Names: Jordan Le & Bar Movshovich
Date: 12/8/19
Emails: jor25@pdx.edu, movshov@pdx.edu

Hw 3 - Gothello AI Write-up:

Approaches:

- The first step was downloading the zip file of half a million shallow evaluated games and preprocessing the outputs. When preprocessing the data, we identified who won the game from the last line of the gthd files. This usually contains a "Black wins" or "White wins" line. This is how we determined who's data to collect. We specifically collected data only from the winners of the game. Then we flattened the output board from the winner's log. Did some string manipulation and then put it into a numpy array in which the winner's pieces are replaced with a number 2, the loser's pieces are replaced with a 1, and blank pieces are replaced with a 0. This is to ensure we are learning from the 2 element. This gave me a decently sized dataset containing 25 features for every state in the winner's games.
- Then to label each of these data values, we went to the "-output" files of the winners and collected the tile that they were placed in and converted those into an index value from 0-24. Those label values were then one-hot encoded. Each of the data values is labeled with the next move that the winner made in a game. Note, this is all the preprocessing.

- The next step was creating a machine learning model for this data. From here, we one-hot encode each feature with winner and loser presence. [1,0] means the winner and [0,1] means the loser.
- The data from here was flattened again to be a numpy array of 50. This was then passed into my model architecture:
    - 50 input nodes
    - 75 hidden nodes
    - 75 hidden nodes
    - 75 hidden nodes
    - 25 output nodes
- We used categorical cross-entropy for loss, metric set for accuracy, a learning rate of 0.001 with the Adam optimizer, and an activation function of RELU (Rectified Linear) on all the dense layers. Note, we used a softmax activation for the final output layer.
- Next, we conducted training with the one-hot data and one hot label values and allowed the model to train for about 50 epochs. The training process was done using google colab and at the end of the 50 epochs, we received a top 1 accuracy of ~21%, top 3 of ~43% and top 5 of ~64%.

- From there, we tested the performance of the model by having it play 100 games against the barton.cs.pdx.edu server and collected the number of wins, losses, and draws between the players.

How to run:

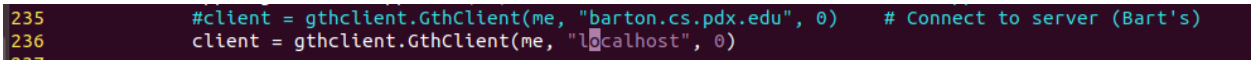
- Set up the environment:
  - virtualenv -p python3 env
  - source env/bin/activate
  - pip3 install -r requirements.txt

- Running with local host:
  - TERMINAL 1: Set up the local host server with bash script in one terminal:
    - cd gothello-gthd/
    - sh run_local_server.sh

  - TERMINAL 2: Set up grossthello white player in another terminal:
    - cd gothello-grossthello/
    - sh run_grossthello.sh

  - TERMINAL 3: Run our AI in third terminal as black
    - cd gothello-libclient-python3/
    - python3 neuro_gth.py black

- Running with the barton.cs.pdx.edu server:
  - Uncomment out the client variable in neuro_gth.py with the barton server. Comment out the client variable with localhost.



  - TERMINAL 1: Run against bart server in third terminal as black
    - cd gothello-libclient-python3/
    - python3 neuro_gth.py black

Experimentation:

- We ran the player against the barton.cs.pdx.edu player for 100 games in 3 experiments to see how it did. We then received the following outputs from our scoring function listed below in the results section.
- Note that, in order to reproduce these results, the player must be set to black and the capture function must be commented out. This is using our own scoring function under the assumption that state collection is accurate.

Results:

- From our experimentation, we received an average win rate of ~74.66%, an average loss rate of ~14.66%, and an average draw rate of ~10.66%.

```
DRAW
Game #100       wins: 76              losses: 13      draws: 11       win perc: 0.76
```

```
WHITE WINS
Game #100       wins: 74              losses: 12      draws: 14       win perc: 0.74
```

```
BLACK WINS
Game #100       wins: 74              losses: 19      draws: 7        win perc: 0.74
```

Hardware and software set up:

- For software development and experimentation set up, we continue to use a 64-bit Ubuntu 18.04 virtual machine with a 2048 MB base memory. We worked on a virtual machine to continue experimenting freely without directly altering our own devices. For this project, we also created a standard python 3 virtual environment.

Issues:

- The board is upside-down in the random player.
- The captures are not implemented correctly.
- Using the random player's show board function turned out to have many unknown bugs. First of all, the board was upside down compared to the output from the official java version. The capture features were not included so proper state collection was variable. Also, a scoring function was not included, so we made one under the assumption that the board placements were correct. Unfortunately, we found this out rather late in the project and had to go back to make modifications to the board and state.
- Note, the server provided to use also had some connectivity issues, especially in the last few days leading to the project deadline. This was likely due to the number of students trying to access the server all at once. To remedy this, we created several shell scripts as shown above that we used to run the game locally.

Future Attempt:

- If we were to attempt this in the future, the first thing we would likely do would be to fix the random player's show board function. Without proper state collection any attempts we would make again would be in vain because they would be based on false data.
- The next thing we would attempt would be to increase the win rate of the Neural Net to be greater than 70% with accurate state collection. To achieve this, we would provide the Neural Net with accurate state collection and some heuristic AI. Most likely the AI heuristic would be an adversarial search meaning that we try to increase our odds by decreasing the other players options.

- We would then test each AI individually for accuracy, then combine them into an ensemble AI. This means weighted average voting based on the individual AI's win rates.
- The next step would then be testing this new combined method to see if using a Neural Net with a heuristic AI gives us a better overall win rate.

What We Learned:

- How to preprocess data into neural network input layers.
- How the game of GO works and its associated rules.
- How the game of Othello works and its associated rules.
- How to test/validate a Neural Net's outputs.

Resources:

- Initial model architecture example: https://github.com/jor25/Dino_Game/blob/master/collect_states.py

- Stratify example: https://stackoverflow.com/questions/29438265/stratified-train-test-split-in-scikit-learn

- Sklearn train_test_split: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

- Bart's Gothello Project: https://github.com/pdx-cs-ai/gothello-project

- Remove newlines from string: https://stackoverflow.com/questions/16566268/remove-all-line-breaks-from-a-long-string-of-text

- Split string into list: https://www.w3schools.com/python/python_regex.asp

- Replace specific instances: https://stackoverflow.com/questions/19666626/replace-all-elements-of-python-numpy-array-that-are-greater-than-some-value

- One hot my data: https://stackoverflow.com/questions/29831489/convert-array-of-indices-to-1-hot-encoded-numpy-array