Commonsense Validation and Explanation in Natural Language Processing

By

Letian Li

Student ID: 2214560

Supervisor: Dr Mark Lee

A thesis submitted to
the University of Birmingham
for the degree of
MSC IN ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
September 2021

ABSTRACT

This project is aiming to solve the first two subtasks in *SemEval 2020 Task 4 - Commonsense Validation and Explanation (ComVE)*. The overall task is a commonsense reasoning problem in natural language processing. In the first subtask, the project needs to determine which of two similar natural language statements is against common sense. In the second subtask, the project finds out the right reason why a given statement defies common sense. The implementation of this work adopts deep learning algorithm and uses pre-trained model BERT to solve the problem. During the experiment, the project investigated the effects of different models (BERT and its variants) and hyperparameters on program performance. Finally, the project achieved 94.3% and 92.3% accuracy in the two subtasks respectively.

Keywords:

Natural Language Processing, Commonsense Reasoning, BERT

ACKNOWLEDGMENTS

I would like to express my heartfelt thanks to my supervisor, Dr Mark Lee, for his kind guidance and great support during my MSc project. I am glad to choose this interesting project topic with the help of Dr Mark Lee. Also, the study would not have been done without his guidance and advice.

I would also like to thank Dr Anis Zarrad, he gave me valuable suggestions in both project inspection and demonstration period.

I would also like to thank all the lecturers and tutors who taught me in this academic year, as well as the Artificial Intelligence and Machine Learning programme I attended. In this academic year, I not only came into contact with artificial intelligence knowledge for the first time, but also studied and understood relevant knowledge deeply, and did corresponding practice in many course assignments. These processes have become my valuable learning experience and important support for my future career.

Also, I need to say thanks to two of my close friends, Zimin Xia and Dun Liang, who started studying in artificial intelligence earlier than me. They shared their knowledge of this field with me without reservation throughout the academic year.

Finally, I would like to thank my parents. They greatly supported and helped me to pursue my dream. Without my parents' financial and emotional support, I could not have come to the UK to study this fascinating technology that I love.

Contents

List of Figures

List of Tables

Chapter One

Overview

This MSc project is aiming to research on the commonsense reasoning problem in natural language processing. The research is based on *Task 4: Commonsense Validation and Explanation* in SemEval-2020. In this chapter, it briefly introduces the project and describes the objective task, aiming to help readers understand what this project is and what it is for.

1.1 Introduction

Commonsense reasoning plays an important role in the research of natural language processing (Davis, 2017). However, it has not been well developed compared with other research on artificial intelligence. Davis and Marcus, 2015 suggest that commonsense reasoning is an important bottleneck in modern natural language understanding systems. To facilitate the researches on commonsense problem, in the International Workshop on Semantic Evaluation 2020, Wang et al., 2020 proposed *Task 4: Commonsense Validation and Explanation*. In this context, the MSc project chooses Task 4 as the target, aiming to build a program that can distinguish whether a sentence defies common sense and be able to select the corresponding reason.

To implement this work, deep learning algorithm was introduced and BERT (Devlin et al., 2019) was adopted as the main architecture for the project. After successfully solving the target task,

the project conducted a number of experiments to investigate the effects of different models and hyperparameters on program performance. Finally, the project achieved 94.3% and 92.3% accuracy in the two subtasks of the overall Task 4.

The paper will describe the research of this project as follows: In Chapter 2, I will give a brief overview of related work. Chapter 3 is about the methodology of this project. It will focus on the BERT architecture. Chapter 4 describes how the project was implemented. The focus of this paper will be on Chapter 5, which describes the experiment process, results, and evaluation in detail. The achievements and limitations of this project will be discussed in Chapter 6. Meanwhile, a new finding of the project is also discussed: the model shows capabilities beyond training expectations. Finally, the Chapter 7 summarizes the entire project.

1.2 Motivation

It is exciting to choose and implement this project, because commonsense reasoning can make a great contribution to both academic research and practical application. In the research of natural language processing, the ability of commonsense reasoning can assist other research areas. For example, in speech recognition tasks, commonsense reasoning can be used to determine whether there are unreasonable errors in speech recognition. In machine translation research, commonsense reasoning can be used to correct translation sentences that do not conform to common sense. In practical applications, a good example is the virtual assistant. With commonsense reasoning, virtual assistants will certainly provide more user-friendly services. They will answer users' questions in a more commonsense way.

1.3 Task Description

In original *Task 4: Commonsense Validation and Explanation*, there are three subtasks. However, my MSc project only focus on the first two subtasks because the first two subtasks are classification

problems while the last subtask is a generation problem. To be specific, the first one is a **Validation** subtask. The project is going to tell which of two similar natural language statements makes sense to humans; The second task is an **Explanation (Multi-Choice)** subtask, trying to find the right reason why a given statement does not make sense. Below I present examples for the first two subtasks, which are the objectives of my MSc project.

- **Subtask A: Validation**

  Task: Which of the two similar statements is against common sense?

  Statement 1: *Jeff ran a mile today.*

  Statement 2: *Jeff ran 100,000 miles today.*

  In this example, we can tell manually that the first statement is a sensical statement, while the second statement is the nonsensical statement. So my program need to make the right prediction to select the statement 2 as the answer to this example.

- **Subtask B: Explanation (Multi-Choice)**

  Task: Select the most appropriate reason as to why this statement is against common sense.

  False Statement: *Jeff ran 100,000 miles today.*

  Option A: *100,000 miles is too long for one person to be able to run in one day.*

  Option B: *Jeff is a four letter name and 100,000 has six numerical digest.*

  Option C: *100,000 miles is longer than 100,000 km.*

  In this example, the option A is the most appropriate reason to say why the given statement is nonsensical. So my program need to make the right prediction to select option A as the answer to this example.

Chapter Two

Related Work

This section describes the related implementation work for Task 4: Commonsense Validation and Explanation (Wang et al., 2020). Since this task was released in 2020, there have been many research teams involved in this task. Table 2.1 shows the achievements of some research teams on subtask A and subtask B. In subtask A, the top five teams all achieved an accuracy over 96%, while in subtask B, the best result also reached 95%. Meanwhile, humans can perform 99.1% and 97.8% accuracy on the two subtasks, respectively.

In both subtask A and subtask B, most research teams use pretrained language models, such as BERT (Devlin et al., 2019), RoBERTa (Y. Liu et al., 2019), ALBERT (Lan et al., 2020) and XLNET (Yang et al., 2019), to solve the problems. Team *CN-HIT-IT.NLP* (Zhang et al., 2020) and team *ECNU-SenseMaker* (Zhao et al., 2020) achieved the best accuracy on subtask A and subtask B respectively. Both teams used the K-BERT (W. Liu et al., 2020) as the main architecture to solve the task and took advantage of an external commonsense repository, ConceptNet (Speer, Chin, and Havasi, 2017), for additional pretraining. Team *IIE-NLP-NUT* (Xing et al., 2020) won third place in both subtask A and subtask B. They used RoBERTa to solve the tasks and added a round of pre-training process with the textual corpus, Open Mind Common Sense (Singh et al., 2002), to improve model performance. Some other teams, like Srivastava et al., 2020, Wan and Huang, 2020 and Saeedi et al., 2020, used RoBERTa as well while Pai, 2020 and Mohammed and Abdullah, 2020 used ensemble method based on model BERT, RoBERTa, ALBERT and XLNet.

To sum up, according to the existing relevant research work, the pretrained models, like BERT and its variants, have a good performance in such problems, and those models are also the preference of the majority of researchers. Besides, when such models add an additional round of pre-training to other language corpus, the performance will be further improved. Therefore, in this project, I also try to use pre-trained models to solve target tasks, and investigate the effects of different pre-trained models and hyperparameters on performance.

Subtask A			**Subtask B**		
Team	**Accuracy**	**Rank**	**Team**	**Accuracy**	**Rank**
Human	99.1	-	Human	97.8	-
CN-HIT-IT.NLP	97.0	1	ECNU-SenseMaker	95.0	1
ECNU-SenseMaker	96.7	2	CN-HIT-IT.NLP	94.8	2
IIE-NLP-NUT	96.4	3	IIE-NLP-NUT	94.3	3
nlpx	96.4	3	Solomon	94.0	4
Solomon	96.0	5	LMVE	93.8	5

Table 2.1: Top 5 in Leaderboard of SemEval-2020 Task4 (Wang et al., 2020)

Chapter Three

Methodology

To solve these two classification subtasks of *SemEval-2020 Task 4*, various natural language processing technologies need to be introduced. Based on what discussed in Chapter 2 Related Work, it can be seen that the pre-trained models have state-of-the-art performance on such tasks. Therefore, in this MSc project, I choose BERT (Devlin et al., 2019) and its variant (Sanh et al., 2020; Y. Liu et al., 2019) as the main architectures for both subtask A and subtask B. In this chapter, I will give a more comprehensive introduction to (1) what is BERT; (2) What is its core mechanism; (3) Why BERT outperforms; and (4) how to use BERT.

3.1 What is BERT

The full name of BERT is **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. It was first proposed by Devlin et al., 2019. As its name suggests, BERT is a model based on Transformer Encoder. Transformer was proposed by Vaswani et al., 2017, and its biggest highlight is that Transformer only uses Attention mechanism, dispensing with recurrence and convolutions completely. There are two similar parts, Encoder and Decoder, in Transformer architecture. The Encoder is mainly used for processing input data, while the Decoder is mainly used for output work. Since BERT is a model only based on Transformer Encoder, it has a better effect on processing classification problems, which are exactly the objectives of my project, but it is not competent for generative

tasks.

To sum up, BERT is a model based on the Encoder of Transformer, while the core mechanism of Transformer is Attention. Figure 3.1 somehow shows the relationship between these three concept.

Figure 3.1: Relationship between Attention, Transformer and BERT

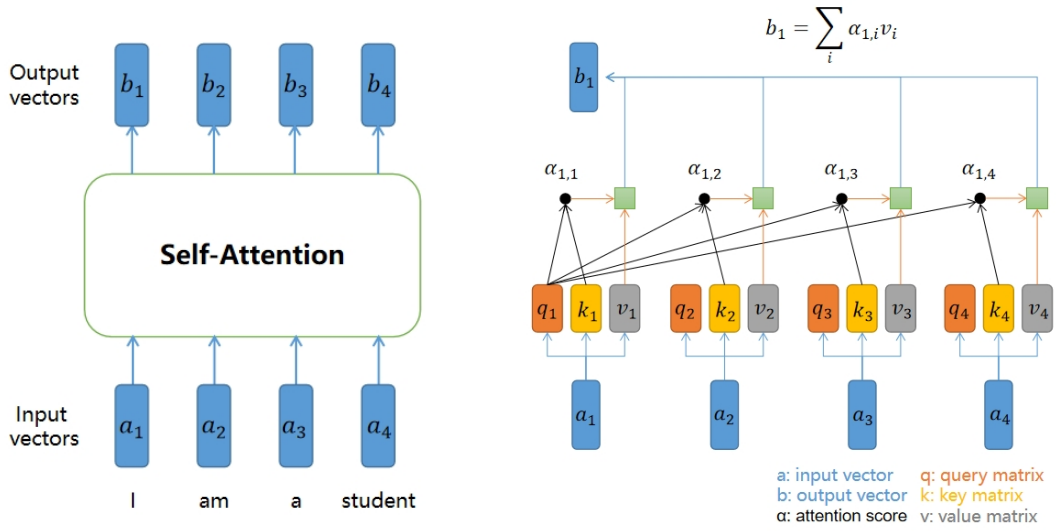
3.2 Self-Attention Mechanism

So, what is this core mechanism, Attention. There are many forms of attention, such as self-attention and cross-attention. The Attention mechanism used in Transformer and BERT is Self-Attention. Next, I will briefly introduce the Self-Attention mechanism based on Figure 3.2.

Imaging the Self-Attention as a block of computation, like what Figure 3.2(a) shows, it just takes a row of vectors and then output another row of vectors. The highlight is that, in the row of output vectors, each vector contains the whole information from the input vectors. The more detailed computation inside of Self-Attention is shown in Figure 3.2(b). It first compute three matrices, Query matrix, Key matrix and Value matrix, for each input vector. And then it use the Query matrix and Key matrix to compute α, which is called attention score. It represent how input vectors are related to each other. And finally it use the Formula 3.1 to compute the output vector.

$$b_1 = \sum_i \alpha_{1,i} \cdot v_i \tag{3.1}$$

Thus, the output vector can contain all the information from input vectors. This is briefly the main idea of Self-Attention.



(a) Function of Self-Attention (b) Schematic diagram of Self-Attention

Figure 3.2: Self-Attention Mechanism

3.3 Why BERT

In my opinion, BERT can outperform other models because of two reasons. One is this powerful Self-Attention mechanism, which can make the model collect all information from each word in the input sentences. An other reason is that BERT is a pretrained model. In fact, BERT has been pretrained on a large corpus of English data and it was trained with two objectives. One is Masked Language Modeling (MLM), which is to train the model to predict masked word in a given sentence. The other objective is Next Sentence Prediction (NSP), which is that when given the previous sentence, let the model predict the next one. By training for these two goals, BERT can achieve state-of-the-art performance on many big-name datasets like GLUE, MultiNLI, and SQuAD (Devlin et al., 2019).

3.4 How to use BERT

The usage of BERT is relatively simple, because there are already many well-developed software libraries supporting BERT network. Theoretically, since BERT is a pre-trained model, we only need to retrain it in downstream tasks when implementing it. The process of retraining the pre-trained model in downstream tasks is called Fine-Tuning. Here, downstream tasks refer to concrete tasks during application. For example, in this project, the downstream tasks means subtask A and subtask B. And the process of training on the dataset provided by these subtasks is Fine-Tuning.

Chapter Four

Implementation

In this chapter, I will introduce how I implement BERT to solve the target tasks in my project. I will start with an introduction to the dataset used for the project, then reveal the third-party software libraries used for the project, and finally describe the five steps of my implementation.

4.1 Dataset Description

The dataset of this project follows the same dataset from *SemEval 2020 Task 4 - Commonsense Validation and Explanation (ComVE)*, which is collected by Wang et al., 2020. In other words, they are identical. More specifically, there are 11,997 instances in this dataset, including 10,000 training data, 997 development (or validation) data, and 1,000 test data. To avoid data imbalance, the labels of the all three dataset were reshuffled. Therefore, the correct label will not be concentrated on a certain option, but relatively evenly distributed on each option, as shown in the Table 4.1.

4.2 Software Libraries

In the spirit of not reinventing the wheel, a large number of third-party libraries were used in this project. In addition, mature third-party libraries can make the project more powerful.

	Subtask A		Subtask B		
	Statement 0	**Statement 1**	**Option A**	**Option B**	**Option C**
Training set	4979	5021	3195	3362	3443
validation set	518	479	344	327	326
test set	508	492	320	355	325

Table 4.1: Distribution of correct labels (or answers)

The project chose Python as the programming language and PyTorch as the deep learning framework. Meanwhile, Jupyter was selected as the IDE (Integrated Development Environment), and TensorBoard was used to assist the visualization of results.

In the selection of python third-party software libraries, this project uses Numpy and Pandas for data processing, and Matplotlib and Seaborn for drawing graphs. Among all software libraries, the most important thing is to use Hugging Face's Transformers (Wolf et al., 2020) to implement BERT and its variants.

4.3 Implementation

Next, I will briefly walk through the five steps of implementation work in the project code.

4.3.1 General Preparation

In the preparation step, the program first imports some common libraries, then sets up the GPU to be used, and finally configures the implemented model and hyperparameters.

4.3.2 Data Processing

In the second step, data processing, the program first reads the three datasets of training, validation and test from CSV file. Then a core data processing work, tokenization, is carried out. Unlike computer vision tasks, tokenization is an important step in natural language processing. It splits sentences into smaller units. These smaller units are called tokens, and in this project tokens will be represented by numbers. In the implementation process, the tokenization of subtask A and subtask B is slightly different. This is because subtask A is a binary classification problem, while subtask B is a multi-classification problem. Hugging Face's Transformer uses different tokenization methods for these two types of problems. So I tokenized the two tasks differently, which can be reflected in the project code.

4.3.3 Loading the Model and Optimizer

The third step is to load the model and optimizer. In this step, there is not much code to write, because the model and optimizer are directly loaded into this project from a third-party library. Hugging Face's Transformers (Wolf et al., 2020) provides a lot of pre-trained models. In the best implementation work of this project, Roberta-Large model (Y. Liu et al., 2019) was adopted, while in the experiment work of this project, model Bert-Base-Uncased (Devlin et al., 2019), Distilbert-Base-Uncased (Sanh et al., 2020) and Roberta-Base (Y. Liu et al., 2019) were tested as well (see Section 5.1.2).

In this project, AdamW (Loshchilov and Hutter, 2019) is adopted as the optimizer.

4.3.4 Training

Model training is an important step to realize the project and solve the target task. In this project, the training procedure is based on PyTorch deep learning framework. More specifically, during the training process, the program first sets the number of epochs and batch size. Then, in the training

of each epoch, the model will load the corresponding number of data (tokenized) for training. After the model makes predictions, the program will calculate the loss based on the difference between prediction value and ground truth value. Here, the program uses the CrossEntropy that comes with the model as the loss function. Finally, after calculating the losses, the program performs back propagation and optimizes the model parameters with the optimizer.

It is worth noting that in the implementation work, model training did not do validation work. The validation data will be treated as training data for direct training and improve the model performance. However, in the experiment work, the validation data will be used for validation process so that it can evaluate the performance of the model. This is because, during the experiment, all the test data will not be used. The test data is only used in the final evaluation of model performance. **In short, in the implementation work, both the training data and the validation data will be used to train the model, and the test data will be used to evaluate the performance of the model. However, in the experiment work, the training data is used to train the model, and the validation data is used to evaluate the performance of the model, while the test data is not used.**

4.3.5 Testing

Finally, the trained model will be evaluated on the unseen test dataset. The testing process is similar to the training process. The difference is that during the testing process, the optimizer will stop working and will no longer update the parameters of the model. In the testing phase, we only evaluate the predicted value of the model to check its performance. The evaluation methods are as follows: 1. Directly view the predicted results and real answers for each instance (like Figure 5.10). 2. Calculate the overall accuracy for the test set. 3. Judge the performance by observing the confusion matrix (like Figure 5.9). In addition, for subtask A, an additional round of test was added, which is manual testing. It allows users to customize other input data and let the model make predictions. (See manual test details in the evaluation Section 5.5.3)

Chapter Five

Experiment

After I successfully implemented the BERT to my task, I got nearly 80% of accuracy for each subtask. To improve the program performance and understand how models and hyperparameters affect the performance, I did a series of experiments on both subtask A and subtask B. And finally, I improved the accuracy of my program to more than 90%. A comparison of model performance at different stages of the experiment can be seen in Figure 5.1. In this chapter, I will present some results of my experiment and do the analysis based on the those results.

5.1 Experiment Overview

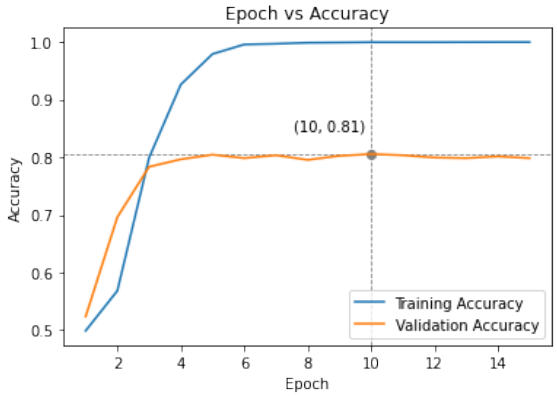
The experiment was carried out based on two objectives, and different models and hyperparameters were tested respectively. The experiment procedure is shown in Section 5.1.3.

5.1.1 Experiment Objectives

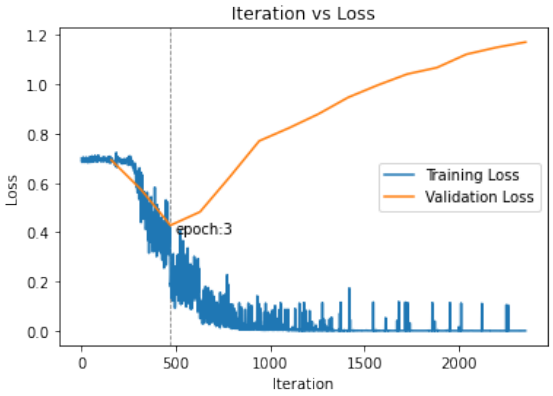
There are two objectives for the experiment.

- Find the best model and hyperparameters.
- Investigate the influence of hyperparameters.

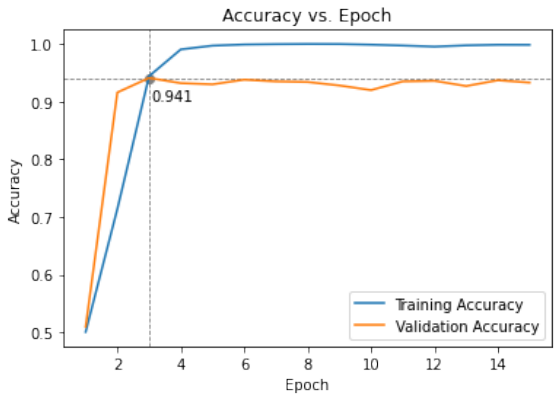
The main purpose of my experiment is to find out the best performance model and hyperparameters for the task. At the same time, I also studied how these hyperparameters influence model training.



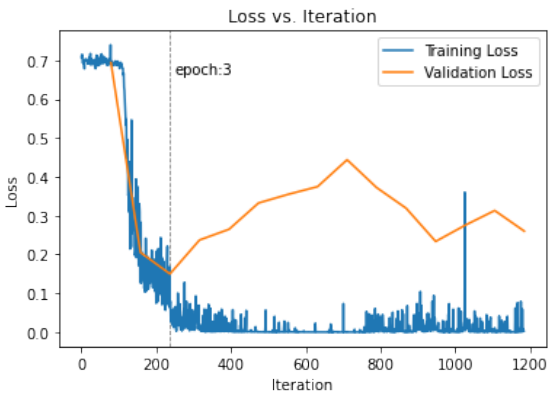
(a) Accuracy in the early stages



(b) Loss in the early stages



(c) Accuracy in the final stage



(d) Loss in the final stage

Figure 5.1: Comparison of model performance at different stages of the experiment (Subtask A)

5.1.2 Experiment Models and Hyperparameters

Experiment Models

In the experiment, four pretrained models were used for testing, which are Bert-Base-Uncased (Devlin et al., 2019), Distilbert-Base-Uncased (Sanh et al., 2020), Roberta-Base and Roberta-Large (Y. Liu et al., 2019). In fact, Bert-Base-Uncased is the original BERT model while Distilbert-Base-Uncased, Roberta-Base and Roberta-Large are three variants of it.

Bert-Base-Uncased is a pretrained model on a large corpus of English data, with two self-supervised learning objectives: masked language modeling (MLM) and Next sentence prediction (NSP). Here, uncased means it is insensitive to whether the first letter is capitalized. This model has 12-layer, 768-hidden, 12-heads, 110M parameters.

Distilbert-Base-Uncased is a distilled version of the normal BERT base model. It is smaller in size but can be trained much faster by dropping a few layers and parameters. In fact, this model has 6-layer, 768-hidden, 12-heads, 66M parameters.

Roberta-Base is another variant of the normal BERT base model. However, it was trained longer and with larger batch size and training data. Another change was that this model dropped Next sentence prediction (NSP) training objective but kept masked language modeling (MLM) training objective. All in all, this model has 12-layer, 768-hidden, 12-heads, 125M parameters.

Roberta-Large is a large version of the Roberta Base model. It uses the BERT-large architecture but is trained in the same way as Roberta-Base. This model is the largest model in this experiment with 24-layer, 1024-hidden, 16-heads, 355M parameters. Because of its large model, the batch size of this model had to be somehow reduced to complete the experiment.

Experiment Hyperparameters

The experiment focused on the three core hyperparameters: epoch, learning rate and batch size.

Epoch is the value of how many times the dataset has been used to train the model.

Learning rate is the hyperparameter which controls how quickly the model learns a problem.

Batch size is the number of training examples utilized in one iteration.

5.1.3 Experiment Workflow

In total, I did four rounds of experiment for both subtask A and subtask B. The first round was grid search. It was used to find out the best model and narrow down the range of hyperparameters. In the next three rounds of experiment, I did the research on Epoch, Learning Rate and Batch Size respectively. To sum up, the workflow of my experiment is as follows,

Round 1 *Grid Search on Different Models*

Grid search is a process of exhaustive search in a certain hyperparameter space. It tries all possible combinations of hyperparameters.

In this round of experiment, I did grid search on different models, and then compared the performance of different models and chose the one with the best performance to be the final model. Besides, by doing the grid search with different hyperparameters, it is also contributed to locating a good range of learning rate and batch size. After grid search, I select the model and hyperparameters which achieve the highest accuracy as a baseline for subsequent experiments.

Round 2 *Research on Epoch*

For the experiment on hyperparameter epoch, different values of epoch was tested. After comparing the accuracy curves of different epoch, an appropriate epoch was selected for the final model.

Round 3 *Research on Learning Rate*

In the experiment of learning rate, several values of learning rate have been tested. By analyzing the experiment results, I first compared how different learning rates affect model training and then determined an appropriate learning rate for the final model.

Round 4 *Research on Batch Size*

This round of experiment is similar to the learning rate one. After testing different batch sizes, the corresponding performances was analyzed and the best performance batch size was selected for the final model.

After four rounds of experiments, the best-performing model and hyperparameters were determined for the final implementation. In the project code, the *Best Implementation* demonstrates its powerful ability to solve the given task. I will discuss the best implementation later in Section 5.5.

5.2 Implementation of Experiment

The implementation of the experiment is similar to the pure project implementation, which is discussed in Section 4.3. The difference is that in the experiment code, various operations are encapsulated into functions. By calling functions reasonably, the program can test multiple hyperparameters at one time during the experiment. In addition, TensorBoard was introduced in the experiment to help visualize the experiment results.

Similar to best implementation work, the implementation of the experiment includes five steps as well: (The following is a brief introduction to the experiment implementation, focusing on the differences from the project implementation. See Section 4.3 for detailed project implementation.)

i) **General Preparation**

In the preparation step, the program will import the relevant public libraries, specify the GPU used in the experiment, and configure the storage path of TensorBoard records.

ii) **Data Processing**

In this step, after reading the dataset from the CSV files, the program will use different tokenization functions to process the data according to different tasks.

iii) **Loading the Model and Optimizer**

In this step, the program will load the specified model and optimizer by calling different functions.

iv) **Training and Evaluation**

Unlike the best implementation code, in the fourth step, the experiment code includes two

operations, training and evaluation. This time, the validation set will be used as the only way to evaluate the performance of the model.

v) **Experiment**

The last step is to perform the experiment. In this step, the program will experiment with the specified hyperparameters by calling various functions. At the end of the experiment, the program will automatically draw relevant graphs and save the experiment results to a new CSV file.

5.3 Experiment of Subtask A

Now, let me introduce the experiment work of subtask A. Subtask A is a validation problem. In fact, it is a binary classification task. In the first successful implementation of this task, the accuracy was 79.3% (with model of distilbert, learning rate of 5×10^{-5}, batch size of 16 and epoch of 10). However, after four rounds of experiment, the best performance was 94.3% (with model of roberta-large, learning rate of 10^{-5}, batch size of 128 and epoch of 15).

5.3.1 Grid Search on Different Models

This section is used to present the experiment result of grid search on four different pretrained models, which are Bert-Base-Uncased (Devlin et al., 2019), Distilbert-Base-Uncased (Sanh et al., 2020), Roberta-Base and Roberta-Large (Y. Liu et al., 2019). Since this is the first round of exploration, only 5 epochs are set to quickly locate a good range of learning rate and batch size. After this round of grid search, the model and the hyperparameters with the highest accuracy will be chosen as the baseline for subsequent experiments.

Original Grid Search Settings are:

Model: *Bert-Base-Uncased, Distilbert-Base-Uncased, Roberta-Base*

Learning rate list: *[10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]*

Batch size list: *[8, 16, 32, 64, 128, 256]*

Epoch: *5*

Further experiment on Roberta-Large:

Comparing the performance of Bert-Base-Uncased model, Distilbert-Base-Uncased model and Roberta-Base model in Figure 5.2, the accuracy of Roberta-base model is the highest. So I did a further experiment on the Roberta-Large model. Since the Roberta-Large model is a fairly Large model, I narrowed down the range of grid search. The Hyperparameters Settings are as follows,

Model: *Roberta-Large*

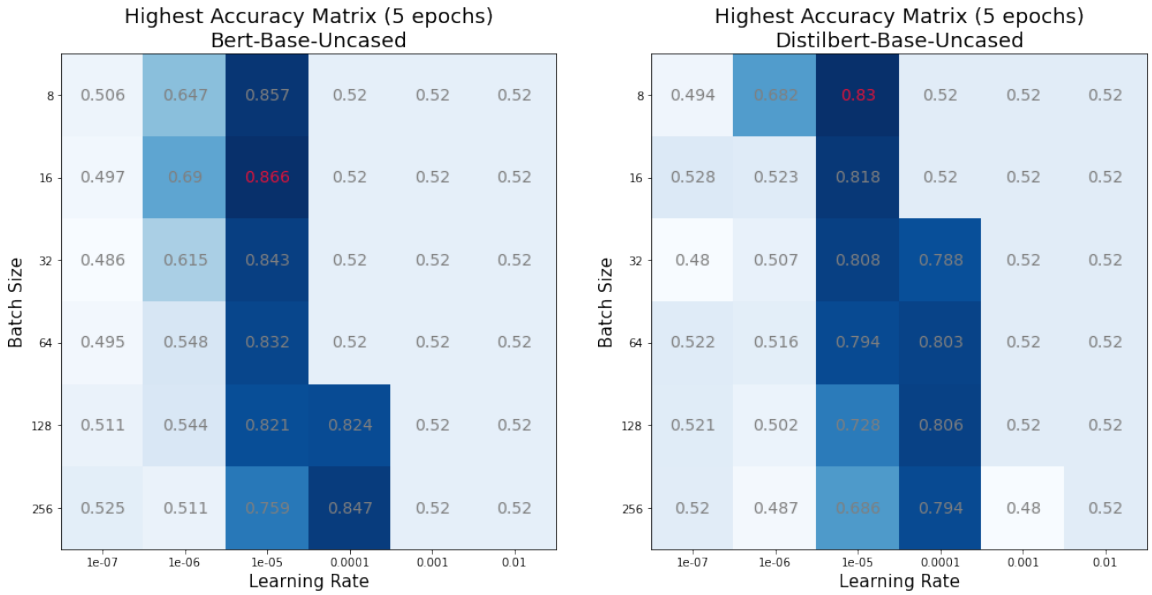
Learning rate list: *[10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}]*

Batch size list: *[8, 16, 32, 64, 128]*

Epoch: *5*

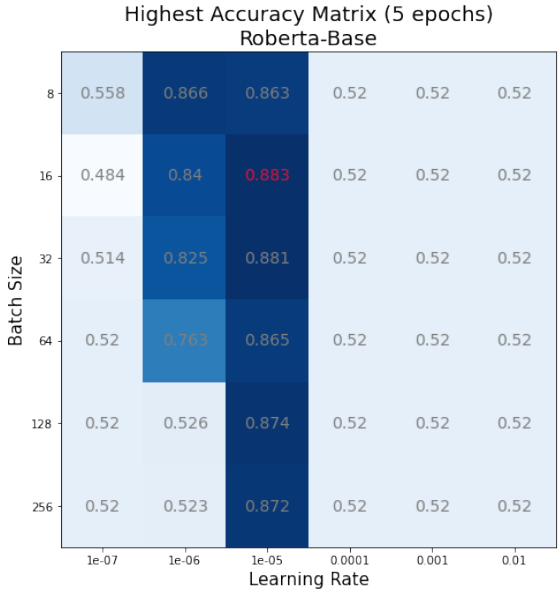
Conclusion:

According to the four accuracy matrices Figure 5.2, Roberta-Large model got the highest accuracy (0.947), with a learning rate of 10^{-5} and a batch size of 128. Therefore, this model and hyperparameters were used as the baseline for subsequent experiments.

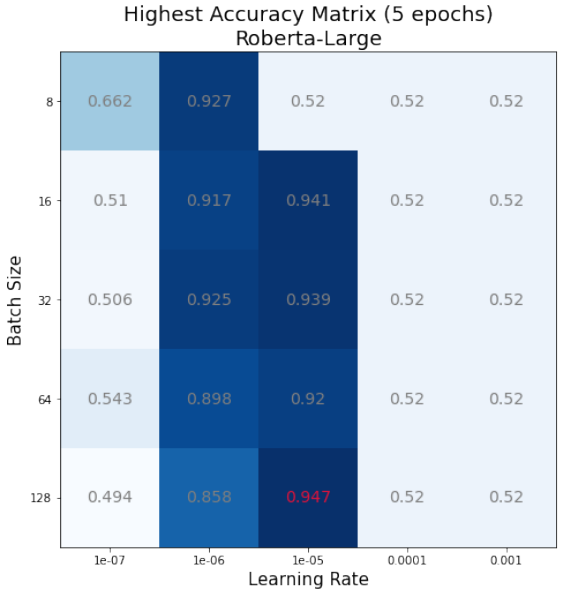


(a) Bert-Base-Uncased

(b) Distilbert-Base-Uncased



(c) Roberta-Base



(d) Roberta-Large

Figure 5.2: The results of round 1: grid search for subtask A. Highest accuracy matrix (5 epochs) of four different pretrained models.

5.3.2 Research on Epoch

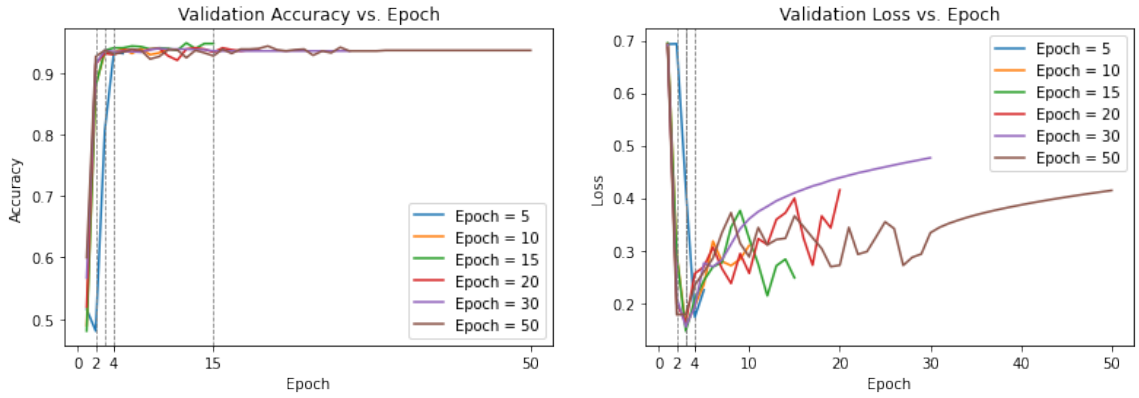
The experiment for epoch is based on the following paramters,

Selected final model: *Roberta-Large*

Baseline learning rate: 10^{-5}

Baseline batch size: *128*

Search list of epoch: *[5, 10, 15, 20, 30, 50]*



(a) Validation Accuracy

(b) Validation Loss

Figure 5.3: The results of round 2: research on epoch for subtask A.

On the one hand, Figure 5.3 shows around 2-4 epoch, the model has achieved good performance (Figure 5.3(a)). On the other hand, in the 6 experiments on epoch, the validation loss no longer decreased but increased in the second, third, and fourth epochs (shown in Figure 5.3(b)). This means that the model has started to overfit. However, the highest accuracy is located over 10 epoch, so I conservatively chose 15 epochs as the final value of this hyperparameter for subsequent experiment.

5.3.3 Research on Learning Rate

In this part, three lists of learning rates were tested to make the experiment more accurate. The experiment model and hyperparameters are as follows,

Selected final model: *Roberta-Large*

Search list of learning rate:

- $[10^{-7}, 2 \times 10^{-7}, 4 \times 10^{-7}, 6 \times 10^{-7}, 8 \times 10^{-7}]$,
- $[10^{-6}, 2 \times 10^{-6}, 4 \times 10^{-6}, 6 \times 10^{-6}, 8 \times 10^{-6}]$,
- $[10^{-5}, 2 \times 10^{-5}, 4 \times 10^{-5}, 6 \times 10^{-5}, 8 \times 10^{-5}, 10^{-4}]$

Baseline batch size: *128*

Selected final epoch: *15*

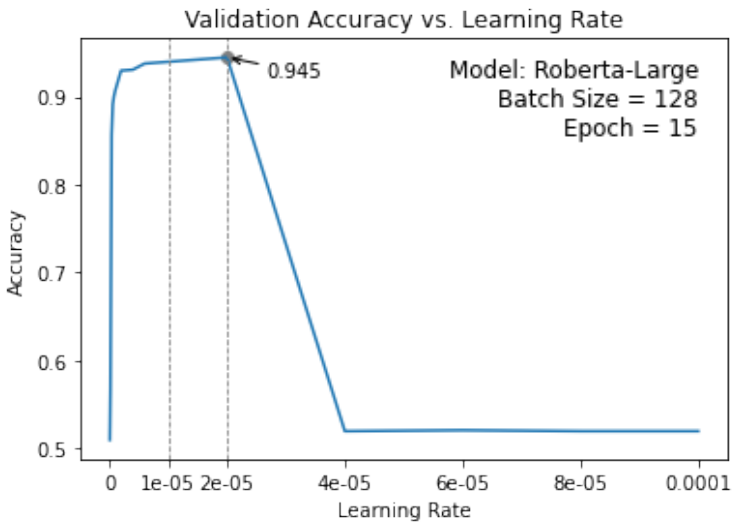
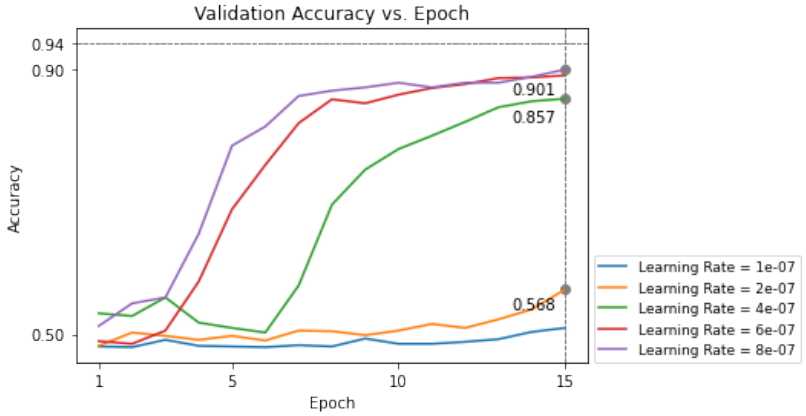
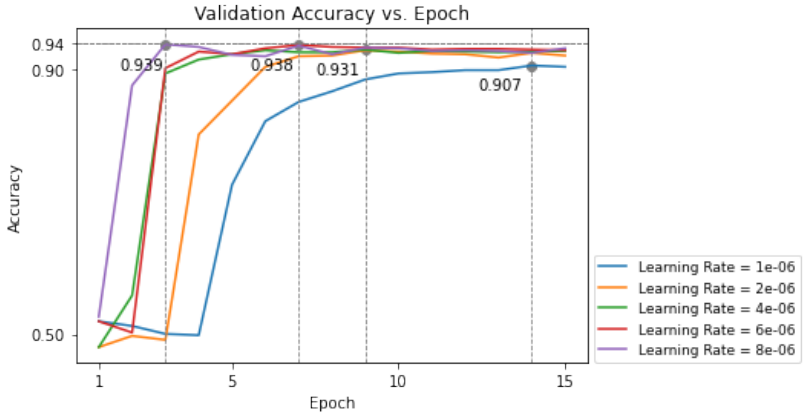


Figure 5.4: The result of round 3: research on learning rate for subtask A. Validation accuracy of different learning rates.

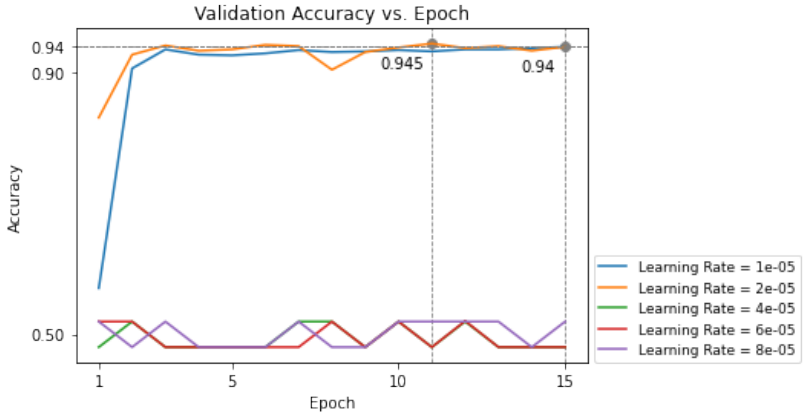
It can be observed from the Figure 5.5 that as the learning rate increases, the training speed becomes faster and faster. At the same time, the accuracy of the trained model has also been improved.



(a) Learning rate of 10^{-7}



(b) Learning rate of 10^{-6}



(c) Learning rate of 10^{-5}

Figure 5.5: The results of round 3: research on learning rate for subtask A. Accuracy curves of different learning rate.

However, when the learning rate is greater than 2×10^{-5}, the accuracy of the model oscillates around 0.5 (Figure 5.5(c)). This suggests that the model is not successfully trained, but is making random predictions. This situation occurs because the learning rate is too large, which leads to too much amplitude when update the model parameters. This may cause the optimal solution to be skipped, resulting in unsuccessful training. Therefore, we can draw such a conclusion: when the learning rate is too small, the training speed is too slow. When the learning rate is too large, the training will fail.

By comparing the successful training examples, it can be seen that when the learning rate was around 10^{-5}, the accuracy reached the highest value (about 0.94). Although the accuracy reached the highest value when the learning rate was 2×10^{-5} (Figure 5.4), it was found in the subsequent experiments that the model training of this learning rate was not stable. Sometimes an effective model can be successfully trained, but sometimes not. This indicates that the learning rate is high enough to reach a critical value. That's why it shows the instability of training. Therefore, I chose 10^{-5}, which was a smaller test value than 2×10^{-5}, as the best learning rate. At the same time, we can also see that when the learning rate was 10^{-5}, the accuracy was only slightly lower than that of 2×10^{-5}.

5.3.4 Research on Batch Size

The model and hyperparameters for batch size experiment are as follows,

Selected final model: *Roberta-Large*

Selected final learning rate: 10^{-5}

Search list of batch size: *[8, 16, 32, 64, 128]*

Selected final epoch: *15*

In the experiment of batch size, when we look into the experiment results, we can draw the conclusion as follows:

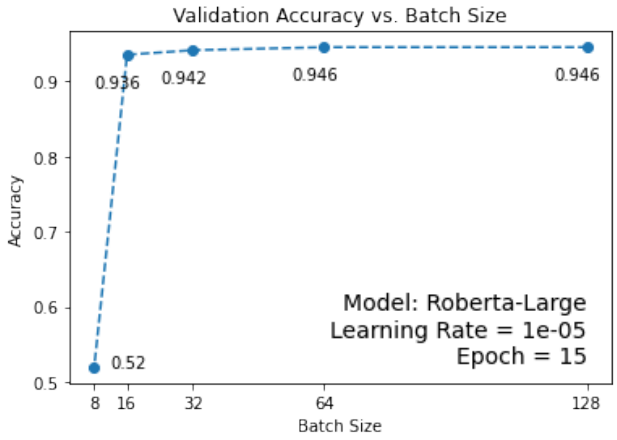


Figure 5.6: The result of round 4: research on batch size for subtask A. Validation accuracy of different batch sizes.

i) The smaller the batch size, the greater the fluctuation of training loss in each iteration. This is because there is less training data in each batch, which leads to more unstable training directions. (Observed from Figure 5.7(b))

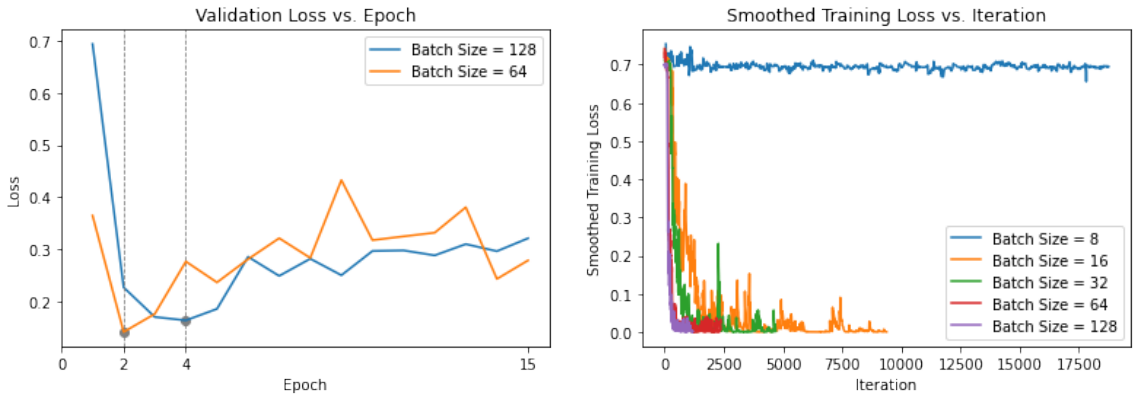
ii) The larger the batch size, the faster the training loss decreases as the iteration progresses. Due to the increase of training data in each iteration, the training direction is more stable, and the training becomes more effective. However, when it comes to epochs, the larger the batch size, the slower the training. The reason is that the number of iterations per epoch decreases as the batch size increases, which means that there is less opportunity for each epoch to update model parameters. (Compare Figure 5.7(a) and Figure 5.7(b))

iii) Finally, 128 was chosen as the best batch size because it showed the best accuracy (in Figure 5.6) and late overfitting (in Figure 5.7(a)).

5.3.5 Final Model and Hyperparameters

Based on the experiments above, I set the final model and hyperparameters of subtask A as follows:

Selected final model: *Roberta-Large*



(a) Validation loss of different batch sizes

(b) Smoothed training loss of different batch sizes

Figure 5.7: The results of round 4: research on batch size for subtask A. Comparing training and validation loss of different batch sizes.

Selected final learning rate: 10^{-5}

Search list of batch size: *128*

Selected final epoch: *15*

5.4 Experiment of Subtask B

Since the experiment of subtask B is basically the same as that of subtask A, only a brief description of the experiment of subtask B will be given here. (Similarly, I plot as many graphs as subtask A but only one Figure 5.8 is shown in this paper. Other graphs can be viewed in the project code.)

For subtask B, I also conducted four rounds of experiments:

i) **Round 1: Grid Search**

In the first round of experiments, I conducted grid search on four models (Bert-Base-Uncased, Distilbert-Base-Uncased, Roberta-Base, Roberta-Large) with different hyperparameters. Only Roberta-Large achieved an accuracy over 90%, as shown in the Figure 5.8. Therefore, I selected the model and the hyperparameters with the highest accuracy as the baseline of subsequent

experiments.

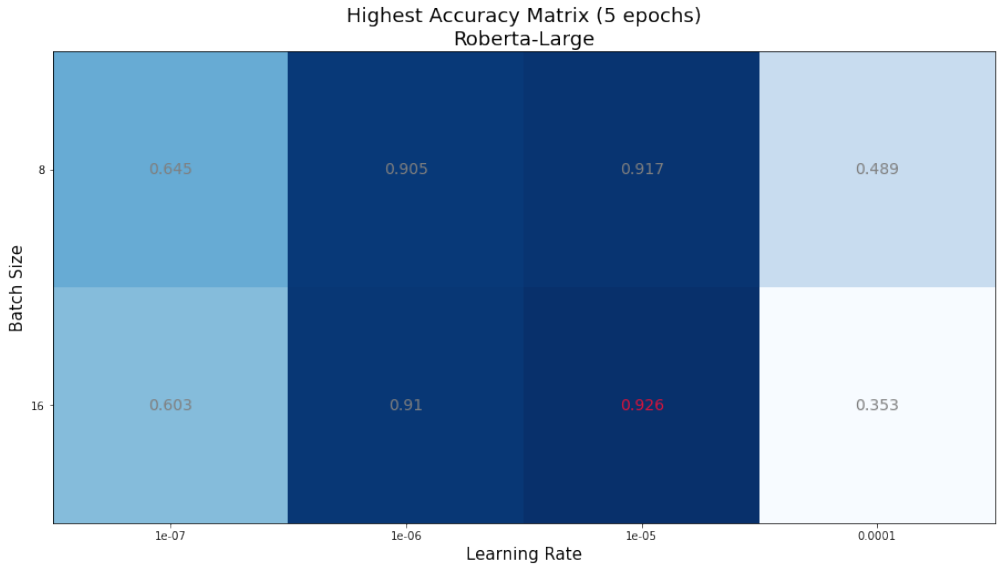


Figure 5.8: Highest Accuracy Matrix of Subtask B

ii) **Round 2: Research on Epoch**

I tried five different values (5, 10, 15, 20, 30) for epoch. The highest accuracy for each experiment can achieve is around 91.9%, and they can get to this accuracy usually in 5 to 10 epochs (only the experiment of 30 epoch fails). So I chose 10 as the final value of the hyperparameter epoch.

iii) **Round 3: Research on Learning Rate**

Again, I tried five different learning rates. They are 5×10^{-7}, 1×10^{-6}, 5×10^{-6}, 1×10^{-5} and 5×10^{-5}. Different learning rates show similar effects as discussed in subtask A (Section 5.3.3). Thus, 1×10^{-5} was selected because of its high accuracy and faster learning speed.

iv) **Round 4: Research on Batch Size**

In this round of experiments, as the model is too large, the maximum batch size that can be loaded is 16. So I only tested the three batch size values of 4, 8 and 16. However, it also showed a similar situation to subtask A (Section 5.3.4). At last, the value of 16 was selected because of its higher validation accuracy and lower validation loss.

Finally, I set the final model and hyperparameters of subtask B as follows:

Selected final model: *Roberta-Large*

Selected final learning rate: 10^{-5}

Search list of batch size: *16*

Selected final epoch: *10*

5.5 Best Implementation after Experiment

Through four rounds of experiments on task A and task B respectively, the best model and hyperparameters were determined (shown in Table 5.1). Subsequently, I used them as the final models for another best implementation. Please note that in the best implementation work, there is no validation process, because it has been carried out in the experiment work. Thus, the validation dataset will be used as the training dataset to improve the performance of the best implementation. In this section, I will discuss the performance of the final model and their evaluation.

5.5.1 Evaluation Method

In both subtask A and subtask B, following the suggestion given by Wang et al., 2020, the prediction **accuracy** serve as the metric to evaluate model performance. At the same time, **confusion matrices** are also added to help visualize the prediction results to better evaluate the performance of the model. In addition, to make the program more interesting and interactive, a **manual test** was introduced for subtask A.

5.5.2 Results and Evaluation

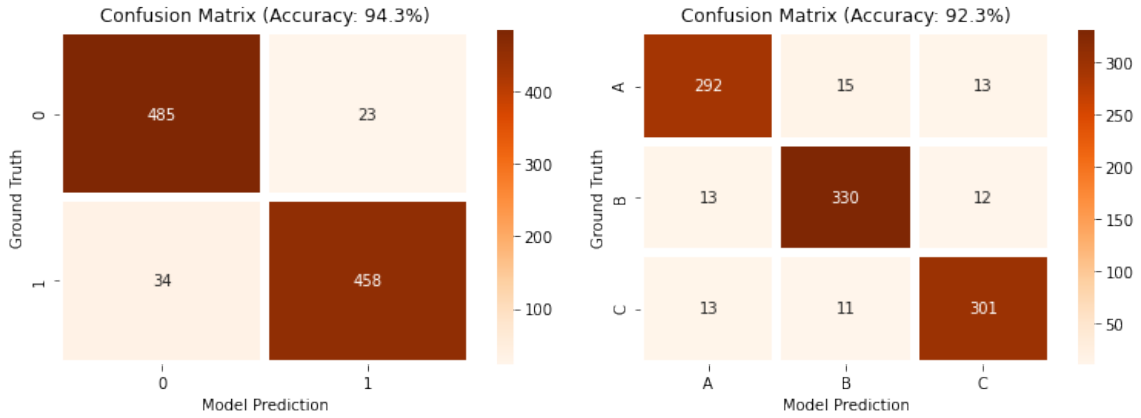
Table 5.1 shows the final model and hyperparameters for subtasks A and subtask B, and present the accuracy they can achieve in each subtask. The highest accuracy of the current project for subtask A and subtask B is 94.3% and 92.3%, respectively. However, these results may vary slightly each time the best implementation procedure is executed. This is because the model is retrained each time the project code executes. The training data obtained during the model training is randomly given by the data loader, which leads to the uncertainty of the training results to a certain extent. But in general, the project always achieved more than 90% accuracy. Moreover, the accuracy of subtask A will always be slightly higher than that of subtask B. This is because subtask A is a binary classification problem, while subtask B is a multi-classification problem.

	Subtask A	**Subtask B**
Model	Roberta-Large	Roberta-Large
Learning Rate	10^{-5}	10^{-5}
Batch Size	128	16
Epoch	15	10
Achieved Accuracy	94.3%	92.3%

Table 5.1: Final model and hyperparameters for best implementation.

Confusion matrix is a standard format for precision evaluation. It can intuitively visualize the prediction results of the model. In the accuracy evaluation, it is mainly used to compare the classification result with the actual measured value. Figure 5.9 shows the confusion matrix for the final prediction of best model and hyperparameters. The abscissa in the figure represents the predicted value of the model, and the ordinate represents the ground truth. So the number on the diagonal (from top left to bottom right) represents the number of correct predictions. Therefore, in subtask A, the number of correct predictions is $485+458 = 943$ (Figure 5.9(a)). While in subtask B, the number of correct predictions is $292+330+301 = 923$ (Figure 5.9(b)). The sum of all numbers in the figure is the total number of data in the test set, which is 1000. At the same time, we can

see that the numbers of wrong prediction in the two confusion matrices are almost symmetric. This suggests that the model's predictions are balanced, with no bias towards any options.



(a) Confusion Matrix of Subtask A

(b) Confusion Matrix of Subtask B

Figure 5.9: Confusion Matrix of Best Implementation

Finally, we can display what the model predicts. Figure 5.10 shows some examples of the model predictions. In Figure 5.10(a), two similar statements are presented along with the actual label value and the model prediction. In Figure 5.10(b), the false statement and three options are presented along with the ground truth and the model prediction. Although these two graphs only show a small fraction of the model's predictions, it can be seen that all of the predictions shown are correct, that is, all the values of model prediction are equal to the values of ground truth.

5.5.3 Manual Test

In the evaluation of subtask A, an additional performance evaluation method was added, which is manual testing. The test allows users to manually type in any two sentences, and the program determines which of the two sentences defies common sense. This means that these manually entered sentences have exceeded the scope of the test set, and are a true test of the performance of the model in practical applications. After training (or fine-tuning) the model with the best hyperparameters, Figure 5.11 shows the program's predictions for some custom input sentences. As it can be seen in the first round of manual test, the program recognized that "I am a program" was more against

	ID	Statement 0	Statement 1	Ground Truth	Model Prediction
0	1175	He loves to stroll at the park with his bed	He loves to stroll at the park with his dog.	0	0
1	452	The inverter was able to power the continent.	The inverter was able to power the house	0	0
2	275	The chef put extra lemons on the pizza.	The chef put extra mushrooms on the pizza.	0	0
3	869	sugar is used to make coffee sour	sugar is used to make coffee sweet	0	0
4	50	There are beautiful flowers here and there in ...	There are beautiful planes here and there in t...	1	1

(a) Examples of Final Prediction of Subtask A

	ID	False Statement	Option A	Option B	Option C	Ground Truth	Model Prediction
0	1175	He loves to stroll at the park with his bed	A bed is too heavy to carry with when strollin...	walking at a park is good for health	Some beds are big while some are smaller	A	A
1	452	The inverter was able to power the continent.	An inverter is smaller than a car	An inverter is incapable of powering an entire...	An inverter is rechargeable.	B	B
2	275	The chef put extra lemons on the pizza.	Many types of lemons are to sour to eat.	Lemons and pizzas are both usually round.	Lemons are not a pizza topping.	C	C
3	869	sugar is used to make coffee sour	sugar is white while coffee is brown	sugar can dissolve in the coffee	sugar usually is used as a sweetener	C	C
4	50	There are beautiful planes here and there in t...	A plane flies upon the garden	You can have a small garden in your private plane	A plane can never be seen in garden	C	C

(b) Examples of Final Prediction of Subtask B

Figure 5.10: Examples of Final Prediction

common sense than "I am a student." When I changed the sentence to say, "I am a student who can fly" and "I am a programmer who can program", the program again successfully judged that the first sentence is against common sense. This shows that the program made the prediction based on that "student can fly" was illogical, not just from words like "student" or "program". In the final round of manual testing, I changed the previous two sentences to logical statements, and then added time constraints to them. Again, the program decided that it was unreasonable to spend 26 hours a day doing one thing. To sum up, through manual testing of custom input, we can see the robustness of the program.

Out [18]:

```
Let's start our manual test.

Please input your first statement (e.g. I drink milk.):
I am a student
Please input your second statement (e.g. Milk drinks me.):
I am a program

Your input statememts are:
Statement 1:  I am a student
Statement 2:  I am a program

The model thinks the SECOND statement is against common sense.

Try again? y/n
y
-------------------------------------- 2 --------------------------------------
Let's start our manual test.

Please input your first statement (e.g. I drink milk.):
I am a student who can fly
Please input your second statement (e.g. Milk drinks me.):
I am a programmer who can program

Your input statememts are:
Statement 1:  I am a student who can fly
Statement 2:  I am a programmer who can program

The model thinks the FIRST statement is against common sense.

Try again? y/n
y
-------------------------------------- 3 --------------------------------------
Let's start our manual test.

Please input your first statement (e.g. I drink milk.):
I am a student who can study 5 hours a day
Please input your second statement (e.g. Milk drinks me.):
I am a programmer who can program 26 hours a day

Your input statememts are:
Statement 1:  I am a student who can study 5 hours a day
Statement 2:  I am a programmer who can program 26 hours a day

The model thinks the SECOND statement is against common sense.

Try again? y/n
n
```

Figure 5.11: Manual test for Subtask A

Chapter Six

Discussion

After the completion of the implementation of the project, a large number of experiments, determination of the best model and hyperparameters, and doing the best implementation, the project is basically formed. Looking at the final results and performance of this project, we can see the outstanding achievements of the project as well as its limitations. In this chapter, I will discuss the project's achievements and limitations. Particularly, I will present a new discovery I made in the project. At the same time, I will also discuss in-depth research that was not attempted in this project due to time constraints. Finally, I will look forward to the practical application of this task.

6.1 Achievements

First of all, the project successfully implemented "Task 4: Commonsense Validation and Explanation" proposed by Wang et al., 2020 in SemEval-2020, and solved subtask A and subtask B of the original task respectively. It explores and implements a solution to one of the natural language processing problems, which is commonsense understanding. To some extent, the model of this project shows the ability of common-sense reasoning.

Second, in the process of continuous optimization and experimentation of the project, the project's ability to solve target tasks has also been continuously improved. In both subtask A and subtask

B, project performance improved from nearly 80% accuracy in the first implementation to over 90% accuracy in the final best implementation (Table 5.1 and Figure 5.1, 5.9).

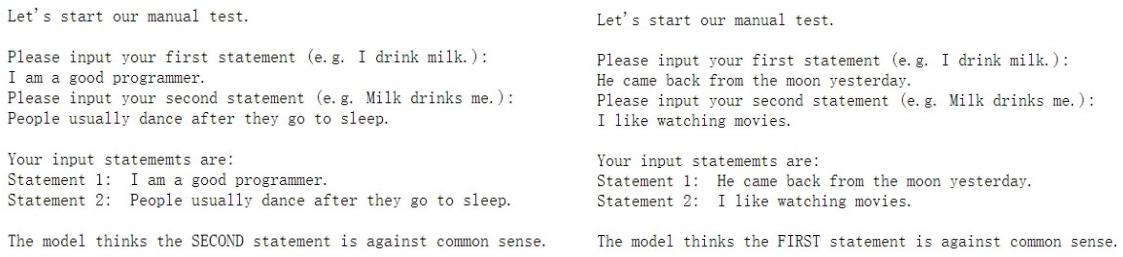
6.2 New Findings

In addition to the previous two achievements, more importantly, a new discovery was made during manual testing of the project. After the successful realization of the target task in this project, the manual test of subtask A shows that **the model trained by the project has the ability beyond the training target.**

In subtask A, the original goal of the training was to determine which of the two similar statements is against common sense. Therefore, no matter in the training dataset, validation dataset, or test dataset, the text given is two similar statements. However, in my homemade manual testing, the model allowed for arbitrary, unconstrained input. This means that the input statements do not have to be similar. Figure 6.1 shows two examples when I input unconstrained statements. Surprisingly, the model still showed excellent performance. In both cases, the model correctly selected the statement that defied common sense despite being given two completely dissimilar statements. However, this is not a unique case. In my subsequent manual tests, I tried to use a variety of statements with different structures, and the model was able to always make correct judgments. (Note that all manual test input data is still in the form of that one statement meets common sense and one statement does not. The difference is that the two statements no longer have similar structures.)

This finding suggests that the project model has capabilities beyond what it was originally trained for. Generally speaking, the performance of the model must be based on the training data presented during training procedure. The reason why the model can show such strong performance, in my personal understanding, the key point lies in the BERT model itself. Before fine-tuning for downstream tasks, the BERT model itself is a pre-trained model (Devlin et al., 2019). It has been pretrained on a large corpus of data with two objectives, which are Masked Language Modeling (MLM) and

Next Sentence Prediction (NSP). Thus, BERT may have had some logical understanding of natural language before it was exposed to the training dateset of downstream tasks (more specifically, my project task).



(a) Example 1 (b) Example 2

Figure 6.1: Manually test with unconstrained input

6.3 Limitations

Although the project model has achieved certain achievements and demonstrated strong performance, it still has the following limitations:

i) **The performance of the model training shows a certain degree of instability.**

On the one hand, in the successful implementations of model training, although the accuracy of model prediction can reach more than 90%, there will still be a fluctuation of 1%. I suggest the reason is that the data extracted for each batch during each training is random. This means that the data on the training set is used in a different order, resulting in a slight change in the effectiveness of the training. The accuracy of model predictions can vary by about 1%.

On the other hand, in the process of massive experiments, I can also find that under the same hyperparameters and model, sometimes the model will fail in training. I think the reason for the training failure may be that the selected batch size and learning rate are relatively large. These result in each epoch, the number of iterations is reduced, and the magnitude of the parameters updating is too large, respectively.

ii) **The models used are too large,** especially the selected final model Roberta-Large, which

makes model training very time consuming. Similarly, because the model is too large, the batch size for each training has to be sacrificed, and a smaller batch size was used to train the model. That's why I chose a batch size of 16 for subtask B. I think larger batch size should give better performance, however my GPU can no longer load larger batches for training.

iii) For the training of BERT and its variant models, **the requirements for hardware facilities are too high.** In this project, I have used a GPU with 24G of video memory, but it also shows a slow training speed and, as mentioned above, a small batch size. Again, the reason is that the model is too big. But it is more of a trade-off. It is because of this huge model that it has such powerful performance.

6.4 Further Research

Although this project has successfully completed the implementation of SemEval-2020 Task4, and conducted a lot of experiments on models and the values of hyperparameters, there are still many points of interest for this task that I can do further research. Next, I will list some in-depth research points that I am interested in:

i) **Further experiment on optimizer and loss function**

Due to the time constraints of the MSc project, I did not have enough time to experiment with the optimizer and loss function. In fact, this is one of the regrets of my project, as I have already included an interface to change the optimizer in my experiment code. What I originally envisioned was that I could try to use different optimizers, such as SGD (stochastic gradient descent, by Robbins and Monro, 1951), SGD with Momentum (Sutskever et al., 2013) and Adagrad (Duchi, Hazan, and Singer, 2011), to compare the effects of model training. As for the loss function, this part of the experiment will require more code modifications. However, I believe that choosing the appropriate loss function may bring better training effect to the model.

ii) **Pre-training with external common-sense repositories**

Comparing other implementation works of SemEval-2020 Task4, there are still at least 5 teams on the leaderboard 2.1 that have a higher accuracy than my project. Most of these implementations use model architectures similar to mine (BERT or its variants). So according to my analysis, other implementations can have higher accuracy, possibly because they did additional pre-training. If there is enough time, I plan to perform one more round of pre-training on my best model on an external common-sense repositories, e.g. ConceptNet (Speer, Chin, and Havasi, 2017).

iii) **Explore the extensibility of the model**

As discussed in previous Section 6.2 (*New Findings*), the model exhibits capabilities beyond original training objectives. Further research on the extensibility of the model is a very interesting topic. This means that the model is more versatile for solving different goals. Similarly, this will greatly increase the efficiency of training and the practical ability of the model. In my opinion, there are two research points in this direction: (1) Further explore the reasons why models show abilities beyond training expectations. (2) In subtask A, whether the model can make commonsense judgments based on only one input statement.

6.5 Application

Although the commonsense reasoning task studied in this project still has many directions and topics that need further research, from the perspective of its achievable accuracy, the work of this project can be applied to many research directions and real life.

- Applied to other research

  In the general research of natural language processing, there are many specific research directions, such as speech recognition, natural language generation, machine translation, question answering system and so on. **Possessing the ability of commonsense reasoning, it can assist in the realization of other research tasks.** For example, in speech recognition tasks, when there are obvious commonsense errors in a recognized sentence, it can be marked

as recognition failure and then re-recognition. Similarly, in generation tasks, such as machine translation and question answering, when the generated sentence violates common sense, the model can be asked to regenerate the sentence. In short, the ability of commonsense reasoning can be used in other research tasks to determine whether the generated sentences are in line with common sense.

- Applications in real life

  Although the commonsense reasoning studied in this project is still in the academic research stage, it is not difficult to expect that a complete commonsense reasoning project will have many applications in practical life. For example, in the existing artificial intelligence products, whether Apple's Siri can provide users with more logical answers and more complete functions through commonsense reasoning. Similarly, in the lives of non-native speakers, whether translation software can provide more powerful and accurate translation functions through commonsense reasoning.

- Applications in future work

  Not only the current life, when looking forward to the life of the artificial intelligence era in the future, commonsense reasoning will be a necessary foundational technology. The function of commonsense reasoning can provide a better language system for intelligent robots. At the same time, when the ability of commonsense reasoning rises to the ability of logical reasoning, the police can apply this set of technology to investigate cases, and ordinary people can identify fake news and false propaganda through this technology.

Chapter Seven

Conclusion

In this project, I successfully implemented the program for the first two subtasks of *SemEval 2020 Task 4 - Commonsense Validation and Explanation (ComVE)*. By solving this task, the project model demonstrated the ability of commonsense reasoning. In addition, I did large numbers of experiments to study the influence of different models and hyperparameters on the program performance. Through experiments, I improved the performance of this project to over 90% accuracy. Last but not least, I analyzed, evaluated and discussed the experiment results, and described the achievements and limitations of this project respectively. Meanwhile, the possible further research of this task and its application are discussed as well.

Regarding the final implementation and performance of the two subtasks, both subtasks adopt the Roberta-Large model, and choose AdamW as the optimizer and CrossEntropy as the loss function. In subtask A, by setting the learning rate as 10^{-5}, batch size as 128, and epoch as 15, the highest accuracy of 94.3% was obtained on the unseen test set. while in subtask B, by setting the learning rate as 10^{-5}, batch size as 16, and epoch as 10, the highest accuracy of 92.3% was finally obtained.

The two highlights of this project are: 1. A large number of experiments and analyses have been carried out on the model and hyperparameters. 2. In the manual test, the model showed capabilities that beyond training expectations. I am excited about the experiment results and the final performance of this project. In my future research career, I will continue to do more in-depth research

on commonsense reasoning and aim to make a contribution to the research of natural language processing.

References

Davis, Ernest (2017). "Logical formalizations of commonsense reasoning: a survey". In: *Journal of Artificial Intelligence Research* 59, pp. 651–723 (cit. on p. 1).

Davis, Ernest and Gary Marcus (Aug. 2015). "Commonsense Reasoning and Commonsense Knowledge in Artificial Intelligence". In: *Commun. ACM* 58.9, pp. 92–103. ISSN: 0001-0782. DOI: 10.1145/2701413. URL: https://doi.org/10.1145/2701413 (cit. on p. 1).

Devlin, Jacob et al. (June 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423 (cit. on pp. 1, 4, 6, 8, 12, 15, 19, 35).

Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61, pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html (cit. on p. 37).

Lan, Zhenzhong et al. (2020). "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=H1eA7AEtvS (cit. on p. 4).

Liu, Weijie et al. (Apr. 2020). "K-BERT: Enabling Language Representation with Knowledge Graph". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.03, pp. 2901–

2908. DOI: 10.1609/aaai.v34i03.5681. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5681 (cit. on p. 4).

Liu, Yinhan et al. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv: 1907.11692 [cs.CL] (cit. on pp. 4, 6, 12, 15, 19).

Loshchilov, Ilya and Frank Hutter (2019). *Decoupled Weight Decay Regularization*. arXiv: 1711.05101 [cs.LG] (cit. on p. 12).

Mohammed, Roweida and Malak Abdullah (Dec. 2020). "TeamJUST at SemEval-2020 Task 4: Commonsense Validation and Explanation Using Ensembling Techniques". In: *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. Barcelona (online): International Committee for Computational Linguistics, pp. 594–600. URL: https://aclanthology.org/2020.semeval-1.75 (cit. on p. 4).

Pai, Liu (Dec. 2020). "QiaoNing at SemEval-2020 Task 4: Commonsense Validation and Explanation System Based on Ensemble of Language Model". In: *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. Barcelona (online): International Committee for Computational Linguistics, pp. 415–421. URL: https://aclanthology.org/2020.semeval-1.50 (cit. on p. 4).

Robbins, Herbert and Sutton Monro (1951). "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3, pp. 400–407. ISSN: 00034851. URL: http://www.jstor.org/stable/2236626 (cit. on p. 37).

Saeedi, Sirwe et al. (Dec. 2020). "CS-NLP Team at SemEval-2020 Task 4: Evaluation of State-of-the-art NLP Deep Learning Architectures on Commonsense Reasoning Task". In: *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. Barcelona (online): International Committee for Computational Linguistics, pp. 507–515. URL: https://aclanthology.org/2020.semeval-1.62 (cit. on p. 4).

Sanh, Victor et al. (2020). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. arXiv: 1910.01108 [cs.CL] (cit. on pp. 6, 12, 15, 19).

Singh, Push et al. (2002). "Open Mind Common Sense: Knowledge Acquisition from the General Public". In: *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE.*

Ed. by Robert Meersman and Zahir Tari. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1223–1237. ISBN: 978-3-540-36124-4 (cit. on p. 4).

Speer, Robyn, Joshua Chin, and Catherine Havasi (Feb. 2017). "ConceptNet 5.5: An Open Multilingual Graph of General Knowledge". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 31.1. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11164 (cit. on pp. 4, 38).

Srivastava, Vertika et al. (Dec. 2020). "Team Solomon at SemEval-2020 Task 4: Be Reasonable: Exploiting Large-scale Language Models for Commonsense Reasoning". In: *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. Barcelona (online): International Committee for Computational Linguistics, pp. 585–593. URL: https://aclanthology.org/2020.semeval-1.74 (cit. on p. 4).

Sutskever, Ilya et al. (June 2013). "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, pp. 1139–1147. URL: https://proceedings.mlr.press/v28/sutskever13.html (cit. on p. 37).

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on p. 6).

Wan, Jiajing and Xinting Huang (Dec. 2020). "KaLM at SemEval-2020 Task 4: Knowledge-aware Language Models for Comprehension and Generation". In: *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. Barcelona (online): International Committee for Computational Linguistics, pp. 543–550. URL: https://aclanthology.org/2020.semeval-1.67 (cit. on p. 4).

Wang, Cunxiang et al. (Dec. 2020). "SemEval-2020 Task 4: Commonsense Validation and Explanation". In: *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. Barcelona (online): International Committee for Computational Linguistics, pp. 307–321. URL: https://aclanthology.org/2020.semeval-1.39 (cit. on pp. 1, 4, 5, 10, 29, 34).

Wolf, Thomas et al. (Oct. 2020). "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://aclanthology.org/2020.emnlp-demos.6 (cit. on pp. 11, 12).

Xing, Luxi et al. (Dec. 2020). "IIE-NLP-NUT at SemEval-2020 Task 4: Guiding PLM with Prompt Template Reconstruction Strategy for ComVE". In: *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. Barcelona (online): International Committee for Computational Linguistics, pp. 346–353. URL: https://aclanthology.org/2020.semeval-1.42 (cit. on p. 4).

Yang, Zhilin et al. (2019). "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf (cit. on p. 4).

Zhang, Yice et al. (Dec. 2020). "CN-HIT-IT.NLP at SemEval-2020 Task 4: Enhanced Language Representation with Multiple Knowledge Triples". In: *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. Barcelona (online): International Committee for Computational Linguistics, pp. 494–500. URL: https://aclanthology.org/2020.semeval-1.60 (cit. on p. 4).

Zhao, Qian et al. (Dec. 2020). "ECNU-SenseMaker at SemEval-2020 Task 4: Leveraging Heterogeneous Knowledge Resources for Commonsense Validation and Explanation". In: *Proceedings of the Fourteenth Workshop on Semantic Evaluation*. Barcelona (online): International Committee for Computational Linguistics, pp. 401–410. URL: https://aclanthology.org/2020.semeval-1.48 (cit. on p. 4).

Appendix: Project Code

Git Repository: https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2020/lxl065

MSc Project (Commonsense Validation and Explanation in Natural Language Processing)

Instructions

Python 3.7 with PyTorch 1.9.0 and Transformers 4.9.1 serve as the backbones of this project.

The code is using one GPU by default, you have to modify the code to make it running on CPU or multiple GPUs.

1. Envs and Dependencies

First, to implement this work, you need to download and install Anaconda3. Here is the official link for Anaconda:
https://www.anaconda.com/products/individual-d
After installing the appropriate version of Anaconda3, please execute the following command block on the console. (The example below is based on the Linux operating system.)

```
# Create a conda envs
conda create -n msc_project_2214560 python=3.7 ipython
conda activate msc_project_2214560

# Install PyTorch
# Please Note: Install the appropriate version of PyTorch, and be aware of your operating system and CUDA version.
# The following command is for the PyTorch installation of Linux operating system and CUDA 10.2.
pip install torch torchvision torchaudio

# Install HuggingFace Transformers
pip install transformers==4.9.1

# Install other libraries
pip install TensorBoard==2.6.0
pip install jupyter==1.0.0
pip install pandas==1.2.4
pip install matplotlib==3.3.4
pip install seaborn==0.11.1
```

2. Viewing and Running the Project Code

After installing the project environment, you can run Jupyter Notebook to view and run the project.

3. The Directory Tree of Project

```
MSc Project (Commonsense Validation and Explanation in Natural Language Processing)
├── MSc Project
│   ├── Best Implementation
│   │   ├── Subtask_A.ipynb -------------------------- This is the best implementation for subtask A.
│   │   └── Subtask_B.ipynb -------------------------- This is the best implementation for subtask B.
│   ├── DataSet -------------------------------------- This is the DataSet of the SemEval-2020 Task 4.
│   └── Experiment
│       ├── ExperimentResult_A ----------------------- This keeps the experiment results of subtask A.
│       ├── ExperimentResult_B ----------------------- This keeps the experiment results of subtask B.
│       ├── TensorBoard ------------------------------ This keeps the TensorBoard records.
│       ├── Experiment Workflow.ipynb ---------------- This shows the experiment workflow and generates the result figures.
│       ├── Experiment_Subtask_A.ipynb --------------- This is the experiment code for subtask A.
│       └── Experiment_Subtask_B.ipynb --------------- This is the experiment code for subtask B.
└── README.md
```