University of San Francisco

Physics 302/CS 686-06 – Scientific Computation and Machine Learning
Spring, 2023

Final Project

Artificial Neural Network Alak Player

Part I: Due Thursday, May 11, at 11 PM
Part II: Due at the start of the tournament on Wednesday, May 17

The purpose of this project is to build an artificial neural network (ANN) algorithm to play the one-dimensional version of the ancient board game Go, or Alak.

We have gone over how to start this project in class. I would like to emphasize the following:

A. Rules and Simulations

1. There are two sides: the "x" side and the "o" side.
2. Starting board configuration: there are 5 pieces lined up on either side and 14 spaces total, i.e., 4 empty spaces between the two groups.
3. You can move one piece at a time.
4. You cannot add any new pieces to the board.
5. The "x" side always makes the first move.
6. Kill: if and only if one or more contiguous pieces on one side is "sandwiched" between two opposing pieces, then the "sandwiched" piece(s) will be removed from the board.
7. Randomly generate the next legal move.
8. Decide if any pieces need to be removed from the board.
9. The opposite side makes its move.
10. Show the positions of the pieces in each round of simulation (0 to 13), by using 0-9 and lower case 'abcd'. Please do NOT use 11 - 13 on the display. Here's the opening round of a simulated game:

```
Starting Game:
Your side is 'o'

round: 0

Board: xxxxx____ooooo
       0123456789abcd
x : 0 ==> 5
Board: _xxxxx___ooooo
       0123456789abcd
```

```
gain: 0

o : 9 ==> 0
Board: oxxxxx___o_ooo
       0123456789abcd
gain: 0
```

11. When a move results in a kill, in the `verbose` mode, you should show the board immediately after the move and again show the board after the killed pieces are removed, as shown in this example:

```
o : 7 ==> 4
Board: _xxxoxo_oo
       0123456789
Board: _xxxo_o_oo
       0123456789
gain: 1
```

12. Repeat until one side wins (the other side should have zero or one piece on the board)
13. Save all the rounds (each round consists of two moves, one by each side) of the simulated game in a pickle file if you use sklearn, or a file type of your choosing if you use TensorFlow (usually .keras or .h5).
14. About suicide moves: in your code, you are allowed check if the highest probability move the neural net suggests is a suicide move. If so, you can check the next highest probability move; repeat if necessary until a non-suicide move is selected. If you don't implement this check, the suicide move will stand, and you will have wasted a move and lost a piece.

Note that while "ko" is something that can happen in the real Go game, where players are adding new pieces to the board, it is not a concern for this version of the game.

B. Training

1. First divide your simulated dataset — the Training Sample, into training and validation sets.

2. You should compute winning rate for the side you have chosen.

3. Feel free to come up with your own training scheme (or you can talk to me). **No lengthy calculations or "brute-force" allowed: a better player has to be achieved through the ANN "learning" process.**

C. Suggestions

Remember below are sonly suggestions. You are free to design your own algorithm.

1. Generate lots of games and then throw away the obviously "dumb" ones (according to (i) and (ii) below) and train using the rest.

   (i) Throw away long games — they likely have lots of purposeless moves.
   (ii) Avoid games that have suicide moves — these are not games to learn from. In fact, once a game has a suicide move, stop playing and discard it.

2. I suggest you start with the training by choosing the "o" side first.
3. If you want to optimize the number of pieces killed by your move, you may consider using a softmax to label the different number of kills.

D. "Unit Tests":

You have to have a method in your class definition to test capture. Your code need to pass these eight tests for checking capture (or kill/removal):

```
off_side = ['x', 'x', 'x', 'x', 'o', 'o', 'o', 'o']

board_list = ['xoxoxx________', 'xooxooxx______',
'__xoo__oxo____', '__xoooox_o____',
'_o__ooxxxo__o_', '___xoxxxoo____', '___x_oxxoo_____',
'____xoxx___x___']
```

Here:

`off_side` — designates the offensive side, i.e., the side that has just made the move. Thus the `off_side` is `'x'` for the first four moves and `'o'` for the next four moves.

`board_list` — each entry shows the board after the move by the `off_side`.

For every move, a test passes if the resulting broad (as determined by you code) is the same as the corresponding entry in this list:

```
board_expect = ['x_x_xx________', 'x__x__xx______',
'__xoo__o_o____', '__x____x_o____'
'_o__oo___o__o_', '___xo___oo____', '___x_o__oo_____',
'____x_xx___x___']
```

These tests include:

- simple kill involving different numbers of pieces being removed
- double kill
- double kill that involves more than one piece

- a suicide move

Please keep in mind that you code should check *all* varieties of kills. You should NOT design your code in such a way that it *only* passes these tests. I cannot provide you with all the different scenarios of kills.

E. "Game Plan" for the Next 1.5 Weeks:

1. Thursday, May 4: Be able to (i) play random games; and (ii) play the game interactively.
2. Tuesday, May 9: (i) Decide on a training scheme and construct the training sample; (ii) Train your neural net model and deploy it against a random "player".
3. Thursday, May 11: Optimize your model.
4. Thursday and Friday, May 11 - 12, Pedro will check in with each of you to make sure your code is making all the legal moves and all the correct removals (kills). Each of you will also be paired up with a classmate to do a "drill game" with Pedro present.

Part I: Training

Train a neural network that can beat a random "player" with a winning percentage of at least 60% over 100 games.

Note that in the construction of the training sample, you can put in as much human intelligence as you'd like — brutal force calculation is allowed. For example: you can find the locations that will result in kills, avoid moves that will "bunch together" your own pieces as it can lead to a large kill by the other side, etc. This would be similar to the AlphaGo training sample (curated from human games)

You may use random games. The would be similar to the AlphaGoZero training sample. In this case, you may also consider training the game twice: first generate the games randomly and train; and then use your trained classifier to play lots of games and then use these to train again.

You should submit your code and the pickle file of the trained model.

Due: Thursday, May 11, at 11 PM.

Part II: Deployment & the Tournament

Your code need to be able to play interactively, and it needs to be able to play both sides. You have to enter your opponent's move interactively and then your ANN-trained player will respond.

Keep in mind that if your neural net makes an illegal move, you lose!

To decide which move to make, you can feed all possible legal moves into your neural net classifier to make predictions. From the returned array of predictions, choose the move with the highest winning probability. (Avoid using For loop for this purpose.)

If a game becomes too long, and therefore possibly a draw (likely due to both sides playing a very defensive game), a "sudden death" game will played. Whichever side plays first in the first game will play second in the sudden death game. In this game, after each round, whichever side makes more kills wins the game. It is not required, but you may consider training two neural net models for the "normal game" and "sudden death"

Right after the tournament, each student will present one slide describing their algorithm, including platform (e.g., sklearn, TensorFlow), neural network architecture, code design, and the construction of your training sample.

Code freeze: before the beginning of the tournament.

The tournament will be on **Wednesday, May 17, 2023, 3:00 - 5:30 PM.**