The key insight is that we’re predicting enemy/user behavior, and only after that prediction is our agent going to make a choice Also we are training our agent to most efficiently learn the enemy/user behavior using its resources, (e.g.) for music recommendations, we don’t want to explore too much and give users really different songs or they will just stop using the app, similarly in poker you don’t want to bet too much just to understand how the enemy will play Not just learn player action probability distributions as that changes for every different player, but LEARNING TO LEARN player distribution as efficiently as possible The act of using our resources efficiently between learning (explore, although it must be said that as described by the paragraph right above, learning/generalizing efficiently (learning to learn/optimizing our agent to learn well) is extremely important) and using the knowledge we learned to bet (exploiting) must be also incorporated into our continuous learning process as the end goal is to win money, learning enemy behaviors is just a step towards that (although it is a major and very important one) Maybe need a prior as well (past enemy data collection for real world at least)

For NN2, we can update the weights of our rnn/lstm every round. The reward/penalty will be based on money gained if we had purely exploited (even if agent chose to not raise because of exploration bias, if it had ignored the exploration and purely exploited, would what would have happened align with our state space) Actually should we just reveal enemy hand strength to determine how good our NN2 is, it can definitely help. See how close our prediction was to the enemy hand strength. When should we reveal enemy hand, after every turn, cycle, round? Realize that in the real game, it won’t be able to see the cards. Also, will the model update its weights at game time? Or will the weights remain unchanged after training and the model (weights) will learn to process any combination of states. The state space x1 … xn should in the end contain the strength of each players hand as an important component. If this is the case we can maybe even train after each cycle or even turn. Maybe even instead of x1 …xn being the exact estimated strength, it should be a range or maybe even a Gaussian with mean and variance, so that NN1 can better predict. Almost like a VAE This is actually allat and we can view it as a Bayesian optimization! Trying to approximate the real world with a surrogate function! For NN1 we will also update weight every round. The reward/penalty will both be based on exploitation (money gained or lost in that round and also exploration (knowledge about opponent gained Possible actions in turn: fold, call, check, raise (x money), Game vs round (pot got vs cycle (all players played once) vs turn

Also maybe a good idea to feed into NN1 a deterministic relative hand strength (so we don’t have to bother making our NN train to guess deterministic information). This relative hand strength would be calculated by (e.g. we have 0.5 and there are 2 opponents, then we have 0.5*0.5 chance of being strongest hand, 0.5*0.5*2 chance of being middle, and 0.5*0.5 chance of being worst. Binomial distribution with win or lose bring outcomes per player. 3 main ideas:

* 1. Train NN2 separately so that it learns states which should represent enemy hand strength and our agent confidence about enemy hand strength (Gaussian). Actually these states should also represent “qualities of players” such as their aggressiveness, unpredictability, patience, etc. This general characteristic prediction of enemy players along with the guess of their current hand strength should allow our agent (NN1) to predict the actions the opponents will take, and learn how to play based on the predicted actions. This is important when we want to bluff (e.g our hand strength is 0.2, we predict enemy hand strength to be 0.8, but we learned that they are not agressive, so there is a chance that he might not go all in and we can exploit that characteristic) Also we can probably use multiple neural networks, one for each enemy player
  2. Train NN1 to act not only on how to exploit what it thinks the NN1 hand is (mean of the Gaussian), but also choose actions that will reduce the variance of the Gaussian to make better predictions later.
  3. Train NN1 to also choose actions that make itself unpredictable (sample from a Gaussian of actions, with the mean of the Gaussian being the “best move” and using a variance to choose around it)

We are developing an intelligent agent that can robustly adapt to changing environments and adversarial agents under incomplete information in a modified version of poker (Texas hold’em). With a focus on learning and operating under incomplete information, the modified version of poker removes much of the “quantitative” aspect of poker and focuses more on the “qualitative” (bluffing, betting patterns) instead. We believe that if the environment is too focused on the strictly “optimal way to play,” for example, determining hand strength based on public knowledge, the agent will simply learn a statistical distribution rather than meaningful, incomplete information that cannot be easily statistically represented. The paper will detail a reinforcement learning approach on learning and operating under incomplete information along with a comprehensive approach to gathering game data through game simulations and visualizations.

We have simulated a simplified version of poker by using RLCard, “a Toolkit for Reinforcement Learning in Card Games). RLCard is able to simulate a varteity of card games but critically, allows users to modify existing card games to match their required needs. Further discussion on game simulation will be discussed \_\_\_\_\_.

Each player will be assigned a random value between 0 and 1, without replacement [2], representing the strength of their hand (i.e., the percentage of other hands it beats) where 0 is the weakest and 1 is the strongest. Unlike poker, there are no public cards; the only information each player has is their own card. On each players’ turn, players will choose from four standard actions: check, call, raise, or fold. Small and big blinds (need citation), round termination, and winners will follow traditional poker regulations.

For simplicity, raises will be limited to fixed increments of 5, maxing out to (add number here). Additionally, to balance game simplicity/predictability and computational complexity/runtime, we will restrict the game to 3 players to begin with.

Within the modified version, the information of each agent can be classified as the following:

Complete Information

1. Player Card/Value

Incomplete Information

1. Adversarial Card/Value
2. Adversarial Betting Patterns
3. Adversarial Bluffing Patterns

With these areas of information, our agent will attempt to primarily exploit incomplete information to make informed decisions on the game state. (detail metrics, what a good model is here).

\*\*\*\*\*\*

The following outline key criteria of the model.

1. Prediction Before Decision-Making
   1. The agent must first predict enemy/user behavior before making any choices.
   2. Decision-making is based on these predictions rather than acting blindly.
2. Efficient Learning of Enemy/User Behavior
   1. The agent should use its resources effectively to learn behavior patterns.
   2. Example: In music recommendation, too much exploration (suggesting radically different songs) may drive users away.
   3. Similarly, in poker, excessive betting just to understand an opponent's strategy is inefficient.
3. Learning to Learn Player Distributions
   1. Instead of just learning a player's action probability distribution (which changes for different players), the agent must learn how to learn these distributions efficiently.
   2. This meta-learning approach ensures adaptability across different opponents.
4. Balancing Exploration vs. Exploitation
   1. Efficient resource allocation between learning (exploring) and using learned knowledge (exploiting) is crucial.
   2. While learning enemy behavior is important, the end goal is to win money, not just to gather knowledge.
   3. The learning process must continuously optimize for both exploration and exploitation.