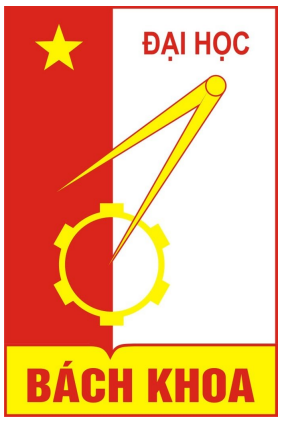
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information Communication and Technology



BACHELOR THESIS

INFORMATION SYSTEM PROGRAM

A PIPELINE OF DUAL ENCODER AND CROSS ENCODER ARCHITECTURE FOR BUILDING OPEN-DOMAIN QUESTION ANSWERING SYSTEM

Supervisor Assoc. Prof. Pham Van Hai

Author Le Vu Loi
Talented class of Computer Science
loi.lv173240@sis.hust.edu.vn

Ha Noi - June 18, 2021

Contents

List of Figures

List of Tables

ACKNOWLEDGEMENT

I want to express my sincere gratitude first to my supervisor Assoc. Prof. Pham Van Hai, who carefully guided me during the period of working on this Thesis. I'll have to thank Prof. Hai for all the patience, carefulness and enthusiastic he spent to me.

Secondly, I want to send millions thanks to my parents, my family for all the support they gave me. Without such a great spiritual motivation, I could not have surpassed the tough time during working on the Thesis.

Third, I am very grateful for being taught by lecturers at SoICT of HUST as well as all lecturers that have taught me. They are all great people that I can learn so much from.

Finally, I want thank all my friends, my classmates. I received a lot of help from them, not only during the period of this Thesis but also throughout 4 years I learned and worked at Hanoi University of Science and Technology.

SUMMARY OF THE THESIS

This Thesis studied the problem of building Open-domain question answering system. An open-domain question answering system takes as input a user question in human language and produce a short, exact answer that user expects. Such a system is highly expected because it can significantly reduce the task of searching and reading documents usually performed by human. Based on existing methods in the literature, this Thesis implemented a pipeline of dual encoder and cross encoder architecture for Open-domain question answering system, efficiently training and evaluating the model on Cloud TPUs. After that, the Thesis proposed a *stratified loss* to train the model that finally resulted in an improvement in performance of model. As a case study, the Thesis trained a model for Vietnamese language with COVID-19 dataset. Numerous experiments were conducted to analyze effectiveness the proposed stratified loss and several initial results on the COVID-19 dataset were presented.

The organization of the Thesis is as follows:

- Chapter I. Introduction: Overview about Open-domain question answering problem.
- Chapter II. Theoretical Preliminaries: Present theoretical prelininaries needed for the work of the Thesis.
- Chapter III. A pipeline of Dual Encoder and Cross Encoder for building Open-domain question answering system: Present in depth the Dual Encoder and Cross Encoder architecture of the system, the proposed *stratified loss* and a use case for Vietnamese open-domain question answering system in COVID-19 topic.
- Chapter IV. Experimental results: Describe datasets, metrics, analysis and discuss on experimental results.
- Chapter V. Conclusion and Future works: Conclude the Thesis and outline several possible future directions.

Chapter 1

Introduction

1.1 Overview

Question answering (QA) is at the medium level of difficulty among various tasks in Natural Language Processing (NLP). This task aims at giving the precise answer to a question given by human. An artificial agent with the capability of question answering is a direct evidence of computer understanding human language, which is the objective of research in NLP. Applications of question answering cover a wide range of domains, such as medical, bank, education, sales, etc. Actually, any service provider can benefit their customers by providing them with an automatic consultant whose core component is a question answerer. QA task is closely related to the task of dialogue system management, which is another challenging task in NLP. It makes sense to think of question answering as a step towards dialogue system management, since the former focuses on single-turn conversation and the latter deals with multi-turn one. Building efficient QA system lies in the middle of the journey to bring machine the ability of fully understanding human language.

Question answering can be classified into closed-domain and open-domain types. Closed-domain means that we need to provide the system with a question and the context containing the answer to that question. The system in this case is in charged of extracting the correct answer from the context, which is also known as machine reading comprehension. In contrast, Open-domain QA (ODQA) tries to find the answer without knowing about the context. ODQA is apparently more challenging than closed-domain one. To be able to answer open-domain questions, the system must either memorize knowledge in its internal structure or access to some information source to find information about the answer. Research about ODQA dated back to the 1960s [3]. In these early days, the main method was transforming the question into some form of query statements, then searching for relevant documents in a pre-constructed data source using lookup table [3] or syntactic pattern matching [4]. Starting in 1992, the Text REtrieval Conference (TREC) has significantly pushed the research on ODQA, especially in the aspect of information retrieval. According to the TREC-8 question answering track report [5], most of participants used the following strategy. First, the input question was classi-

fied by its question word, i.e. "how", "what", "who", "why", "when", "where". The system then retrieved a set of documents using standard textual information retrieval method. Based on type of the question, which was determined in the first step, the system parsed the returned documents to get expected answer. Since this strategy is very intuitive, it turns out to become the choice of many later works on ODQA. In recent decades, deep learning or neural networks has extraordinarily boosted research in artificial intelligence (AI). Computer vision and Natural Language Processing are two fields that are the most benifited from this. As a critical task in NLP, ODQA has also moved towards the neural nets era. In 2017, D. Chen et. al proposed a retriever – reader architecture for reading Wikipedia to answer open-domain questions [6]. This work has drawn a huge attention of research community and promoted a large number of subsequent publications on ODQA [7, 8, 9, 10, 11, 12, 13, 14, 15, 1, 16, 17, 18, 19, 20, 21].

Motivated by the active research in ODQA, this Bachelor Thesis attempted to propose to improve existing results on ODQA. Furthermore, the Thesis also conducted a case study of ODQA for Vietnamese COVID-19 data. A demo web interface was also provided to illustrate results from ODQA system.

1.2 Problem formulation

This section provides definition of the Open-domain question answering task. This task is formulated as the followings:

Input:

- A question in human natural language.

  E.g. Who is the founder of Google?

Output:

- A list of answers for the input question.

  E.g. [Larry Page, Sergey Brin]

Constraints:

- The system answers only factoid question. Factoid means that the question is about a fact and often can be answered with a short phrase. Yes/no questions, muliple-choice questions and reasoning questions are not factoid.

  E.g.
  - Factoid question: What is the capital of Vietnam?
  - Reasoning question: If 3 cats can catch 3 mice in 3 minutes, how many mices can 6 cats catch in 6 minutes?

An ideal ODQA is expected to be able to answer any type of questions. To achieve that

goal, however, is very challenging and often requires to build a complex system, which makes the system difficult to deploy into real applications. Besides, it makes more sense to build a system holding the facts about world, not a system that helps us solve math. Many questions even do not have an answer, like “Why does Nam hate Long?”. Answering more complex questions other than factoid questions were covered in other research, but is out of scope of this work.

1.3 Thesis contribution

The contributions of this Bachelor Thesis is as following:

- Propose to improve accuracy of Open-domain question answering system using *stratified loss*.
- Conduct a case study for Open-domain question answering system in Vietnamese language for COVID-19 topic.
- Build a web interface to illustrate results from Open-domain question answering system.

The remaining of this Thesis is organized as follows: chapter 2 shows theoretical preliminaries needed for the work presented in this Thesis. Chapter 3 is the main part of this Thesis, which presents in depth the architecture of Open-domain question answering system and proposals to improve the system performance using *stratified loss*. Method for building an ODQA system for Vietnamese COVID-19 data is also provided in this chaper. Experiment results and analysis are provided in chapter 4. Chapter 5 gives an demo web interface to illustrate results from the system. Finally, conclusion and futures are discussed in chapter 6.

Chapter 2

Theoretical preliminaries

This chapter presents theoretical preliminaries needed for the work of this Thesis, including concepts in Deep Learning and Natural Language Processing.

2.1 Neural Networks

Neural Networks or Deep learning is getting more and more robust for solving many tasks in computer science, especially in Computer Vision and Natural Language Processing (NLP). The building block of neural network is neuron. Neuron in the field of deep learning is inspired by biological neuron in human brain. Figure 2.1 illustrates a neuron along with its components. In short, a biological neuron receives multiple input signals and produce an output if the aggregation of the input signals is greater than some threshold.

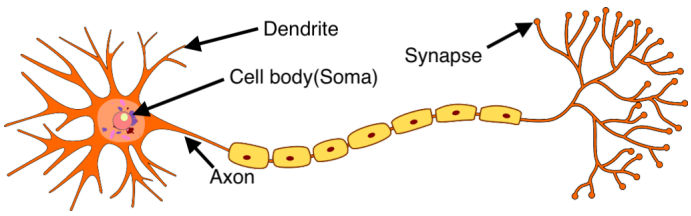


Figure 2.1: Biological neuron. Source: LINK

Artificial neuron has the same mechanism as biological neurons. They take as input some variables, aggregate these variables in some way (normally taking the weighted sum), pass the output through an activation function and produce the final result (Figure 2.2).

A neural network is composed of multiple neurons connecting together, often organized in layer fashion. Since neural network is a computation model, its inputs must be represented in numbers (usually real numbers). Those numbers are fed into the network layer by layer to produce the final output. Figure 2.3 gives an example of a neural network.

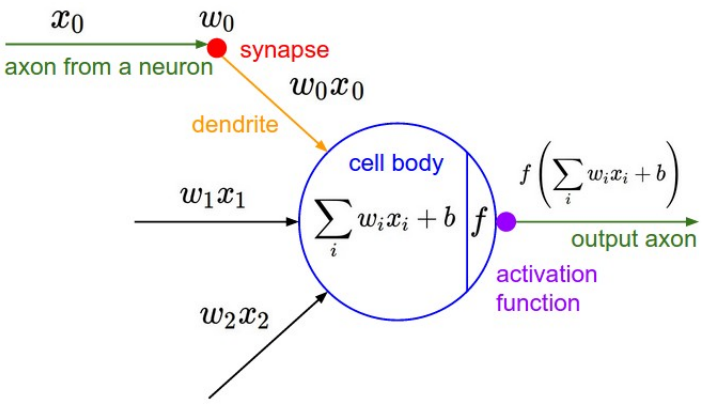


Figure 2.2: Computation of an artificial neuron. Source: LINK

Spefically, this neural network consists of 3 layers: input layer (in green), which is actually the numbers given as input of the network; hidden layer (in red), which take care of neural network computation and output layer (in yellow), which is what the network produces.

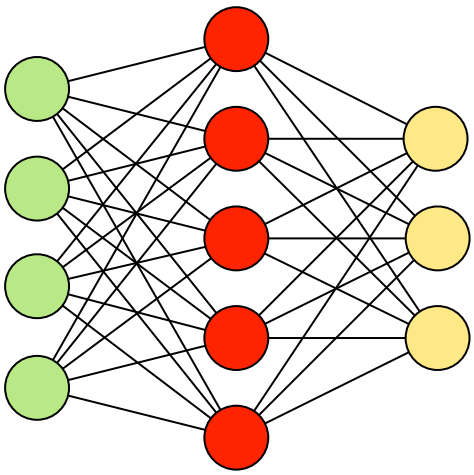


Figure 2.3: A simple neural network

What a neural network actually does is mapping the inputs to the expected outputs. Neural network is designed specially for learning the mapping function from available data. It reduces the tedious task of feature engineering. You just give the network the raw input, no need to worry about what features of the input should be selected to produce output, and then the neural network takes care of the whole process. It is like a magic black box that can give solution to many problems.

Up to now, there have been tons of different types of neural networks being proposed to solve different problems. Although neural network is very robust, it is not trivial to make it work on a specific problem. What data scientists are currently doing is to design good architecture for neural network for various tasks and problems. There are two main classes of neural networks, which are Convolution Neural Networks (CNNs) and Recur-

rent Neural Networks. The former was invented for the image processing tasks and the latter was original used for natural language processing.

2.2 Language Model

Language model is a model that predicts some unseen words based on a given context. For example, given an incomplete sentence and predict the next word (Figure 2.4a). Another example is predict some masked words in a sentece (Figure 2.4b). Language model is very important in the field of NLP since many downstream tasks like named entity recognition, question answering, part-of-speech tagger, etc. need to use it. By correctly predicting unseen words, language model can be considered as understanding the human natural language. Building a language model thus becomes the starting point when researching in NLP. Types of language model includes statistical language model and deep language model (language model using neural network). Along with the development of neural networks, deep language model has also gained many promise results and gradually become the replacement for most of traditional statistical models. It takes time to train a language model to understand human language, often measured in days or weeks. In 2018, Google released a fancy language model called BERT, which then has been used extensively by the research community as well as in production.

(a) (b)

Figure 2.4: Language modeling examples: (a) next word prediction; (b) masked word prediction

2.3 Transformers

In the early days of deep learning in NLP, a very common neural network architecture called Recurrent Neural Networks (RNNs) was proposed and used extensively for various NLP problems. The idea of RNNs is using a shared weights across all input words to learn the contextual representation of words. This idea successfully worked and quickly became default choice for solving many NLP tasks. However, it has several drawbacks: (1) it processes the input words sequentially so parallel computing cannot be utilized to speed up training; (2) since it used shared weights for all input words, the gradient signals when performing backward propagation must go through a very long path, causing to gradient vanishing and gradient exploding. The latter problem was addressed by

two architectures named Long-Short Term Memory (LSTM) and Gated Recurrent Units (GRU), which create a "highway" for gradient signal to properly flow into every part of the network. However, problem of unable to parallelize remained. In 2017, transformers architecture was introduced in [2] and then marked a milestone in NLP. This new architecture addressed all previous problems and produced much better results than the existing baselines. Figure 2.5 describes the architecture of transformers. In essence, it is a sequence-to-sequence model containing two parts: encoder and decoder. The encoder encodes the input sentence into a sequence of vectors in hidden space. The decoder then uses these vectors to produce outputs. Different from RNNs, which compacts the contextual information into a word by sequentially process each word of the sentence from left to right or right to left, transformers uses a mechanism called attention, where each word in the sentence directly attends to all other words. Thus, it avoids the gradient vanishing or gradient exploding problem (there is no long path when flowing gradient signals backward) and can capture contextual information of words that are far apart.

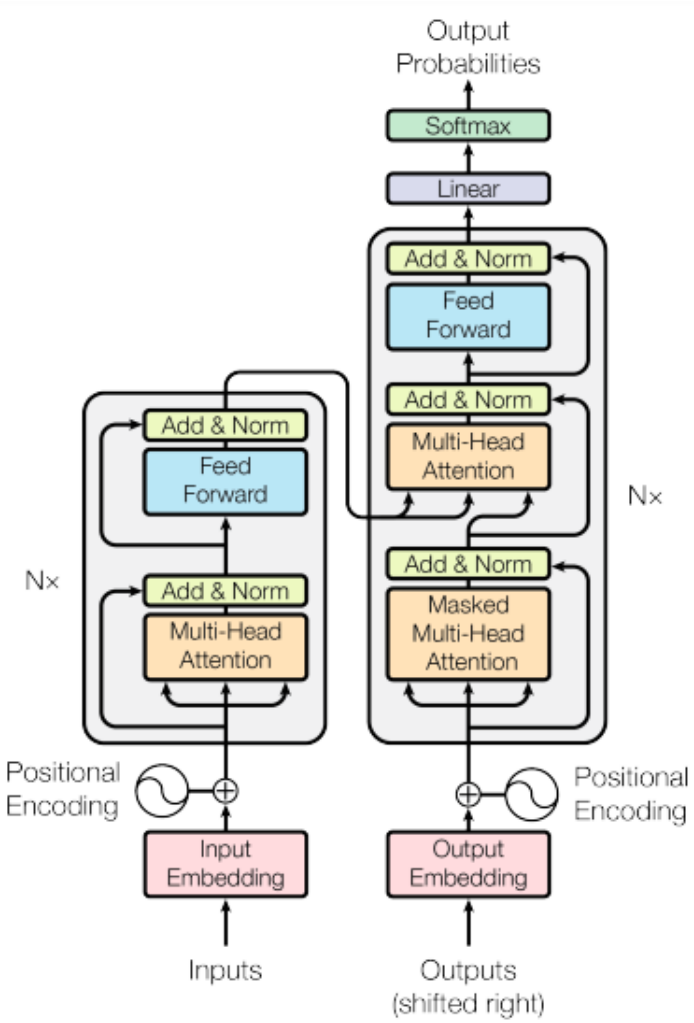


Figure 2.5: Transformers architecture. Source: [2]

Attention mechanism is described as follows. A query Q needs to know which items are important in a list of given items, where each item is represented as a key K and a value

V. To know that, it computes the similarity scores between Q and each key K. How to compute the similarity scores is not restricted. The common choice is dot product or bi-linear function. The similarity scores are then normalized into a probability vector called attention scores, where each value of the attention scores is the importance of the corresponding item. As the last step, Q takes the weighted sum of all values V with respect to the probability vector and get the output as its new representation. The above process is mathematically formulated as belows:

Input

- $Q \in \mathbb{R}^{q \times d}, K \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{n \times d}$, where q is the number of queries, n is the number of items, d is the size of query, key and value vectors.

Computation

- Similarity scores: $S = QK^T \in \mathbb{R}^{q \times n}$
- Attention scores: $\alpha = \text{softmax}(S) \in \mathbb{R}^{q \times n}$

Output

- $\tilde{Q} = \alpha V \in \mathbb{R}^{q \times d}$, the new representation of Q.

Encoder and decoder components of transformers are built by stacking multiple encoder or decoder blocks on top each other. Each block in turn composes of several layers. There are two types of layers in transformers architecture, which are feed-forward layer and multihead attention layer. Multihead attention is basically the same as the abovementioned attention mechanism, but uses attention multiple times and then concatenates the output together. The encoder is responsible for transforming the input sentence, which is in human language, to vectors in a vector space. The output vectors of encoder are expected to hold the contextual information of input words (the contextual information is captured by multihead self-attention layer). In the decoder side, output words are generated sequentially. A special [START] token is first fed into decoder, then goes through a series of decoder blocks, each contains a **masked multihead attention** layer, a normal multihead attention layer with keys and values taken from encoder outputs, and a feed-forward layer. On top of all decoder block is a softmax classifier which produces the probability of the next word. The word with highest probability is chosen as next word and this next word is then concatenated with previously produced words to create input for the next feed forward through decoder. The decoding process stops when a [END] token appears. Here, we have the concept of masked multihead attention layer. During training decoder, we want each word to attend only the words preceeded it. Hence, we mask out the subsequent words by setting attention scores for those words to zeros.

The original transformers applied to Neural Machine Translation task, but in general in can be used in any sequence-to-sequence task such as text summarization, question answering,... After transformers was released, there were appearances of many fancy modern deep language models. These language models either based on encoder or decoder components of transformers. Typical examples are BERT (encoder-based) and T5 (decoder-based). The next section discusses about BERT, which stands for Bidirectional

Encoder Representations from Transformers, released by Google in 2018 [22].

2.4 BERT - The state-of-the-art pretrained language model

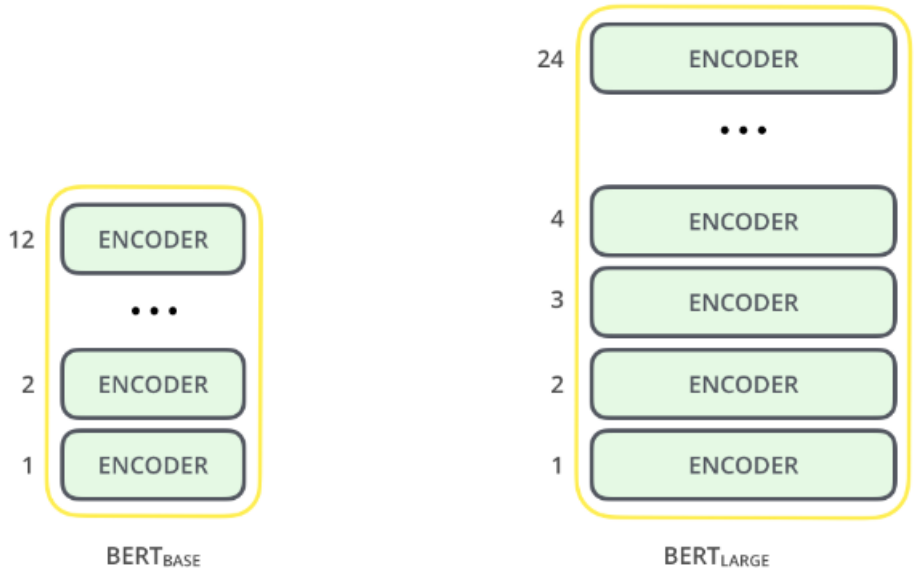


Figure 2.6: BERT architecture. Source: LINK

In recent years, NLP community has seen the word "BERT" ubiquitously. BERT is a language model trained on two tasks: masked language modeling and next sentence prediction. BERT architecture is the same as encoder of transformers (Figure 5). In essence, it encodes the input sentence into a sequence of word vectors, where each word vector holds the contextual information of the corresponding word. These contextual word vectors were proved to significantly boost many downstream tasks in NLP. By learning to correctly predict the masked words and the next sentence, BERT model achieves ability to understand human language. Currently, there have a variety of high-quality pretrained language models, but BERT is a good candidate for starting researching in the field of NLP.

2.5 FAISS -

Chapter 3

A pipeline of Dual Encoder and Cross Encoder architecture for building Open-domain question answering system

This chapter first presents high-level architecture of an Open-domain question answering system, then dives into details of each component in the system. The last section of the chapter considers a case study of Open-domain question answering system for Vietnamese COVID-19 data.

3.1 Open-domain question answering system pipeline

Standing from a non-tech user's point of view, a computer that has the ability of question answering should take as input a human-language question (through typing text or through speech) and produces the answer to the user (also in form of human language). This is exactly what an open-domain question answering is. Another type of question answering system is closed-domain. In this type of system, computer is provided with a context to search for answer alongside input question. Closed-domain question answering is known with a more popular name, which is Machine Reading Comprehension (MRC). MRC problem has been studied for a long time and it is a core component of Open-domain question answering system. Since there is no context provided to the open-domain question answering system, it must retrieve contexts itself based solely on the input question. Look up information for a question and then figure out the answer seems not to be so complicated to human but is very challenging for computer. To perform this task, an open-domain question answering system usually contains two components: a retriever to get relevant contexts for an input question, and a reader to produce answer from the retrieved contexts. Almost modern open-domain question answering system follows this retriever-reader pipeline. Figure 3.1 is an illustration of this pipeline.

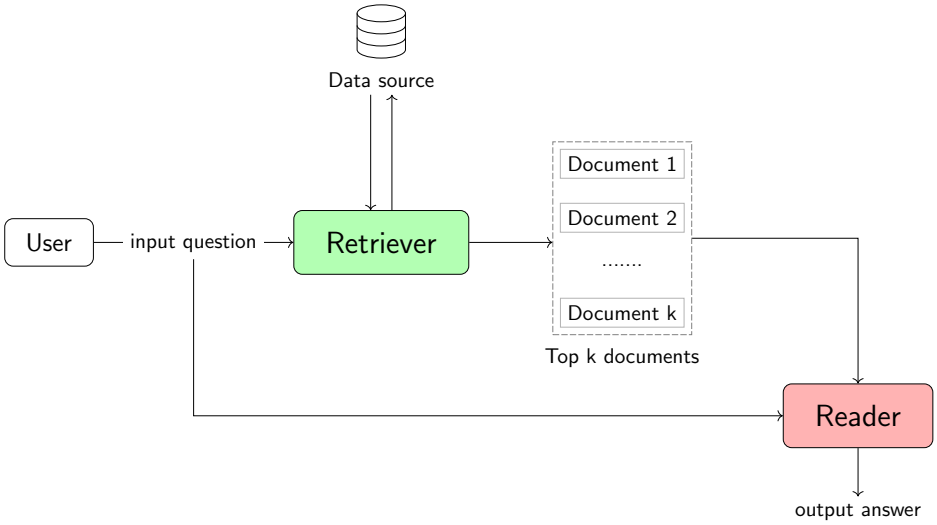


Figure 3.1: Retriever - reader pipeline of Open-domain question answering system

There are many different ways of how to retrieve and how to read. Traditional retriever often get relevant contexts based on keyword overlapping between question and contexts. A well-known algorithm for this type of search is TF-IDF or BM25. This search method is also known as sparse vector searching because questions and contexts are represented by a vector with many zeros. Prior to nowadays' deep learning era, BM25 was the most popular method not only for open-domain question answering, but also for a broader field in computer science which is information retrieval. In recent years, with the development of neural network, a.k.a deep learning and hardware accelerators like GPUs and TPUs, many problems have been addressed by adopting modern neural network model. Specific to open-domain question answering problem, sparse retriever is now replaced by dense retriever, in which question and contexts are encoded to a low-dimension vector space and there is no longer redundant zeros.

On the side of reader, the most common used types are extractive reader and generative reader. Different from retriever, reader does not have a long time of development and both extractive and generative types are based on recent neural network models. Given a question and several contexts (the results of retriever component), extractive reader tries to extract the answer by predicting the start token and end token of the answer inside these contexts (Figure 3.2). In contrast, generative reader is not restricted to produce answers that are contained in contexts but can freely generate any answers. As shown in Figure 3.3, the phrase "Maths teacher" does not appear in the context, but is the right answer for the input question. Intuitively, generative reader is more challenging than extractive one since there is no constraint about the output answers. Actually, generative model needs to learn more than extractive model to get the expected results. This thus lead to the challenge of bigger model size and longer training time. This Thesis studies extractive reader and dense retriever. Dense retriever was shown to better than sparse retriever in existing research [1]. In general, neural networks are now performing much better than traditional methods. However, traditional information retrieval method like BM25 is helpful for generating training data for dense retriever. Although results re-

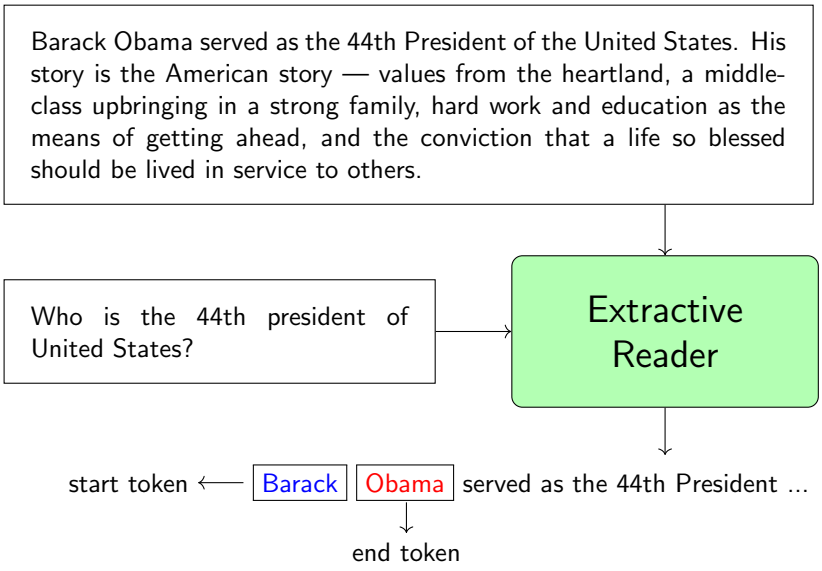


Figure 3.2: Extractive reader

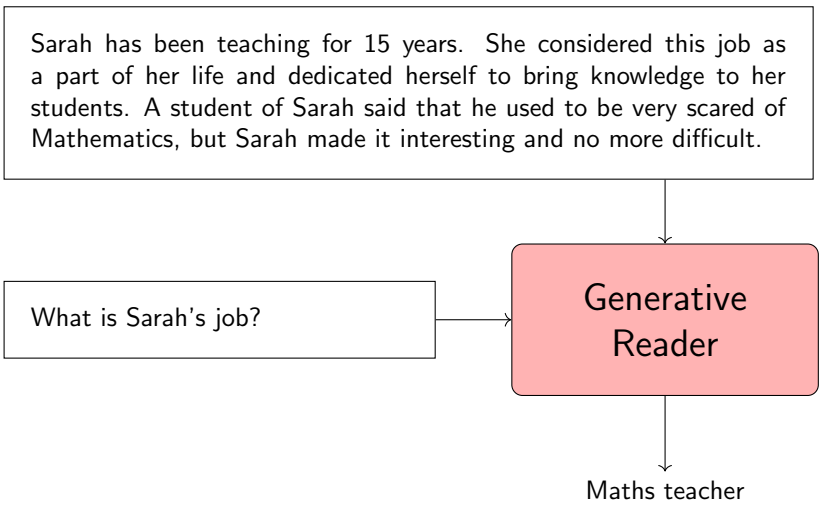


Figure 3.3: Generative reader

turned by BM25 are not always exact, it can significantly reduce time used for manually tagging data since it provides annotator with several potential choices. Later, annotator can decide whether or not to use the results of BM25.

3.2 Dual encoder and cross encoder architecture of Open-domain question answering system

As aforementioned, open-domain question answering system includes two core components, which are retriever and reader. Retriever and reader implemented in this Thesis are dense retriever and extractive reader, in which dense retriever follows dual encoder architecture and extractive reader follows cross encoder architecture. The following sections provides these two architectures in detail.

3.2.1 Dense retriever

Retriever model used in this Thesis is based on dense retriever model proposed in [1]. From the base model, this Thesis has proposed to use *stratified loss* function to improve model's accuracy. Details about retriever model are presented in the following sections.

Model architecture

Dense retriever follows dual encoder architecture which contains two neural networks, which are question encoder and context encoder. Each encoder can be any type of neural network that take as input a text sequence and produce a sequence of embedding vectors for that sequence. How dense retriever works is illustrated in Figure 3.4.

As shown in Figure 3.4, work flow of dense retriever contains two phases. Offline phase takes long time to complete but we only need to do this once. In this phase, we need a data source which contains large number of contexts. Each context goes through the context encoder and output an embedding vector. These embedding vectors are then indexed using faiss indexer [23], in which faiss is an efficient approximate similarity search algorithm that can scale to billion vectors. In the online phase, system receives the input question and encodes that question to an embedding vector using question encoder. Next, this question embedding vector is fed to faiss indexer to perform k-nearest neighbor search. The final output of the online phase is top k most relevant contexts to the input question, which then further be fed to the reader to produce the final answer.

Training retriever

To train the dense retriever, this Thesis simultaneously trained the context encoder and the question encoder. Subsequent sections presents details of the training process.

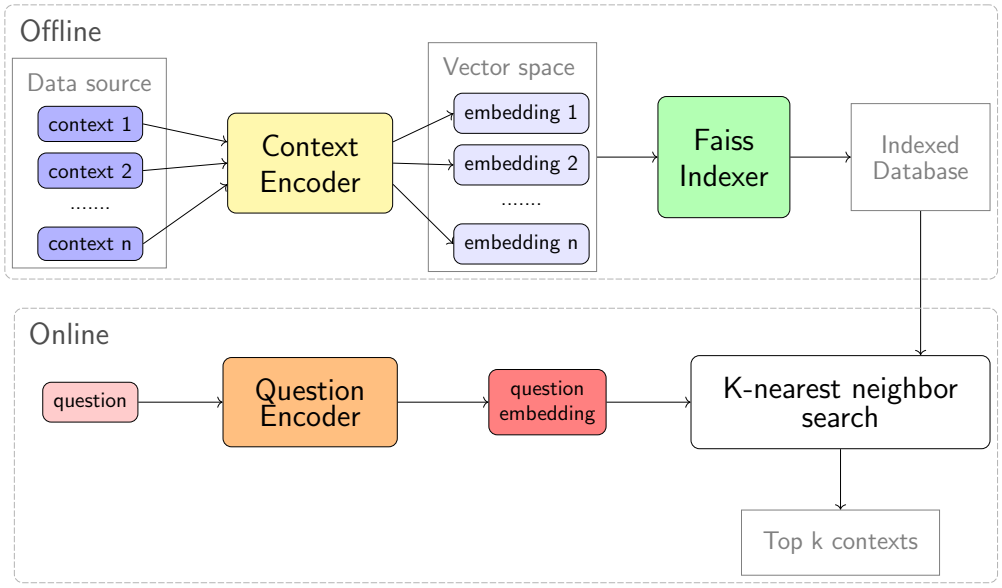


Figure 3.4: Dense retriever pipeline

a) Training data

Each training sample contains the followings:

- 1 question;
- 1 positive context, which is the context that has answer for the question;
- m negative contexts, which do not contain answer for the question;

An example of a training sample will look like Figure 3.5.

```
{
    "question": "Who sang the song Mirrors?",
    "positive context": "\"Mirrors\", also titled \"Mirrors (I Don't Wanna Lose You Now)\", is a song recorded by
    ↪ American singer-songwriter Justin Timberlake for his third studio album, The 20/20 Experience (2013).
    ↪ First conceived in 2009, the track was inspired by his relationship with Jessica Biel and the marriage
    ↪ of his grandparents.",
    "negative contexts": [
        "The World's Billionaires is an annual ranking by documented net worth of the wealthiest billionaires
        ↪ in the world, compiled and published in March annually by the American business magazine
        ↪ Forbes. The list was first published in March 1987",
        "Harry Potter is a series of seven fantasy novels written by British author J. K. Rowling. The novels
        ↪ chronicle the lives of a young wizard, Harry Potter, and his friends Hermione Granger and Ron
        ↪ Weasley, all of whom are students at Hogwarts School of Witchcraft and Wizardry.",
        .....
    ]
}
```

Figure 3.5: Retriever training sample. Blue text is answer for the sample's question.

b) Loss function

Let $\mathbb{E}_Q, \mathbb{E}_C$ be the question encoder and context encoder respectively. Let $D_i = \left\langle q_i, p_i^+, \left\{p_{i,j}^-\right\}_{j=1}^m \right\rangle$

be the i-th training sample, where:

- q_i: the question;
- p_i^+: the positive context;
- $\left\{p_{i,j}^-\right\}_{i=1}^m$: m negative contexts

After forwarding D_i to the retriever, we will get:

- $qe_i = \mathbb{E}_Q(q_i)$: embedding vector of the input question;
- $pe_i^+ = \mathbb{E}_C(p_i)$: embedding vector of the positive context;
- $pe_{i,j}^- = \mathbb{E}_{\mathbb{C}}(p_{i,j}^-),\ j = \overline{1,m}$: embedding vectors of m negative contexts.

Objective of training retriever is to maximize similarity between qe_i and pe_i^+ while minimizing similarity between qe_i and $pe_{i,j}^-,\ j = \overline{1,m}$. Similarity score can be calculated using any metrics. But to serve the purpose of later indexing using faiss algorithm, the metric should be one of L2 norm, cosin and dot product. These metrics are supported by faiss. To obtain the training objective, we use negative log likelihood loss function as below:

$$\mathscr{L}_i = -\log\left\{\frac{\exp\left[\text{sim}\left(qe_i, pe_i^+\right)\right]}{\exp\left[\text{sim}\left(qe_i, pe_i^+\right)\right] + \sum_{j=1}^{m}\exp\left[\text{sim}\left(qe_i, pe_{i,j}^-\right)\right]}\right\} \tag{3.1}$$

, where *sim* is the similarity function.

Here, $\mathscr{L}_i$ is the loss value over the i-th training sample. Suppose we have n training samples, the loss value over the training dataset is average of loss value over each training sample:

$$\mathscr{L} = \frac{1}{n}\mathscr{L}_i \tag{3.2}$$

To efficiently train the retriever using this loss function, Karpukhin et. al [1] has proposed to use the in-batch strategy, in which negative contexts of a training sample are taken from positive context of other samples in the same batch. Thus, each forward to the retriever model requires much less data than normally using seperate negative contexts for each training sample. Alongside this in-batch strategy, the authors use another type of negative context called hard negative context. Hard negative contexts do not contain answer for the input question but are very similar to the positive one. These hard negative contexts are often obtained by using another algorithm like BM25 or the retriever model in the previous epoch of training process. Using hard negative contexts and in-batch loss were shown to be very efficient and those are the keys to the success of dense retriever model. Karpukhin et. al used one hard negative context per a sample and they pointed out that using 2 or more hard negative contexts did not improve the results (using 2 hard negative contexts was even worse). However, a closer look into the model shows that dense retriever can easily distinguish between positive context and non-relevant negative contexts, but has difficulty distinguishing positive context and hard negative ones. Thus,

this Thesis propose a loss function called *stratified loss* which can help the model better learning the difference between positive and hard negative contexts.

The idea of stratified loss function is simple. It is sum of two loss functions, where each is a negative log likelihood loss as presented above. The difference between these two loss functions is that the first is designed to learn to differentiate between positive and hard negative contexts, while the second learns to differentiate between hard negative and normal negative ones. The consequence of using stratified loss is that the model will learn to know which context is more negative. In contrast, the original loss function treats all negative contexts the same. Mathematically, stratified loss is computed as followings:

Inputs:

- A batch $\mathscr{D}$ of b training samples. Here, embedding vectors are used instead of raw text input since loss function only applies to embedding vectors (state differently, loss calculation is performed after forwarding raw text input through retriever model). Each sample $\mathscr{D}_i$, $i = \overline{1,b}$ contains:
  - q_i: question embedding;
  - p_i^+: positive context embedding;
  - $p_{i,j}^-$, $j = \overline{1,w}$: w hard negative context embeddings.

Loss formula:

$$\begin{aligned}\mathscr{L}_i = \quad & -\log\left\{\frac{\exp\left[\text{sim}\left(q_i, p_i^+\right)\right]}{\exp\left[\text{sim}\left(q_i, p_i^+\right)\right] + \sum_{j=1}^{w}\exp\left[\text{sim}\left(q_i, p_{i,j}^-\right)\right]}\right\} \\ & -\sum_{j=1}^{w}\log\left\{\frac{\exp\left[\text{sim}\left(q_i, p_{i,j}^-\right)\right]}{\exp\left[\text{sim}\left(q_i, p_{i,j}^-\right)\right] + \sum\limits_{k\in\{1,2,\ldots,b\}\setminus\{i\}}\exp\left[\text{sim}\left(q_i, p_k^+\right)\right]}\right\}\end{aligned} \tag{3.3}$$

The first term of equation (3.3) is written exactly the same as in equation (3.1), but note that hard negative contexts are used instead of normal negative ones. The second term is what stratified loss differs from the original loss, which makes retriever give higher similarity score (with respect to the input question) to hard negative contexts than normal negative ones. This loss formula is taken over a batch of input samples, not a single one. The benefit is each single sample contains only positive and hard negative contexts while normal negative contexts are taken from other positive contexts in the same batch.

3.2.2 Extractive Reader

A reader, or a machine reader, is one that can read a set of documents and output the answer. While retriever "skims through" a huge amount of documents to get a much smaller

document subset, reader "swallows" a small amount of documents to produce the exact answer. In case of extractive reader, if number of documents to be read is greater than 1, reader needs to perform 2 tasks: re-ranking and reading. Re-ranking means selecting the best document out of the documents retrieved by the retriever. Each document after re-ranking has a re-rank/selection score and the document with the highest score is selected. Reading or extracting is to find the start and end position of the answer inside the selected document. The work of re-ranking and reading can be implemented using a common neural networks or two seperate ones. To avoid confusing, let single-document reader be used to refer to the reader that reads only one document at a time. This Thesis uses two different neural networks for the re-ranker and the single-document reader. Computational cost of re-ranker is much more expensive than single-document reader since re-ranker operates on a set of documents while single-document reader only needs to operate on a single one. Hence, seperating re-ranking and reading work has the advantage of training the single-doc reader with more data during training re-ranker. Beside, we can flexibly use different neural networks architecture for these two components.

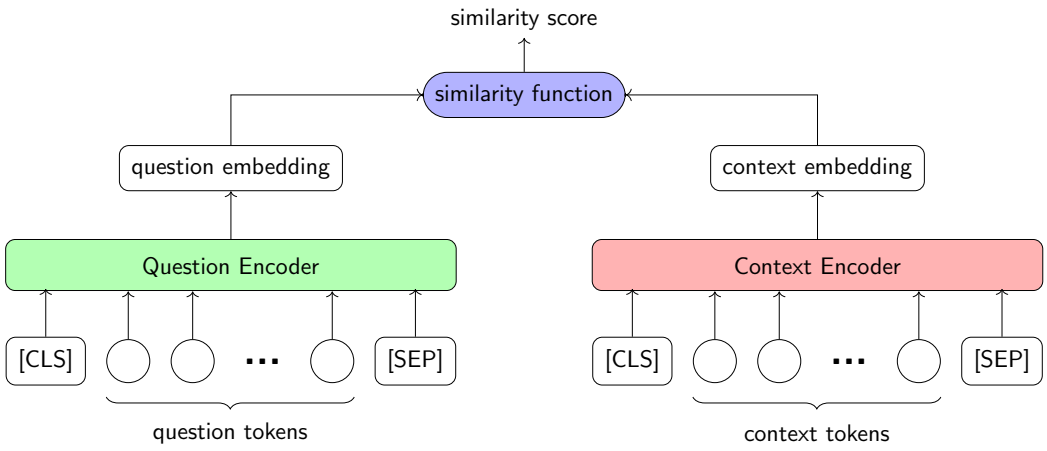
Model architecture

a) Re-ranker

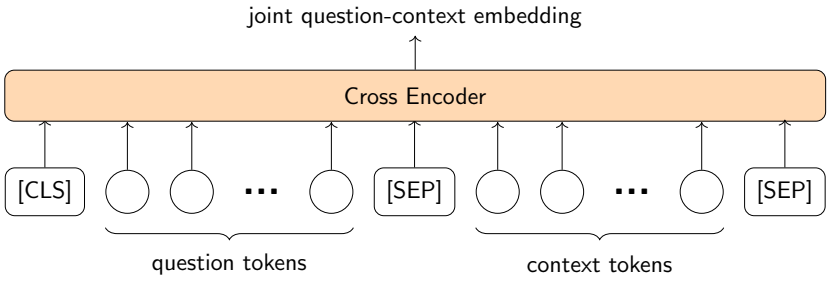
The task performed by re-ranker is quite similar to what is done by retriever. Both re-ranker and retriever assigns some similarity score to the document with respect to the input question. The difference is at the scope where retriever and re-ranker work. Since retriever deals with huge amount of documents, it does not need to thoroughly look into each single document but only acts as a simple filter. The task of actually reading each single document is left to the extractive reader, which contains re-ranker and single-document reader. In the literature, there are two neural architectures used to compute similarity between input question and documents. The first is known as dual encoder or bi-encoder, which is the architecture of previously presented retriever. The second is called cross encoder, which is usually used for the reader (all types of reader, including re-ranker, single-document reader and generative reader). Figure 3.6 shows the difference of the two architectures.

Re-ranker used in this Thesis follows cross encoder architecture. As shown in Figure 3.6b, cross encoder jointly encode question and context into one single embedding vector. Given a question and n contexts, we concatenates the question to each context, adds special [CLS] and [SEP] tokens and then feed each concatenated sequence to the re-ranker. At the output, we have n jointly question-context embedding vectors. These vectors then go through a softmax classifier to produce a probability vector. The context that has the highest probability is finally chosen as the best context (the context that is the most likely to have the answer). Figure 3.7 is the full picture of re-ranker. Mathematically, computation of re-ranker happens as follows:

- Input:
  - $\mathbf{X} \in \mathbb{R}^{K \times L \times d}$: K input sequences, each of which is the concatenation of the question and a context. Each token of a sequence is represented by vector in $\mathbb{R}^d$. Here, d is



(a)



(b)

Figure 3.6: Dual encoder and cross encoder architecture: (a) Dual encoder; (b) Cross encoder

the embedding vector size and L is length of input sequence.

- Joint question-context embedding vector of i-th input sequence $\mathbf{X}_i$ ($\mathbf{X}_i \in \mathbb{R}^{L \times d}$):
  - $\mathbf{H}_i = crossEncoder(\mathbf{X}_i) \in \mathbb{R}^{L \times d}$
  - $\mathbf{E}_i = pooled(\mathbf{H}_i) \in \mathbb{R}^d$.
    pooled is a function that take as input a sequence of vectors and return a single one. Popular pooling function is max pooling, mean pooling, averaging pooling, .etc. Since cross encoder allows each token in the input sequence to interact with all others, a more powerful pooling operator is using directly [CLS] token. Embedding vector corresponding to this token is also the embedding vector of the whole sequence.
- Projecting to scalar value:
  - $\alpha_i = \mathbf{E}_i^T \mathbf{w} \in \mathbb{R}$: scalar value corresponding to the i-th input sequence. The higher this value is, the more relevant the question and the context are.
- Softmax function: Let $\alpha = (\alpha_1, \alpha_2, ..., \alpha_K) \in \mathbb{R}^K$, then:
  - $\tilde{\alpha} = \text{softmax}(\alpha) \in [0, 1]^K$: probability distribution over K input contexts.

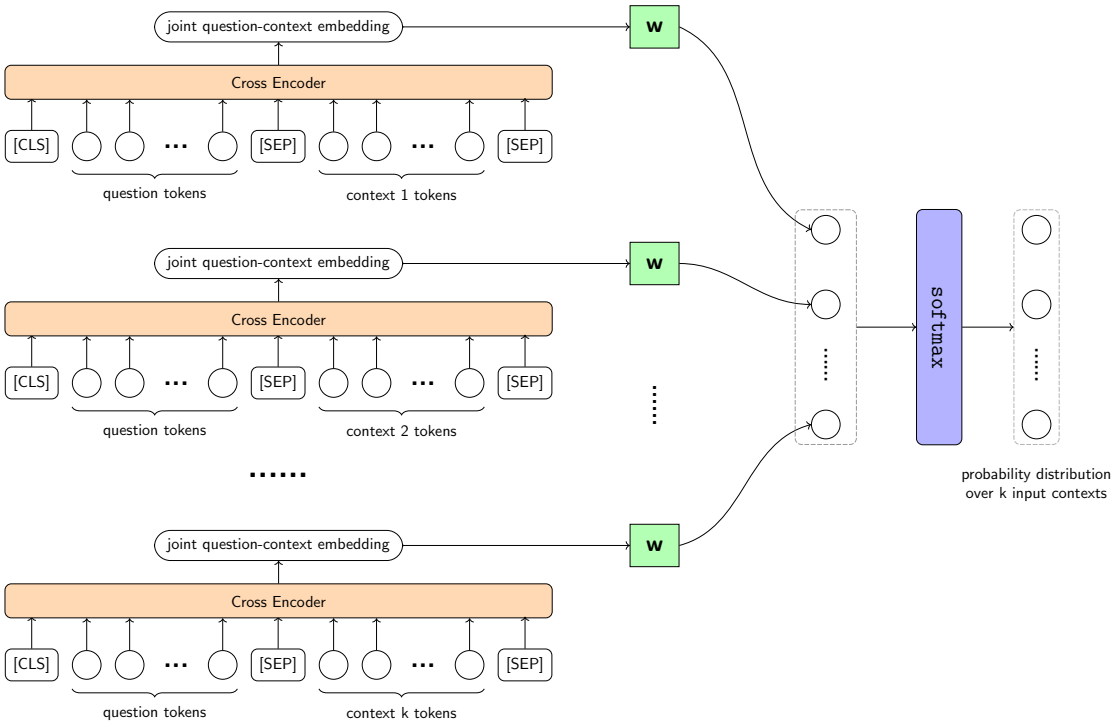


Figure 3.7: Model architecture for re-ranker

b) Single-document reader

Single-document reader also uses cross encoder architecture. Cross encoder architecture is very suitable for single-document reader since it deals with the interaction among tokens in a sequence, i.e. every token in question and context interacts with each other. These interactions cannot be captured directly in dual encoder architecture. Most part of single-document reader is the same as re-ranker. The difference is that single-document reader produce two probability distributions, one for predicting answer's start position

and one for predicting answer's end position, while re-ranker produces one probability distribution for predicting the best context. Figure 3.8 shows the detailed architecture of the single-document reader. It looks a bit different from Figure 3.7 and Figure 3.6b because there is an extra *token embeddings* block. This turns out to be no different at all. Since re-ranker only considers the embedding vector of the whole question-context sequence, token embeddings part is hide for saving space.

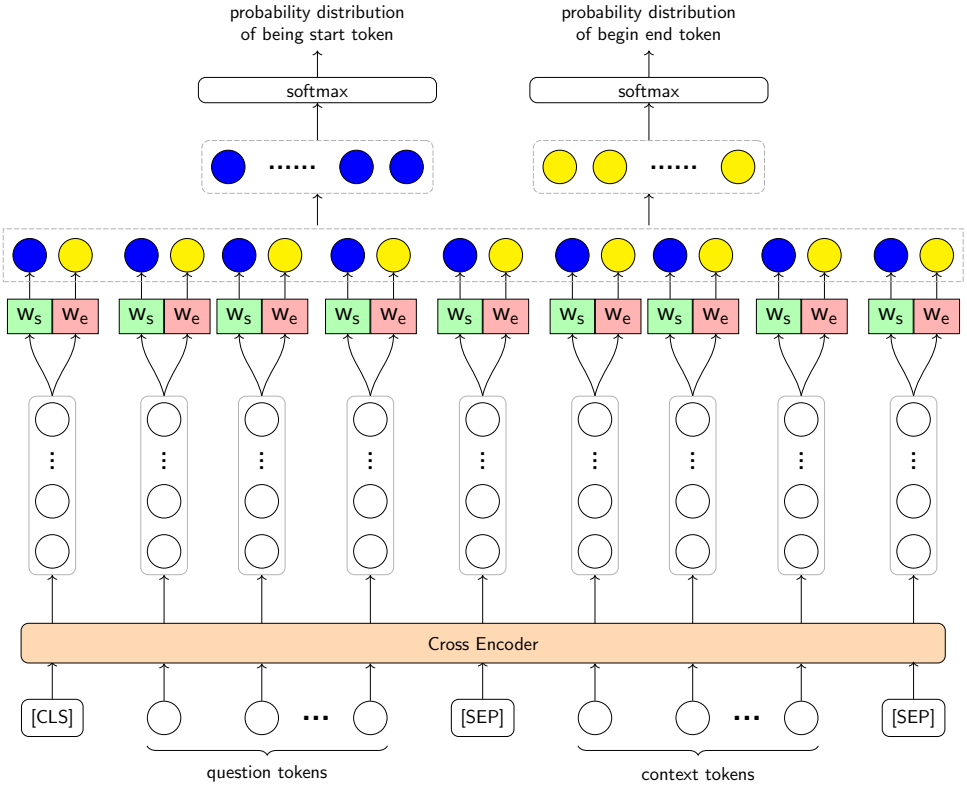


Figure 3.8: Single-document architecture

Computation flow of single-document reader is described as follows:

- Input
  - $\mathbf{X} \in \mathbb{R}^{L \times d}$: A sequence of length L composed of question token embeddings and context token embeddings. Each embedding vector has size d.
- Forwarding through cross encoder
  - $\mathbf{H} = crossEncoder\,(\mathbf{X}) \in \mathbb{R}^{L \times d}$
- Project to scalar value
  - $\alpha = \mathbf{H}.\mathbf{w}_s \in \mathbb{R}^L$, where $\mathbf{w}_s \in \mathbb{R}^{d \times 1}$
  - $\beta = \mathbf{H}.\mathbf{w}_e \in \mathbb{R}^L$, where $\mathbf{w}_e \in \mathbb{R}^{d \times 1}$
- Softmax function
  - $\tilde{\alpha} = \text{softmax}(\alpha)$: probability distribution of being start token.
  - $\tilde{\beta} = \text{softmax}(\beta)$: probability distribution of being end token.

Training reader

a) Training re-ranker

Training data

Training data for re-ranker is described as follows:

- $\mathscr{G} = \left\{\left\langle q_i, p_i^+, \left\{p_{i,j}\right\}_{j=1}^n \right\rangle\right\}_{i=1}^m$: dataset containing m training samples, in which:
  - q_i: i-th input question
  - p_i^+: i-th positive context
  - $\left\{p_{i,j}\right\}_{j=1}^n$: n negative contexts in the i-th training sample.

Loss function

Loss function used to train re-ranker is negative-log likelihood. Re-ranker tries to maximize the probability of the positive context over all input contexts.

- Assumptions
  - e_i^+: joint question-context embedding vector of i-th input question and i-th positive context produced by cross encoder.
  - $\left\{e_{i,j}^-\right\}_{j=1}^n$: n joint question-context embedding vectors of i-th input question and n corresponding negative contexts of this input question.
- Loss formula

$$\mathscr{L}_i = -\log\left\{\frac{\exp\left[\text{sim}\left(e_i^+\mathbf{w}\right)\right]}{\exp\left[\text{sim}\left(e_i^+\mathbf{w}\right)\right] + \sum_{j=1}^{n}\exp\left[\text{sim}\left(e_{i,j}^-\mathbf{w}\right)\right]}\right\} \tag{3.4}$$

b) Single-document reader

Training data

- $\mathscr{H} = \{L_i, s_i, e_i\}_{i=1}^m$: dataset containg m samples, in which:
  - L_i: concatenation of a question and a positive context (context that has answer).
  - s_i: start position of the answer in the concatenated sequence L_i
  - e_i: end position of the answer in the concatenated sequence L_i

Loss function

- Assumptions
  - h_i: joint question-context embedding vector produced by cross encoder.
  - α_i: probability distribution of being the start token.

  - β_i: probability distribution of being the end token.
- Loss formula

$$\mathscr{L}_i = -\log \alpha_i[s_i] - \log \beta_i[e_i] \tag{3.5}$$

Note that this loss function contains two parts. The first part is the negative log-likelihood loss for the start token and the second is also negative log-likelihood loss but for end token. Loss formula in equation (3.5) means that the model will try to maximize probability of joint probability of s_i being the start token and e_i being the end token.

3.3 Building Vietnamese question answering system for COVID-19 topic

This section present an use case of open-domain question answering system for COVID-19 topic in Vietnamese language. To build Vietnamese question answering system for COVID-19 topic, the first thing to do is preparing dataset. Currently, there is no publicly avaible dataset specifically built for open-domain question answering for COVID-19 topic. Hence, following sections will show the details about the process of crawling, annotating and preparing a COVID-19 dataset.

3.3.1 Data crawling

To build open-domain question answering system, the first thing is to build a data source. Data source or context source is where retriever looks for information to answer question. It should be large enough to cover almost information required by users. As one of the largest data source on the Internet, Wikipedia contains millions of articles, cover information about any thing in the world. Since data about COVID-19 is restrict in a narrow domain, it is likely that several thousands of contexts can cover everything in the domain.

This Thesis targets to build a data source that contains about 200.000 contexts. Although these large amount of contexts are not required to cover everything about COVID-19 topic, this large amount challenge the system and thus we can measure how well the system can perform. Usually, text documents are not simply a sequence of continuous words. They are organized into sections, subsections,..., has highlight or emphasize part. These structures contains rich information about the document. To capture some of these structure information, each document used to train open-domain question answering system is represented as the concatenation of document title and document content.

This Thesis focuses on crawling COVID-19 data. The remain contexts in the data source can be about any domain. just need to be large enough to challenge the system. A common data source that one might think of is Wikipedia. However, the pattern of data in Wikipedia is very different from data related to COVID-19 topic, thus might be too easy for the system. Hence, this Thesis uses context source in medical domain, in which

COVID-19 is a sub-domain. Since crawling data source is not the focus of this Thesis, as well as this work is very time consuming, the data source is taken from Social Data Lab of CIST Institute of CMC Corporation, in which the author of this Thesis currently works. These data are taken with permission and is not publicly available. It is used only in the scope of this Thesis.

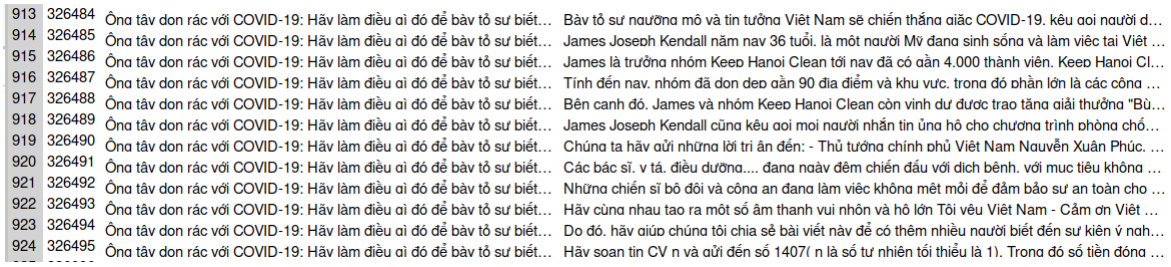
COVID-19 data is crawled using `requests-html` library of python, which is a library capable of retrieve dynamic javascript content of the web page. Data is crawled from several Vietnamese medical websites, includes:

- https://suckhoecongdong.net.vn/
- https://yhoccongdong.com/
- https://vinmec.com/vi/
- https://ncov.moh.gov.vn/
- https://www.doctors24h.vn
- https://suckhoehangngay.vn
- https://danang.gov.vn
- https://tuoitre.vn
- https://vienyhocungdung.vn
- https://nhidong.org.vn
- https://www.doctors24h.vn
- https://www.tcsuckhoe.com
- https://mamamy.vn
- http://www.medinet.hochiminhcity.gov.vn

Raw html content is first processed by removing html tag using regular expression. Regular expression is also used to capture headings or sections of the crawled article. Since the retriever and reader is a neural network based on BERT-architecture, which can process up to 512 tokens in a sequence, crawled article must be break into several smaller passages. Specifically, each passage must have number of single word inside the range (70, 120).

MySQL database is used to store data. In total, there have been 1470 passages that are stored in MySQL database. Storaging data in database is more beneficial than just using normal files because we can easily perform CRUD operators using database.

An example of crawled COVID-19 data is given in Figure 3.9. As shown in this figure, each context source has title (the second column) and content (the third column). Usually, several contexts has the same title, since these contexts come from one single article on the Internet.



913	326484	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	Bày tỏ sự ngưỡng mộ và tin tưởng Việt Nam sẽ chiến thắng giặc COVID-19. kêu gọi người d…
914	326485	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	James Joseph Kendall năm nay 36 tuổi. là một người Mỹ đang sinh sống và làm việc tại Việt …
915	326486	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	James là trưởng nhóm Keep Hanoi Clean tới nay đã có gần 4.000 thành viên. Keep Hanoi Cl…
916	326487	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	Tính đến nay. nhóm đã don dẹp gần 90 địa điểm và khu vực. trong đó phần lớn là các công …
917	326488	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	Bên cạnh đó. James và nhóm Keep Hanoi Clean còn vinh dự được trao tặng giải thưởng "Bù…
918	326489	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	James Joseph Kendall cũng kêu gọi mọi người nhắn tin ủng hộ cho chương trình phòng chố…
919	326490	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	Chúng ta hãy gửi những lời tri ân đến: - Thủ tướng chính phủ Việt Nam Nguyễn Xuân Phúc. …
920	326491	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	Các bác sĩ. y tá. điều dưỡng.... đang ngày đêm chiến đấu với dịch bệnh. với mục tiêu không …
921	326492	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	Những chiến sĩ bộ đội và công an đang làm việc không mệt mỏi để đảm bảo sự an toàn cho …
922	326493	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	Hãy cùng nhau tạo ra một số âm thanh vui nhộn và hô lớn Tôi yêu Việt Nam - Cảm ơn Việt …
923	326494	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	Do đó. hãy giúp chúng tôi chia sẻ bài viết này để có thêm nhiều người biết đến sự kiện ý nghĩ…
924	326495	Ông tây don rác với COVID-19: Hãy làm điều gì đó để bày tỏ sự biết…	Hãy soạn tin CV n và gửi đến số 1407(n là số tự nhiên tối thiểu là 1). Trong đó số tiền đóng …

Figure 3.9: Example of a context

3.3.2 Data annotating

Training data for open-domain question answering system needs to be annotated. Data for training an open-domain question answering system should contain the following fields:

- question: a human-language question
- answers: a list of answers corresponding to the question
- positive contexts: a list of contexts that contains at least one of the answers
- hard negative contexts: a list of contexts that related to the question, i.e. has the same keywords as the question, mention the same thing as in the question, .etc, but not contain the answer.

Intuitively, more positive contexts and more hard negative contexts is better for training the system. But using more of these contexts results in significantly increasing training time while may not be really necessary. Beside, the job of annotating data is very time consuming. Hence, only one positive and one hard negative is annotated per an input question.

Each training sample is annotated as follows. Given a context, which is one of the crawled context from data crawling phase, we need to make a question based on this context, satisfying that answer for that question is contained with the context. This context is then consider as positive context for the question. After making question and extract answers from positive context, the next thing is to make a hard negative context. Hard negative context usually mentions the same object as in positive context but does not provide information to answer the question. For example, when an user asks "Number of new COVID-19 cases in Hanoi today", positive context exactly mention this information, while hard negative context may talk about "number of new COVID-19 cases in Ho Chi Minh city". Choosing hard negative context is totally dependant on the annotator and is the most time consuming part of annotating data.

To efficiently annotate data, this Thesis uses `django`, which is a python web framework. This framework provide an easy way for CRUD data in web interface. Figure 3.10 shows the web interface for annotating data. There are several rules applied for annotating data as follows:

- At least one of answers to the question must be contained in positive context;

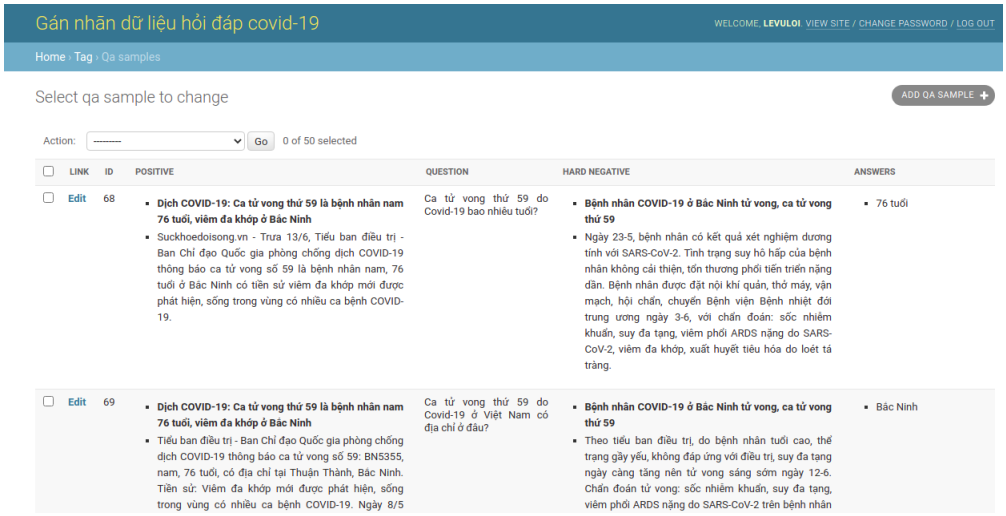


Figure 3.10: Annotating data web interface using `django`

- A single answer to a question should not be longer than 10 single words;
- Length of hard negative context in single words should be in the range (70, 150);
- Avoid yes/no question
- Avoid question that requires reasoning

Note that the open-domain question answering considered in this Thesis is targeted at answering factoid question. The reader component of the system is extractive reader, which is not suitable for answering question that requires reasoning. Since yes/no question also requires reasoning, we try to avoid this type of question. Usually, answer for a factoid question is short, and the design of extractive reader does not perform well for long answer. Other rules are already mentioned in previous sections.

In the final, we have total of 630 training samples.

3.3.3 Training the system on annotated data

Training open-domain question answering system for Vietnamese COVID-19 dataset follows the architectures and processes presented in section **??** and section **??**. For pretrained Vietnamese language model, this Thesis uses `NlpHUST/vibert4news-base-cased` from `huggingface`. For Vietnamese, there are also another well-known pretrained language model, which is `PhoBERT` [24]

Chapter 4

Experimental results

This chapter focuses on conducting experiments on dense retriever. All experiments are conducted using Natural Question dataset [25]. The built Vietnamese COVID-19 dataset mentioned in the previous section serves as an use case of applying open-domain question answering in real life. This dataset is still small and has not been thoroughly experimented. For the extractive reader, this Thesis tries to reproduce the bench mark results in the literature following the existing architecture and configurations.

4.1 Datasets

4.1.1 Natural Question

Natural question is a dataset provided by Google. This dataset contains real user queries issued to the google search engine [25]. Natural Question contains annotated data for answering both long answer and short answer, where short answer is the exact phrase that represents the answer, and long answer is the smallest HTML bounding box that contains all information required to answer the question. In total, there are 307,373 training samples, 7,830 development samples and 7,842 test samples. Each sample contains a question, a context corresponding to that question. start position and end position of long and short answers inside that context.

Natural Question is built specifically for question answering problem. Context source of Natural Question dataset is from Wikipedia. This Thesis doest not perform processing Natural Question dataset from the beginning, but use a pre-processed data provided by [1]. A training sample for dense retriever of this processed dataset look like Figure 4.1.

As shown in Figure 4.1, each training samples has a question, a list of answers, a list of positive contexts, a list of negative contexts and a list of hard negative contexts. Each context has a title and a text content. According to [1], each context has 100 single words, retrieved by non-overlapping breaking an Wikipedia article into small contexts. In the end, the processed dataset contains 58,880 training samples, 8,757 development samples

```
[
    {
        "question": "....",
        "answers": ["...", "...", "..."],
        "positive_ctxs": [{
            "title": "...",
            "text": "..."
        }],
        "negative_ctxs": ["..."],
        "hard_negative_ctxs": ["..."]
    },
    ...
]
```

Figure 4.1: Natural Question training data sample for dense retriever

and 3,610 test samples. For the context source, there is totally 21,015,324 contexts.

4.1.2 Vietnamese COVID-19 dataset

This section recap some features of the built Vietnamese COVID-19 dataset. This dataset contains 630 training samples. The development and test set have not been built yet. Each training sample contains a question, a list of answers to that question, a positive context and a hard negative context. Compare to Natural Question dataset, COVID-19 dataset is much smaller. However, COVID-19 dataset serves as an use case of open-domain question answering system when applying to Vietnamese language. In the aspect of application, this dataset can completely be used to build a well-performed system. But in the research aspect, this dataset needs to be improved and expanded.

4.2 Metrics

4.2.1 Top-k hits score

The most popular metrics to measure the performance of dense retriever is top-k hits. The dense retriever gets a top-k hit if at least one of k contexts returned by it contains answer for the input question.

4.2.2 Exact match

To estimate performance of extractive reader as well as end-to-end system, exact match metrics is used. As the name suggests, the system get one exact match hit if the answer returned by the system matches exactly one of the true answers.

4.3 Experimental settings

4.3.1 Environment settings

All experiments are conducted using Google Cloud TPUs and Google Compute Engine. Table 4.1 gives details about these devices.

Table 4.1: Devices for running experiments

Cloud TPUs	VM Compute Engine
TPU v3-8 on-demand: • TPU version 3 • 8 TPU cores • 16GiB memory / TPU core	• OS: Ubuntu 20.04 • Disk: 30GB • RAM: 16GB • nCPUs: 4

TPUs are very beneficial for training huge data. It is much faster than using several GPUs simultaneously. As reported in [1], the author used eight 32GB GPUs for training model, 50 2-gpu nodes for inference (generating embedding vectors for 21 millions contexts) and a server with Intel Xeon CPU E5-2698 v4 @ 2.20GHz and 512GB memory for indexing and retrieving embedding vectors. These are a huge computational resources that seems impossible for one to have. This Thesis successfully reproduce the training and inference process on Cloud TPUs. Price for hiring a TPU v3-8 on-demand on Google Cloud is 8$ USD/hour. This is much cheaper than hiring GPUs, but still a very high price. Luckily, Google offers a TPU Research Cloud (TRC) program with access to 5 TPU v3-8 on-demand, 5 TPU v2-8 on-demand and 100 preemptible TPU v2-8. Details about types of Cloud TPUs is available at `https://cloud.google.com/tpu/`. Also, user of Google Colab can access Cloud TPUs as well but for a very limited time (less than 12 hours).

Training and inference of the system is implemented using `Tensorflow 2.4.1`. which is one of the most well-known framework for Deep Learning. Using tensorflow is the best way to run code on Cloud TPU. Both tensorflow and Cloud TPUs are owned by Google. Hence, they are designed for working best together.

4.3.2 Data settings

Testing performance of dense retriever requires huge amount of time. We need to generate embedding vectors for all 21 millions contexts in the data source, which takes 40 minutes on 50 2-GPUs node [1]. This is unrealistic for one to build and test an open-domain question answering system. Hence, this Thesis only uses a subset of context source to test the performance of dense retriever. Specifically, all contexts that contains answers for question in test set must be included in the new subset. Since there are a number of question in the test set that do not have a corresponding positive context in the context source, we need to add additional contexts to the new subset. These additional contexts

are manually annotated by searching Wikipedia for the correct answer of a question. Additional contexts guarantee that the maximum accuracy of the model is 100%, which is necessary for comparing different models. If the context source does not contain positive contexts for some question, we cannot know how well retriever perform on that question and thus, not fair for comparing models.

There are about 450 out of 3610 questions in the test set do not have corresponding positive context in the context source. This Thesis uses 700,000 out of 21 millions contexts as the new context source. Along with 450 annotated contexts, the new context source contains about 700,450 contexts.

The training data is left unchanged. Dense retriever was trained on full training data, which take several minutes on each epoch.

4.4 Experimental results on dense retriever

This section provides analysis on effect of different parameters to the performance of dense retriever. There are several default parameters used consistently throughout this section. Table 4.2 shows values of these parameters. The meaning of each parameter is as follows:

- `pretrained language model`: the pretrained language model used to initilize the question encoder and context encoder of dense retriever.
- `tokenizer`: the tokenizer used to tokenize question and contexts into tokens and the convert each token to a token id.
- `adam-eps, adam-betas`: hyperparameters for Adam optimizer. Refer to [26] to get details about this optimizer.
- `max context length`: max length of each context counting in single words. Each context is truncated and padded before being trained.
- `epochs`: number of times the whole training dataset is iterated through

Table 4.2: Default parameter settings

`pretrained language model`	`bert-base-uncased`
`tokenizer`	`bert-base-uncased`
`adam-eps`	`1e-8`
`adam-betas`	`(0.9, 0.999)`
`max context length`	`256`
`epochs`	`40`

Table 4.3: List of loss functions

`inbatch`	which is the original loss function, defined in equation (3.1)
`threeLevel`	try to distinguish between positive and hard negative contexts, between positive and normal negatives, between hard negatives and normal negatives, using a mix of binary cross entropy and negative log-likelihood.
`twolevel`	try to distinguish between positive and hard negative contexts, and between positive and normal negative contexts, using only negative log-likelihood.
`hardnegvsneg`	try to distinguish between positive and hard negative contexts, between hard negatives and normal negatives, using a mix of binary cross entropy and negative log-likelihood.
`hardnegvsnegsoftmax`	try to distinguish between positive and hard negatives, between hard negatives and normal negatives, using only negative log-likelihood.
`threelevelsoftmax`	try to distinguish between positive and hard negatives, between positive and normal negatives, between hard negatives and normal negatives, using only negative log-likelihood.

4.4.1 Effects of different loss functions

Before coming up with the proposed stratified loss, the author of this Thesis has done extensively experiments on a set of different loss functions to find out which is the best. This section provide a thorough analysis on effect of different loss functions.

First, let define several loss functions. These loss functions contains:

The proposed *stratified loss* mentioned in 3 In short, all these loss functions involve three types of context: positive, hard negative and normal negative. There is only one positive context but multiple hard negative and normal negative ones. Assume α is the similarity score between input question and positive context, $\{\beta_i\}_{i=1}^{x}$ are similarity scores between input question and x hard negative contexts and $\{\gamma_j\}_{j=1}^{y}$ are similarity scores between input question and y normal negative contexts. Followings are the loss formula defined between each two types of contexts:

- Loss defined on positive and hard negatives (softmax)

$$\mathscr{L}_{\text{posVsHard}} = -\log \frac{\exp(\alpha)}{\exp(\alpha) + \sum_{i=1}^{x} \beta_i} \tag{4.1}$$

- Loss defined on positive and normal negatives (softmax)

$$\mathscr{L}_{\text{posVsNeg}} = -\log \frac{\exp(\alpha)}{\exp(\alpha) + \sum_{i=1}^{x} \gamma_i} \tag{4.2}$$

- Loss defined on hard negatives and normal negatives
  - Binary cross entropy:

$$\begin{aligned}
&\tilde{\beta} = \text{softmax}(\beta) \\
&\tilde{\gamma} = \text{softmax}(\gamma) \\
&l = (1, 1, ..., 1, 0, 0, ..., 0, 0), \quad \text{where } x \text{ first values are 1 and } y \text{ remaining values are 0} \\
&c = l / \sum_{i=1}^{x+y} l \\
&\mathscr{L}_{\text{hardVsNeg}} = -\sum_{i=1}^{x} c_i \log\left(\tilde{\beta}_i\right) - \sum_{j=x+1}^{x+y} (1 - c_j) \log(1 - \tilde{\gamma}_{j-x}),
\end{aligned} \tag{4.3}$$

  - Softmax

$$\mathscr{L}_{\text{hardVsNeg}} = -\log \sum_{i=1}^{x} \frac{\exp(\beta_i)}{\exp(\beta_i) + \sum_{j=1}^{y} \exp\left(\gamma_j\right)} \tag{4.4}$$

Loss function used in (4.1), (4.2) and (4.4) are negative log-likelihood, which are the same as the loss functions presented in chapter **??**. In (4.3), binary cross entropy loss is used. Binary cross entropy between a scalar e and a scalar f, $e, f \in [0, 1]$ is defined as:

$$\text{binaryCrossentropy}(e, f) = -e \log(f) - (1 - e) \log(1 - f) \tag{4.5}$$

Assume e is a predefined constant, this function has minimum at $f = e$. According to that, the function in (4.3) has its minimum at $\tilde{\beta}_i = c_i, \forall 1 \leq i \leq x$ and $\tilde{\gamma}_j = c_{j+x}, \forall 1 \leq j \leq y$, meaning that hard negative contexts have higher similarity value than normal negative contexts. Both equation (4.3) and (4.4) aim to give higher similarity score to hard negative contexts than to normal negative contexts. However, computation of (4.4) is more expensive.

Using combinations of (4.1), (4.2), (4.3), (4.4) give us the different losses as shown in Table 4.3.

Comparison between `inbatch` and `threeLevel` loss functions

Performance of dense retriever using `inbatch` loss and `threeLevel` loss are shown in Figure 4.2. This comparison is done using following configurations:

- `batch_size=16`
- `num_hard_negatives=8`

- `max_grad_norm=2.0`

`num_hard_negative` here is the number of hard negative contexts in each training sample. This parameter does not apply for `inbatch` loss. For `inbatch` loss, we always use 1 hard negative context for training. `max_grad_norm` is used to clip gradient when computing backpropagation if the gradient value is to large. These three parameters are applied for all subsequent sections inside **??** if there is no explicit settings for these parameters.

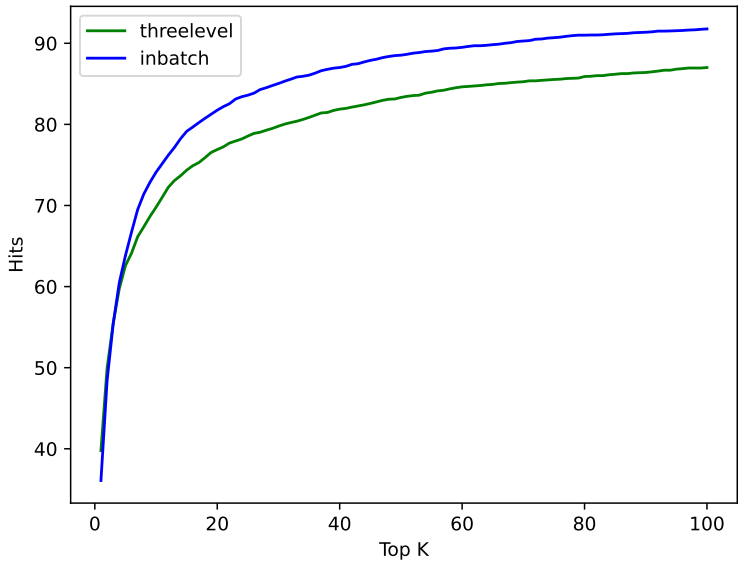


Figure 4.2: Top-K hits when using inbatch and threelevel loss

We can see that `threelevel` loss has higher top-k hits score than `inbatch` when k is small. For retriever, it is not too important to have high top-k hits score for small values of K because those contexts retrieved by retriever are further feed to re-ranker, which takes the job of finding what context having the answer. The important thing is retriever can retrieve at least one positive context. However, this is not true if we use retriever independently from reader, i.e. if we use retrieve as a search engine, the answer should be contained in a few top contexts retrieved by the retriever. No matter which is more important: overall top-k hits scores, or top-k hits scores for small K, we can clearly see that `inbatch` outperforms `threelevel` loss in this case.

Comparison between `inbatch` and `twolevel` loss functions

This experiments also has the same configuration as the previous: max question length 32, trained for 40epochs. Figure 4.2 shows the comparison.

Actually, there is no significant difference between these two loss. The difference is `inbatch` loss considers all contexts of other training samples in the same batch as the negative context, while `twolevel` considers only positive contexts of other training sam-

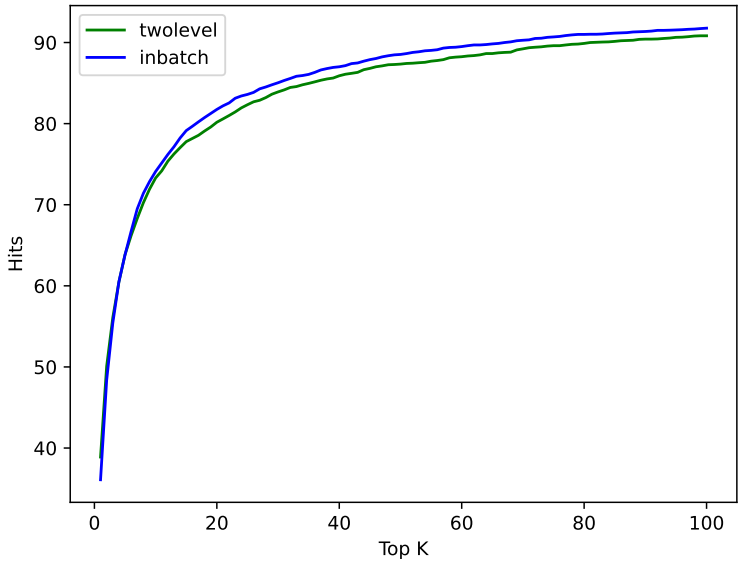


Figure 4.3: Top-K hits when using inbatch and twolevel loss

ples from the sample batch. When we train retriever on large data for many epochs, top-K hits corresponding to these two loss tend to be similar. In Figure 4.3, we see only a small gap between twolevel and inbatch curve, which is expected for these two loss functions.

Comparison between `inbatch` and `hardnegvsneg` loss functions

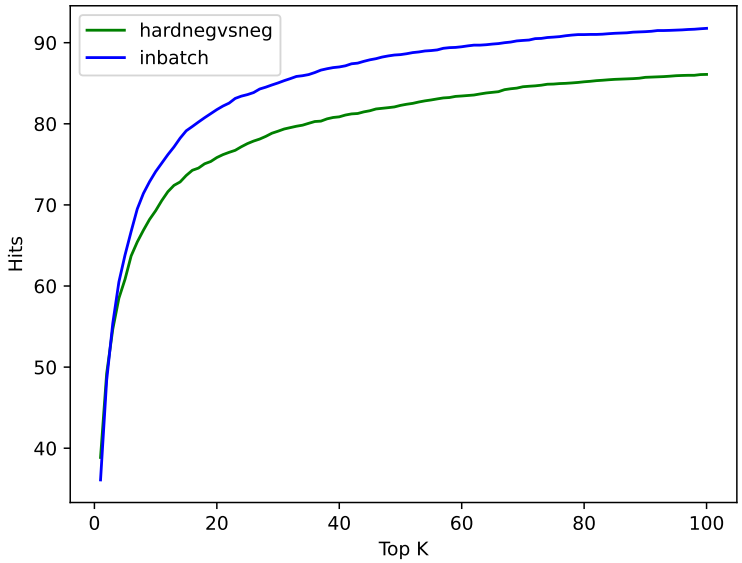


Figure 4.4: Top-K hits when using inbatch and hardnegvsneg loss

As shown in Figure 4.4, there is a big gap between top-K hits score between hardnegvsneg

and inbatch loss. This phenomena is the same as the first scenario considered in 4.4.1. At this point, we can see that using binary cross entropy loss is not efficient. This may be due to the essence of the function. In case of softmax, maximizing the probability of the positive also minimizing probability of all other negatives (because sum of the probability distribution is 1, and we are trying to assign probability 1 for the positive, whichs mean other negatives will receive the probability 0). In contrast, when using binary cross entropy, each probability is independently optimized. The fact that positive has high probability does not affect the probability of other negatives. We now remove `hardnegvsneg` and `threelevel` from consideration.

Comparison between `inbatch` and `threelevelsoftmax` loss functions

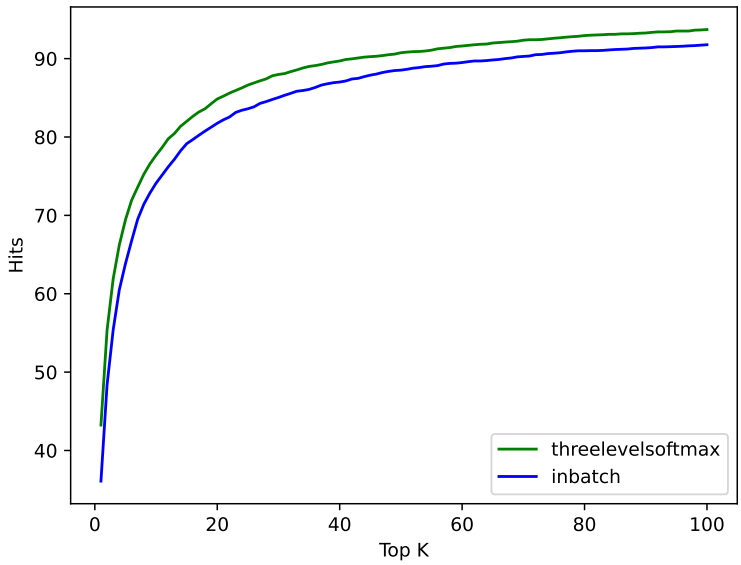


Figure 4.5: Top-K hits when using inbatch and threelevelsoftmax loss

Figure 4.5 shows the consistently better top-K hits scores of `threelevelsoftmax` over `inbatch` loss. The gap between two curves at small values of K is remarkable. Using `threelevelsoftmax`, we obtain both objective: high top-K hits score for small values of K and high top-K hits score over a wide range of K's values. Then, `threelevelsoftmax` is better than `inbatch` loss both in case used as search engine and in case used in combination with reader.

Comparison between `inbatch` and `hardnegvsnegsoftmax` loss functions

The last loss function is considered in this section. Figure 4.6 is the comparison between `hardnegvsnegsoftmax` and `inbatch`. This figure is almost the same as Figure 4.5, showing that using `hardnegvsnegsoftmax` and `threelevelsoftmax` give the same effect.

However, using `hardnegvsnegsoftmax` is cheaper since it only compute two types of loss.

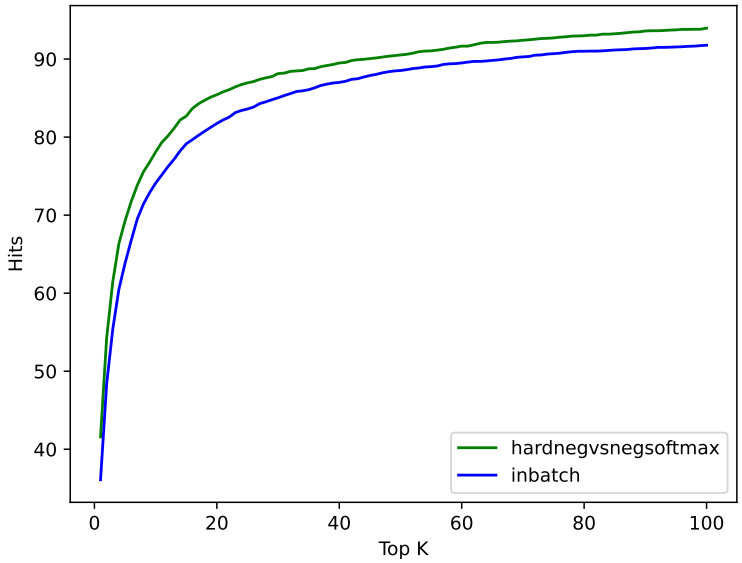


Figure 4.6: Top-K hits when using inbatch and threelevelsoftmax loss

Comparison with Dense Passage Retriever model[1]

This section compares Dense Passage Retriever (DPR) model with the best tuned model in this Thesis. Among current research on retriever, retriever model proposed by this work is one of the best model available. The author of Dense Passage Retriever shows that their dense retriever outperforms built-in search engine of Wikipedia. Figure 4.7 shows the comparison between this Thesis's best model and the baseline model proposed in [1]. Top-K hits score of the baseline model is produced using this Thesis's author code, which using Tensorflow and inference on Cloud TPUs. The original checkpoint is provided in PyTorch. The results produced by the Thesis's author code exactly match the results reported in github repository of the baseline model. (github repo only provides top-1, top-5, top-20 and top-100 hits). The checkpoint that is currently made publicly avaible is different from what is reported in original paper. There is an big improvements between the currently public model and the model reported originally. Table 4.4 show the improvements of new DPR model.

As shown in Figure 4.7, the baseline DPR model performs significantly better than the best tuned model when $K < 20$. When $K \geq 20$, the best tuned model starts to outperform the baseline model. As K becomes greater, the gap between two curves is getting more clear.

Karpukhin et. al. pointed in [1] that using 1 hard negative context significantly boost the model, but using more give no improvements or even worse. However, baseline DPR

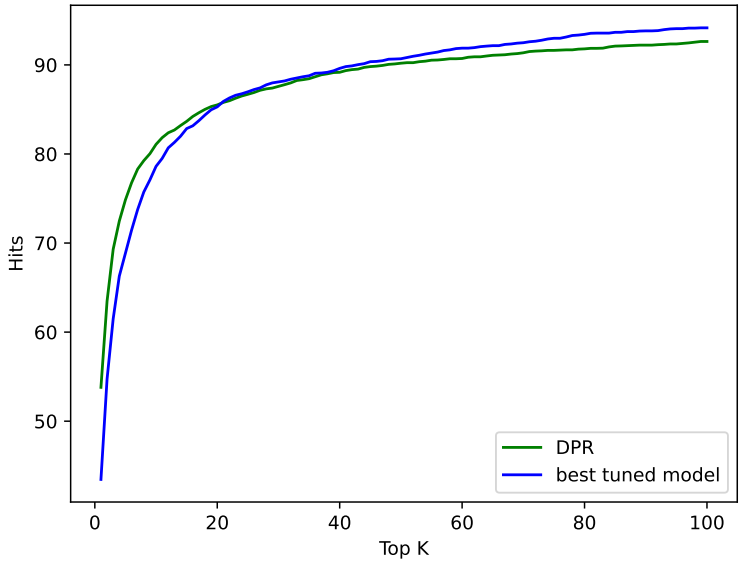


Figure 4.7: Top-K hits when using baseline DPR model and best tuned model used in this Thesis

Table 4.4: Top-K hits between original DPR model and new DPR model

Top-K contexts	Original DPR model	New DPR model
1	45.87	52.47
5	68.14	72.24
20	79.97	81.33
100	85.87	87.29

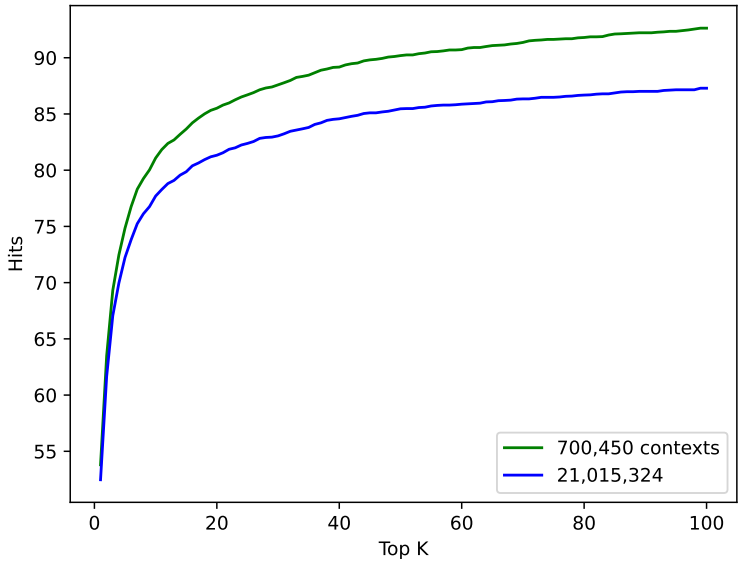


Figure 4.8: Difference in top-K hits score when using 700,045 contexts and 21,015,324 contexts

model uses `inbatch` loss, which considers hard negative and normal negative context the same. As shown in **??** and **??**, taking the difference between hard negative and normal negative context into consideration gives consistently better results.

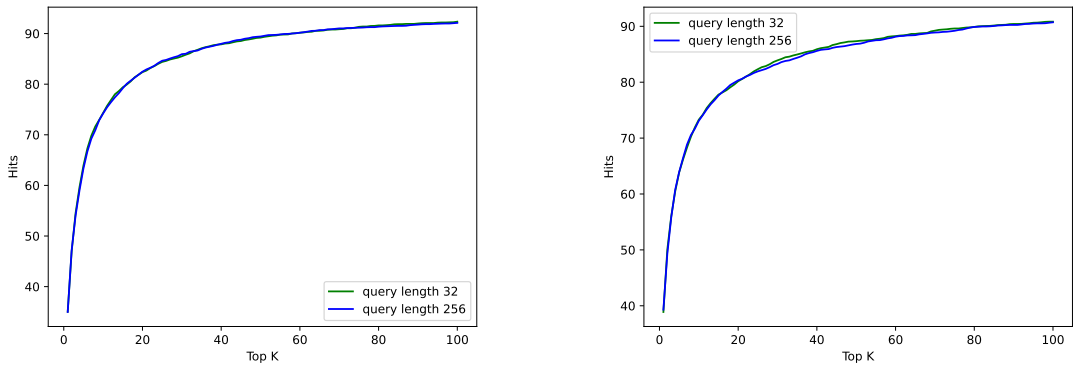
Note that this experiment is conducted on a subset of original context source (this experiments consider 700,450 contexts while the original context source contains 21,015,324 contexts). Then, the top-K hits score is slightly different. This difference is shown in Figure 4.8. From Figure 4.7, Table 4.4 and Figure 4.8, we can deduce that best tune model is competitive with the original DPR with small K values and is significantly better for high values of K. Note that the improvements introduced by new DPR model compared to original DPR model is related to hard negative context. New DPR model was trained with hard negative contexts mined from an existing DPR model.

After thoroughly analysis performance of dense retriever using different loss functions as well as study existing results, we can see that hard negative contexts are the key for success of an efficient open-domain question answering system. Such efficient loss function like *stratified loss* can nicely boost the retriever performance.

4.4.2 Effects of different question length

To efficiently train dense retriever on TPUs, each question is padding to the same length and each context is also padding to the same length. Since length of question is much smaller than length of context, we can pad question to a length that is smaller than the context padding length. In this experiment, two values of max question length are examined, which are 32 and 256. These values are power of 2 and such power of 2 is

beneficial for TPUs to efficiently compute. Figure 4.9 shows top-K hits scores when using the same configuration training retriever except for that max question length. As shown in this figure, question length 32 is competitive with question 256, sometimes even better. This implies that padding tokens to the input sequence does not affect the ability of the retriever model to learn. Hence, in the best tuned model, this Thesis uses the question length of 32 (note that this number is counted in tokens).



(a) (b)

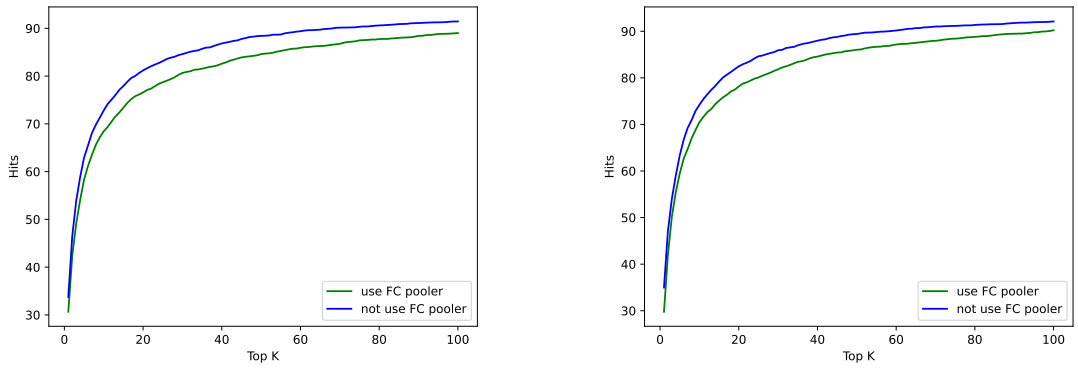
Figure 4.9: Comparison of top-K hits scores using different question length: (a) Using `inbatch` loss, `batch_size=64`; (b) Using `twolevel` loss, `batch_size=16`

4.4.3 Effects of using different poolers for getting the sequence embedding

Dense retriever, or BERT model in general, take as input a sequence of tokens, produces a sequence of token embeddings. To represent the whole sequence, we need to use a pooler operator over all tokens. There are several ways to do this: sum, average, max, .etc. Since BERT is a neural architecture that can capture direct interaction between every pair of tokens in the input sequence, we can use a "pooler" token to be the representation of the whole sentence. This is actually done very commonly. Before feeding the input sequence to BERT model, we append a special `[CLS]` token at the beginning of the sequence. After forwarding through BERT, the embedding vector of this `[CLS]` is used as embedding vector of the whole sequence.

To give the model more flexibility to learn, embedding vector of `[CLS]` token is further feed into a fully connected layer to produce the final embedding for the sequence. This type of pooler is actually implemented into some Deep Learning library as the default behavior. This Thesis also studies the effect of using fully connected layer on top of `[CLS]` token embedding. The results is shown in Figure 4.10

Both Figure 4.10a and Figure 4.10b shows that using Fully Connected layer tears down top-K hits scores significantly. This might be due to the fact that weights of Fully Connected layer do not interact directly with the token embedding vectors. Fully Connected



(a) (b)

Figure 4.10: Comparison of top-K hits scores when using fully connected pooler and when not use: (a) Using `inbatch` loss, `batch_size=16`; (b) Using `inbatch` loss, `batch_size=64`

layer only transfer information contained in `[CLS]` token to the final output. The job of transfering information might require large training steps to be successfully.

4.4.4 Choosing best tuned model

After conducting many experiments, there are some highlight findings:

- Using negative log-likelihood loss is better than binary cross entropy. Hence, `threelevelsoftmax` and `hardnegvsnegsoftmax` are chosen as the best loss functions. Since `hardnegvsneg` offers cheaper computation, this `hardnegvsnegsoftmax` is used for the best tuned model. `hardnegvsnegsoftmax` is essentially the *stratified loss* proposed in 3
- Using different question length does not affect performance of the model. Hence, the best tuned model uses the shortest question length possible. This length must be long enough to cover every question in the dataset. If too short, question will be truncated and thus lead to lost of information.
- Using fully connected layer on top of `[CLS]` token embedding downgrades the model performance significantly. Hence, best tuned model doest not use this fully connected layer

Beside these findings, a try to speed up training has also been done. Speed up is done by increase `learning_rate` and `max_grad_norm`. While `learning_rate` does not help, increase `max_grad_norm` results in a minor improvements.

To summary, Figure 4.11 shows a bigger picture of the performance of dense retriever with all types of loss functions.

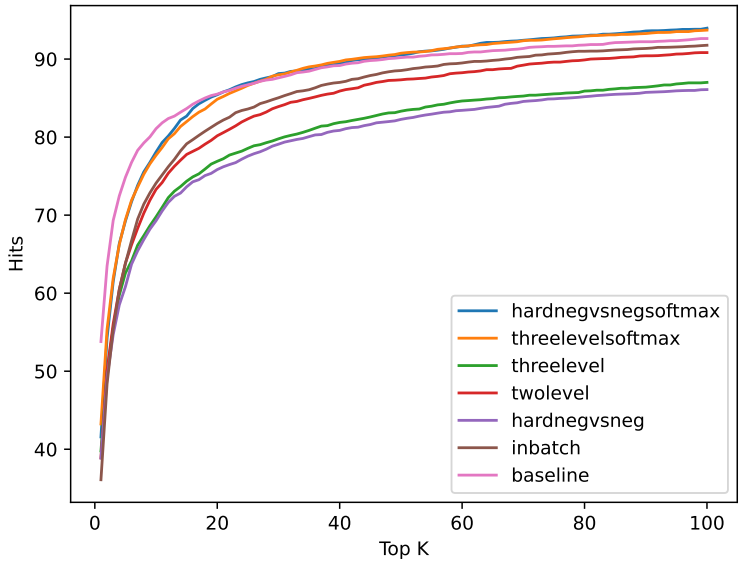


Figure 4.11: Comparison of top-K hits scores using different loss functions

4.5 Experimental results on extractive reader

Re-ranker and single-document reader have not been fully implemented in this Thesis. As an initial results on training single-document reader, this Thesis achieves exact match score of 56.6 on the Natural Question dataset (the preprocessed data set provided in [1])

4.6 Experimental results on Vietnamse COVID-19 dataset

This Thesis trained a dense retriever on COVID-19 dataset. Since the training samples is small (630 training sample), the model took only 1.4s on a TPU v3-8 to complete a train epoch. Dense retriever model for COVID-19 dataset was trained for 20 epochs with parameter settings be the same as used for training retriever on Natural Question dataset. The initial pretrained language model used is `NlpHUST/vibert4news-base-cased` from `https://huggingface.co/`.

Table 4.5 is the complete list of parameters used to train this model.

Results from the model are shown in 4.12 and Figure 4.13. As the results show, model retrieves very relevant contexts within first few retriever results.

Table 4.5: Parameters used for training dense retriever on Vietnamese COVID-19 dataset

batch_size	8
pretrained_language_model	NlpHUST/vibert4news-base-cased
learning_rate	2e-5
epochs	20
loss function	inbatch
max_context_length	256
max_query_length	64
max_grad_norm	3.0

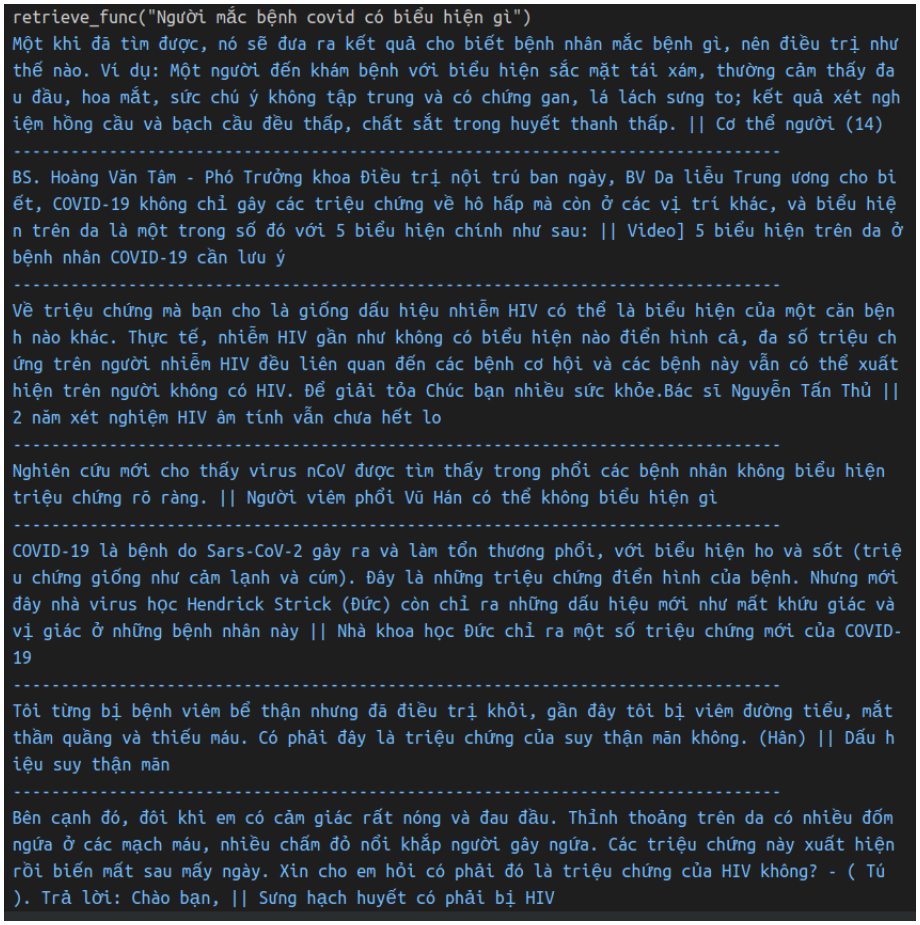


Figure 4.12: Retriever results of dense retriever trained on COVID-19 dataset

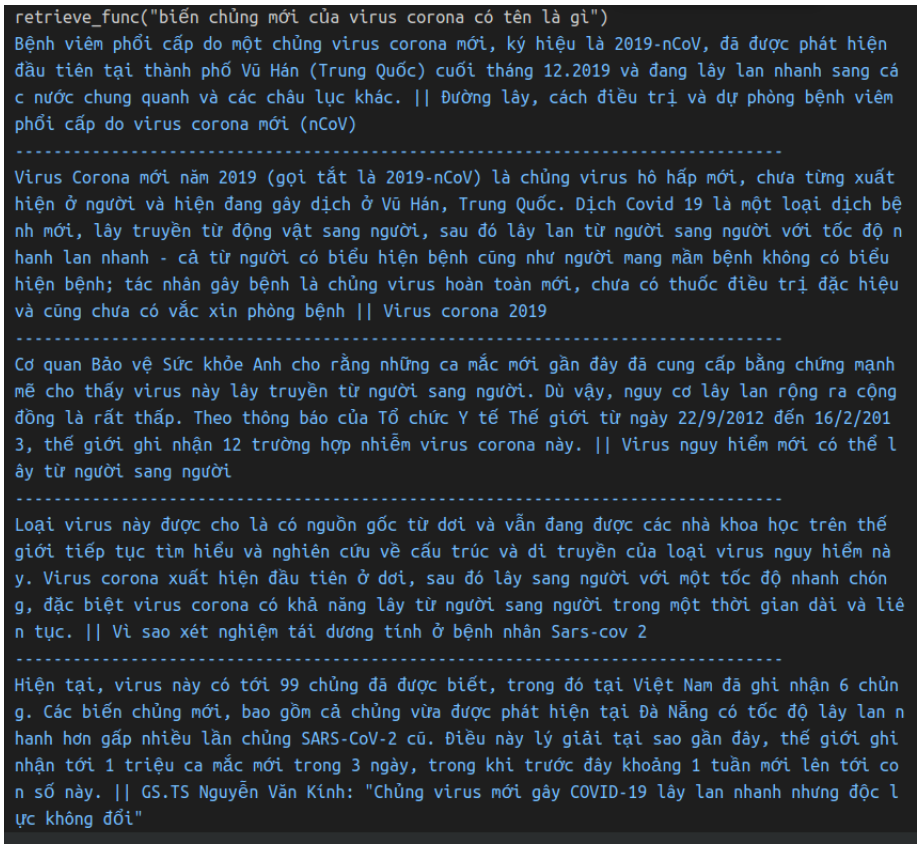


Figure 4.13: Retriever results of dense retriever trained on COVID-19 dataset

Chapter 5

Demo web interface

Chapter 6

Conclusion and future works

6.1 Conclusion

This Bachelor Thesis attempted to build an Vietnamese Open-domain question answering for COVID-19 topic. To successfully conduct an use case in Vietnamese language for COVID-19 data, the Thesis first studied on the basis of Open-domain question answerning system. After thoroughly diving into the architecture and detailed components of the system, this Thesis proposed to efficiently train the system using *stratified loss* function. By conducting numerous experiments, the proposed loss function is shown to consistently outperform existing loss function used in the literature. Most of the Thesis's effort was to effectively implement the system using Cloud TPUs. By successfully making use of Cloud TPUs, the Thesis can conduct large number of experiments in short time, reduced the time for experiments from impossible to several hours. Being able to rapidly produce experimental results allows the Thesis to come up with the proposed *stratified loss* function. After implementing and evaluating the open-domain question answering on bench mark dataset, this Thesis performed the job of building COVID-19 dataset. Data about COVID-19 topic was crawled from several medical articles and then was annotated using an easy web interface built with Django. A retriever model was built with this COVID-19 dataset and initial results of applying open-domain question answering system to COVID-19 dataset are quite positive.

6.2 Future Works

There are many works that are left to successfully build an qualified Vietnamese open-domain question answering system. To build an real application, we need a pipeline from data crawling, data management, data annotating, model training and evaluation. In the research aspect, re-ranker and reader model needs to be studied more in depth. Open-domain question has the potential to be applied to many domains and can tackle many business problems. For example, if the company wants to build a service that can automatically answer users with the question about the company itself, this is

where OpenQA comes in. Following the open-domain question answering system pipeline and collect a dataset about the company, open-domain question answering will be likely to perform well for the required task. Open-domain question answering also relates to several other problems such as: information retrieval, machine reading comprehension, knowledge base construction, etc. Relation between open-domain QA and knowledge base/knowledge graph is a really interesting problem that have not been yet studied widely.

Bibliography

[1] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. arXiv preprint arXiv:2004.04906, 2020.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. arXiv preprint arXiv:1706.03762, 2017.

[3] Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference, pages 219–224, 1961.

[4] Julian Kupiec. Murax: A robust linguistic approach for question answering using an on-line encyclopedia. In Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval, pages 181–190, 1993.

[5] Ellen M Voorhees et al. The trec-8 question answering track report. In Trec, volume 99, pages 77–82. Citeseer, 1999.

[6] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051, 2017.

[7] Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerald Tesauro, Bowen Zhou, and Jing Jiang. Rθ3: Reinforced reader-ranker for open-domain question answering. arXiv preprint arXiv:1709.00023, 2017.

[8] Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. Learning to retrieve reasoning paths over wikipedia graph for question answering. arXiv preprint arXiv:1911.10470, 2019.

[9] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. arXiv preprint arXiv:1906.00300, 2019.

[10] Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. A discrete hard em approach for weakly supervised question answering. arXiv preprint arXiv:1909.04849, 2019.

[11] Sewon Min, Danqi Chen, Luke Zettlemoyer, and Hannaneh Hajishirzi. Knowledge guided text retrieval and reading for open domain question answering. arXiv preprint arXiv:1911.03868, 2019.

[12] Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. Real-time open-domain question answering with dense-sparse phrase index. arXiv preprint arXiv:1906.05807, 2019.

[13] Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. Multi-passage bert: A globally normalized bert model for open-domain question answering. arXiv preprint arXiv:1908.08167, 2019.

[14] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. arXiv preprint arXiv:2002.08909, 2020.

[15] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. arXiv preprint arXiv:2007.01282, 2020.

[16] Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. arXiv preprint arXiv:2005.11401, 2020.

[17] Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. Generation-augmented retrieval for open-domain question answering. arXiv preprint arXiv:2009.08553, 2020.

[18] Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. Ambigqa: Answering ambiguous open-domain questions. arXiv preprint arXiv:2004.10645, 2020.

[19] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? arXiv preprint arXiv:2002.08910, 2020.

[20] Wenhan Xiong, Hong Wang, and William Yang Wang. Progressively pre-trained dense corpus index for open-domain question answering. arXiv preprint arXiv:2005.00038, 2020.

[21] Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. Reader-guided passage reranking for open-domain question answering. arXiv preprint arXiv:2101.00294, 2021.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[23] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. arXiv preprint arXiv:1702.08734, 2017.

[24] Dat Quoc Nguyen and Anh Tuan Nguyen. Phobert: Pre-trained language models for vietnamese. arXiv preprint arXiv:2003.00744, 2020.

[25] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. Transactions of the Association for Computational Linguistics, 7:453–466, 2019.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.