Applying Multi-Agent Reinforcement Learning to Candidate/Employer Job Matching and Salary Negotiations

Alexander H. Chen

Yale University

CSEC 491 Senior Project

December 15, 2022

Advisor: Dr. James Glenn

Abstract

In this project, we explore the use of reinforcement learning to train candidate and employer agents to choose actions that maximize their respective payoffs in the job search and salary negotiation process. To do this, we first used the PettingZoo open source library to create a multi-agent reinforcement learning environment that models this process. Breaking down the job search and salary negotiation process into steps, each candidate agent can choose to apply to a position, accept an offer, reject an offer, or negotiate an offer, and each employer agent can choose to reject an applicant, make an offer, accept a counter offer, or reject a counter offer. Each agent also has its own observations, which reflect an agent's knowledge of the overall game state. This environment allowed us to simulate the interactions between candidate and employer agents as they make decisions and negotiate salaries based on their objectives and rewards. Next, we used the Ray RLlib open source library to train reinforcement learning agents to optimize their decision-making in this environment. The candidate agents were trained to maximize their offer values, while the employer agents were trained to maximize the difference between candidate strength values and offer values. Our results show that these trained agents exhibit improved decision-making when played against agents with a random strategy, resulting in an increase in reward value. This suggests that reinforcement learning can be a powerful tool for modeling and optimizing the job search and salary negotiation process. This project opens an opportunity for further experimentation and modeling of the job matching process.

Applying Multi-Agent Reinforcement Learning to Candidate/Employer Job Matching and Salary Negotiations

Motivation and Introduction

Navigating the hiring process can be confusing and stressful for job seekers, with salary negotiation a crucial aspect of the process that can result in compensation increases in the thousands to tens of thousands of dollars. As a result, a large market for career education content and professional career services has emerged, offering advice on how to manage the application and negotiation processes. Further, salary negotiation services, including Levels and Candor, provide expert help for negotiating offers, leveraging teams of recruiters and coaches who work with individuals to craft a script to increase their offers.

On the employer side, tactics and data are designed to hire the right caliber of candidates while minimizing compensation costs. Negotiation tactics for hiring managers as well as a wealth of industry compensation data offered by third-party companies attempt to balance reducing compensation costs with successfully hiring top candidates.

For college graduates who have the least amount of experience with this process, being able to observe how different strategies for applying to positions, making counter offers, and deciding when to accept an offer was an interesting problem to tackle, enabled by the relatively new PettingZoo multi-agent reinforcement learning library developed in 2020.

Approach

Approaching this problem involved three separate stages: creating a game environment that modeled the hiring and salary negotiation process, training reinforcement learning agents to interact with the game environment, and finally simulating a game with the trained agents interacting with fixed strategy agents.

PettingZoo is a multi-agent reinforcement learning Python library that defines a standard model for multi-agent games and environment APIs that can be used to interact with the environment model (Terry et al., 2020). Observations and actions must conform to formats defined in the game environment. Actions for all agents are compiled and executed one step at a time, reducing complex interactions into step-wise actions that update observations and rewards for each agent. We designed the environment to create a simplified model of the recruitment process. Candidates and employers can only execute a single action with each iteration of the cycle (for example, candidates are not allowed to apply to more than one job at the same time).

Additionally, it is assumed that candidates cannot observe their own strength value, whereas employers can. Since employers are in charge of evaluating candidates, while a candidate would likely have a good sense of their strength in real life,

Training Agents with Ray RLlib

Ray RLlib is a Python framework for reinforcement learning (Liang et al., 2017). Compatible with PettingZoo environments, this framework manages the training process and offers wrappers and utilities to preprocess observation data, configure training parameters, and train multiple policies within the same multi-agent environment.

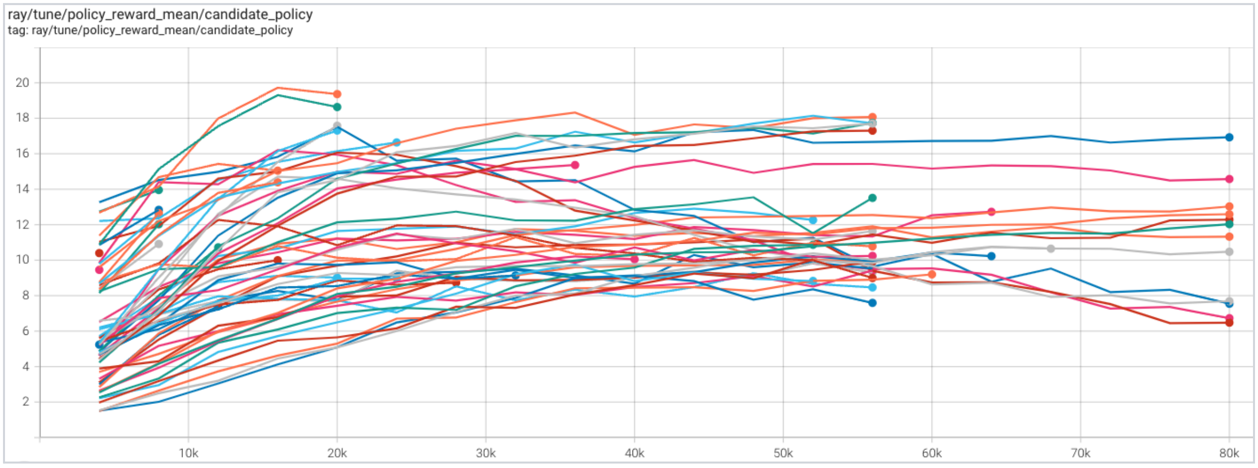
This project uses the proximal policy optimization (PPO) algorithm included in RLlib with a TensorFlow fully connected network model and a custom forward function that discards the action mask and flattens the observation into a binary NumPy array. This model was used to create two separate policies to train, one for candidate agents and one for employer agents.

27 different training sessions were conducted, with the environment parameters consisting of the Cartesian product of {(5 candidates, 5 employers), (5 candidates, 10

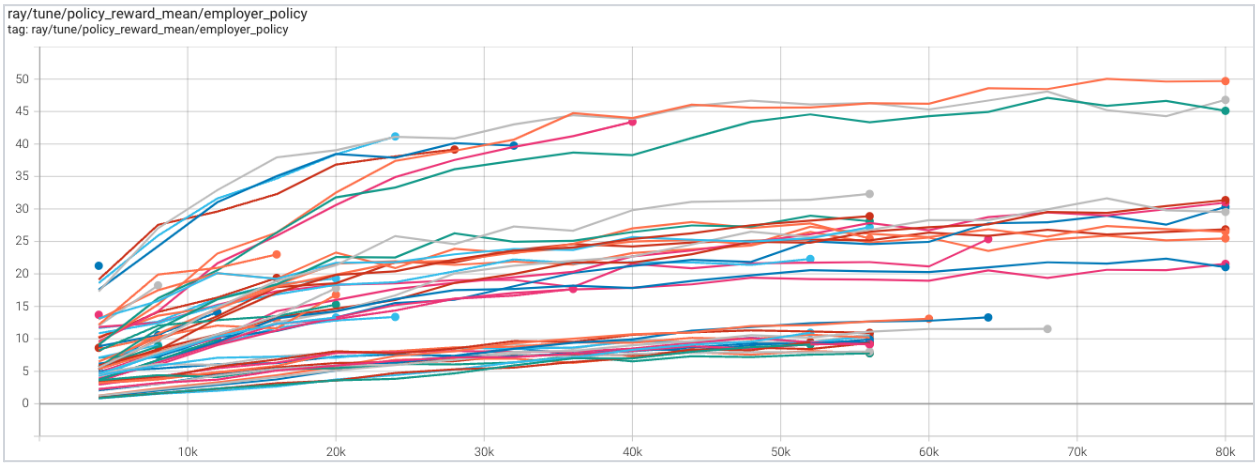
employers), (10 employers, 5 candidates)} X {50 max budget, 100 max budget, 200 max budget} X {10 max iterations, 20 max iterations, 40 max iterations}.

Results

Analyzing Training Metrics



Candidate Policy: Mean Reward vs Training Steps



Employer Policy: Mean Reward vs Training Steps

Each of these 27 training sessions ran 3 trials in parallel. For both the candidate policy and the employer policy, the PPO algorithm was able to improve the mean reward over the iterations. For candidates, their reward was calculated as their accepted offer value, subject to a time based discount rate. For employers, their reward was calculated as the difference between candidate strength and accepted offer value, also subject to a time based discount rate.

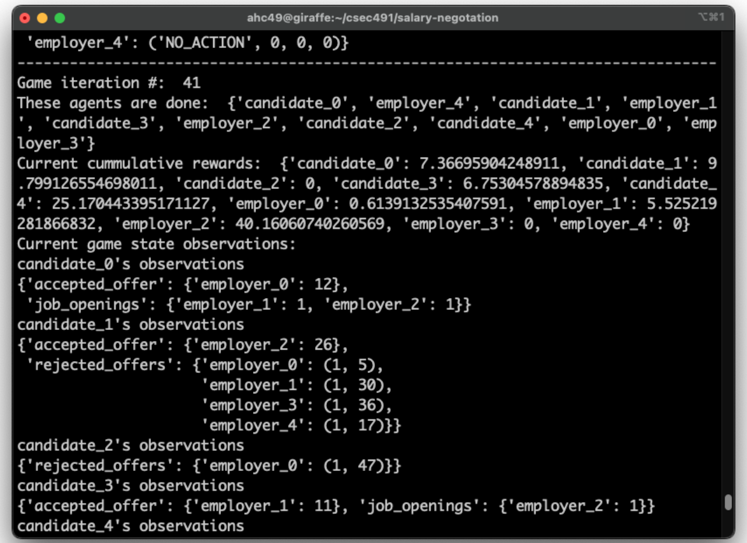
Although several trials halted early due to environment errors or other causes (at least one machine had issues with insufficient memory), a clear clustering of the reward lines emerged over the course of the training period.

Counterintuitively, for the candidate policy, the 3 clusters roughly correspond to maximum employer budgets of 100, 50, and 200 from top to bottom. This is likely due to the assumption made in the environment that any offers made by employers would “lock” that amount from the remaining budget. Thus, even though employers had higher budgets, candidates had lower mean rewards because employers could use that higher budget to make more offers to multiple candidates at once, increasing the likelihood of a candidate accepting a lower offer value earlier.

The employers are stratified into three clusters based on the dominating factor of the number of candidates compared to the number of employers. From top to bottom, the 3 distinct clusters represent 10 candidates, 5 employers; 5 candidates, 5 employers; and 5 candidates, 10 employers. This makes sense, as increased competition for a smaller pool of candidates would require higher offers to be made to get offer acceptances and thus lower employer rewards (and vice versa for decreased competition with a larger pool of candidates).

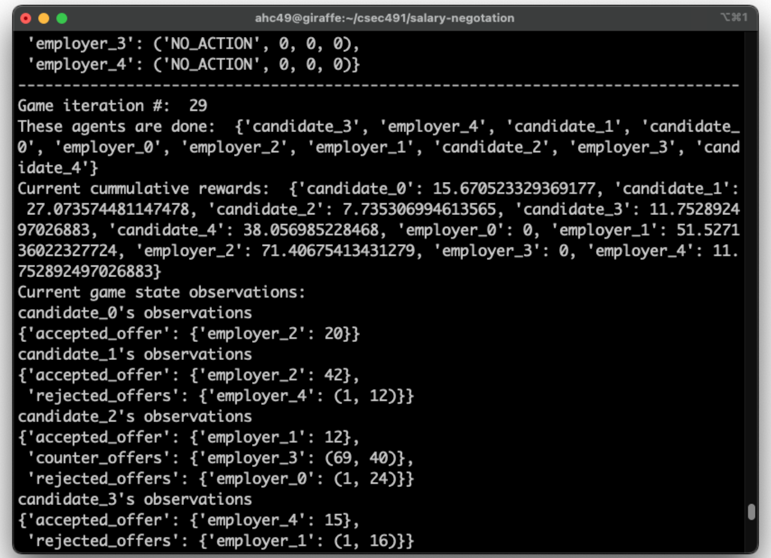
Simulating Play-Through with Trained Agents

A separate simulation program was used to compare the trained agents to a random strategy, and observations were made over the course of many simulated play-throughs.



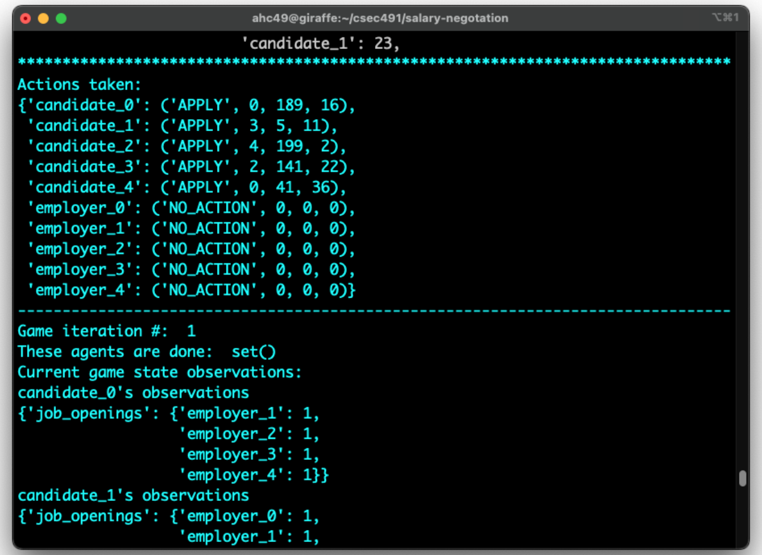
```
ahc49@giraffe:~/csec491/salary-negotation
 'employer_4': ('NO_ACTION', 0, 0, 0)}
--------------------------------------------------------------------------------
Game iteration #:  41
These agents are done:  {'candidate_0', 'employer_4', 'candidate_1', 'employer_1
', 'candidate_3', 'employer_2', 'candidate_2', 'candidate_4', 'employer_0', 'emp
loyer_3'}
Current cummulative rewards:  {'candidate_0': 7.36695904248911, 'candidate_1': 9
.799126554698011, 'candidate_2': 0, 'candidate_3': 6.75304578894835, 'candidate_
4': 25.170443395171127, 'employer_0': 0.6139132535407591, 'employer_1': 5.525219
281866832, 'employer_2': 40.16060740260569, 'employer_3': 0, 'employer_4': 0}
Current game state observations:
candidate_0's observations
{'accepted_offer': {'employer_0': 12},
 'job_openings': {'employer_1': 1, 'employer_2': 1}}
candidate_1's observations
{'accepted_offer': {'employer_2': 26},
 'rejected_offers': {'employer_0': (1, 5),
                     'employer_1': (1, 30),
                     'employer_3': (1, 36),
                     'employer_4': (1, 17)}}
candidate_2's observations
{'rejected_offers': {'employer_0': (1, 47)}}
candidate_3's observations
{'accepted_offer': {'employer_1': 11}, 'job_openings': {'employer_2': 1}}
candidate_4's observations
```

Using an example to illustrate, with 5 candidates, 5 employers, a maximum employer budget of 200 and a maximum number of iterations of 40, candidates and employers with random strategies ultimately result in the cumulative rewards shown in the above terminal output, and the highest candidate reward was 9.8.



```
ahc49@giraffe:~/csec491/salary-negotation
 'employer_3': ('NO_ACTION', 0, 0, 0),
 'employer_4': ('NO_ACTION', 0, 0, 0)}
--------------------------------------------------------------------------------
Game iteration #:  29
These agents are done:  {'candidate_3', 'employer_4', 'candidate_1', 'candidate_
0', 'employer_0', 'employer_2', 'employer_1', 'candidate_2', 'employer_3', 'cand
idate_4'}
Current cummulative rewards:  {'candidate_0': 15.670523329369177, 'candidate_1':
 27.073574481147478, 'candidate_2': 7.735306994613565, 'candidate_3': 11.7528924
97026883, 'candidate_4': 38.056985228468, 'employer_0': 0, 'employer_1': 51.5271
36022327724, 'employer_2': 71.40675413431279, 'employer_3': 0, 'employer_4': 11.
752892497026883}
Current game state observations:
candidate_0's observations
{'accepted_offer': {'employer_2': 20}}
candidate_1's observations
{'accepted_offer': {'employer_2': 42},
 'rejected_offers': {'employer_4': (1, 12)}}
candidate_2's observations
{'accepted_offer': {'employer_1': 12},
 'counter_offers': {'employer_3': (69, 40)},
 'rejected_offers': {'employer_0': (1, 24)}}
candidate_3's observations
{'accepted_offer': {'employer_4': 15},
 'rejected_offers': {'employer_1': (1, 16)}}
```

In this run-through, in fewer iterations the trained candidate agents were able to obtain significantly higher rewards when playing against a random strategy. Consistently between simulations, all of the trained candidate agents were able to obtain higher rewards, and no candidate agents had a reward of 0.



```
ahc49@giraffe:~/csec491/salary-negotation
                                'candidate_1': 23,
*******************************************************************************
Actions taken:
{'candidate_0': ('APPLY', 0, 189, 16),
 'candidate_1': ('APPLY', 3, 5, 11),
 'candidate_2': ('APPLY', 4, 199, 2),
 'candidate_3': ('APPLY', 2, 141, 22),
 'candidate_4': ('APPLY', 0, 41, 36),
 'employer_0': ('NO_ACTION', 0, 0, 0),
 'employer_1': ('NO_ACTION', 0, 0, 0),
 'employer_2': ('NO_ACTION', 0, 0, 0),
 'employer_3': ('NO_ACTION', 0, 0, 0),
 'employer_4': ('NO_ACTION', 0, 0, 0)}
-------------------------------------------------------------------------------
Game iteration #:  1
These agents are done:  set()
Current game state observations:
candidate_0's observations
{'job_openings': {'employer_1': 1,
                  'employer_2': 1,
                  'employer_3': 1,
                  'employer_4': 1}}
candidate_1's observations
{'job_openings': {'employer_0': 1,
                  'employer_1': 1,
```

All of the candidate agents started by applying to positions in the early cycles. If a high offer was made, the trained candidate agents would immediately accept, otherwise, they would negotiate for a higher offer value.

Next Steps

With the environment, reinforcement learning model training program, and simulation program created, there are many opportunities to extend this project to examine other factors of the job matching process. First, different policies could be trained for strong, medium, and weak candidates. Since in real life candidates have a rough sense of their strength, this would allow different algorithms to be trained to reflect stronger candidates executing different strategies compared to weaker candidates. Furthermore, the environment can be improved by allowing agents to execute more than one action at each step, increasing the action space and increasing the realism of the model.

Finally, a better visualization engine to interpret simulation results of trained agents playing against fixed strategies would be helpful to understand the strategies resulting from the reinforcement learning tuning.

References

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzalez, J., Goldberg, K., & Stoica, I. (2017). Ray RLLib: A Composable and Scalable Reinforcement Learning Library. CoRR, abs/1712.09381. http://arxiv.org/abs/1712.09381

Terry, J. K., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sulivan, R., Santos, L., Perez, R., Horsch, C., Dieffendahl, C., Williams, N. L., Lokesh, Y., Sullivan, R., & Ravi, P. (2020). PettingZoo: Gym for Multi-Agent Reinforcement Learning. ArXiv Preprint ArXiv:2009.14471.