Implementation of a Chatbot using Jupyter Python

Thawatchai Sangthep | P2681054 | De Montfort University

IMAT5118 - Natural Language Processing based on Deep Learning

Introduction

This project deals with the implementation of a Chatbot using Jupyter Python. The chatbot has a structure based on the concept of Natural Language Processing (NLP). It is a field that focuses on making computer programs understand natural human language (Jabloski, 2021). The processes develop the project such as preparing data, creating formatted data, defining models (seq2seq, encoder, decoder), training iterations and run evaluation. The implementor must learn and understand the concepts to explore the conversation model from the sequence-to-sequence chatbot and be able to modify coding to make a robot can respond in communication with the human. The report shows and describes modifying and developing the chatbot.

Dataset

According to Inkawhich (2021), he explained the dataset has a huge metadata collection of conversations from raw movie scripts. The Cornell Movie-Dialogs Corpus is a dataset that we used. It has many values of conversation but the chatbot cannot relate to being a better answer. So, we add the iteration to make the chatbot has more script dialogs to train the robot and increase ability intelligence that makes sense when conversing with people.

Development

Cornell Movie Dialogue Dataset has content from raw movie scripts. When the robot cannot understand if we ask a question beyond 4000 words. So, we upgraded the dataset to increase intelligence by training the chatbot to iterate over 6000 times.

Drive mount to access a dataset Cornell Movie-Dialog Corpus file from google drive.

Word Embedding is a table of the words that use the matrix to present the values of word to word. According to Brownlee (2017), he explained Word embedding are a type of word presentation that accepts words with similar meaning. This is a tool to manage and analyze the words. For example, if the words are closer in the vector space, that could be similar in meaning such as “Hi” and “Hello”.

Data preparation is a step to prepare the dataset to tune. There are steps to do such as spitting each line from the file and adding dictionary to fields, grouping the data from load-lines and adding conversations, Extracting pairs of sentences from conversations.

Loss graph has been plotted and shows the value that it has decreased average loss with every iteration. That means the chatbot has learned and is smarter than past.

BLEU (bilingual evaluation understudy) According to Brownlee (2017) he explained “It is a score for comparing a candidate translating of text to one or more reference translations” So, the score is a tool to use for evaluating the machine-translated text. The score gained from 0 to 1.

Conclusion

The project, Chatbot, has been developed by using Jupyter Python to respond to a human message. It is promising that the application has potential for digital marketing.

References

Jabloski Joanna (2021). Natural language Processing With Python's NLTK Package. Available at https://realpython.com/nltk-nlp-python. (Accessed 30th December 2021)

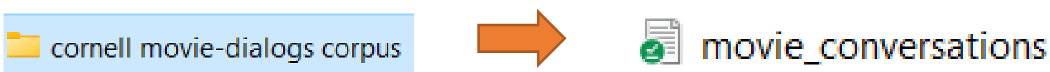
Inkawhich, Mathew (2011). Related corpus: Cornell Mobile-Dialogs Corpus. Available at https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html. (Accessed 30th December 2021)

Brownlee Jason (2017). What are word Embeddings for Text? Available at https://machinelearningmastery.com/what-are-word-embeddings (Accessed 30th December 2021)

Brownlee Jason (2017). A Gentle Introduction to Calculating the BLEU Score for text in Python. Available at https://machinelearningmastery.com/calculate-bleu-score-for-text-python (Accessed 30th December 2021)

Appendices

1. Download and Modify Cornell Movie-Dialog Corpus dataset and put many sentence scripts into movie_conversations.txt.



2. Setting Drive mount to access a dataset Cornell Movie-Dialog Corpus file from the google drive

```
[5]  from google.colab import drive
     drive.mount("/content/drive")
```

3. Modify the path to access the dataset and test printLines from "movie_lines.txt"

```
corpus_name = "cornell movie-dialogs corpus"
corpus = os.path.join("/content/drive/My Drive/Colab Notebooks", corpus_name)
```

```
printLines(os.path.join(corpus, "movie_lines.txt"))

b'L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!\n'
b'L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!\n'
b'L985 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I hope so.\n'
b'L984 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ She okay?\n'
b"L925 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Let's go.\n"
b'L924 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ Wow\n'
b"L872 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Okay -- you're gonna need to le
b'L871 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ No\n'
b'L870 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I\'m kidding.  You know how so
b'L869 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ Like my fear of wearing pastels
```

4. The example data will show on the "movie_lines.txt"

L and number is lineID || u and number is characterID || m and number is movieID || BIANCA or CAMERON is character || The last of sentence is text

```
b'L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!\n
b'L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!\n
```

5. The process,

firstly, the function loadLines used filename and fields to split between +++$+++ to get output lines.

```python
# Splits each line of the file into a dictionary of fields
def loadLines(fileName, fields):
    lines = {}
    with open(fileName, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")
            # Extract fields
            lineObj = {}
            for i, field in enumerate(fields):
                lineObj[field] = values[i]   # dictionaly of dictionaly
            lines[lineObj['lineID']] = lineObj
    return lines
```

Secondly, the function loadConversations used filename, lines (from function Loadlines) and fields to group fields of line get output conversations.

```python
# Groups fields of lines from `loadLines` into conversations based on *movie_convers
def loadConversations(fileName, lines, fields):
    conversations = []
    with open(fileName, 'r', encoding='iso-8859-1') as f:
        for line in f:
            values = line.split(" +++$+++ ")
            # Extract fields
            convObj = {}
            for i, field in enumerate(fields):
                convObj[field] = values[i]
            # Convert string to list (convObj["utteranceIDs"] == "['L598485', 'L5984
            utterance_id_pattern = re.compile('L[0-9]+')
            lineIds = utterance_id_pattern.findall(convObj["utteranceIDs"])
            # Reassemble lines
            convObj["lines"] = []
            for lineId in lineIds:
                convObj["lines"].append(lines[lineId])
            conversations.append(convObj)
    return conversations
```

Lastly, the function extractSentencePairs used conversations (from function loadConversations) to extract pairs of sentences from conversations to get output qa_pairs

```
def extractSentencePairs(conversations):
    qa_pairs = []
    for conversation in conversations:
        # Iterate over all the lines of the conversation
        for i in range(len(conversation["lines"]) - 1):  # We ignore the last line
            inputLine = conversation["lines"][i]["text"].strip()
            targetLine = conversation["lines"][i+1]["text"].strip()
            # Filter wrong samples (if one of the lists is empty)
            if inputLine and targetLine:
                qa_pairs.append([inputLine, targetLine])
    return qa_pairs
```

6. Word Embedding and Data preparation

Embedding is a type of word presentation that accepts words with similar meaning

The Encode RNN iterates through the input sentence one token at a time, at each time step outputting an “output” vector and a hidden state. The encoder change the context in the sequence into a set of point

```
class EncoderRNN(nn.Module):
    def __init__(self, hidden_size, embedding, n_layers=1, dropout=0):
        super(EncoderRNN, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.embedding = embedding

        # Initialize GRU; the input_size and hidden_size params are both set to 'hidden_size'
        #   because our input size is a word embedding with number of features == hidden_size
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers,
                          dropout=(0 if n_layers == 1 else dropout), bidirectional=True)
```

The Decoder RNN creates the response sentence in a token by token. It used the encoder context and internal hidden state in the next word in the sequence.

```
# Luong attention layer
class Attn(nn.Module):
    def __init__(self, method, hidden_size):
        super(Attn, self).__init__()
        self.method = method
        if self.method not in ['dot', 'general', 'concat']:
            raise ValueError(self.method, "is not an appropriate attention method.")
        self.hidden_size = hidden_size
        if self.method == 'general':
            self.attn = nn.Linear(self.hidden_size, hidden_size)
        elif self.method == 'concat':
            self.attn = nn.Linear(self.hidden_size * 2, hidden_size)
            self.v = nn.Parameter(torch.FloatTensor(hidden_size))
```

```python
class LuongAttnDecoderRNN(nn.Module):
    def __init__(self, attn_model, embedding, hidden_size, output_size, n_layers=1, dropout=0.1):
        super(LuongAttnDecoderRNN, self).__init__()

        # Keep for reference
        self.attn_model = attn_model
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.dropout = dropout

        # Define layers
        self.embedding = embedding
        self.embedding_dropout = nn.Dropout(dropout)
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers, dropout=(0 if n_layers == 1 else dropout))
        self.concat = nn.Linear(hidden_size * 2, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

        self.attn = Attn(attn_model, hidden_size)
```

7. Run model, we have to customize the values n_iteration to 6000 to increase the ability to respond to the chatbot

```python
# Configure training/optimization
clip = 50.0
teacher_forcing_ratio = 1.0
learning_rate = 0.0001
decoder_learning_ratio = 5.0
n_iteration = 6000
print_every = 1
save_every = 500
```

8. The pictures show to compare between the values of Iteration 4000 and 6000 that the average loss has a different. If the Average loss is decreased that means the robot has more intelligence.

```
Iteration: 4000; Percent complete: 100.0%; Average loss: 2.5253
Iteration: 6000; Percent complete: 100.0%; Average loss: 2.1285
```

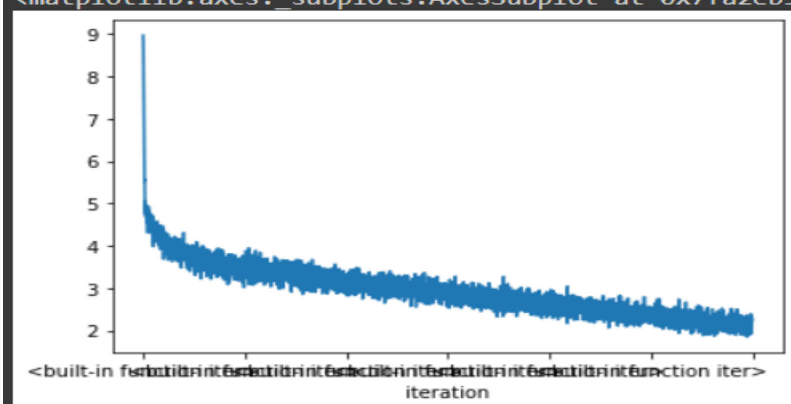
9. Test & Result Chatbot

```
> Hey
Bot: what ? ? ? ? ?
> How are you
Bot: fine . how i m doing
> What are you doing
Bot: i m sorry i m just dressed .
> Where are you going
Bot: i m going home . !
```

10. Loss Graph shows the value that it has decreased average loss with every iteration

```
loss_frame = pd.DataFrame(training_loss)
loss_frame.set_index('iteration', inplace = True)
loss_frame.loss.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa2eb997d90>
```



11. BLEU, the score is a tool to use for evaluating the machine-translated text

If the score is nearly 1.0 that means a nearly perfect match.

```
test_references = [] # true value
test_candidates = [] # predicted value
for i in test_pair:
  test_references.append(i[1].split())
  test_candidates.append(i[0].split())
```

```
calculate_bleu(references=test_references, candidates=test_candidates, weights=1)
```

```
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
0.029794149512459376
```