Assignment 7

CS 421: Natural Language Processing

1 Introduction

In this assignment, you will build a corpus-based Chatbot, implement a slot-filling algorithm, and a machine translator to translate German to English.

Chatbots are systems designed for extended conversations, set up to mimic the unstructured human conversations or 'chats', and used for entertainment, making task-oriented agents more natural. In this assignment, you will build a corpus-based chatbot with an information retrieval approach.

Goal-oriented dialog systems allow users to accomplish tasks, such as reserving a table at a restaurant. For dialog systems to fulfill such requests, they first need to extract the parameter or *slot* values of the request. **Slot filling** is the task that tags each word subsequence in an input utterance with the slot label. In this assignment, you will implement the slot-filling algorithm on Airline Travel Information Systems data.

Machine Translation is an important application of NLP where the machine translates from one speech to another using RNNs. Sequence-to-sequence models or encoder-decoder networks, are models capable of generating contextually appropriate, arbitrary-length, output sequences. Encoder-decoder networks have been applied to a very wide range of applications including machine translation, summarization, question-answering, and dialogue.

2 Instructions

Each question is labeled as `Code` or `Written` and the guidelines for each type are provided below.

Code

This section needs to be completed using Python 3.6+. You will also require following packages:

- `pandas`
- `numpy`
- `NLTK` or `SpaCy`
- `PyTorch or Keras or Tensorflow`

If you want to use an external package for any reason, you are required to get approval from the course staff prior to submission. Submit the code files on Gradescope under Assign-

ment 7 - Code.

3 Questions

Q1. Corpus-Based Chatbot: Code [10]

In this task, you will learn to implement the corpus-based chatbot using a retrieval approach.

1. Use the NPS chat dataset from NLTK and load the sentences using `nltk.corpus.nps_chat.posts()`.
2. From the sentences, discard all the sentences which are questions. You may use `re` package to find all the sentences starting with words like 'what', 'why', 'when', 'where', 'is', 'how', 'do', 'does', 'which', 'are', 'could', 'would', 'should', 'has', 'have', 'whom', 'whose', and 'don't'. Also, remove the sentences with lengths less than or equal to 4 unless the sentence contains words like 'hello', 'hi', 'greetings', 'what's up', and 'hey'.
3. Using the TF-IDF vectorization implemented in assignment 4, calculate the TF-IDF vectors for the sentences.
4. Write a method to calculate the cosine similarity of a vector with all the documents and return the document with the highest similarity score.
5. Build a chatbot by iteratively taking the user input, converting it to a TF-IDF vector, and finding and displaying the most similar document using the cosine similarity of the vector with all documents in the corpus.
6. **Written:** Explore the chatbot with your own input. Specify 10 inputs you provided and output generated from the chatbot and give your analysis based on the following criteria:
   - **Engagingness**: On a scale of 1-5, how much did you enjoy talking to the chatbot?
   - **Making sense**: On the following scale [1-4], how often did the chatbot say something which did NOT make sense?
     (a) Never made any sense (1)
     (b) Most responses didn't make sense (2)
     (c) Some responses didn't make sense (3)
     (d) Everything made perfect sense (4)
   - **Avoiding Repetition**: On the following scale [1-3], how repetitive was the chatbot?
     (a) Repeated themselves over and over (1)
     (b) Sometimes said the same thing twice (2)
     (c) Always said something new (3)
   - **Fluency**: On a scale of 1-5, how fluent were the responses of the chatbot (grammatical correctness, clarity, readability)?

Q2. Slot Filling Algorithm: Code [20]

In this question, you will learn to implement the slot-filling algorithm using LSTM. You can use Keras, PyTorch, or TensorFlow to build the model. You are provided with 3 files: `atis.train.csv`, `atis.test.csv`, `atis.val.csv` for this task.

1. Load `atis.train.csv` provided for training your model and `atis.test.csv` provided for testing the model in a DataFrame. Load `atis.val.csv` as validation data for your model. You need to use `tokens` and `slots` columns only.
2. Tokenize both columns and create a vocabulary of all words in `tokens` column and count the number of slots in `slots` column (including slot `'0'`).
3. Create a dictionary with the word as a key and a unique index for the word as a value (index starting with 1). Create another dictionary with the given index as a key and the word as the value. Create a dictionary with a slot label as the key and index as the value (index starting with 0).
4. Convert the words in all the sentences of the corpus to their respective index. Find the maximum sequence length of the corpus and pad each sentence with 0 such that the length of each input vector is the same as the maximum sequence length. Execute the same data preprocessing tasks for test and validation sets.
5. Build a model using layers: Embedding, LSTM, Dense layers, TimeDistributed layer, and softmax activation function.
6. Train the model with input of the index vector generated in step 4 and hyper-parameters of your choice. Tune the model on the validation set. To calculate the loss, use the index vector of the labels and the Softmax output of the model. You may use cross-entropy loss for this purpose. Report the F-score, precision, and recall on the test set.

Q3. Neural Machine Translation: Code [20]

In this question, you will learn to implement the Neural Machine Translation task using a sequence-to-sequence model with attention.

1. Load WMT14[1] data from Huggingface. Use `de-en` (German to English) data configuration. Note that this will need a lot of space (approximately 950 MB) for the data and a few minutes for generating the train-test split.
2. The training dataset contains a dictionary with languages 'de' and 'en' as keys and the respective sentences as values. Create a DataFrame with 2 columns 'German' and 'English' and the respective sentences. Build two separate vocabularies for German and English words and maintain a dictionary for German words with the word as key corresponding to the index and vice versa (as in Q2. step 3). Add <s> and </s> tags at the start and the end of each English sentence and add the tokens to the vocabulary as well.
3. With the dictionary created above, convert the German sentence to the array of indexes of each word in the sentence. e.g "`I learn NLP`" might be converted to `[10,7,29]` where in the dictionary, 'i' is mapped to index 10, 'learn' is mapped to 7 and 'nlp' is mapped to 29. Find the maximum length of the sentences in the German corpus. Pad each array with 0 such that the length of each index vector calculated above will be of the same

[1] https://huggingface.co/datasets/wmt/wmt14

length. For example, if the maximum length of a sequence in the dataset is 5, 'I learn NLP' will be converted to [10, 7, 29, 0, 0].

4. Convert each word in the English vocabulary to a one-hot encoded vector of length equal to the number of words in the vocabulary (including the start and the end tag). Perform steps 3 and step 4 on the test set and validation set as well.
5. Build an Encoder with an Embedding layer, LSTM layer(s), and an attention layer. For Decoder, use LSTM(s) and fully connected layers. Use Softmax activation for decoder output.
6. Train the model using Cross entropy loss, an optimizer of your choice (use of adam is recommended), and 10 epochs. You are free to vary the number of epochs to boost performance.
7. To generate the translation of the given German sentence from the trained model, convert the sentence to the index vector, feed it to the model and convert the output to the English word with the highest Softmax probability. Terminate the process once you encounter </s> token.
8. With the help of `nltk.translate.bleu_score.sentence_bleu()`[2], find the average BLEU score on the test set, which measures the similarity of the machine-translated text to a set of reference translations.

Tips: Unless you already have access to some powerful GPUs, I strongly recommend using Google Colab (https://colab.research.google.com/), and subscribe Colab Pro (https://colab.research.google.com/signup). It costs just $10 per month, and provides access to more powerful GPUs. You can feel free to choose from PyTorch, TensorFlow, and Keras. Perhaps Keras is easier to use.

For debugging, you can use a subset (e.g., 1% or even smaller) of the training and test set. Try a small batch size, smaller model, or a small number of epochs. pdb is highly recommended for debugging as it allows you to walk through your code line by line, and to identify where the code gets stuck. This will provide a lot of convenience for you to decide which simplifications mentioned above help the best.

For training and testing after debugging, you can also use a small subset of the training and test sets (e.g., 1%). You can terminate training any time that you think is "good enough" (at your own discretion). There won't be any penalty if the performance is (much) below the optimal. We do not require the training to be done to perfection in Assignments 6-8 - we will also read your code for grading.

[2] `https://www.nltk.org/howto/bleu.html`

4 Rubrics

This assignment will be graded according to the rubric below. Partial points may be awarded for rubric items at the discretion of the course staff.

Q1. Chatbot: Code (10 points possible)	
Q 1.1 - Q 1.4	+5
Q 1.5 - Q 1.6	+5
Q2. Slot Filling Algorithm: Code (20 points possible)	
Q 2.1 - Q 2.3	+5
Q 2.4 - Q 2.5	+5
Q 2.6	+10
Q3. Machine Translation: Code (20 points possible)	
Q 3.1 - Q 3.4	+5
Q 3.5	+5
Q 3.6 - Q 3.8	+10