Final Iteration Report

Team name: Hooli

Minghao Li (ml4025), Yihan Lin (yl3820), Yihao Li (yl3744), Yiming Sun(ys3031)

Part 0: Language, Platform and Technologies

We would like to choose Python as our language, and Mac OS as our developing environment. In addition, we would like to integrate multiple cutting-edge technologies into our project, including but not limited to Tensorflow, Keras, and various AWS services, such as Elastic Beanstalk, API Gateway, DynamoDB, S3, Cognito, etc., If we have enough time to do so.

Github Link: https://github.com/ysun647/Nucleus

The version for the second iteration was tagged as v2.0.

Part 1: Project Summary

SQuAD challenge, raised by Stanford University is a famous and popular reading comprehension problem. The target of SQuAD challenge, is to train a neural network that is able to answer a specific question, given a passage from Wikipedia which contains the answer of this question, or to output that this question is not answerable, if the given passage does not contains the answer. Stanford University provides training datasets to the public, and researchers all over the world are trying the best on their networks to approaching the human's performance - it's almost there.

In our project, we are planning to do three cool things related to SQuAD; ~~first, design a web crawler to collect data from Wikipedia as the training and testing dataset;~~ second, train and test a deep neural network based on the SQuAD challenge; third, if we have enough time, we will try to design a web application and deploy it based on Amazon Web Service, and thus allow our users to test our network and submit feedbacks.

Main updates since second iteration:

1. Enhanced our deep neural model: instead of r-net, now we are using BERT - a far more powerful network;
2. Added the new feature that queries the asking history. We saved every asking message including question and context. When the user queries the history we will show the user the most recent several question-asking activities.
3. At context-free mode, instead of putting whole wikipedia summary as input, we divided it into several paragraphs, the lengths of which are similar to the length of each training data point.
4. We imported a frontend framework that looks better.
5. User can submit review and feedback.

Part 2: Some User Stories

1. General Web Surfer: As a guy surfing the Internet, I want an answer of a specific question based on the web pages I am looking through within several seconds so that it saves my time on searching for the answer. My conditions of satisfaction are that it could output the answer within 3 seconds, or output “no answer” if the answer does not exist.

2. Students: As a student who suffers from difficulty in reading, I want to extract the key information in some long articles. My conditions of satisfaction are I can get the short answer immediately avoid reading the whole article.

3. Developers: As a developer, I want to pinpoint my technical issue inside a technical documentation so that I don’t need to look through all of it. My conditions of satisfaction are that I can find what I want immediately."

Part 3: User-level Test Cases

A user of our product may

- Sign up a new account;
- Sign in to an existing account;
- Type in an article and ask a related question;
- Select an article and ask a related question;
- Ask a question, whose answer cannot be found in the article;
- Input a meaningless message or a meaningless question;
- Sign out the currently signed-in account;
- See history
- Submit feedback and review after a question-asking activity.
- Ask a question without providing context.

Workflow Exceptions (error cases)

- No account / password incorrect;
- Empty question or article;
- Irrelevant question;
- Question is too long;
- Article is too long;

Part 4: Use Cases

In this part we show some use cases that is expanded or merged from the user stories listed above.

Use case 1

1. **Title**: Account signing up
2. **Actor(s)**: regular users
3. **Descriptions**: Regular users can sign up an account to access the functionality offered by our website. Username and password format as well as verification code are required. Automatically logged on after successful registration.
4. Triggers: Regular users click “sign up” in the welcome page.
5. Preconditions: the regular user is not registered.
6. Guarantees: the regular user becomes a register player and automatically logged in.
7. **Basic flows**:
    a. System provides several blank editor area asking user to fill in registration information.
    b. Regular user types in the required information: username, password, email address.
    c. The regular user clicks the signup button to confirm and proceed to next steps.
    d. System calls cognito module to check if the username has existed or not.
    e. System validates the password in view of length is longer than 8 as well as special symbols; email in view of the email format.
    f. The system sends verification code to user’s email and redirects the user to verification page.
    g. User types the verification code into the required blank.
    h. The system sets up connection between mysql server on Amazon RDS to store the same user information.
    i. The user becomes a registered one and automatically signed in.
8. **Alternate flows**:
   D1. The username has already existed. The system notifies the user with a message: ‘username already existed.’
   E1. The password length is smaller than 8. System notifies the user with a message: ‘password too short!’
   E2. The password or usernames contains some special symbols. System redirects to an error page which display the errors to be corrected.
   E3. The email address has invalid formats. System redirects to an error page which display the errors to be corrected.
   G1. The verification code is incorrect. System prints out message: ‘unable to verify, please try again.’’

Use case 2

1. **Title**: Account signing in
2. **Actor(s)**: Registered users
3. **Descriptions**: Registered user can sign in an existed account to access the functionality offered by our website.

4. Triggers: User click “LOGIN” in the login page.
5. Precondition: The user is not yet logged in.
6. **Basic flows**:
    a. System provides several blank editor area asking user to fill in login information.
    b. User types his/her user name;
    c. User types his/her password;
    d. System checks if the given login parameters are valid;
    e. System creates a new session for the user.
7. **Alternate flows**:
   c1. The user name that user provides is not registered -> login failed -> System informs the user that he/she typed a non-registered user name and redirects to the login page.
   c1. The password that user provides is incorrect -> login failed -> System informs the User that he/she typed an incorrect password and redirects to the login page.
8. Success guarantees: The user can access the main page to take further operations.

Use case 3

1. **Title**: Account asking general question
2. **Actor(s)**: Registered users
3. **Descriptions**: A user logs in and asks a general question, without providing a related context. In this case, we call the API of wikipedia to find the potential answer.
4. **Basic flows**:
    a. The user logs in to Nucleus system.
    b. The user clicks “I want to ask a questions without context”
    c. The user input a question, such as “Where is New York City”, and clicked “submit”.
    d. The system calls the wikipedia API, try to find the basic introduction of “New York City”, and put this article to our neural model.
    e. The pre-trained neural model use the article provided by wikipedia API and the question from the user as input, and then suggests a possible answer.
    f. The user’s action of asking question is recorded and sent to our database on Amazon RDS.
    g. The user see the suggested answer on the webpage.
    h. The user choose the score as feedback and the score is saved on the database.
    i. The user clicks “go back to main page”
    j. The user logs out.
5. **Alternate flows**:
   c1. The user provided a question that contains an ambiguous keyword. For example, When the user asks “Where is New York?”, “New York” may refer to New York City or New York State. In this case we will try to find answer in both the wikipedia page of “New York City” and “New York State”. And we will use different contexts and run several time to get the answer and confidence. Among the answers we will choose the one with highest confidence.

c2. The user may input an empty question. In this case, we will suggest a warning saying that empty questions are not allowed.
h1. The user may close the webpage directly without logging out. In this case, we will still maintain the session for a short time until the http connection is time out, thus we provide convenience in case when a user come back in a short time - this user does not need to log in again.

Use case 4

1. **Title**: Account asking questions and providing context
2. **Actor(s)**: Registered users
3. **Descriptions**: A user logs in and asks a question providing a related context. In this case, we put the context and question, expecting a correct answer extracted from the context, None if no answer found.
4. **Basic flows**:
    a. The user logs in to Nucleus system.
    b. The user clicks “I want to ask a questions with context”
    c. The user inputs a question, such as “Where is New York City”, provided a context, and clicked “submit”.
    d. The pre-trained neural model uses the article provided by wikipedia API and the question from the user as input, and then suggests a possible answer.
    e. The user’s action of asking question is recorded and sent to our database on Amazon RDS.
    f. The user see the suggested answer on the webpage.
    g. The user choose the score as feedback and the score is saved on the database.
    h. The user clicks “go back to main page”
    i. The user logs out.
5. **Alternate flows**:
    c1. The user may provide an unrelated context, which does not contain the answer to the question. In this case, Nucleus suggests “no answer” to the user.
    c2. The user may input an empty question. In this case, we will suggest a warning saying that empty questions are not allowed.
    h1. The user may close the webpage directly without logging out. In this case, we will still maintain the session for a short time until the http connection is time out, thus we provide convenience in case when a user come back in a short time - this user does not need to log in again.

Use case 5

6. **Title**: Account query the history
7. **Actor(s)**: Registered users
8. **Descriptions**: A user logged in and query the history.
9. **Basic flows**:

    a. The user logs in to Nucleus system.
    b. The user clicks “My asking history”
    c. The user sees the history on the webpage.
    d. The user clicks “go back to main page”
    e. The user logs out.

10. **Alternate flows**:
    c1. The new user may don’t have any asking history. In this case, we will show the default message such as “You haven’t ask anything yet.”.
    c2. If (not too possible) a user ask so many questions such as more than 10000. In this case, we will only show him the first 100.

Part 5: Test Suites

Test suite of neural model of R-net

We divide all the test cases of neural models into the following test cases. All of the test cases are in the <root>/testcases directory.

1. Meaningful questions and contexts. This suite includes test cases composed of meaningful questions, meaningful contexts, and the answer of this question is in the contexts.
2. Meaningless questions and contexts. This suite includes test cases in which the questions are not related to the context; that is, we cannot find the answer related to the question.
3. Nonsense questions and contexts. This suite includes test cases that does not make any sense.
4. Explainable nonsense. This suite includes test cases that contains some nonsense word, but it is explainable anyway. For example, say the context contains a sentence “the rf8gnv9wgtcw is the governor of State Cderefbuer”. Despite that “rf8gnv9wgtcw” does not make any sense, it is explainable anyway.
5. Injection attack. This suite includes test cases that may be related to injection attacks, such as “sudo rm -rf /” or “drop database”

Test suite of neural model of bert

We provided bert unittest cases in <root>/test/test_bert.py to test the functionality of <root>/models/bert/inference_bert.py

6. We input a valid test case, which is actually a dictionary with question and contxt list.
7. We input some invalid inputs like missing question, missing context, or opposite position. All gets wrong answer.

Test suite of database

For the database unit tests which are located in the file <root>/test/test_database.py, we originally have 5 methods in database class (the last one which is called get_history_list has not been fully developed yet so no test cases for that). We design valid and invalid test cases for each methods, as described below. From the second iteration, we perfected database class by adding user_feedback method.

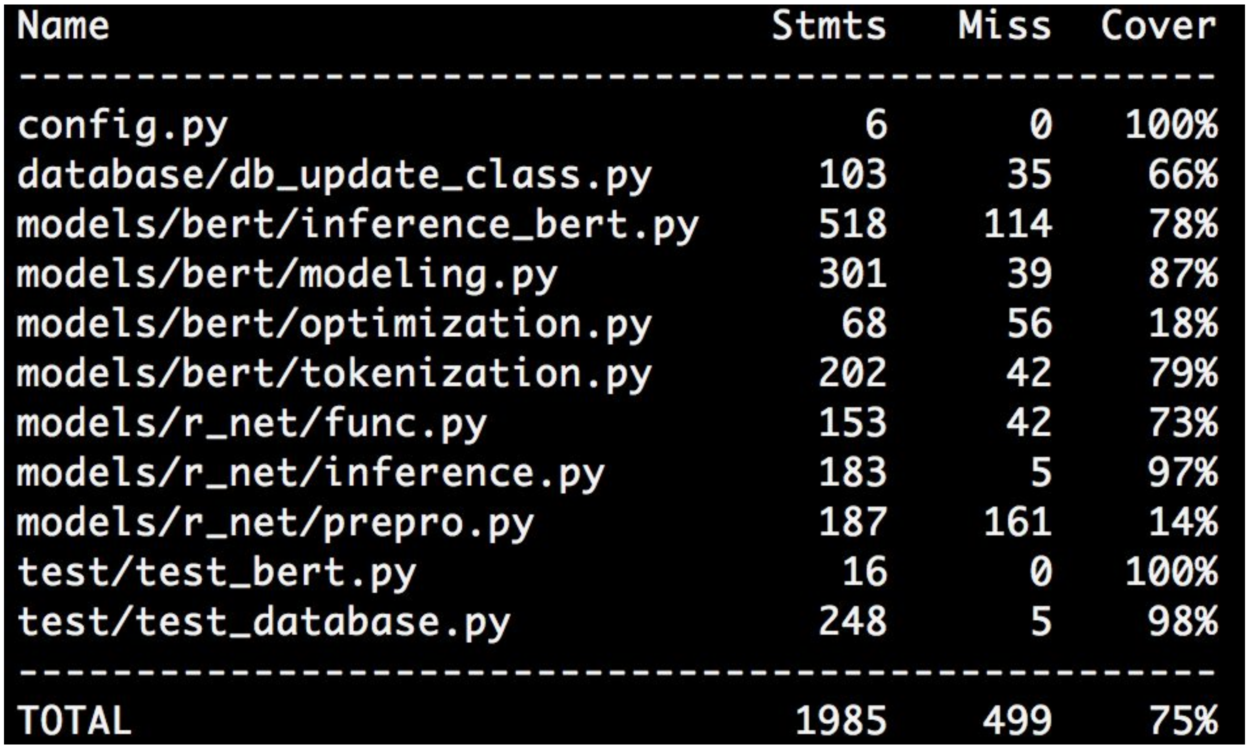
1. Valid input cases with all the required parameters. Specifically, we pass in
    1. article content and article title for add_article method;
    2. user_name, user_password, user_email for add_user method;
    3. art_id, q_content for add_question method;
    4. U_id, q_id for add_history method;
    5. User_id, title, content, question for update method
    6. Username, question, answer, satisfaction, expected for user_feedback method.
2. Invalid input test cases with some required parameters missing or wrong formatted
    1. Missing article content or missing content title for add_article method;
    2. Long titles for add_article method;
    3. Missing user_name, user_password or user_email for add_user method;
    4. Missing u_id or q_id for add_history method;
    5. Bound conditions (0 index) for add_history method;
    6. Negative index for add_history method;
    7. Missing art_id or q_content for add_question method;
    8. Negative article id or string id for add_question method;
    9. Long questions for add_question method;
    10. Missing User_id, title, content or question for update method;
    11. Long input for update method.
    12. Missing username, question, answer for user_feedback method.

Part 6: Coverage Test

We are using package `coverage` to see the coverage of our test cases. The commands about it is stored in our repo's following path:

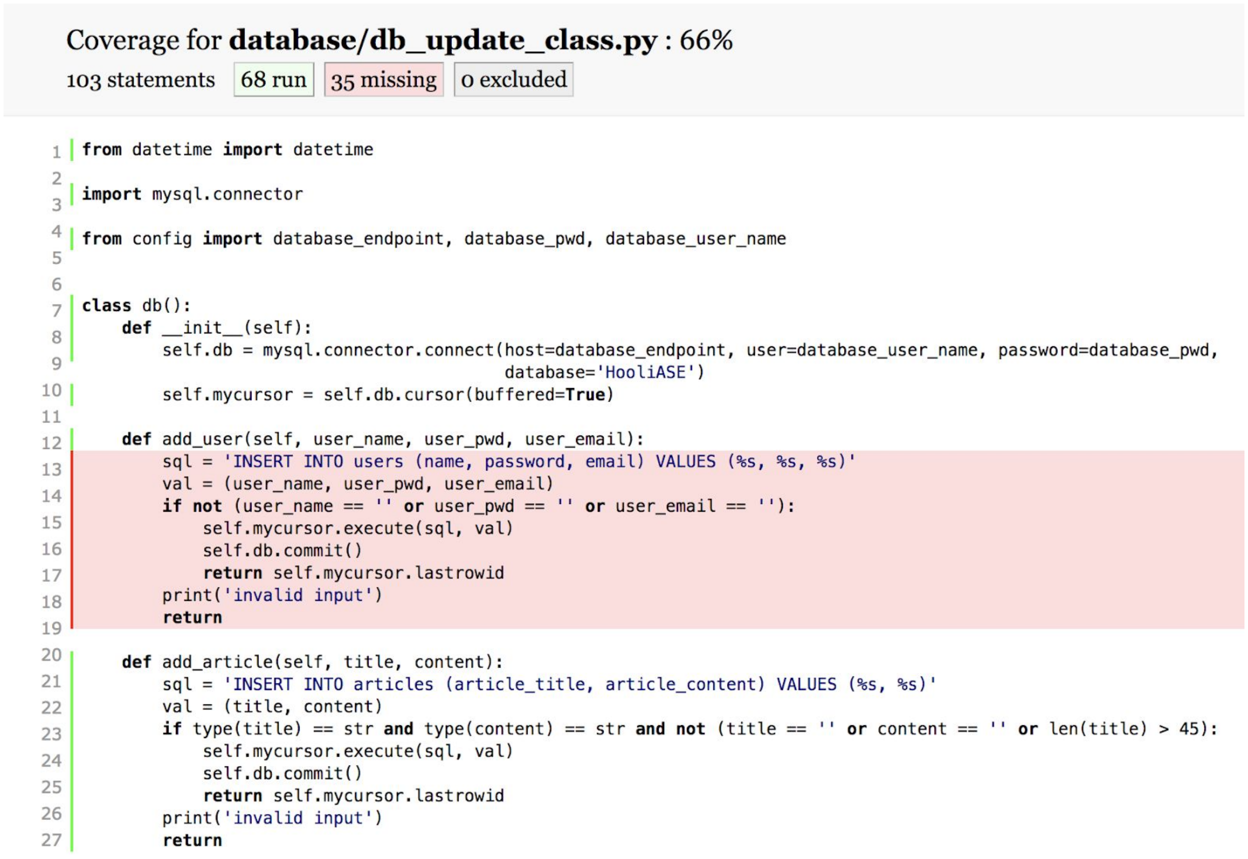
<root>/results/post-commit-build-shell.txt

The results are as follows as well as in the repository (<root>/coverage_report.txt):



```
Name                                Stmts   Miss  Cover
-------------------------------------------------------
config.py                               6      0   100%
database/db_update_class.py           103     35    66%
models/bert/inference_bert.py         518    114    78%
models/bert/modeling.py               301     39    87%
models/bert/optimization.py            68     56    18%
models/bert/tokenization.py           202     42    79%
models/r_net/func.py                  153     42    73%
models/r_net/inference.py             183      5    97%
models/r_net/prepro.py                187    161    14%
test/test_bert.py                      16      0   100%
test/test_database.py                 248      5    98%
-------------------------------------------------------
TOTAL                                1985    499    75%
```

You can also download the coverage_report_html folder and see exactly which line is not covered in index.html.



Coverage for **database/db_update_class.py** : 66%

103 statements 68 run 35 missing 0 excluded

```
from datetime import datetime

import mysql.connector

from config import database_endpoint, database_pwd, database_user_name


class db():
    def __init__(self):
        self.db = mysql.connector.connect(host=database_endpoint, user=database_user_name, password=database_pwd,
                                          database='HooliASE')
        self.mycursor = self.db.cursor(buffered=True)

    def add_user(self, user_name, user_pwd, user_email):
        sql = 'INSERT INTO users (name, password, email) VALUES (%s, %s, %s)'
        val = (user_name, user_pwd, user_email)
        if not (user_name == '' or user_pwd == '' or user_email == ''):
            self.mycursor.execute(sql, val)
            self.db.commit()
            return self.mycursor.lastrowid
        print('invalid input')
        return

    def add_article(self, title, content):
        sql = 'INSERT INTO articles (article_title, article_content) VALUES (%s, %s)'
        val = (title, content)
        if type(title) == str and type(content) == str and not (title == '' or content == '' or len(title) > 45):
            self.mycursor.execute(sql, val)
            self.db.commit()
            return self.mycursor.lastrowid
        print('invalid input')
        return
```

Basically, our branch coverage rate is about 75% for several reasons:

1. Some of the modules are not covered in this test, because these modules are used for training the model, not for implementing the whole system. When Nucleus is launched, these modules will not be called. This is why 161 lines are missing from prepro.py and 42 lines are missing from func.py. As well, for the same reason, we didn't achieve 100% for bert module.
2. There is one extension database function in db_update_class.py, which is still under development. So we haven't written any test cases for this method. This is why we have 35 lines missing in this file.
3. We didn't take the front end (flask based) code coverage into consideration according to IA's suggestion.