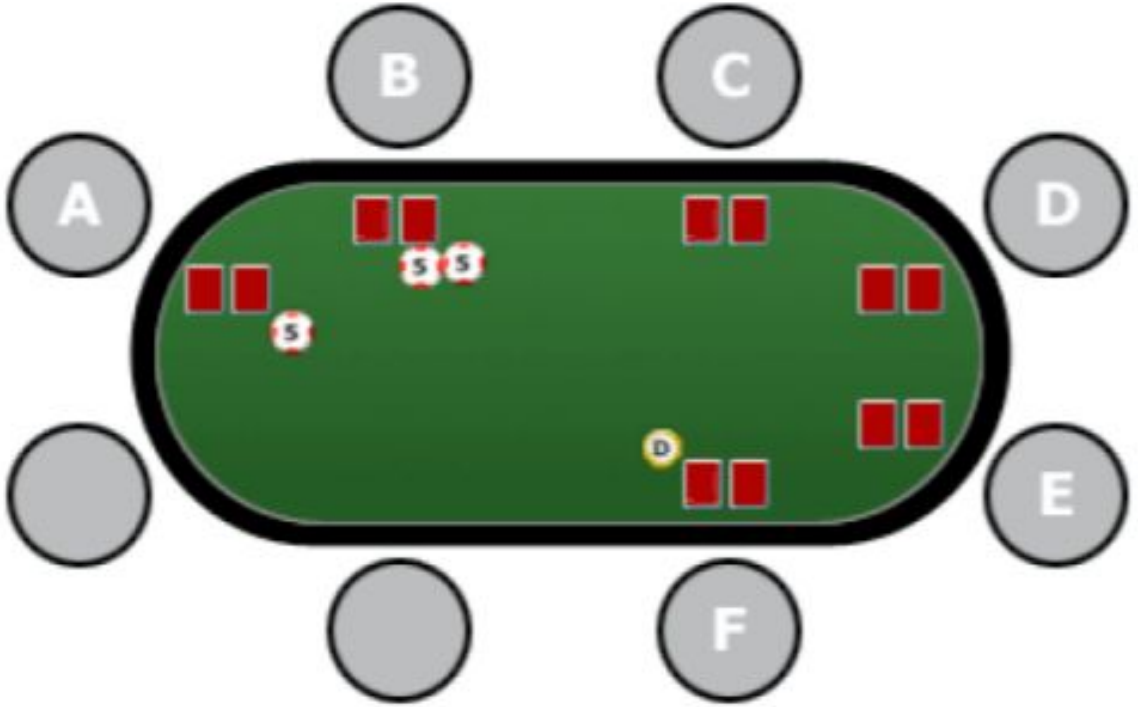
Texas Hold’Em and Reinforcement Learning

Speaker: Anh Tran

Background Information

No Limit Texas Hold’em (NLHE): Game rules

- 2 hole cards
- 5 community cards
- 4 rounds
  - Pre-Flop
  - Flop
  - Turn
  - River
- 6 actions
  - Check
  - Call
  - Bet
  - Raise
  - Fold
  - All-in



LHE vs NLHE

Limit Hold'em

- 2 private cards
- 5 public cards
- 4 rounds of betting
- 3 betting actions
  - Check/Call
  - Bet/Raise
  - Fold
- 10^14 game states

No Limit Hold'em (200 BB)

- 2 private cards
- 5 public cards
- 4 rounds of betting
- Up to 200 betting actions
  - Check/Call
  - Bet/Raise any size
  - Fold
- 10^170 game states
  - Go has 10^160 game states

Source: Nikolai Yakovenko – Columbia University Deep Learning Seminar

State of the Art

- DeepStack (2017): Scalable Approach to Win at Poker
- Libratus (2017): Masters Two-Player Texas Hold ’Em
- Pluribus (2019): Superhuman Poker-Playing Bot

Objective: The Ideal NLHE Agent

Using RL, the ideal NLHE agent should possess the ability to produce optimal betting strategy based on:

- **Hand strength/potential (*)**
  - Hole cards
  - Community cards
- Opponent Modelling
  - Hand strength prediction
  - Tight/loose play style
  - Passive/aggressive play style
  - Past strategy (bluffing?)
- Current game state
  - Agent’s stack
  - Opponents’ stacks
  - Agent’s position
  - Opponents’ positions
  - Pot size / pot odds

Hand Strength and Potential

$$EHS = HS \times (1 - NPOT) + (1 - HS) \times PPOT$$

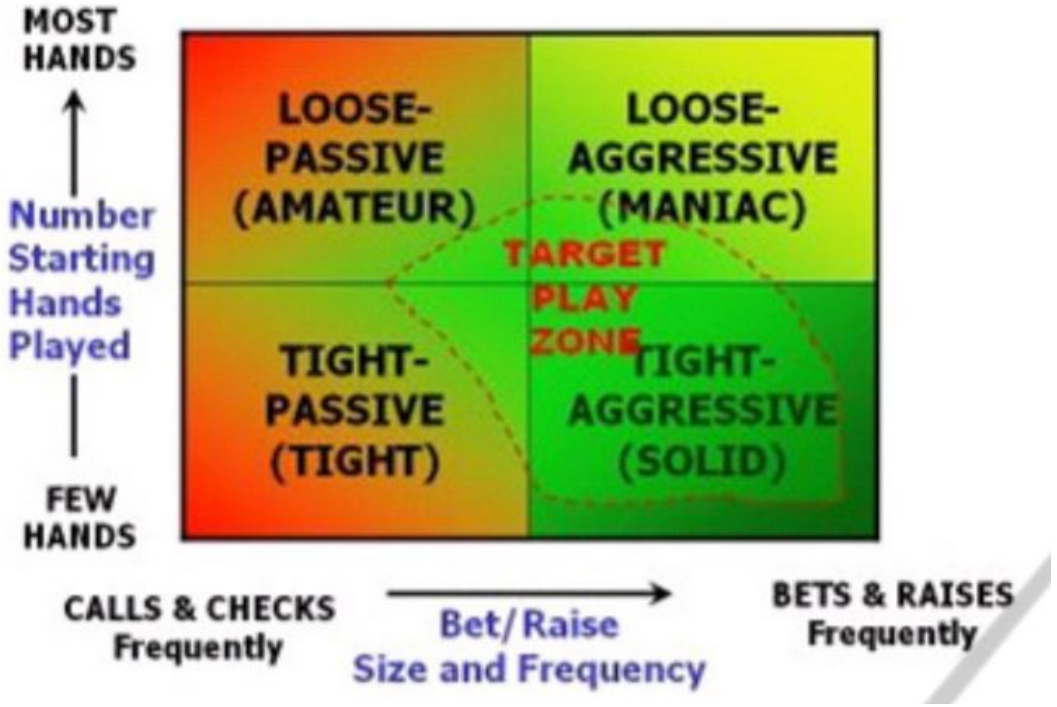
where:

- EHS is the Effective Hand Strength
- HS is the current Hand Strength (i.e. not taking into account potential to improve or deteriorate, depending on upcoming table cards
- $NPOT$ is the Negative POTential (i.e. the probability that our current hand, if the strongest, deteriorates and becomes a losing hand)
- $PPOT$ is the Positive POTential (i.e. the probability that our current hand, if losing, improves and becomes the winning hand)

Opponent Classification

	AF<=1	AF>1
%GP>=28	Loose Passive	Loose Aggressive
%GP<28	Tight Passive	Tight Aggressive

GP = % of hands participated
AF = Num Raises/Num Calls.



Playing Position and Pot odds, and Stack management

- A player’s position strongly affects his betting strategy
- Pot odds calculation in relation to hand strength is important in determining whether to participate in a hand.
- A player’s stack size also strongly affects his betting strategy

PokerBot

Hand Strength Reinforcement Learning

Angela Ramirez, Solomon Reinman, Dr. Narges Norouzi

IEEE, 2019

Approach

- THE-specific RL agent
  - Handstrength-dependent betting strategy (*)
- Actor-Critic Model
- Include specific domain knowledge in unison with reinforcement learning
- Sub-component solving

Data

- PyPokerEngine

  Simulations (cards, number of players, pot size)
  Built-in bots
  Hand strength calculation

- Example input at a given point in game:

  Hole cards: Q❤️, J❤️
  Table cards: 7♠️, 10♠️, 10❤️
  Num opponents: 3
  Stack: 70
  Pot: 135

Strategy: Action-reward table learning

	Fold	Call	Raise
Low	s_{11}	s_{12}	s_{13}
Mid	s_{21}	s_{22}	s_{23}
Mid-High	s_{31}	s_{32}	s_{33}
High	s_{41}	s_{42}	s_{43}

$$s_{ij}^{(t)} = s_{ij}^{(t-1)} + (v^{(t)} - v^{(t-1)}),$$

$$a = \max_i (s_j),$$

Low [0, 0.25]
Mid [0.25, 0.50]
Mid-High [0.50, 0.75]
High [0.75, 1.00]

Implementation: Three variations

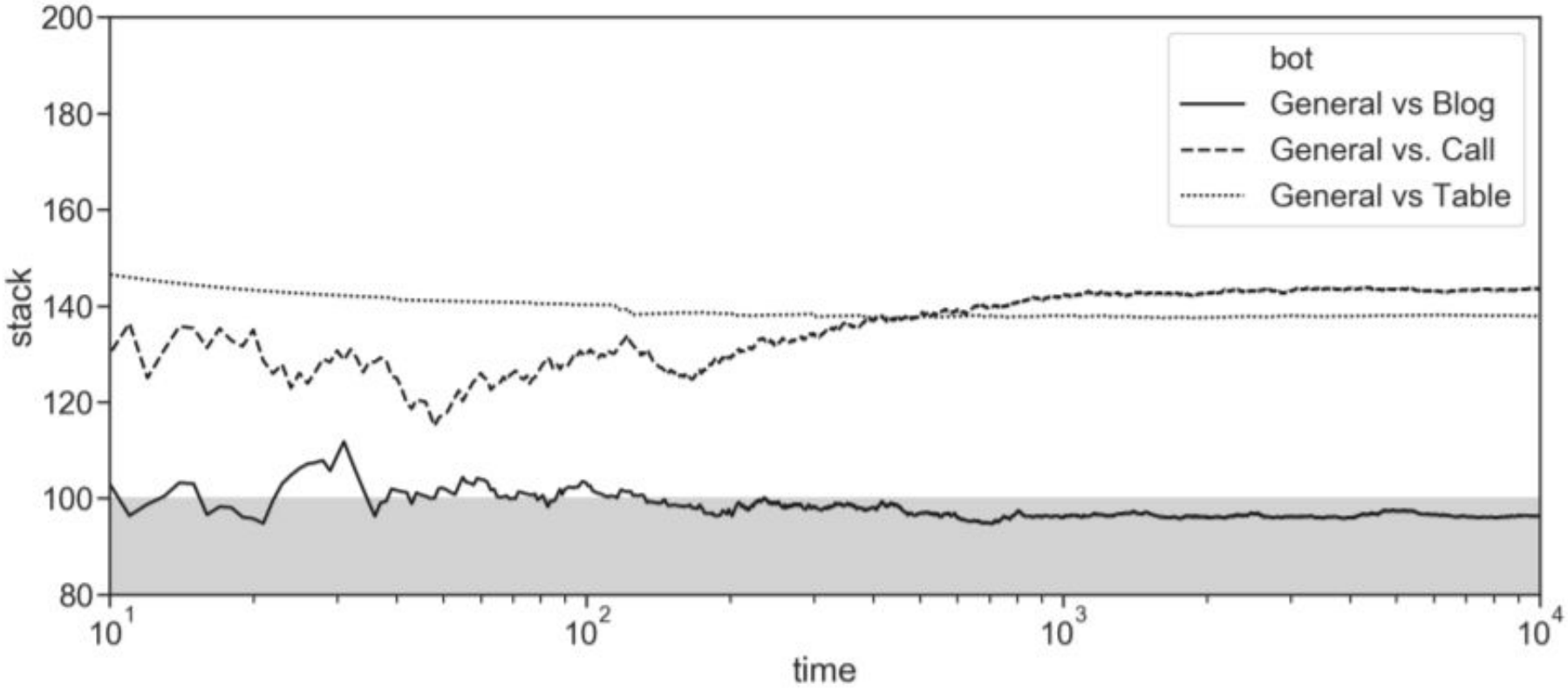
1) TableBot:
   Represents a larger input space.
   4 separate action-reward tables, specific to each game phase: preflop, flop, turn, and river
2) GeneralBot:
   1 action-reward table for all game phases

Play against pre-built PyPokerEngine bots:
- CallBot: calls the current bet.
- BloggerBot: uses PyPokerEngine's handEval() function to choose the action most likely to yield a positive result at each stage of the game.

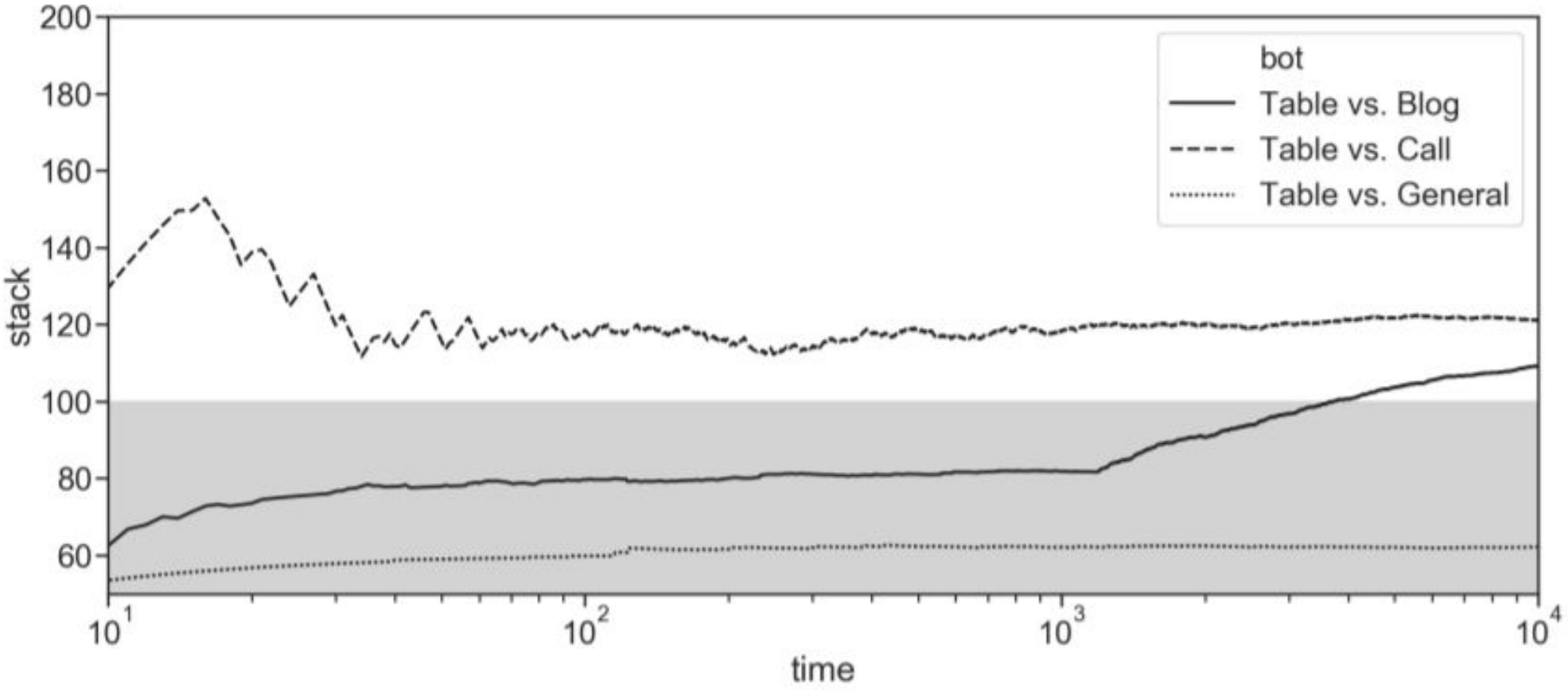
3) TableBotNNQ:
   Similar strategy with TableBot
   1NN of 3 layers
   - Input vector: 18 nodes
   - Dense layer: 48 nodes
   - Softmax output layer
   - Learning rates: 0.01, 0.1, 1, 2, 5, and 10
   - Optimization functions: gradient descent, Adam optimizer

Play against:
- CallBot
- BloggerBot
- TableBot
- GeneralBot

Results and Analysis: General Bot



Results and Analysis: Table Bot



Results and Analysis: TableBotNNQ

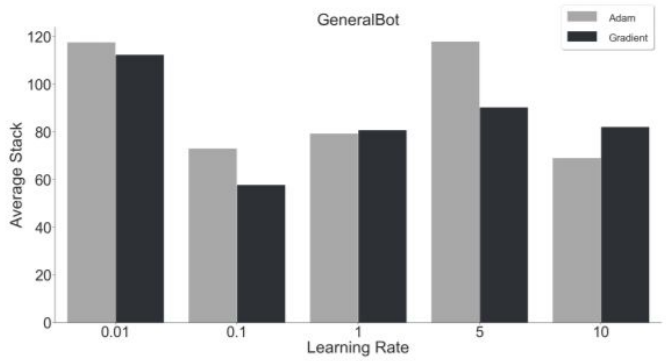


Fig. 3. *Average stack of TableBotNNQ vs. GeneralBot, with different learning rates and optimization function*

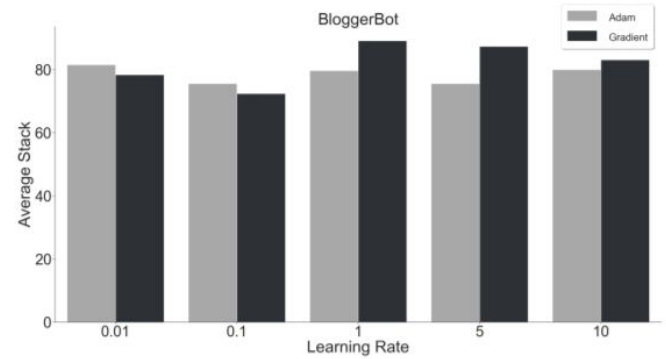


Fig. 4. *Average stack of TableBotNNQ vs. BloggerBot, with different learning rates and optimization function*

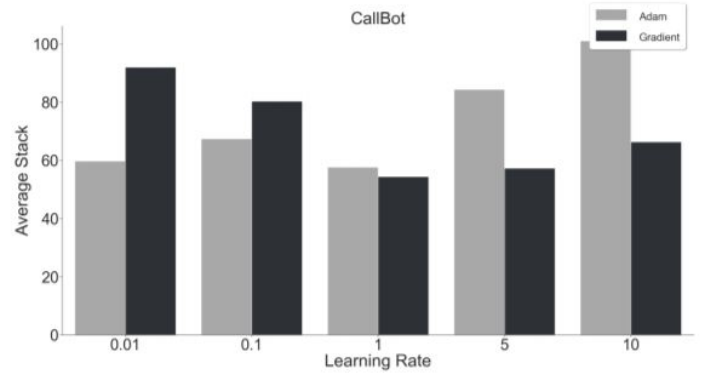


Fig. 5. *Average stack of TableBotNNQ vs. CallBot, with different learning rates and optimization function*

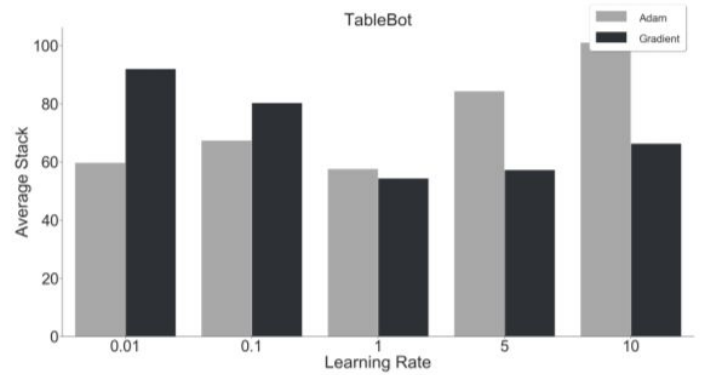
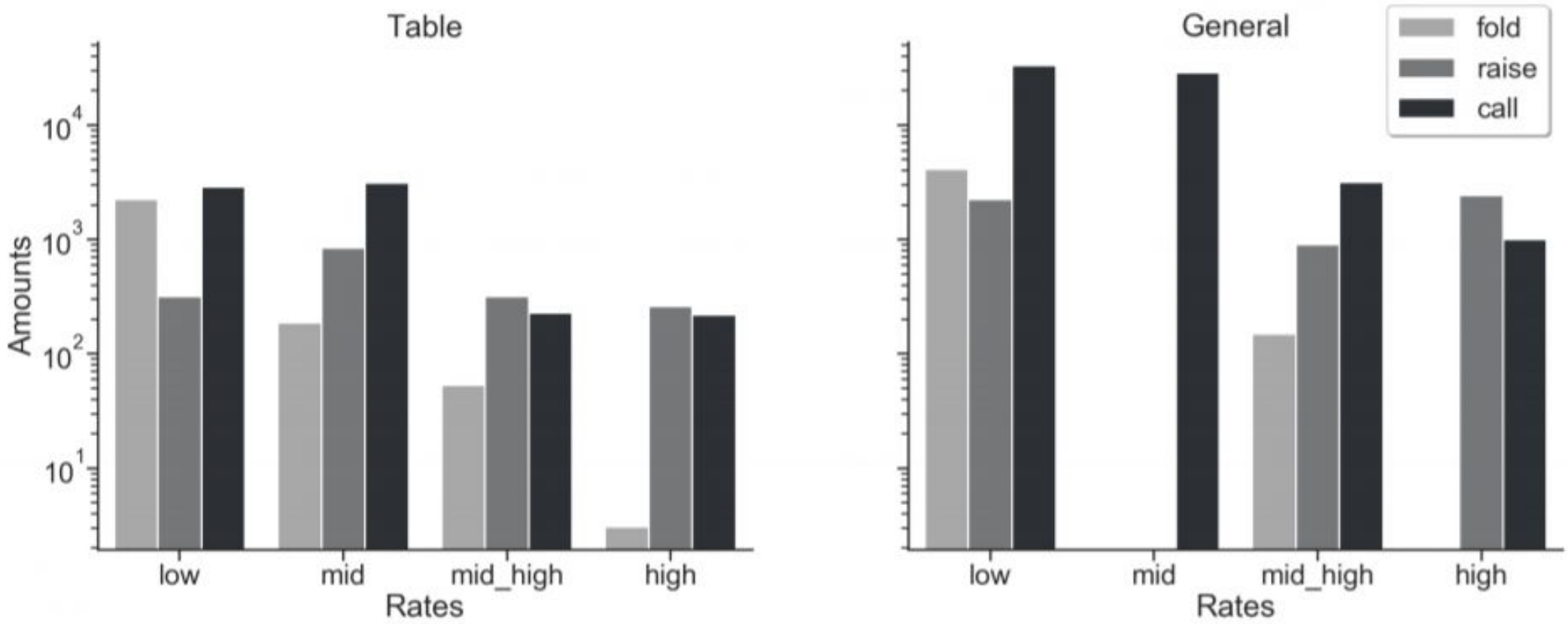


Fig. 6. *Average stack of TableBotNNQ vs. TableBot, with different learning rates and optimization function*

Results and Analysis: Action Preference



Results and Analysis: Implications, Conclusions & Future Work

- In a short amount of time, GeneralBot was able to learn more quickly than TableBot against CallBot and BloggerBot, achieving higher stack average
- Given more time, TableBot begins to approach a similar level of performance
- Both GeneralBot and TableBot show increased consistency in performance over time
- Agent with more input parameters is able to create a more diverse output space

⇒ Though simple, the approach has produced promising results, given time.

⇒ Neural network structure is useful in creating a hierarchical model that treats certain aspects of the input space (opponent action, cards available, etc.) independently and plays accordingly