CS5740: Assignment 4

Submission Due: May 13, 11:59pm

Report page limit: 3 pages

In this assignment, you will implement a sequence to sequence model to map instructions to action in a small environment. You will experiment with different network architectures, embedding approaches, and parameters, and pick the network that performs the best for your submission. Below, we break the assignment into multiple tasks to structure your work and guide you about the type of model ablations that you should report.

Your goal is to map instructions to actions in the ALCHEMY environment. Each example includes a sequence of instructions to be executed in order. Each instruction is mapped to a sequence of actions. The ALCHEMY environment includes seven beakers. The system can move colored liquid content from one beaker to another, mix content, or remove content from the set of beakers. There are six possible colors. We treat each beaker as a stack. This enables two types of actions: `pop` and `push`. The `pop` action takes a beaker ID as its argument, and removes the top color unit from the beaker. The `push` action takes two arguments: beaker ID and color ID. Figure 1 shows an example sequence of instructions from ALCHEMY. The Alchemy environment is adapted from the SCONE corpus.[1]

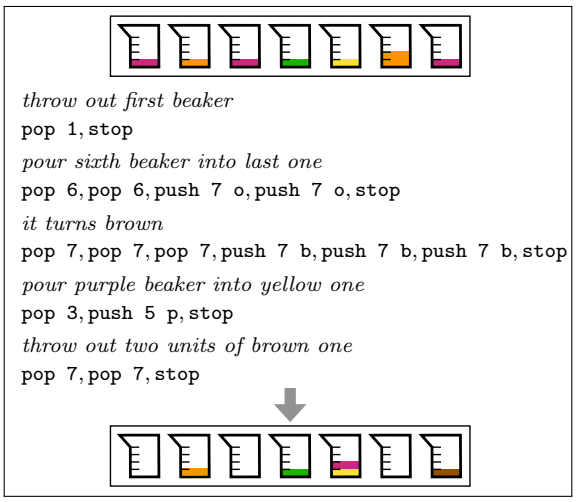

Figure 1: Example from the ALCHEMY domain, including a start state (top), sequence of instructions, and a goal state (bottom). Each instruction is annotated with the sequence of actions we define for ALCHEMY. This visualization is for illustration only. You are not provided code to create such visualizations.

Forming Groups Groups are specified on Canvas. You will need to copy the Canvas groups to Github Classroom when you create your repository. Upon submission, you will also need to copy the group to Gradescope as well. Please use the forum to form groups. If we identify cases of students that do not contribute equally to the group's work, we will be forced to penalize the assignment heavily.

GitHub Classroom Setup Follow the *starter repository invitation* link. If your partner has already created a team, join their team. Otherwise, enter a name for your team and continue. GitHub Classroom will provide a link to your team's private repository for you to clone to your computer. All team code should exist in the repository. *Only the code that is present in the master branch of your repository by the due date will be evaluated as part of your submission!*

Submission **Please see the submission guidelines at the end of this document.** In addition to your source code, your repository must contain the sequence of actions predicted for all test examples.

[1] https://nlp.stanford.edu/projects/scone/

Please submit this in the same format as the training data, and specify the file name in your report. This will allow us to run the actions and verify they achieve the appropriate final state.

Starter Repository:
https://classroom.github.com/g/3vS1g0b5 (GitHub Classroom invitation link)

Leaderboard:
https://github.com/cornell-cs5740-21sp/leaderboards/blob/master/assignment4/leaderboard.csv

Report Template:
https://www.overleaf.com/read/pjtdvdtxyxyz

Training, Development and Test Data Download the data listed at the top of the page. The data is split into training, development and a held-out test set. The test set does not include the labeled actions. Similar to previous assignments, you generated an output file for the test data to submit to evaluation.

The data is in JSON format. Each file contains a single line representing a list of examples. Each example is a dictionary, containing three keys: `identifier`, a unique identifier for the example; `initial_env`, the representation of the initial environment before the interaction takes place,and `utterances`, a list of utterances that take place in the example. In the training and development data, each `utterance` is a dictionary with keys `instruction` (the natural language instruction), `actions` (the gold list of actions corresponding to the instruction), and `after_env` (a representation of the environment after taking the actions). In the test data, the `actions` and `after_env` of each `utterance` are empty (this is what you need to predict).

Evaluation We evaluate the system performance using two metrics: single-sentence task completion and interaction task completion. The evaluation script executes the actions generated by your model to compare the final state to the annotated goal state. For single sentences, the goal state is the annotated final state of executing this instruction. For interaction performance, the goal state is the annotated state following the execution of all the instructions.

The `evaluate.py` starter script has a similar format to previous evaluation scripts. It will work with both evaluation metrics. See the `*_y.csv` files in the top-level directory for the format of single sentence-level (also called instruction-level) and interaction-level prediction files. **For test set predictions, we only consider the interaction-level performance, where we compare the final state at the end of the interaction with a gold world state**. See the placeholder file in `results/` for the format.

```
python evaluate.py
     --predicted [YOUR PREDICTION]
     --labels data/*_y.csv
```

Task 1: Sequence-to-Sequence Model Build a simple sequence-to-sequence model. The input sequence is the current instruction. The output sequence is the actions to execute. You will need to decide how to generate the actions as a sequence. This model is very simple. It does not take into account the world state or the history of the interaction. While it will not be able to handle many instructions, it should get quite a bit right.

Task 2: Adding State Information Incorporate state information into your decoder. You will need to find a way to encode the state of the world to a vector representation. At the least, you must include the state information of the initial world state. This is not the only possible way to incorporate state information, especially if you execute actions on-the-fly as they are generated. This will make your model much more powerful. It can now correctly understand what phrases like *the green beaker* refer to.

Task 3: Adding Attention Add an attention mechanism to your generation. An attention mechanism computes a weighted combination of multiple inputs. The weights are determined by the current state of the decoder. There are multiple elements you can attend on. Your model can now better handle its inputs by focusing on the appropriate parts.

Task 4: Adding Interaction History Add the history of the interaction to your model. At the least, you will have to encode previous utterances and incorporate them into the model. You get quite a few tokens from previous utterances. You probably want to use attention to focus on the most relevant ones for each decoding step. Your model can now reason reason about many contextual phenomena. For example, it should be able to resolve references like *it*.

Implementation Tips Develop with smaller dimensions for embeddings and layers to speed up things. In general, the small vocabulary and simplicity of the domain does not require large layers or embedding dimensionality. We observed strong results with a layer size of 100 using LSTMs and word embeddings of size 50. You must use batching for reasonable speed.

Expected Results The results for this model can range quite a bit, and are sensitive to initialization (you should randomly initialize your model). Generally, you should easily get over 80% instruction-level accuracy with the full model. A good model will get closer to 90% accuracy or higher. Interaction-level accuracy is much lower. It requires you to correctly execute all the instructions in the sequence. Above 50% accuracy starts to be interesting. At 60% the model starts to be interesting, and 70% is a very strong result.

Performance Grading The grading of your test performance on the leaderboard will be computed as $\min(24, (\frac{a}{0.6})^{1.05} \times 20)$, where a is the sequence-level test accuracy from the leaderboard.

Current CS5740 State of the Art The highest performance on this assignment is 60.58 (19sp).

Development Environment and Third-party Tools All allowed third-party tools are specified in the `requirements.txt` file in the assignment repository.[2] The goal of the assignment is to gain experience with specific methods, and therefore using third-party tools and frameworks beyond these specified is not allowed. You may only `import` packages that are specified in the `requirements.txt` file or that come with Python's Standard Library. The version of Python allowed for use is 3.8.x. Do not use older or newer version. We strongly recommend working within a fresh virtual environment for each assignment. For example, you can create a virtual environment using conda and install the required packages:

```
conda create -n cs5740a4 python=3.8
conda activate cs5740a4
pip install -r requirements.txt
```

Leaderboard We will consider the most recent leaderboard result, and it must match what you provide in your report. Please be careful with the results you submit, and validate them as much as possible on the development set before submitting. If your results go down, they go down. This aims to approximate testing in the wild (and in good research). We will refresh the leaderboard every 48 hours at 8pm, then eight and four hours before deadline. Each refresh will use what your repository contains at that point.

Submission, Grading, and Writeup Guidelines Your submission on Gradescope is a writeup in PDF format. **The writeup must follow the template provided. Please replace the TODOs with your content. Do not modify, add, or remove section, subsection, and paragraph headers. Do not modify the spacing and margins.** The writeup must include at the top of the first page: the names of the students, the NetIDs of both students, the team name, and the URL of the Github repository. The writeup page limit is **3 pages**. We will ignore any page beyond the page limit

[2] https://realpython.com/lessons/using-requirement-files/

in your PDF (do not add a cover page). We have access to your repository, and will look at it. Your repository must contain the code in a form that allows to run it form the command line (i.e., Jupyter notebooks are not accepted).

The following factors will be considered: your technical solution, your development and learning methodology, and the quality of your code. If this assignment includes a leaderboard, we will also consider your performance on the leaderboard. Our main focus in grading is the quality of your empirical work and implementation. Not fractional difference on the leaderboard. We value solid empirical work, well written reports, and well documented implementations. Of course, we do consider your performance as well. The assignment details how a portion of your grade is calculated based on your empirical performance.

In your write-up, be sure to describe your approach and choices you made. Back all your analysis and claims with empirical development results, and use the test results only for the final evaluation numbers. It is sometimes useful to mention code choices or even snippets in write-ups — feel free to do so if appropriate, but this is not necessary.

Some general guidelines to consider when writing your report and submission:

- Your code must be in a runnable form. We must be able to run your code from vanilla Python command line interpreter. You may assume the allowed libraries are installed. Make sure to document your code properly.
- Your submitted code must include a `README.md` file with execution instructions.
- Please use tables and plots to report results. If you embed plots, make sure they are high resolution so we can zoom in and see the details. However, they must be readable to the naked eye (i.e., without zooming in). Specify exactly what the numbers and axes mean (e.g., F1, precisions, etc).
- It should be made clear what data is used for each result computed.
- Please support all your claims with empirical results.
- All the analysis must be performed on the development data. It is OK to use tuning data. Only the final results of your best models should be reported on the test data.
- All features and key decisions must be ablated and analyzed.
- All analysis must be accompanied with examples and error analysis.
- Major parameters (e.g., embedding size, amount of data) analysis must include sensitivity analysis. Plots are a great way to present sensitivity analysis for numerical hyper parameters, but tables sometimes work better. Think of the best way to present your data.
- If you are asked to experiment with multiple models and multiple tasks, you must experiment and report on all combinations. It should be clear what results come from which model and task.
- Clearly specify what are the conclusions from your experiments. This includes what can be learned about the tasks, models, data, and algorithms.
- Make figures clear in isolation using informative captions.

Updated on April 27, 2021