DEVELOPING A GENERATIVE CHATBOT

By

Muhammad Ali
Benjamin Schlup
Chinedu Abonyi
Victor Manuel Armenta-Valdés

Submitted to

The University of Liverpool

MSc Artificial Intelligence

CSCK507 - Natural Language Processing and Understanding

Word Count: 1,481

23/05/2022

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. Introduction

Natural language processing (NLP) is a subset of artificial intelligence (AI) that utilises machine learning (ML) algorithms to process and analyse linguistic data (Thomas, 2019). Typical applications of NLP include text generation, machine translation and sentiment analysis (Thomas and Latha, 2021). Another popular NLP application is question-answering (QA) systems based on human input (Acheampong *et al.*, 2016). QA systems are divided into closed and open domain (Shah, 2020). The former is limited to a pre-set scope of questions, while the latter generates responses based on previously stored data and interactions (B, 2020). Hence, the latter is also known as a generative chatbot (Shah, 2020).

The University of Liverpool assigned the postgraduate cohort to develop and compare two sequence-to-sequence chatbot models. This report outlines this group's modelling process and outcomes. Section 2 shares a literature review of the WikiQA dataset and generative chatbots. Section 3 outlines the methodology for the proposed chatbot models. Section 4 analyses the models' performances. Finally, Section 5 concludes the report and offers ideas for future improvements.

2. Literature Review

2a. Dataset

The dataset used in this study is the Microsoft Research WikiQA Corpus, a publicly available, open-domain dataset primarily used in research (Shahane, 2021). The dataset consists of questions and answers from Bing queries and Wikipedia articles, as shown in Figure 1 (Yang, Yih and Meek, 2015).

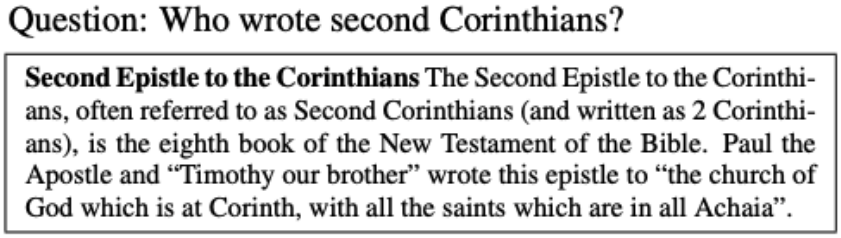

Figure 1: Sample WikiQA question and answer (Yang, Yih and Meek, 2015)

Yang, Yih and Meek (2015) created the WikiQA dataset as a substitute for QASent, a benchmark QA dataset in research. Table 1 illustrates statistics from the WikiQA dataset, and Table 2 compares both datasets.

	Train	Dev	Test	Total
# of ques.	2,118	296	633	3,047
# of sent.	20,360	2,733	6,165	29,258
# of ans.	1,040	140	293	1,473
Avg. len. of ques.	7.16	7.23	7.26	7.18
Avg. len. of sent.	25.29	24.59	24.95	25.15
# of ques. w/o ans.	1,245	170	390	1,805

Table 1: WikiQA dataset statistics (Yang, Yih and Meek, 2015)

Class	QASENT	WIKIQA
Location	37 (16%)	373 (12%)
Human	65 (29%)	494 (16%)
Numeric	70 (31%)	658 (22%)
Abbreviation	2 (1%)	16 (1%)
Entity	37 (16%)	419 (14%)
Description	16 (7%)	1087 (36%)

Table 2: Comparison between QASent and WikiQA question classes (Yang, Yih and Meek, 2015)

The group conducted exploratory data analysis (EDA) on the WikiQA dataset ahead of the model design. However, since EDA is not the primary scope of the chatbot exercise, this paper will not delve into details and instead focus on the NLP models.

2b. Sequence-to-sequence

Sequence-to-sequence (seq-2-seq) models are a type of recurrent neural network (RNN) class used in complex NLP exercises, such as machine translation and QA systems (Singh, 2020). Generally, seq-2-seq models are utilised in natural language generation (NLG) tasks, as shown in Figure 2. (Chollet, 2017).

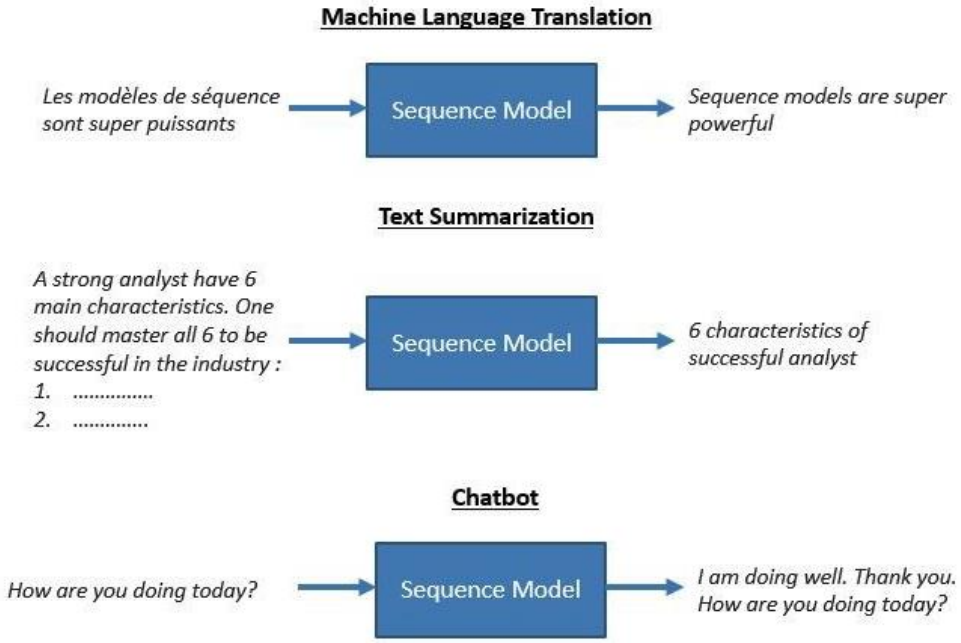

Figure 2: Sample seq-2-seq tasks (Singh, 2020)

In a conventional RNN architecture, input and output vectors are fixed with same lengths; however, this architecture is not suitable for NLG tasks where the input and output sizes differ (Chollet, 2017). As such, seq-2-seq models solve this problem by using an encoder and decoder to process different length vectors for NLG (Alla, 2021). Figure 3 demonstrates the basic seq-2-seq architecture.

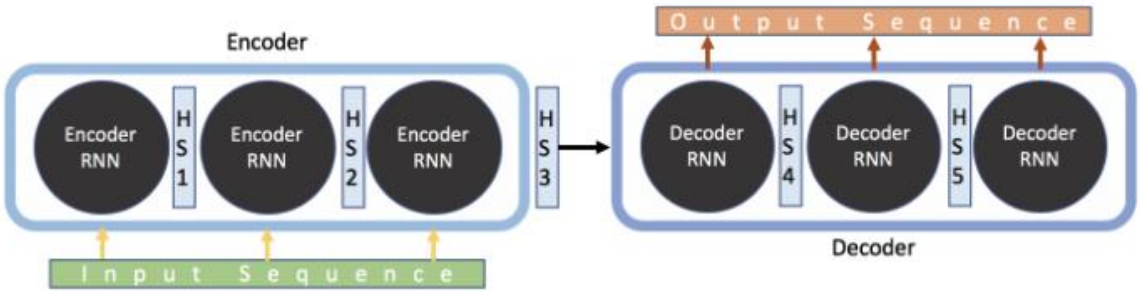

Figure 3: The seq-2-seq encoder and decoder architecture (Dugar, 2019)

In seq-2-seq models, the encoder and decoder are RNN or variants thereof, like long short-term memory networks (LSTM) (Chollet, 2017). The encoder processes the input sequence as a hidden state vector and feeds this into the decoder to produce the output sequence (Alla, 2021).

A problem with conventional seq-2-seq is that the model can forget information at the encoder level due to data compression into a single vector (Ferdjouni, 2021). Bahdanau, Cho and Bengio (2014) proposed the seq-2-seq with attention variation that conserves input contexts by compressing multiple hidden states into a context vector and feeding this into a new attention hidden state layer. Figure 4 demonstrates the seq-2-seq with attention model's architecture.

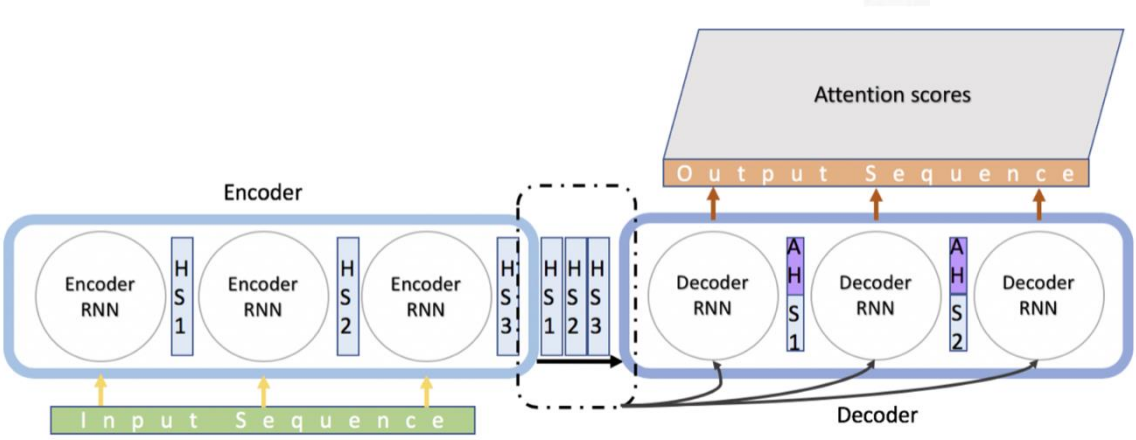

Figure 4: The seq-2-seq with attention model (Dugar, 2019)

This report compares the performance of both conventional seq-2-seq and attention models for generative chatbots.

2c. Existing Seq-2-Seq Models

Chatbots are intelligent software that provides assistance in various fields, such as medicine and education (Sojasingarayar, 2020). Seq-2-seq algorithms are commonly used in QA chatbots due to their sequential learning and predictive properties (Manning, n.d.). For example, Khin and Soe (2020) propose a seq-2-seq with an attention layer model for parents and teachers to inquire about Myanmar universities. Figure 5 demonstrates the model's process flow, and Table 3 illustrates the solution's sample outputs.

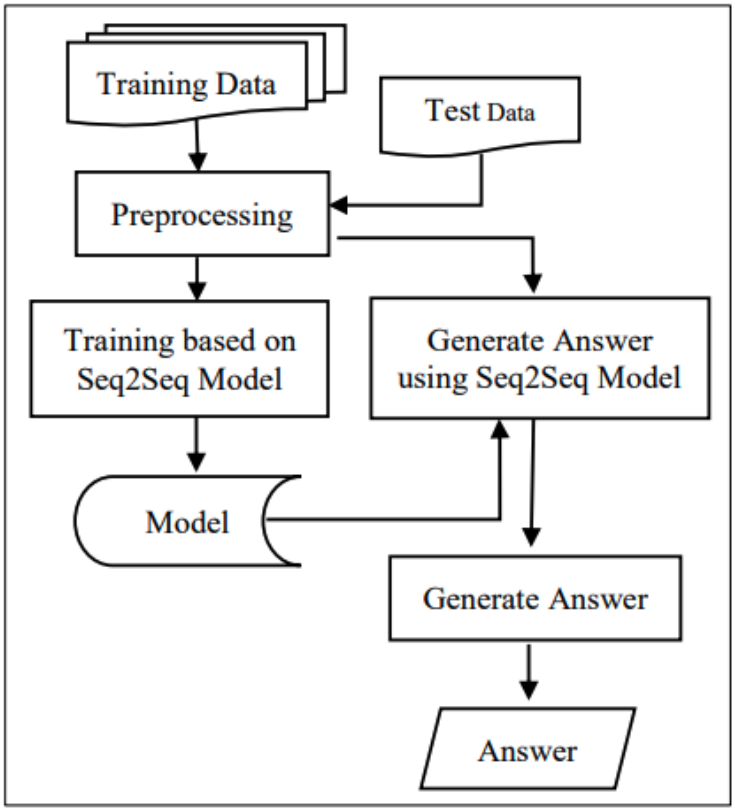

Figure 5: Khin and Soe's (2020) proposed process flow

Question: (Human)	မင်္ဂလာ ပါ ရှင့်	Hello
Answer: (Bot)	ဟုတ်ကဲ့ မင်္ဂလာ ပါ	Yes, hello.
Question: (Human)	ရန်ကုန် ကွန်ပျူတာ တက္ကသိုလ် ရဲ့ လိပ်စာ ကို သိ ချင် ပါတယ်	I want to know the address of University of Computer Studies, Yangon.
Answer: (Bot)	အမှတ် ၄ လမ်းမကြီး၊ ရွှေပြည်သာ မြို့၊ ရန်ကုန်တိုင်း ဖြစ်ပါတယ်	It is No.4 Main Street, Shwe Pyi Thar Township, Yangon Region.
Question: (Human)	အဆောင် လျှောက် ချင် လို ဘယ်လို လျှောက် ရ မ လဲ	How to apply for the accommodation?
Answer: (Bot)	ကျောင်းသား ရေးရာ ဌာန မှာ အသေးစိတ် သွားရောက် စုံစမ်း နိုင် ပါတယ်	Please inquire at Student Registration Department.
Question: (Human)	အမှတ် စာရင်း ထုတ် ဖို ဘာတွေ လိုအပ် သလဲ	What are needed for grading?

Table 3: Sample output from Khin and Soe's (2020) proposed system

Abdullahi *et al.* (2019) propose a seq-2-seq with an attention layer model using gated recurrent units (GRU), an LSTM variant, trained on the Cornell movie dialogue corpus. Figure 6 demonstrates the solution's accuracy and loss plots, and Figure 7 illustrates a sample conversation scenario.

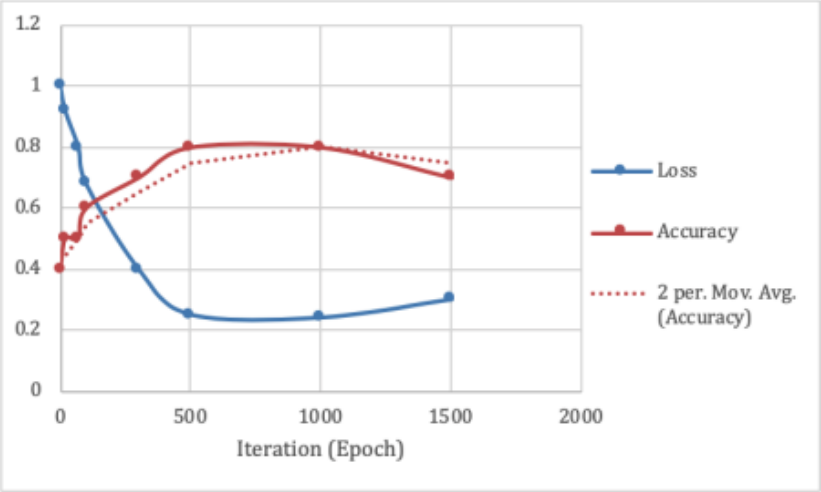

Figure 6: Abdullahi et al.'s (2019) proposed model's accuracy and loss plots

```
HUMAN → hi
BOT → hi
HUMAN → how are you?
BOT → i ' m okay .
HUMAN → really?
BOT → i don ' t know what to say .
HUMAN → how're you?
BOT → i ' m fine .
HUMAN → where're you from?
BOT → california .
HUMAN → which city?
BOT → let's talk about something
   else.
HUMAN → you don't like california?
BOT → oh . . .
HUMAN → why not?
BOT → i can ' t.
HUMAN → how old are you?
BOT → eighteen.
```

Figure 7: Abdullahi et al.'s (2019) proposed model's sample conversation

3. Methodology

Schneider's (2020) seq-2-seq model inspired the group's chatbot solution.
The first step is importing the Python libraries, as shown in Figure 8.

```
# Imports
import os
import re
import urllib.request
import zipfile

import numpy as np
import pandas as pd

from IPython.display import Image

from tensorflow.keras.activations import softmax
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical, plot_model

from keras_preprocessing.text import Tokenizer

import matplotlib.pyplot as plt

from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
```

Figure 8: Imported Python libraries

The next step is importing the WikiQA Corpus dataset from Microsoft using the *urllib* module. The dataset is already divided into training, validation and test sets, as shown in Figures 9 & 10.

```
# Download data: If link does not work any longer, access file manually from here: https://www.
urllib.request.urlretrieve("https://download.microsoft.com/download/E/5/F/E5FCFCEE-7005-4814-85
```

```
('WikiQACorpus.zip', <http.client.HTTPMessage at 0x7fb59b673910>)
```

```
# Extract files
with zipfile.ZipFile('WikiQACorpus.zip', 'r') as zipfile:
   zipfile.extractall()
```

```
# Import questions and answers: training, validation and test datasets
train_df = pd.read_csv( f'./WikiQACorpus/WikiQA-train.tsv', sep='\t', encoding='ISO-8859-1')
val_df = pd.read_csv( f'./WikiQACorpus/WikiQA-dev.tsv', sep='\t', encoding='ISO-8859-1')
test_df = pd.read_csv( f'./WikiQACorpus/WikiQA-test.tsv', sep='\t', encoding='ISO-8859-1')
```

Figure 9: Dataset download

```
# Quality checks and exploratory data analysis removed: dataset has proven clean
# Print gross volumes:
print(f'Gross training dataset size: {len(train_df)}')
print(f'Gross validation dataset size: {len(val_df)}')
print(f'Gross test dataset size: {len(test_df)}')
```

```
Gross training dataset size: 20347
Gross validation dataset size: 2733
Gross test dataset size: 6116
```

Figure 10: Dataset sizes

The group then configured several variables to assist with the pre-processing and transformation parameters, such as setting maximum token length and handling duplicate and invalid questions. Figure 11 demonstrates the parameters in Python.

```
# The dataset includes invalid answers (labelled 0) and some questions
# even have no valid answer at all: Switches allow test runs excluding invalid
# answers.
# Note that the assignment says that answers must be provided by the chatbot:
# there is no mention that answers must be correct!
train_with_invalid_answers = True
validate_with_invalid_answers = True
test_questions_without_valid_answers = True

# The dataset contains questions with multiple valid answers
train_with_duplicate_questions = True
validate_with_duplicate_questions = True
test_with_duplicate_questions = True

# Configure the tokenizer
vocab_size_limit = 6000 + 1 # set this to None if all tokens from training shall be included (add one to number of tokens)
vocab_include_val = False   # set this to True if tokens from validation set shall be included in vocabulary
vocab_include_test = False  # set this to True if tokens from test set shall be included in vocabulary
oov_token = 1               # set this to None if out-of-vocabulary tokens should be removed from sequences
remove_oov_sentences = True # set this to True if any sentences containing out-of-vocabulary tokens should be removed from training,

# Limit sentence lengths // not yet implemented
max_question_tokens = None  # set this to None if no limit on question length
max_answer_tokens = None    # set this to None if no limit on answer length
```

Figure 11: Pre-processing parameters

Based on these parameters, the group normalised the questions, counted the number of tokens, dropped sentences exceeding the max token length, and handled invalid and duplicate questions (depending on notebook parameters). Figure 12 demonstrates the associated code.

```
# Derive normalized questions and answers and count number of tokens
for df in [train_df, val_df, test_df]:
    df.loc[:,'norm_question'] = [ re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", q).lower().strip() for q in df['Question'] ]
    df.loc[:,'norm_answer'] = [ '_START_ '+re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", s).lower().strip()+' _STOP_' for s in df['Sentence']]
    df['question_tokens'] = [ len(x.split()) for x in df['norm_question'] ]
    df['answer_tokens'] = [ len(x.split()) for x in df['norm_answer'] ]
```

```
# Drop sentences which are too long
for df in [train_df, val_df, test_df]:
    if max_question_tokens is not None:
        df.drop(df[df['question_tokens']>max_question_tokens].index, inplace=True)
    if max_answer_tokens is not None:
        df.drop(df[df['answer_tokens']>max_answer_tokens+2].index, inplace=True)
```

```
# Remove q/a pairs depending on configuration of the notebook
if not train_with_invalid_answers:
    train_df = train_df[train_df['Label'] == 1]
if not validate_with_invalid_answers:
    val_df = val_df[val_df['Label'] == 1]
if not test_questions_without_valid_answers:
    test_df = test_df[test_df['Label'] == 1]
```

```
# Remove duplicate questions in case configured to do so
if not train_with_duplicate_questions:
    train_df.drop_duplicates(subset=['Question'], inplace=True)
if not validate_with_duplicate_questions:
    validate_df.drop_duplicates(subset=['Question'], inplace=True)
if not test_with_duplicate_questions:
    test_df.drop_duplicates(subset=['Question'], inplace=True)
```

Figure 12: Initial pre-processing steps

Next, the group tokenised and padded the data for the seq-2-seq algorithms. Figures 13 & 14 demonstrate these steps.

```
# Data preparation:
# Tokenization:
# Reconsider adding digits to filter later, as encoding of numbers may create excessive vocabulary
# Also check reference on handling numbers in NLP: https://arxiv.org/abs/2103.13136
# Note that I do not yet train the tokenizer on validation and test datasets - should be challenged.
# my be added to Tokenizer filters=target_regex = '!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\''

if remove_oov_sentences:
    oov_token = None
tokenizer = Tokenizer(num_words=vocab_size_limit, oov_token=oov_token)

tokenizer.fit_on_texts(train_df['norm_question'] + train_df['norm_answer'])
if vocab_include_val:
    tokenizer.fit_on_texts(val_df['norm_question'] + val_df['norm_answer'])
if vocab_include_test:
    tokenizer.fit_on_texts(test_df['norm_question'] + test_df['norm_answer'])

vocab_size = len(tokenizer.word_index) + 1
if vocab_size_limit is not None:
    vocab_size = min([vocab_size, vocab_size_limit])
print(f'Vocabulary size based on training dataset: {vocab_size}')

for df in [train_df, val_df, test_df]:
    # Tokenize
    df['tokenized_question'] = tokenizer.texts_to_sequences(df['norm_question'])
    df['tokenized_answer'] = tokenizer.texts_to_sequences(df['norm_answer'])

    # Optionally remove sentences with out-of-vocabulary tokens
    if remove_oov_sentences:
        df.drop(df[df['question_tokens']!=df['tokenized_question'].str.len()].index, inplace=True)
        df.drop(df[df['answer_tokens']!=df['tokenized_answer'].str.len()].index, inplace=True)
```

```
Vocabulary size based on training dataset: 6001
```

Figure 13: Data tokenisation

```
# Transform data for training and validation by aligning lengths (i.e. padding)
maxlen_questions = max(len(t) for t in train_df['tokenized_question'].to_list())
maxlen_answers = max(len(t) for t in train_df['tokenized_answer'].to_list())

train_encoder_input_data = pad_sequences(train_df['tokenized_question'], maxlen=maxlen_questions, padding='post')
val_encoder_input_data = pad_sequences(val_df['tokenized_question'], maxlen=maxlen_questions, padding='post')
print(f'Encoder input data shape: {train_encoder_input_data.shape}')

train_decoder_input_data = pad_sequences(train_df['tokenized_answer'], maxlen=maxlen_answers, padding='post')
val_decoder_input_data = pad_sequences(val_df['tokenized_answer'], maxlen=maxlen_answers, padding='post')
print(f'Decoder input data shape: {train_decoder_input_data.shape}')

tokenized_answers = [ ta[1:] for ta in train_df['tokenized_answer'] ]
padded_answers = pad_sequences(tokenized_answers, maxlen=maxlen_answers, padding='post')
train_decoder_output_data = to_categorical(padded_answers, vocab_size)
tokenized_answers = [ ta[1:] for ta in val_df['tokenized_answer'] ]
padded_answers = pad_sequences(tokenized_answers, maxlen=maxlen_answers, padding='post')
val_decoder_output_data = to_categorical(padded_answers, vocab_size)
print(f'Decoder output data shape: {train_decoder_output_data.shape}')
```

```
Encoder input data shape: (2181, 21)
Decoder input data shape: (2181, 52)
Decoder output data shape: (2181, 52, 6001)
```

Figure 14: Data padding

The resulting net dataset decreased due to the pre-processing steps, as shown in Figure 15.

```
# Print net volumes
print(f'Net training dataset size: {len(train_df)}')
print(f'Net validation dataset size: {len(val_df)}')
print(f'Net test dataset size: {len(test_df)}')
```

```
Net training dataset size: 2181
Net validation dataset size: 108
Net test dataset size: 252
```

Figure 15: Net dataset sizes after pre-processing

Following the pre-processing steps, the next phase was building the seq-2-seq models. Instead of conventional RNN, the group used LSTM as the encoder and decoder components. Figure 16 demonstrates the model's parameters.

```
# Model parameters
lstm_units = 150
embedding_units = 200
encoder_lstm_dropout = 0.2
encoder_lstm_recurrent_dropout = 0.2
```

Figure 16: Model parameters

Following this, the group built the two seq-2-seq models: with and without the attention layer. Figures 17 & 18 demonstrate these models' architectures.

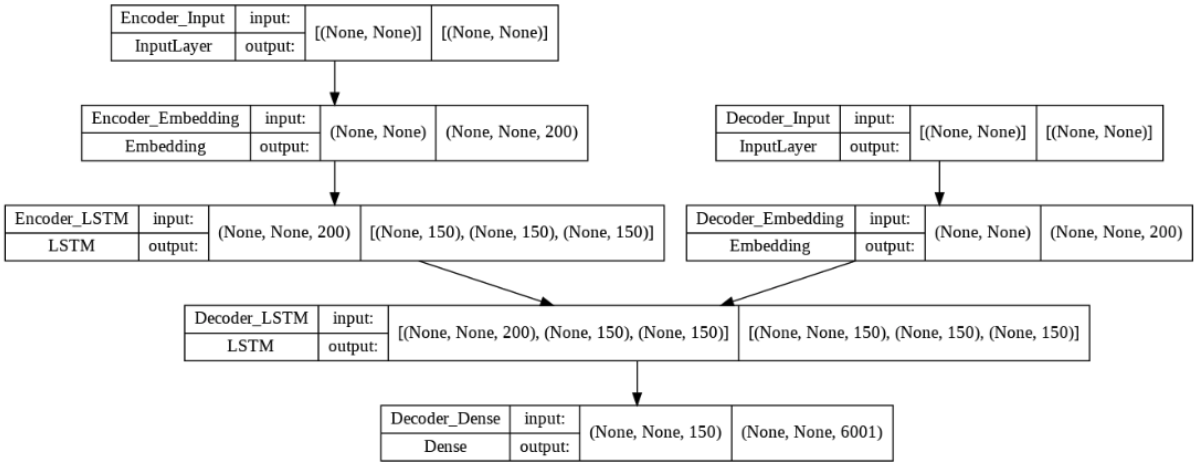

Figure 17: Model one architecture (without attention layer)

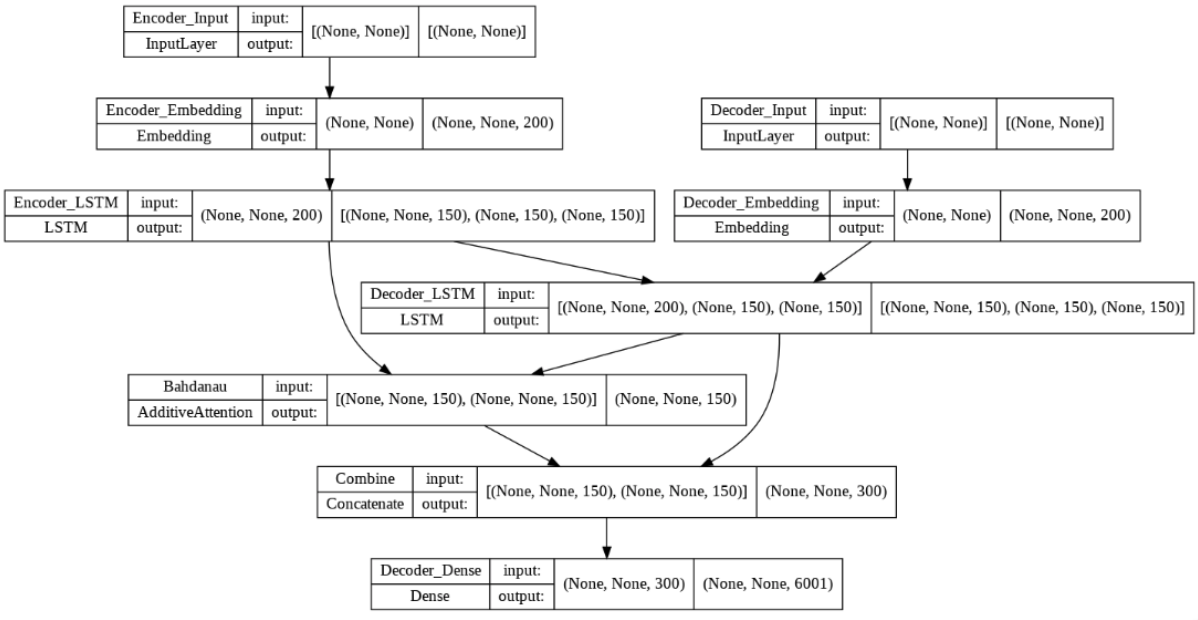

Figure 18: Model two architecture (with attention layer)

Finally, the group trained the models in batches of fifty questions over one hundred epochs. Figures 19 & 20 demonstrate accuracy and loss plots for both models.

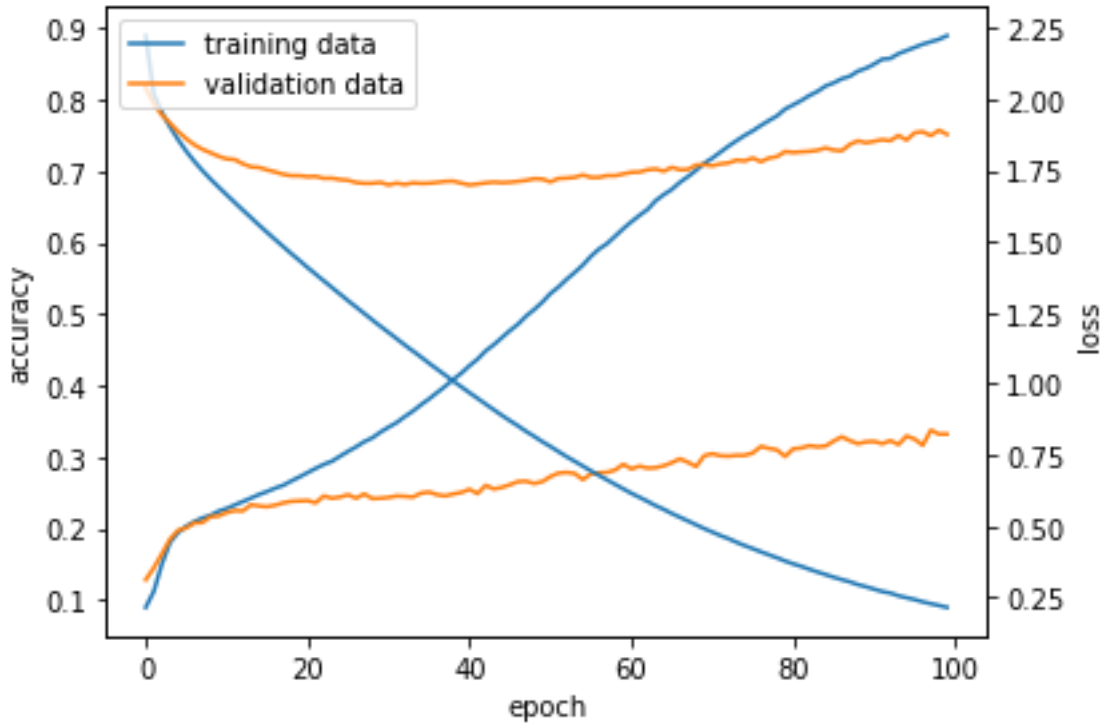

Figure 19: Model One accuracy and loss plots

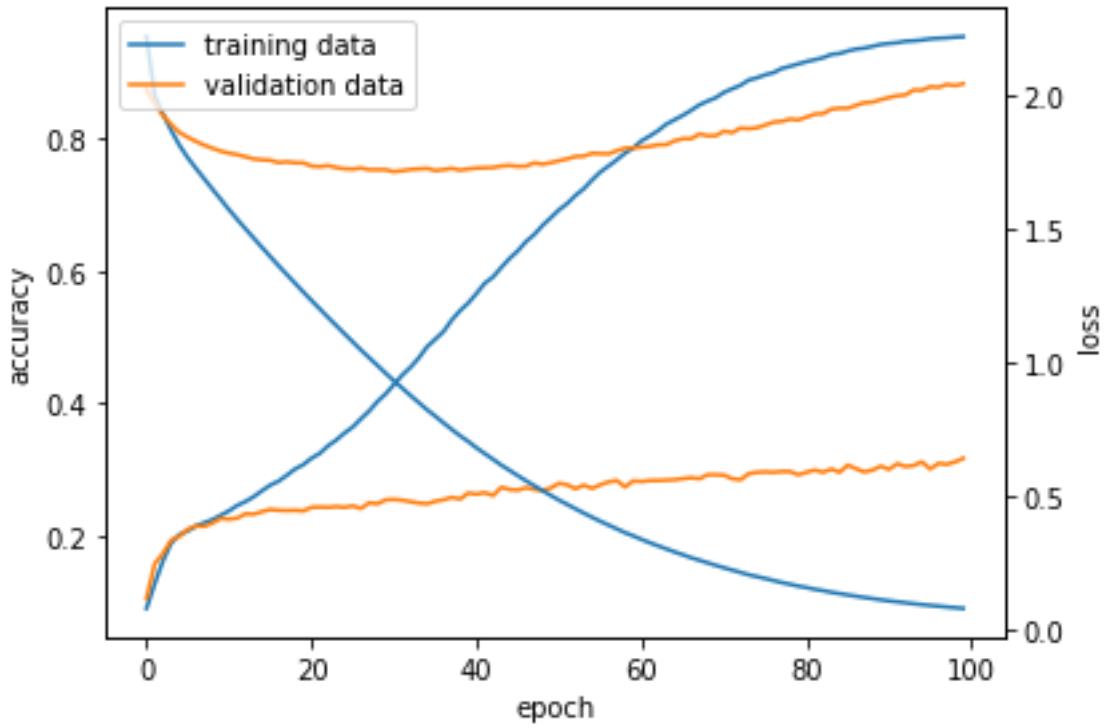

Figure 20: Model Two accuracy and loss plots

While performance on the training data is improving strongly, accuracy on the (unseen) validation data remains low.

4. Validation and Results

After training the models, the group built separate encoder and decoder models for inferencing. Figure 21 demonstrates the models' architecture.

```
Model: "Inference_Encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Encoder_Input (InputLayer)  [(None, None)]            0

 Encoder_Embedding (Embeddin  (None, None, 200)        1200200
 g)

 Encoder_LSTM (LSTM)         [(None, 150),             210600
                              (None, 150),
                              (None, 150)]

=================================================================
Total params: 1,410,800
Trainable params: 1,410,800
Non-trainable params: 0
_________________________________________________________________
Model: "Inference_Decoder"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 Decoder_Input (InputLayer)     [(None, None)]       0           []

 Decoder_Embedding (Embedding)  (None, None, 200)    1200200     ['Decoder_Input[0][0]']

 input_1 (InputLayer)           [(None, 150)]        0           []

 input_2 (InputLayer)           [(None, 150)]        0           []

 Decoder_LSTM (LSTM)            [(None, None, 150),  210600      ['Decoder_Embedding[0][0]',
                                 (None, 150),                     'input_1[0][0]',
                                 (None, 150)]                     'input_2[0][0]']

 Decoder_Dense (Dense)          (None, None, 6001)   906151      ['Decoder_LSTM[1][0]']

==================================================================================================
Total params: 2,316,951
Trainable params: 2,316,951
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 21: Encoder and Decoder Inference Architectures

Next, the group prepared questions for testing by defining functions that tokenise questions, predict answers and score the models. Figure 22 demonstrates the code for tokenising questions, and Figure 23 outlines the code for scoring answers.

```
# Prepare question for inferencing

def tokenize(question):
    words = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", question).lower().split()
    tokens_list = list()
    for current_word in words:
        result = tokenizer.word_index.get(current_word, '')
        if result != '':
            tokens_list.append(result)
        else:
            print(f'Warning: out-of-vocabulary token \'{current_word}\'')
            if oov_token is not None:
                tokens_list.append(oov_token)

    return pad_sequences([tokens_list],
                         maxlen=maxlen_questions,
                         padding='post')
```

Figure 22: Tokenise question function

```
# Predict answer and compare to ground truth options

def predict_answer(question, qa_df=None):
    states_values = enc_model.predict(tokenize(question))
    empty_target_seq = np.zeros((1, 1))
    empty_target_seq[0, 0] = tokenizer.word_index['start']

    decoded_answer = ''
    while True:
        dec_outputs, h, c = dec_model.predict([empty_target_seq] + states_values)
        sampled_word_index = np.argmax(dec_outputs[0, -1, :])
        sampled_word = None
        for word, index in tokenizer.word_index.items():
            if sampled_word_index == index:
                if word != 'stop':
                    decoded_answer += ' {}'.format(word)
                sampled_word = word

        if sampled_word == 'stop' or len(decoded_answer.split()) > maxlen_answers:
            break

        empty_target_seq = np.zeros((1, 1))
        empty_target_seq[0, 0] = sampled_word_index
        states_values = [h, c]

    # Skip START token
    decoded_answer = decoded_answer[1:]

    print(f'Original question: {question}')
    print(f'Predicated answer: {decoded_answer}\n')

    if qa_df is not None:
        # The following should contain all acceptable answers
        reference_answers = qa_df.loc[qa_df['Question']==question, 'norm_answer'].to_list()
        reference_answers = [ re.sub(' +', ' ', answer[8:-7]) for answer in reference_answers]
        print(f'{reference_answers}')

        # Calculate BLEU score: Note that little differences may result from e.g.
        # spaces that were added to norm_answer when replacing punctuation earlier
        bleu_score = sentence_bleu(reference_answers, decoded_answer, smoothing_function=SmoothingFunction().method0)

        print(f'BLEU score: {bleu_score}\n')

    else:
        bleu_score = None

    return bleu_score
```

Figure 23: Predicting and scoring answers function

The group scored the predicted answers using the Bilingual Evaluation Understudy (BLEU) score, whose scores range from 0 to 1 (worst to best) (Shmueli, 2021). Figure 24 demonstrates a sample question with predicted and actual responses.

```
Original question: what spanish speaking countries have the most world cup titles
Predicated answer: the current champions are spain who won the 2010 tournament

['the championship has been awarded every four years since the inaugural tournament in 1930 except in 1942 and 1946 when it was not held b
ecause of the second world war', 'the current champions are spain who won the 2010 tournament', 'brazil have won five times and they are t
he only team to have played in every tournament']
BLEU score: 1.0
```

Figure 24: Sample question with answers and BLEU score

The group tested seq-2-seq models with twenty trained questions, twenty unseen questions, and nine custom targeted validation questions. Overall, the seq-2-seq with attention model performed better on the trained questions with an average BLEU score of 0.976 (compared to 0.743 for the non-attention model). However, both models performed similarly poorly on unseen questions.

Nevertheless, validating the model with unseen questions is not a helpful practice due to the dataset's small variance and size. Therefore, future implementation of the chatbot models can include additional capabilities like answer triggering, which detects and returns only correct answers if they exist in a dataset (Acheampong *et al.*, 2016). Lastly, the group tested the robustness of the seq-2-seq models by compiling and rewording nine training and unseen questions. Figure 25 demonstrates the results from the seq-2-seq with attention model test.

```
TEST CASE 1: Accurate question from actual training dataset
Original question: How are glacier caves formed?
Predicated answer: a glacier cave is a cave formed within the ice of a glacier

TEST CASE 2: Varying the question from the actual training dataset
Original question: How are glacial caves formed?
Predicated answer: a glacier cave is a cave formed within the ice of a glacier

TEST CASE 3: Accurate question from actual training dataset
Original question: When was Apple Computer founded?
Predicated answer: the company was founded on april 1 1976 and incorporated as apple computer inc on january 3 1977

TEST CASE 4: Varying the question from the actual training dataset
Original question: When was Apple founded?
Predicated answer: the company was founded on april 15 2013

TEST CASE 5: Varying the question from the actual training dataset
Original question: When was Apple Computer established?
Predicated answer: the company was founded on april 1 1976 and incorporated as apple computer inc on january 3 1977

TEST CASE 6: Unseen question from test set
Original question: How many players on a side for a football game?
Predicated answer: the number of the 1930s is now only it but on the us of the 1930s 12 that have been named after the world war it is not
born it

TEST CASE 7: Varying the unseen question from test set
Original question: How many players per team in a football game?
Predicated answer: the usa of the 1930s has been awarded the 1930s by 2011 and the third and final of the 1930s that are now place college
s and had located by international countries and are of the united states

TEST CASE 8: Long question from training dataset
Original question: What group took home the award for best rock album at the Australian Recording Industry Association (ARIA) Music Award
s?
Predicated answer: for 2010 aria introduced public voted awards for the first time

TEST CASE 9: Varying the long question from the training dataset
Original question: What group took got the award for best album at the ARIA Music Awards?
Predicated answer: in 2010 aria introduced public voted in the united states with canada with the first as second after use after world wa
r ii
```

Figure 25: Seq-2-seq with attention test

5. Conclusion

The deliverables for this project include three notebooks in Python:

- exploratory data analysis on the WikiQA dataset;
- the conventional seq-2-seq model; and
- the seq-2-seq model with attention layer.

Overall, the seq-2-seq model with the attention layer was the higher performing model as expected. Although the group's chatbot effectively generates responses based on the WikiQA dataset, further developments can enhance the quality of the chatbot. For example, the model could incorporate answer triggering mentioned in Section 4.

Additionally, the seq-2-seq algorithm could integrate dropout layers to avoid overfitting (Brownlee, 2018). Lastly, the group could experiment with token dropout to enhance data augmentation (Zhang *et al.*, 2020). Together with other NLP additions, these enhancements can significantly improve the chatbot's performance in future study iterations.

References

Abdullahi, S. S., et al. (2019). 'Open Domain Chatbot Based on Attentive End-to-End Seq2Seq Mechanism', *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence.* Sanya, China, Association for Computing Machinery**:** 339–344. Available at: https://dl.acm.org/doi/abs/10.1145/3377713.3377773 (Accessed: 22 May 2022).

Acheampong, K. N., Pan, Z. H., Zhou, E. Q. and Li, X. Y. 'Answer triggering of factoid questions: A cognitive approach'. *2016 13th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 16-18 Dec. 2016, 33-37.

Alla, S. (2021) *Introduction to Encoder-Decoder Sequence-to-Sequence Models (Seq2Seq)*. Available at: https://blog.paperspace.com/introduction-to-seq2seq-models/ (Accessed: 18 May 2022).

Bennick, S. (2020) *What Is a Generative Chatbot? What You Need to Know*. Available at: https://yakbots.com/what-is-a-generative-chatbot-what-you-need-to-know/ (Accessed: 17 May 2022).

Bahdanau, D., Cho, K. and Bengio, Y. (2014) 'Neural Machine Translation by Jointly Learning to Align and Translate', *ArXiv,* 1409. Available at: https://arxiv.org/abs/1409.0473 (Accessed: 22 May 2022).

Brownlee, J. (2018) *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks* Available at: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/ (Accessed: 21 May 2022).

Chollet, F. (2017) *A ten-minute introduction to sequence-to-sequence learning in Keras*. Available at: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html (Accessed: 18 May 2022).

Dugar, P. (2019) *Attention — Seq2Seq Models* Available at: https://towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263 (Accessed: 18 May 2022).

Ferdjouni, M. (2021) *How Seq2Seq (Sequence to Sequence) Models got improved Into Transformers Using Attention Mechanism*. Available at: https://medium.com/analytics-vidhya/how-seq2seq-sequence-to-sequence-models-improved-into-transformers-using-attention-mechanism-9905fb88d5ef (Accessed: 19 May 2022).

Khin, N. N. and Soe, K. M. 'Question Answering based University Chatbot using Sequence to Sequence Model'. *2020 23rd Conference of the Oriental COCOSDA International Committee for the Co-ordination and Standardisation of Speech Databases and Assessment Techniques (O-COCOSDA)*, 5-7 Nov. 2020, 55-59.

Manning *Chapter 11. Sequence-to-sequence models for chatbots*. Available at: https://livebook.manning.com/book/machine-learning-with-tensorflow/chapter-11/102 (Accessed: 21 May 2022).

Schneider, N. (2020) *How To Design Seq2Seq Chatbot Using Keras Framework*. Available at: https://medium.com/swlh/how-to-design-seq2seq-chatbot-using-keras-framework-ae86d950e91d (Accessed: 19 May 2022).

Shah, D. (2020) *Generative chatbots using the seq2seq model!* Available at: https://towardsdatascience.com/generative-chatbots-using-the-seq2seq-model-d411c8738ab5 (Accessed: 17 May 2022).

Shahane, S. (2021) *Microsoft Research WikiQA Corpus*. Available at: https://www.kaggle.com/datasets/saurabhshahane/wikiqa-corpus?select=WikiQA-train.tsv (Accessed: 18 May 2022).

Shmueli, B. (2021) *NLP Metrics Made Simple: The BLEU Score* Available at: https://towardsdatascience.com/nlp-metrics-made-simple-the-bleu-score-b06b14fbdbc1 (Accessed: 21 May 2022).

Singh, P. (2020) *A Simple Introduction to Sequence to Sequence Models*. Available at: https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/ (Accessed: 18 May 2022).

Sojasingarayar, A. (2020) *Seq2Seq AI Chatbot with Attention Mechanism.*

Thomas, C. (2019) *Recurrent Neural Networks and Natural Language Processing.* Available at: https://towardsdatascience.com/recurrent-neural-networks-and-natural-language-processing-73af640c2aa1 (Accessed: 30 April 2022).

Thomas, M. and Latha, C. A. (2021) 'Sentimental analysis of transliterated text in Malayalam using recurrent neural networks', *Journal of Ambient Intelligence and Humanized Computing,* 12(6), pp. 6773-6780.

Yang, Y., Yih, W.-t. and Meek, C. (2015) 'WikiQA: A Challenge Dataset for Open-Domain Question Answering', *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, September: ACL - Association for Computational Linguistics.

Zhang, H., Qiu, S., Duan, X. and Zhang, M. (2020) 'Token Drop mechanism for Neural Machine Translation', *In Proceedings of the 28th International Conference on Computational Linguistics, pages 4298–4303, Barcelona, Spain (Online). International Committee on Computational Linguistics.* Available at: https://aclanthology.org/2020.coling-main.379/ (Accessed: 22 May 2022).