

Project Description

The goal of the project was to develop an AI player of a perfect information board game. This program plays the game Santorini, and allows for four different player types: (1) A human player, (2) a random CPU player, (3) a simple heuristic AI CPU player, and (4) an ANN cpu player.

This document will provide a high-level description of the project's components.

Config.h

Config.h is a header file that holds commonly referenced global variables. This is also where most libraries are included in the program, so it is a good place to reference all of the libraries required to run the program.

Player.h

Player.h is a header file that holds the definitions of all structs and classes, as well as many of the function prototypes. This file is a good place to see documentation on the various different objects used (Players, Worker, etc).

Player.cpp

Player.cpp is the source file that holds definitions for all the class functions and some related free functions as well. This is a good place to see documentation on how the objects work and interact with the game.

Main.cpp

Main.cpp is a source file to be loaded when you want to play a single game. The user gets to select the player types used.

Training.cpp

Training.cpp is a source file to be used when training the ANN and developing a new version of it.

NOTE Main.cpp and Training.cpp CANNOT be loaded in simultaneously as they both have a main() function. It is intended to have only one at a time.

ANN

The ANN player is made up of several components.

Input layer: X

- X is a boardstate vector that is inputted into the ANN

Hidden layers: $h^{(1)}, h^{(2)}, h^{(3)}$

- The hidden layers are vectors, each being one set of transformation (a weight multiplication, bias addition, and non-linear activation function) more than the previous.

Weights: $W^{(1)}, W^{(2)}, W^{(3)}, W^{(4)}$

- Weights are matrices that are multiplied by the layer before to contribute to the next layer.
- Layers 1, 2, and 3 have 50x50 matrices.
- Layer 4 has a 1x50 "matrix"

Biases: $b^{(1)}, b^{(2)}, b^{(3)}$

- Biases are vectors that are added by the layer before to contribute to the next layer.
- Layers 1, 2, and 3 have 50x1 bias vectors.

Logit / Unactivated Layer: $z^{(1)}, z^{(2)}, z^{(3)}$

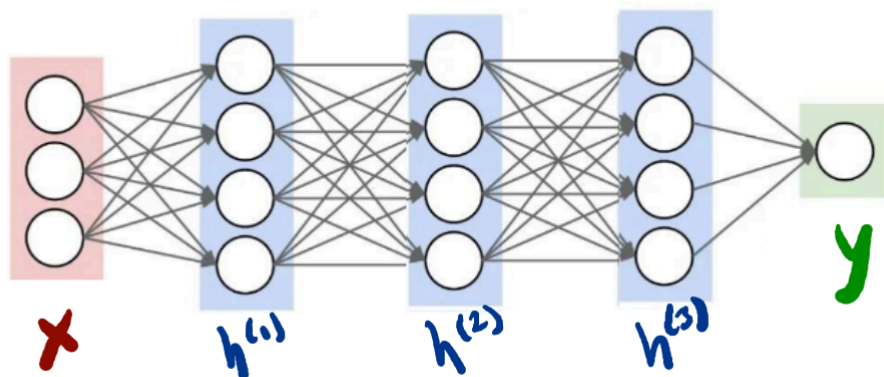
- Unactivated layers are unprocessed layers after the weight multiplication and bias addition.
- They become hidden layers by being "activated" by the activation function.

Activation Function (ReLU):

- ReLu is a basic activation function that makes negative values 0 but does not alter positive values.

Output: y

- The output of the ANN is a double representing the expected 'value' of the input.



Free Functions

```
//Free functions
/// ...
int checkWorkers(int x, int y){ ... }
/// ...
bool checkOccupied(int x, int y){ ... }
/// ...
bool checkLevels(int x, int y, int x2, int y2){ ... }
/// ...
bool checkFourth(int x, int y){ ... }
/* ... */
void exportBoardStateStream(int turn, ostream& out){ ... }
/* ... */
void exportBoardState(int turn, string& out){ ... }
/// ...
void importBoardState(int& turn, Player *p1, Player *p2, istream& in){ ... }
/// ...
void importBoardState(int& turn, Player* p1, Player* p2, double v[50]){ ... }
/* ... */
void extractBoardState(int turn, double v[50]){ ... }
/* ... */
void extractNewBoardState(int worker, int move, int placement, int turn, double v[50]){ ... }
/* ... */
void printVector(double v[50]){ ... }
/* ... */
bool adjacentSpace(string move, int& x, int& y){ ... }
/* ... */
bool adjacentSpace(int move, int& x, int& y){ ... }
/* ... */
void printBoard(int space, ostream& out){ ... }
/* ... */
void setColor(int color){ ... }
/* ... */
void selectPos(int& x, int& y, string ask){ ... }
```

The free functions are functions that do not have association with an object.

checkWorkers, checkOccupied, checkLevels, and checkFourth are all functions that help validate moves and maintain the game logic.

exportBoardState,importBoardState,extractBoardState,extractNewBoardState are all functions that have to do with representing the game as a boardstate vector, loading in and saving out boardstates, and pushing/pulling boardstates in/out of the Board and Workers global arrays (that are used to maintain the game).

adjacentSpace is used to decode a relative move as follows.

0 is upper left, 1 is upper, 2 is upper right, 3 is left, 4 is right, 5 is lower left, 6 is down, and 7 is lower right.

printBoard, setColor, and selectPos all have to do with the user interface, particularly with how the game is shown to the player and how the player can interact with it.

Main.cpp

Main.cpp features a standard game loop that plays a single game. On a given players the turn variable is swapped between 0 and 1 to indicate whose turn it is.

The primary function call is taketurn, which is used to update the board when a player does their turn. It also returns the result of the game.

Training.cpp

Training.cpp features how to train the ANN.

When A is selected, it runs 10 games and records all encountered boardstates (non-repeating). When B is selected, it plays 1000 games from each of those boardstates to measure a win rate among them.

When C is selected, it calculates the gradient (contribution of error) with respect to each variable in the ANN (all the weights and biases), and outputs a new ANN that is adjusted accordingly.

Player Object

There are 4 different player objects. On a given turn, the player will generate all given possible turns, considering both workers, all 8 movements, and all 8 placements. There is a total of 2x8x8 moves considered, generated by the getTransitions() function.

For the ANN and Heuristic player, the possible board moves available are fed into an evaluation function that provides the 'worth' of each move. These respective players will select a move within a threshold of the maximum value found.

Alternatively, the minimax algorithm is a variant way to select a move This algorithm can look ahead "depth" many moves, and minimize the best moves the opponent has available to them, increasing the current players long term win potential.