, S. Overview of the trec 2017 precision medicine track. In *TREC* (2017).

[57] Sandhaus, E. The new york times annotated corpus. *Linguistic Data Consortium, Philadelphia 6*, 12 (2008), e26752.

[58] See, A., Liu, P. J., and Manning, C. D. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368* (2017).

[59] Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909* (2015).

[60] Sequiera, R., Baruah, G., Tu, Z., Mohammed, S., Rao, J., Zhang, H., and Lin, J. Exploring the effectiveness of convolutional neural networks for answer selection in end-to-end question answering. *arXiv preprint arXiv:1707.07804* (2017).

[61] Shen, S., Li, Y., Du, N., Wu, X., Xie, Y., Ge, S., Yang, T., Wang, K., Liang, X., and Fan, W. On the generation of medical question-answer pairs. *arXiv preprint arXiv:1811.00681* (2018).

[62] Shickel, B., Tighe, P. J., Bihorac, A., and Rashidi, P. Deep ehr: a survey of recent advances in deep learning techniques for electronic health record (ehr) analysis. *IEEE journal of biomedical and health informatics 22*, 5 (2017), 1589–1604.

[63] Shih, C.-W., Day, M.-Y., Tsai, T.-H., Jiang, T.-J., Wu, C.-W., Sung, C.-L., Chen, Y.-R., Wu, S.-H., and Hsu, W.-L. Asqa: Academia sinica question answering system for ntcir-5 clqa. In *NTCIR-5 workshop, Tokyo* (2005), pp. 202–208.

[64] Singh, S. P., Kumar, A., Darbari, H., Singh, L., Rastogi, A., and Jain, S. Machine translation using deep learning: An overview. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)* (2017), IEEE, pp. 162–167.

[65] Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450* (2019).

[66] Su, W., Boyd, S., and Candes, E. A differential equation for modeling nesterov's accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems* (2014), pp. 2510–2518.

[67] Su, Y., Sun, H., Sadler, B., Srivatsa, M., Gur, I., Yan, Z., and Yan, X. On generating characteristic-rich question sets for qa evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (2016), pp. 562–572.

[68] Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (2014), pp. 3104–3112.

[69] Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3104–3112.

[70] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 2818–2826.

[71] Tang, P. C., Fafchamps, D., and Shortliffe, E. H. Traditional medical records as a source of clinical data in the outpatient setting. In *Proceedings of the Annual Symposium on Computer Application in Medical Care* (1994), American Medical Informatics Association, p. 575.

[72] Tian, Y., Ma, W., Xia, F., and Song, Y. Chimed: A chinese medical corpus for question answering. In *Proceedings of the 18th BioNLP Workshop and Shared Task* (2019), pp. 250–260.

[73] Tran, Q. H., Lai, T., Haffari, G., Zukerman, I., Bui, T., and Bui, H. The context-dependent additive recurrent neural net. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (2018), pp. 1274–1283.

[74] Unger, C., Freitas, A., and Cimiano, P. An introduction to question answering over linked data. In *Reasoning Web International Summer School* (2014), Springer, pp. 100–140.

[75] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems* (2017), pp. 5998–6008.

[76] Voorhees, E. M., et al. The trec-8 question answering track report. In *Trec* (1999), vol. 99, Citeseer, pp. 77–82.

[77] Wang, L., Zhao, W., Jia, R., Li, S., and Liu, J. Denoising based sequence-to-sequence pre-training for text generation. *arXiv preprint arXiv:1908.08206* (2019).

[78] Wang, M., and Manning, C. D. Probabilistic tree-edit models with structured latent variables for textual entailment and question answering. In *Proceedings of the 23rd International Conference on Computational Linguistics* (2010), Association for Computational Linguistics, pp. 1164–1172.

[79] Wei, M., and Zhang, Y. Natural answer generation with attention over instances. *IEEE Access 7* (2019), 61008–61017.

[80] Woods, W. A. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the June 4-8, 1973, national computer conference and exposition* (1973), ACM, pp. 441–450.

[81] Xie, Z. Neural text generation: A practical guide. *arXiv preprint arXiv:1711.09534* (2017).

[82] Xu, J., Licuanan, A., and Weischedel, R. M. Trec 2003 qa at bbn: Answering definitional questions. In *TREC* (2003), pp. 98–106.

[83] Young, T., Hazarika, D., Poria, S., and Cambria, E. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine 13*, 3 (2018), 55–75.

[84] Zhang, J., and Zong, C. Deep neural network in machine translation. *Institute of Automation, Chinese Academy of Sciences*.

[85] Zhang, Y., Sun, S., Galley, M., Chen, Y.-C., Brockett, C., Gao, X., Gao, J., Liu, J., and Dolan, B. Dialogpt: Large-scale generative pre-training for conversational response generation. *arXiv preprint arXiv:1911.00536* (2019).