PokerPal AI | Shivali Halabe, Andrew Dettmer, Ian Paul

project topic

goal

to build a poker-playing bot with to evaluate the effectiveness of novel reinforcement learning strategies in solving imperfect information games

extension

to solve real-world issues with transferable applications in handling multiple adversaries in environments where players have limited domain knowledge

problem approach

- poker is a **game of imperfect information**: players have no prior domain knowledge
- we want to **learn approximate Nash equilibria**— the best strategy for every player with respect to other players
- instead of common RL algos, we must **consider novel ones**

proposed RL strategies

Deep Q Network (DQN)
creates a neural network that approximates a reward value for a given state, from which a function to continuously update the network is learned

Neural Fictitious Self-Play (NFSP)
finds an approximation of the average best response to the other player's strategy using two neural networks, one of which is a DQN

algorithm overview: DQN

foundation

Q learning

- maintains a 2D matrix mapping states to a reward value for each action taken
- using the matrix, obtains a function which is used to update the matrix

problem

state space

- in games like poker, must enumerate an extremely large number of states
- makes matrix mapping impractical and computationally expensive

solution

neural networks

- replaces matrix with a neural network that takes a state as input and outputs a reward value for each action
- does not require the number of states to be predefined

pokerpal ai

algorithm overview: NFSP

foundation

Fictitious Self-Play

- averages several best response (BR) strategies into one in extensive form games (e.g. poker) using supervised learning
- is proven to converge to a Nash Equilibrium

problem

definitiveness

- chooses only one strategy for a player without considering game anomalies or alternatives
- does not compare reward of chosen action to others

solution

neural networks

- uses two neural networks (one DQN)
- predicts BR in DQN using BR history and in non-DQN using past experience
- chooses a network with a parameter

effective hypotheses

algorithm comparison

NFSP uses a DQN but incorporates another neural network and correction parameters, allowing it to stabilize choices and average over actions

conjecture

a bot trained with NFSP will yield better results in small Limit Texas Hold'em poker matches than would a bot trained with DQN

rlcard package

overview

a reinforcement learning toolkit for developing and training models for use in common card games (e.g. Texas Hold'em, Gin Rummy, Blackjack)

tools

a tested environment for running games and collecting performance data, pre-defined TensorFlow representations for DQN & NFSP agents

experimental methodology

independent variables

- reinforcement learning **algorithm** (NFSP, DQN)
- training **epochs**

dependent variables

- final average **training** payoff (reward)
- average **competition** reward

high-level code overview

agents

nfsp.py & dqn.py

- make environment; in each episode…
  - sample a policy for the episode
  - generate environment data
  - feed transitions into the agent memory and train
  - simulate game with random agents to evaluate performance

tournament

play.py

- load DQN and NFSP models pre-trained against random agents
- play agents against each other and and evaluate how models perform in a tournament
- log the average payoffs for each agent (player) in the tournament

high-level code overview (ctd.)

model class

pretrained.py

- wrap pre-trained models in the RLCard Model class
- instantiate a TensorFlow network
- restore model to network from raw save files

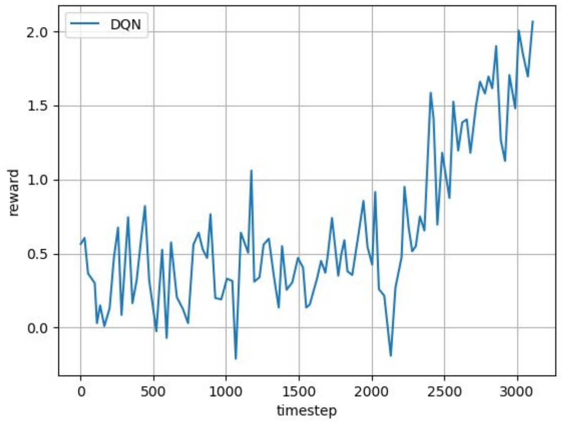
registration

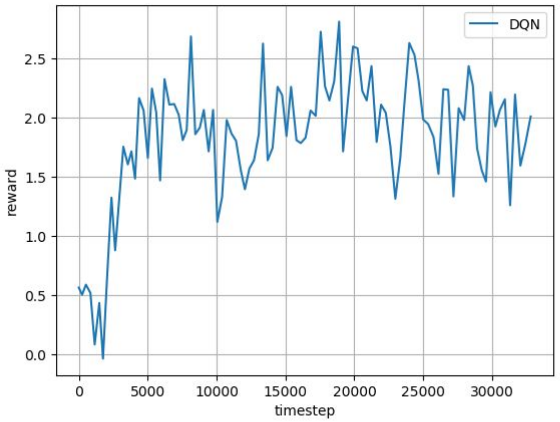
__init__.py

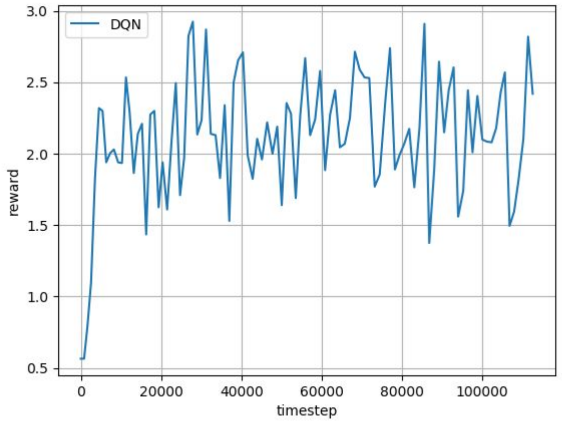
- register pre-trained or rule-based models defined by RLCard
- file is modified to register our pre-trained DQN and NFSP models within context of RLCard

training rates

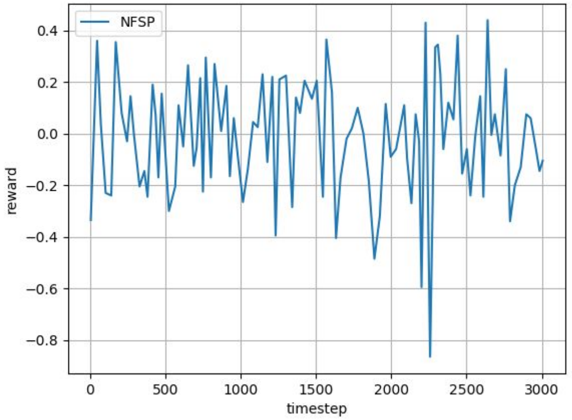
DQN

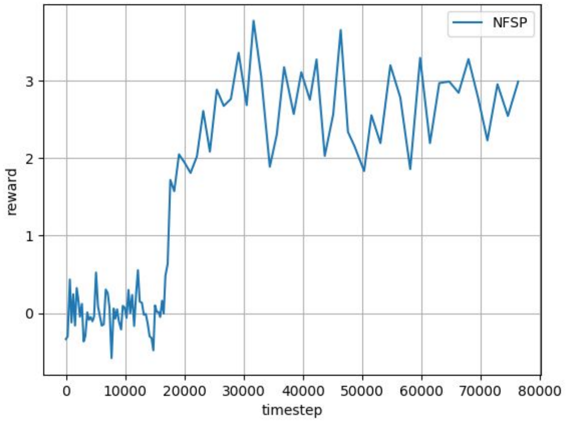


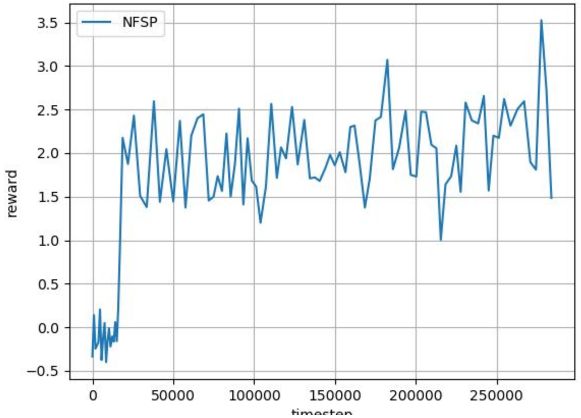




NFSP







tournament rewards

Model	DQN v1	DQN v2	DQN v3
NFSP v1	-1.8245	-2.0665	-2.1935
NFSP v2	0.8835	0.1195	-0.297
NFSP v3	0.447	-0.2215	0.284

strategy analysis

training rate

- both converge to similar average reward values against random agents
- DQN improves faster and has a higher variance in payoffs

competition results

- NFSP models perform better than similarly trained DQN models as training time increases
- reward values did not vary by large margins

overfitting

- limited training time (1000 epochs) was insufficient
- NFSP was more susceptible to overfitting; V3 performed unreliably

thank you!

any questions?