Blocksworld Architecture

Remy Bubulka, Coleman Gibson, Kieran Groble, and Lewis Kelley

April 23, 2018

The Goal

The 2D Blocksworld web game has been produced in order to more easily gather data to bootstrap the AI tied with the ongoing Human-Robot Collaboration project. The eventual goal is to have a 3D version, but in the meantime the AI should be proven in a simpler, 2D case from which it can learn and have its structure tweaked to best solve the problem.

AI Architecture

Since a good portion of what is interpreted by the AI is natural language, a series of neural networks are used. The training takes place in the neural network suite of programs (see below for more details).

In order to simplify what the AI has to interpret, multiple small AI's are run off of the input, each of which "annotates" the input with a different piece of information. For example, given an input of "Move the Green A there," one neural network flags the text as wanting a "green" block, and another will say the user wants to "move" a block rather than flip one.

Software Architecture

There are three major functionalities in the Blocksworld system: the neural network construction and training, the Python web server, and the web page that is shown to the end user. See the attached architecture diagram here for a general overview.

Neural Network Construction and Training

source

Everything in the neural network repository is related to training or running the various different neural networks that together form the Blocksworld AI. The training happens before the server is ever started, but this repository also includes python files used to run the neural network models it trains.

It's important to note that the Neural Network repository is a suite of programs, not a library. Some are standalone programs, some generate data, and some train the models.

Firstly is `generate_text_instructions.py`, which generates the training data for the trainer to use. It outputs the inputs into one file, then splits up the expected outputs into multiple files for each neural network to use.

The most important file is `trainer_core.py`, which does the majority of the work when it comes to training. It's called by each of the individual trainers to train on their model with their specific expected outputs. This outputs `.h5` files that store the models, and can be re-run on stored models to improve an existing model.

Finally is `model_runner.py`, which is used by the server as an interface to actually load and run the models to get some output from them. There is a user-friendly frontend to this, called `model_repl.py`, which is useful for quick testing of a newly trained model.

Python Server

source

Made up of one program, the server opens a connection to the database (set up in `create-baseline.sql`), uses an interface to the neural network (`neural_network_interface.py`), and communicates with the client using the `socket.io` library (`emits.py`).

If the server cannot reach the database, it will instantly shut down.

Also of note is that the neural network repository is cloned in as a submodule. See this repository's README for more details on properly installing it.

Web Page

source

There are only a few pages of importance, with the most important being `game.html`. The included JavaScript files are roughly organized by their functionalities, with things like `movesTracker` tracking what moves were made to later be exported to the server, or `blocks` being used to hold and modify information about the onscreen blocks.

Of special note is that like the server, this too uses the `socket.io` library for communication. Sequence diagrams detailing the interactions between the server and client can be found here.

Testing / Continuous Integration

Due to the different repositories and languages used, different programs have different testing setups.

The server simply uses `pytest`, with its tests located in the `tests` folder.

The web page uses a combination of the Jasmine framework and Angular's Protractor for testing. Jasmine is used for the unit tests, which are stored in the `spec` folder.

Protractor is used for end to end testing, and its syntax is identical to Jasmine's. It's configuration is set up in `conf.js`, and currently the only test file is `e2eTesting.js` (we don't plan on having many).

We use Travis CI for our continuous integration environment of Blocksworld. Details are in the `.travis.yml` file. Note that the CI has to start up a Postgres database, start a Selenium service, test the server, test the web page, start the server, and finally run the end to end testing.

There is additionally some buggy behavior with starting up the Selenium driver used by Firefox. More details are given in a comment in the CI file.

TODO Deployment

Current plans for deployment are to write a script for the machine serving Blocksworld that automatically pulls the most recent version of the master branch and restart the server with the new code. This could either be run manually or as a cron job.