#include "network.h"

#include "utils.h"

#include "parser.h"

#include "option\_list.h"

#include "blas.h"

int inverted = 1;

int noi = 1;

//static const unsigned int n\_ind = 5;

#define n\_ind 5

typedef struct {

char \*\*data;

int n;

} moves;

char \*fgetgo(FILE \*fp)

{

if(feof(fp)) return 0;

size\_t size = 94;

char\* line = (char\*)xmalloc(size \* sizeof(char));

if(size != fread(line, sizeof(char), size, fp)){

free(line);

return 0;

}

return line;

}

moves load\_go\_moves(char \*filename)

{

moves m;

m.n = 128;

m.data = (char\*\*)xcalloc(128, sizeof(char\*));

FILE \*fp = fopen(filename, "rb");

int count = 0;

char \*line = 0;

while((line = fgetgo(fp))){

if(count >= m.n){

m.n \*= 2;

m.data = (char\*\*)xrealloc(m.data, m.n \* sizeof(char\*));

}

m.data[count] = line;

++count;

}

printf("%d\n", count);

m.n = count;

m.data = (char\*\*)xrealloc(m.data, count \* sizeof(char\*));

fclose(fp);

return m;

}

void string\_to\_board(char \*s, float \*board)

{

int i, j;

//memset(board, 0, 1\*19\*19\*sizeof(float));

int count = 0;

for(i = 0; i < 91; ++i){

char c = s[i];

for(j = 0; j < 4; ++j){

int me = (c >> (2\*j)) & 1;

int you = (c >> (2\*j + 1)) & 1;

if (me) board[count] = 1;

else if (you) board[count] = -1;

else board[count] = 0;

++count;

if(count >= 19\*19) break;

}

}

}

void board\_to\_string(char \*s, float \*board)

{

int i, j;

memset(s, 0, (19\*19/4+1)\*sizeof(char));

int count = 0;

for(i = 0; i < 91; ++i){

for(j = 0; j < 4; ++j){

int me = (board[count] == 1);

int you = (board[count] == -1);

if (me) s[i] = s[i] | (1<<(2\*j));

if (you) s[i] = s[i] | (1<<(2\*j + 1));

++count;

if(count >= 19\*19) break;

}

}

}

void random\_go\_moves(moves m, float \*boards, float \*labels, int n)

{

int i;

memset(labels, 0, 19\*19\*n\*sizeof(float));

for(i = 0; i < n; ++i){

char \*b = m.data[rand()%m.n];

int row = b[0];

int col = b[1];

labels[col + 19\*(row + i\*19)] = 1;

string\_to\_board(b+2, boards+i\*19\*19);

boards[col + 19\*(row + i\*19)] = 0;

int flip = rand()%2;

int rotate = rand()%4;

image in = float\_to\_image(19, 19, 1, boards+i\*19\*19);

image out = float\_to\_image(19, 19, 1, labels+i\*19\*19);

if(flip){

flip\_image(in);

flip\_image(out);

}

rotate\_image\_cw(in, rotate);

rotate\_image\_cw(out, rotate);

}

}

void train\_go(char \*cfgfile, char \*weightfile)

{

srand(time(0));

float avg\_loss = -1;

char \*base = basecfg(cfgfile);

printf("%s\n", base);

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

printf("Learning Rate: %g, Momentum: %g, Decay: %g\n", net.learning\_rate, net.momentum, net.decay);

char\* backup\_directory = "backup/";

char buff[256];

float\* board = (float\*)xcalloc(19 \* 19 \* net.batch, sizeof(float));

float\* move = (float\*)xcalloc(19 \* 19 \* net.batch, sizeof(float));

moves m = load\_go\_moves("backup/go.train");

//moves m = load\_go\_moves("games.txt");

int N = m.n;

int epoch = (\*net.seen)/N;

while(get\_current\_batch(net) < net.max\_batches || net.max\_batches == 0){

clock\_t time=clock();

random\_go\_moves(m, board, move, net.batch);

float loss = train\_network\_datum(net, board, move) / net.batch;

if(avg\_loss == -1) avg\_loss = loss;

avg\_loss = avg\_loss\*.95 + loss\*.05;

printf("%d, %.3f: %f, %f avg, %f rate, %lf seconds, %ld images\n", get\_current\_batch(net), (float)(\*net.seen)/N, loss, avg\_loss, get\_current\_rate(net), sec(clock()-time), \*net.seen);

if(\*net.seen/N > epoch){

epoch = \*net.seen/N;

char buff[256];

sprintf(buff, "%s/%s\_%d.weights", backup\_directory,base, epoch);

save\_weights(net, buff);

}

if(get\_current\_batch(net)%100 == 0){

char buff[256];

sprintf(buff, "%s/%s.backup",backup\_directory,base);

save\_weights(net, buff);

}

if(get\_current\_batch(net)%10000 == 0){

char buff[256];

sprintf(buff, "%s/%s\_%d.backup",backup\_directory,base,get\_current\_batch(net));

save\_weights(net, buff);

}

}

sprintf(buff, "%s/%s.weights", backup\_directory, base);

save\_weights(net, buff);

free\_network(net);

free(base);

free(board);

free(move);

}

void propagate\_liberty(float \*board, int \*lib, int \*visited, int row, int col, int side)

{

if (row < 0 || row > 18 || col < 0 || col > 18) return;

int index = row\*19 + col;

if (board[index] != side) return;

if (visited[index]) return;

visited[index] = 1;

lib[index] += 1;

propagate\_liberty(board, lib, visited, row+1, col, side);

propagate\_liberty(board, lib, visited, row-1, col, side);

propagate\_liberty(board, lib, visited, row, col+1, side);

propagate\_liberty(board, lib, visited, row, col-1, side);

}

int \*calculate\_liberties(float \*board)

{

int\* lib = (int\*)xcalloc(19 \* 19, sizeof(int));

int visited[361];

int i, j;

for(j = 0; j < 19; ++j){

for(i = 0; i < 19; ++i){

memset(visited, 0, 19\*19\*sizeof(int));

int index = j\*19 + i;

if(board[index] == 0){

if ((i > 0) && board[index - 1]) propagate\_liberty(board, lib, visited, j, i-1, board[index-1]);

if ((i < 18) && board[index + 1]) propagate\_liberty(board, lib, visited, j, i+1, board[index+1]);

if ((j > 0) && board[index - 19]) propagate\_liberty(board, lib, visited, j-1, i, board[index-19]);

if ((j < 18) && board[index + 19]) propagate\_liberty(board, lib, visited, j+1, i, board[index+19]);

}

}

}

return lib;

}

void print\_board(float \*board, int swap, int \*indexes)

{

//FILE \*stream = stdout;

FILE \*stream = stderr;

int i,j,n;

fprintf(stream, "\n\n");

fprintf(stream, " ");

for(i = 0; i < 19; ++i){

fprintf(stream, "%c ", 'A' + i + 1\*(i > 7 && noi));

}

fprintf(stream, "\n");

for(j = 0; j < 19; ++j){

fprintf(stream, "%2d", (inverted) ? 19-j : j+1);

for(i = 0; i < 19; ++i){

int index = j\*19 + i;

if(indexes){

int found = 0;

for (n = 0; n < n\_ind; ++n) {

if(index == indexes[n]){

found = 1;

/\*

if(n == 0) fprintf(stream, "\uff11");

else if(n == 1) fprintf(stream, "\uff12");

else if(n == 2) fprintf(stream, "\uff13");

else if(n == 3) fprintf(stream, "\uff14");

else if(n == 4) fprintf(stream, "\uff15");

\*/

if(n == 0) fprintf(stream, " 1");

else if(n == 1) fprintf(stream, " 2");

else if(n == 2) fprintf(stream, " 3");

else if(n == 3) fprintf(stream, " 4");

else if(n == 4) fprintf(stream, " 5");

}

}

if(found) continue;

}

//if(board[index]\*-swap > 0) fprintf(stream, "\u25C9 ");

//else if(board[index]\*-swap < 0) fprintf(stream, "\u25EF ");

if(board[index]\*-swap > 0) fprintf(stream, " O");

else if(board[index]\*-swap < 0) fprintf(stream, " X");

else fprintf(stream, " ");

}

fprintf(stream, "\n");

}

}

void flip\_board(float \*board)

{

int i;

for(i = 0; i < 19\*19; ++i){

board[i] = -board[i];

}

}

void predict\_move(network net, float \*board, float \*move, int multi)

{

float \*output = network\_predict(net, board);

copy\_cpu(19\*19, output, 1, move, 1);

int i;

if(multi){

image bim = float\_to\_image(19, 19, 1, board);

for(i = 1; i < 8; ++i){

rotate\_image\_cw(bim, i);

if(i >= 4) flip\_image(bim);

float \*output = network\_predict(net, board);

image oim = float\_to\_image(19, 19, 1, output);

if(i >= 4) flip\_image(oim);

rotate\_image\_cw(oim, -i);

axpy\_cpu(19\*19, 1, output, 1, move, 1);

if(i >= 4) flip\_image(bim);

rotate\_image\_cw(bim, -i);

}

scal\_cpu(19\*19, 1./8., move, 1);

}

for(i = 0; i < 19\*19; ++i){

if(board[i]) move[i] = 0;

}

}

void remove\_connected(float \*b, int \*lib, int p, int r, int c)

{

if (r < 0 || r >= 19 || c < 0 || c >= 19) return;

if (b[r\*19 + c] != p) return;

if (lib[r\*19 + c] != 1) return;

b[r\*19 + c] = 0;

remove\_connected(b, lib, p, r+1, c);

remove\_connected(b, lib, p, r-1, c);

remove\_connected(b, lib, p, r, c+1);

remove\_connected(b, lib, p, r, c-1);

}

void move\_go(float \*b, int p, int r, int c)

{

int \*l = calculate\_liberties(b);

b[r\*19 + c] = p;

remove\_connected(b, l, -p, r+1, c);

remove\_connected(b, l, -p, r-1, c);

remove\_connected(b, l, -p, r, c+1);

remove\_connected(b, l, -p, r, c-1);

free(l);

}

int makes\_safe\_go(float \*b, int \*lib, int p, int r, int c){

if (r < 0 || r >= 19 || c < 0 || c >= 19) return 0;

if (b[r\*19 + c] == -p){

if (lib[r\*19 + c] > 1) return 0;

else return 1;

}

if (b[r\*19 + c] == 0) return 1;

if (lib[r\*19 + c] > 1) return 1;

return 0;

}

int suicide\_go(float \*b, int p, int r, int c)

{

int \*l = calculate\_liberties(b);

int safe = 0;

safe = safe || makes\_safe\_go(b, l, p, r+1, c);

safe = safe || makes\_safe\_go(b, l, p, r-1, c);

safe = safe || makes\_safe\_go(b, l, p, r, c+1);

safe = safe || makes\_safe\_go(b, l, p, r, c-1);

free(l);

return !safe;

}

int legal\_go(float \*b, char \*ko, int p, int r, int c)

{

if (b[r\*19 + c]) return 0;

char curr[91];

char next[91];

board\_to\_string(curr, b);

move\_go(b, p, r, c);

board\_to\_string(next, b);

string\_to\_board(curr, b);

if(memcmp(next, ko, 91) == 0) return 0;

return 1;

}

int generate\_move(network net, int player, float \*board, int multi, float thresh, float temp, char \*ko, int print)

{

int i, j;

for(i = 0; i < net.n; ++i) net.layers[i].temperature = temp;

float move[361];

if (player < 0) flip\_board(board);

predict\_move(net, board, move, multi);

if (player < 0) flip\_board(board);

for(i = 0; i < 19; ++i){

for(j = 0; j < 19; ++j){

if (!legal\_go(board, ko, player, i, j)) move[i\*19 + j] = 0;

}

}

int indexes[n\_ind];

top\_k(move, 19\*19, n\_ind, indexes);

if(thresh > move[indexes[0]]) thresh = move[indexes[n\_ind-1]];

for(i = 0; i < 19; ++i){

for(j = 0; j < 19; ++j){

if (move[i\*19 + j] < thresh) move[i\*19 + j] = 0;

}

}

int max = max\_index(move, 19\*19);

int row = max / 19;

int col = max % 19;

int index = sample\_array(move, 19\*19);

if(print){

top\_k(move, 19\*19, n\_ind, indexes);

for(i = 0; i < n\_ind; ++i){

if (!move[indexes[i]]) indexes[i] = -1;

}

print\_board(board, player, indexes);

for(i = 0; i < n\_ind; ++i){

fprintf(stderr, "%d: %f\n", i+1, move[indexes[i]]);

}

}

if(suicide\_go(board, player, row, col)){

return -1;

}

if(suicide\_go(board, player, index/19, index%19)) index = max;

return index;

}

void valid\_go(char \*cfgfile, char \*weightfile, int multi)

{

srand(time(0));

char \*base = basecfg(cfgfile);

printf("%s\n", base);

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

set\_batch\_network(&net, 1);

printf("Learning Rate: %g, Momentum: %g, Decay: %g\n", net.learning\_rate, net.momentum, net.decay);

float\* board = (float\*)xcalloc(19 \* 19, sizeof(float));

float\* move = (float\*)xcalloc(19 \* 19, sizeof(float));

moves m = load\_go\_moves("backup/go.test");

int N = m.n;

int i;

int correct = 0;

for(i = 0; i <N; ++i){

char \*b = m.data[i];

int row = b[0];

int col = b[1];

int truth = col + 19\*row;

string\_to\_board(b+2, board);

predict\_move(net, board, move, multi);

int index = max\_index(move, 19\*19);

if(index == truth) ++correct;

printf("%d Accuracy %f\n", i, (float) correct/(i+1));

}

free(board);

free(move);

}

void engine\_go(char \*filename, char \*weightfile, int multi)

{

network net = parse\_network\_cfg(filename);

if(weightfile){

load\_weights(&net, weightfile);

}

srand(time(0));

set\_batch\_network(&net, 1);

float\* board = (float\*)xcalloc(19 \* 19, sizeof(float));

char\* one = (char\*)xcalloc(91, sizeof(char));

char\* two = (char\*)xcalloc(91, sizeof(char));

int passed = 0;

while(1){

char buff[256];

int id = 0;

int has\_id = (scanf("%d", &id) == 1);

scanf("%s", buff);

if (feof(stdin)) break;

char ids[256];

sprintf(ids, "%d", id);

//fprintf(stderr, "%s\n", buff);

if (!has\_id) ids[0] = 0;

if (!strcmp(buff, "protocol\_version")){

printf("=%s 2\n\n", ids);

} else if (!strcmp(buff, "name")){

printf("=%s DarkGo\n\n", ids);

} else if (!strcmp(buff, "version")){

printf("=%s 1.0\n\n", ids);

} else if (!strcmp(buff, "known\_command")){

char comm[256];

scanf("%s", comm);

int known = (!strcmp(comm, "protocol\_version") ||

!strcmp(comm, "name") ||

!strcmp(comm, "version") ||

!strcmp(comm, "known\_command") ||

!strcmp(comm, "list\_commands") ||

!strcmp(comm, "quit") ||

!strcmp(comm, "boardsize") ||

!strcmp(comm, "clear\_board") ||

!strcmp(comm, "komi") ||

!strcmp(comm, "final\_status\_list") ||

!strcmp(comm, "play") ||

!strcmp(comm, "genmove"));

if(known) printf("=%s true\n\n", ids);

else printf("=%s false\n\n", ids);

} else if (!strcmp(buff, "list\_commands")){

printf("=%s protocol\_version\nname\nversion\nknown\_command\nlist\_commands\nquit\nboardsize\nclear\_board\nkomi\nplay\ngenmove\nfinal\_status\_list\n\n", ids);

} else if (!strcmp(buff, "quit")){

break;

} else if (!strcmp(buff, "boardsize")){

int boardsize = 0;

scanf("%d", &boardsize);

//fprintf(stderr, "%d\n", boardsize);

if(boardsize != 19){

printf("?%s unacceptable size\n\n", ids);

} else {

printf("=%s \n\n", ids);

}

} else if (!strcmp(buff, "clear\_board")){

passed = 0;

memset(board, 0, 19\*19\*sizeof(float));

printf("=%s \n\n", ids);

} else if (!strcmp(buff, "komi")){

float komi = 0;

scanf("%f", &komi);

printf("=%s \n\n", ids);

} else if (!strcmp(buff, "play")){

char color[256];

scanf("%s ", color);

char c;

int r;

int count = scanf("%c%d", &c, &r);

int player = (color[0] == 'b' || color[0] == 'B') ? 1 : -1;

if(c == 'p' && count < 2) {

passed = 1;

printf("=%s \n\n", ids);

char \*line = fgetl(stdin);

free(line);

fflush(stdout);

fflush(stderr);

continue;

} else {

passed = 0;

}

if(c >= 'A' && c <= 'Z') c = c - 'A';

if(c >= 'a' && c <= 'z') c = c - 'a';

if(c >= 8) --c;

r = 19 - r;

fprintf(stderr, "move: %d %d\n", r, c);

char \*swap = two;

two = one;

one = swap;

move\_go(board, player, r, c);

board\_to\_string(one, board);

printf("=%s \n\n", ids);

print\_board(board, 1, 0);

} else if (!strcmp(buff, "genmove")){

char color[256];

scanf("%s", color);

int player = (color[0] == 'b' || color[0] == 'B') ? 1 : -1;

int index = generate\_move(net, player, board, multi, .1, .7, two, 1);

if(passed || index < 0){

printf("=%s pass\n\n", ids);

passed = 0;

} else {

int row = index / 19;

int col = index % 19;

char \*swap = two;

two = one;

one = swap;

move\_go(board, player, row, col);

board\_to\_string(one, board);

row = 19 - row;

if (col >= 8) ++col;

printf("=%s %c%d\n\n", ids, 'A' + col, row);

print\_board(board, 1, 0);

}

} else if (!strcmp(buff, "p")){

//print\_board(board, 1, 0);

} else if (!strcmp(buff, "final\_status\_list")){

char type[256];

scanf("%s", type);

fprintf(stderr, "final\_status\n");

char \*line = fgetl(stdin);

free(line);

if(type[0] == 'd' || type[0] == 'D'){

FILE \*f = fopen("game.txt", "w");

int i, j;

int count = 2;

fprintf(f, "boardsize 19\n");

fprintf(f, "clear\_board\n");

for(j = 0; j < 19; ++j){

for(i = 0; i < 19; ++i){

if(board[j\*19 + i] == 1) fprintf(f, "play black %c%d\n", 'A'+i+(i>=8), 19-j);

if(board[j\*19 + i] == -1) fprintf(f, "play white %c%d\n", 'A'+i+(i>=8), 19-j);

if(board[j\*19 + i]) ++count;

}

}

fprintf(f, "final\_status\_list dead\n");

fclose(f);

#ifdef \_WIN32

FILE \*p = \_popen("./gnugo --mode gtp < game.txt", "r");

#else

FILE \*p = popen("./gnugo --mode gtp < game.txt", "r");

#endif

for(i = 0; i < count; ++i){

free(fgetl(p));

free(fgetl(p));

}

char \*l = 0;

while((l = fgetl(p))){

printf("%s\n", l);

free(l);

}

} else {

printf("?%s unknown command\n\n", ids);

}

} else {

char \*line = fgetl(stdin);

free(line);

printf("?%s unknown command\n\n", ids);

}

fflush(stdout);

fflush(stderr);

}

}

void test\_go(char \*cfg, char \*weights, int multi)

{

network net = parse\_network\_cfg(cfg);

if(weights){

load\_weights(&net, weights);

}

srand(time(0));

set\_batch\_network(&net, 1);

float\* board = (float\*)xcalloc(19 \* 19, sizeof(float));

float\* move = (float\*)xcalloc(19 \* 19, sizeof(float));

int color = 1;

while(1){

float \*output = network\_predict(net, board);

copy\_cpu(19\*19, output, 1, move, 1);

int i;

if(multi){

image bim = float\_to\_image(19, 19, 1, board);

for(i = 1; i < 8; ++i){

rotate\_image\_cw(bim, i);

if(i >= 4) flip\_image(bim);

float \*output = network\_predict(net, board);

image oim = float\_to\_image(19, 19, 1, output);

if(i >= 4) flip\_image(oim);

rotate\_image\_cw(oim, -i);

axpy\_cpu(19\*19, 1, output, 1, move, 1);

if(i >= 4) flip\_image(bim);

rotate\_image\_cw(bim, -i);

}

scal\_cpu(19\*19, 1./8., move, 1);

}

for(i = 0; i < 19\*19; ++i){

if(board[i]) move[i] = 0;

}

int indexes[n\_ind];

int row, col;

top\_k(move, 19 \* 19, n\_ind, indexes);

print\_board(board, color, indexes);

for (i = 0; i < n\_ind; ++i) {

int index = indexes[i];

row = index / 19;

col = index % 19;

printf("%d: %c %d, %.2f%%\n", i+1, col + 'A' + 1\*(col > 7 && noi), (inverted)?19 - row : row+1, move[index]\*100);

}

//if(color == 1) printf("\u25EF Enter move: ");

//else printf("\u25C9 Enter move: ");

if(color == 1) printf("X Enter move: ");

else printf("O Enter move: ");

char c;

char \*line = fgetl(stdin);

int picked = 1;

int dnum = sscanf(line, "%d", &picked);

int cnum = sscanf(line, "%c", &c);

if (strlen(line) == 0 || dnum) {

--picked;

if (picked < n\_ind){

int index = indexes[picked];

row = index / 19;

col = index % 19;

board[row\*19 + col] = 1;

}

} else if (cnum){

if (c <= 'T' && c >= 'A'){

int num = sscanf(line, "%c %d", &c, &row);

row = (inverted)?19 - row : row-1;

col = c - 'A';

if (col > 7 && noi) col -= 1;

if (num == 2) board[row\*19 + col] = 1;

} else if (c == 'p') {

// Pass

} else if(c=='b' || c == 'w'){

char g;

int num = sscanf(line, "%c %c %d", &g, &c, &row);

row = (inverted)?19 - row : row-1;

col = c - 'A';

if (col > 7 && noi) col -= 1;

if (num == 3) board[row\*19 + col] = (g == 'b') ? color : -color;

} else if(c == 'c'){

char g;

int num = sscanf(line, "%c %c %d", &g, &c, &row);

row = (inverted)?19 - row : row-1;

col = c - 'A';

if (col > 7 && noi) col -= 1;

if (num == 3) board[row\*19 + col] = 0;

}

}

free(line);

flip\_board(board);

color = -color;

}

}

float score\_game(float \*board)

{

FILE \*f = fopen("game.txt", "w");

int i, j;

int count = 3;

fprintf(f, "komi 6.5\n");

fprintf(f, "boardsize 19\n");

fprintf(f, "clear\_board\n");

for(j = 0; j < 19; ++j){

for(i = 0; i < 19; ++i){

if(board[j\*19 + i] == 1) fprintf(f, "play black %c%d\n", 'A'+i+(i>=8), 19-j);

if(board[j\*19 + i] == -1) fprintf(f, "play white %c%d\n", 'A'+i+(i>=8), 19-j);

if(board[j\*19 + i]) ++count;

}

}

fprintf(f, "final\_score\n");

fclose(f);

#ifdef \_WIN32

FILE \*p = \_popen("./gnugo --mode gtp < game.txt", "r");

#else

FILE \*p = popen("./gnugo --mode gtp < game.txt", "r");

#endif

for(i = 0; i < count; ++i){

free(fgetl(p));

free(fgetl(p));

}

char \*l = 0;

float score = 0;

char player = 0;

while((l = fgetl(p))){

fprintf(stderr, "%s \t", l);

int n = sscanf(l, "= %c+%f", &player, &score);

free(l);

if (n == 2) break;

}

if(player == 'W') score = -score;

#ifdef \_WIN32

\_pclose(p);

#else

pclose(p);

#endif

return score;

}

void self\_go(char \*filename, char \*weightfile, char \*f2, char \*w2, int multi)

{

network net = parse\_network\_cfg(filename);

if(weightfile){

load\_weights(&net, weightfile);

}

network net2 = net;

if(f2){

net2 = parse\_network\_cfg(f2);

if(w2){

load\_weights(&net2, w2);

}

}

srand(time(0));

char boards[300][93];

int count = 0;

set\_batch\_network(&net, 1);

set\_batch\_network(&net2, 1);

float\* board = (float\*)xcalloc(19 \* 19, sizeof(float));

char\* one = (char\*)xcalloc(91, sizeof(char));

char\* two = (char\*)xcalloc(91, sizeof(char));

int done = 0;

int player = 1;

int p1 = 0;

int p2 = 0;

int total = 0;

while(1){

if (done || count >= 300){

float score = score\_game(board);

int i = (score > 0)? 0 : 1;

if((score > 0) == (total%2==0)) ++p1;

else ++p2;

++total;

fprintf(stderr, "Total: %d, Player 1: %f, Player 2: %f\n", total, (float)p1/total, (float)p2/total);

int j;

for(; i < count; i += 2){

for(j = 0; j < 93; ++j){

printf("%c", boards[i][j]);

}

printf("\n");

}

memset(board, 0, 19\*19\*sizeof(float));

player = 1;

done = 0;

count = 0;

fflush(stdout);

fflush(stderr);

}

//print\_board(board, 1, 0);

//sleep(1);

network use = ((total%2==0) == (player==1)) ? net : net2;

int index = generate\_move(use, player, board, multi, .1, .7, two, 0);

if(index < 0){

done = 1;

continue;

}

int row = index / 19;

int col = index % 19;

char \*swap = two;

two = one;

one = swap;

if(player < 0) flip\_board(board);

boards[count][0] = row;

boards[count][1] = col;

board\_to\_string(boards[count] + 2, board);

if(player < 0) flip\_board(board);

++count;

move\_go(board, player, row, col);

board\_to\_string(one, board);

player = -player;

}

free(board);

free(one);

free(two);

}

void run\_go(int argc, char \*\*argv)

{

//boards\_go();

if(argc < 4){

fprintf(stderr, "usage: %s %s [train/test/valid] [cfg] [weights (optional)]\n", argv[0], argv[1]);

return;

}

char \*cfg = argv[3];

char \*weights = (argc > 4) ? argv[4] : 0;

char \*c2 = (argc > 5) ? argv[5] : 0;

char \*w2 = (argc > 6) ? argv[6] : 0;

int multi = find\_arg(argc, argv, "-multi");

if(0==strcmp(argv[2], "train")) train\_go(cfg, weights);

else if(0==strcmp(argv[2], "valid")) valid\_go(cfg, weights, multi);

else if(0==strcmp(argv[2], "self")) self\_go(cfg, weights, c2, w2, multi);

else if(0==strcmp(argv[2], "test")) test\_go(cfg, weights, multi);

else if(0==strcmp(argv[2], "engine")) engine\_go(cfg, weights, multi);

}