CONTENTS

- **Updated April 6**
  - **Official release of public test cases - fixes a problem with sorting the moves the version posted to the forum**
  - **Fixed the same problem and another output issue in the example below**
- **Updated April 3**
  - **added clarification about rounding**
  - **a fix to the description of a command line option**

Cmput 496 Assignment 4

Due Apr 9, **11:55pm** on eClass. Submit via eClass submission link on the main eClass course page.
Late submission (with 20% deduction) Apr 11, 11:55pm. Submit via separate eClass link for late submissions.

In this assignment, you use the results of machine learning to initialize the nodes in MCTS with prior knowledge. You add your improvements to our `Go5` player.

In class, we have discussed using knowledge in simulations, and machine learning with simple features. The learning method was Remi Coulom's minorisation-maximization method (MM for short). The features with their learned weights were then used in `Go4` in its probabilistic simulation policy. However, this simulation policy was quite slow.

`Go5` implements a Monte Carlo Tree Search (MCTS), using the basic UCT formula to select children in the tree. Beyond the tree, it can use any of the simulation policies we discussed to play out the simulated games.

Setup

1. First, as in previous assignments, make sure you have your Python 3 and GoGui set up.
2. Now download Go4+5.tgz. This contains directories `Go4`, `Go5` and `util`.
3. **Added Apr 6:** A public GTP test file assignment4-public-tests.gtp .

   As discussed in class, `Go4` is a simple simulation-based player which uses a probabilistic simulation policy, while `Go5`

implements MCTS with UCT.

4. **Modify/add code in subdirectory Go5 only to implement your solution.**
5. **Start your work early and leave enough time for testing.**
6. Study the Go4 and Go5 documentation.
7. For more information about the concepts related to this assignment, see the following resources:
   - **Lecture 19 - Assignment 4 preview; using Knowledge in UCT**
   - Lecture 14 - Probabilistic simulation policies
   - Lecture 15 and 16 - MCTS
   - Lecture 17 and 18 - Introduction to Machine Learning for Heuristic Search, Learning with Simple Features, Move prediction in Go

Goals of the Assignment

In Lecture 19 we discussed three ways to improve the in-tree move selection of MCTS by prior knowledge:

1. By using knowledge as node initialization
2. As an additive term
3. As a multiplicative term

KNOWLEDGE-BASED INITIALIZATION

The main goal of this assignment is to implement the first of the three ways above in our MCTS program `Go5`: knowledge-based node initialization. In order to do this, you apply the features and their weights learned by MM which are included with the `Go4` codebase. As discussed in class, the knowledge consists of weights for simple move features and 3x3 patterns.

Initialize each new node in `Go5` by prior knowledge. (You can skip initializing the root node itself since it is irrelevant here.) New nodes are created in the `expand` method of `class TreeNode` in `mcts.py` . This is a good place to call your new code for initialization.

First, compute the probability of each move according to the learned model. You need to get the probability of each legal move, including the pass move. As an example of how to compute similar probabilities from features, you can check the function `generate_moves_with_feature_based_probs` in `Go4/board_util_go4.py` . Next, use these probabilities for initializing wins and simulation count of each new node as follows:

Each `TreeNode` stores these statistics in two fields, `_black_wins` and `_n_visits`. Both of these values are initialized to 0 in the `TreeNode __init__` method. Your initialization should overwrite those 0 values with values computed from the prior knowledge, as detailed below.

Also, for the root only, initialize its `_black_wins` and `_n_visits` with the sum of its children's `_black_wins` and `_n_visits`. This will avoid problems with computing `log` n_p in the UCT formula if n_p=0.

HOW TO CONVERT A PROBABILITY INTO SIMULATION COUNT AND WINS

Assume there are n moves, with probabilities $\{p_1,...p_n\}$, and that p_{max} is the largest probability - the probability of the best move according to prior knowledge. This best move will get initialized with `PRIOR_KNOWLEDGE_STRENGTH = 10` simulations. The number of simulations for all other moves i should be computed by:

First compute floating point number `sim(i)` by scaling down `PRIOR_KNOWLEDGE_STRENGTH` by the ratio p_i / p_{max}. Use this floating point number in the wins(i) calculation below.
For the output, round `sim(i)` to the nearest integer.

The `winrate(i)` of each move i should also be scaled according to a similar linear scaling formula. The highest winrate, for the best-evaluated move, should be 1, and the winrate for a probability of 0 should be 0.5 .

Finally, the number `wins(i)` for a move i should be computed as follows:
Compute the product `winrate(i) * sim(i)`, then round the result to the nearest integer. Use `int(round(x))` to convert a floating point number x to the nearest integer.

Example: Given three moves with probabilities
{$p_1 = 0.29$, $p_2 = 0.7$, $p_3 = 0.01$}
Compute the following:

```
Sim: {10*.29/.7=4.1428571429, 10*.7/.7=10, 10*.01/.7=0.1428571429}
Winrates: {0.7071428571428571, 1, 0.5071428571428571}
Number of simulations: {4, 10, 0}
Number of wins: {3, 10, 0}
```

Then initialize move 1 with 3 wins out of 4 simulations, move 2 with 10 wins out of 10 simulations, and move 3 with 0 wins out of 0 simulations.

Clarification about pass moves

Two pass features are included in the features. Here is how to handle pass moves in this assignment:

1. In the MCTS tree, Pass moves should always be allowed. The `Go5` code already implements that, and you need to compute the knowledge initialization of pass moves in the same way as for any other move.
2. In the probabilistic simulation policy in Go4, a pass move is only generated if there is no other move. This behavior is different from the node initialization in this assignment, and you should not use it as a model for developing your code.

PLAYER

Modify the `Go5.py` player to add the ability to use knowledge-based initialization. Beyond that, it should use MCTS as already implemented.

Go5 has an option `--in_tree_knowledge`, which currently does nothing. In your solution, if this command line option is used, then your knowledge initialization should be called. By default, this option is off, so your program should not call the knowledge initialization. Test your player in the following configuration:
`python3 Go5.py --movefilter --in_tree_knowledge="probabilistic" --num_total_sim=1`

With these options, Go5 will use default almost-random simulations, the move filter, and only run a single simulation. This is enough for us to test the node initialization of the root's children. However, your code should initialize all new nodes in expand, and you can use larger searches to verify that this works.

Also test your player without the option `--in_tree_knowledge`,
`python3 Go5.py --movefilter --num_total_sim=1`
and make sure that your knowledge initialization is not called in this case.

GTP Commands

Modify the existing GTP command `genmove` and add a new command `prior_knowledge`:

1. `genmove color`

   Your `genmove` command in Go5.py should generate a move using MCTS as in the original Go5, but your player should use your new initialization of nodes.

2. `prior_knowledge`

   Print the prior knowledge initialization for moves from the current position. These are the same values that would be used in your search to initialize the children of the root, i.e. _black_wins and _n_visits.

   Sort by the (true, not rounded) winrate in descending order (best move first), but print only the (integer) wins and number of simulations for each move, as in the example below, using the format `move wins simulations`, as in the example below. Break ties in alphanumeric order of the move.

   Example:**Fixed Apr 6**

```
boardsize 3
play b b2
play w a2
play b b3
play w a3
prior_knowledge
= a1 10 10 b1 2 3 Pass 1 1 c3 1 1 c2 0 1 c1 0 0
```

Testing Procedure

Public test cases **added Apr 6** assignment4-public-tests.gtp .

Test cases will test the `genmove` command for the MCTS player, and the `prior_knowledge` command. Many of the tests currently fail, since the program's MCTS engine uses standard UCT with no initialization, and the `prior_knowledge` command is not implemented. Note that our given examples cover only a small subset of possibilities. For evaluation, we will use a more comprehensive set of tests.

Other GTP Commands

Many other GTP commands are already implemented in the Go6 sample code. Do not break their functionality, since we will test your program using those commands, such as:

`boardsize, clear_board, name, version, play,...`

These commands are used by the test script. As in previous assignments, for convenience we will use multiple calls to `boardsize` and `clear_board` within the same gtp file, so make sure this use case still works in your solution.

Do Your Own Pre-submission Test

Follow the steps below on a standard undergraduate machine, and submit a text file `presubmission.log` that shows a copy of your command line presubmission testing.

1. Create `assignment4.tgz` with your solution as described under Submission Checklist below.
2. Use the Linux script command to log your testing session.
3. Copy `assignment4.tgz` into a new directory
4. unpack it
5. run gogui-regress with your Go5 program on file `assignment4-public-tests.gtp` (**not yet released**) as input.
6. Stop `script` logging here with the command `exit`.
7. Add file `presubmission.log` to your submission, and compress it again
8. Use `tar -tf` for a final check of your `assignment4.tgz` contents. Use the checklist below.
9. Fix any problems, then submit on eClass when you're ready

Submission Checklist

Submit a single tgz file called `assignment4.tgz` which contains exactly the following (and nothing else):

- A single directory `Go5` which contains all the files in your solution, namely:
  1. All the files from the original Go5 directory, but with python code modified/added to solve the assignment. You may add more .py files but do not remove or rename any existing ones.
  2. The file `presubmission.log` from your presubmission testing.
  3. A file `readme.txt` which lists the names and student IDs of your team.

Marking

There will be a total of 5 marks for this assignment.

- 2 marks for your modified Go5 player passing our public test cases.
- 1 mark for doing your own presubmission check of both programs and supplying the evidence in `presubmission.log`.
- 2 marks for passing our private test cases.
- For both public and private test cases, there will be roughly equal weight for testing `genmove` and `prior_knowledge` commands.

Code that does not run will receive a mark of 0.

EXTRA COMMENTS

- No special error handling code is required in this assignment. We will only test with valid moves and valid GTP commands which have valid arguments.
- We will only test with alternating play, so any `play` commands will be for the color toPlay. Clearing the board resets toPlay to Black.
- By default, we assume that teams will stay the same as in assignment 3. If you change your team, inform the TA before the assignment 4 deadline.

Created: Mar 19, 2018 Last modified: Apr 3, 2018

Martin Müller
