CPSC 481 – Fall 2019

Artificial Intelligence

Project Two: Go BOT model through ch. 7

due at beginning of class – 16 Oct (W)

Part Two of our Go bot engine involves building much of the infrastructure of the game. In particular,

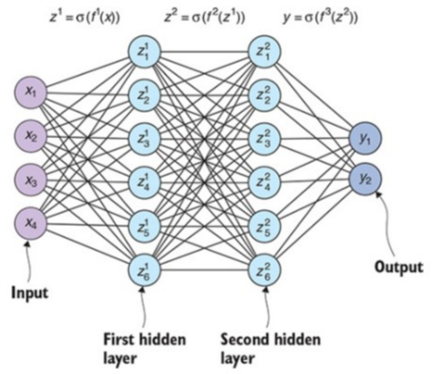
Your challenge, in this project, is for your team to continue building the infrastructure of your Go bot, including the neural network machinery it needs to do Deep learning,
The second part of your Go bot is based on the code from chapters 4 through 7 of your text: Deep Learning and the Game of Go, which creates Monte Carlo Tree Search, alpha-beta pruning, minimax scoring, encoding, creates networks using CNN's, and builds the networks for deep learning.

The second part of this project has a fair amount of code, and the Python code is more complex than before. In particular, you should make sure that your project's output matches that of the text at all of its checkpoints, to minimize long-term issues.

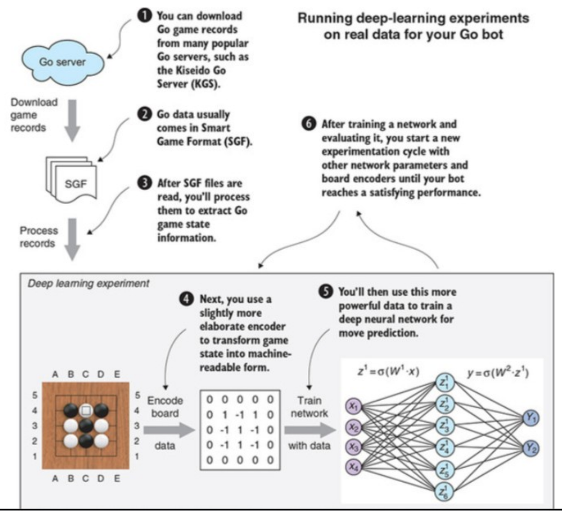
Do not wait until the last minute to try and get it written, debugged, and tested.

If you have not already done so, add the normal and fast versions of the goboard code to your Go bot engine (goboard.py and goboard_fast.py). Recall the first version of your bot used goboard_slow.py, code easier to understand, but not as fast.

Implement Monte-Carlo tree search, alpha-beta pruning and minimax (ch. 4). Create an MCTSAgent and let it play against itself.



Implement layers from ch. 5, including the sigmoid activation functions, and classes Layer, ActivationLayer, and DenseLayer. Note, you might have to modify the sigmoid functions to prevent underflow/overflow.


Running deep-learning experiments on real data for your Go bot
Go server
1 You can download Go game records from many popular Go servers, such as the Kiseido Go Server (KGS).
Download game records
2 Go data usually comes in Smart Game Format (SGF).
SGF
3 After SGF files are read, you'll process them to extract Go game state information.
Process records
6 After training a network and evaluating it, you start a new experimentation cycle with other network parameters and board encoders until your bot reaches a satisfying performance.
Deep learning experiment
4 Next, you use a slightly more elaborate encoder to transform game state into machine-readable form.
5 You'll then use this more powerful data to train a deep neural network for move prediction.
Encode board data
Train network with data
$z^1=\sigma(W^1 \cdot x)$
$y=\sigma(W^2 \cdot z^1)$

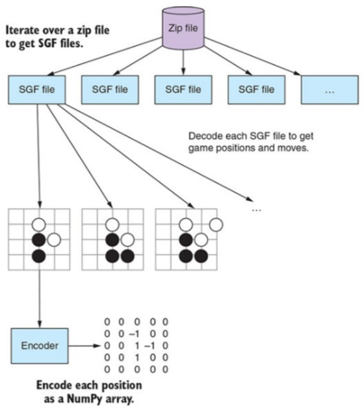
Implement the network from ch. 5, including the MSE class and SequentialNetwork, and demonstrate your network works with the mnist data

Create the base encoder class (from ch. 6), and the OnePlaneEncoder for encoding the Go board data.

Create the generate_mcts_games file, which will generate games by encoding the game state before each move, encode the move as a one-hot vector, and then apply the move to the board.

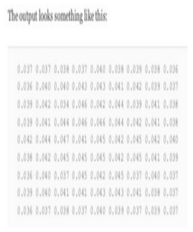
Create a program to create, and run Go games, and save them. Use it to generate 20 9x9 Go games, and store the features in features.npy, and the labels in labels.npy.

Install keras, and tensorflow (for ambitious students, consider building tensorflow from source, for higher performance). Note: tensorflow has more than 20,000 files; it will take a few hours to build it from source. You might want to consider building it overnight. You will need java8, and bazel 0.26.1 in order to build it.


Iterate over a zip file to get SGF files.
Zip file
SGF file
SGF file
SGF file
SGF file
...
Decode each SGF file to get game positions and moves.
Encoder
0 0 0 0 0
0 0 −1 0 0
0 0 1 −1 0
0 0 1 0 0
0 0 0 0 0
Encode each position as a NumPy array.

Confirm the non-convolutional neural network code from listings 6.14 and 6.15 of your text runs, and produces the output shown in your text.

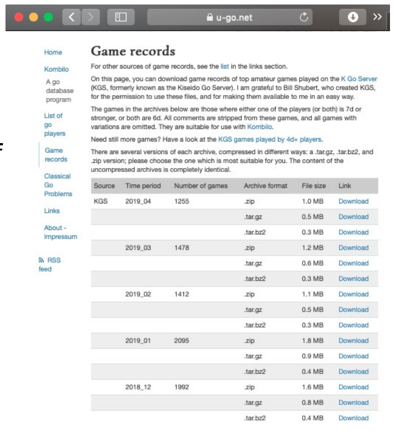
Confirm the CNN from listings 6.24, 6.25, and 6.26 of your text runs, and produces the output shown in your text. Make sure you print out the probabilities of its recommended moves (just after 6.26).


The output looks something like this:

```
0.037 0.037 0.038 0.037 0.040 0.038 0.039 0.038 0.036
0.036 0.040 0.040 0.043 0.043 0.041 0.042 0.039 0.037
0.039 0.042 0.034 0.046 0.042 0.044 0.039 0.041 0.038
0.039 0.041 0.044 0.046 0.046 0.044 0.042 0.041 0.038
0.042 0.044 0.047 0.041 0.045 0.042 0.045 0.042 0.040
0.038 0.042 0.045 0.045 0.045 0.042 0.045 0.041 0.039
0.036 0.040 0.037 0.045 0.042 0.045 0.037 0.040 0.037
0.039 0.040 0.041 0.041 0.043 0.043 0.041 0.038 0.037
0.036 0.037 0.038 0.037 0.040 0.039 0.037 0.039 0.037
```

Review the Smart Game Format (SGF) for Go. Go to https://u-go.net/gamerecords to see all of the games available for download (the games are only between masters (6 dan or above) – note these games are all on 19x19 boards).

Create the KGSIndex class that downloads these files, and download the files. (see listing 7.1).



Game records

For other sources of game records, see the list in the links section.

On this page, you can download game records of top amateur games played on the K Go Server (KGS, formerly known as the Kiseido Go Server). I am grateful to Bill Shubert, who created KGS, for the permission to use these files, and for making them available to me in an easy way.

The games in the archives below are those where either one of the players (or both) is 7d or stronger, or both are 6d. All comments are stripped from these games, and all games with variations are omitted. They are suitable for use with Kombilo.

Need still more games? Have a look at the KGS games played by 4d+ players.

There are several versions of each archive, compressed in different ways: a .tar.gz, .tar.bz2, and .zip version; please choose the one which is most suitable for you. The content of the uncompressed archives is completely identical.

Source	Time period	Number of games	Archive format	File size	Link
KGS	2019_04	1255	.zip	1.0 MB	Download
			.tar.gz	0.5 MB	Download
			.tar.bz2	0.3 MB	Download
	2019_03	1478	.zip	1.2 MB	Download
			.tar.gz	0.6 MB	Download
			.tar.bz2	0.3 MB	Download
	2019_02	1412	.zip	1.1 MB	Download
			.tar.gz	0.5 MB	Download
			.tar.bz2	0.3 MB	Download
	2019_01	2095	.zip	1.8 MB	Download
			.tar.gz	0.9 MB	Download
			.tar.bz2	0.4 MB	Download
	2018_12	1992	.zip	1.6 MB	Download
			.tar.gz	0.8 MB	Download
			.tar.bz2	0.4 MB	Download

Replay the (pretend) game from Listing 7.2. Make sure it replays the game correctly. Note, aa is at the upper-left-hand corner). Review the SGF again if your game does not look like it was played correctly on your Go bot.

Create the Go data processor that can transform raw SGF data into features and labels for a machine learning algorithm. Create files processor.py, parallel_processor.py, and networks small.py, medium.py, and large.py. Note, you may have to change the encoding of your file from NCWH (1, 19, 19) to NCWH (19, 19, 1) using utility functions, as Keras by default wants the number of channels to come last.

Create the training and test generators that use the GoDataProcessor, so that Keras can use those generators to fit the model and evaluate it. Confirm your generators and model against the output at the end of section 7.3.

Create a SevenPlaneEncoder (Figures 7.23-7.25) that models more aspects of Go correctly (black or white groups with 1, 2, or 3 liberties, or points that cannot be played because of an ongoing ko fight)

Add the Adagrad optimizer to allow adaptive gradient methods (methods that adapt the rate at which the model modifies its parameters depending on the state of the game) (see Listing 7.27).

Train your Go bot using different combinations of hyperparameters to see how to get the best performance from your engine.

- How many layers should your deep learning network have?
- How many dense layers should it have?
- Should I have dropout layers? What should the dropout percentage be?
- What type of convolution kernels should it have? How large should they be?
- What should be the stride of the kernels? Should I have zero padding?
- How many epochs should it use?
- Should I process the data in batches? If so, what should the batch size be?
- What type of activation should I use? What activation should the final layer have?
- What type of pooling makes sense? Average, maximum, minimum, or a mixture?
- What type of optimizer should I consider?
- How should loss be calculated?

- How do I know when training has stopped, and the model is now overfitting?
- What metrics should I use? Should I track training and validation loss?
- How can I measure how strong my bot is?

Note: the code used in this project will be the basis for the full Go bot engine, which will include Monte Carlo Tree Search, alpha-beta pruning, and Deep Learning.

Submission

Turn in the code for this homework by uploading all of the Python source files you created, the images directory, and the sounds directory to a single public repository on GitHub. While you may discuss this homework assignment with other students. Work you submit must have been completed on your own.

Push the contents of your project to a new GitHub repository using a git client. Do not submit files using drag-and- drop onto the repository web page, and do not push this assignment to the same repository as your previous homework assignments.

To complete your submission, print the following sheet, fill out the spaces below, and submit it to the professor in class by the deadline. Failure to follow the instructions exactly will incur a **10%** penalty on the grade for this assignment.

Bibliography:

1. Mastering the game of Go with deep neural networks and tree search,
by David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis
Nature volume 529, pages 484–489 (28 January 2016)
https://www.nature.com/articles/nature16961

2. A general reinforcement learning algorithm that masters chess, shogi and Go through self-play, by David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, **Science, 2018.**

CPSC 481 Project Two: Go bot engine

due 16 Oct at beginning of class

Your name 1337 crew; Juan Linares; Yafu Deng; Louis Tagatac; Abhyuday Vatsavai

Repository: **https://github.com/**_YafuDeng_______________________________**/**
__https://github.com/YafuDeng/CPSC481_HW2_Team1337crew______

Finished	Not Finished	Verify each of the following items and place a checkmark in the correct column. Each item incorrectly marked will incur a 5% penalty on the assignment's grade.
✓	❑	Create the generate_mcts_games file, which generates games by encoding the game state before each move, encodes the move as a one-hot vector, and applies it.
✓	❑	Implement Monte-Carlo tree search, alpha-beta pruning and minimax (ch. 4). Create an MCTSAgent and let it play against itself.
✓	❑	Create a program to create, and run Go games, and save them. Use it to generate 20 9x9 Go games, and store the features in features.py, and the labels in labels.py.
✓	❑	Confirm the CNN from listings 6.24-26 of your text runs, and produces the output shown in your text. Print out the probabilities of its recommended moves (see 6.26).
✓	❑	Create the KGSIndex class that downloads SGF files from https://u-go.net/gamerecords, and download the files. (see listing 7.1).
✓	❑	Replay the (pretend) game from Listing 7.2. Make sure it replays the game correctly.
✓	❑	Create the Go data processor that can transform raw SGF data into features and labels for a machine learning algorithm.
✓	❑	Create the OnePlaneEncoder and SevenPlaneEncoder, and verify that they produce the correct output from the text.
✓	❑	Create the training and test generators that use the GoDataProcessor, so that Keras can use those generators to fit the model and to evaluate it.
❑	❑	Add the Adagrad optimizer to allow adaptive gradient methods.
❑	❑	Train your Go bot using different hyperparameters to get best performance.
✓	❑	Be written in Python. No issues are shown in PyCharm (all source code screens shown a green checkmark at the top right hand corner).
✓	❑	Project directory pushed to new GitHub repository listed above

Comments:
deleted venv folder due to GitHub desktop refused to commit

it to master because of the size.