



TITLE:

Create a chatbot – Using Python, Recurrent Neural Networks and TensorFlow

CANDIDATE NUMBER(S):

10007

DATE: 30/11-18	SUBJECT CODE: IE303612	SUBJECT: Bachelor thesis, Data	DOCUMENT ACCESS:
STUDY: Computer Engineering		NO. PAGES/ATTACHMENTS: 48 / 1	BIBL. NR:

SUPERVISOR(S):

Ibrahim A. Hameed

SUMMARY:

The purpose of this thesis is to explain how to create a general-purpose chatbot using Python, TensorFlow and Recurrent Neural Networks.

This thesis also discusses the advantages and disadvantages of different algorithms, use of different amount of layers, parameters etc. for creating a chatbot.

This thesis was written by one student at NTNU in Ålesund

PREFACE

At the time when I read the problem description of the thesis, I did not have any knowledge of Neural networks and recurrent neural networks. Neither any knowledge of TensorFlow or Python. But I thought that the creation of a chatbot sounded like a very fascinating topic. And the creation of chatbots was something I would have liked to be working on after the time of studying.

My goal of the thesis was to find advantages and disadvantages of different types of training data, number of layers used in the NMT-chatbot (NMT - Neural machine translation). One more thing I wanted to look for was the advantages and disadvantages of different types of algorithms used in a chatbot.

My advisor was for this project: Ibrahim A. Hameed. I would like to thank him for providing me with material for writing this thesis and for telling me how I could create a chatbot. Also, I would like to thank two master students, Waleed Elzoghby and Saumitra Dwivedi, for very useful discussions about the thesis. They also provided me with useful materials about theory that is related to creating a chatbot.

ABSTRACT

A lot of people are hired in today's society to work as customer support to help out with software issues that the customers might have at a given time. Companies need to pay a lot of money to these employees in terms of wages for this type of job. All companies globally have the desire to save money spent for wages, to spend this money for other purposes which benefits the company. This money could for example be used for safe guards to protect against cyber-attacks which may happen from time to time.

This is where a chatbot comes in. A chatbot can be trained with different datasets depending on the given job it is supposed to do. When the chatbot has been trained with a dataset once, it doesn't have to be trained once again with the exact same dataset. It does actually recall the questions and answers provided earlier. So the chatbot can be given questions and answers found in for example a file with the extension ".yaml" or ".json", where each file represents questions and answers within a separate topic.

A few widely popular applications already use this type of service along with a chat interface to make it easy for potential customers to ask about a specific product or when the shop opens or closes. The two most popular applications where a chatbot service is being used are Facebook Messenger and text messages.

The result of the thesis is talk about the basic things that a chatbot is built upon, like recurrent neural networks, bidirectional neural networks, etc. My own chatbot did not end up working like it should have been doing, but the thesis will provide some sources of errors that may exist for my creation of chatbot. Some suggestions for better datasets are included in this thesis.

TERMINOLOGY

Abbreviations

JSON	<JSON info>
SQL	<SQL info>
NMT	Neural Machine Translation
RNN	Recurrent Neural Networks
BRNN	Bidirectional Recurrent Neural Networks
SWOT	Analysis of a system's Strengths, Weaknesses, Opportunities and Threats

CONTENTS

SUMMARY/ABSTRACT	4
TERMINOLOGY	5
CONCEPTS	FEIL! BOKMERKE ER IKKE DEFINERT.
NOTATION	FEIL! BOKMERKE ER IKKE DEFINERT.
SYMBOLS	FEIL! BOKMERKE ER IKKE DEFINERT.
ABBREVIATIONS	5
1 INTRODUCTION	8
2 THEORETICAL BASIS	9
2.1 BACKGROUND & HISTORY	9
2.1.1 <i>What is a chatbot?</i>	9
2.1.2 <i>Chatbot history</i>	9
2.2 NEURAL NETWORKS	10
2.3 RECURRENT NEURAL NETWORKS	10
2.3.1 <i>How to implement a Recurral Neural Network</i>	Feil! Bokmerke er ikke definert.
2.4 BIDIRECTIONAL RECURRENT NEURAL NETWORKS	11
2.5 SEQUENCE TO SEQUENCE MODELS FOR CHATBOTS	11
2.6 ATTENTION MECHANISMS	12
2.7 USEFUL THINGS TO KNOW ABOUT NMT-CHATBOT	12
2.7.1 <i>Introduction to the NMT-chatbot project</i>	12
2.7.2 <i>Standard vs BPE/WPM-like (subword) tokenization, embedded detokenizer</i>	12
2.7.3 <i>More detailed information about training a model</i>	13
2.7.4 <i>Utils</i>	14
2.7.5 <i>Inference</i>	14
2.7.6 <i>How to import nmt-chatbot</i>	15
2.7.7 <i>How to deploy chatbot/model</i>	16
2.7.8 <i>Demo chatbot</i>	16
3 MATERIALS AND METHOD	17
3.1 METHOD	17
3.1.1 <i>Acquiring training data for the chatbot</i>	17
3.1.2 <i>Chat Data Structure</i>	18
3.1.3 <i>Buffering through the data</i>	20
3.1.4 <i>Insert Logic</i>	22
3.1.5 <i>Building database</i>	25
3.1.6 <i>Training Dataset</i>	30
3.1.7 <i>Training a model</i>	32
3.1.8 <i>Interacting with the chatbot</i>	35
3.2 MATERIALS	39
3.2.1 <i>Python</i>	39
3.2.2 <i>PyCharm</i>	39
3.3 LIBRARIES AND FRAMEWORKS	39
3.3.1 <i>TensorFlow</i>	39
3.3.2 <i>tqdm</i>	39
3.3.3 <i>colorama</i>	40
3.3.4 <i>Regex</i>	40
3.3.5 <i>Python-levenhstein</i>	40
3.3.6 <i>NumPy</i>	41
3.3.7 <i>Requests</i>	41
4 RESULTS	42
5 DISCUSSION	44
5.1 REVIEW OF THE PRACTICAL WORK OF THE PROJECT	46
5.2 REVIEW OF THE PROJECT AS A WHOLE	FEIL! BOKMERKE ER IKKE DEFINERT.
6 REFERENCES	47
ATTACHMENTS	48

1 INTRODUCTION

This bachelor thesis was written at NTNU in Ålesund by one student. From discussions with my advisor Ibrahim along with two master students who worked on a similar project before, I got ideas of which topics I could be writing about in this thesis. These people inspired me to write this bachelor thesis.

The goal of the thesis is to describe the core building components of the chatbot, about RNNs, the programming language used etc.

I expect the reader to have some type of knowledge about chatbots or about programming in general before reading this text.

1.1 Limitations

This thesis only talks about the basics of a chatbot, what a chatbot is, about the components and algorithms with the bot etc. It does not go into great detail.

2 THEORETICAL BASIS

2.1 Background & History

This section will tell some information about chatbots and the history of some popular chatbots that already exist on the market.

2.1.1 What is a chatbot?

A chatbot is a service, built on rules and at times built with artificial intelligence in place, that a human-being can interact with by using a chat interface. This service can be used for several different applications, either for serious applications or for fun. Facebook Messenger, Slack, Telegram, Text Messages are some examples of some applications where chatbots are already being used today. [1]

A general example of where a chatbot can be used is at a fast-food shop. It is possible to visit this shop's Facebook page, and use Messenger to message them. But instead of chatting to a real human working at that shop, there would be communication with a chatbot. This chatbot would suggest some things to ask, like for instance 'can I see the menu?' or 'can I order food?'. Question in this type of form the chatbot would be able to answer accurately. The work done by the chatbot would save the employees of the fast food shop a lot of time and energy. Each employee can now focus more on cooking food or receiving orders from their customers. The responses from the chatbot would give each customer the impression that they are actually talking to an actual employee.

2.1.2 Chatbot history

ELIZA was just the start of a lot more chatbots to come afterwards, like for instance ALICE, Mitsuku, Albert One, as well as the one which was probably the most popular, SmarterChild. SmarterChild was developed in 2000 and took advantage of the popularity of text messaging services in addition to adding the nice ability to process natural language – or in another way, it did understand the language which we humans speak. The chatbot would also respond to our questions. [2]

Some of the voice chat assistants that are popular today have adopted the technologies of the early-day chatbots, like SmarterChild. This chatbot could be able to answer questions about general topics in real life, like news and sport related questions. Siri is an example of a voice assistant that have adopted features from these early chatbots.

For humans today, it is difficult to imagine a world where we don't have Siri or any other type of voice-based assistants, but only a few years back, the thought of talking to the phone to ask about questions or tell it specific instructions seemed like a strange and very futuristic thing.

The thought of asking an application on your phone for questions about general things, like for instance a weather forecast, seemed very strange and almost impossible for only a few years back, but now this has become the reality of our lives. Today we are using

Siri or Google's Android Voice assistant to ask about things that interest us, like the results from the football match last night, about any upcoming events that have been added to the calendar etc.

Some companies today have discovered that an increasing number of owners of a smartphone don't want any more apps to be downloaded to their phones. For this reason, these companies are developing chatbots to make it more convenient for the phone users to find answers to the questions they are looking for. By the use of chatbots with a user interface, people don't have to download applications to their phone anymore, but now have the possibility of heading to a specific web site to ask questions and receive answers that are satisfactory.

2.2 Neural Networks

A simple neural network takes input as a vector $X(t)$ and generates output $Y(t)$ based on this input. These things are done within a time frame (t). With enough input and output examples the model will be able to learn the parameters of the network in TensorFlow.

After fine-tuning the model for a long time, you would probably want to test the model in a real-life scenario. Often at times this would be done by calling the model multiple times, maybe repeatedly one after the other.

The neural network consists of input weights (w_{in}), output weights (w_{out}) and one hidden layer $Z(t)$. The first half of the model would have the formula: $Z(t) = X(t) * w_{in}$, while the second half of the model would be: $Y(t) = Z(t) * w_{out}$. [3]



Figure 1: Architecture of a regular neural network made in Adobe XD

2.3 Recurrent Neural Networks

The previous section (2.3) described briefly what pieces a neural network consists of. A recurrent neural network is very similar but is smarter in a sense.

Recurrent Neural Networks (RNN) introduce transition weights to transfer information between time.

The introduction of transition weights means that the next state is dependent on the previous model, as well as the previous state. This means that the model has memory of what has happened earlier. [4]

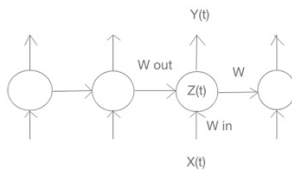


Figure 2: Recurrent Neural Network diagram made in Adobe XD

2.4 Bidirectional recurrent neural networks

Bidirectional recurrent neural networks are similar to recurrent neural networks, but in a BRNN the hidden layer contains nodes moving in opposite directions to each other. So, this means that in a bidirectional recurrent neural network there will also be input and output layers, as well as hidden layers. In BRNNs the hidden layer inside passes data up and down (or forward and backward). Because the hidden layer has this job of passing data in two directions, the network understands what has happened before and what the sequence of words the sequence will contain in the future.

2.5 Sequence to sequence models for chatbots

There exist real people in this world that earn their money from responding to customers about all sorts of questions they may have. Questions can be about a wide variety of topics, but what they have in common is that they are issued either by the use of phone calls, text chat on the internet or e-mails. The job of responding to these questions can be assigned to a programmable chatbot instead, which can be trained to answering questions in most languages of the world. [5]

Chatbots are getting increasingly popular because new research has made it possible to develop these kinds of bots using natural language processing and deep learning technologies.

A chatbot could possibly be able to respond accurately to customers or other people's questions if it is trained with sufficient and suitable data for the given purpose.

2.6 Attention Mechanisms

With only the use of Bidirectional recurrent neural networks, the model would still not remember sentences containing more than 3-10 words/tokens at a time maximum.

BLEU means bilingual evaluation understudy, which is the best possible way to decide the overall effectiveness of a translation algorithm. BLEU will only be relative to the sequences that are translated. [6]

From a chatbot's point of view one sentence is translated to another sentence and these sentences are both written in English. Because of this fact, one should not expect a high value for the BLEU, but the value should slowly increase as the time goes. If sentences were to be translated from one language to another, for example from English to German, the BLEU would have a much higher value over time compared to chatbots.

2.7 Useful things to know about NMT-Chatbot

This section will include information which could be nice to know about the NMT-Chatbot before reading the rest of this thesis.

2.7.1 Introduction to the NMT-chatbot project

Nmt-chatbot is the creation of a chatbot by using NMT - neural machine translation (seq2seq models). This model includes WPM/BPE-tokenizator. The main idea of the project is to create a NMT chatbot, but it is still possible to use NMT with this chatbot and the model can be used to translate words and expressions between languages.

The code in this project makes use of code from NMT, but because no interfaces are available, some things are improvised, and some segments of code needed to be copied into this project.

2.7.2 Standard vs BPE/WPM-like (subword) tokenization, embedded detokenizer

The standard tokenizer makes use of moses-smt one module. Things have been modified a lot in this file to fit the chatbot application. A positive of a tokenizer like this is a smaller number of duplicates in the vocab file. The downside with a tokenizer like this is that it needs a lot of rules based on regex for the job of detokenization. And it is of high difficulty to write a detokenizer which covers all cases.

The functionality of the standard tokenizer is that in-between each sentence there will be either whitespace, a period sign (and other grammar characters), digits, etc.

BPE/WPM-like (subword) tokenizer is built on the project subword-nmt one, but it has been edited to suit the needs and speeds that are available. The biggest positive about this tokenizer is that it has the ability to fit any number of words or tokens in much smaller vocabulary with the use of subwords.

BPE/WPM-like (subword) tokenizer splits sentences similar to the way the standard tokenizer splits sentences. But it also splits every sentence by character, includes a counter of how many pairs of characters have been found (characters next to other characters in the same sentence), and join these pairs to each other to create a vocabulary of the desired number of tokens. This means that the words with the highest count will be joined back to how they were initially, while the words with a low count will be kept split into several parts (subwords) shared with other words. In this manner, almost any kind of number of tokens from the vocabulary can be lowered to the size of just the number 2000 (or even lower, but this is dependent on the training set available).

Standard vs. BPE/WPM-like (subword) tokenization examples:

```
> Isn ' t it broadcasted live on TV ... ?
> Isn ' t it broadcasted live on TV ... ?

> The greatest TV show ever !
> The gre at est TV show ever !

> Come hike along with us, there are a lot of great mountains to hike on ...
>C ome hi ke a long wit h us, there are a lot of gr eat mount ains to hike on ...
```

Figure 3: Standard detokenization vs. BPE/WPM-like (subword) tokenization comparison example.

Embedded detokenizer (possibility for standard tokenizer, forced to 'on' for BPE/WPM-like one – explanation above) makes it possible to do a smooth detokenization instead of lower number of special symbols in vocabulary – duplicate-like symbols in vocabulary (mixes of with and without meta symbol '___', where that meta symbol is the only thing which is needed for detokenization). The number one positive thing about the detokenizer is that no rules exist for this type of detokenization. Detokenization works in the way that two replacements are made in the same sentence. First thing to be done is to remove all whitespace which exist, then replace the so-called meta symbol '___' with a space character instead.

Here are a couple of examples of how this works:

```
> _C ome hi ke a long_wit h_us, _there are a lot of gr eat_moun
ains to hike on ...
>_How_do es_this_dish washer_work?_
```

Figure 4: Embedded tokenizer example.

2.7.3 More detailed information about training a model

setup/settings.py consist of multiple files to change the settings of the model:

- Unchanged file or folder paths should be working most of the time

- "preprocessing" file should be easily-understandable for most people
- "hparams" file will be handed to NMT with the use of command line options with standard way of use
- "score" file with settings for changing the score of the answers

prepare_data.py:

- Works by iterating through the files placed in the folder "new_data" which are: train.from, train.to, tst2012.from, tst2012.to, tst2013.from, tst2013.to
- Tokenizes every sentence found in these files (dependent on the setting and each internal rule which is found)
- For every "train" file – automatically builds vocabulary files, makes them in a way stand out from the others and save up to the number of sentences set in the file setup/settings.py, vocab entities which do not end up being used (if there exist any at all – depends on settings) will all be saved to individual files

train.py – starts the whole training process of the chatbot

2.7.4 Utils

Utils/run_tensorboard.py – file which is simple to use as a wrapper, makes it possible to start Tensorboard with the folder 'model'

Utils/pairing_testing_outputs.py – file which joins the two files model/output_dev and data/tst2012.from together and prints the result to a console allowing to easily have a look if things look ok. The console will show information about the input phrase, inference output frame, as well as a separator.

utils/prepare_for_deployment – file which makes copies of files needed for inference.

2.7.5 Inference

Every time a model is training, running the file inference.py allows to communicate with AI in interactive mode. This file will take care of the start and setup needed to use the already trained model (by the use of the saved hparams file inside the model folder and setup/settings.py for all remaining settings or a missing hparams file).

When a question is asked several responses will be printed depending on the file setup/settings.py. All responses will have one of the following three colors explained:

- Green – the first reply of this color is the top reply for being printed. Replies of this color passed the blacklist test as described in the following file: setup/response_blacklist.txt"
- Orange – replies of this color are good replies, but with a lower rating compared with the ones marked in green

- Red – not a good reply, below the score limit

All the steps which are gone through in the process of question to the answers:

1. Insert the comment/question to be answered by the keyboard
2. Compute fixed number of replies configured in setup/settings.py
3. Perform detokenization on replies based on the embedded detokenizer or the rules which are located in the file setup/answers (depends on the settings which have been configured)
4. Replace whole or only parts of responses by the use of additional rules
5. Set a specific score on every single reply
6. Return the chosen responses (the command 'show' in interactive mode)

There is also an option to have multiple questions asked at the same time by the use of command redirection:

```
Last login: wed may 30 18:50:30 on ttys000
Oles-MacBook-Pro:~ olemartintvad$ python inference.py < input file
```

or:

```
Last login: Wed May 30 18:50:30 on ttys000
Oles-MacBook-Pro:~ olemartintvad$ python inference.py < input_file > output_file
```

There is also a possibility of passing a specified checkpoint as a parameter to use inference at this given checkpoint, for example:

```
Oles-MacBook-Pro:~ olemartintvad$ python inference.py < input_file > output_file
python inference.py translate.ckpt-2000
```

2.7.6 Importing the nmt-chatbot

The project is available for import for the needs of inference. This is done by embedding the folder within the project directory and by the use of import (one thing to notice here is that the folder name needs to be modified to not have the dash (-) character -a module is not possible to be imported if it has a dash in its name). Then the following can be done:

```
from nmt_chatbot.inference import inference

print(inference(«Hello!»))
```

nmt_chatbot is the directory name of the chatbot module

inference() takes one parameter as input which is required from the user:

- Question

For every question asked, the function will return a dictionary containing the following:

- answers – contains a list of all relevant answers
- scores – shows a list of scores for all the answers
- best_index – keeps track of the index for the best answer
- best_score – keeps track of the score of the best answer found

Score:

Every reply initially has a starting score (score[‘starting_score’]) which is a value taken from the file setup/settings.py. This value is by default set to the integer ‘10’. This value is changed by checks which happen later on. The score which each reply has at the end of the process, is what will determine what is the “best” response.

When a lot of questions are issued at the same time, then the function will return several lists of dictionaries.

The function will return ‘None’ for every empty question it manages to find, instead of an actual dictionary.

2.7.7 How to deploy chatbot/model

At the moment when the model has been successfully trained, there might be the necessity for exporting the files only which are needed for the function of inference (for instance to be embedded and imported for use in another project).

This can be achieved automatically with the use of prepare_for_deployment utility:

```
cd utils
python prepare_for_deployment.py
```

This script will create a folder called _deployment inside project’s root directory and copy over all the files which are needed depending on the current settings.

2.7.8 Demo chatbot

The demo model has already been successfully trained. It is in a deployment-ready package. Because it is a demo model, it was not trained on a large data-set. A data-set consisting of one million pairs of rows was used on this model. The neural network used on this model was fairly small as well. The model has only been made to see how it works in practice. The demo model can be found here:

https://www.dropbox.com/s/2w4i77fancf4voc/nmt_chatbot.zip?dl=0

The project includes every file needed for the possibility to run the inference file. All the settings which have been configured in the file setup/settings.py file.

3 MATERIALS AND METHOD

3.1 Method

This method part is part of the Chatbot tutorial found at the website www.pythonprogramming.net. Chatbot tutorial consists of 8 different steps of how to create a chatbot using Python, TensorFlow and Recurrent Neural Networks.

3.1.1 Acquiring training data for the chatbot

There exists a wide variety of datasets which can be used for training a chatbot on the internet, most popular is the Cornell movie dialogue corpus. But I am choosing to use a dataset which consists of a large number of comments and replies taken from Reddit.

Reddit is not like a forum where every single comment posted in that forum is structured linearly, instead the comments are put in a tree-form. Each parent comment is structured linearly, while the replies to the parent comment is indented and every reply to the replies are indented one step further. The diagram below shows how each type of comment is organized.

- Top-level comment 1
 - Reply to parent comment 1
 - - - reply to reply
 - - - reply to reply
- Top-level comment 2
 - Reply to top-level comment 1
 - Reply to top-level comment 1
 - - - Reply to reply

Figure 5: Shows how Reddit comments and replies are structured on the website.

Deep learning is all about input and output pairs of data, so what needs to be done is to transform these comments into comment-reply pairs. From the diagram shown above, the following can be used as pairs of comments and replies:

- Top level comment 1 — Reply to parent comment 1
- Reply to parent comment 1 --- reply to reply

Figure 6: Diagram shows how the different comments/replies can be paired to create a training data set for the chatbot

I took the data set and began to receive input-output pairs based on all this data. The dataset I ended up using consisted of comments all written in the month of January 2015.

3.1.2 Chat Data Structure

When the download of the training data set has been completed, like in general with machine learning the data has to be taken care of and then produce input and output data of this data set. From neural networks point of view, this means that input and output layer have to be decided for the neural network. When implementing the concepts of neural networks and deep learning for the chatbot, data needs to be structured into comments as well as replies. Each comment represents the input, while all the replies represent the output. On the Reddit website some comments will not have any replies to them, but on the other hand many comments could also have a lot of replies. But only one reply is picked for each comment with this chatbot model.

These data dumps of Reddit comments are possibly larger than 32 GB, which is way too large to be put into memory all at the same time, therefore the data needs to be iterated through in relatively small batches. The Reddit comment files will be buffered through and the data which I found interesting was be stored into an SQLite database. The comment data is stored into this database. Every single comment is ordered in a chronologic way, which means each comment is a "parent" at the start and they will have no parents themselves. After a while I got some replies, and every "reply" was stored, and each reply has a parent in the database that we can be found by searching for its respective ID number. In this way, there are rows which consist of parent comments as well as each comment's related reply.

Every time there exists a parent comment with a higher rating compared to the one currently stored, the row is possibly updated with information included in the reply with the higher score. Ultimately, only the rows with the highest score will be left in the database.

I made the following imports to the project:

```
1. # Import database language
2. import sqlite3
3. # Load in the lines from the datadump we are going to use
4. import json
5. # Import datetime to output logging information as we are iterating through data dumps
6. from datetime import datetime
```

SQLite3 is used for the purpose of the database, JSON to load in all the comments from the dataset, while datetime is used to output some useful logging information.

A large dataset was downloaded by the use uTorrent. Every folder was ordered by the year of creation, and every single file contained JSON dataset labeled by year and month of creation ('YYYY-MM'). Every dataset downloaded from uTorrent originally had the .bz2 file extension.

This is how the code is written to handle this process:

```
1. # Search through the file with this format ('yyyy-mm')
2. timeframe = '2015-01'
3. # Build up a big transaction of rows in the database and
   insert all rows at the same time
4. sql_transaction = []
5.
6. '''Connect to SQLite 3 database and make the database have a
   name consisting of month and year of the Reddit Comment file.
7. If the connection to the database fails, the database will be
   created automatically.'''
8. connection = sqlite3.connect('{}db'.format(timeframe))
9. # Define the cursor
10. c = connection.cursor()
```

The timeframe value represents the year and month of data we are using for the database. Now there is something called 'sql_translation' in the code which essentially builds up all the rows in a dataset into one single transaction, executes all of the rows and then finally put all of these rows into bulk groups. After this is done the table is created. SQLite provides the function connect, which includes an if statement that makes sure that a new database will be created one is not created already.

```
1. # Store parent ID, comment ID, parent comment, reply comment, subreddit, the time and
   the scores (votes) for the comment
2. def create_table():
3.     c.execute(
4.         "CREATE TABLE IF NOT EXISTS parent_reply(parent_id TEXT PRIMARY KEY, comment_id
   TEXT UNIQUE, parent TEXT, comment TEXT, subreddit TEXT, unix INT, score INT)")
```

From the above code, `parent_id`, `comment_id`, the parent comment, the reply (comment), subreddit, the time and then the rating for the comment are being prepared for storage.

The code looks like this up until this point:

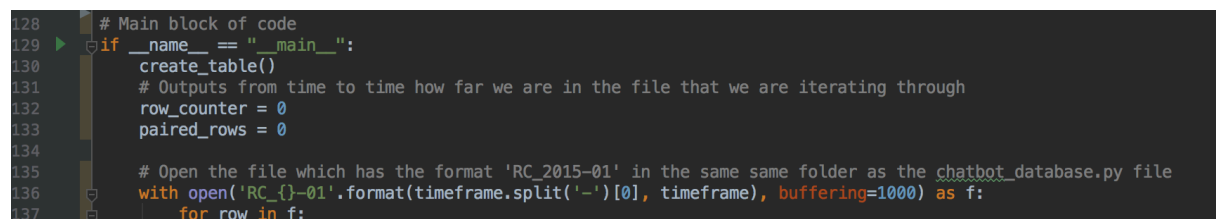
```

1. # Import database language
2. import sqlite3
3. # Load in the lines from the datadump we are going to use
4. import json
5. # Import datetime to output logging information as we are iterating through data dumps
6. from datetime import datetime
7.
8. # Search through the file with this format ('yyyy-mm')
9. timeframe = '2015-01'
10. # Build up a big transaction of rows in the database and insert all rows at the same time
11. sql_transaction = []
12.
13. '''Connect to SQLite 3 database and make the database have a name consisting of month and year of the Reddit Comment file.
14. If the connection to the database fails, the database will be created automatically.'''
15. connection = sqlite3.connect('{}db'.format(timeframe))
16. # Define the cursor
17. c = connection.cursor()
18.
19.
20. # Store parent ID, comment ID, parent comment, reply comment, subreddit, the time and the scores (votes) for the comment
21. def create_table():
22.     c.execute(
23.         "CREATE TABLE IF NOT EXISTS parent_reply(parent_id TEXT PRIMARY KEY, comment_id TEXT UNIQUE, parent TEXT, comment TEXT,
24.         subreddit TEXT, unix INT, score INT)")
25.
26. if __name__ == '__main__':
27.     create_table()

```

3.1.3 Buffering through the data

After all this work was finished, all the data was buffered through as well as a few counters were initialized to keep track of progress over a given time period.



```

128 # Main block of code
129 if __name__ == "__main__":
130     create_table()
131     # Outputs from time to time how far we are in the file that we are iterating through
132     row_counter = 0
133     paired_rows = 0
134
135     # Open the file which has the format 'RC_2015-01' in the same same folder as the chatbot_database.py file
136     with open('RC_{}-01'.format(timeframe.split('-')[0], timeframe), buffering=1000) as f:
137         for row in f:

```

Figure 7: Screenshot of main block of code from PyCharm IDE.

The parameter `'row_counter'` is used for the purpose of finding out how many rows of data have been iterated through so far, while `'paired_rows'` tells how many pairs of comments and replies that have been found. These pairs of comments and replies make the content of the training data.

From figure 7 one can see that there is a `'buffering'` parameter. This parameter is there in case the size of the dataset is too large to be able to deal with in memory. The

buffering parameter makes it possible to read the file we have on our hands for some lines at a time. This is okay to do because only one line at a time will be taken care of.

The row needs to be read which is of the JSON format:

```
135 # Open the file which has the format 'RC_2015-01' in the same same folder as the chatbot_database.py file
136 with open('RC_{}-01'.format(timeframe.split('-')[0], timeframe), buffering=1000) as f:
137     for row in f:
138         row_counter += 1
139         # Read the data into python object
140         row = json.loads(row)
141         parent_id = row['parent_id']
142         body = format_data(row['body'])
143         created_utc = row['created_utc']
144         score = row['score']
145         comment_id = row['name']
146         subreddit = row['subreddit']
```

Figure 8: Diagram which shows how to read each row of the file

There also needs to be a format_data function call, which was created like so:

```
26 # Normalize the comments and convert the new line character to a word
27 def format_data(data):
28     # The line below is used to normalize comments and to convert the new line character to a word
29     data = data.replace('\n', 'newlinechar ').replace('\r', 'newlinechar ').replace('\'', '')
30     return data
31
```

Figure 9: Screenshot of the code of the format_data function.

The format_data function was created to make comments look more normal and convert newline characters to words instead of just being characters.

The function json.loads() was created to read data into a python object, which requires a string formatted like a JSON object. To repeat what was mentioned earlier: Not every single comment will have a parent at the start. There can be two reasons why a parent does not exist:

1. Because the comment is the highest rated one available
2. Because the parent is not included in the document

As the document is iterated through though, there will be found some comments which have parents that already exist in the database. Each time this happens the comment needs to be connected to the already existing parent. After a successful iteration of a file, or a series of files, the parent and comment pairs will be output from the database. These pairs will be used for training data which can be used to train the chatbot model. After having trained the model it will be possible to communicate with the Chatbot. There is a statement in the file which checks if a parent already exists in the database for the comment that has been found:

```
147 # Make sure we can find the parent comment before we put data into the database
148 parent_data = find_parent(parent_id)
```

After this was created, the find_parent function also needs to be implemented:

```

98 # Find new comments attached to a parent ID with the actual parent ID text (body)
99 def find_parent(pid):
100     try:
101         sql = "SELECT comment FROM parent_reply WHERE comment_id = '{}'.format(pid)
102         c.execute(sql)
103         result = c.fetchone()
104         if result != None:
105             return result[0]
106         else:
107             return False
108     except Exception as e:
109         # print(str(e))
110         return False

```

If the database already contains a comment_id that has a matching parent_id of another comment, then the new comment should be connected to the parent that already is found inside the database.

3.1.4 Insert Logic

There is a rule implemented in the code to restrict the number of comments that are dealt with, regardless if any other comments exist as well. The rule is that only comments of a rating of two or higher will be added to the database. The code so far looked like this (shown in 3 parts):

```

1. import sqlite3
2. import json
3. from datetime import datetime
4.
5. timeframe = '2015-05'
6. sql_transaction = []
7.
8. connection = sqlite3.connect('{}db'.format(timeframe))
9. c = connection.cursor()
10.
11. def create_table():
12.     c.execute("CREATE TABLE IF NOT EXISTS parent_reply(parent_id TEXT PRIMARY KEY, comment_id TEXT UNIQUE, parent TEXT, comment
13.     TEXT, subreddit TEXT, unix INT, score INT)")
14.
15. def format_data(data):
16.     data = data.replace('\n', ' ').replace('\r', ' ').replace("'", "")
17.     return data

```

Figure 10: Part 1 of how the code looks like at this point.

```

18. def find_parent(pid):
19.     try:
20.         sql = "SELECT comment FROM parent_reply WHERE comment_id = '{}' LIMIT 1".format(pid)
21.         c.execute(sql)
22.         result = c.fetchone()
23.         if result != None:
24.             return result[0]
25.         else: return False
26.     except Exception as e:
27.         #print(str(e))
28.         return False
29.
30.
31. if __name__ == '__main__':
32.     create_table()
33.     row_counter = 0
34.     paired_rows = 0

```

Figure 11: Part 2 of what the code looks like at this point.

```

36. with open('J:/chatdata/reddit_data/{}/RC_{}'.format(timeframe.split('-')[0], timeframe), buffering=1000) as f:
37.     for row in f:
38.         row_counter += 1
39.         row = json.loads(row)
40.         parent_id = row['parent_id']
41.         body = format_data(row['body'])
42.         created_utc = row['created_utc']
43.         score = row['score']
44.         comment_id = row['name']
45.         subreddit = row['subreddit']
46.         parent_data = find_parent(parent_id)

```

Figure 12: Part 3 of how the code should like up until this point.

Now the requirement of the rating restriction to be two or higher is added to the code. The code also checks to see if there is an existing reply to the parent comment in the database, as well as attempts to see what the score of the comment is:

```

1. if __name__ == '__main__':
2.     create_table()
3.     row_counter = 0
4.     paired_rows = 0
5.
6.     with open('J:/chatdata/reddit_data/{}/RC_{}'.format(timeframe.split('-')[0], timeframe), buffering=1000) as f:
7.         for row in f:
8.             row_counter += 1
9.             row = json.loads(row)
10.            parent_id = row['parent_id']
11.            body = format_data(row['body'])
12.            created_utc = row['created_utc']
13.            score = row['score']
14.            comment_id = row['name']
15.            subreddit = row['subreddit']
16.            parent_data = find_parent(parent_id)
17.            # maybe check for a child, if child, is our new score superior? If so, replace. If not...
18.
19.            if score >= 2:
20.                existing_comment_score = find_existing_score(parent_id)

```

The next thing done was the creation of the find_existing_score function:

```

1. # Find existing score of a comment by using the parent ID
2. def find_existing_score(pid):
3.     try:
4.         sql = "SELECT score FROM parent_reply WHERE parent_id = '{}' LIMIT 1".format(pid)
5.         c.execute(sql)
6.         result = c.fetchone()
7.         if result != None:
8.             return result[0]
9.         else:
10.            return False
11.    except Exception as e:
12.        # print(str(e))
13.        return False

```

If the database already contains a comment with a rating lower than the comment that was found recently, then the comment already in the database will be replaced:

```

149         parent_data = find_parent(parent_id)
150         if score >= 2:
151             existing_comment_score = find_existing_score(parent_id)
152             if existing_comment_score:
153                 if score > existing_comment_score:

```

On Reddit several comments are not there anymore, either because they have been deleted or removed. But some comments on this forum also contain a lot of characters, or the opposite: very few characters. In the program there is a statement which makes sure that comments contain a sufficient amount of characters for training, as well as that the comment was not removed or deleted.


```

84 # Make sure the comments are of an acceptable length for training, and that the comment wasn't removed deleted
85 def acceptable(data):
86     if len(data.split(' ')) > 50 or len(data) < 1:
87         return False
88     elif len(data) > 1000:
89         return False
90     elif data == '[deleted]':
91         return False
92     elif data == '[removed]':
93         return False
94     else:
95         return True
96

```

Now the insertion of the data into the database starts, which will be discussed in the next section.

3.1.5 Building database

If an existing comment score is possible to find in the database, this means also that a comment exists, which in this case means there will have to be an update statement.

The following was how the code looked like up until this point:

```

149     if score >= 2:
150         existing_comment_score = find_existing_score(parent_id)
151         if existing_comment_score:
152             if score > existing_comment_score:
153                 if acceptable(body):
154                     sql_insert_replace_comment(comment_id, parent_id, parent_data, body, subreddit, created_utc,
155                                                 score)
156

```

The next thing which was done was to build the 'sql_insert_replace_comment' function:

```

1. def sql_insert_replace_comment(commentid,parentid,parent,comment,subreddit,time,score):
2.     try:
3.         sql = """UPDATE parent_reply SET parent_id = ?, comment_id = ?, parent = ?, comment = ?,
4.         subreddit = ?, unix = ?, score = ? WHERE parent_id =?;""".format(parentid, commentid, parent,
5.         comment, subreddit, int(time), score, parentid)
6.         transaction_blldr(sql)
7.     except Exception as e:
8.         print('s0 insertion',str(e))

```

The code above takes care of some situations where a comment has already been connected to a parent comment.

In the next image there will be code showing how I decided to handle situations where comments do not have parents and these parents do not already have a reply to them.

```

149     if score >= 2:
150         existing_comment_score = find_existing_score(parent_id)
151         if existing_comment_score:
152             if score > existing_comment_score:
153                 if acceptable(body):
154                     sql_insert_replace_comment(comment_id, parent_id, parent_data, body, subreddit, created_utc,
155                                                 score)
156
157         else:
158             if acceptable(body):
159                 if parent_data:
160                     sql_insert_has_parent(comment_id, parent_id, parent_data, body, subreddit, created_utc, score)
161                     paired_rows += 1
162             else:
163                 sql_insert_no_parent(comment_id, parent_id, body, subreddit, created_utc, score)

```

Figure 13: Code for dealing with certain situations of missing parent comments and missing replies to the parents.

Up next is the `sql_insert_has_parent` and `sql_insert_no_parent` functions that was built:

```

1. def sql_insert_has_parent(commentid,parentid,parent,comment,subreddit,time,score):
2.     try:
3.         sql = """INSERT INTO parent_reply (parent_id, comment_id, parent, comment, subreddit,
4.         unix, score) VALUES ("{}","{}","{}","{}","{}",{},{})""".format(parentid, commentid, parent,
5.         comment, subreddit, int(time), score)
6.         transaction_bldr(sql)
7.     except Exception as e:
8.         print('s0 insertion',str(e))
9.
10. def sql_insert_no_parent(commentid,parentid,comment,subreddit,time,score):
11.     try:
12.         sql = """INSERT INTO parent_reply (parent_id, comment_id, comment, subreddit, unix,
13.         score) VALUES ("{}","{}","{}","{}",{},{})""".format(parentid, commentid, comment, subreddit,
14.         int(time), score)
15.         transaction_bldr(sql)
16.     except Exception as e:
17.         print('s0 insertion',str(e))

```

To see how many steps the iteration process has been going on for, there is some output which shows some useful information every 100 000 step:

```

164     # Output every 100,000 rows of data some information
165     if row_counter % 100000 == 0:
166         print('Total Rows Read: {}, Paired Rows: {}, Time: {}'.format(row_counter, paired_rows,
167         str(datetime.now())))
168

```

The last lines of code that are needed for now will be used for building the `transaction_bldr` function. The purpose of this function is to create a set of insertion statements and commit them in groups, instead of dealing with one statement at a time. This will speed up the process a lot:

```

33 # build up insertion statements and commit them in groups
34 def transaction_bldr(sql):
35     global sql_transaction
36     sql_transaction.append(sql)
37     if len(sql_transaction) > 1000:
38         # Start transaction
39         c.execute('BEGIN TRANSACTION')
40         # For each SQL statement
41         for s in sql_transaction:
42             try:
43                 c.execute(s)
44             except:
45                 pass
46         connection.commit()
47         # empty transaction
48         sql_transaction = []

```

How the code looks like up until this point:

```

1. import sqlite3
2. import json
3. from datetime import datetime
4.
5. timeframe = '2015-05'
6. sql_transaction = []
7.
8. connection = sqlite3.connect('{}db'.format(timeframe))
9. c = connection.cursor()
10.
11. def create_table():
12.     c.execute("CREATE TABLE IF NOT EXISTS parent_reply(parent_id TEXT PRIMARY KEY, comment_id TEXT UNIQUE, parent TEXT, comment
13.     TEXT, subreddit TEXT, unix INT, score INT)")
14.
15. def format_data(data):
16.     data = data.replace('\n', '\n').replace('\r', '\r').replace("'", "'")
17.     return data
18.
19. def transaction_bldr(sql):
20.     global sql_transaction
21.     sql_transaction.append(sql)
22.     if len(sql_transaction) > 1000:
23.         c.execute('BEGIN TRANSACTION')
24.         for s in sql_transaction:
25.             try:
26.                 c.execute(s)
27.             except:
28.                 pass
29.         connection.commit()
30.         sql_transaction = []
31.
32. def sql_insert_replace_comment(commentid, parentid, parent, comment, subreddit, time, score):
33.     try:
34.         sql = """UPDATE parent_reply SET parent_id = ?, comment_id = ?, parent = ?, comment = ?, subreddit = ?, unix = ?, score =
35.         ? WHERE parent_id =?;""".format(parentid, commentid, parent, comment, subreddit, int(time), score, parentid)
36.         transaction_bldr(sql)
37.     except Exception as e:
38.         print('s0 insertion', str(e))

```

```

38. def sql_insert_has_parent(commentid,parentid,parent,comment,subreddit,time,score):
39.     try:
40.         sql = """INSERT INTO parent_reply (parent_id, comment_id, parent, comment, subreddit, unix, score) VALUES ("{}","{}","
{}","{}","{}",{});""".format(parentid, commentid, parent, comment, subreddit, int(time), score)
41.         transaction_blldr(sql)
42.     except Exception as e:
43.         print('s0 insertion',str(e))
44.
45. def sql_insert_no_parent(commentid,parentid,comment,subreddit,time,score):
46.     try:
47.         sql = """INSERT INTO parent_reply (parent_id, comment_id, comment, subreddit, unix, score) VALUES ("{}","{}","{}","{}",
{});""".format(parentid, commentid, comment, subreddit, int(time), score)
48.         transaction_blldr(sql)
49.     except Exception as e:
50.         print('s0 insertion',str(e))
51.
52. def acceptable(data):
53.     if len(data.split(' ')) > 50 or len(data) < 1:
54.         return False
55.     elif len(data) > 1000:
56.         return False
57.     elif data == '[deleted]':
58.         return False
59.     elif data == '[removed]':
60.         return False
61.     else:
62.         return True
63.
64. def find_parent(pid):
65.     try:
66.         sql = "SELECT comment FROM parent_reply WHERE comment_id = '{}' LIMIT 1".format(pid)
67.         c.execute(sql)
68.         result = c.fetchone()
69.         if result != None:
70.             return result[0]
71.         else: return False
72.     except Exception as e:
73.         #print(str(e))
74.         return False

```

```

76. def find_existing_score(pid):
77.     try:
78.         sql = "SELECT score FROM parent_reply WHERE parent_id = '{}' LIMIT 1".format(pid)
79.         c.execute(sql)
80.         result = c.fetchone()
81.         if result != None:
82.             return result[0]
83.         else: return False
84.     except Exception as e:
85.         #print(str(e))
86.         return False
87.
88. if __name__ == '__main__':
89.     create_table()
90.     row_counter = 0
91.     paired_rows = 0
92.
93.     with open('J:/chatdata/reddit_data/{}/RC_{}'.format(timeframe.split('-')[0], timeframe), buffering=1000) as f:
94.         for row in f:
95.             row_counter += 1
96.             row = json.loads(row)
97.             parent_id = row['parent_id']
98.             body = format_data(row['body'])
99.             created_utc = row['created_utc']
100.            score = row['score']
101.            comment_id = row['name']
102.            subreddit = row['subreddit']
103.            parent_data = find_parent(parent_id)
104.            if score >= 2:
105.                existing_comment_score = find_existing_score(parent_id)
106.                if existing_comment_score:
107.                    if score > existing_comment_score:
108.                        if acceptable(body):
109.                            sql_insert_replace_comment(comment_id, parent_id, parent_data, body, subreddit, created_utc, score)
110.
111.                else:
112.                    if acceptable(body):
113.                        if parent_data:
114.                            sql_insert_has_parent(comment_id, parent_id, parent_data, body, subreddit, created_utc, score)
115.                            paired_rows += 1
116.                        else:
117.                            sql_insert_no_parent(comment_id, parent_id, body, subreddit, created_utc, score)
118.
119.            else:
120.                if acceptable(body):
121.                    if parent_data:
122.                        sql_insert_has_parent(comment_id, parent_id, parent_data, body, subreddit, created_utc, score)
123.                        paired_rows += 1
124.                    else:
125.                        sql_insert_no_parent(comment_id, parent_id, body, subreddit, created_utc, score)
126.
127.            if row_counter % 100000 == 0:
128.                print('Total Rows Read: {}, Paired Rows: {}, Time: {}'.format(row_counter, paired_rows, str(datetime.now())))

```

After I got all this code, I started running the code. As the time went on the output started looking like the following (see below):

```
Total Rows Read: 100000, Paired Rows: 3221, Time: 2017-11-14  
15:14:33.748595  
Total Rows Read: 200000, Paired Rows: 8071, Time: 2017-11-14  
15:14:55.342929  
Total Rows Read: 300000, Paired Rows: 13697, Time: 2017-11-14  
15:15:18.035447  
Total Rows Read: 400000, Paired Rows: 19723, Time: 2017-11-14  
15:15:40.311376  
Total Rows Read: 500000, Paired Rows: 25643, Time: 2017-11-14  
15:16:02.045075
```

When the iteration over the files was finished, I then made sure the files were of a suitable form before training of the chatbot could take place. The training of the dataset will be explained next.

3.1.6 Training Dataset

This section will include how to work on creating the training data for the chatbot model. "Parent" and "reply" text files need to be created, where each line was used for training data. This means that line 15 of the file which contains parent comments will correspond to line 15 in the file which contains all the rows of replies.

To make sure all these files are created, pairs comment-reply pairs were picked from the database and then these pairs were added to the training files.

The following code shows how some of this is performed:

```
1 import sqlite3  
2 import pandas as pd  
3  
4 timeframes = ['2015-01']  
5  
6 for timeframe in timeframes:
```

Here only a single month of data will be run through, because only one database will be dealt with. But there are other options available.

Next, the loop is built:

```

6     for timeframe in timeframes:
7         connection = sqlite3.connect('{}db'.format(timeframe))
8         c = connection.cursor()
9         # Limit - Size of chunk we are pulling from the database
10        limit = 5000
11        # Last_unix helps us make pulls from the database
12        last_unix = 0
13        # Cur_length to tell when we are done pulling
14        cur_length = limit
15        # Counter allows us to show debugging information
16        counter = 0
17        # Test_done for when we're done building test data
18        test_done = False

```

The first line makes the connection to the sqlite database established, then the second line defines the cursor, while in the third line the limit is assigned a value of 5000. This is the number of rows that are pulled from the database simultaneously. As told earlier, the data which is worked with is a lot larger than the actual RAM which is available to us. The limit variable has a value of 5000 to make some testing data available. This value could be increased later on if desired. Last_unix variable makes it much easier to pull rows from the database. cur_length gives information when the pulling of data from the database is completed. counter variable allows to give some information in case there needs to be debugging. The purpose of test_done is to tell when the test data has been successfully built.

The limit can be changed to have a higher value. The variable limit can be changed to for example 100k or something similar after the condition test_done = True has been fulfilled. Now the training code was built:

```

39         else:
40             with open('train.from', 'a', encoding='utf8') as f:
41                 for content in df['parent'].values:
42                     f.write(content + '\n')
43
44             with open('train.to', 'a', encoding='utf8') as f:
45                 for content in df['comment'].values:
46                     f.write(str(content) + '\n')
47

```

The code segment could have been put into a function to make sure the action of copying and pasting did not have to be performed, but here it will be ignored.

```

48         counter += 1
49         if counter % 20 == 0:
50             print(counter * limit, 'rows completed so far')
51

```

For every 20 steps that was taken there was some output, which means every 100k pairs of rows if the value of the limit variable is kept at 5 000. That is what was done in this case.

All code that has been told about so far in this section:

```

1 import sqlite3
2 import pandas as pd
3
4 timeframes = ['2015-01']
5
6 for timeframe in timeframes:
7     connection = sqlite3.connect('{}db'.format(timeframe))
8     c = connection.cursor()
9     # Limit - Size of chunk we are pulling from the database
10    limit = 5000
11    # Last_unix helps us make pulls from the database
12    last_unix = 0
13    # Cur_length to tell when we are done pulling
14    cur_length = limit
15    # Counter allows us to show debugging information
16    counter = 0
17    # Test_done for when we're done building test data
18    test_done = False
19
20    while cur_length == limit:
21
22        df = pd.read_sql(
23            "SELECT * FROM parent_reply WHERE unix > {} and parent NOT NULL and score > 0 ORDER BY unix ASC LIMIT {}".format(
24                last_unix, limit), connection)
25        last_unix = df.tail(1)['unix'].values[0]
26        cur_length = len(df)
27
28        if not test_done:
29            with open('test.from', 'a', encoding='utf8') as f:
30                for content in df['parent'].values:
31                    f.write(content + '\n')
32
33            with open('test.to', 'a', encoding='utf8') as f:
34                for content in df['comment'].values:
35                    f.write(str(content) + '\n')
36
37            test_done = True
38
39        else:
40            with open('train.from', 'a', encoding='utf8') as f:
41                for content in df['parent'].values:
42                    f.write(content + '\n')
43
44            with open('train.to', 'a', encoding='utf8') as f:
45                for content in df['comment'].values:
46                    f.write(str(content) + '\n')
47
48            counter += 1
49            if counter % 20 == 0:
50                print(counter * limit, 'rows completed so far')
51

```

3.1.7 Training a model

There is a wide range of models which can either be used or found on the Internet and tweaked to fit one's needs. What is chosen to be used here is sequence to sequence model, because this model can mainly be used with chatbots, but also for other applications. Information about everything can be made into sequences, which means a lot of things can be used for training. But in this occasion, a chatbot was created with the use of comments and replies found on Reddit.

In this bachelor project I have been working on a project which Daniel Kukiela created and uploaded to Github. The project is set to public access, so everyone who has the link can make use of the project. [7]

1. `$ git clone --recursive https://github.com/daniel-kukiela/nmt-chatbot`
(or)
`$ git clone --branch v0.1 --recursive https://github.com/daniel-kukiela/nmt-chatbot.git` (for a version featured in Sentdex tutorial)
2. `$ cd nmt-chatbot`
3. `$ pip install -r requirements.txt` TensorFlow-GPU is one of the requirements. You also need CUDA Toolkit 8.0 and cuDNN 6.1. (Windows tutorial: <https://www.youtube.com/watch?v=r7-WPbx8VuY> Linux tutorial: <https://pythonprogramming.net/how-to-cuda-gpu-tensorflow-deep-learning-tutorial/>)
4. `$ cd setup`
5. (optional) edit settings.py to your liking. These are a decent starting point for ~4GB of VRAM, you should first start by trying to raise vocab if you can.
6. (optional) Edit text files containing rules in the setup directory.
7. Place training data inside "new_data" folder (train.(from|to), tst2013.(from|to), tst2013(from|to)). We have provided some sample data for those who just want to do a quick test drive.
8. `$ python prepare_data.py` ...Run setup/prepare_data.py - a new folder called "data" will be created with prepared training data
9. `$ cd ../`
10. `$ python train.py` Begin training

Figure 14: README file at <https://github.com/daniel-kukiela/nmt-chatbot/blob/master/README.md>

The things shown in the figure above will be explained in the following section. Everything was set up and prepared for running.

One thing to note about the requirements.txt file is that it is not possible to install the package 'tensorflow-gpu' if there is not a GPU on the computer which supports this. There needs to be a NVIDIA GPU to be able to support the 'tensorflow-gpu' package. If the computer does not have one of these GPUs, the line can be edited to just say 'tensorflow', then the standard TensorFlow package will be installed instead. [3]

```

1  tensorflow-gpu>=1.4.0
2  tqdm
3  colorama
4  regex
5  python-Levenshtein
6  requests

```

Figure 15: Screenshot of how the requirements.txt file how it looks originally

The NMT package was downloaded recursively from <https://github.com/tensorflow/nmt> (the official TensorFlow source).

When the package has been downloaded recursively, there is an option to edit things in setup/settings.py which anyone can do, but this can also just be left un-edited.

```

1. hparams = {
2.     'attention': 'scaled_luong',
3.     'src': 'from',
4.     'tgt': 'to',
5.     'vocab_prefix': os.path.join(train_dir, "vocab"),
6.     'train_prefix': os.path.join(train_dir, "train"),
7.     'dev_prefix': os.path.join(train_dir, "tst2012"),
8.     'test_prefix': os.path.join(train_dir, "tst2013"),
9.     'out_dir': out_dir,
10.    'num_train_steps': 500000,
11.    'num_layers': 2,
12.    'num_units': 512,
13.    'override_loaded_hparams': True,
14.    'learning_rate': 0.001,
15.    # 'decay_factor': 0.99998,
16.    'decay_steps': 1,
17.    # 'residual': True,
18.    'start_decay_step': 1,
19.    'beam_width': 10,
20.    'length_penalty_weight': 1.0,
21.    'optimizer': 'adam',
22.    'encoder_type': 'bi',
23.    'num_translations_per_input': 30
24. }
```

Figure 16: The content of the settings.py file

When the settings have been edited to as you would like them to be, inside the main directory (where the directories utils, tests and setup are located), the train.to and train.from files as well as the corresponding tst2012 and tst2013 files can all be placed

inside the new_data directory. Now type 'cd setup' in the terminal window to change directory. Then run the prepare_data.py file:

```
$ python3 prepare_data.py
```

When the preparation of the data has been completed, then type 'cd ../' and then type the following command:

```
$ python3 train.py
```

By running this python-file, the chatbot model will start training.

3.1.8 Interacting with the chatbot

With the chatbot model training the default is that a new checkpoint file is saved for every 1000nd step. If there is a necessity or desire to stop the training, then this is possible to do safely and continue training at a later time from the last saved checkpoint.

Each checkpoint file contains a few different logging parameters, as well as other useful information for the model. Because of these logging parameters, it is possible to take these checkpoints/model files and use them to continue the training of the model or use them in production of another model.

The location for saving every checkpoint file by default is in the model directory. Saved in these directories the files will be called translate.ckpt-XXXXX, where the X's represent the step number. In the model directory there will also be files with the extensions .data, .index, and a .meta file, as well as a checkpoint file. The content of the checkpoint file will look something similar to the following:

```
model_checkpoint_path: "/Users/olemartintvad/nmt-chatbot/model/translate.ckpt-229000"  
all_model_checkpoint_paths: "/Users/olemartintvad/nmt-chatbot/model/translate.ckpt-225000"  
all_model_checkpoint_paths: "/Users/olemartintvad/nmt-chatbot/model/translate.ckpt-226000"  
all_model_checkpoint_paths: "/Users/olemartintvad/nmt-chatbot/model/translate.ckpt-227000"  
all_model_checkpoint_paths: "/Users/olemartintvad/nmt-chatbot/model/translate.ckpt-228000"  
all_model_checkpoint_paths: "/Users/olemartintvad/nmt-chatbot/model/translate.ckpt-229000"
```

This checkpoint file lets the model know which files to be used.

To use this model on another machine, there are 4 files in total that need to be on this machine. As an example, step 229 000 will be used. That means that there needs to be the following four types of files to run that model, or to load the model to continue training:

- checkpoint
- translate.ckpt-229000.meta
- translate.ckpt-229000.index
- translate.ckpt-229000.data-00000-of-00001

These are the files one will need if that person is to move from training the model on their local machine to continue train the model in a desktop in the cloud, for instance with a machine by using Amazon Web Services.

As well as saving checkpoints for every 1000 steps, there also needs to be some examples of bulk (from the `tst.to` and `tst.from` files that are available). These output files are just the top output sentence result responses to the test files which exist. Because any examples could have been put into the testing files, this is the first moment that interaction the chatbot will happen. There is a pairing script which will output the pairs of comment-replies of the testing files and output files.

For instance, there can be the following content in the `tst2013.from` file:

```
Aren't they streaming it for free online...?  
try to get loud please  
I'm trying to eat a fucking fajita here dammit  
E  
It's been 3 innings and Spanton almost hit a dong.  
Looks-wise yes, play-wise no  
But we'd both pay $9.99 to see that. newlinechar newlinechar Isn't he doing stuff for CZW? Aren't they like extreme stuff, they should do a  
Punjabi Prison Match with CJ Parker.  
'I simply feel as though the game is not for me.' *Zaffre states, turning back to Ambrose, a frown on his face.*  
The fire escape is there. You hear wood splintering, and look to see that a raptor has managed to break a hole in the top of the door, just  
above the dresser. Its head pokes through, then disappears. There's another thud, and the dresser moves forward a few inches.  
[!(/fritteehhe) I wonder how I'll make the eyes all red...  
360,678  
I like the idea...have an upvote!  
who talks shit about Giannis?  
C  
I'm pretty sure that's the peace music
```

And this is content from the `output_dev` file:

```
http://www.youtube.com/watch?v=5a5a58c8s newlinechar newlinechar http://www.youtube.com/  
watch?v=a_m55c58s)  
http://www.youtube.com/watch?v=JJJ8x8Y8Y  
http://www.youtube.com/watch?v=a_t5a8Y8Y newlinechar newlinechar http://www.youtube.com/  
watch?v=a_t_tYYYY  
S  
I don't think you're right.  
I don't think they're right.  
I don't think they're right.  
'I don't think you're right.'  
Thanks!  
[!(/abcute)  
362,888  
Thank you!
```

The following screenshot shows code which represents a pairing script:

```

1  from nmt_chatbot.inference import inference
2
3  while True:
4      # Take text entered by the user and ask bot what it thinks
5      text = input(":> ")
6      output = inference(text)
7      # Print best response available
8      print(output["answers"][output["best_index"]])

```

Figure 17: Screenshot of the code from *Chatbot_questions.py*

The code above takes text entered by the user and asks the bot what it thinks about the question. A number of possible replies will be thought of, but only the response with the highest score will be shown as the output with the run of this file.

Output should look something similar to the below:

```

:> hello
^^^^^^^^^^
:> hey
Thanks!
:> hi
^^
:> how are you?
I don't know.

```

Inference.py file is used for asking a chatbot questions and have a live preview has it is happening. This file is run in the terminal just like all the other related files for the NMT-chatbot. One sees that the chatbot does not always give very intellectual replies to the issued questions, but the model can always be modified to actually answer in a more intelligent way.

When running the inference script, one will have to choose some of the responses shown in a list. It is possible to go for the first response shown in the list or choose all of them if desired.

```
> what's up?
- https://www.youtube.com/ [15.1]
- https://www.reddit.com/ [14.45]
- What's up? [-87.5]
- https://en.wikipedia.org/ [-185.75]
- https://i.imgur.com/ [-186.5]
- https://en.m.wikipedia. [-86.05]
- what's up? [-88.0]
- https://m.youtube.com/ [-186.7]
- https://imgur.com/a/ [-187.0]
- https://imgur.com/gallery/ [13.9]
- https://www.google.com/ [13.45]
- https://m.imgur.com/ [-187.0]
- I don't know what's up. [1.4499999999999993]
- I don't know what's up, [-98.55]
- I don't know what's up up [-98.25]
- I don't know what's up? [1.4499999999999993]
- I don't know what's up [-98.7]
- I don't know what's up yet [-98.1]
- I don't know what's up but [-98.1]
- https://youtu.be/x_ [-187.15]
```

Figure 18: Possible replies to the comment 'what's up?' when running the script *inference.py*

```
> hi
- hi [6.8]
- oi [6.3]
- ki [6.3]
- ^^^^ [12.1]
- mi [5.8]
- Hi [5.8]
- Ni [5.8]
- I [5.15]
- k [5.15]
- s [5.15]
- Wi [5.3]
- y [5.15]
- ni [5.3]
- Ki [5.3]
- hi? [10.45]
- It's a [-89.1]
- I'm not [-88.95]
- It's not [-88.8]
- It's the [-88.8]
- I'm a [-89.25]

> ■
```

Figure 19: Possible replies to the comment 'hi' shown when running the script *inference.py*

3.2 *Materials*

This section includes some information about the programming language used for this bachelor project along with the Python libraries.

3.2.1 Python

Python is a high-level programming language which can be used for several purposes. The programming language stands out from other programming languages with the focus on readability by the use of a lot of whitespace.

A dynamic system and automatic memory management are among the features that Python offers, but there are several other features which Python supports. Features like imperative, functional, procedural and objective-oriented programming are also supported. Python also offers a library with many packages for use by anyone. [8]

3.2.2 PyCharm

PyCharm is an integrated development environment optimized for the use of code from the Python programming language. Some the features of PyCharm are as follows: Data analysis, a graphical debugger, an integrated unit tester, integration with version control systems and also supports Web development with Django. [9]

3.3 *Libraries and frameworks*

This section gives a little bit of detailed information about the Python libraries and Frameworks which have been used in this project.

3.3.1 TensorFlow

TensorFlow is an open-source software library for dataflow programming meant for different tasks. It is a symbolic math library, which is also commonly used in machine learning applications, such as neural networks. TensorFlow is used in production and research at Google. [10]

The early versions of TensorFlow did not support more than one device, while later versions actually provided support for several GPUs and CPUs with optional extensions for basic computing on devices which use graphical processing units.

3.3.2 tqdm

tqdm is 'progress' in Arabic and in Spain this is used as an abbreviation for "I love you so much" (te quiero demasiado).

This package is an extra styling to the loops to make them show a smart progress meter. One can wrap any iterable with `tqdm(iterable)`, and that's it!

```
from tqdm import tqdm
for i in tqdm(range(10000)):
    ...
```

Figure 20: Screenshot from page <https://tqdm.github.io/>, showing an example how to use the tqdm package in the code

The tqdm package works across every single platform, which means Linux, Windows, Mac, FreeBSD, NetBSD, as well as Solaris/SunOS, in any console or in a GUI. tqdm also provides support for IPython/Jupyter notebooks.

3.3.3 colorama

Colorama – cross-platform colored text in the terminal window

Colorama makes ANSI abandon sequences of characters for the purpose of producing colored text in the terminal and positioning of the cursor. This package works for MS Windows. [6]

The function ANSI escape character sequences have been used for a long duration of time to produce colored text in the terminal window and positioning of the cursors on Unix and Mac systems. Colorama enables this function on Windows as well, by the wrapping of stdout, stripping every ANSI sequence on its path and converting them into win32 calls to change the terminal's state. On other platforms than Windows, Colorama does not make any changes.

3.3.4 Regex

Regex is a short name which means regular expression, a string of text which allows a programmer to create patterns that help match, locate and manage text. Perl is a prime example of a programming language that makes good use of regular expressions. But this is just one of the many places where one can find the use of regular expressions. Regular expressions can also be used to find text inside a file via the command line interface or in text editors. [14]

3.3.5 Python-levenhstein

The Levenhstein Python C extension module includes for quick computation of

- Levenshtein (edit) distance, and edit operations
- String similarity
- Approximation of median strings, and generally find out the average of strings
- String sequence and set similarity

The extension supports normal and Unicode strings.

The Python version has to be of version 2.2 or later.

StringMatcher.py is an example SequenceMatcher-like class which has been built on top of Levenshtein. It lacks some of SequenceMatcher's functionality, and has some more OTOH. [9]

3.3.6 NumPy

NumPy is a fundamental package for scientific computing specifically made for Python. What NumPy contains:

- A powerful N-dimensional array object
- Some sophisticated (broadcasting functions)
- Tools which are used for integrating code of C/C++ and Fortran
- The capabilities of useful linear algebra, Fourier transform, and random numbers

NumPy is not only for scientific uses but can be used as an efficient multi-dimensional container of generic data. Data-types which are arbitrary can be defined. Because of this, NumPy can integrate in a quick and good way with several different databases.

The Python library NumPy enables use of mathematical manipulation in the Python programming language.

Without using the Python libraries TensorFlow and NumPy there would be a lot more time-consuming work in coding to be done, like capturing a photo without using auto focus. [15]

3.3.7 Requests

Requests is an HTTP library licensed under Apache2, which has been written in Python. The library has been designed for humans to be able to interact with the language. This makes it a little bit easier for humans, in the way that they do not have to add query strings to URLs, and they don't have to form-encode their POST data.

What can Requests do?

The Requests library will allow someone to send HTTP/1.1 requests by the use of Python code. It can be used for adding content like for instance headers, form data, multipart files, as well as parameters via easy-to-use Python libraries. The library allows a person to access the response data of Python in the same manner. [13]

How to install Requests

There are quite a few different ways of installing the Requests library.

One can choose to use pip, easy_install, as well as tarball.

How to install using pip: Type 'pip install requests' in the terminal to install. [13]

3.3.8 Adobe XD

Adobe XD is an application developed by Adobe, and the application itself makes things easy for web designers to create good-looking thumbnails for potential web pages. The most interesting part about this application is that it's possible to preview how the specific page will look on different type of screens, like for instance iPhone devices, Android screens, PC screens etc.

3.3.9 Adobe Illustrator CC 2019

Adobe Illustrator is an application created by Adobe, which lets designers create illustrations and figures by the use of a wide variety of default tools available in the application. The application is part of an annual or monthly subscription of the package Creative Cloud, which contains a set of related applications that are useful for those who work within the design field.

4 RESULTS

A model was trained for 229 000 steps, but because I changed some code by mistake, I was not able to test this model with questions to see what actual answers the chatbot would give.

The demo model did not include any data for analyzing perplexity, BLEU etc. So I could not do much with this model other than testing it with some questions.

By running the file `utils/run_tensorboard.py` I got the following graphs in the web browser (`localhost:22222` was the URL used):

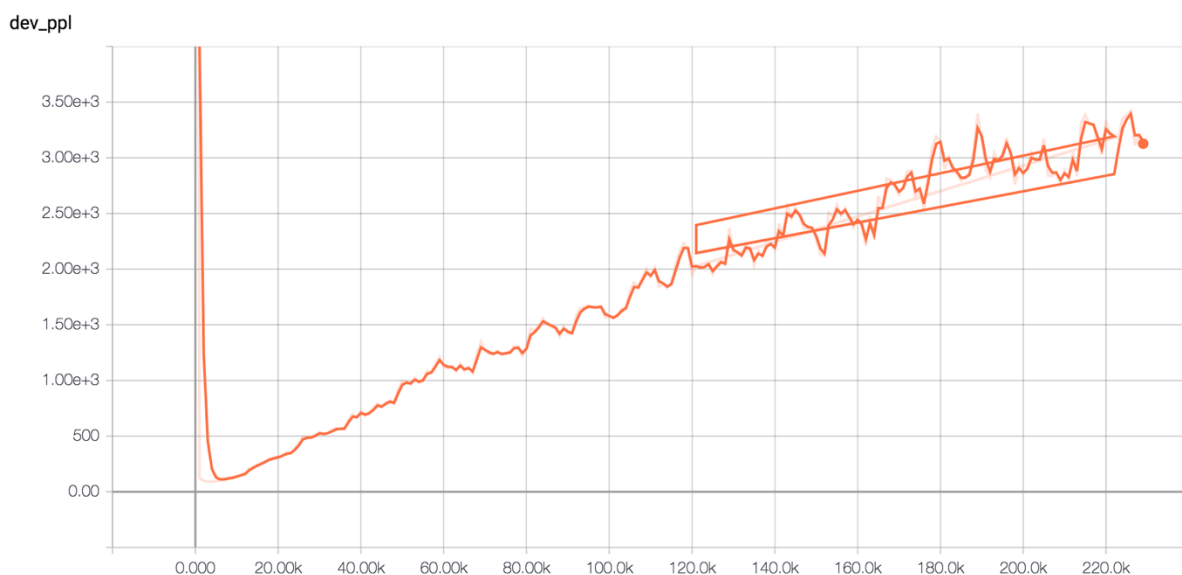


Figure 21: Shows how the perplexity developed during training of the chatbot model over 229 000 steps

The Figure 21 shows a graph of how the perplexity for the model developed during 229 000 steps. Perplexity variable seemed to increase over time, which is a bad sign. BLEU is a measure of how effective the model is at predicting the output from the sample. This happens with the use of language translation.

train_loss

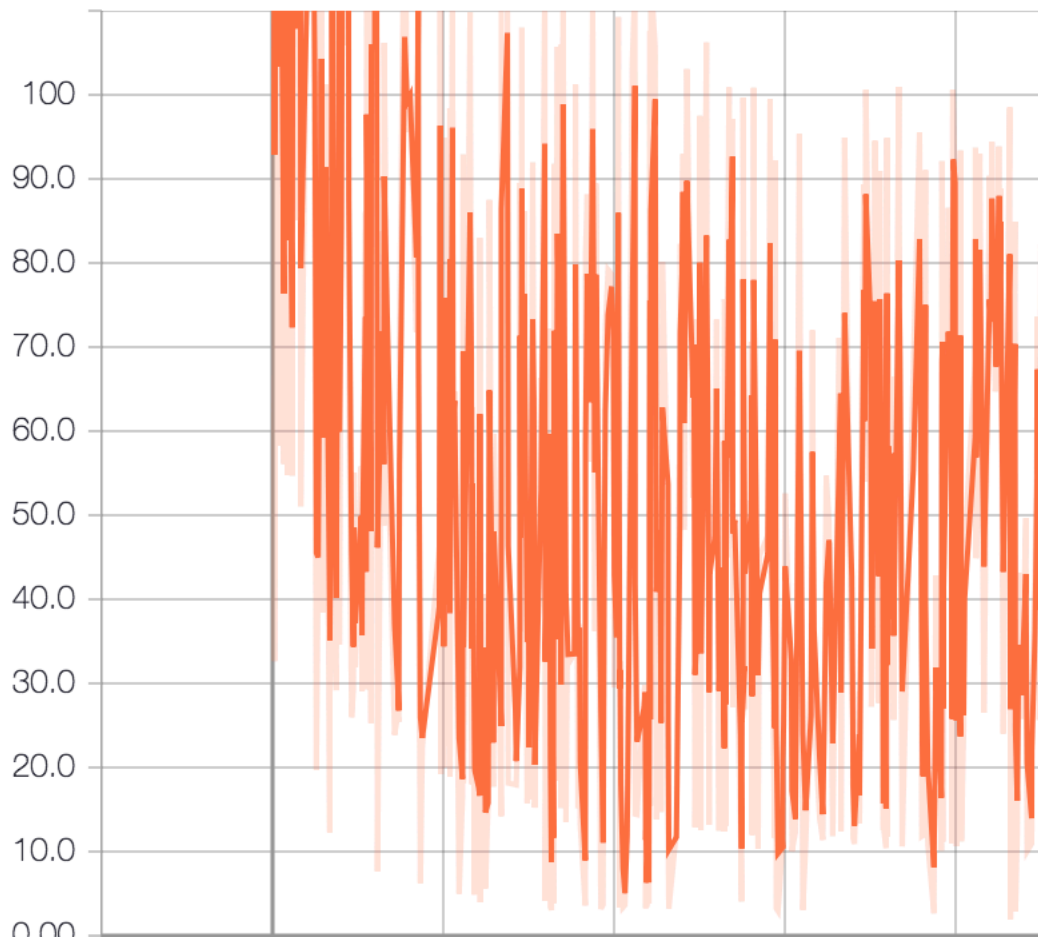


Figure 22: Shows how the loss developed during 229 000 steps of training. Loss along the y-axis, amount of steps along the x-axis.

Figure 22 shows how the loss developed during 229 000 steps of training. Loss is how far the output layer of the neural network was compared to our sample data. And the loss here seems to be dropping as the steps go further, in general.

On my computer it took 31 days for the model to train to step number 229 000.

I was supposed to test the model with students to see what they thought of the answers they received from the chatbot, but I did not have the time to set this up.

The output I got from the demo model when I asked it some questions was like the following:

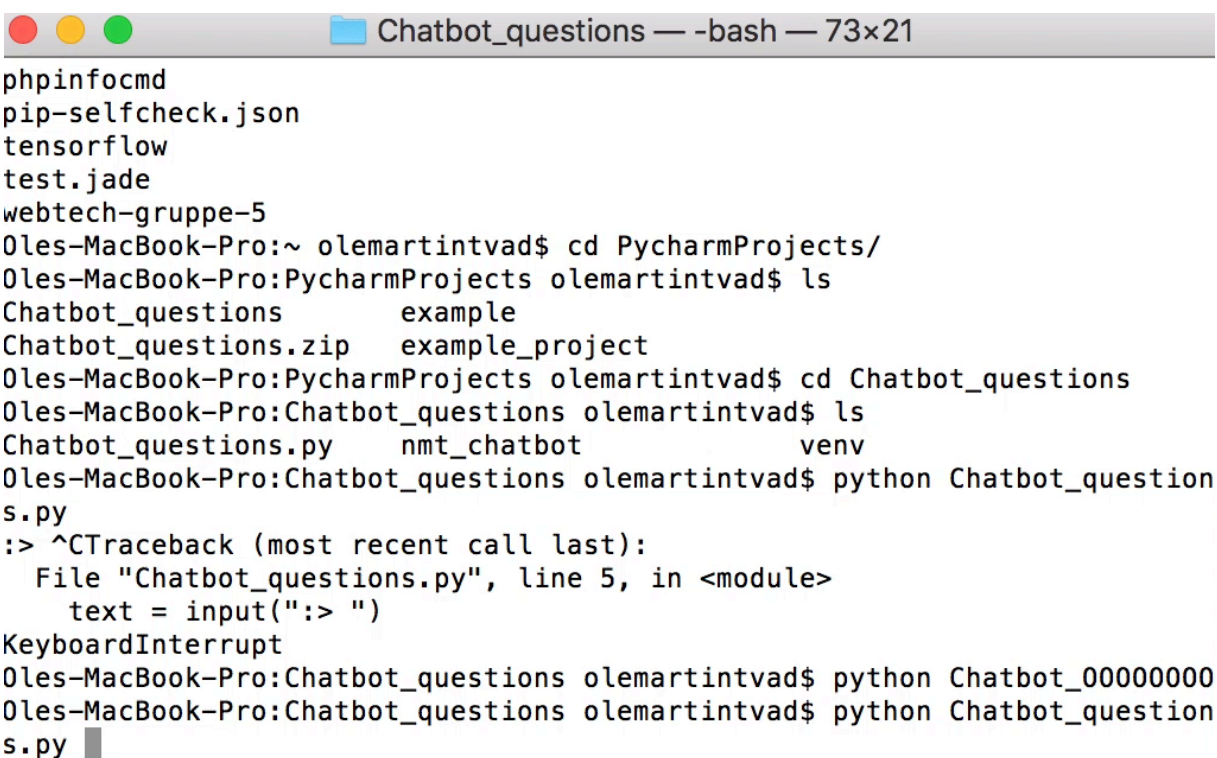
```

:> hello
^^^^^^^^^^
:> hey
Thanks!
:> hi
^^
:> how are you?
I don't know.

```

And this result is not the best, because the model was trained with 1 million pairs of comment-reply pairs. The model could only answer simple yes/no questions, while it would not give any relevant answers to questions containing several words.

The following video shows example of what happens when I ask the demo chatbot some questions:



```

phpinfocmd
pip-selfcheck.json
tensorflow
test.jade
webtech-gruppe-5
Oles-MacBook-Pro:~ olemartintvad$ cd PycharmProjects/
Oles-MacBook-Pro:PycharmProjects olemartintvad$ ls
Chatbot_questions      example
Chatbot_questions.zip  example_project
Oles-MacBook-Pro:PycharmProjects olemartintvad$ cd Chatbot_questions
Oles-MacBook-Pro:Chatbot_questions olemartintvad$ ls
Chatbot_questions.py  nmt_chatbot      venv
Oles-MacBook-Pro:Chatbot_questions olemartintvad$ python Chatbot_questions.py
:> ^CTraceback (most recent call last):
  File "Chatbot_questions.py", line 5, in <module>
    text = input("> ")
KeyboardInterrupt
Oles-MacBook-Pro:Chatbot_questions olemartintvad$ python Chatbot_00000000
Oles-MacBook-Pro:Chatbot_questions olemartintvad$ python Chatbot_questions.py

```

5 DISCUSSION

5.1 Sources

One of the first things I did before or when writing this thesis was to make sure the information gathered was trustworthy. The purpose of this was to try and understand the theory basic theory of creating a chatbot. Important concepts about chatbots were explained in different papers written by students at other universities, in videos found on

YouTube, as well as other websites. It was a jungle of information, and difficult to decide exactly which sources to actually trust.

Along the way, I changed my mind on which was the better way to build a chatbot. There were some YouTube videos which briefly explained how to develop a chatbot, but really went quickly over on how to do it. From my understanding these videos were created for people who either already had a lot of knowledge about programming or just were very fascinated in programming.

I came across Sentdex' website [17], which I found very interesting and ended up teaching me a lot about Python programming, about creation of a chatbot, as well as about TensorFlow and other libraries that chatbots are built upon.

5.2 Technological choices

5.2.1 Programming language (Python)

I decided to use Python for developing a chatbot, because this programming language can be used for a wide variety of purposes. Python is used for creating games, analyzing data, controlling robots and hardware, creating GUIs as well as websites [17]. I am more interested in the data analysis part, which can be used to see the progress of the chatbot training live as it is happening.

5.3 The project

I did not try to use any software development methods for writing this bachelor thesis. The reason for this is because I did not think it would be necessary, as I was working on this project by myself only. This was not the best decision for the final outcome of the thesis. A plan for how much time needed for development of code, as well as another plan for time needed for writing the thesis should have been made very early on in the semester.

I spent too much time on the coding part of things, like debugging and testing if the code worked like it should. This decision left me with very little amount of time to write a good thesis.

5.4 The use of corpus data

The choice of using Reddit corpus data did not lead to creation of a useful or good chatbot that could reply to questions in a good manner. From this project I learned that the decision of selecting a certain type of training corpus is important for how well a chatbot will respond to questions issued by the human-user.

Reddit corpus data did not make the chatbot reply to questions issued in a good way. The reason for this may be that a lot of strange or unusual comments are posted on this website. Natural language is not used in the whole website, a lot of abbreviations may be used in the comments and also curse words could be present in certain threads.

I think maybe the use of dialogue corpus from a movie would be better for making a more accurate chatbot. Because these dialogues do not include special characters or unknown tokens, like smiley faces. The chatbot would understand these sequences more easily.

5.5 Review of the practical work of the project

At the end of the project I got a chatbot which can reply to simple text input from the user, but not so much more than that. The chatbot has only been trained with general comments and replies from Reddit, which means it will not be able to give good replies about educational questions for example about university subjects, professors or anything like that.

6 REFERENCES

- [1] Schlicht, M. (2016, 20th April). The Complete Beginner's Guide To Chatbots. [Internet Magazine]. [Cited 30th of May 2018], Available from <https://chatbotsmagazine.com/the-complete-beginner-s-guide-to-chatbots-8280b7b906ca>
- [2] («Python Programming Tutorials», cited on 29th of November 2018) Available from: <https://pythonprogramming.net/introduction-to-python-programming/>
- [3] Nushant, Shikla (2017). MEAP Edition Machine Learning with TensorFlow version 10, Manning Publications Co. page 134
- [4] Nushant, Shikla (2017). MEAP Edition Machine Learning with TensorFlow version 10, Manning Publications Co. page 182-185
- [5] Nushant, Shikla (2017). MEAP Edition Machine Learning with TensorFlow version 10, Manning Publications Co. page 194-197
- [6] colorama. [Internet]. [Cited 24th of May 2018], Available from <https://pypi.org/project/colorama/>
- [7] Kukiela, D. (2018a). *nmt-chatbot: NMT Chatbot*. Python. [Internet]. [Cited 1st of June 2018], Available from <https://github.com/daniel-kukiela/nmt-chatbot> (Original work published 2017)
- [8] Python (programming language). In *Wikipedia*. Cited 27th May 2018 from: [https://en.wikipedia.org/w/index.php?title=Python_\(programming_language\)&oldid=843140763](https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=843140763)
- [9] («PyCharm», 2018) [Internet]. [Cited on 30th of November 2018]. Available from: <https://en.wikipedia.org/wiki/PyCharm>
- [10] "TensorFlow", 2018 [Internet]. [Cited on 1st of June 2018]. Available from: <https://www.tensorflow.org/>
- [11] (Kukiela, 2017/2018a) [Internet] [Cited on 30th of November 2018]. Available from: <https://github.com/daniel-kukiela/nmt-chatbot/blob/master/README.md>
- [12] Python Programming Tutorials. [Internet]. [Cited on 29th of November 2018]. Available from: <https://pythonprogramming.net/introduction-to-python-programming/>
- [13] Using the Requests Library in Python. [Internet] [Cited 1st of June 2018]. Available from <http://www.pythonforbeginners.com/requests/using-requests-in-python>
- [14] What is a Regex (Regular Expression)? [Internet]. [Cited 1st of June 2018]. Available from <https://www.computerhope.com/jargon/r/regex.htm>
- [15] Godara, A. (2016, november 28). A brief history of bots. Cited 1. juni 2018, from <https://chatbotsmagazine.com/a-brief-history-of-bots-9c45fc9b8901>

ATTACHMENTS

Attachment 1 <https://github.com/daniel-kukiela/nmt-chatbot> (Kukiela, 2017/2018b)