System Report

Of

Group Project

(COMP5423)

CHONG Kit Sang (19005168g)

CHOW Man Kit ()

CHU Kin Tung (20006316G)

Ng Kai Pong (18002565G)

YAU Pui Hang (20070958G)

Table of Content

Introduction 3

Design of the System 4

Function of the System 4

System Architecture 6

Suite of Tools 6

Analytics on Proposed Data 7

MCTest 7

RACE 7

DREAM 7

Peer And Self-Assessment 8

Member Contribution 8

# Introduction

Automated reading comprehension can be applied to many applications. In general, users may require training well to operate and enquire information from typical systems. Imagine that users able to enquire the information by simple sentence(s) instead of well-designed user interfaces and no of compulsory attributes. Such as technical support and troubleshooting, customer service, and the understanding of healthcare records, etc.

This project target to create a Question Answering Machine Learning model system which will take comprehension and questions as input and process the multiple-choice reading comprehension on MCTest, RACE and Dream dataset. Compared to MCTest, the answers of RACE and Dream could not directly be extracted from the passage. Solving the question needs more complicate reasoning and challenges foresee as follow:

* Manipulate the input data and adapting relevant libraries and corresponding data model.
* To handle various domains and writing styles.
* Size of training data required corresponding computing power and time consuming.

# Design of the System

## Function of the System

The QA system composed of three major components. The modules for user to answering 3 types of multiple-choice questions.

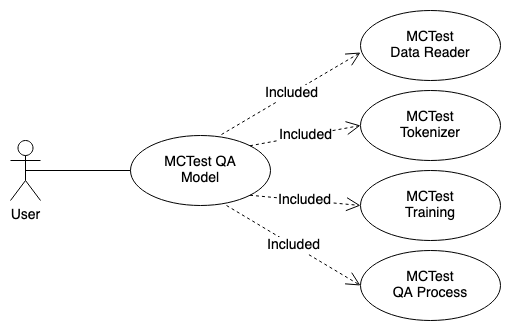


Figure 1 - MCTest Data QA module

Diagram

Description automatically generated

Figure 2 - RACE Data QA module

Diagram

Description automatically generated

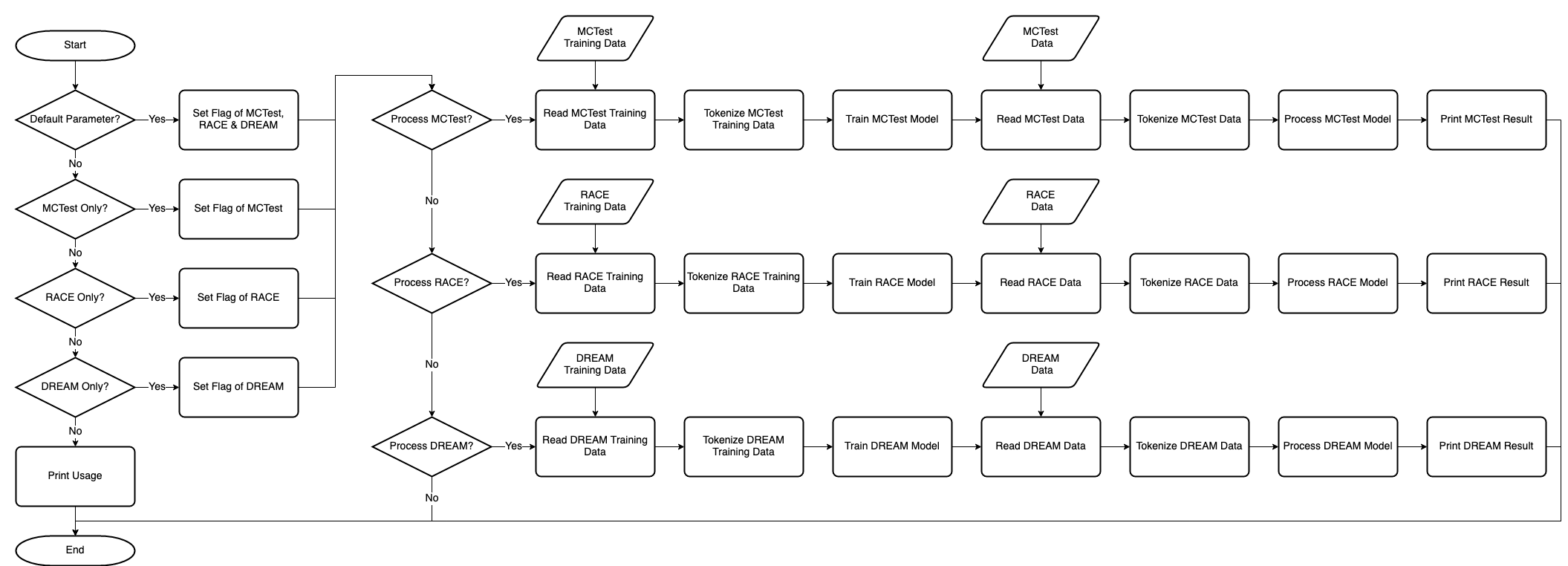
Figure 3 – Dream Data QA module

The design of the system with the mapping of use cases as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Matching User Case | Function ID | Description | Actor |
| MCTest QA Model | MC-001 | The main module of MCTest QA Model | User |
| MCTest Data Reader | MC-002 | The MCTest data reader. The data would be stored in structured data format. | User |
| MCTest Tokenizer | MC-003 | HuggingFace tokenizer was used and modelling the input data. | User |
| MCTest Training | MC-004 | Train the MCTest QA process and reporting the accuracy of trained model. | User |
| MCTest QA Process | MC-005 | Process the MC questions and reporting the result. | User |
| RACE QA Model | RA-001 | The main module of RACE QA Model | User |
| RACE Data Reader | RA-002 | The RACE data reader. The data would be stored in structured data format. | User |
| RACE Tokenizer | RA-003 | BertTokenizer was used and modelling the input data. | User |
| RACE Training | RA-004 | Train the RACE QA process and reporting the accuracy of trained model. | User |
| RACE QA Process | RA-005 | Process the MC questions and reporting the result. | User |
| DREAM QA Model | DR-001 | The main module of RACE QA Model | User |
| DREAM Data Reader | DR -002 | The DREAM data reader. The data would be stored in structured data format. | User |
| DREAM Tokenizer | DR -003 | BertTokenizer was used and modelling the input data. | User |
| DREAM Training | DR -004 | Train the DREAM QA process and reporting the accuracy of trained model. | User |
| DREAM QA Process | DR -005 | Process the MC questions and reporting the result. | User |

## Program Flow of the System

The following diagram illustrates the program flow of the system.



## System Architecture

The following diagram illustrates the architecture of the system.

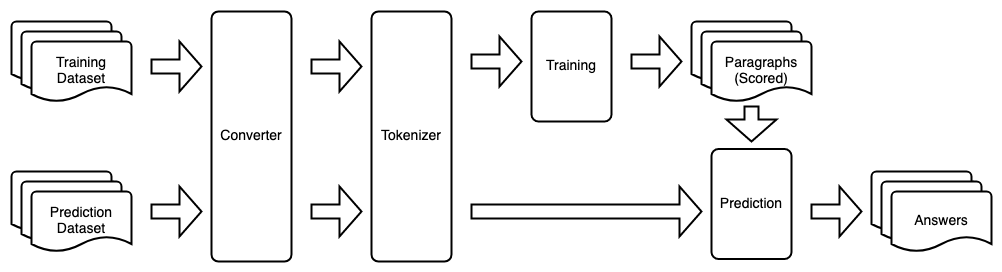


Figure 4 - System Architecture

## Suite of Tools

The following tools will be adopted as the components in development.

1. Python v3.8.8
2. IPython7.18.1

Usage of open-source Library/Framework.

1. PyTorch
2. DistilBERT
3. Pytorch\_pretrained\_bert
4. Numpy
5. NLTK
6. Logging
7. os
8. Torch
9. Tensorflow
10. Tqdm
11. Pandas
12. Json
13. String
14. Math
15. random
16. gzip

# Analytics on Proposed Data

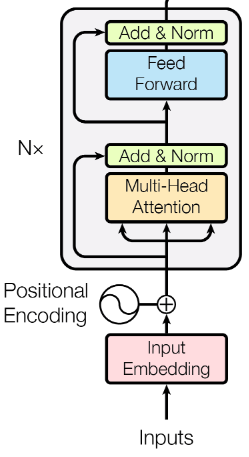
## MCTest

## RACE

To solve the RACE Multiple Choice reading comprehension question, we adopt the BERT based pretrained model to learn from RACE training Dataset. We used the BERT pretrained model with fine tune approach apply at Pytorch machine learning library to train the model to answer the multiple question for the RACE dataset.

BERT become one of the most popular one due to its superior performance on many NLP task which it leveraged the bidirectional information of a sequence used the information to the full potentially.

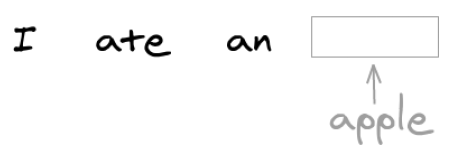
The Architecture of BERT is the same as the encoder of a transformer network which mainly consists of a series of self-attention layer (12 in case of the base model and 24 in the large model) combined with layer normalization and residual layers.

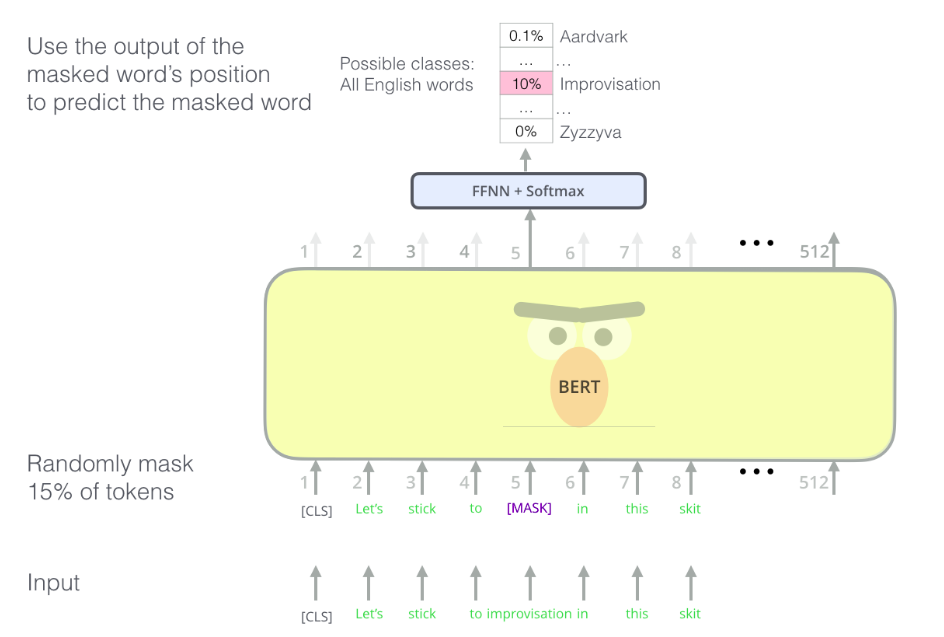


BERT is trained with two unsupervised learning tasks intuitively designed to pre-train words on the large corpora and sentence representation, which is the core of the BERT algorithm. There are 2 types of Tasks can work in BERT.

Task 1: Masked language model (MLM)

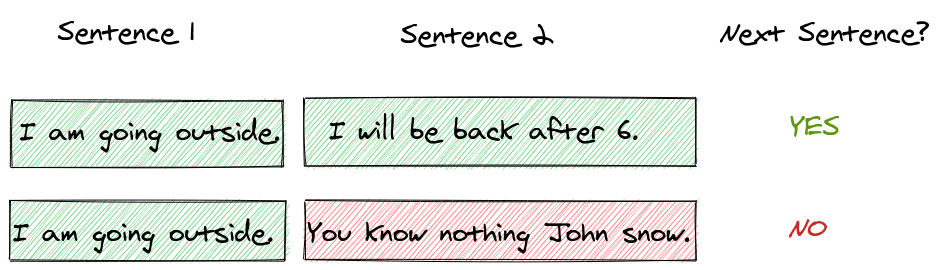
The task is based on most probabilistic language model to predict the target word according to both left and right words. The training just needs to mask a certain proportion of the words randomly in a sentence and then use those masked words as the predicting targets instead of predicating next word. The final layer is a SoftMax layer to convert hidden representation to normalized probability distribution over the vocabulary.

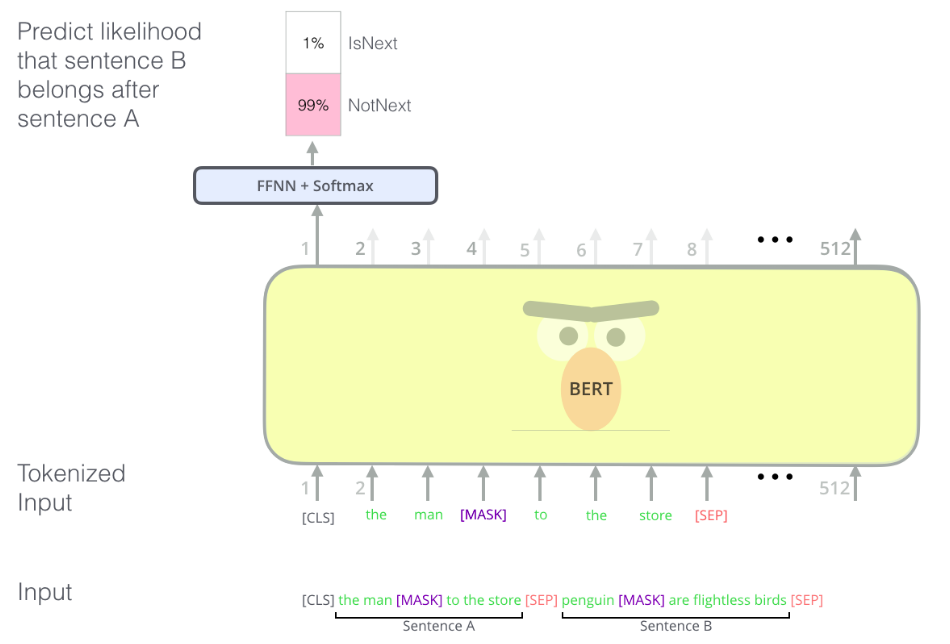




Task 2: Next Sentence Predication (NSP)

NLP task based on understanding the relationship between two sentences, but this information cannot be directly captured by previous pertaining algorithm. This task required to capture the sentence level information which objective is to generate a vector used for classify the sentence. After that, the language model can object word-level representation by maximizing the likelihood estimation of a sequence of words, and then it computes the similar likelihood estimation of the sequence of sentences.





For instance, given 2 sentences are separated by a [SEP] token and noticed the [CLS] token in figure, it is BERT required added to the beginning of each sequence (or pair of sequences) and is given the responsibility of sequence level classification. We used the embeddings of [CLS] token to predict if sentence A comes before sentence B in the next sentence prediction task.

After BERT is trained on these 2 tasks, the learned model can be then used as for different NLP problems (such as Multiple-choice Question Answering), where we can either keeping the learned weights fixed for another task or learn the new added task-specific layers or fine-tune the pre-trained layer depending on the NLP applications need.

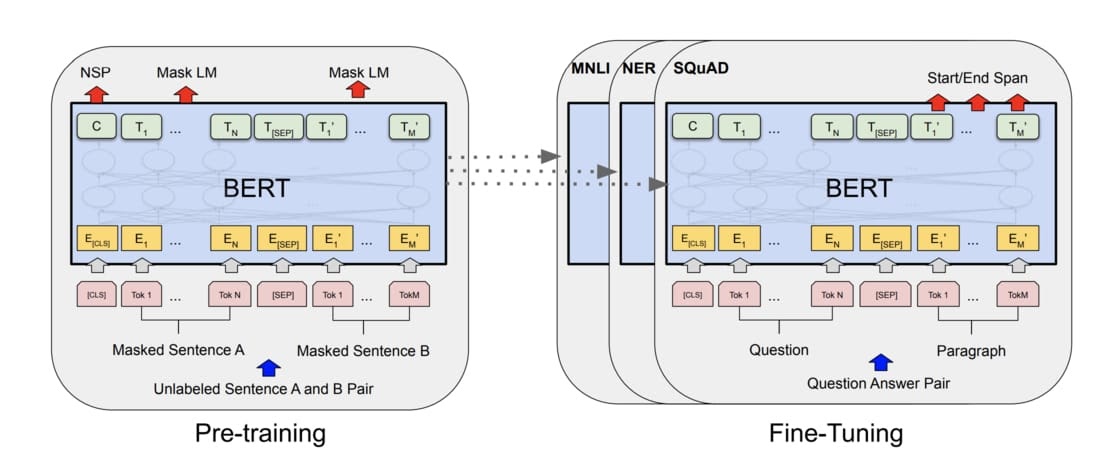
3rd party Transformer module library

In this project, we intend to answer the multiple Questions in RACE datasets. it used pytorch-pretained-bert library to train the RACE.

pip install pytorch-pretrained-bert

The library provided a lot of BERT APIs suitable for different tasks like BertForSequenceClassifciation, BertForMultipleChoice , etc. We selected the BertForMultipleChoice model in this project to solve RACE multiple choice questions. To initial BERT with pretrained model which we called the BERT “from\_pretrained” method to create instance of BERT with preloaded weight and also giving “bert-base-uncased" parameter for input, it returns the base model (with 12-layer, 768-hidden, 12-heads, 110M parameters) pre-trained on uncased sequence which model is a subclass of Pytorch’s nn.Module and hence can be used just like any other Pytorch module.

Fune-Tune Model for question answer Pair in BERT

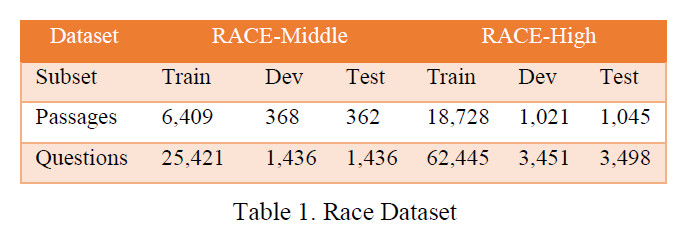


We used the BERT pretrained model with fine tune approach apply at Pytorch machine learning library to train the model to answer the multiple question for the RACE dataset to generate the Question Answer Pair for solve the RACE multiple choice question answering.

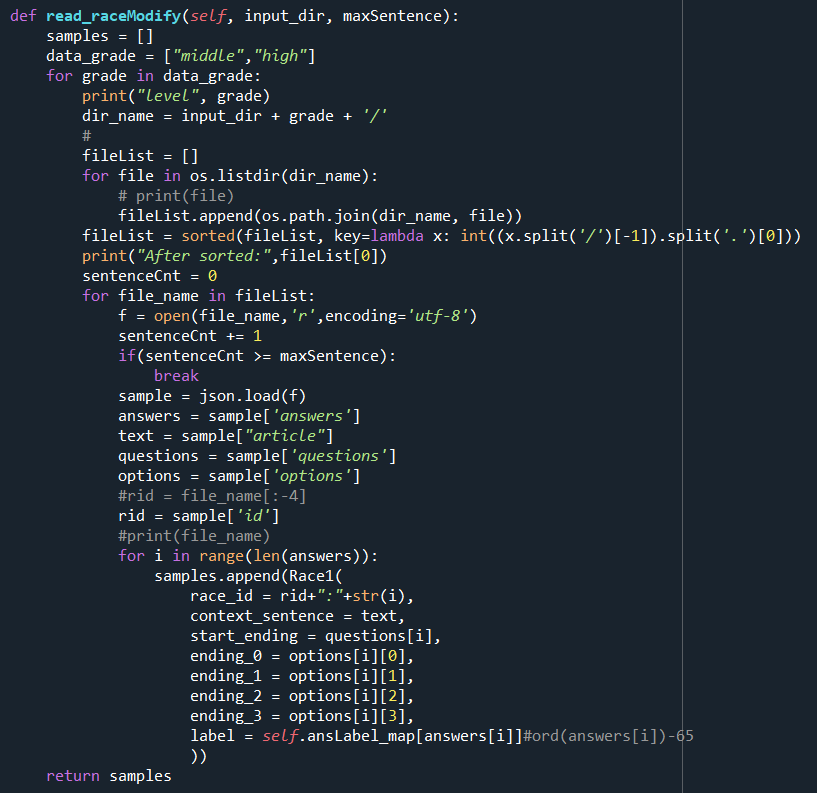
RACE Data Preprocessing

The RACE contains 2 levels of difficult (middle and high) dataset and also its categories into 3 group of questions for different purposes used in this project. Training dataset for train the BERT model, Dev dataset for evaluate the trained BERT model performance. Test dataset for general purpose test the trained BERT multiple choice question answering abilities.

The total number of passages and questions are 27,933 and 97,687 respectively. Middle dataset averages about 250 words per passage while the High dataset averages 350 words per passage.

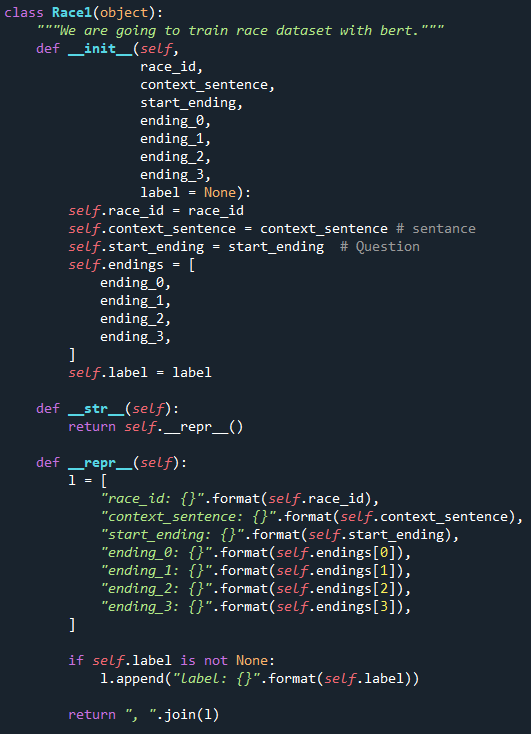


The RACE datasets are very large and also it stores into a lot of small text files (each sentence for one file) respective. In this project, we design the program to search the file list in each datasets directory and extract the data from file. The data format in the file is based on JSON format (Key-value pair), we just used the key words like (id, answer, article, question, option) to extract specific type of data.



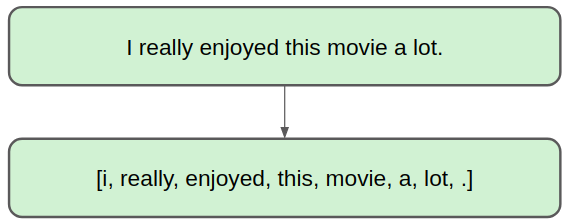
RACE data model

After extract the dataset from text file, we create the RACE data model class to handle the dataset which includes race\_id (file name), context\_sentence, start\_ending(Question), ending 0 (option 0), ending\_1 (option 1), ending\_2 (option 2), ending\_3(option 3), label (answer index). Use the Race data class model easily to extract the data feature from class model.



Tokenize the sequence.

The Tokenize a sequence meaning taking a sentence and converting it to a list of word appearing in the sequence.

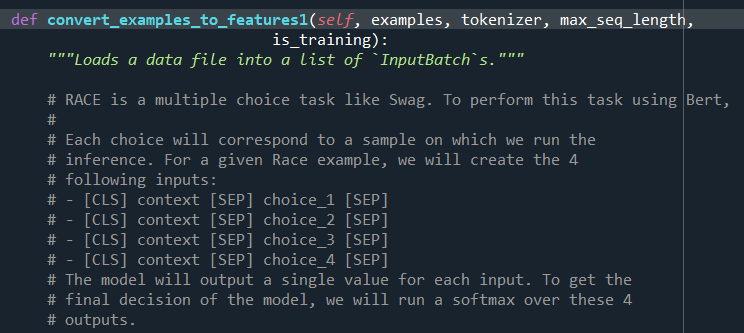


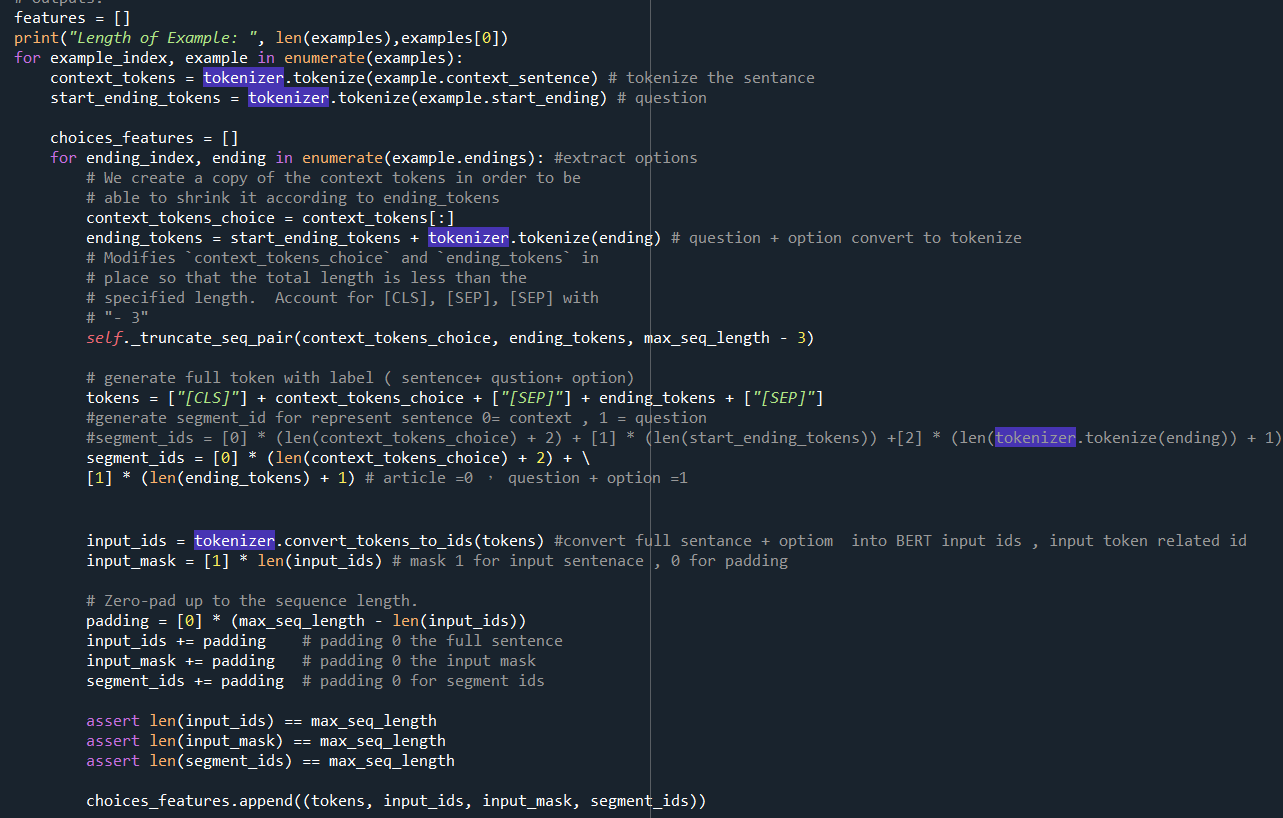
We can use the BertTokenizer class in the transformers module to do this which it ensures that the input sentence don’t get any out of vocabulary tokens (e.g., an unknow word “kabir” will be split into bytes “ka###” and “##bir”, which are present in BERT’s vocabulary).

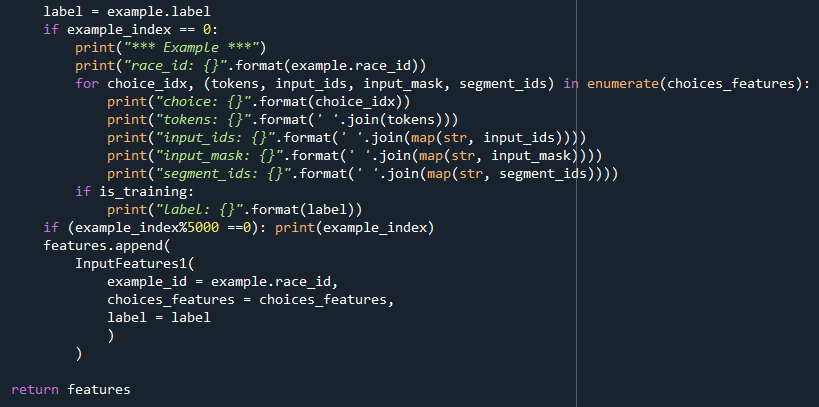
Feature Engineering

Before feed the input data input BERT for training, we create the Input Feature object model to store the feature extracted from feature convert function which converted the Race Data model sample data into the BERT recognize format.

Convert Race Sample Data into feature extraction.







We need to concatenate the sentences, question and option together with special token CLS and SEP as the input sequence for BERT model.

Example : Add [CLS] and [SEP] token for RACE data model

Input: [CLS] Sentence [SEP] question + option0[SEP]

[CLS] Sentence [SEP] question + option1[SEP]

[CLS] Sentence [SEP] question + option2[SEP]

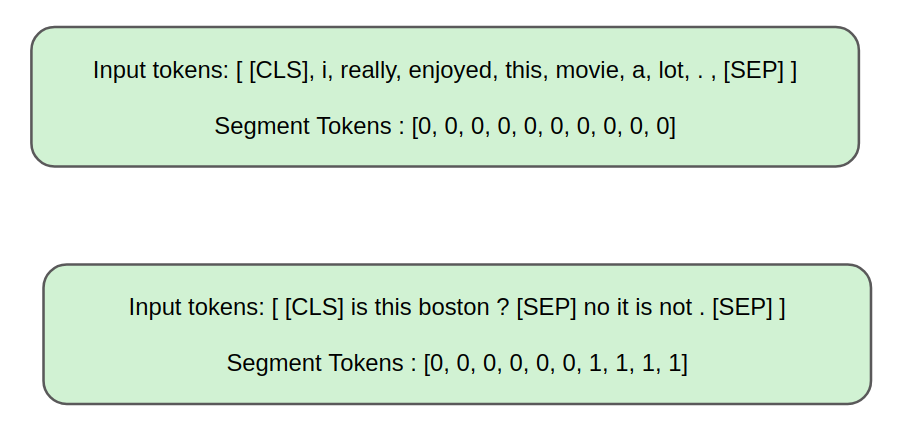
[CLS] Sentence [SEP] question + option3[SEP]

Padding zero the input

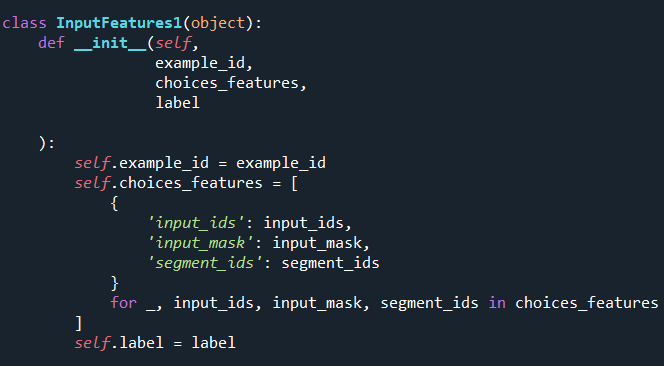
During training, we would like to feed some let's say 128 different sequences together to BERT model which is easily possible has 128 different length of sequence feed into a single tensor. But the tensor (pytorch) required of all of data length must same size. In order to fix the maximum length, we need to pad all the sequence by 0 for generate all sequences with same data length. In RACE datasets, the average words length in middle (difficult) is 250 words and high (difficult) averages is 350 words, So, we set the maximum sequence length is 450 words.

List of Segments tokens

These segment tokens are used to indicate whether a particular tokens in the input belongs to the first portion of the sentence (label 0) or to second portion of the input (labeled 1).



After the feature extraction from RACE data model, the data feature will store in inputFeature Class which have 4 types of data features categorize. The “input\_ids” is store the full sentence (sentence+ question+ option) token . The “input\_mask” is represent the padding input position( 0 = padding input, 1 = normal data input). The “segment\_ids” is represent the difference sentence position in input (0 = Sentence A, 1 = Sentence B) . Label is represented answer index for question.



Example for Feature extraction feed into BERT

\*\*\* Example \*\*\*

race\_id: middle13.txt:0

choice: 0

tokens: [CLS] what is color? why do some of the things around us look red, some green, others blue? colors are really made by def ##le ##cted light. we see color because most of the things reflect light. in the same way, if something is green, it reflects most of the green light. if something reflects all light, it is white. if it doesn't reflect any light, it is black. some of the light is reflected and some is taken in and turned into heat. the darker the color is, the less light is reflected, the more light is taken in. So dark - colored clothes are warmer in the sun than light - colored clothes. [SEP] when something reflects light, we can see its color [SEP]

input\_ids: 101 2054 2003 3609 1029 2339 2079 2070 1997 1996 2477 2105 2149 2298 2417 1010 2070 2665 1010 2500 2630 1029 6087 2024 2428 2081 2011 13366 2571 10985 2422 1012 2057 2156 3609 2138 2087 1997 1996 2477 8339 2422 1012 1999 1996 2168 2126 1010 2065 2242 2003 2665 1010 2009 11138 2087 1997 1996 2665 2422 1012 2065 2242 11138 2035 2422 1010 2009 2003 2317 1012 2065 2009 2987 1005 1056 8339 2151 2422 1010 2009 2003 2304 1012 2070 1997 1996 2422 2003 7686 1998 2070 2003 2579 1999 1998 2357 2046 3684 1012 1996 9904 1996 3609 2003 1010 1996 2625 2422 2003 7686 1010 1996 2062 2422 2003 2579 1999 1012 2061 2601 1011 6910 4253 2024 16676 1999 1996 3103 2084 2422 1011 6910 4253 1012 102 2043 2242 11138 2422 1010 2057 2064 1035 1012 2156 2049 3609 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

input\_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

segment\_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

choice: 1

tokens: [CLS] what is color? why do some of the things around us look red, some green, others blue? colors are really made by def ##le ##cted light. we see color because most of the things reflect light in the same way, if something is green, it reflects most of the green light. if something reflects all light, it is white. if it doesn't reflect any light, it is black. some of the light is reflected and some is taken in and turned into heat. the darker the color is, the less light is reflected, the more light is taken in. So dark - colored clothes are warmer in the sun than light - colored clothes. [SEP] when something reflects light, we can see its heat [SEP]

input\_ids: 101 2054 2003 3609 1029 2339 2079 2070 1997 1996 2477 2105 2149 2298 2417 1010 2070 2665 1010 2500 2630 1029 6087 2024 2428 2081 2011 13366 2571 10985 2422 1012 2057 2156 3609 2138 2087 1997 1996 2477 8339 2422 1012 1999 1996 2168 2126 1010 2065 2242 2003 2665 1010 2009 11138 2087 1997 1996 2665 2422 1012 2065 2242 11138 2035 2422 1010 2009 2003 2317 1012 2065 2009 2987 1005 1056 8339 2151 2422 1010 2009 2003 2304 1012 2070 1997 1996 2422 2003 7686 1998 2070 2003 2579 1999 1998 2357 2046 3684 1012 1996 9904 1996 3609 2003 1010 1996 2625 2422 2003 7686 1010 1996 2062 2422 2003 2579 1999 1012 2061 2601 1011 6910 4253 2024 16676 1999 1996 3103 2084 2422 1011 6910 4253 1012 102 2043 2242 11138 2422 1010 2057 2064 1035 1012 2156 2049 3684 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

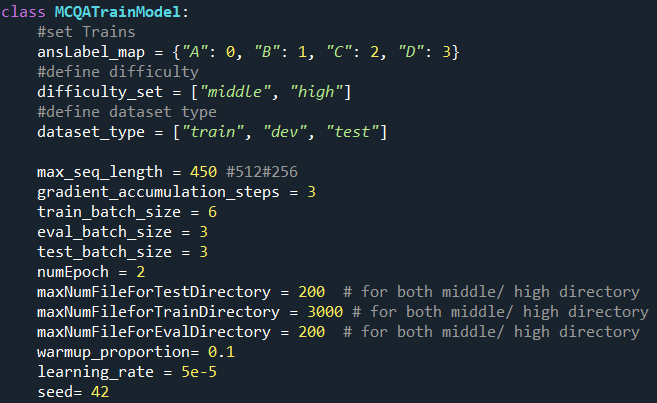
input\_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

segment\_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Fine Tune the BERT Model

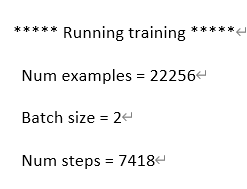
There are several parameters can used to fine the BERT Train performance. The max\_seq\_length parameter is defined the maximum length of feature input to BERT. In initial development stage, it selects 256 for training. We found the accurate is low around 33%, and then adjust the value 450 and 512 intend to encode more feature (average sentence length of middle is 250 words, high is 350 words) from dataset where we can see the accurate is increase several percentages (around 38.6%). However, our training machine GPU (GTX1060 6GB RAM) resource limitation, we can’t put more length for each sequence (will out of memory), finally we selected 450 for suitable run at the develop environment.

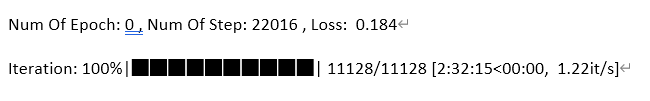
The second critical parameter is MaxNumFileForTrainDirectory. Because the dataset large, we cannot put all the sentence for train at limited resource hardware. In initial development stage, we set the maxTrainfileForDirectory is 200 (obtain 200 files from middle and 200 files from high). The Training time is dependent on the number of train files, for total 400 files we need spend around 45 minutes to complete 2 epoch training, the result of accurate only around 38.6% (for max\_seq\_length = 450). Then we increase the training file size to 300 (totals 600), the result of accurate has increased to 40.8%. After that, we found the trend increase the training file sample size can increase the accuracy performance. Then we try to train 10 times larger training file size (2000 = totals 4000 files) which need training time around 2 hours, and then we obtain accurate around 53.6% that is a significant improvement. Then we try to set training file size (3000 = totals 6000 files) which training time around 5 hours (over double times of 4000 files), and then we obtain accurate around 54.4% that is less than 1% improvement, but we need long time to train the model. So, we use 6000 files from training dataset for final setting.

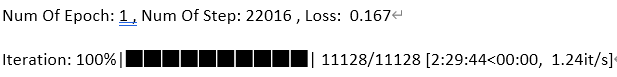


### Measuring Performance for RACE

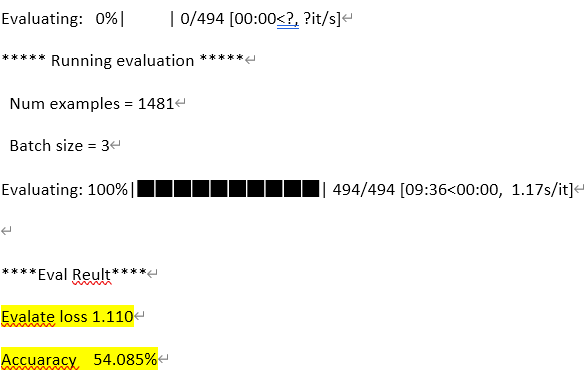
Train Data: 22256, Duration: 5:02:00, Epoch =2 times



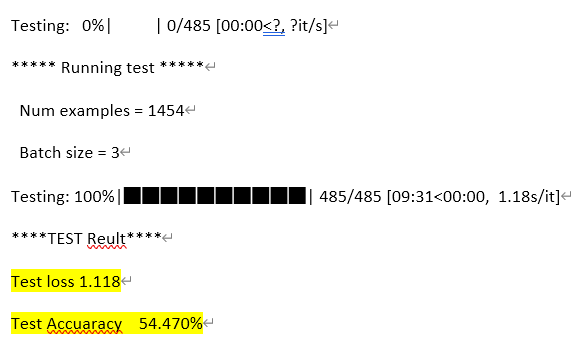




Dev Data:  1481, Duration: 00:09:36, Accuracy: 54.085%

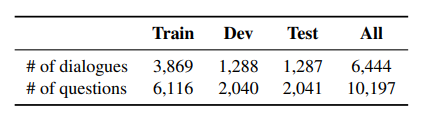


Test Data: 1454, Duration: 00:09:31, Accuracy: 54.47%

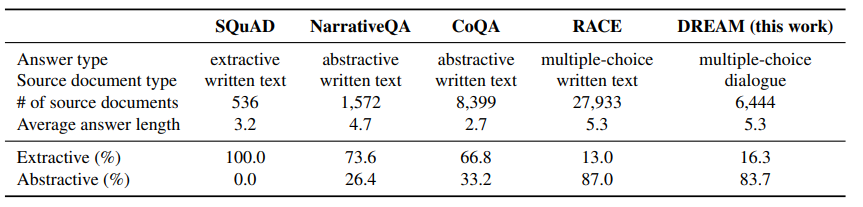


## DREAM

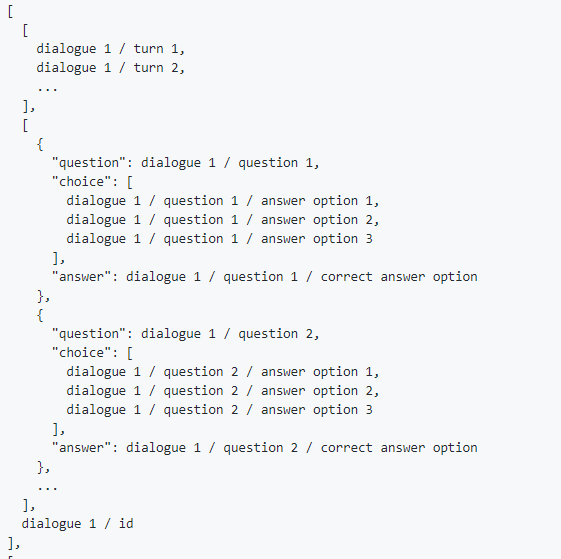
Dream is the first multiple-choice Dialogue-based Reading comprehension dataset with multiple-choice questions. In Dream, there is a total of 10197 questions for 6444 dialogues that are involved in the train, dev, and test dataset separately, those questions aim to focus on the understanding of two-way and multi-person dialogues.



Compare with other datasets, Dream also provides a lot of challenges for developing the document-based question answering system, as most of the answers in Dream cannot be extracted from the dialogues paragraph directly, a large number of questions involved multiple sentence inference. Solving these questions may require some reasoning skills or even commonsense knowledge.



For the data in Dream, they are stored in JSON format, each Dialogue at least has two conversation sentences and may contain more than one question for the dialogue. For each question, they are associated with exactly three answer options and only one of them is the correct answer. You can see the example below.

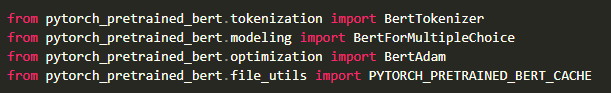


Dream - without Commonsense

Dream - With Commonsense

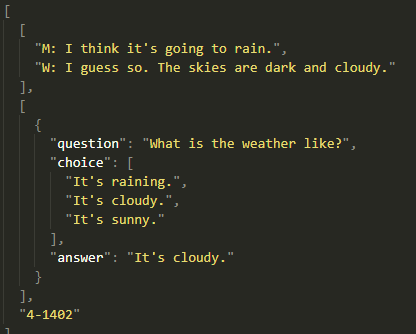
**Pretrained Model**

Dream has a similar characteristic to RACE as most of the answers are not extractive and it requires the model to learn the relationship between sentences. Therefore, we can simply adopt the same pre-trained model in the RACE, which is the Pytorch\_pretrained\_bert. In addition, we used the BertForMultipleChoice, it is for multiple choice classification and contains input\_ids that Indices of input sequence tokens in the vocabulary, and labels for computing the multiple-choice classification loss.

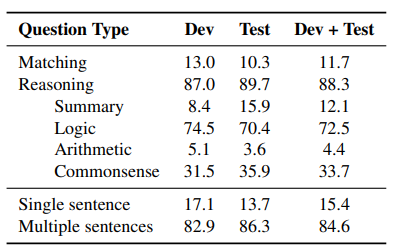


**Exploratory Data Analysis**

The question in Dream can be divided into two categories. First is the Matching, the answer to these questions can be easily verified from the sentence. For example, the question is “*What is the weather like?*” and the answer is “*It's cloudy.*”, it can be extracted from the sentence “*W: I guess so. The skies are dark and cloudy.*”



Second category is Reasoning, the answer to these questions cannot be verified by the surface meaning of dialogue sentences, and there are also four types of questions in this category. The first type of questions is required to summarize the whole picture of the dialogue, it may ask the questions like “*What was the probable relationship between the two speakers?*” and "*Where are the two speakers?*", etc. The second type are the Arithmetic questions, one example is the dialogue sentences mentioned about the price, the question is "*How much will the woman pay in total*?", and the answer just sums up the price. The third type are the logic questions, most questions in Dream belong to this type, it requires the logical reasoning to solve the question. For example, the man said “*my wife strongly suggested that we go to the airport right after we finish our work this afternoon*”, and the question is “*How will the man go to his wife’s parents’ home?*”, we can inference that the answer is “*By plane*”. The last type of the questions is the commonsense, for answering these questions, it requires additional commonsense knowledge that do not mention in the dialogue sentence. The example is a woman said, “*tomorrow is Christmas Day*” and a man replied that two days later is his birthday. In this case, we should know the Christmas Day is on December 25, then we can inference the birthday of the man is December 26.



**Data Preparation**

1. **Obtain predictions from commonsese\_prediction.py**

We first obtain the predictions of train data by the commonsese\_prediction.py. This python file is based on the speaker-focused distance-based score to make the prediction. For more details, you can view the paper “DREAM: A Challenge Dataset and Models for Dialogue-Based Reading Comprehension.” In commonsese\_prediction.py, it imports the get\_vector() function from commonsense\_utilities.py, the get\_vector() function can help to create a token vector that can be used for comparing word meanings numerically.Before the function converts the required data into a token vector, it tokenized the raw data with several functions.

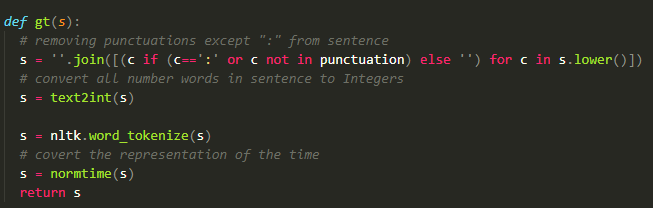
The text2int() function, it converts the number words to integers.



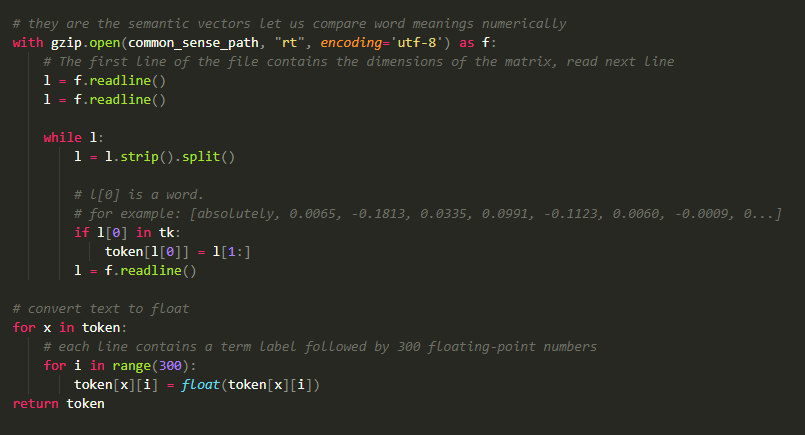
The normtime() function normalize the time in dialogue sentences.



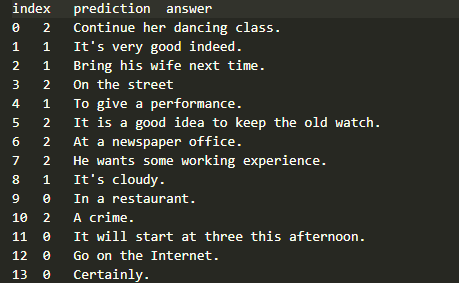
The gt() function first remove the punctuation from sentences then execute the previous function and uses NLTK to tokenize the words.



In get\_vector() function, there is a path that includes a ConceptNet Numberbatch file, it is a set of semantic vectors that can be used directly as a representation of word meanings. It is used to add the semantic value to the tokenized words, thereby we can compare the word meaning by calculating the cosine similarity and then find the relationship between the sentences, question and answer options.

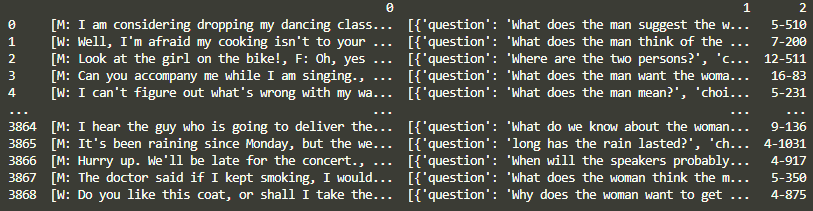


Back to the commonsense\_prediction.py, it extracts the words according to a speaker-based approach which words are stored in a list that belong to some person, it means the words in the list are said by someone. For predicting the answer, it first finds the subject of the question, it should be a person, then it compares the words said by the person and the words in the answer option. Each answer option can get a score by processing the cosine similarity with token vector obtained before. The highest score answer is the predict answer. And finally, it will generate a commonsense\_prediction.tsv which contains the prediction answers of train data.

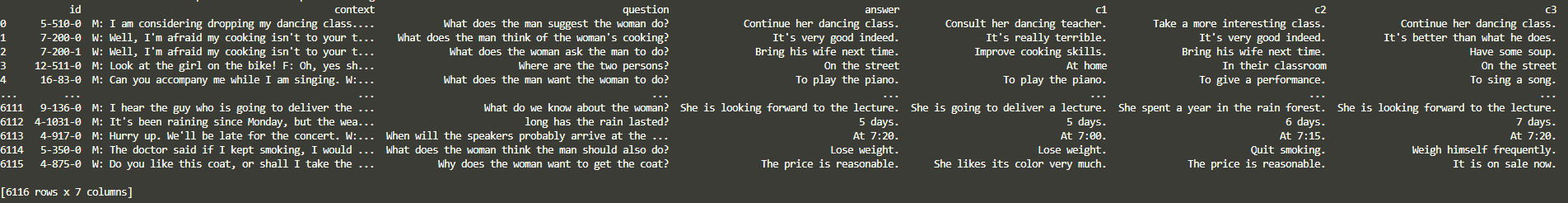


1. **Adopt the commonsense predictions to the train data**

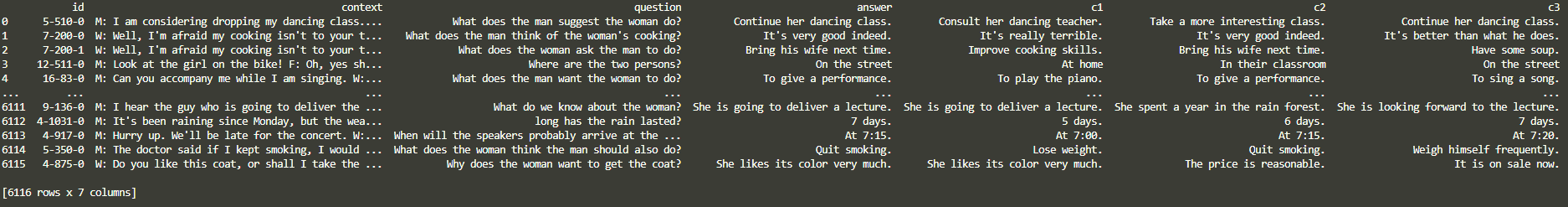
First get the raw data from train. Json



Extract each question with corresponding context, correct answer and answer choices.



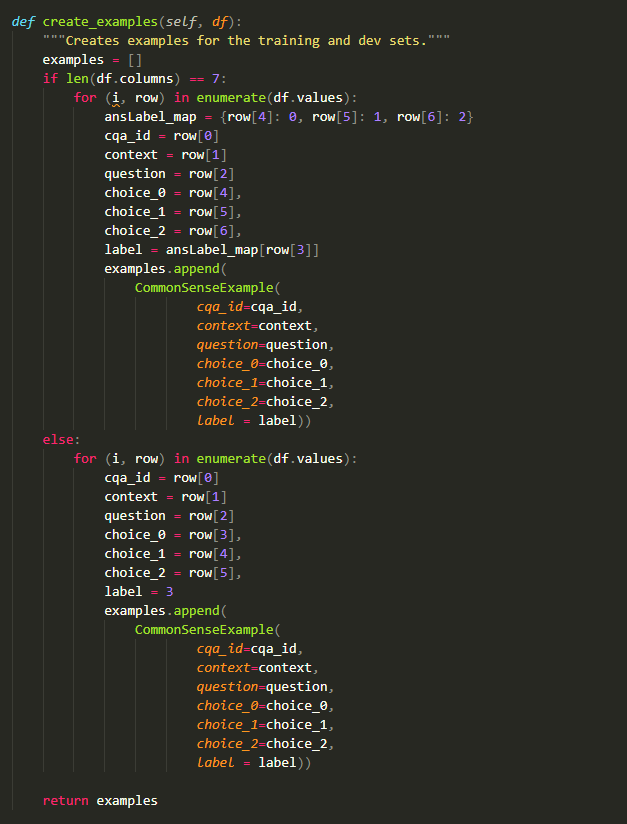
We know that BERT learns to model relationships between sentences, we want Bert to build relationships for the context, question, answer choice, and commonsense predictions. Here, we replace the commonsense predicted answers with the original answer in the data frame.



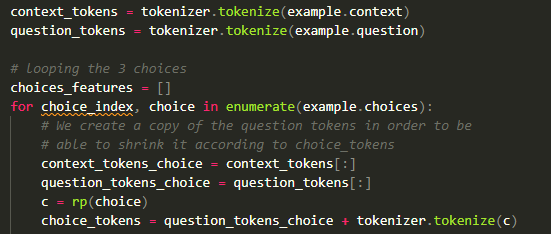
**Feature Engineering**

We need to convert our data into a format that BERT can understand. 

create\_examples() read data frame and loads input. Here we have two different cases. If the data are used for training or evaluation, the inputs should contain the correct answer and label for that answer. If the data are used for prediction, the input does not contain an answer and the label is set to 3 which means it is an invalid label.



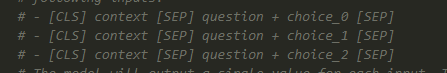
Next, we call convert\_examples\_to\_features() to convert our data into features that BERT can understand. The method first tokenizes the sentences, questions, and choices.

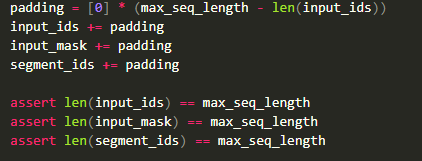
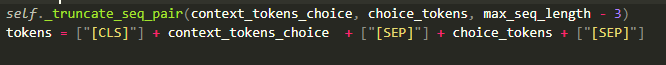
Inserting image...

Tokenizer for convert\_examples\_to\_features()

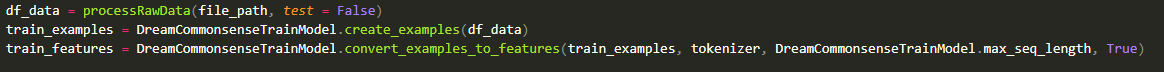


It adds the special “CLS” and “SEP” tokens at the beginning of, between and at the end of context\_tokens\_choice and choice\_tokens, then they can be identified by BERT. It also applies the Zero-padding up to the sequence length. The details of zero padding are mentioned above.

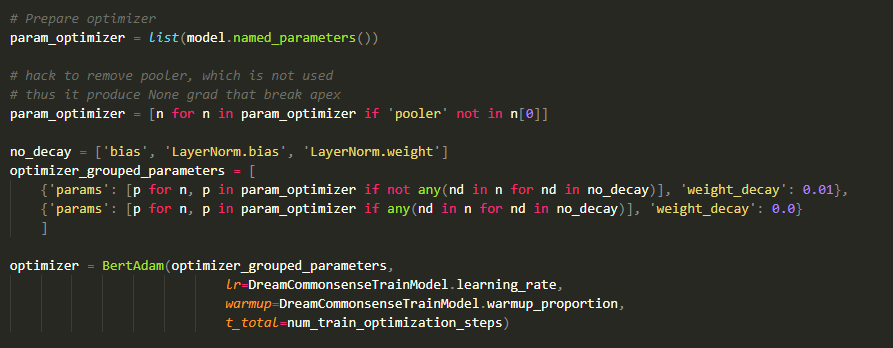




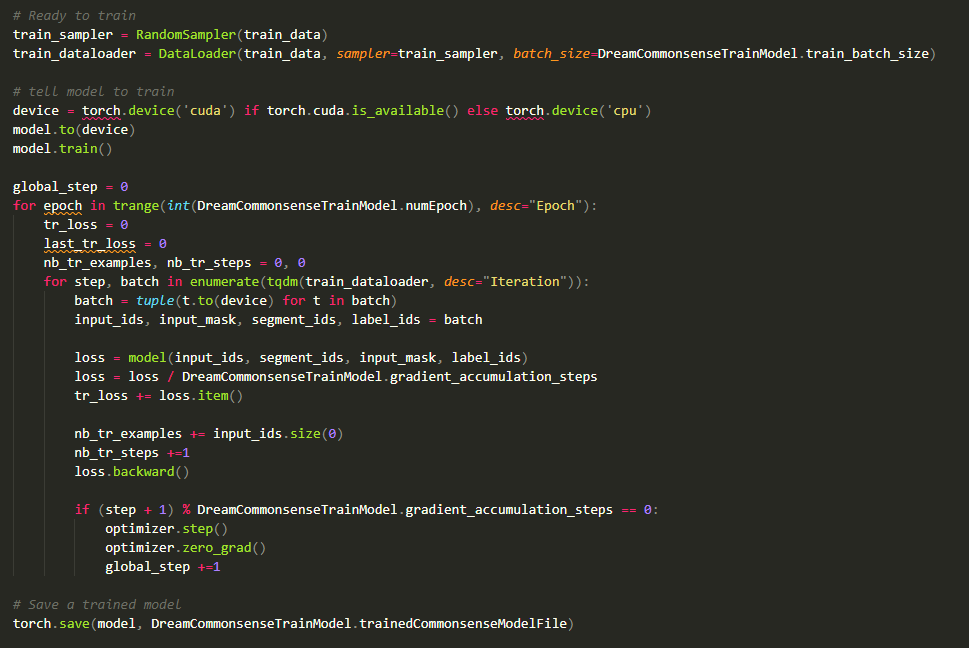
From raw data to train features.



Optimizer for training



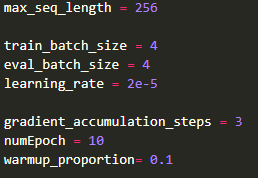
Train and save the model



**Fine-Tuning**

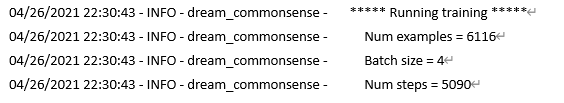
We have a set of hyperparameters that can be used to fine-tune our model. We first tried 128 max sequence length, 3 training batch size and the learning rate is 2e-5, the training epoch is 2, it spent around 18 minutes to train, we achieved 48% accuracy in dev data set and 47% accuracy in test data set. Then we increased the max sequence length to 256, the training batch size, the learning rate, and the training epoch remain unchanged, it spent a similar time period and we achieved 50% accuracy in the dev data set and 49% accuracy in the test data set.

Due to the hardware limitation, we can’t increase the max sequence length and batch size, otherwise it will cause the out of memory. Therefore, we tried to increase the training epoch to 10, other parameters remain unchanged, it spent around 1.5 hours to train. As a result, we achieved 52% accuracy in the dev data set and 51% accuracy in the test data set. We observed that as the max sequence length and the number of training epochs increase, the prediction accuracy tends to increase.

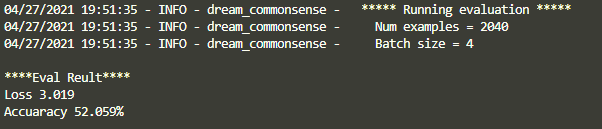


**Measuring Performance**

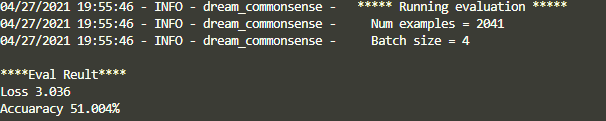
Train Data: 6116, Duration: 1:30:06.



Dev Data: 2040, Duration: 02:01, Accuracy 52.059%



Test Data: 2041, Duration: 02:00, Accuracy 51.004%

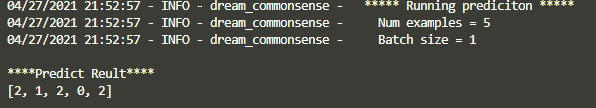


**Making prediction**

Input a document which contains a context, some questions with three answer choices, and the document is in the same format as the Dream dataset.



The system can successfully predict the answers for those five questions. 0 is the first answer choice, 1 is the second answer choice, and 2 is the third answer choice.



# Peer And Self-Assessment

This section is score given each member to reflect his/her contribution towards the project, as evaluated by self and by peers in groups. It is a score between 0 to 10 for each member.

## Member Contribution

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task Description | CHONG  Kit Sang (19005168G) | CHOW  Man Kit  () | CHU  Kin Tung  (20006316G) | NG  Kai Pong (18002565G) | YAU  Pui Hang (20070958G) |
| Group Project Initiation | ✓ | ✓ | ✓ | ✓ | ✓ |
| Python and Bert Models Research | ✓ | ✓ | ✓ | ✓ | ✓ |
| Python Main Function |  |  |  | ✓ |  |
| MCTest Model Development |  |  | ✓ |  |  |
| RACE Model Development | ✓ |  |  |  |  |
| DREAM Model Development |  | ✓ |  |  | ✓ |
| Final Report | ✓ | ✓ | ✓ | ✓ | ✓ |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Score gain for each member | | | | |
| CHONG  Kit Sang (19005168G) | CHOW  Man Kit  () | CHU  Kin Tung  (20006316G) | NG  Kai Pong (18002565G) | YAU  Pui Hang (20070958G) |
| Score given by each member | CHONG  Kit Sang (19005168G) | 10 | 10 | 10 | 10 | 10 |
| CHOW  Man Kit  () | 10 | 10 | 10 | 10 | 10 |
| CHU  Kin Tung  (20006316G) | 10 | 10 | 10 | 10 | 10 |
| NG  Kai Pong (18002565G) | 10 | 10 | 10 | 10 | 10 |
| YAU  Pui Hang (20070958G) | 10 | 10 | 10 | 10 | 10 |
| Total | 50 | 50 | 50 | 50 | 50 |

End