**Deepbot**



Distributed Information Systems Laboratory

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Deepbot

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Professor : Karl Aberer

Assistant : Nguyen Thanh Tam

Cyril van Schreven | LSIR-EPFL | Jul 2019

**Table of contents**

[I- Abstract 4](#_Toc1970438481)

[II- Introduction 4](#_Toc1688040915)

[A- Bla 4](#_Toc744552560)

## I- Abstract

The

## II- Introduction

### A- SubChapter

MAIN POINTS

**Idea:**

What is the objective?

- 6max   
- Sit and go?  
- Online play at low stakes

However:  
- Must find realistic solution with limited computing power  
- Lots of poker bots / AI exist

How to contribute to the field?

- Focus on exploitative play, instead of Game Theory Optimal (GTO)  
Exploitative play is especially relevant for multiplayer games which involves the kingmaker effect. A weak bot exploiting weak players can win more game than a perfect GTO bot.  
Exploiting means adapting to the opponent.  
- Focus on real-world testing (against humans, online)  
- Focus on explainability? Writing a new strategy or visualize motivations for decision?

**Poker fundamentals and strategy:**

HU-limit : 10¹³ decision points

HUNL : 10¹⁶¹ decision points

-1326 two-card combinations

Action abstraction:

s

Hand abstraction:

s

**API:**

Approach:   
Take screenshot and use object detection

Capturing state:   
1. Take repeated screenshots until a table is spotted, then ‘initialize’.   
2. Take repeated screenshots and wait for hero’s turn. It is hero’s turn when either ‘call’ or ‘check’ is available.   
3. Take a screenshot and do full table (state) scan. When action is taken go back to point 2.

Screen capture summarizes all info necessary:  
- buttons: fold, check, call, bet, raise\_to, bet\_sizer, bet\_value\_box. As well as corresponding call / min raise values  
- Cards: hero\_cards, community\_cards (suit + number)  
- Dealer-button position  
- Players: state (folded, available), stack, bet, (and will need player name?)  
- Main pot

Processing info and taking action:  
Info is sent to the ‘Bot’ Class. Same class is used for local simulated bot vs bot games. Bot class returns action and amount, and the action is handled by API.

Generalizing for other website:   
Developped with generalization in mind (especially no absolute position values). Is done for Pokerstars, must develop for site without user agreement -> ReplayPoker **(to-do)**

Other points:  
- Will be used for final ‘real-world’ validation  
- Written OOP  
- Can be slow to look for screen-elements. Speedup implemented: for players detection (or cards, or buttons): First look everywhere. When all players (resp cards, resp buttons) are spotted, remember position and look there next time, with small margin. To-do for bets and dealer button, if applicable.  
- Avoiding detection by website: randomize click timings and click positions  
- Explain how numbers are treated: Read, then ordered, then attributed (to player or main pot), then understood.  
- Reference images are required for detection  
- To-do: handle side pots

**Decision making algorithm: (to-do)**

**Approach 1: Deep RL**

The network’s input and output:

Inputs (state + action):   
- Hero’s stack (in big blinds)  
- current street (preflop, flop, turn or river. As one-hot encoding)  
- equity (regular; if the game ends after river)  
- equity if premature ending (on preflop, flop or turn)  
- number of players in the hand  
- action taken (check, call or raise. As one hot encoding)   
- amount (if raise, else = 0)  
-> size: 14 (fill to 16)   
Can and should be extended with action description (first raiser, total chips commited, total number of raises) as well as stats about player (VPP, PFR, Continuation bet rate, 3B rate, fold rate).

Output (or label)   
- chips won (in big blinds. Can be negative. Measured for the whole hand)

Can be seen as a transition function from a state+action straight to the end of the hand.

Taking a decision:   
1. Consider each possible action and compute estimate winnings through neural net  
2. Choose action with the estimated return (chips won)

Architecture (to explore and test):  
- 3 dense (aka fully-connected) layer  
- Sizes: 16 -> 32 -> 8  
- Activation function: ReLu  
- Loss: MSE  
- Optimizer: ADAM

**Approach 2: LSTM with genetic algorithm** (as in paper: http://nn.cs.utexas.edu/downloads/papers/xun.aaai17.pdf)

The network’s input and output

Inputs ():  
- current street (preflop, flop, turn or river. As one-hot encoding)  
- equity (aka winning probability)  
- amount of chips commited by the opponent  
- amount of chips commited by the player  
- pot odds  
-> size: 8

Output (or label):  
- Percentage of starting stack size to commit between 0 and 1

Taking a decision:   
1. Compute chips to commit  
2. - if lower than call value: Check/Fold  
 - elif lower than minimum raise value: Check/Call   
 - else: raise to amount, rounded to closest BB (adjusted version discriminates further into only 5 categories, to mitigate opponent’s readings)

Architecture  
Three neural network modules:  
# a decision network; fully connected with one hidden layer  
 - input size: 10  
 - number of hidden units: unknown  
 - output size: 1  
# an opponent module composed of 1 LSTM block  
 - cell size: 5  
# a game module composed of 10 LSTM blocks  
 - cell size : 5

Trained with genetic algorithm:  
 - Serialize and concatenate all weights (weight matrices, weight vectors, bias vectors). Number of weights in an LSTM block: 355 (8\*8\*5+7\*5)  
 - Number of hands per generation (per LSTM player per opponent player): 1000  
 - Number of generations: 250  
 - Number of LSTM player (population size): 50

- Number of opponent players : 4  
 - Survival rate: 0.30  
 - Mutation rate: 0.25/0.05 (decrease linearly at each generation)  
 - Mutation strength: 0.50/0.10 (decrease linearly at each generation)  
 - Total hands simulated for training: 50 millions (1000\*4\*50\*250)

NOTES:

-LSTM weights initialized as U(-k,k) where k = 1/sqrt(hidden\_size)

- required packages: python=3.5, numpy, pytorch, pandas

- number of weights : 31701, nb of layers : 70

- “generic opponent modeling”

Conda create -n poker\_env python=3.5

Source activate poker\_env

Conda install numpy

conda install pytorch-cpu torchvision-cpu -c pytorch

Conda install pandas

Can do for report:

- show that crossover by layer gives closer outputs than crossover by odd-even (see debugging.py)

REFS note

Ref 1: “As a result of the ANN study, two particularly strong features for prediction were identified: previous action, and previous amount to call”