Week 2 Pokerbots

CFR Review

<u>Mixed strategies:</u>

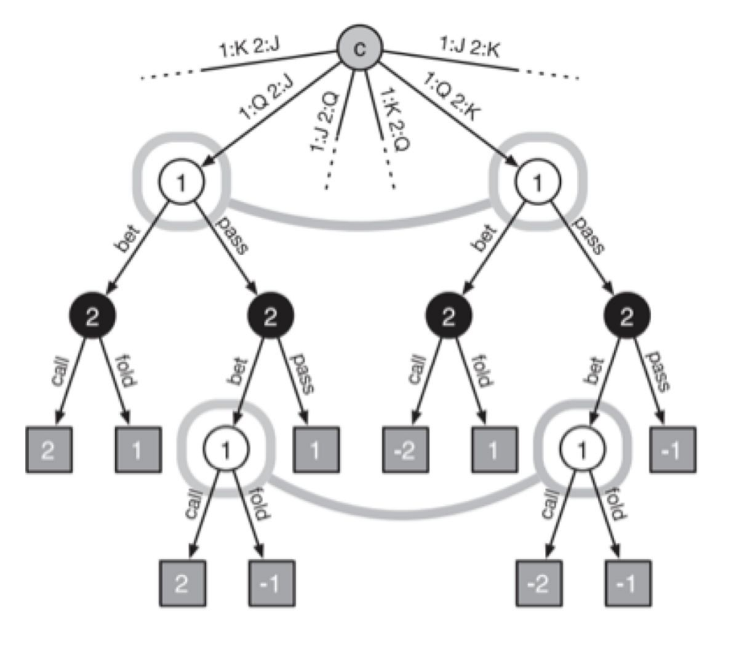
Which action to take at a specific action node is not deterministic, it is a set of probabilities

<u>Regret for taking an action:</u>

Expected value of taking this action – Expected value of following strategy

<u>Chance sampling:</u>

One combination of cards dealt to players is used/sampled for the algorithm, in vanilla CFR all combinations are used



Regret matching

- Last week, we implemented the cfr algorithm, except for one crucial part:

  Regret Matching

Algorithm 1 Counterfactual Regret Minimization (with chance sampling)

Initialize cumulative regret tables: $\forall I, r_I[a] \leftarrow 0$.
Initialize cumulative strategy tables: $\forall I, s_I[a] \leftarrow 0$.
Initialize initial profile: $\sigma^1(I,a) \leftarrow 1/|A(I)|$

function CFR(h, i, t, π_1, π_2):
if h is terminal **then**
    return $u_i(h)$
else if h is a chance node **then**
    Sample a single outcome $a \sim \sigma_c(h,a)$
    return CFR(ha, i, t, π_1, π_2)
end if
Let I be the information set containing h.
$v_\sigma \leftarrow 0$
$v_{\sigma_{I \to a}}[a] \leftarrow 0$ for all $a \in A(I)$
for $a \in A(I)$ **do**
    if $P(h) = 1$ **then**
        $v_{\sigma_{I \to a}}[a] \leftarrow$ CFR(ha, i, t, $\sigma^t(I,a) \cdot \pi_1$, π_2)
    else if $P(h) = 2$ **then**
        $v_{\sigma_{I \to a}}[a] \leftarrow$ CFR(ha, i, t, π_1, $\sigma^t(I,a) \cdot \pi_2$)
    end if
    $v_\sigma \leftarrow v_\sigma + \sigma^t(I,a) \cdot v_{\sigma_{I \to a}}[a]$
end for
if $P(h) = i$ **then**
    for $a \in A(I)$ **do**
        $r_I[a] \leftarrow r_I[a] + \pi_{-i} \cdot (v_{\sigma_{I \to a}}[a] - v_\sigma)$
        $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma^t(I,a)$
    end for
    $\sigma^{t+1}(I) \leftarrow$ regret-matching values computed using Equation 5 and regret table r_I
end if
return v_σ

function Solve():
for $t = \{1, 2, 3, \ldots, T\}$ **do**
    for $i \in \{1, 2\}$ **do**
        CFR($\emptyset$, i, t, 1, 1)
    end for
end for

Regret Matching cont

- Compute the proportion of regret over sum of all regrets experienced at the Information state
- Determines the strategy player i will take at iteration T+1

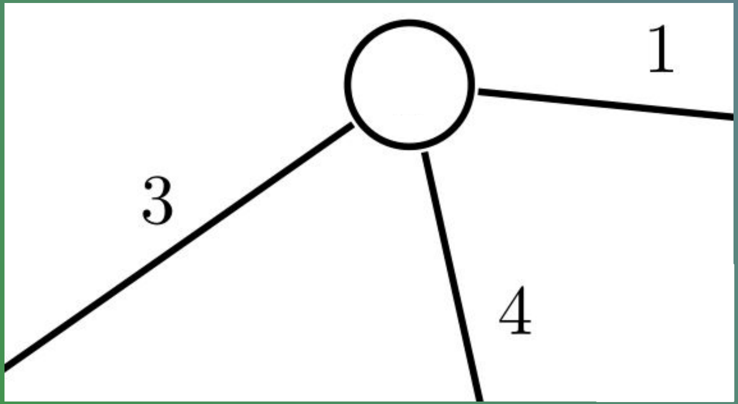
$$R_i^T(I,a) = \sum_{t=1}^{T} r_i^t(I,a)$$

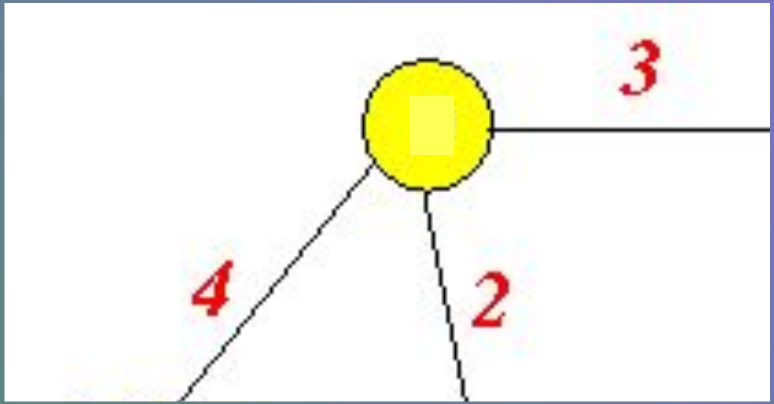
$$R_i^{T,+}(I,a) = \max(R_i^T(I,a), 0)$$

$$\sigma_i^{T+1}(I,a) = \begin{cases} \frac{R_i^{T,+}(I,a)}{\sum_{a \in A(I)} R_i^{T,+}(I,a)} & \text{if } \sum_{a \in A(I)} R_i^{T,+}(I,a) > 0 \\ \frac{1}{|A(I)|} & \text{otherwise.} \end{cases}$$

Practice

The numbers on each line are the regrets, compute the new probabilities for taking each action





Number of States in Poker

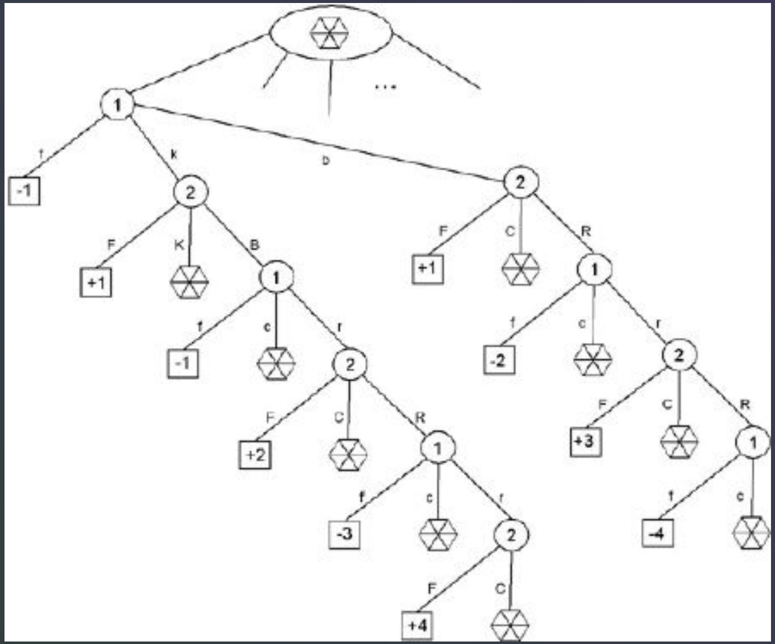
No-limit Texas Holdem ~10^75 game states

Limit Texas Holdem ~ 10^18 game states

For reference:

Game states in chess ~ 10^44

Particles in the universe ~ 10^80



Abstraction

- To handle the vast amount of game states that we must deal with in poker, we must abstract down the game state to a simpler representation
- Want abstraction to cover all relevant parts of game tree without over-simplifying game tree
- No rigorous way to define the 'best' possible abstractions, often trial and error/ knowledge of poker determines the best abstractions to make

Leduc Hold'em

Deck: 2 suits of Jack, Queen, King

Rounds: At the beginning of the game, each player receives one card and, after betting, one public card is revealed. Another round follows. At the end, the player with the best hand wins and receives a reward.

Rewards:

- Winner: +(Raised chips)/2
- Loser: -(Raised chips)/2

State representation in Leduc Hold'em

- State represented as dictionary containing game obs and legal actions
- Game obs is represented as 36 dim vector
- Legal actions is a list of numbers from 0-3

Index	Description
0 - 2	Current Player's Hand `0`: J, `1`: Q, `2`: K
3 - 5	Community Cards `3`: J, `4`: Q, `5`: K
6 - 20	Current Player's Chips `6`: *0 chips*, `7`: *1 chip*, ..., `20`: *14 chips*
21 - 35	Opponent's Chips `21`: *0 chips*, `22`: *1 chip*, ..., `35`: *14 chips*

Action ID	Action
0	Call
1	Raise
2	Fold
3	Check

Speeding up current implementation of CFR

- The current version of cfr that we wrote last week works, but converges slowly (around 1000 iterations of walk_trees)

- Today you will try to apply abstractions to the game tree of CFR and try to get the code to converge in fewer iterations, as well as implementing the regret-matching algorithm discussed on slide 2

Activity Here