About project3-2.zip

EE488C Special Topics in EE <Deep Learning & AlphaGo>, 2016 Fall
Information Theory & Machine Learning Lab., School of EE, KAIST

- Project3-2.zip contains more files for mini AlphaGo for project #3. The files are:
  - go_train_SL_policy.py: for training SL policy networks using value networks
  - go_train_RL_policy.py: for training RL policy networks
  - SL_policy_black.ckpt, SL_policy_white.ckpt: ckpt files of trained SL policy networks (same as player #8 in ‘players’ sub-directory)
  - RL_policy_black.ckpt, RL_policy_white.ckpt: ckpt files of trained RL policy networks (same as player #9 in ‘players’ sub-directory)
  - ‘players’ sub-directory contains trained value and policy networks for mini AlphaGo (players #1 ~ #9)
  - test_players.py: plays games among 9 players in the ‘players’ sub-directory and prints out the result
  - policy_interactive.py: plays interactive games between a human and policy networks
  - game.py & strategy.py: updated versions of game.py & strategy.py so that the computer takes greedy actions when you run policy_interactive.py. You can use the previous version of game.py & strategy.py, but then policy_interactive.py will take stochastic actions according to the outputs of the policy network.

- Files in ‘players’ sub-directory are as follows and they are provided as baseline value and policy networks for you to experiment with. In case you choose not to train your own value or policy networks, you can use some of the following in your implementation of mini AlphaGo.
  - Player 1: first-generation value networks, generated by go_train_value.py with n_train=n_test=1,000 (black1.ckpt and white1.ckpt)
  - Player 2: second-generation value networks, generated by go_train_value.py with n_train=n_test=2,500
  - Player 3: first-generation value networks, generated by go_train_value.py with n_train=n_test=10,000
  - Player 4: second-generation value networks, generated by go_train_value.py with n_train=n_test=25,000
  - Player 5: first-generation value networks, generated by go_train_value.py with n_train=n_test=100,000
  - Player 6: second-generation value networks, generated by go_train_value.py with n_train=n_test=250,000
  - Player 7: third-generation value networks, generated by go_train_value.py with n_train=n_test=250,000
  - Player 8: SL policy networks, generated by go_train_SL_policy.py with n_train=n_test=250,000 from player 7
  - Player 9: RL policy networks generated by go_train_RL_policy.py using player #8 as initial policy networks, where opponents are taken from the opponent pool 50% of the time and the value network is taken as the opponent 50% of the time. A total of 300 iterations were performed and the RL policy networks whose performances are the best against the SL policy networks are chosen and saved as player #9.

- Game results among 9 players in ‘players’ sub-directory. In the following table, player #6 with black won against player #3 with white and player #6 with black lost against player #7 playing white. You can get the following table by running ‘test_players.py’.

b \ w	1	2	3	4	5	6	7	8	9
1	b	b	w	w	w	tie	w	w	w
2	b	b	b	w	b	w	w	w	w
3	b	b	tie	b	b	w	w	b	b
4	b	b	b	b	b	tie	tie	b	tie
5	b	b	b	w	tie	b	w	b	b
6	b	b	b	b	b	b	w	b	b
7	b	b	b	b	b	b	b	b	b
8	b	b	b	b	b	b	tie	b	b
9	b	b	b	b	b	b	b	b	b

  - The following table shows the total score (+1 for win, -1 for loss, 0 for tie) for each player. As you can see more training helps and player #7 performs the best among all value networks (players #1 ~ #7). SL policy networks (player #8) do not perform well because they are trained using game plays by value networks with randomness, but RL policy networks (player #9) are better than SL policy networks due to reinforcement training. However, RL policy networks are still worse than player #7. You may want to use MCTS that utilizes both value and policy networks to create a strong player.

Player	1	2	3	4	5	6	7	8	9
Score	-13	-10	-2	3	-2	4	12	3	5