DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2020

Automatic Question Generation with Pre-trained Masked Language Models

CHUN HUNG LIN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Automatic Question Generation with Pre-trained Masked Language Models

CHUN HUNG LIN

Master in Machine Learning
Date: December 20, 2020
Supervisor: Johan Boye
Examiner: Joakim Gustafsson
School of Electrical Engineering and Computer Science
Swedish title: Automatisk frågegenerering med förtränade maskerade språkmodeller

Abstract

In this project, we study the task of generating a question from a given passage-answer pair using pre-trained masked language models. Asking questions is of importance in artificial intelligence development because it makes a machine look intelligent when it raises a reasonable and well-constructed question. Also, question generation has its applications such as drafting questions for a reading comprehension test and augmenting data for expanding the training set of a question answering task.

We focus on using pre-trained masked language models throughout this project. Masked language modeling is relatively new in question generation, but it has been being explored in the machine translation domain. In our experiments, we used two training techniques and two types of generation orderings. We are the first to adopt one of these training techniques for the question generation task. In our evaluation, n-gram based precision-recall evaluation and a human evaluation were conducted for comparing and analyzing.

The experiment results showed that the best of our methods was as good as LSTM-based methods by comparing the results with the previous research literature. Moreover, all combinations of the training techniques and the generation orderings are acceptable according to our human evaluation results. We also demonstrated that one of our techniques enables us to control how long the generated question would be.

Sammanfattning

I detta projekt studerar vi uppgiften att generera en fråga från ett givet par av ett textstycke och ett svar med förtränade maskerade språkmodeller. Att ställa frågor är viktigt i utvecklingen av artificiell intelligens eftersom det får en maskin att se intelligent ut när den ställer en rimlig och välkonstruerad fråga. Frågegenerering har också sina applikationer som att formulera frågor för ett läsförståelsetest och att utöka datamängder som kan användas för att träna frågebesvarande program.

Vi fokuserar på att använda förtränade maskerade språkmodeller under hela detta projekt. Maskerade språkmodeller är relativt nya i samband med frågegenerering men det har undersökts i maskinöversättningsdomänen. I våra experiment använde vi två träningstekniker och två typer av genereringsordningar. Vi är de första att anta en av dessa träningstekniker för frågegenerering. För utvärdering använde vi n-grambaserad precision-täckning. Vi gjorde även en utvärdering med försökspersoner.

Experimentresultaten visade att den bästa metoden var lika bra som LSTM-baserade metoder genom att jämföra resultaten med den tidigare forskningslitteraturen. Dessutom är alla kombinationer av träningsteknikerna och genereringsordningarna acceptabla enligt våra mänskliga utvärderingsresultat. Vi visade också att den nyligen föreslagna tekniken gör det möjligt för oss att kontrollera hur lång den genererade frågan skulle vara.

Acknowledgments

First of all, I would like to thank my supervisor, Professor Johan Boye, for his constructive discussions on my work and valuable suggestions on my thesis writing. His ideas and suggestions broaden my horizons in the field of natural language generation.

Secondly, I am so grateful to my family for supporting my master degree. I also appreciate KTH offered me one-year scholarship as it greatly relieves my burden of my study.

Lastly, I would like to say thank you to Ying-Hong Chan and Yao-Chung Fan for providing me their source code as references. Although this work does not use any source code from them, yet their information helped me a lot for understanding their work.

Contents

1 Introduction **1**
- 1.1 Problem Definition . . . 1
- 1.2 Motivation . . . 3
  - 1.2.1 Applications of question generation . . . 3
  - 1.2.2 Pre-training in Deep Language Processing . . . 4
  - 1.2.3 Masked Language Model . . . 4
- 1.3 Research Questions . . . 5
- 1.4 Limitation . . . 5
- 1.5 Contributions . . . 5

2 Background **7**
- 2.1 Tokenization . . . 7
- 2.2 Conditional Language Model . . . 8
  - 2.2.1 Autoregressive Language Model . . . 9
  - 2.2.2 Masked Language Model . . . 9
- 2.3 Sequence to Sequence learning . . . 10
- 2.4 Transfer Learning . . . 11
- 2.5 Transformer Model . . . 12
  - 2.5.1 Attention . . . 12
  - 2.5.2 Scaled Dot Product Attention . . . 14
  - 2.5.3 Multi-Head Attention . . . 15
  - 2.5.4 Encoder Architecture . . . 17
- 2.6 BERT . . . 20
  - 2.6.1 Architecture . . . 21
  - 2.6.2 Unsupervised Pre-training for BERT . . . 23

3 Related Work **25**
- 3.1 Rule-based question generation . . . 25
- 3.2 Template-based question generation . . . 26

3.3 Neural-network-based question generation 28
3.4 Pre-training for text generation 32

4 Methods **34**
4.1 HLSQG 35
4.1.1 Generation 35
4.1.2 Training 36
4.2 u-PMLM 39
4.2.1 Generation 39
4.2.2 Training 42
4.2.3 Length Prediction 43
4.3 Decoding 44
4.3.1 Beam-search 44
4.3.2 N-gram repetition banning 45
4.4 Post-processing 46

5 Experiment **47**
5.1 Datasets 47
5.1.1 Data cleansing 48
5.1.2 Answer selection in SQuAD Dev set 51
5.1.3 Data splits 51
5.2 Implementation Details 53
5.2.1 Models 53
5.2.2 Data Processing 54
5.2.3 Training 55
5.2.4 Question Generation 57
5.3 Evaluation Methods 57
5.3.1 Automatic Evaluation 57
5.3.2 Human Evaluation 59

6 Results **60**
6.1 Length Prediction Model Estimation 60
6.2 Training Performance 61
6.3 Automatic Evaluation 67
6.4 Human Evaluation 70
6.5 Case Study: Question Generation about KTH 72
6.6 Comparison with Other Works 73

7 Discussion **76**
7.1 Pre-training 76

7.2 HLSQG 77
7.3 u-PMLM 77
7.4 Model size 78
7.5 Research Questions 79
7.6 Societal Impacts 80
7.7 Ethics and sustainability 80
7.8 Future work 80

8 Conclusions **82**

Bibliography **83**

A VM specification **90**

B BLEU scores **91**

Chapter 1

Introduction

Natural language processing (NLP) is a study concerning the process of human languages by computer software and algorithms. In our daily life, there are numerous natural language processing applications such as a conversational dialogue system, automatic speech recognition, and machine translation.

Natural language generation is a subfield of natural language processing, which is using a computer to convert data, including image, text, structured records, and unstructured information, into natural language. Natural language generation also includes generating text from scratch. Text summarization, machine translation, story writing, and question generation are examples of natural language generation.

One can consider natural language generation as the opposition of natural language understanding, a subject to make computers understand and analyze human languages.

1.1 Problem Definition

In this project, we frame the question generation task as: ”producing a question, given a given paragraph passage and the target answer with its location in the passage as the input”.

In fig. 1.1, we show an example of the question generation task. For generating question 1, the passage, the answer (i.e. the set of life-sustaining chemical reactions), and the character position (i.e. 14, zero-based indexing) are provided for creating a question.

As a passage may have the same answer text in multiple places, we give the answer location to specify which answer text we focus. In the example, catabolic appears twice, and we want to generate a question looking at the first "catabolic".

Passage:
Metabolism is the set of life-sustaining chemical reactions in organisms. Metabolic reactions may be categorized as catabolic, the breaking down of compounds, or anabolic, the building up of compounds. Usually, catabolism releases energy, and anabolism consumes energy. Catabolic reactions generate ATP, and anabolic reactions consume it. It also serves as a carrier of phosphate groups in phosphorylation reactions.

Answer 1:
the set of life-sustaining chemical reactions

Generated question 1:
What is the definition of metabolism?

Answer 2:
catabolic

Generated question 2:
What kind of metabolic reaction refers to the breaking down of compounds?

Figure 1.1: An example of the question generation task. The passage in this example was extracted from the Wikipedia article: **Metabolism** [1].

1.2 Motivation

In this section, we explain the reason why we would like to have a pre-trained masked language model to generate questions.

1.2.1 Applications of question generation

Why do we want to generate questions with input information? The most obvious answer is for language education, especially the reading comprehension task. Imagine that an English teacher would like to test whether a student understands an English paragraph at a certain level. The most traditional way is to give a reading comprehension task to the student. The teacher needs to draft the questions regarding the reading passage and the target answer. If there is software which can write sensible questions given the passage and answer, it helps a lot in reducing the workload of the teacher.

Secondly, question generation can be also applied to test knowledge learned by a student in a course. Similar to language learning, a course responsible wants to have an effective way to inspect how much a course participant has learned the content presented in the course. Asking the course participant to have a quiz is the most intuitive way to check one's knowledge towards the given materials. Then, a computer program for question generation is helpful for quiz preparation. Moreover, answering questions can improve one's understanding of the course content. Looking for answers to the questions forces students to review and give thought to the related material. Then the students shall have a deeper understanding of the specific part of the course. [2]

Thirdly, the question generation can be regarded as a technique for data augmentation. In machine learning, data augmentation means to modify the existing data in order to increase the amount of training data. When training a deep learning model for question answering, we would like to have tuples of a passage, a question, and an answer, as many as possible. If there is a system that can create relevant and reasonable questions, it definitely can be a module for data augmentation of question-answer datasets.

1.2.2 Pre-training in Deep Language Processing

In 2018, GPT-2 and BERT were released and achieved great success in natural language understanding tasks. Their approaches are two-stage training procedures: pre-training and fine-tuning. GPT-2 and BERT used large unlabeled text corpora for pre-training in order to learn a general language representation and then transfer the learned knowledge to the supervised downstream tasks. [3, 4] The downstream tasks in Radford et al. and Devlin et al. works are natural language understanding tasks like text classification and question answering.

Although GPT-2 and BERT are designed for language understanding task, Song et al. [5] and Lewis et al. [6] tried the two-stage approach for text generation task.

Given the promising results of language understanding tasks using pre-trained models, we would like to consider question generation as a fine-tuning task using a pre-trained network. Since pre-training allows models to learn language representations, we would expect that a downstream task with pre-training outperforms one without pre-training.

1.2.3 Masked Language Model

A masked language model emulates a fill-in-the-blank task. The objective of the model is to predict a mask token using the surrounding words. And it is very similar to the cloze task in our language exercise.

One of the advantages of a masked language model is that it can predict tokens in parallel for speeding up the generation yet the quality of the token predictions is worse than predicting tokens one-by-one. It is because the iterative generation limits the next possible token according to generated tokens.

1.3 Research Questions

The above motivations lead us to the research questions of this project:

1. *How well does sequence-to-sequence learning work for question generation?*
2. *Could we fine-tune a network pre-trained with a language modeling objective for question generation?*
3. *How does the pre-trained network size affect the quality of generated questions?*

1.4 Limitation

Due to limited computational resources, we used the pre-trained models available on the Internet for this project. This limitation made us consider BERT for our project because BERT has multi-size pre-trained models.

Moreover, we could not test all possible training hyper-parameters. It is because the training time typically took from 4 hours to 1 day. Therefore, we chose the hyper-parameters employed by other researchers in our experiments.

1.5 Contributions

In this project, we have done the following items:

- Re-implemented one of the methods proposed Chan and Fan [7]
- Inspired by Liao, Jiang, and Liu [8], we came up with another fine-tuning method to use BERT as a question generation model.

- Trained different sizes of pre-trained BERT models using these two approaches.
- Evaluated and analyzed the generated questions.
- Open-sourced this project source code[1].

[1] https://github.com/chris4540/StudyMaskedLMForQG

Chapter 2

Background

This chapter introduces the various concepts used in this project. Regarding natural language processing, we talk about tokenization and language models. Neural networks, deep learning, and transfer learning are also included in this chapter for getting us familiar with the approaches used in this project. Finally, the recently developed models like transformers and BERT are presented.

2.1 Tokenization

In natural language processing, the first step to process and analyze a piece of text is tokenization. Tokenization splits a piece of text, or a sequence of characters, into **tokens** for further processing. A Token is not necessary to be a word but it is a basic unit for language modeling and generation tasks. The simplest way for tokenization is splitting on white-spaces.

Take a Swedish sentence as an example. ”Det här är en gul fågel.” shall be tokenized into 7 tokens: ”Det”, ”här”, ”är”, ”en”, ”gul”, ”fågel”, and ”.”.

Some tokenizers (e.g. WordPiece) separate text into subword level tokens. In order words, a word can be chopped into smaller pieces. For

instance, the Swedish word ”daglig” could be split into ”dag” and ”-lig” as the second token forms adjectives in Swedish like ”-able” and ”-ly” in English.

2.2 Conditional Language Model

Given an input token sequence $\mathbf{x} = \{x_i\}_{i=1}^{|\mathbf{x}|}$ of length $|\mathbf{x}|$, we formulate the probability parameterized by $\boldsymbol{\theta}$ of the output token sequence $\mathbf{y} = \{y_i\}_{i=1}^{|\mathbf{y}|}$ of length $|\mathbf{y}|$ as $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$.

When we have a set of input-output pairs $(\mathcal{X}, \mathcal{Y})$, we can estimate the parameters $\boldsymbol{\theta}$ by the maximum likelihood method:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{(\mathbf{x},\mathbf{y})\in(\mathcal{X},\mathcal{Y})} p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) \tag{2.1}$$

There are three kinds of language generation tasks using this framework. They are machine translation, abstractive text summarization, and question generation.

Machine translation has $\mathbf{x}$ as a source language sentence and $\mathbf{y}$ as a target language sentence. In abstractive text summarization, $\mathbf{x}$ is a long piece of text and $\mathbf{y}$ is a generated summary of the text. In our task, we have $\mathbf{x}$ as the context or passage with an answer inside, and $\mathbf{y}$ is the question closely related to the context and the answer.

By considering the similarity of these tasks, it is not surprising that the many of the same methods can be applied to solve these tasks. However, one of the differences among the tasks is that the lengths of $\mathbf{x}$ and $\mathbf{y}$ are approximately the same in machine translation but not in text summarization and question generation.

In the following parts, we briefly go through the autoregressive language model and masked language model conditioning on a given text $\mathbf{x}$ using the notion from Yang et al. [9] and Lample and Conneau [10].

2.2.1 Autoregressive Language Model

We can factorize the conditional probability function $p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ over $\mathbf{y}$ in multiple ways by the probability chain rules. Generally, we factorize the function from left to right.

$$p(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}) = \prod_{i=1}^{|\mathbf{x}|} p(y_i|y_1, y_2, y_3, ..., y_{i-1}, \mathbf{x};\boldsymbol{\theta}) \tag{2.2}$$

$$= \prod_{i=1}^{|\mathbf{x}|} p(y_i|\mathbf{y}_{<i}, \mathbf{x};\boldsymbol{\theta}) \tag{2.3}$$

, where $\mathbf{y}_{<i} = \{y_1, y_2, y_3, ..., y_{i-1}\}$.

The term autoregressive means the token y_i only depends on the previous tokens $\mathbf{y}_{<i}$ and it is helpful when generating a sequence. Once we have the model, we can generate a sequence of tokens by maximizing each factor $p(y_i|\mathbf{y}_{<i}, \mathbf{x};\boldsymbol{\theta})$ in the chained probabilities or sample tokens from that factor successively from left to right.

This model is called the *autoregressive* language model. It has other names like *left-to-right* language model [4, 6] and *causal* language model [10]. However, we refer the autoregressive language model to the formulation in eq. (2.3) throughout this project.

2.2.2 Masked Language Model

Masked language model aims to reconstruct mask tokens (i.e. `[MASK]`) from other surrounding tokens so that the reconstructed sequence of tokens sounds coherent and plausible. [11] The model denoises the mask tokens and reconstructs a token to replace the corresponding mask token. And therefore the masked language model is also called *denoising autoencoding* language model. [9, 12] The denoising and reconstructing approach is similar to a fill-in-the-blank exercise or a text infilling task. [13]

Following the formulation from Liao, Jiang, and Liu [8], we have the probability of mask tokens conditional on observable tokens and the

input tokens:

$$p(\mathbf{y}_{\Pi}|\mathbf{y}_{\backslash\Pi}, \mathbf{x}; \boldsymbol{\theta}) = \prod_{i=1}^{m} p(y_{\pi_i}|\mathbf{y}_{\backslash\Pi}, \mathbf{x}; \boldsymbol{\theta}) \tag{2.4}$$

, where $\Pi = \{\pi_1, \pi_2, ..., \pi_m\}$ is a set of mask token positions, $\backslash\Pi$ is the unmasked token positions, and m is the number of masks.

This definition assumes that all mask tokens are reconstructed independently. [9]

The approach gives multiple ways to sample tokens from the distribution. For example, the model allows generating tokens individually in an arbitrary order or from left to right. Also, we can generate N tokens in one shot and then iteratively modify the sequence by re-masking unlikely tokens. [14]

However, using the masked language model to generate a sequence of tokens in arbitrary order, we need to decide the length of the sequence before generating it. [8, 15]

In the case of no input text, the language model becomes unconditional. The unconditional masked language model is a self-supervised framework for natural language understanding using a fixed masking ratio in a few papers. [4, 6, 16] However, Liao, Jiang, and Liu [8] proved that if we have a uniform prior probabilistic masking scheme during training, the model itself is equivalent to an autoregressive permutated language model, which means the generation order is permutated and not sequential.

2.3 Sequence to Sequence learning

Sequence to sequence learning is an end-to-end learning task to have a model convert a sequence to another sequence. This method was proposed by Sutskever, Vinyals, and Le [17], and they used the encoder-decoder LSTM model for the English to French translation task.

The typical sequence-to-sequence learning task uses the encoder-decoder architecture, which maps the input to a fixed size representation vec-

tor and then converts the vector to the target domain [18]. However, sequence-to-sequence learning is not necessary to employ the encoder-decoder architecture.

2.4 Transfer Learning

Transfer learning is a method to reuse a model that is originally for a *base* task and apply that model to another *target* task. [19] Typically, a *base* network is trained with a large and general dataset. And then we transfer the learned weights from the *base* network to our *target* network, change the training objective, and train the target network with a smaller dataset [19].

For example, a deep network is trained over ImageNet [20] for a classification task. After that, we propose another network for classifying medical images with a limited amount of data. That model shall be initialized with the first n layers from the ImageNet network and randomly initialize the remaining layers.

Generally, using the transferred weights on the target model is better than random initialization and improves the *target* model generalization. Also, the closer the objectives of two tasks, the better the transferability of the learned features from the *base* model to the *target* model.

We call training the model on the *base* task as ***pre-training*** and training the model using the learned features on the *target* task as ***fine-tuning***. [19]

Self-supervised learning

Noroozi and Favaro [21] proposed the visual Jigsaw puzzles solving task using unlabeled data as the *base* task. And then they transferred the learned features for classification, detection, and segmentation tasks [21].

The puzzle-solving task is unsupervised as we do not provide labels. Yet their Context-Free Network [21] was trained to predict the puzzle

arrangement and the arrangement comes from the data itself. Therefore, we call this task a self-supervised learning task.

2.5 Transformer Model

In this section, basic knowledge of the neural networks and the deep architectures network is required. For those who are not familiar with neural networks and deep architectures in the neural networks, the book written by [22] gives a concise introduction to neural networks in the chapter 6 of the book.

As this project using only the encoder part of the Transformer, the decoder part is then skipped.

There are two excellent guides called "The Annotated Transformer" [1] and "The Illustrated Transformer"[2]. Readers are suggested to read these guides together with this section.

2.5.1 Attention

The attention mechanism was proposed by Bahdanau, Cho, and Bengio [23] in 2015. In neural machine translation, we generally use the encoder-decoder approach [17] to first convert a source language token sequence to a fixed-length vector and then convert the fixed-length vector to the target language token sequence. However, this approach is not friendly to long sentence translations as it is not quite possible to encode a great amount of information into a fixed-length vector.

Then, Bahdanau, Cho, and Bengio [23] designed an auxiliary network to learn both the translation and alignment task jointly. The module representing the alignment was called the attention mechanism. Instead of using only the last hidden state from the recurrent encoder network when decoding, they used all hidden states of the source words

[1] http://nlp.seas.harvard.edu/2018/04/03/attention.html
[2] http://jalammar.github.io/illustrated-transformer/

in the encoder network and the previous hidden state of the recurrent decoder network to predict the next target word.

The idea of attention mechanism can be traced back to bitext word alignment, a task to identify the relationships among source and target words. In bitext word alignment, we want to find a bipartite graph to explicitly show the relationships as edges between the source and target words as vertices. In this attention mechanism, we find the alignment density, annotated as α in [23], between source hidden states and the decoder hidden state on hand when decoding.

The attention mechanism from Bahdanau, Cho, and Bengio can be summarized as: [24]

$$\alpha_{ij} = \text{softmax}(\text{score}(\boldsymbol{s}_{i-1}, \boldsymbol{h}_j)) \tag{2.5}$$

$$\text{score}(\boldsymbol{s}_i, \boldsymbol{h}_j) = \boldsymbol{v}_a^\top \tanh(\boldsymbol{W}_a[\boldsymbol{s}_i; \boldsymbol{h}_j]) \tag{2.6}$$

$$\boldsymbol{c}_i = \sum_{j=1}^{T_x} \alpha_{ij} \boldsymbol{h}_j \tag{2.7}$$

, where

i	is the decoding step to decode the i-th word,
j	is the index for j-th word in the source sequence,
T_x	is the length of the source sequence,
$\boldsymbol{h}_j$	is the j-th hidden state in the encoder,
$\boldsymbol{s}_i$	is the i-th hidden state in decoder,
$\boldsymbol{c}_i$	is the context vector for decoding i-th word,
$\boldsymbol{W}_a$, $\boldsymbol{v}_a$	are the weights and bias in the one layer auxiliary network,
$[.;.]$	is the vector concatenation operator.

Note that in eq. (2.5), the softmax function in the denominator sums over the index j.

2.5.2 Scaled Dot Product Attention

In another point of view, we can consider the attention mechanism as a function of a query vector, a key vector, and a value vector and the output is the weighted average of value vector. [25] Take the attention mechanism in the previous section as an example. We can regard $\boldsymbol{s}_{i-1}$ as a query vector and $\boldsymbol{h}_j$ as a key and value vector, and the output averaged vector is the context vector $\boldsymbol{c}_i$.

Instead of using a feed-forward neural network, we can simplify the score function in eq. (2.6) as a dot-product.

However, the dot-product will scale-up as the vectors are not unit vectors. Also, it is not desirable to have a vector normalization in the network as it is an "expensive" operation involving a lot of calculation operations. Therefore, we estimate the scaling factor as a constant by considering all elements from both vectors are normal distributed with mean 0 and variance 1. Then, the dot-product p is also a random variable.

$$p = \boldsymbol{q}\boldsymbol{k}^\top = \sum_{i=1}^{d} q_i k_i \tag{2.8}$$

$$\mathbb{E}[p] = \sum_{i=1}^{d} \mathbb{E}[q_i]\,\mathbb{E}[k_i] = 0 \tag{2.9}$$

$$\mathbb{E}[p^2] = \mathbb{E}\left[\sum_{i=1}^{d} q_i k_i \sum_{j=1}^{d} q_j k_j\right] \tag{2.10}$$

$$= \mathbb{E}\left[\sum_{i=1}^{d}\sum_{j=1}^{d} q_i q_j k_i k_j\right] \tag{2.11}$$

$$= \sum_{i=j}^{d} \mathbb{E}[q_i q_j]\,\mathbb{E}[k_i k_j] \tag{2.12}$$

$$= \sum_{i=j}^{d} 1 \tag{2.13}$$

$$= d \tag{2.14}$$

, where $\boldsymbol{q}$ and $\boldsymbol{k}$ are d-dimensional row vectors.

In eq. (2.11), four random variables are independent and therefore we count only when $i = j$.

Therefore, the variance of p is $\mathbb{E}[p^2] - \mathbb{E}[p]^2 = d$, and we pick the standard deviation $\sqrt{d}$ as the approximated norm of p.

As a result, we have a scoring function for the scaled dot product attention in vector form:

$$\text{score}(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}\boldsymbol{k}^\top}{\sqrt{d}} \tag{2.15}$$

Let's consider we have a sequence of length L and each token is projected to a d-dimensional vector. We can group L d-dimensional vectors into a single matrix.

Therefore, we can rewrite eq. (2.15) into matrix form:

$$\text{score}(\mathbf{Q}, \mathbf{K}) = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} \tag{2.16}$$

, where $\mathbf{Q}$ is a query matrix and $\mathbf{K}$ is a key matrix of size $L \times d$.

And the attention is:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V} \tag{2.17}$$

Remember that $\mathbf{Q}\mathbf{K}^\top$ is a $L \times L$ matrix, and we take summation over the second dimension. Let's say each entry of the softmax term is a_{ij}:

$$a_{ij} = \left[\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\right]_{ij} \tag{2.18}$$

$$= \frac{1}{\sqrt{d}} \frac{\exp(\sum_k q_{ik} k_{jk})}{\sum_l \exp(\sum_k q_{ik} k_{lk})} \tag{2.19}$$

2.5.3 Multi-Head Attention

As the scaled dot product attention does not have any trainable parameters, we then introduce learnable weights. Vaswani et al. suggested

having learnable linear projections for the query, key, and value vectors h times. [25] The attention output vectors are concatenated and then mapped to a certain dimension. We can parallelly calculate the attentions h times and h is the number of heads in this attention mechanism.

Why does the multi-head mechanism attend the query, key, and value multiple times? It is because the mechanism allows learning different representations in different subspaces formed by those learnable linear projections. [25] In another point of view, it creates small ensembles of subspaces that keep tracking different representations from the incoming query, key, and value vectors.

We can write down equations for the Multi-Head Attention:

$$\text{MultiHeadAttn}(\tilde{\mathbf{Q}}, \tilde{\mathbf{K}}, \tilde{\mathbf{V}}) = [\boldsymbol{\phi}_1; \boldsymbol{\phi}_2; \boldsymbol{\phi}_3; ...; \boldsymbol{\phi}_h]\mathbf{W}^O \tag{2.20}$$

$$\boldsymbol{\phi}_i = \text{Attention}(\tilde{\mathbf{Q}}\mathbf{W}_i^Q, \tilde{\mathbf{K}}\mathbf{W}_i^K, \tilde{\mathbf{V}}\mathbf{W}_i^V) \tag{2.21}$$

, where

i	is the head index from 1 to h,
h	is the number of attention heads,
L	is the number of tokens in the input sequence,
$\boldsymbol{\phi}_i \in \mathbb{R}^{L \times d}$	is the i-th head attention,
$\tilde{\mathbf{Q}} \in \mathbb{R}^{L \times d_{\text{model}}}$	is the query matrix input,
$\tilde{\mathbf{K}} \in \mathbb{R}^{L \times d_{\text{model}}}$	is the key matrix input,
$\tilde{\mathbf{V}} \in \mathbb{R}^{L \times d_{\text{model}}}$	is the value matrix input,
$\mathbf{W}_i^Q, \mathbf{W}_i^Q, \mathbf{W}_i^Q$	are the projection weights per head and they belong to $\mathbb{R}^{d_{\text{model}} \times d}$,
$\mathbf{W}^O \in \mathbb{R}^{hd \times d_{\text{model}}}$	is the trainable projection to transformer concatenated vectors from the size hd to d_{model},
$[\boldsymbol{\phi}_1; ...; \boldsymbol{\phi}_h]$	is the concatenated matrix of size $L \times hd$.

d_{model} is the hidden layer size and also the embedding layer size. d is the dimension of the query, key, and value vectors, and we usually choose d to be d_{model}/h.

In fig. 2.1 shows the idea of how Multi-Head Attention works.

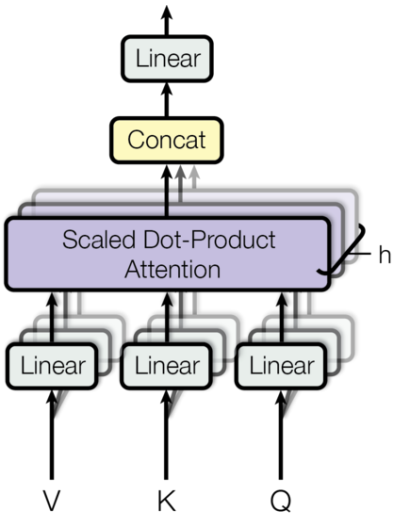


Figure 2.1: The illustration of Multi-Head Attention from Vaswani et al. [25]

2.5.4 Encoder Architecture

Encoder

The transformer employs an encoder-decoder architecture, which is very popular in neural machine translation modeling. In this project, only the Transformer encoder is involved and therefore we will focus on the encoder and the related components.

The encoder consists of a stack of N identical encoder layers. Each layer input is the previous layer output except for the first layer. The first layer input is the element-wise sum of the token embeddings and the positional embeddings. The output of the encoder is the attention-based representation of the input sequence. Fig. 2.2a summarizes the architecture we have mentioned.

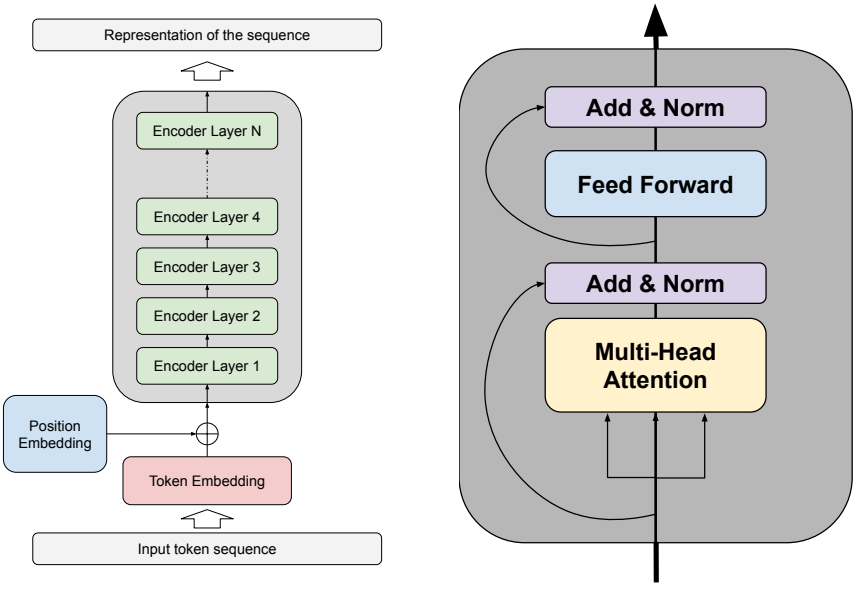
An encoder layer has two sublayers: a multi-head self-attention layer and a fully connected feed-forward layer. We have discussed the multi-head self-attention in section 2.5.3, and we are going to discuss the fully connected layer in the latter part of this section. The encoder layer has a

residual connection around each sublayer and the layer normalization applies to the output of every sublayer. [25]

In other words, the output $\tilde{\mathbf{X}}$ of a sublayer f is:

$$\tilde{\mathbf{X}} = \text{LayerNorm}(\mathbf{X} + f(\mathbf{X})) \tag{2.22}$$

The architecture of an encoder layer is depicted in fig. 2.2b.



(a) The transformer encoder

(b) An encoder layer

Figure 2.2: The architecture of the transform encoder part.

From input token sequence to output representations

When an input token sequence feeds into the Transformer encoder, the tokens are converted into vectors of size d_{model} by the token embedding layer and the position embedding layer.

The vectors become the first encoder layer input. We refer the vectors to be a stacked matrix $\mathbf{Z}_0$ of size $L \times d_{\text{model}}$.

After that, we let $\tilde{\mathbf{Q}} = \tilde{\mathbf{K}} = \tilde{\mathbf{V}} = \mathbf{Z}_0$ and put the query, key, and value matrices into eq. (2.21). The output of multi-head attention passes into

layer normalization and a feed-forward layer together with residual connections as depicted in fig. 2.2b. The resulting output of the first encoder layer is denoted as $\mathbf{Z}_1$. To sum up, we have described how an input matrix flows in an encoder layer.

For the second encoder layer, we do the same as the first layer but replace $\mathbf{Z}_0$ with $\mathbf{Z}_1$ and so-on.

These transformations can be written as:

$$\mathbf{Z}_1 = \text{Encoder}_1(\mathbf{Z}_0) \tag{2.23}$$

$$\mathbf{Z}_l = \text{Encoder}_l(\mathbf{Z}_{l-1}) \quad \forall\, l \in \{2, 3, ..., N\} \tag{2.24}$$

Token Embeddings

Similar to other deep language models, the Transformer uses an embedding layer to learn a transformation from a one-hot encoded vocabulary vector to d_{model}-dimensional embeddings during training.

Positional Embeddings

With only the token embeddings, the model cannot know the positional information of tokens. That means if we swap two tokens, the Transformer model cannot see any difference. It is called permutation invariant. To give the model the information of token orders in the input sequence, we need to add positional embeddings to the token embeddings. As we can see, the positional embeddings $\mathbf{P}$ must have the same size ($L \times d$) of the token embeddings for adding them element-wise.

Vaswani et al. [25] proposed to use a pair of the sine and cosine function with different wavelengths at different dimensions. Thus, the elements of the positional embeddings are:

$$p_{i\delta} = \begin{cases} \sin(i/10000^{\frac{2m}{d}}) & \text{if } \delta = 2m, m \in \mathbb{I} \\ \cos(i/10000^{\frac{2m}{d}}) & \text{if } \delta = 2m+1, m \in \mathbb{I} \end{cases} \tag{2.25}$$

, where i is the token position in the sequence ranged from 1 to L, and δ is the dimension ranged from 1 to d. If δ is an even number, then we take the sine function. We take cosine function in the odd number case.

Vaswani et al. chose this function because for any constant offset Δ, $p_{(i+\Delta),\cdot}$ can be represented by a linear combination of $p_{i,\cdot}$.

Define the wave number $k(m) = 1/10000^{\frac{2m}{d}}$, we have:

$$p_{(i+\Delta),\delta} = \begin{cases} \sin(ki + k\Delta) & \text{if } \delta \text{ is even} \\ \cos(ki + k\Delta) & \text{if } \delta \text{ is odd} \end{cases} \tag{2.26}$$

The offset k becomes a constant phase shift of $k\Delta$. And eq. (2.26) could be decomposed as:

$$\sin(ki + k\Delta) = \sin ki \cos k\Delta + \cos ki \sin k\Delta \tag{2.27}$$

$$\cos(ki + k\Delta) = \cos ki \cos k\Delta - \sin ki \sin k\Delta \tag{2.28}$$

, where $\sin k\Delta$ and $\cos k\Delta$ are constants.

For example, representing $p_{(i+\Delta),15}$, we need $p_{i,14}$ and $p_{i,15}$ and m is 7.

Position-wise Feed-Forward Networks

In each encoder layer, we have one feed-forward network and the network takes input separately from each position independently. As the network ignores the order of the input vectors, we call it is a position-wise feed-forward network.

Each fully connected network has three layers corresponding to two sets of weights and bias. The size of input and output layer is d_{model} and the size of hidden layer is d_{hidden}. The feed-forward network uses ReLU activation in the Transformer.

2.6 BERT

In 2018, the Google AI team presented a new language model called BERT, standing for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers,

for natural language understanding tasks. Similar to OpenAI GPT, training BERT has two stages: unsupervised pre-training and supervised fine-tuning.

The authors of BERT pointed out that the left-to-right architecture which OpenAI GPT uses is harmful to language understanding tasks. It is because this architecture only allows a token to attend to the tokens before it. In other words, the model only understands the current word by the context right before this word, while language understanding task usually requires the context before and after this word.

Therefore, BERT uses the Transformer encoder instead of the decoder, which has a mask in the self-attention module to prevent a token from attending to "future" tokens for preserving the Transformer as an autoregressive language model.

2.6.1 Architecture

The architecture of BERT is almost identical to the Transformer encoder except for the input representation, the output layer, and the activation function. The activation function in BERT is Gaussian Error Linear Unit (GELU) [26].

The input representation in the Transformer model is a sum of token embeddings and positional embeddings. BERT adds one more embedding module, the segmentation embeddings, for indicating the different parts of tokens in the input. These embeddings are learned during training. Figure 2.3 shows the input representation of BERT.

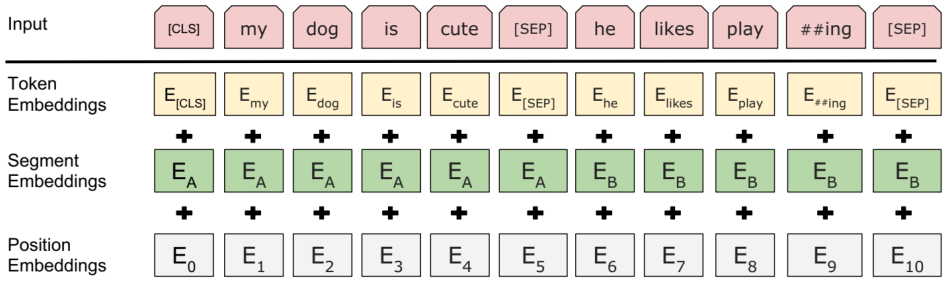


Figure 2.3: BERT Input Representation from Devlin et al. [4]

The output layer is a task-specific layer. The purpose of this layer is to convert the attended sequence representation to the task-specific output. For example, when we use a pre-trained BERT for sentiment analysis, then we create a layer to output the label for classification training with the cross-entropy loss function. When BERT is prepared for pre-training, the output layer is set as a stack of two dense layers. The last dense layer is responsible to map a hidden size vector to a vocabulary size vector. The output of the last dense layer is logit values of tokens over the vocabulary dimension. The overall architecture of BERT for masked language modeling is shown in fig. 2.4.

Note that the last softmax layer is not necessary. It is because we can also predict the token by finding the maximum logit value.

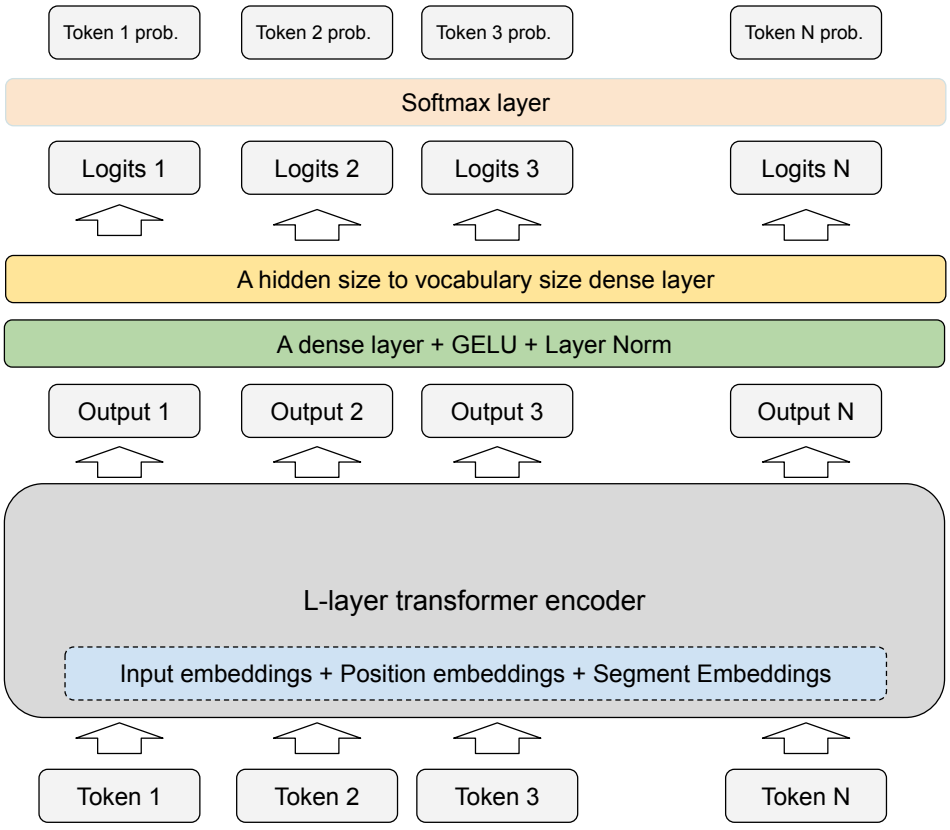


Figure 2.4: The BERT architecture for masked language model

2.6.2 Unsupervised Pre-training for BERT

The key to unsupervised pre-training is that whether we can design self-supervise tasks to train the model to learn from unlabeled data. BERT has two unsupervised pre-training tasks: masked language model and the next sentence prediction. The loss function for the whole pre-training is the sum of the loss from the two tasks.

As we introduced how the masked language model applies to conditional text generation task, we will have a detailed description of this pre-training objective.

To make BERT understand the text in both left-to-right and right-to-left directions, Devlin et al. introduce the masked language model (MLM) training objective instead of generative language model objective used in OpenAI GPT pre-training stage. The authors of BERT said the MLM objective is inspired by the cloze task [27], which is a common exercise type for native and second language learning.

In their implementation, 15% of the input tokens except for special tokens are masked and then train the BERT model to predict mask tokens with cross-entropy loss.

BERT is designed for language understanding tasks, and those tasks do not have any artificial mask tokens in the input sequence of downstream tasks. In order to reduce the pre-training fine-tuning discrepancy, the 15% input tokens would be:

1. replaced with the special mask token `[MASK]` at 80% probability,
2. replaced with a random token at 10% probability, and
3. unchanged at 10% probability.

The last item means that we keep the token remain unchanged although the token is selected to be masked.

For dealing with the tasks based on knowing the relationship between two sentences, the authors designed a self-supervised task that predicting if the input pair of sentences from a text corpus has a ”next-to”

relation. If the second sentence in the pair actually follows the first sentence, then the input pair has the ”next-to” relation and vise versa. The objective function for this task is also cross-entropy loss.

These two tasks use different output layer but share the same set of parameters in the encoder and embeddings.

Chapter 3

Related Work

3.1 Rule-based question generation

Early works in question generation were based on human-designed rules. With sophisticated syntactic transformation rules, we can transform a declarative sentence to the corresponding question. The rules do not support transforming a paragraph into a question because we need to have a main verb as a pivot for some lexical transformations. But one can select a sentence from a paragraph or trim it down into a sentence. In this section, three examples are introduced in this approach.

Mitkov developed a method for generating multiple-choice tests from instructional documents like textbooks or encyclopedias [28]. They first used shallow parsing to identify nouns or noun phrases from the raw text. Next, domain-specific terms were extracted by the term frequency approach. Those sentences having the retrieved terms were transformed into questions by the parsed syntactic information of the sentence, the hypernym of the term from WordNet, and a simple generation rule. Agreement rules were applied to ensure the generated questions are grammatically correct.

In Heilman and Smith's work, questions were over-generated by manually written rules and those generated questions were ranked by a

trained logistic model. In their approach, they first simplified the input text with a set of transformation rules to produce multiple declarative sentences. And then, they transform a declarative sentence into a set of possible questions [29, 30] by syntactic and lexical transformations. By a pre-trained logistic regression model using a question and the corresponding input text as input features, the question can be rated as a value from 0 to 1, representing the range from ”unacceptable” to ”acceptable” [29].

Recently, Dhole and Manning has developed Syn-QG, a collection of semantic and syntactic rules and heuristics to convert a declarative sentence to a question and answer pair. They argued that the traditional rule-based method generates most likely ”what” and ”yes-no” questions. Syn-QG makes use of VerbNet [32] to obtain the semantic information for generating semantically richer questions. [31] With the VerbNet thematic relations and arguments in a predicate, the system can then create different kinds of questions by filling out corresponding templates.

As the rule-based question generation approach requires extensive work on the NLP pipeline and the generation rules, this approach is not the answer to our first research question: *end-to-end question generation framework*.

Summary

The rule-based question generation methods require extensive knowledge in linguistics and well-designed rules to transform a sentence into a question. As we have more robust methods such as the deep learning approach, we do not use the rule-based methods.

3.2 Template-based question generation

When talking about templates, we could easily associate it with a letter for a specific purpose requiring us to fill out several places. A question template works the same. Once we have structured data, the corre-

sponding question can then be populated from an appropriate template.

In Wang, Hao, and Liu [2]'s work, a system generating questions for a medical knowledge assessment was built. [2] A parser was applied to medical articles to extract and classify medical terms and converted unstructured medical texts to structured entries with several fields. Wang, Hao, and Liu [2] created nearly one hundred templates and proposed a method to match a parsed data entry to a question template. However, the disadvantages of this approach are obvious. First, the question templates require extensive human-work. Second, those templates are domain-specific and cannot apply to other fields.

Recently, Lewis, Denoyer, and Riedel [33] has explored how to train a question answering model without using labeled data, which means without using human-labeled questions and answers. [33] Fabbri et al. [34] follow this idea and make use of a template to create questions given a context-answer pair forming a syntactic dataset. They first sampled context paragraphs from the Wikipedia corpus and then select answers by named entity recognition with the spaCy pipeline. A sentence containing the corresponding answer from a paragraph becomes a query to retrieve a similar but not identical sentence as the base for creating a question. The retrieved sentence then is pivoted on the answer to fit the question template by re-structuring the sentence as `[Part A] [answer] [Part B]`. Then the question template `[Wh-word] [Part B] [Part A]` can be populated. The interrogative word of the generated question comes from the named-entity type of the answer and prior statistics from the SQuAD dataset. [34]

Summary

The template method is also not an end-to-end method as we have to design question templates to tackle different types of parsed data. Moreover, we could expect that the generated questions are monotonous as the variation of questions depends on how many templates we create.

3.3 Neural-network-based question generation

In this section, we have a throughout and comprehensive discussion to the state-of-the-art neural-network-based question generation methods.

Almost all the approaches in this section generate questions at sentence-level. Sentence-level means we input a sentence instead of a paragraph, and we transform that sentence to a question by sequence-to-sequence learning.

By preprocessing a paragraph into sentences, we pick a sentence carrying the target answer. In the case that the answer spreads over two or more sentences, these sentences would be combined and called as a ”sentence” or ”passage” in some contexts.

Sequence-to-sequence model

Both rule-based and template-based question generation methods involve human designs from rules and templates to pipeline and procedure. With the success of neural machine translation [23], researchers started to apply the recurrent neural network encoder-decoder architecture with the attention mechanism to text summarization [35] as well as question generation. [36, 37].

Serban et al. [38] presented the model using the architecture for converting knowledge-based facts into natural language questions. Although the embeddings to encode a (subject, relationship, and object) tuple were not learned from the training data, this work inspired the text-to-text question generations.

Du, Shao, and Cardie and Zhou et al. both conducted preliminary studies on data-driven and sequence-to-sequence natural language question generation models.

The model proposed by Du, Shao, and Cardie has two encoders, one

decoder, and an attention module for generating question at paragraph-level. One of the encoders is a bidirectional LSTM network for encoding the sentence with an answer in a paragraph. Another one is also a bidirectional LSTM encoder without the attention mechanism for encoding the input paragraph to a vector representation. The decoder is a unidirectional LSTM network for decoding the concatenated representation from the two encoders. To summarize, the model proposed by Du et al. has a Y-shape architecture.

The attention module uses **General** scoring function from [39]:

$$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i \tag{3.1}$$

, where $\boldsymbol{s}_t$ is the decoder hidden state at step t, $\boldsymbol{h}_i$ the encoder bidirectional concatenated hidden state for word i, and $\mathbf{W}_a$ is a trainable weight matrix.

Zhou et al. model used the RNN encoder-decoder architecture with the attention mechanism for generating questions at sentence-level instead of paragraph-level. The network in the encoder is bidirectional GRU and left-to-right GRU in the decoder. The attention score function is a one layer neural network:

$$\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i) = \boldsymbol{v}_a^\top \tanh(\mathbf{W}_a \boldsymbol{s}_{t-1} + \mathbf{U}_a \boldsymbol{h}_i) \tag{3.2}$$

where $\boldsymbol{s}_t$ is the decoder hidden state at step t, $\boldsymbol{h}_i$ the encoder hidden state for word i, and $\boldsymbol{v}_a$, $\mathbf{W}_a$, and $\mathbf{U}_a$ are trainable parameters for the attention layer.

Moreover, the encoder takes not only a token as input for each word but also the lexical information. The word vector as the encoder network input consists of word embedding vector, lexical feature embedding vector, and answer position indicator. [37] Therefore, Zhou et al. [37]'s encoder takes more features than Du, Shao, and Cardie [36]'s model when encoding a sentence.

In addition, the copy mechanism by Gulcehre et al. [40] was applied in their model to copy rare words from the input sentence.

Song et al. [41] made use of the multi-perspective context matching algorithm [42] in the sequence-to-sequence LSTM encoder-decoder framework with the copy mechanism. A context passage and an answer were

encoded by two bi-directional LSTMs, resulting two sets of hidden vectors representing the passage and the answer. [41] They used a cosine similarity matching function with a trainable weight matrix to create a perspective vector m_k, k means the k-th perspective. [42] Precisely, they matched each passage hidden state at position j, h_j^p against all hidden states of the answer $\{h_i^a\}_{i=1}^{|A|}$, where $|A|$ is the length of the tokenized answer. As we can have multiple cosine similarity matching functions, and therefore it is called multi-perspective context matching. There were three matching strategies, and they further encoded the matching vectors with another bi-directional LSTM; however, we will not go through here. The motivation of applying this matching algorithm is to model the interaction between the answer and the passage for better capturing the relation between them in the encoder part. Song et al. [41] suggested that the matching algorithm can retrieve the contextual information relating the answer and the passage.

Tang et al. [43] considered that the question generation task is conjugated to the question answering task. As a result, the authors reformulated the two tasks as a dual supervised learning task [44]. In other words, the question-answering model and the question generation model trained jointly. It is easy to associate this approach to Generative Adversarial Network (GAN) [45] yet the authors disagreed with this point of view. The authors suggested that their work is dissimilar to GAN since question answering and question generation are both primary learning tasks while the learning a powerful generator is primary and the discriminator is auxiliary. Their results showed that joint training improved the generated question quality comparing to training only the question generation model.

Yuan et al. showed a method to train a model using both supervised learning and reinforcement learning. [46] Their training method has two-phases: train the model with minimizing cross-entropy loss first and then maximize the reward function with the policy gradient method. Usually, we use the reinforcement learning method to train our model because the reward function is non-differentiable [47]. In their work, the reward function is a combination of the F1-score of the generated question and the perplexity of the question. The F1-score is the answer conditional on the generated question and the related paragraph given by another question answering system to the reference answer.

The perplexity is to improve the fluency of the question and given by another LSTM language model.

Kim et al. [48] claimed that some approaches mentioned above encoding answer positions [37, 41] and using the copy mechanism [40] have a serious issue. That is, a great amount of generated questions contains words from their answer. [48] It is because the sequence-to-sequence framework prefers to encode all information from the passage [49]. Also, the copy mechanism does not stop copying the answer from the passage. Therefore, the input answer shall appear in the output question.

To resolve this issue, Kim et al. [48] proposed to encode the answer and the sentence with the answer masking separately. Specifically, answer masking means replacing the answer text with a special token. The attention module cooperating with the answer encoder is called **keyword-net**, which is a stack of attention layers with the dot-product alignment score function. They also used a retrieval style word generator instead of a simple layer, which is the output gate in a LSTM, to predict a token. The BLEU-4 scores of the generated questions in different data-splits outperformed other RNN-based question generation models.

Summary

The works presented in this section are impressive and created some milestones in neural question generation. However, there is a disadvantage of the RNN-based approaches.

The vanilla version seq2seq model with attentions generates poor quality questions in terms of automatic evaluation metrics (BLEU-4 score: 4.26). [36] We can improve the quality by adding different components or changing the model structure to better model the relations between the answer, the context, and the question. For instance, Zhou et al. [37] added the copy mechanism by Gulcehre et al. [40] to deal with rare words and additional feature encoding: answer positions and lexical information. Kim et al. [48] added an encoder for encoding answers and changed the decoder to retrieval style word generator.

However, the robustness of the RNN-based approach is not satisfactory. It is because it requires a brilliant design of the model architecture and specific components for resolving particular flaws from the previous models. The state-of-the-art RNN-based models are complex and designed to solve specific issues in the question generation task. Therefore, we put our eyes on the frontier large-scale language models.

3.4 Pre-training for text generation

This section presents the models using a two-stage training approach: pre-training and fine-tuning. The overviews of the models are described.

GPT-2

GPT-2 is a stack of Transformer decoders, developed by OpenAI. [3] It is the most famous autoregressive language model, or called the left-to-right language model, for generating human-like text. The pre-training object is the causal language modeling (we will talk about it later) to a 40 GB WebText, created by web crawling. Radford et al. [50] fine-tuned the GPT-2 model on various fine-tuning tasks including generative question answering, abstractive summarization, and language translation.

The quality of the generated text is very high but it is not feasible for us. Even the smallest GPT-2 has 117M parameters, and we do not have resources to fine-tune such a large model. In addition, the smallest GPT-2 model is not that good compared to other larger GPT-2 models. [50]

UniLM

UniLM, **Uni**fied pre-trained **L**anguage **M**odel, combines masked, left-to-right or right-to-left, and sequence-to-sequence language models into one shared transformer network. The model can switch between different modes by different types of self-attention masks. During pre-

training, the model has an equal probability to train different types of the language model. The self-supervised task in pre-training is to predict a randomly masked token in a sentence from a text corpus. Dong et al. showed that the model was up to bar in various natural language understanding tasks as well as natural language generation tasks such as abstractive summarization, generative question answering, and question generation. [51]

MASS

MASS stands for **MA**sked **S**equence to **S**equence pre-training. It is a transformer model pre-trained with masked sequence-to-sequence objective. The pre-training objective is a self-supervised task by letting the Transformer model predict randomly masked consecutive tokens. As the encoder and decoder in the Transformer model jointly trained in the pre-training task, MASS performs better in 3 fine-tuning conditional text generation tasks: neural machine translation, text summarization, and conversational response generation comparing to two baseline encoder-decoder models that are separately pre-trained. [5]

Summary

These methods and models are very impressive and worth trying on question generation task. However, these models are large and have limit sizes released to the public. On the other hand, Google Research has released pre-trained BERT miniatures. Therefore, this project focus on using BERT, a masked language model, for question generation.

Chapter 4

Methods

This chapter covers the methods of training and inference for question generation given a context passage and an answer.

We have two training methods for a pre-trained BERT. One generation method is for the first training method and two generation methods for the second training method.

Our question generation methods are autoregressive, which means the model input depends on the output from the last step. Yet the order of generating a sequence of tokens is not limited to from left to right. We will also explore random text generation order in this chapter.

The training method for sequence generation is typically replicating the generation steps but with teacher forcing, which means ignoring what token a model predicts during training. For random order text generation, we can have another training method inspired by the masked language model training.

Our training methods are based sequence-to-sequence learning, which means we assume that we have a training dataset contains the desired input and output sequences. We have a context passage with an answer as an input sequence, and a question as an output sequence. Our training objective is to recreate the questions in the training corpus.

However, the latter method requires confirming the size of the sequence

being generated. We propose a robust method to estimate the size of the generation sequence.

Common generation-related methods like beam search technique, repeated n-gram penalization, and post-processing are presented at the end of this chapter.

4.1 HLSQG

HLSQG is proposed by Chan and Fan [7] and it means sequential question generation using BERT with a highlighted answer phrase. In Chan and Fan [7]'s experiments, HLSQG is the best approach out of their alternative models. They explained that highlighting the answer span removes the ambiguity when the answer appears multiple times in the context passage, comparing with sequential question generation (SQG) which appends the answer at the end of the context passage. In addition, they stated that SQG had poor results on automatic evaluation metrics if the context passage was very long.

4.1.1 Generation

HLSQG is essentially a masked language model and it generates a sequence in a left-to-right manner. Adopting the input format of BERT, we have two sections in the BERT input sequence I: the context section $\langle C \rangle$ and the question section $\langle Q \rangle$.

$$I = [\texttt{[CLS]}, \langle C \rangle, \texttt{[SEP]}, \langle Q \rangle] \tag{4.1}$$

In the context section, a passage containing an answer is:

$$C = [c_1, c_2, c_3, ..., \texttt{[HL]}, a_1, a_2, a_3, ..., a_{|A|}, \texttt{[HL]}, ..., c_{|C|}] \tag{4.2}$$

, where $|C|$ and $|A|$ are the number of tokens of context passage and the answer respectively. We add a pair of tokens `[HL]` to the context tokens $\{c_i\}_{i=1}^{|C|}$ for indicating the answer span in the context passage. As the BERT model has special token embeddings originally of size 2,

we extended that embedding size to 3, as there are 3 special tokens: `[CLS]`, `[SEP]`, and `[HL]`.

In the question section $\langle Q \rangle$, we have the partial generated question at step t as $\hat{Q}_t$:

$$\hat{Q}_t = [\hat{q}_1, \hat{q}_2, \hat{q}_3, ..., \hat{q}_t] \tag{4.3}$$

It means a sequence of generated question tokens from 1 to t.

At the beginning of generating a question, we add a mask token in the question section. We have the BERT input sequence I_1:

$$I_1 = [\texttt{[CLS]}, C, \texttt{[SEP]}, \texttt{[MASK]}] \tag{4.4}$$

Then we ask BERT to predict the masked token by picking the most likely token as $\hat{q}_1$:

$$\hat{q}_1 = \text{argmax}\, p(x_{\texttt{[MASK]}} | I_1) \tag{4.5}$$

, where $x_{\texttt{[MASK]}}$ is a random variable representing the masked token. The generated question token $\hat{q}_1$ is then added before the `[MASK]` token. The input sequence is updated as:

$$I_2 = [\texttt{[CLS]}, C, \texttt{[SEP]}, \hat{q}_1, \texttt{[MASK]}] \tag{4.6}$$

$$= [\texttt{[CLS]}, C, \texttt{[SEP]}, \hat{Q}_1, \texttt{[MASK]}] \tag{4.7}$$

And the same procedure for predicting $\hat{q}_2$:

$$\hat{q}_2 = \text{argmax}\, p(x_{\texttt{[MASK]}} | I_2) \tag{4.8}$$

We do these steps iteratively until the predicted token is `[SEP]` indicating end-of-question:

$$I_t = [\texttt{[CLS]}, C, \texttt{[SEP]}, \hat{Q}_{t-1}, \texttt{[MASK]}] \tag{4.9}$$

$$\hat{q}_t = \text{argmax}\, p(x_{\texttt{[MASK]}} | I_t) \tag{4.10}$$

fig. 4.1 shows the process of generating a question token sequence.

4.1.2 Training

The training principle is *maximum likelihood method* and *teacher forcing*, which are also the common approaches to train a recurrent neural network for conditional text generation. Basically, we train the model to

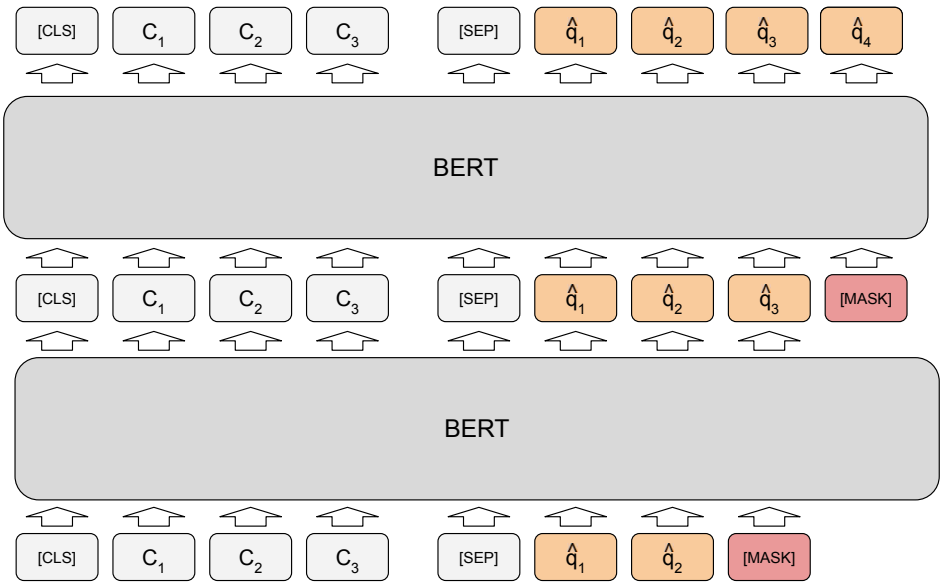


Figure 4.1: The generation process of HLSQG. The process goes from bottom to the top. In this figure, $\hat{q}_3$ and $\hat{q}_4$ are generated. After generate $\hat{q}_3$, we append the `[MASK]` at the end of the input sequence for the next generation step.

predict the ground-truth question token with the input sequence built from the preceding ground-truth question tokens. In other words, the input sequence is from the ground-truth and the loss of the model for training is computed between the predicted token and the ground-truth token as the correct token.

We first introduce the annotation of question tokens as Q:

$$Q = [q_1, q_2, q_3, ..., q_{|Q|}] \tag{4.11}$$

, where $|Q|$ is the number of tokens of the question.

For maximum likelihood method, we estimate the model parameters $\boldsymbol{\theta}$ by maximize the likelihood:

$$\max_{\boldsymbol{\theta}} p(x_{\texttt{[MASK]}} = q_t | I_t; \boldsymbol{\theta}) \tag{4.12}$$

, where the question section $\langle Q \rangle$ of I_t uses the ground-truth question tokens before q_t: $\{q_j\}_{j=1}^{t-1}$.

Step t	Input tokens I_t	$\hat{q}_t$
1	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] [MASK]`	`What`
2	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What [MASK]`	`is`
3	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is [MASK]`	`the`
4	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is the [MASK]`	`priciest`
5	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is the priciest [MASK]`	`component`
6	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is the priciest component [MASK]`	`of`
7	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is the priciest component of [MASK]`	`a`
8	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is the priciest component of a [MASK]`	`CD`
9	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is the priciest component of a CD [MASK]`	`?`
10	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is the priciest component of a CD ? [MASK]`	`[SEP]`

Table 4.1: The generation steps for HLSQG method

For each question in the training set, we can formulate $|Q| + 1$ pairs of $\{I_t\}_{t=1}^{|Q|+1}$ and $\{q_t\}_{t=1}^{|Q|} \cup$ `[SEP]` from the context passage C and the ground-truth question Q. As we put ground-truth question tokens q_t into the BERT input sequences I_t, this technique is called teacher forcing. [52, 53]

Considering the negative log likelihood of the BERT model output, we can write down the loss function for one question as:

$$L(\boldsymbol{\theta}; C, Q) = -\sum_{t=1}^{|Q|+1} \log p(x_{\texttt{[MASK]}} = q_t | I_t; \boldsymbol{\theta}) \tag{4.13}$$

As the model predicts one token at a generation step, we can view the model solves a classification problem over all possible tokens at each step. Therefore, we use *cross-entropy loss* with one-hot encoding in our implementation. By collecting all questions and their I_t and q_t pairs, we can extend the eq. (4.13) to the whole training dataset easily.

Regarding teacher forcing, one advantage is we can keep the training data the same throughout the training. However, as we cannot guarantee that every single token can be successfully predicted during generation. When we have a mistake during generation, the consequent tokens are sampled from a different distribution and the quality of generated text would be lower than expected. This is called *exposure bias*. There are several solutions to reduce exposure bias [54, 53, 55], but we shall not use them due to the scope of this project. We used teacher forcing exclusively in our implementation and our experiments.

4.2 u-PMLM

The full name of u-PMLM is *uniform prior probabilistically masked language model*, which was proposed by Liao, Jiang, and Liu [8].

Liao, Jiang, and Liu [8] gave out the proof that a masked language model with a special training method is equivalent to an autoregressive language model with a permutated generation order.

Inspired by their approach, we shall twist the HLSQG approach and add the essence of u-PMLM into HLSQG.

4.2.1 Generation

In the text generation algorithm from Liao, Jiang, and Liu, they first decided the number of tokens K they will generate. [8] Then, they prepared an initial sequence of K mask tokens `[MASK]` and randomly picked a generation order from the permutations of the set from 1 to K, $\{1, 2, 3, ..., K\}$. Next, the algorithm iteratively predicted a token at the position given by the decided order with the model, and replaced the mask token `[MASK]` with the predicted token.

In table 4.2, a sample text "Is this a generated question ?" is generated using the prediction position order: $3 \rightarrow 6 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2$.

Notice that the sequential order, $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$, is also

a permutation of the set from 1 to 6. We call this order as *sequential*, whereas the order uniformly sampled from the permutations as *random* in chapter 6.

Step	Predict Position	State of the generation sequence
0	N/A	`[MASK] [MASK] [MASK] [MASK] [MASK] [MASK]`
1	3	`[MASK] [MASK] a [MASK] [MASK] [MASK]`
2	6	`[MASK] [MASK] a [MASK] [MASK] ?`
3	4	`[MASK] [MASK] a generated [MASK] ?`
4	5	`[MASK] [MASK] a generated question ?`
5	1	`Is [MASK] a generated question ?`
6	2	`Is this a generated question ?`

Table 4.2: The text generation procedure of the u-PMLM

In the case of generating text given input tokens, they put these input tokens in the initial sequence and filled out the rest of the sequence with the mask token. [8] Then, they used the algorithm to predict tokens at masked positions as stated above.

We make use of this generation approach together with HLSQG to generate questions.

Instead of having an input token sequence I_t of incremental length, we first decide the length of the question being generated, say K. The question length K is sampled from a Poisson distribution (see section 4.2.3). Then, a prediction order $\{\pi_t\}_{t=1}^{K}$ is sampled from the permutation set of 1 to K. Remember that the order is counting from the start of the question section. Next, we create an input sequence I_1 similar to HLSQG:

$$I_1 = [\texttt{[CLS]}, C, \texttt{[SEP]}, \underbrace{\texttt{[MASK]}, \texttt{[MASK]}, ..., \texttt{[MASK]}}_{K \text{ mask tokens}}, \texttt{[SEP]}] \quad (4.14)$$

Similar to u-PMLM, the mask tokens in the sequence shall be replaced iteratively by the predicted tokens until all mask tokens are replaced.

For each step t from 1 to K:

Predict the question token $\hat{q}_t$:

$$\hat{q}_t = \text{argmax}\, p(x_{\pi_t} | I_t; \boldsymbol{\theta}) \quad (4.15)$$

Replace with predicted token $\hat{q}_t$:

$$I_{t+1} = (I_t \setminus \texttt{[MASK]}|_{\pi_t}) \cup \hat{q}_t \tag{4.16}$$

, which means replacing `[MASK]` at position π_t with $\hat{q}_t$

We demonstrate the same question generation example in table 4.1 but in the way of the u-PMLM approach and it shows in table 4.3.

Step t	π_t	Input tokens I_t	$\hat{q}_t$
1	2	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] [MASK] [MASK] [MASK] [MASK] [MASK] [MASK] [MASK] [MASK] [SEP]`	`is`
2	3	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] [MASK] is [MASK] [MASK] [MASK] [MASK] [MASK] [MASK] [SEP]`	`the`
3	5	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] [MASK] is the [MASK] [MASK] [MASK] [MASK] [MASK] [SEP]`	`component`
4	8	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] [MASK] is the [MASK] component [MASK] [MASK] [MASK] [SEP]`	`?`
5	7	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] [MASK] is the [MASK] component [MASK] [MASK] ? [SEP]`	`CD`
6	1	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] [MASK] is the [MASK] component [MASK] CD ? [SEP]`	`What`
7	4	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is the [MASK] component [MASK] CD ? [SEP]`	`priciest`
8	6	`[CLS] The most expensive part of a CD is the [HL] jewel case [HL] . [SEP] What is the priciest component [MASK] CD ? [SEP]`	`a`

Table 4.3: Question generation example using u-PMLM method. π_t is the prediction position at step t. I_t is the input token sequence to the model at step t. $\hat{q}_t$ is the token predicted by the model at position o_t

4.2.2 Training

The training method of u-PMLM is similar to BERT self-supervised pre-training task, which is a learning task to reconstruct the mask tokens from the input sequence. BERT masked 15% tokens in each input sequence in its self-learning task. To us, the portion of tokens being masked for training is indeed a random variable.

For each question Q and the corresponding context passage C, the input sequence I is:

$$I = [\texttt{[CLS]}, C, \texttt{[SEP]}, Q, \texttt{[SEP]}] \tag{4.17}$$

We randomly mask tokens over the question Q by a masking ratio $r_{\texttt{[MASK]}}$ sampled from a uniform distribution $\mathcal{U}(0, 1)$.

$$r_{\texttt{[MASK]}} \sim \mathcal{U}(0, 1) \tag{4.18}$$

Then for each question token, we sample from the Bernoulli distribution of probability $r_{\texttt{[MASK]}}$ to decide if that question token q_i would be masked.

After the masking procedure, we have N_m out of $|Q|$ tokens are masked and the corresponding masked input sequence I_{masked}.

A binary masking array $M = \{m_t\}_{t=1}^{|Q|}$ indicates which question tokens are masked. $m_t = 1$ indicates q_t is masked and vice versa. Combining the masking ratio, the probability of the masking array is:

$$p(M|r_{\texttt{[MASK]}}, Q) = \frac{1}{Z}[r_{\texttt{[MASK]}}^{N_m}(1 - r_{\texttt{[MASK]}})^{|Q|-N_m}] \tag{4.19}$$

, where Z is a normalization constant.

For each masked input sequence I_{masked}, the loss function is:

$$L(\boldsymbol{\theta}; C, Q) = \mathbb{E}_{M|Q}[-\sum_{t=1}^{|Q|} m_t \log p(q_i|I_{\text{masked}}; \boldsymbol{\theta})] \tag{4.20}$$

As the expectation is hard to evaluate, we sample only one masking array $\hat{M} = \{\hat{m}_t\}_{t=1}^{|Q|}$ to estimate the expectation.

Therefore, the loss function becomes:

$$L(\boldsymbol{\theta}; C, Q) = -\sum_{t=1}^{|Q|} \hat{m}_t \log p(q_i | I_{\text{masked}}; \boldsymbol{\theta}) \tag{4.21}$$

As we train the model with mini-batch gradient descent, we have to scale the loss of each sequence by the number of mask tokens in order to have the same order of magnitude. Therefore, we divide the number of mask tokens into the loss in eq. (4.21):

$$L(\boldsymbol{\theta}; C, Q) = -\frac{1}{\hat{N}_m} \sum_{t=1}^{|Q|} \hat{m}_t \log p(q_i | I_{\text{masked}}; \boldsymbol{\theta}) \tag{4.22}$$

, where $\hat{N_m} = \sum_{t=1}^{|Q|} \hat{m}_t$ is the number of mask tokens in a sequence.

4.2.3 Length Prediction

As we do not know the number of question tokens being generated, K, at test time, a simple statistical method is adopted for solving this issue.

We argue that K is a random variable which empirically follows a Poisson distribution. It is because we tend to write a question that not too long and not too short. Also, K is a discrete value variable and the Poisson distribution shall be close to a normal distribution if the expected value λ of the Poisson distribution goes large, say 10.

We estimate two parameters from the training dataset. They are the minimum number of question tokens $|Q|_{\text{min}}$ and the mean of the number of question tokens $\mathbb{E}[|Q|]$ to model.

Since the domain of the Poisson distribution starting from 0 and K has the minimum number of tokens$|Q|_{\text{min}}$, we shift K to $X \sim \text{Pois}(\lambda)$ by $|Q|_{\text{min}}$.

$$X = K - |Q|_{\text{min}} \tag{4.23}$$

$$\lambda = \mathbb{E}[|Q|] - |Q|_{\text{min}} \tag{4.24}$$

When we generate a question at test time, we sample K from the shifted Poisson distribution:

$$X \sim \text{Pois}(\lambda) \tag{4.25}$$
$$K = X + |Q|_{\text{min}} \tag{4.26}$$

4.3 Decoding

4.3.1 Beam-search

The generation methods described above is to choose a token at each step to maximize the conditional probability. Picking the most likely token from the given conditional probability is actually a greedy algorithm. And the greedy algorithm shall not guarantee to yield a locally optimal solution.

We can change the greedy approach by keeping B partially generated results during generation. At each step, we generate top-B tokens at a certain position and add them to our partially generated results. This creates $B \times B$ generation hypotheses. We estimate the score of each generation hypothesis by summing the log probabilities of generated tokens. Only B hypotheses were kept and the remaining were discarded before going to the next generation step.

If the model is HLSQG, which is an autoregressive generation model, we move the partially generated hypothesis once it appends `[EOS]` from the beam to the container of completely generated hypotheses.

Beam search is generally better than greedy search as beam search explores more nodes in the search tree and includes the search path in greedy search. In addition, beam search can reduce to greedy search by setting $B = 1$.

In fig. 4.2, we illustrate an example of beam search of size 2. Notice that the illustration using a left-to-right language model. We can see that when we encounter branching, we keep $B = 2$ hypotheses and rank them using the product of probabilities as the score.

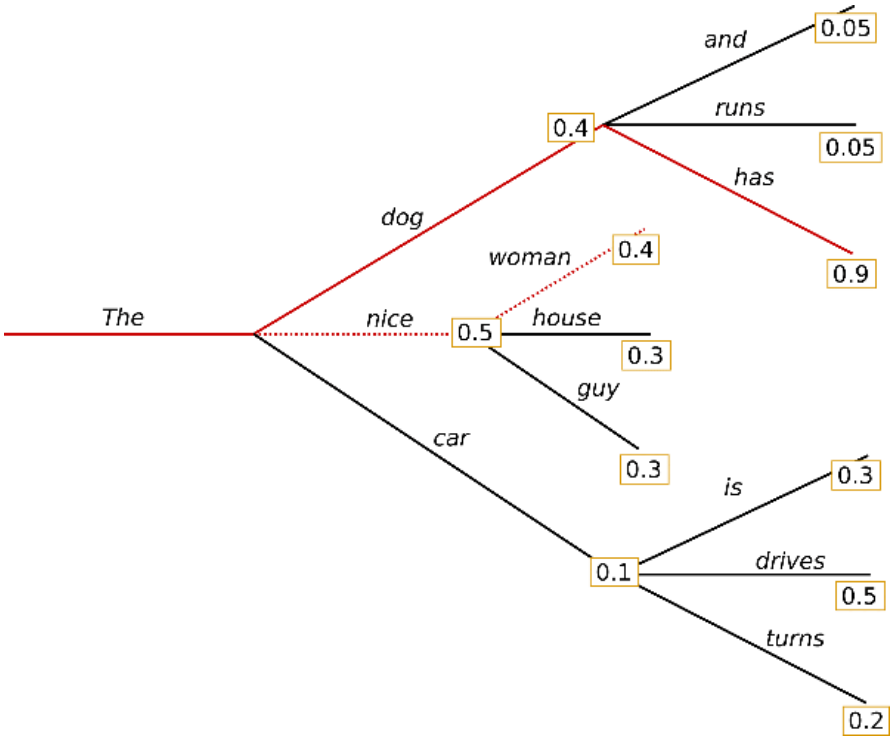


Figure 4.2: Beam search illustration of beam size 2 [56]

4.3.2 N-gram repetition banning

In some cases, the model tends to create repeated patterns during generation. To avoid repetitive phrases in the generated question, we employed the repeated n-grams penalization introduced by Paulus, Xiong, and Socher [57]. We forced the models to not output the same bi-gram more than once by letting the corresponding token logit values be negative infinity.

We collected bi-grams in the generated part of a question and created a list of banning tokens for assigning negative infinity to their log probability. It is obvious to implement the banning algorithm for HLSQG as it is autoregressive.

However, for the u-PMLM, we considered the whole question including the mask tokens `[MASK]`, and the same algorithm was applied to the model. As the generation order is arbitrary in u-PMLM, this method would be failed yet it was not usual.

4.4 Post-processing

We performed a simple post-processing step on the generated question text. The text after the first question mark was truncated.

For example, we have a generated question before the post-processing step: ”What is a good question? clear? or natural?”. Then, the section after the first question mark is considered redundant and that part is removed afterward.

After the post-processing step, the question becomes ”What is a good question?”.

Chapter 5

Experiment

In this chapter, we introduce the dataset used in our experiments by examples. Then the data processing procedures such as question removing, question correction, and answer selection are fully described. Further, the hyper-parameters of the model and for data processing will be mentioned. Finally, we discuss the evaluation methods used in our experiments.

5.1 Datasets

In our experiments, we study the question generation task on the Stanford Question Answering Dataset (SQuAD). [58] The dataset has 107,785 questions given by crowdworkers on 536 Wikipedia articles. Under each Wikipedia article, we have a few paragraphs and each paragraph has several question-answer pairs.

The dataset was original designed for evaluating a machine learning model on the extractive reading comprehension task, which means the answer to the posed question is a span of text extracted from the context. As the questions are posed by the crowdworkers in English-speaking countries, i.e. the United States and Canada, we can use the questions in the dataset as the targets in sequence-to-sequence learning.

In fig. 5.1, we show two examples of the context-question-answer tuple in the SQuAD dataset from a Wikipedia Article: *Queen Victoria*.

Article Title:
Queen Victoria

Context:
In 1853, Victoria gave birth to her eighth child, Leopold, with the aid of the new anaesthetic, chloroform. Victoria was so impressed by the relief it gave from the pain of childbirth that she used it again in 1857 at the birth of her ninth and final child, Beatrice, despite opposition from members of the clergy, who considered it against biblical teaching, and members of the medical profession, who thought it dangerous. Victoria may have suffered from post-natal depression after many of her pregnancies. Letters from Albert to Victoria intermittently complain of her loss of self-control. For example, about a month after Leopold's birth Albert complained in a letter to Victoria about her "continuance of hysterics" over a "miserable trifle".

Question 1:
What was the name of Victoria's eighth child?

Answer to Question 1:
Leopold

Question 2:
Why did members of the clergy oppose the use of chloroform?

Answer to Question 2:
considered it against biblical teaching

Figure 5.1: Two examples of the context-question-answer tuple from SQuAD [58]. The answers are also highlighted in the context.

5.1.1 Data cleansing

After browsing the dataset, we found a limited number of questions were invalid. An invalid question is an incomplete question or a question with a flaw like missing an interrogative word or a question mark.

Since an interrogative word and a question mark play important roles in characterizing a question, the primary objective of data cleansing is to fix the questions with these two issues.

To analyze each question, we used a python package called Stanza [59] to access a Java toolkit called Stanford CoreNLP [60] to perform constituency parsing. Also, we used a simple program to check if a question ends with a question mark.

In the constituency parsing, we kept all the questions which do not have any interrogative tags[1] such as `WDT`, `WP`, `WP$`, and `WRB`. These tags indicate "WH-" words. We also kept the questions not tagged as `SQ`, which means it is a yes-no question. A yes-no question starts with an auxiliary or a modal verb such as "do", "would", "must", "can". For example, "Do you like Swedish food?" is a yes-no question.

After the parsing, we had a set of questions which were regarded as incomplete.

We collected around 2000 incomplete questions. Then we reviewed them one-by-one to decide if we would discard the question or would rewrite the question.

Remove invalid questions

We removed 58 questions because they were not grammatically correct questions. An invalid question could be one or more of the following:

- a cloze style question,
- a question without a question word,
- a declarative sentence, and
- a meaningless sentence.

Some invalid questions are shown in fig. 5.2

[1] The definition of constituent tags could be found at `http://brahma.pub/content/tutorial/NLP/Penn%20Treebank%20Constituent%20Tags.html`

Cloze style questions:
- Philo of Byzantium _____ surmised that air converted to fire
- Feynman wrote many books and gave many _____

Questions without a question word:
- Some of the county attractions are
- The good motor system presently allows for

Declarative sentences:
- This state grew at a rapid pace during the Porfiriato.
- It was believed that this woman could bring death with her arrows.
- This program was awarded the National Medal of Technology and Innovation.

Meaningless sentences:
- I couldn't could up with another question. But i need to fill this space because I can't submit the hit.
- dd

Figure 5.2: Examples of invalid questions to be removed in data cleansing.

Correcting flawed questions

We corrected 1598 questions with mistakes with human judgement. We found out these flawed questions with the constituency parsing because we were not able to find question words in the parsed results.

Flawed questions have one or more of the following :

- spelling errors especially the interrogative word,
- no interrogative word,
- incorrect interrogative word,
- no main verb,
- a wrong word order, and
- no question mark.

The wrong word order is an arguable flaw. Usually, those questions have no comma to separate the main clause and phrases, and then result in ambiguities. We wanted, however, ambiguous free labels and therefore we fixed them.

Fig. 5.3 shows some corrected questions.

5.1.2 Answer selection in SQuAD Dev set

In the SQuAD development set, each question has multiple answers because the dataset is original for question answering. On the other hand, the question generation task requires only one answer span per context. Therefore, we have to choose an appropriate answer per question.

Our strategy is to pick the common answer per question in the development set and use the shortest one as the fallback. Let's look at the example picked from the dataset. [61]

An example tuple has a question: "What color was used to emphasize the 50th anniversary of the Super Bowl?" and three answers:

1. "gold" at character position 488
2. "gold" at character position 488
3. "gold" at character position 512

Therefore, the most common one is answer 1. In the case that we cannot pick the most common one, we use the shortest one as it is concise and exact.

5.1.3 Data splits

The released SQuAD dataset consists of two sets: a training set and a development set. The test set is not released to preserve the integrity of test results. [61] Therefore, we need to divide the accessible part of the dataset into training, test, and validation sets.

Spelling errors:
Original:
Hoe did everyone learn that Beyonce performed for Kaddafi?
Corrected:
How did everyone learn that Beyonce performed for Kaddafi?

No interrogative word:
Original:
According to actor Tom Hiddleston, that is special about Eton?
Corrected:
According to actor Tom Hiddleston, what is special about Eton?

Incorrect interrogative word:
Original:
Ehat is the vegetation type on Antarctica?
Corrected:
What is the vegetation type on Antarctica?

No main verb:
Original:
whats another word for cell membrane?
Corrected:
What's another word for cell membrane?

Wrong word order:
Original:
Throughout European history was child labour seen as a positive?
Corrected:
Was child labour seen as a positive throughout European history?

No question mark:
Original:
How many bills did Andrew Jackson veto
Corrected:
How many bills did Andrew Jackson veto?

Figure 5.3: Examples of the corrected questions.

The accessible part has 490 articles and Du, Shao, and Cardie [36] randomly separated the dataset at the article level. The article titles for

Data Split	Articles	Passage-answer pairs	Avg. passage tokens	Avg. question tokens
Train	442	75668	154.32	12.26
Validation	24	10566	159.61	12.58
Test	24	11877	133.64	12.55

Table 5.1: Statistics of the data split used by Du, Shao, and Cardie [36]. The first two columns are the number of articles and passage-answer pairs. The last two columns are the average number of tokens in a passage and a question.

each set are released[2], and we used these titles as our data split settings in order to make comparisons in literature. The data split statistics are shown in table 5.1.

5.2 Implementation Details

5.2.1 Models

We used five pre-trained BERT models of different sizes downloaded from the Hugging Face model hub[3] automatically by the Python `transformer` library. They are $\text{BERT}_{\text{tiny}}$, $\text{BERT}_{\text{mini}}$, $\text{BERT}_{\text{small}}$, $\text{BERT}_{\text{medium}}$, and $\text{BERT}_{\text{base}}$. The weights of these models can be also obtained from the BERT GitHub repository[4]. These models were pre-trained with a tokenizer that converts all tokens to **lowercase** and therefore our generated questions were all in lowercase.

Pre-training is proven to be effective in various contents and research works. Hence, we did not use any randomly initialized models in our experiments due to the resource limitation and the project time limitation.

We summarized the model architecture related parameters in table 5.2. Notice that the hidden layer dimension H equals the size of embedding layers and the number of heads is always set to $H/64$. [4, 62]

[2] https://github.com/xinyadu/nqg

[3] `https://huggingface.co/google/bert_uncased_L-2_H-128_A-2`

[4] `https://github.com/google-research/bert`

Size	**L**	**H**	**A**	No. of parameters (in millions)
tiny	2	128	2	4.4
mini	4	256	4	11.3
small	4	512	8	29.1
medium	8	512	8	41.7
base	12	768	12	110.1

Table 5.2: The model architecture parameters. **L** is the number of transformer encoder layers. **H** is the dimension of hidden layers. **A** is the number of attention heads. The last column is the number of parameters in millions.

5.2.2 Data Processing

We implemented the data processing module in Python. The text data cannot directly feed into our models and therefore this module is to compile text data into an input-ready format.

The data processing module has three purposes. First, it converts text to integer token ids. Second, it adds highlighting tokens `[HL]` to indicate the answer span. Third, it selects a context span in the case that we have to truncate a context passage.

Tokenization

Due to fine-tuning from BERT, our tokenization method is **WordPiece**, the same tokenizer used for BERT [4]. WordPiece is a sub-word level tokenization method, which was originally used in the Japanese and Korean speech recognition system by Google [63] and further used in Google's Neural Machine Translation System [64].

As WordPiece is a data-driven tokenizer requiring training, we used the trained tokenizer from Google. The number of tokens of the trained tokenizer is 30522. Also, the trained tokenizer is *uncased*. It means the tokenizer ignores cases by forcing input text to lowercase.

Context selection

As our models have the maximum number of input tokens, we need to truncate the tokens when necessary. We set the maximum number of tokens in a sequence to be 384, while the model maximum is 512. It was because we tested 384 tokens and that was optimal between the training speed and the number of data being truncated.

The maximum number of question tokens was 30. In the case that the question had more than 30 tokens, we kept only the first 30 tokens.

Moreover, we considered that keywords are indicative in a question [48], and therefore we kept keywords in the context as many as possible. A statistical approach was used to extract keywords by rareness. The keywords are of the word frequency below 1000 in the training set. In the development set and test set, we did not have a keyword list.

Here is the equation to show how many tokens we keep for the context:

$$N_c = N_{\text{max}} - N_q - N_{\text{special}} \tag{5.1}$$

, where N_c is the number of context tokens, N_q is the number of question tokens, N_{max} is the maximum number of input tokens, and N_{special} is the number of special tokens like `[SEP]` and `[CLS]`.

There were two ways to truncate the context part: keeping the first N_c context tokens and selecting a sliding window of N_c context tokens.

The first approach was used in truncating the context part of an input sequence from the development set and the test set. The second approach used a window of size N_c sliding over the context section with slide 128. We then picked the window having the most keywords together with the answer span.

5.2.3 Training

Our models and training procedures were implemented using *PyTorch* v1.5.0 together with *Transformers* v3.0.2, a Python library providing various transformer architectures from Hugging Face. [65]

Mixture precision training

Mixture precision training is a technique to boost the training speed and halve the memory consumption without losing the model performance and altering hyper-parameters. [66] We used a Python package `apex` from NVIDIA supporting mixed precision training in PyTorch[5] throughout training.

However, this method has a limitation on the choice of GPU. Only Volta and Turing architecture GPU having Tensor Cores can accelerate mixed-precision training. [67]

Platform

All of our model training and the evaluation were done with Google Cloud Compute Engine. We used one virtual machine with one 16 GB NVIDIA Tesla T4 GPU. Other specifications of the virtual machine are listed in the Appendix.

Training parameters

The dropout probability of all fully connected layers, the self-attention layers, and the sum of all embeddings, which are token embeddings, position embeddings, and segment embeddings was set to 0.1. These are the same setting as the original transformer model [25].

The Adam optimization algorithm was used to train our models [68] with the initial learning rate of 5e-5. The batch size of BERT-base was 24 and other models were 32 because of the GPU memory limitation. We trained all models with 2 epochs in the HLSQG approach and 27 epochs in the u-PMLM approach equivalently. As the limitation of resources, we had only two epochs for training. We made sure that the two approaches have the almost same number of iterations for every model.

We did not have early stopping because we have only 2 epochs. The

[5] `https://github.com/NVIDIA/apex`

number of epochs is smaller than 5 which were used by Chan and Fan [7] in their experiments. Also, we used a validation set to keep tracking the training performance.

5.2.4 Question Generation

There were two generation orders for generating a question used in our experiments, and they are sequential and random generation order. The two generation orders are mentioned in section 4.1.1 and section 4.2.1.

We used beam search of beam size 3 for generating questions on the test dataset and greedy search during training for tracking the model generation performance. All text generation methods banned bi-gram repetition in the generated text.

5.3 Evaluation Methods

In our experiments, we have two types of evaluations: automatic evaluation and human evaluation.

For the automatic evaluation, we use the Python package called `nlg-eval`[6] published by Sharma et al. [69]. It covers the evaluation metrics we are going to present.

For the human evaluation, we built a survey on the generated questions and asked respondents to evaluate each question on three characteristics: *naturalness*, *relevance*, and *answer-specificity*.

5.3.1 Automatic Evaluation

Automatic evaluation is a group of methods to evaluate the quality of a sentence from a machine with that from a human. Typically, the evalu-

[6] `https://github.com/Maluuba/nlg-eval`

ation metrics are n-gram based similarity measurements [70], and they show how close a hypothesis is to its reference. Usually we call the generated sentences as *hypotheses* and the human annotations as *references*.

Sticking to Zhou et al. [37], Song et al. [41], Kim et al. [48], and Chan and Fan [7], we employed the following automatic evaluate metrics to examine the generated question quality:

- BLEU-n: measure n-gram precision of a hypothesis against the corresponding reference [71]. n usually ranges from 1 to 4. This metric is originally used in machine translation.
- METEOR: a weighted F-score F_β of uni-gram precision and recall between a hypothesis and a reference. The recall is weighted 9 times as important as precision. [72]
- ROUGE$_L$: calculates the F-score of the longest common subsequences of a hypothesis and a reference. It is originally for evaluating text summarization. [73]

There are criticisms for using the automatic evaluation metrics in natural language generation tasks [74, 75, 76]. Also, Reiter [77] suggested BLEU should not be used outside of machine translation.

However, automatic evaluation metrics are reproducible, affordable, and immediately available. [78] Reproducible means that other people can easily re-evaluate the generated text to the reference text with the definitions of the metrics and therefore they are objective. They are affordable and immediately available because the evaluation software packages for these metrics are publicly available and usually open-sourced. [78]

That is the reason why automatic evaluation metrics remain popular. In addition, an automatic evaluation metric can be used during the training phrase to keep tracking the model generation performance instead of the loss function which only tells how well the model is in teacher forcing mode.

Therefore, we evaluated our methods using automatic evaluation metrics.

All scores are between 0 and 1 but typically people will report the numbers time 100 for removing the leading zero(s). In our results, we followed this convention.

5.3.2 Human Evaluation

To have a comprehensive evaluation of the questions generated from different methods, a survey was designed to ask respondents to value a question in 3 different aspects:

- Naturalness: Respondents were asked to consider the grammatical correctness, fluency, and the naturalness of the question being rated.
- Relevance: Respondents rated the specificity of the question to the given context passage.
- Answer-specificity: Respondents were asked to judge how well the highlighted answer in the passage is correlated to the question.

We have four scales for each aspect, and they are "Very Poor", "Poor", "Good", and "Very Good". We do not use an odd-numbered scale to prevent respondents from answering a neutral option.

The survey has 30 sets of questions and each set has one context passage and one highlighted answer. Each set has three questions and the question is generated by:

- HLSQG + sequential decoding: We train the BERT model in the HLSQG approach and generate question tokens from left to right.
- u-PMLM + sequential decoding: We train the BERT model in the u-PMLM approach and generate question tokens in from left to right.
- u-PMLM + random decoding: We train the BERT model in the u-PMLM approach and generate question tokens in random order.

Chapter 6

Results

In this section, the results of the length prediction model are introduced first. Then, three types of results shall be presented, and they are training metrics, automatic evaluation results, and human evaluation results. Finally, we have an explicit example of our question generation methods.

To summarize, we have two training methods: HLSQG and u-PMLM, five sizes of pre-trained BERT models, two approaches to generate text: sequential (from left to right) and random order, and two methods to decide the question token length: sampling from the Poisson distribution and peeking into the true length from the dataset itself.

6.1 Length Prediction Model Estimation

In the u-PMLM method, we need to predict the question token length first and then arrange masks for token predictions during generation as we mentioned in section 4.2.3.

In fig. 6.1, the frequency distributions of the number of question tokens over the training set and the validation set are shown. Although we did not use the normal distribution for token length estimation, we estimated and plotted out the normal distribution for reference. Since

the normal distribution is a continuous probability distribution, therefore it is *not suitable* for describing a discrete variable (i.e. the token length).

When estimating the parameters for the distributions, we ignored the counts of the token lengths over 30, which is the maximum number of question tokens.

The estimated parameters can be found in table 6.1. The negative log-likelihoods of the distributions are presented in the same table. We estimate the minimum, mean, and variance of the question token length from the training data. After that, we put those parameters to the Poisson and normal distribution and calculate the negative log-likelihoods (NLL) on the training and validation dataset.

As we set the maximum number of tokens in a question, the average number of tokens in table 6.1 is not identical to that in table 5.1.

Estimated parameters	Training dataset	Validation dataset
Minimum token length	3.0	4.0
Mean of the token length	12.22	12.55
Variance of the token length	15.81	16.06
Poisson NLL	2.82	2.83
Normal NLL	2.8	2.81

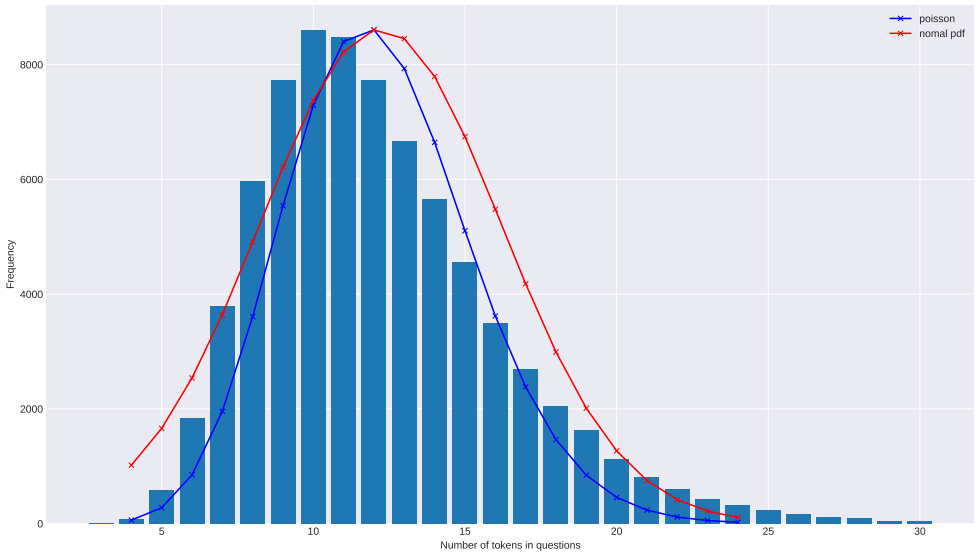
Table 6.1: The estimated parameters for modeling the question token length distributions.

6.2 Training Performance

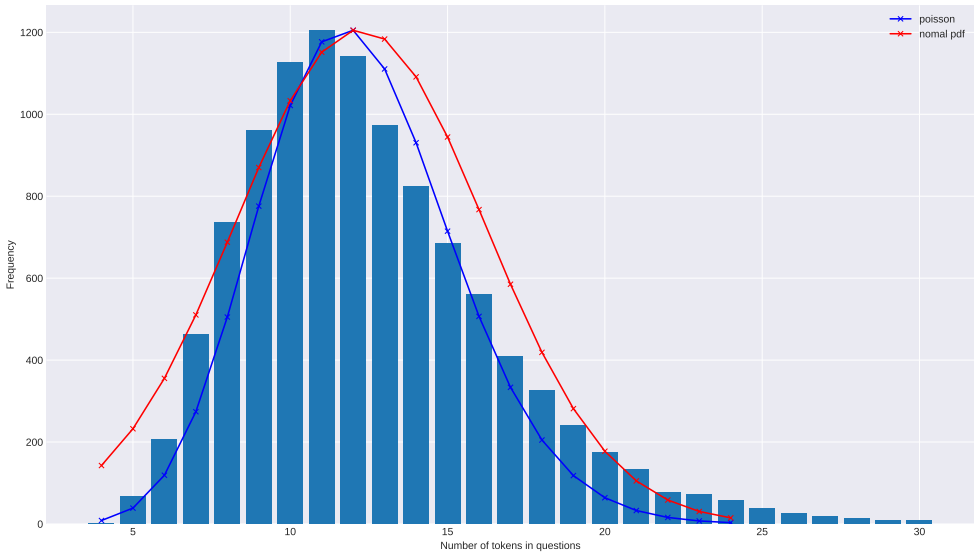
In this section, we will present the training results and the curves when training the BERT models in the HLSQG and the u-PMLM approaches.

Since the numbers of iterations per epoch of two training schemes are different, we make the iteration steps as the x-axis for comparing across two training methods on different sizes of models.

Table 6.2 summarizes the information related to the number of itera-



(a) Training set question token length frequencies



(b) Validation set question token length frequencies

Figure 6.1: Plots of the question token length frequency distributions. The blue curves and the red curves were the Poisson distribution and the normal distribution of parameters estimated by only the training set.

tions. The last four columns in the table are the training batch size, the number of training epochs, the number of training iterations, and the number of samples per epoch.

Method	Size	Batch size	Epochs	Iterations	No. of samples
HLSQG	base	24	2	83540	1002476
u-PMLM	base	24	27	85131	75668
HLSQG	except base	32	2	62656	1002476
u-PMLM	except base	32	27	63855	75668

Table 6.2: Training parameters, the number of iterations, and the number of samples in training set

The largest model BERT-base of 110.1 million parameters took 24 hours for training on our platform stated in section 5.2.3. The second-largest model BERT-medium of 41.7 million parameters took 12 hours and the smallest model BERT-tiny of 4.4 million parameters took 3 hours for training.

Training loss and validation loss

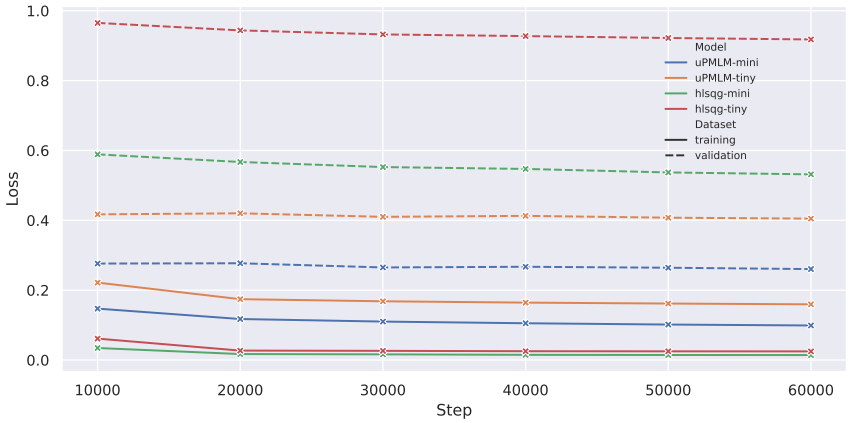
In fig. 6.2, we can find loss function plots per 10000 iterations over the training set and the validation set. We grouped different sizes of models into three groups for better visualization.

The loss functions of two training schemes are mentioned in section 4.1.2 and section 4.2.2.

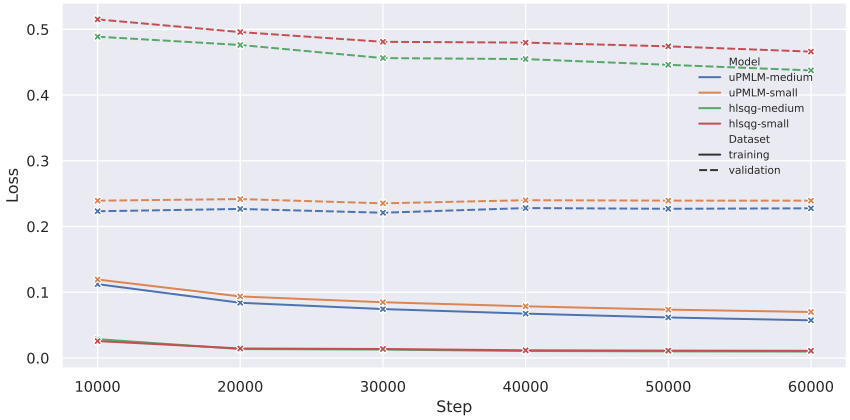
We can see that all models kept going to converge except BERT-mini and BERT-tiny. We shall expect that the models perform better when we can have more training iterations.

Training Accuracy

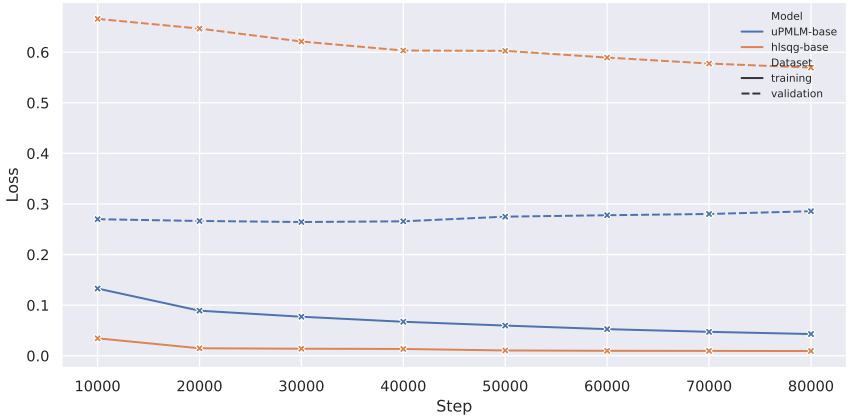
Given that predicting a word is a classification problem regarding tokens as labels, we could also check how good the model predicts a word in teacher forcing mode.



(a) Mini & tiny size models



(b) Medium & small size models



(c) Base size model

Figure 6.2: The training loss and validation loss against the training steps of all models.

Fig. 6.3 shows the word prediction accuracy on the training set and the validation set over training steps.

The figure shows generally HLSQG is better than u-PMLM in terms of the word prediction accuracy. However, by considering that the u-PMLM could predict a few tokens on each sentence, the training accuracy and validation accuracy are reasonable and not that bad.

Like what we observed in the loss plots, the small, medium, and base size BERT can further improve performance by adding more training iterations.

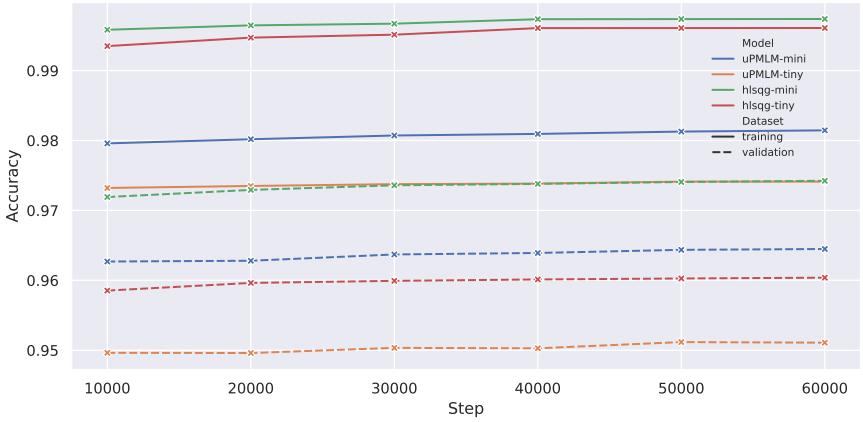
We should notice that the u-PMLM base model validation loss slightly goes down but it does not mean that the u-PMLM model is going to be overfitting. It is because the masking scheme is stochastic and the accuracy measurements are not deterministic. Therefore, we cannot draw a concrete conclusion regarding this issue.

BLEU-4

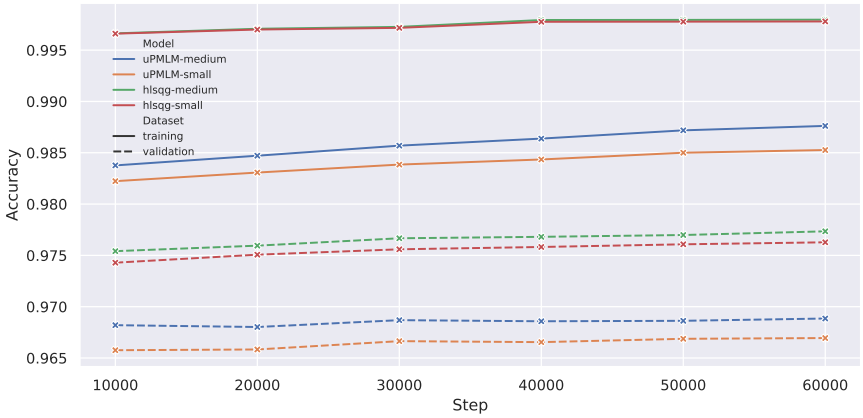
As our target is to generate questions instead of predicting a word in teacher forcing mode, we sampled 2000 data in the validation set and evaluated the quality of generated questions by BLEU-4 score during training.

The token generation method for HLSQG was sequential, from left to right, and greedy search until generating the stop token, `[SEP]`. For u-PMLM, we used the true question length in the unit of token instead of sampling, which is described in section 4.2.3. As we trained the BERT models in two approaches separately, the sampled data per 10000 step shall not be the same.

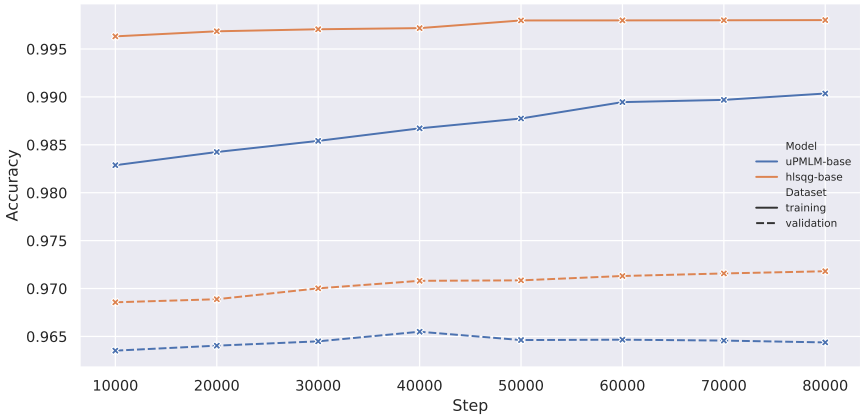
In fig. 6.4, we can see that the two models are comparably good. Looking at fig. 6.4b, we notice that the BLEU-4 scores of two approaches were improving while more training steps updated the models. For the smaller models in fig. 6.4a, the BLEU-4 scores increased by a small amount overall. It is suspicious that for the models other than $\text{BERT}_{\text{base}}$, the representation power of those models is limited. Therefore, the smaller models did not improve that much throughout the training.



(a) Mini & tiny size models



(b) Medium & small size models



(c) Base size model

Figure 6.3: The word prediction accuracy in teacher forcing mode on training and validation set.

Surprisingly, the u-PMLM approach can train the model achieving an acceptable level in few ten thousand steps given that the token length of a question being generated is provided.

6.3 Automatic Evaluation

This section describes the automatic evaluations on the generated questions from the test set.

The questions were generated using beam search of beam size 3 and the size of penalizing repeated n-grams was set as 2.

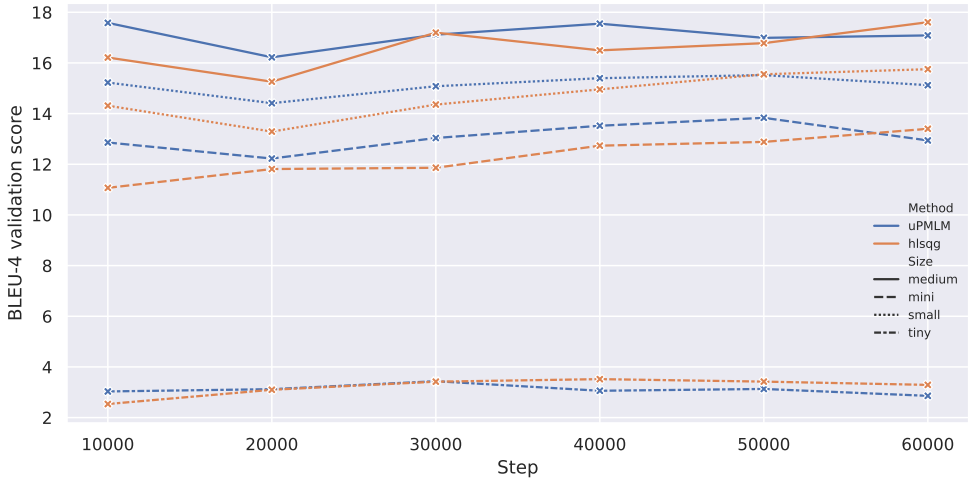
Due to the layout limitation, we cannot present all metrics. We selected BLEU-4, METEOR and ROUGE_L to be listed in this section. Other BLUE scores were arranged in appendix B.

Table 6.3 and table 6.4 show the evaluation results on $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{medium}}$ respectively. Table 6.5 shows the scores for all combinations of methods, text generation orders, sizes, and the question token length prediction methods.

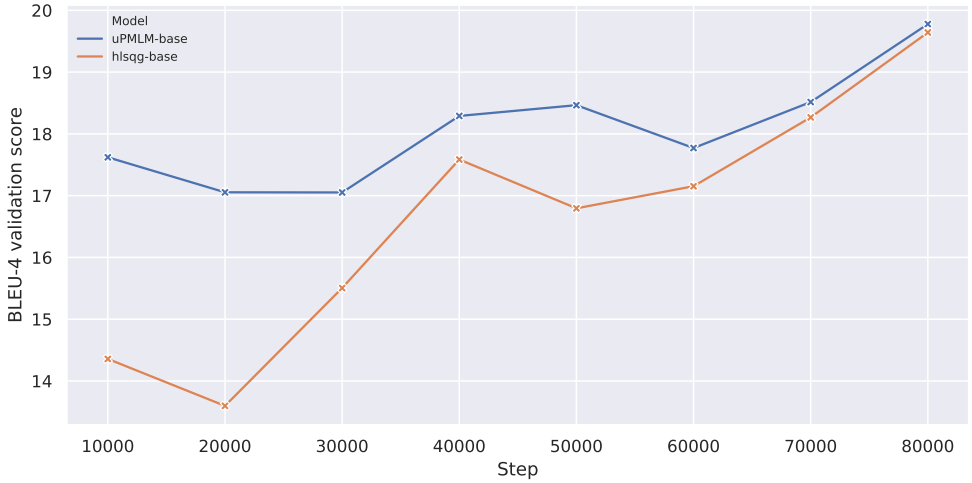
For the HLSQG training method, we found that two models behaved almost the same regarding these three metrics. $\text{BERT}_{\text{base}}$ is slightly better than $\text{BERT}_{\text{medium}}$ but we should keep in mind that the number of parameters of BERT-base is four times as large as that of BERT-medium.

In the u-PMLM method, four combinations of text generation orders and question token length decisions are listed in table 6.3 and also table 6.4. Comparing two methods to decide the generation length, we should notice that a significant drop takes place when shifting from using the truth to sampling.

One possible reason for the drop is insufficient places when sampling a small token length. The question structure words like interrogative words (WH-words) and the main verbs take a certain amount of token places. As a result, only one keyword or no keyword shall be copied from the context passage to the question being generated.



(a) Models of all sizes except the base



(b) Base size model

Figure 6.4: The BLEU-4 score evaluated on the model generated questions from 2000 data sampled from the validation set.

An example that fell into this case appeared in our survey for human evaluation. And that example is demonstrated in fig. 6.5. In that example, we can see that the questions generated from u-PMLM missed all important keywords like ”hard drive” and ”original iPod” and the indicative question words ”How large”.

Reference question:
How large was the hard drive on the original iPod?

HLSQG question:
how much hard drive did apple have that put ”1,000 songs in your pocket” in 2001

u-PMLM (sequential, sample token length) question:
how big was it?

u-PMLM (random, sample token length) question:
how big was drive?

Figure 6.5: An example of sampling the small number of the question token lengths for u-PMLM question generation.

Comparing the generation order, using sequential decoding is better than random order decoding. We do not have any concrete supports to explain this phenomenon, yet we shall have an educated guess.

We argue that the layout of the input sequence forces the models to favor generating text from left to right.

In the input sequence of the models, the context passage is placed before the question. After the question section, the models have nothing to attend to. Moreover, the question holds tokens appearing in the context passage as those are keywords for specifying the answers. In addition, the number of tokens in a context passage is typically much more than that of a question.

Therefore, the information flows from left to right, and the bi-directional language model was trained to consider the information merely from the left-hand side. In order words, the arrangement of the input sequence and the imbalance between the token lengths of a context pas-

sage and a question suggest the text generation direction from left to right.

Method	Gen. Order	Token Len.	BLEU-4	METEOR	ROUGE$_L$
HLSQG	sequential	N/A	15.89	23.74	39.11
u-PMLM	sequential	true	17.28	24.04	41.0
u-PMLM	sequential	sample	12.95	21.27	35.26
u-PMLM	random	true	15.68	23.56	40.26
u-PMLM	random	sample	11.7	20.86	34.69

Table 6.3: Selected automatic evaluation results of the BERT$_{\text{base}}$ model

Method	Gen. Order	Token Len.	BLEU-4	METEOR	ROUGE$_L$
HLSQG	sequential	N/A	15.04	23.32	38.09
u-PMLM	sequential	true	17.49	23.71	40.52
u-PMLM	sequential	sample	13.23	21.05	34.9
u-PMLM	random	true	15.35	23.07	39.54
u-PMLM	random	sample	11.44	20.39	33.89

Table 6.4: Selected automatic evaluation results of the BERT$_{\text{medium}}$ model

6.4 Human Evaluation

The section reports the results of the human evaluation mentioned in section 5.2.3. We asked each respondent to score 90 questions from 30 sets of test data in terms of language naturalness, context-relevance, and answer-specificity on a scale from 1 to 4. The scores correspond to the scales mentioned in section 5.3.2; that is, ”Very Poor” maps to 1, ”Poor” maps to 2, ”Good” maps to 3, and ”Very Good” maps to 4.

We generated questions using the only BERT$_{\text{base}}$ model but in total 3 different training and generation approaches. The BERT-base was trained by two different methods and generate questions in two orders. We chose BERT-base due to its relatively good performance. For the

Method	Size	Gen. Order	Token Len.	BLEU-4	METEOR	ROUGE$_L$
HLSQG	base	sequential	N/A	15.89	23.74	39.11
HLSQG	medium	sequential	N/A	15.04	23.32	38.09
HLSQG	small	sequential	N/A	14.29	22.43	36.77
HLSQG	mini	sequential	N/A	12.14	20.15	33.86
HLSQG	tiny	sequential	N/A	4.33	9.95	22.14
u-PMLM	base	sequential	true	17.28	24.04	41.0
u-PMLM	base	sequential	sample	12.95	21.27	35.26
u-PMLM	base	random	true	15.68	23.56	40.26
u-PMLM	base	random	sample	11.7	20.86	34.69
u-PMLM	medium	sequential	true	17.49	23.71	40.52
u-PMLM	medium	sequential	sample	13.23	21.05	34.9
u-PMLM	medium	random	true	15.35	23.07	39.54
u-PMLM	medium	random	sample	11.44	20.39	33.89
u-PMLM	small	sequential	true	15.8	22.53	38.3
u-PMLM	small	sequential	sample	12.39	20.24	33.57
u-PMLM	small	random	true	12.87	21.58	36.88
u-PMLM	small	random	sample	10.09	19.44	32.64
u-PMLM	mini	sequential	true	14.15	21.09	35.9
u-PMLM	mini	sequential	sample	11.25	19.04	31.93
u-PMLM	mini	random	true	10.29	19.66	34.02
u-PMLM	mini	random	sample	8.43	17.98	30.46
u-PMLM	tiny	sequential	true	4.4	10.84	23.04
u-PMLM	tiny	sequential	sample	4.22	10.43	21.91
u-PMLM	tiny	random	true	2.68	10.34	22.2
u-PMLM	tiny	random	sample	2.59	9.92	21.05

Table 6.5: Selected automatic evaluation results for all combinations

model trained by u-PMLM, the question token lengths were sampled from the Poisson distribution estimated by maximum likelihood using the token lengths of training questions.

Three persons returned the surveys, and we then averaged the scores per evaluation category for each approach. The summarized results are tabulated in table 6.6.

Overall, all HLSQG and u-PMLM using sequential order have scores greater than 2.5 in all aspects. The model trained with the HLSQG approach and generating questions from left to right direction performed the best over all evaluations.

We also analyzed the inter-rater agreement with Krippendorff's alpha [79] and Fleiss's kappa [80]. The Krippendorff's alpha was 0.084 and Fleiss's Kappa was 0.087. Two coefficients suggest that raters have a poor inter-rater agreement and little inter-rater reliability.

Method	Gen. Order	Naturalness	Relevance	Answer-Specificity
HLSQG	sequential	**2.92**	**3.32**	**3.42**
u-PMLM	sequential	2.69	2.99	3.16
u-PMLM	random	2.24	2.81	2.81

Table 6.6: Human Evaluation results

6.5 Case Study: Question Generation about KTH

To further evaluate different methods, we conduct a case study using data not coming from SQuAD v1.0. A short passage about KTH from Wikipedia was selected.

We generated questions using BERT-base model training by HLSQG and u-PMLM. Only the sequential generation order was employed for this case study. Instead of sampling the question length, we specified the token lengths for the questions. We used beam search of size 3 as well. Other generation settings are the same as what we have men-

tioned in section 5.2.4.

The results are shown in fig. 6.6. We can see that the overall qualities of the generated question coming from the two models were comparable in this case study. One advantage of u-PMLM is that we can control how many tokens we need. The more tokens we give to the model, the more specific the question is. In other words, we gave more spaces to the model filling out the details.

Two methods generated few questions which are wrongly placed ”the” in front of ”kth” as KTH is a proper noun.

The tokens like ”original”, ”originally”, and ”also” do not appear in the context passage. It shows that the models were not just copying words to create a question but also using new words to make the generated text more natural and fluent.

6.6 Comparison with Other Works

In this section, we compare our models with previous works. We collected the automatic evaluation results on the neural question generation models using the data split proposed by Du, Shao, and Cardie [36].

Our results as well as the previous work results are arranged in table 6.7.

Chan and Fan [7] sent us the data they used for training and generation. However, we found that their data split is not at the article level. Hence, we do not put their results into our table.

In general, the HLSQG model was on a par with the answer-separated seq2seq model (**ASs2s**) proposed by Kim et al. [48] by comparing with the BLEU-4 score and the METEOR score.

In the case that question token lengths were sampled from the Poisson distribution, the u-PMLM performance was much lower than **ASs2s** for sequential and random generation orders. The u-PMLM with se-

Article Title:
KTH Royal Institute of Technology

Context:
KTH was established in 1827 as Teknologiska Institutet (Institute of Technology), and had its roots in Mekaniska skolan (School of Mechanics) that was established in 1798 in Stockholm. But the origin of KTH dates back to the predecessor to Mekaniska skolan, the Laboratorium Mechanicum, which was established in 1697 by Swedish scientist and innovator Christopher Polhem. Laboratorium Mechanicum combined education technology, a laboratory and an exhibition space for innovations.[4] In 1877 KTH received its current name, Kungliga Tekniska högskolan (KTH Royal Institute of Technology). The King of Sweden Carl XVI Gustaf is the High Protector of KTH.

HLSQG:
what was the original name of the kth?

u-PMLM, sequential order, 7 tokens:
what was kth originally named?

u-PMLM, sequential order, 10 tokens:
what was the original name of the kth?

u-PMLM, sequential order, 20 tokens:
what was the original name of the school of kth that was established in the year of 1827?

u-PMLM, sequential order, 30 tokens:
what was the name of the school that was founded in the mekaniska skolan, and it was also the original kth name?

Figure 6.6: The questions generated by different approaches on a passage related to KTH history. We have two approaches, HLSQG and u-PMLM, but the latter one can specify the generation sequence length.

quential generation order (u-PMLM-base-seq-sample_len) is slightly better than **NQG** by Du, Shao, and Cardie [36] in terms of BLEU-4 and METEOR but not ROUGE$_L$.

However, if we remove the length prediction model and give u-PMLM the true question length, u-PMLM with random generating order is comparable to **ASs2s**. The sequential u-PMLM is better than **ASs2s** regarding the scores except ROUGE$_L$.

Models	BLEU-4	METEOR	ROUGE$_L$
NQG by Du et al. [36]	12.28	16.62	39.75
M2S+cp by Song et al. [41]	13.98	18.77	42.72
ASs2s by Kim et al. [48]	16.20	19.92	43.96
s2s-a-at-mp-gsa by Zhao et al. [81]	16.38	20.25	44.48
Our works:			
HLSQG-base-seq	15.89	23.74	39.11
u-PMLM-base-seq-sample_len	12.95	21.27	35.26
u-PMLM-base-rand-sample_len	11.70	20.86	34.69
u-PMLM-base-seq-true_len	17.28	24.04	41.00
u-PMLM-base-rand-true_len	15.68	23.56	40.26

Table 6.7: Automatic Evaluation results of our base-size models and the previous neural question generation models.

Chapter 7

Discussion

In this project, we successfully tuned pre-trained BERT models for the question generation task. Although BERT is originally designed for natural language understanding, we turned BERT for conditional text generation by the fine-tuning tasks which are closely related to the masked language model pre-training objective.

7.1 Pre-training

Pre-training definitely helps the question generation task.

Although we do not have an experiment to directly verify that pre-training works for question generation task, yet pre-training strategy is verified by previous researches and widely adopted by the research community especially for natural language understanding tasks. We did not have an experiment to compare with or without pre-training, yet we did a small trial to verify this idea.

In our small trial, $\text{BERT}_{\text{small}}$ without pre-training cannot generate any meaningful questions. The original purpose of the trial was to test whether a $\text{BERT}_{\text{small}}$ without pre-training can overfit a tiny dataset. The $\text{BERT}_{\text{small}}$ was trained with only one passage-answer tuple in the trial

following a sanity check[1]. The result of this test was that the $\text{BERT}_{\text{small}}$ without pre-training generated only padding tokens (`[PAD]`) and the $\text{BERT}_{\text{small}}$ can generate a syntax-correct question with few iterations.

In order to use the limited computation resources efficiently, we decided directly start our experiments from pre-trained models.

7.2 HLSQG

Highlighting the answer span in the context passage is the simplest and the most effective way to tell the model where the answer is without explicit modeling. With the use of a pair of special tokens `[HL]`, the model can capture the location of the answer in the input passage and dig out the contextual dependency between the answer and the passage through training.

However, this fine-tuning approach is not closely related to the BERT pre-training objective and therefore we doubt that HLSQG transfer a limit amount of knowledge from the pre-training task.

We suggest that this approach should be more applicable for training a left-to-right sequence-to-sequence model with an embedding layer for special tokens.

7.3 u-PMLM

According to our best knowledge, we are the first adopting the probabilistic masking scheme to fine-tune BERTs for question generation task. The generation procedure accepts the generation order in both sequential and random. Our results show that generating tokens from left to right is preferred. One possible explanation why left-to-right direction is favorable is mentioned in section 6.3.

The generation ability of u-PMLM is high when the correct number

[1] `https://cs231n.github.io/neural-networks-3#sanitycheck`

of tokens for generation is given. We can figure out it in the training curves as well as the automatic evaluation results.

Pre-determining the length of the generation sequence has its advantage and also disadvantage. The advantage is that the generation results vary with the pre-determined length. If we consider the question generation task as a data augmentation technique for a deep question answering (QA) model, the length of question is definitely valuable for expanding the QA training dataset. When it comes to education purpose, teachers can control the length of the question to fit their actual purposes, like fitting the test paper layout.

The disadvantage is we have to take care of the question length. In this project, a simple and robust statistical approach was proposed to deal with the question token length prediction. However, this approach is far from satisfactory because there is a huge performance gap between sampling token length or using the true one.

As our model is not encoder-decoder architecture, we cannot predict the question length from the encoded contextual information using a simple and thin feed-forward network suggested by Lee, Mansimov, and Cho [15] or a co-training approach proposed by [14].

7.4 Model size

From the automatic evaluation, the base size and the medium size almost behaved the same in our question generation task. Our $\text{BERT}_{\text{base}}$ was pre-trained with the masked language model objective and next sentence prediction according to Devlin et al. [4], which is declared by Turc et al. [62] and the release website[2]. And our $\text{BERT}_{\text{medium}}$, $\text{BERT}_{\text{small}}$, $\text{BERT}_{\text{mini}}$, and $\text{BERT}_{\text{tiny}}$ were pre-trained only with the masked language model objective. [62]

Overall, a larger model performed better than small models when we compare them with their automatic evaluation scores. It is because a model of few layers and parameters would be less expressive. The

[2] https://github.com/google-research/bert

expressiveness decreases along with the depth of the model as well as the number of parameters. [82] We can see the trend from $\text{BERT}_{\text{medium}}$ to $\text{BERT}_{\text{tiny}}$ across all methods and scores.

In practical use, a small model is more favorable than a large one because of the inference speed. If $\text{BERT}_{\text{medium}}$ achieves the same performance as $\text{BERT}_{\text{base}}$, we will take $\text{BERT}_{\text{medium}}$ into our production environment. Therefore, our work shows that for question generation task using SQuAD as training set, $\text{BERT}_{\text{medium}}$ is competent for the task.

7.5 Research Questions

In this section, we discuss and answer the research questions posed in section 1.3.

1. How well does sequence-to-sequence learning work for question generation?

Sequence-to-sequence learning is a feasible method to solve the question generation task described in section 1.1. We conducted the experiments and the case study to show that a masked language models trained under sequence-to-sequence learning framework can generate a human-understandable question.

2. Could we fine-tune a network pre-trained with a language modeling objective for question generation?

We can fine-tune a network pre-trained with the masked language model objective to produce questions. We offered two methods for fine-tuning a masked language model without altering the model architecture. In our experiments, the weights of the models were not randomly initialized but loaded from the pre-trained BERTs.

3. How does the pre-trained network size affect the quality of generated questions?

The quality of generated questions increases with the model size generally. Our automatic evaluation results showed that the more param-

eters a model has, the better the generated question quality is. It is because the evaluation scores reflect the text level similarity between a generated question and the corresponding question posed by a person.

7.6 Societal Impacts

Our topic shall be helpful to education especially language education. It is because the models we have used can write natural, relevant, and answer-specific questions regarding the given passage. Though our experiments suggested that the questions created by the models are not perfect, the model output can be the first draft for further modifications.

7.7 Ethics and sustainability

As our project involves human evaluation and collecting surveys, we try to minimize the personal data we can obtain. In our survey, we do not require our respondents to provide any personal information.

We used the Google Cloud Compute Engine [83] to do our experiments including training and evaluation. Google claims that the net emissions directly associated with the Google Compute Engines including virtual machines and data storage will be zero. Therefore, our project and the related experiments shall be counted as environmentally friendly and sustainable.

7.8 Future work

In this work, we focused on applying the BERT models of various sizes to the question generation task. Yet we can try different approaches and methods in the future.

First, we can consider using the Transformer of the encoder-decoder

framework and combine it with a uniform prior masking probability scheme in pre-training (i.e. MASS). It would be interesting to see how to combine that masking scheme with MASS. In the case of lack of resources, we still can use truncated pre-trained $\text{BERT}_{\text{base}}$ or miniature BERTs as the initialization of the encoder and decoder. As the Transformer model is originally autoregressive, this approach does not need to predict the length of the generated sequence.

Next, we can use the approach of predicting tokens in a non-autoregressive manner. Proposed by Ghazvininejad et al. [14] for machine translation, the method predicts all tokens at the first step and modify them by re-masking and regenerating a portion of resulting tokens in a few steps. This approach is to ensure that the decoding speed is faster than the autoregressive approach.

After that, we need a better approach to evaluate generated questions. Question generation task unlikes machine translation because the length of the generated question is weakly related or even not related to the input context passage. Recently, researchers have started putting generated questions in question answering models to check the generated question quality. They want to have an indirect evaluation of the question generation model through another auxiliary task. However, this approach is not that reliable as the auxiliary task itself is not deterministic. Using different hyper-parameters for training for the question answering models shall affect the evaluation results.

We believe that rule-based methods combining with constituency parsing can give a systematic way to evaluate a question in different aspects.

Finally, question generation is not only limited to the passage-answer pair input. Generating questions to dialogue is also a popular topic in chatbot development. Also, if the question generation can be combined with a full-text search facility or able to input a huge amount of text, we can generate frequently asked questions for a website or exam questions from a textbook.

Chapter 8

Conclusions

In this paper, we studied two methods to train a masked language model for the question generation task in which a context passage and the answer are given. Mainly, we used pre-trained BERT models of various sizes to conduct our study. We replicated the HLSQG approach, namely highlighted answer sequential question generation for the BERT model. Also, we applied a recently developed method u-PMLM to the question generation task. u-PMLM stands for the probabilistically masked language model with a uniform prior masking ratio training scheme. A simple method to predict the output sequence length by the Poisson distribution was used, assuming that the question token length is independent of the context.

The experiment results show that our best approach is HLSQG. In addition, HLSQG is on a level with the LSTM approaches from the previous researches. If we remove the length prediction model and provide the exact number of tokens in the reference, u-PMLM works better than the HLSQG method.

To summarize, both methods are promising to generate questions. But the proposed length prediction model is still the unsolved length prediction in this project. Some methods for the length prediction were discussed, but they cannot work in BERT due to the architecture constrain.

Bibliography

[1] Wikipedia contributors. *Metabolism — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Metabolism&oldid=986521023.

[2] Weiming Wang, T. Hao, and W. Liu. "Automatic Question Generation for Learning Evaluation in Medicine". In: *Advances in Web Based Learning – ICWL 2007* 4823 (2007), pp. 242 –251.

[3] Alec Radford et al. "Improving Language Understanding by Generative Pre-Training". In: 2018.

[4] J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT*. 2019.

[5] K. Song et al. "MASS: Masked Sequence to Sequence Pre-training for Language Generation". In: *ICML*. 2019.

[6] M. Lewis et al. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *ACL*. 2020.

[7] Ying-Hong Chan and Yao-Chung Fan. "A Recurrent BERT-based Model for Question Generation". In: *MRQA@EMNLP*. 2019.

[8] Yi Liao, X. Jiang, and Qun Liu. "Probabilistically Masked Language Model Capable of Autoregressive Generation in Arbitrary Word Order". In: *ACL*. 2020.

[9] Z. Yang et al. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *NeurIPS*. 2019.

[10] Guillaume Lample and Alexis Conneau. "Cross-lingual Language Model Pretraining". In: *ArXiv* abs/1901.07291 (2019).

[11] Chris Donahue, Mina Lee, and Percy Liang. "Enabling Language Models to Fill in the Blanks". In: *ArXiv* abs/2005.05339 (2020).

[12] Hangbo Bao et al. "UniLMv2: Pseudo-Masked Language Models for Unified Language Model Pre-Training". In: *ArXiv* abs/2002.12804 (2020).

[13] W. Fedus, Ian J. Goodfellow, and Andrew M. Dai. "MaskGAN: Better text generation via filling in the _____". In: *ArXiv* abs/1801.07736 (2018).

[14] Marjan Ghazvininejad et al. "Mask-Predict: Parallel Decoding of Conditional Masked Language Models". In: *EMNLP/IJCNLP*. 2019.

[15] Jason Lee, Elman Mansimov, and K. Cho. "Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement". In: *ArXiv* abs/1802.06901 (2018).

[16] Zhenzhong Lan et al. "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020* (2020). URL: https://openreview.net/forum?id=H1eA7AEtvS.

[17] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[18] Kyunghyun Cho et al. "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *SSST@EMNLP*. 2014.

[19] J. Yosinski et al. "How transferable are features in deep neural networks?" In: *NIPS*. 2014.

[20] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *CVPR 2009*. 2009.

[21] M. Noroozi and P. Favaro. "Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles". In: *ECCV*. 2016.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Vol. 1. 2016.

[23] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[24] Lilian Weng. "Attention? Attention!" In: *lilianweng.github.io/lil-log* (2018). URL: http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html.

[25] Ashish Vaswani et al. "Attention is All you Need". In: *ArXiv* abs/1706.03762 (2017).

[26] Dan Hendrycks and Kevin Gimpel. "Gaussian Error Linear Units (GELUs)." In: *arXiv: Learning* (2016).

[27] W. L. Taylor. ""Cloze Procedure": A New Tool for Measuring Readability". In: *Journalism & Mass Communication Quarterly* 30 (1953), pp. 415 –433.

[28] R. Mitkov. "Computer-aided generation of multiple-choice tests". In: *International Conference on Natural Language Processing and Knowledge Engineering, 2003. Proceedings. 2003* (2003), pp. 15–.

[29] Michael Heilman and Noah A. Smith. "Good Question! Statistical Ranking for Question Generation". In: *HLT-NAACL*. 2010.

[30] Michael Heilman and Noah A. Smith. "Question Generation via Overgenerating Transformations and Ranking". In: 2009.

[31] Kaustubh D. Dhole and Christopher D. Manning. "Syn-QG: Syntactic and Shallow Semantic Rules for Question Generation". In: *ArXiv* abs/2004.08694 (2020).

[32] S. W. Brown et al. "VerbNet Representations: Subevent Semantics for Transfer Verbs". In: 2019.

[33] Patrick Lewis, Ludovic Denoyer, and S. Riedel. "Unsupervised Question Answering by Cloze Translation". In: *ACL*. 2019.

[34] A. R. Fabbri et al. "Template-Based Question Generation from Retrieved Sentences for Improved Unsupervised Question Answering". In: *ACL*. 2020.

[35] Alexander M. Rush, S. Chopra, and J. Weston. "A Neural Attention Model for Abstractive Sentence Summarization". In: *ArXiv* abs/1509.00685 (2015).

[36] X. Du, Junru Shao, and Claire Cardie. "Learning to Ask: Neural Question Generation for Reading Comprehension". In: *ACL*. 2017.

[37] Qingyu Zhou et al. "Neural Question Generation from Text: A Preliminary Study". In: *NLPCC*. 2017.

[38] Iulian Vlad Serban et al. “Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus”. In: *ArXiv* abs/1603.06807 (2016).

[39] Thang Luong, Hieu Pham, and Christopher D. Manning. “Effective Approaches to Attention-based Neural Machine Translation”. In: *ArXiv* abs/1508.04025 (2015).

[40] Caglar Gulcehre et al. “Pointing the Unknown Words”. In: (Aug. 2016), pp. 140–149. DOI: 10.18653/v1/P16-1014.

[41] Linfeng Song et al. “Leveraging Context Information for Natural Question Generation”. In: *NAACL-HLT*. 2018.

[42] Z. Wang, W. Hamza, and Radu Florian. “Bilateral Multi-Perspective Matching for Natural Language Sentences”. In: *IJCAI*. 2017.

[43] Duyu Tang et al. “Question Answering and Question Generation as Dual Tasks”. In: *ArXiv* abs/1706.02027 (2017).

[44] Yingce Xia et al. “Dual Supervised Learning”. In: *ArXiv* abs/1707.00415 (2017).

[45] Ian J. Goodfellow et al. “Generative Adversarial Nets”. In: *NIPS*. 2014.

[46] Xingdi Yuan et al. “Machine Comprehension by Text-to-Text Neural Question Generation”. In: *Rep4NLP@ACL*. 2017.

[47] Marc’Aurelio Ranzato et al. “Sequence Level Training with Recurrent Neural Networks”. In: *CoRR* abs/1511.06732 (2016).

[48] Yanghoon Kim et al. “Improving neural question generation using answer separation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (2019), pp. 6602–6609. DOI: 10.1609/aaai.v33i01.33016602. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4629.

[49] Reinald Kim Amplayo, Seonjae Lim, and S. Hwang. “Entity Commonsense Representation for Neural Abstractive Summarization”. In: (2018), pp. 697–707. DOI: 10.18653/v1/N18-1064.

[50] A. Radford et al. “Language Models are Unsupervised Multitask Learners”. In: 2019.

[51] Li Dong et al. “Unified Language Model Pre-training for Natural Language Understanding and Generation”. In: *NeurIPS*. 2019.

[52] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9 (1997), pp. 1735–1780.

[53] Anirudh Goyal et al. "Professor Forcing: A New Algorithm for Training Recurrent Networks". In: *ArXiv* abs/1610.09038 (2016).

[54] S. Bengio et al. "Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks". In: *ArXiv* abs/1506.03099 (2015).

[55] Wen Zhang et al. "Bridging the Gap between Training and Inference for Neural Machine Translation". In: *ArXiv* abs/1906.02448 (2019).

[56] Inc. Hugging Face. *How to generate text: using different decoding methods for language generation with Transformers*. Accessed: 2020-12-14. 2018. URL: https://github.com/huggingface/blog/blob/master/notebooks/02_how_to_generate.ipynb.

[57] Romain Paulus, Caiming Xiong, and R. Socher. "A Deep Reinforced Model for Abstractive Summarization". In: *ArXiv* abs/1705.04304 (2018).

[58] Pranav Rajpurkar et al. "SQuAD: 100, 000+ Questions for Machine Comprehension of Text". In: *ArXiv* abs/1606.05250 (2016).

[59] Peng Qi et al. "Stanza: A Python Natural Language Processing Toolkit for Many Human Languages". In: *ACL*. 2020.

[60] Christopher D. Manning et al. "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Association for Computational Linguistics (ACL) System Demonstrations*. 2014, pp. 55–60.

[61] Pranav Rajpurkar. *The Stanford Question Answering Dataset 2.0 Home Page*. URL: https://rajpurkar.github.io/SQuAD-explorer/.

[62] Iulia Turc et al. "Well-Read Students Learn Better: On the Importance of Pre-training Compact Models". In: *arXiv: Computation and Language* (2019).

[63] Mike Schuster and Kaisuke Nakajima. "Japanese and Korean voice search". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2012), pp. 5149–5152.

[64] Y. Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *ArXiv* abs/1609.08144 (2016).

[65] Thomas Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *ArXiv* abs/1910.03771 (2019).

[66] P. Micikevicius et al. "Mixed Precision Training". In: *ArXiv* abs/1710.03740 (2018).

[67] Nvidia. *Training With Mixed Precision :: NVIDIA Deep Learning Performance Documentation*. URL: https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html.

[68] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2015).

[69] Shikhar Sharma et al. "Relevance of Unsupervised Metrics in Task-Oriented Dialogue for Evaluating Natural Language Generation". In: *ArXiv* abs/1706.09799 (2017).

[70] Preksha Nema and Mitesh M. Khapra. "Towards a Better Metric for Evaluating Question Generation Systems". In: *ArXiv* abs/1808.10192 (2018).

[71] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *ACL*. 2002.

[72] S. Banerjee and A. Lavie. "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments". In: *IEEvaluation@ACL*. 2005.

[73] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *ACL 2004*. 2004.

[74] Ananthakrishnan et al. *Some Issues in Automatic Evaluation of English-Hindi MT : More Blues for BLEU*. 2006.

[75] Chris Callison-Burch. "Fast, Cheap, and Creative: Evaluating Translation Quality Using Amazon's Mechanical Turk". In: *EMNLP*. 2009.

[76] Chris Callison-Burch, M. Osborne, and Philipp Koehn. "Re-evaluation the Role of Bleu in Machine Translation Research". In: *EACL*. 2006.

[77] Ehud Reiter. "A Structured Review of the Validity of BLEU". In: *Computational Linguistics* 44.3 (2018), pp. 393–401.

[78] Ehud Reiter. *How Would I Automatically Evaluate NLG Systems?* Accessed: 2020-10-21. 2018. URL: https://ehudreiter.com/2018/07/25/how-would-i-automatically-evaluate-nlg/.

[79] K. Krippendorff. "Validity in Content Analysis". In: *Computerstrategien für die Kommunikationsanalyse*. Frankfurt, Germany: Campus: Mochmann, Ekkehard, 1980, pp. 69–112. URL: http://repository.upenn.edu/asc_papers/291.

[80] J. Fleiss. "Measuring nominal scale agreement among many raters." In: *Psychological Bulletin* 76 (1971), pp. 378–382.

[81] Y. Zhao et al. "Paragraph-level Neural Question Generation with Maxout Pointer and Gated Self-attention Networks". In: *EMNLP*. 2018.

[82] Yoshua Bengio, Aaron C. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (2013), pp. 1798–1828.

[83] Google. *Google Cloud Environment | Go Green - Google Cloud*. Accessed: 2020-10-30. URL: https://cloud.google.com/sustainability.

Appendix A

VM specification

This project used a Google Cloud compute instance to train and evaluate our models. We used the command `gcloud compute instances create` from Google Cloud SDK to create the instance.

The compute instance configurations are:

- CPU: Intel(R) Xeon(R) CPU @ 2.00GHz $\times$2
- RAM: 7.5 GB
- Disk size: 200 GB
- GPU: NVIDIA Tesla T4 GPU with 16 GB VRAM $\times$1

The argument names and values for the virtual machine creation command are listed in table A.1.

Argument Name	Argument Value
accelerator	type=nvidia-tesla-t4,count=1
image	pytorch-1-4-cu100-20200128
image-project	deeplearning-platform-release
machine-type	n1-standard-2

Table A.1: The Command argument names and values for virtual machine creation

Appendix B

BLEU scores

The scores from BLEU-1 to BLEU-4 of all our models are shown in table B.1.

Method	Size	Gen. Order	Token Len.	BLEU-1	BLEU-2	BLEU-3	BLEU-4
HLSQG	base	sequential	N/A	39.15	27.8	20.75	15.89
HLSQG	medium	sequential	N/A	38.03	26.75	19.8	15.04
HLSQG	small	sequential	N/A	37.13	25.82	18.96	14.29
HLSQG	mini	sequential	N/A	34.44	23.15	16.53	12.14
HLSQG	tiny	sequential	N/A	22.78	12.22	7.15	4.33
u-PMLM	base	sequential	true	43.02	30.41	22.6	17.28
u-PMLM	base	sequential	sample	37.26	25.32	17.93	12.95
u-PMLM	base	random	true	42.09	29.08	21.08	15.68
u-PMLM	base	random	sample	36.55	24.31	16.78	11.7
u-PMLM	medium	sequential	true	42.59	30.28	22.69	17.49
u-PMLM	medium	sequential	sample	36.94	25.24	18.08	13.23
u-PMLM	medium	random	true	41.31	28.48	20.65	15.35
u-PMLM	medium	random	sample	35.66	23.67	16.34	11.44
u-PMLM	small	sequential	true	40.64	28.24	20.81	15.8
u-PMLM	small	sequential	sample	35.68	24.11	17.13	12.39
u-PMLM	small	random	true	38.93	25.9	18.06	12.87
u-PMLM	small	random	sample	34.44	22.25	14.91	10.09
u-PMLM	mini	sequential	true	38.3	26.16	18.96	14.15
u-PMLM	mini	sequential	sample	33.9	22.55	15.76	11.25
u-PMLM	mini	random	true	36.32	23.02	15.24	10.29
u-PMLM	mini	random	sample	32.51	20.19	12.95	8.43
u-PMLM	tiny	sequential	true	24.65	13.1	7.51	4.4
u-PMLM	tiny	sequential	sample	23.25	12.37	7.13	4.22
u-PMLM	tiny	random	true	23.63	11.24	5.46	2.68
u-PMLM	tiny	random	sample	22.74	10.79	5.23	2.59

Table B.1: Automatic Evaluation results for all model

TRITA -EECS-EX-2020:902