Meta-Learning Friend or Foe Detection Via Q-Learning in Mixed-Agent Environments

Paul Reesman PAUL.R.REESMAN@GMAIL.COM

Abstract

Imperfect knowledge games are a highly studied problem in machine learning and game theory. One such game in this area is that of friend-or-foe detection. This specific type of game has many real life components that make it easily applicable to the real world. Two such scenarios are aircraft identification and network setup. The military uses friend-or-foe detection for aircraft identification but has constraints since human lives are at risk; one such constraint is that aircraft can be marked as friendly, however they cannot be marked as foe, since it is entirely possible that the aircraft is friendly but is experiencing a malfunction in its reply back to the station. In the network setup scenario, a machine is connecting to a network and must determine which machines are trustworthy and which are malicious. In this case, it is imperative that all malicious machines are marked as foe, unlike the aircraft identification scenario. This paper produces an algorithm extension of Q-Learning called Friend-or-Foe Detection Q-Learning. Unlike counterfactual regret minimization techniques, this algorithm creates a trust table from crowdsourcing all agents in the network and refines it over time by continuously crowdsourcing. This algorithm is shown to work well when compared to counterfactual regret minimization algorithms such as DeepRole. The Friend-or-Foe Detection Q-Learning has a quick convergence speed with a similar reward loss and should be considered as a simpler and quicker alternative for calculating imperfect knowledge games.

1. Introduction

Friend or Foe detection is a difficult and highly constrained problem, yet completely integrated in many real life situations. Friend identification requires an agent to gain full trust from another; foe detection happens when an agent loses trust past a predetermined threshold. While foes attempt to gain the agent's trust, it is easier to mark an entity as a friend than it is to mark them as a foe. This is true because when an agent responds correctly it is straightforward to mark them as a friend. Conversely, if an agent does not respond correctly there might be several reasons for it, other than the agent being a foe. For example, there might be situations where an agent must determine whether other individuals it encounters in its environment are friendly or adversarial to its goal. One interesting scenario is the idea that these other individuals are hiding their true identity from one another to achieve their own goals and only reveal their identity to another individual whom they have come to trust. A modern framework called CounterFactual Regret Minimization (CFRM) is an attempt to turn this scenario into a k-player zero-sum partially observable game (Serrino, Kleiman-Weiner, Parkes, & Tenenbaum, 2019) which we will be using as a basis for state-of-the-art comparisons against a combined technique of Q-Learning and Meta-Learning.

While CFRM has proven to be a successful solution to this problem, it has efficiency issues (Li, Hu, Ge, Jiang, Qi, & Song, 2018; Zinkevich, Johanson, Bowling, & Piccione,

2008) which we aim to improve upon. Q-Learning is a reinforcement learning algorithm that assigns qualities to discrete states in the domain. Friend-or-Foe Q-Learning (FFQ) (Littman, 2001) is an algorithm that extends the basic Q-Learning technique in which an optimal policy is determined by maximizing friendly agent policies and minimizing foe policies. One issue with this provided algorithm is that the friend and foe agents are already known to the agent learner. This paper extends Q-Learning with this FFQ inspiration to introduce Friend-or-Foe Detection Q-Learning (FFDQ). The difference between FFQ and FFDQ is that FFDQ detects the identities of the agents in the environment, while FFQ utilizes a policy based on already knowing the identities of said agents. Furthermore, the algorithm Reptile (Nichol & Schulman, 2018) will be used to help find optimal hyper-parameters for FFDQ. We hypothesize that FFDQ will converge to an average solution that is not significantly worse than state-of-the-art CFRM framework and will train in fewer iterations for the friend-or-foe detection experiment we run.

This paper will be structured as follows. In section 2 we will go over the background and give more insight into why this problem needs more research, while also informing what research has already been done in this area. Following that, in section 3, this paper will go over relevant literature to provide the reader with knowledge needed in understanding the research performed in this paper. In section 4 the algorithm FFDQ is presented. Reviewing the experimental design and setup will come next in section 5, as we dive into exactly what experiments were run. Next we will go over the results of the experiments and discuss the meaning behind them in section 6. Finally, in section 7, we will wrap up the paper itself with ideas for future work and a conclusion.

2. Background

Friend or Foe detection is a crucial real world problem that has many faces. The military has created a system called Identification Friend-or-Foe (IFF) (Pollack & Ranganathan, 2018), in which an agent pings a detected entity and expects a response. This response helps determine the identification of the detected entity. One constraint of IFF is that an entity can only be marked as friendly, it cannot be marked as a foe. This is because noise can be introduced in the system (such as faulty transponders on aircraft) preventing entities from responding correctly. One such scenario to get around this constraint comes through network security when an agent connects to a new network (Nguyen, Alpcan, & Basar, 2009). The earlier constraint from IFF can be removed, since instead of relying solely on the response of an entity, we can now accumulate responses from several agents to determine identification. Imagine a situation where there exists a network of k machines $M = \{m_1, ...M_i, ...m_k\}$ where $k-1$ machines are already connected together in a complete graph. Within these connected machines there are b nefarious machines $M_n = \{M_1^n, ...M_b^n\}$ s.t. $M_n \subset M$ that wish to gain sole trust over any newcomer to the network. Within this network, each machine knows who is truly trustworthy and who is not. Finally we add the last machine which is our agent. It is now this agent's job to identify the b nefarious machines without knowing the true count of b in order to gain accurate fidelity within the network. This can be accomplished by choosing a single machine to scrutinize and then asking a subset of the rest of the agents how trustworthy this scrutinized agent is.

A second scenario extends itself from the previous one mentioned. Instead of a complete graph, an arbitrary amount of connections between the existing $k-1$ nodes are removed as long as $\frac{(k-1)(k-2)}{8}$ connections remain. This constraint exists to limit the potential negative influence the nefarious nodes have on the agent learner. This entire set of nodes M represent the only nodes that the k^{th} node can directly see, so when the k^{th} node is added back into the network, it gains a connection to every other node in the network. However, not every node this agent can see can see each other. Thus, individual nodes might not have context to present to the agent about other nodes in the network.

There have been a few attempts at solving this type of problem such as DeepRole (Serrino et al., 2019) and Q-Learning (Kash, Sullins, & Hofmann, 2019). Both instances take a CFRM approach. The latter extends Q-Learning with CFRM to help with the observable state inherent with this problem. Additionally, this bridges the reinforcement learning — supervised learning bridge as the ultimate solution can be found and then directly used in an error function to update hyper-parameters for the meta-learner. Reinforcement learning can be a good approach to solve these middle-of-the-road problems instead of using supervised learning (Wiering, van Hasselt, Pietersma, & Schomaker, 2011).

Solving this type of problem for an entire set of instances is an important area of research. Having a network security system be able to learn a generalized form of friend or foe detection could make these systems much more verbose and provide better network security.

3. Literature Review

This paper provides research into the areas of reinforcement learning, counterfactual regret minimization, and meta-learning. Within section 3.1 reinforcement learning is reviewed with topics of Markov Decision Processes, Partially Observable Markov Decision Processes, and Q-Learning. The following section, 3.2, covers counterfactual regret minimization and a specific algorithm that is used for friend-or-foe detection called DeepRole. In the final section, 3.3, meta-learning is reviewed, as well as three popular algorithms including the one used in this research, Reptile.

3.1 Reinforcement Learning

Reinforcement learning is an interesting area of machine learning that does not deal with static problems. Both supervised and unsupervised learning — although their techniques are quite different — deal with unchanging data sets. Comparatively, reinforcement learning is interactive (Syed, 2010).

The agent learner in a reinforcement learning problem is given a goal and must explore its environment in order to accomplish it. The necessity for exploration stems from the constraint that the agent learner has no prior knowledge about the environment at all. This agent only has knowledge of what actions it can take to interact with the environment and a sensor to observe the change in the environment from the chosen action. A simple way to define reinforcement learning is through Markov Decision Processes (MDPs).

3.1.1 Markov Decision Processes

An MDP can be defined by the tuple (S, A, τ, R, γ) where $S = \{s_1, ...s_i, ...s_n\}$ is the set of all states, $A = \{a_1, ...a_i, ...a_m\}$ is the set of all actions the agent can take, and $\tau : S \times A \rightarrow P(s' \in S)$ is the probability transition function from state s to s' s.t. $s, s' \in S$ after taking action $a \in A$. This transition function may not always be deterministic, in such stochastic cases it defines the probability to end up in state s' given both s and a. R is the reward function which is the agent observing the environment after action a and determining how the action progressed the agent toward its goal. Usually this reward function is expected to be known (Syed, 2010), thus the reward is directly observable from the environment and no interpretation is necessary. γ is the discount factor which helps with infinite horizon problems — problems not constrained by time. A discount factor strictly less than one helps these problems eventually converge to a finite sum of rewards. This turns reinforcement learning into a maximization problem, such that the agent is trying to maximize its reward in its attempt to achieve a goal state $s_G \in S$. The agent attempts to learn a policy to maximize its reward.

Many real world scenarios do not allow the agent to fully know the entire state space S, which leads to partially observable problems. There are several ways to deal with these types of reinforcement learning problems, such as Partially Observable Markov Decision Processes (POMDPs) (Saad, 2010) and Counterfactual Regret Minimization (CFRM) (Jin, Keutzer, & Levine, 2017).

3.1.2 Partially Observable Markov Decision Processes

There are many cases in which the entire state space cannot be known for multiple reasons, such as the state space itself is too large — like the state space of chess or infinite state spaces. Alternatively, the agent itself cannot fully observe the state space — like a mountain obscuring the view of what lies beyond. There are several ways of solving these POMDPs, but one way is to treat the entire state space as observable and turn the POMDP into a regular MDP (Cheng, 1988). From here the agent begins creating beliefs as to where it believes it is after observing environment responses that follow an action.

This belief is built upon observations from the environment. As opposed to a regular MDP where the agent knows exactly where it is, an agent in a POMDP cannot trust the exact state it is in and relies on actual observations from the environment after taking an action from a particular state. From this, we can expand the earlier definition of MDPs to the 7-tuple $(S, A, \Omega, \tau, O, R, \gamma)$. The additions of Ω and O are these observations. Ω is the actual set of observations the agent can receive from the environment and O is the observation function such that $\omega \in \Omega, S \times A \rightarrow P(\omega)$. This function defines the probability of observing a particular observation from the environment after taking a specific state-action pair.

3.1.3 Q Learning

Q-Learning (Watkins, 1989) is a popular reinforcement learning algorithm that teaches an agent to behave optimally in constrained MDP environments. This method of learning is known as "model-free," as in it does not require a model of the environment. Being an iterative algorithm, Q-Learning enumerates the expected reward from each state and has an agent learn what actions to take at each state based on the expected future reward from taking the optimal action (Watkins & Dayan, 1992).

The reason this algorithm is called Q-Learning is because it defines a quality function Q s.t. $Q : S \times A \to \mathbb{R}$. This algorithm is a value-iteration algorithm that iterates over every state and determines the agent's expected reward for taking the optimal action from that state. The algorithm updates the quality of each state until no more changes have been made and the convergence has been found. Q-Learning is based around the Bellman equation and thus the quality update function takes the form:

$$Q^{new}(s_t, a_t) = Q^{old}(s_t, a_t) + \alpha * (r_t + \gamma * \max_a Q(s_{t+1}, a) - Q^{old}(s_t, a_t)) \tag{1}$$

Equation 1 is a temporal difference equation that updates the quality of the current state. It takes the current reward r_t and adds it to the discounted γ optimal next state subtracted by the current value of the current state's quality. We add all of that to the value of the current state's quality. This produces a backward propagation of quality values. States adjacent to the goal state are updated on the first iteration, then the states adjacent to them, and so on. Eventually the entire state space is modeled and the quality of every state converges to a single number.

Q-Learning has been extended several times around this friend or foe topic (Littman, 2001; Kash et al., 2019). FFQ knows that it is in a mixed environment of friendly and foe entities. However, the detection of each entity has already happened and the agent learner only needs to learn how to optimize its policy. This policy optimization comes through the form of maximizing the policies of friendly entities, while minimizing the policies of the foe entities. The other extension is directly minimizing regret via Q-Learning.

3.2 CounterFactual Regret Minimization

Regret is an online learning concept that has inspired many powerful learning algorithms (Marchiori & Warglien, 2008; Kasperski & Zielinski, 2006; Hazan & Kale, 2011; Zhang & Ji, 2019). Regret is the idea of how far off an agent is from the optimal policy. Essentially, the agent looks back at a given time step and criticizes its action and reward and determines its regret of not taking another, possibly better, action at that exact moment. The idea of CFRM extends this regret concept by decomposing regret into individual additive terms that can be individually minimized or maximized (Zinkevich et al., 2008).

CFRM is a great way to solve imperfect knowledge games but it has a restraint that the states, actions, and time steps must be discrete. Thus, CFRM is often represented in a tabular form. However, this is a very large constraint that is lessened through the use of neural networks (Li et al., 2018; Steinberger, 2019).

3.2.1 Deep CounterFactual Regret Minimization

The paradigm shift of CFRM from discrete to continuous (with the use of neural networks), has prompted researchers to make further use of deep learning. Another reason to use deep learning for CFRM is for function approximation (Brown, Lerer, Gross, & Sandholm, 2018). Many games, like chess and go, have too many states to enumerate in practice, thus the game cannot be fully tabularized. DeepRole (Serrino et al., 2019) is a good example of the function approximation algorithm. DeepRole used a deep neural network to semi-tabularize the state system of their example game, The Resistance: Avalon, to detect which players were friendly to their objective and which were adversarial. The game, although simple, has too many states to enumerate, so DeepRole approximated states and transitions to be able to solve this game. One downside of CFRM in general is it takes a long time to converge — the deeper neural networks become, the more it compounds this problem.

3.3 Meta-Learning

Machine learning algorithms heavily rely on parameters to be tuned correctly for the algorithm to function ideally. The purpose of meta-learning is to learn these parameters, known as hyper-parameters, instead of tuning them by hand, which can be a long and tedious process (Cicirello & Smith, 2000; Schweighofer & Doya, 2003). Meta-learning is essentially "learning how to learn" (Zhou, Wu, & Li, 2018); it learns the appropriate hyper-parameters for the given model to perform. A popular meta-learning algorithm is Model Agnostic Meta-Learning (MAML) (Finn, Abbeel, & Levine, 2017) which takes an agnostic view of the model it is trying to learn and iteratively marches the learned hyper-parameters over all tasks toward near optimality, which allows for quick adjustments to any task to find the optimal parameters. This method relies heavily on direct rewards from the environment, which might not always be available. An extension of MAML called No Reward Meta-Learning (Yang, Caluwaerts, Iscen, Tan, & Finn, 2019) was created just for this purpose. However both approaches are burdensome to implement. Thus a simpler approach was created that performs just as well called Reptile (Nichol & Schulman, 2018).

3.3.1 Reptile

Reptile is a meta-learning algorithm that takes a joint learning approach to learning (Nichol, Achiam, & Schulman, 2018). This means that the algorithm is presented a task list and tries to learn hyper-parameters that best fit each task jointly. Thus, these parameters are not necessarily optimal for any task, but are at least within an acceptable threshold of each task's true optimal parameters. Reptile does this by learning the hyper-parameters for each task at the same time. A task is randomly selected from the task distribution, then trained and tested on the global hyper-parameters. A stochastic gradient descent (SGD) is used to find the error of the training/testing cycle to update the global hyper-parameters. After enough epochs, the algorithm ends with global parameters tuned to the joint distribution of tasks.

4. Friend-or-Foe Detection Q-Learning

As an extension to Q-Learning, FFDQ uses qualities to determine how good a state is. However, instead of taking a direct action every iteration, the agent learner selects a random entity to scrutinize and subsequently selects a random set of other entities to query about the scrutinized entity. This stochastic nature of selecting the jury is an attempt to find agents to trust. The agent learner keeps track of a weight for each agent. This weight is used in two ways. First, it is what is updated when an agent is being scrutinized and its value is incremented by the sum of the jury's findings. Second, the weight is used to

Algorithm 1 Friend-or-Foe Detection Q-Learning

```
1: procedure FFDQ($A, \epsilon$)
2:     $Z \leftarrow G(Z)$                                  ▷ The globally learned trust store G
3:     while not done do
4:         $s \leftarrow a \sim A$ s.t. $a$ not marked      ▷ Randomly select unmarked agent to scrutinize
5:         $J \leftarrow \{a_i, ...a_k\} \sim A - s$        ▷ Randomly select a random sized set as a jury for $s$
6:         $Z_s \leftarrow T(s, J)$                         ▷ Retrieve weighted trust sum from equation 2
7:         if $Z_s > \epsilon$ then
8:             $Z_s \leftarrow$ marked friendly
9:         else if $Z_s < -\epsilon$ then
10:            $Z_s \leftarrow$ marked foe
11:        end if
12:        if $Z_s$ marked then
13:            $G(Z_s) \leftarrow B(Z_s)$                   ▷ Equation 4 updates global trust store
14:        end if
15:        if $\forall a \in A$ marked then
16:            done
17:        end if
18:    end while
19: end procedure
```

determine how much weight to apply to each of the jury members. For example, if there are two members of the jury and so far one seems to be more trustworthy than the other, the agent learner will put more value into the trustworthy agent. This ends up being:

$$T(s, J) : \sum_{i}^{J} S(\frac{i_w}{t}) * Q_i(s) \tag{2}$$

The weight of each juror w is divided by the time step t to devalue trust over time and normalize the input to the sigmoid function S. The sigmoid function is used to devalue untrustworthy agents while keeping the magnitude of the weight between zero and one. This sigmoid function is defined with equation 3. The quality function Q_i of agent i returns the agent's thoughts of the scrutinized agent s.

$$S(x) : \frac{1.0}{1.0 + e^{-x}} \tag{3}$$

FFDQ is shown via algorithm 1. For every epoch iteration, FFDQ is called and the local trust store Z is updated from the previous epoch's update to the global trust store G. After selecting the scrutinized agent and the jury, the agent learner uses the global trust of each juror as the weight for each juror in the jury as we update the local trust for the scrutinized agent. Once the local trust passes a threshold of ϵ or $-\epsilon$, then it becomes marked. The global trust store is updated and the scrutinized agent is no longer allowed to be scrutinized again for this epoch. Once all agents have been marked, then the epoch finishes. This algorithm utilizes an alternative form of the Bellman equation shown in equation 1, this equation becomes:

$$Q^{new}(s,a) = Q^{old}(s,a) + \alpha * (r(s,a,t) + \gamma * Z_s - Q^{old}(s,a)) \tag{4}$$

Equation 4 shows a few differences from equation 1. First, the quality is not derived from the exact state but from the agent being scrutinized, s, and neither s nor the action taken are indexed by time. Second, the reward function derives the reward from s, the action, and the time. Finally, the largest difference is that instead of using the best quality of the next state, we use the derived local trust gathered from enough juries scrutinizing s.

5. Experimental Evaluation

The core question this paper asks is: how well can Q-Learning perform while extended by exchanging state-quality for agent-quality, when compared to state-of-the-art techniques such as CFRM? Furthermore, a secondary question is: how well can meta-learning be used to generalize learning a set of tasks? In order to evaluate this second question properly, we created two similar tasks for our agents to learn. These tasks are described in section 2; both will be learned by the agents and will also be used as generalization tests. FFDQ will learn task 1 and another instance will learn task 2. We will use the already existing friend-or-foe detection deep learning CFRM algorithm, DeepRole, to do the same to compare performance results. Reptile will then use FFDQ to learn generalized hyper-parameters for both tasks.

During instance set up, each agent is aware of which other agents it can see and whether or not they are friendly to that agent's goal. For each agent in the agent pool, a quality is assigned to it for each other agent. If the agent is friendly to the current agent's cause, it is assigned a quality of $(0.0, 1.0]$, while foe agents are assigned a quality of $[-1.0, 0.0)$. This allows agents to have an amount of uncertainty to their environment which the agent learner needs to reason about.

FFDQ is set up using parameters that seemed to do well during initial testing. It is given $\alpha = 0.6, \gamma = 1.5, \epsilon = 5.0$ with an epoch count of 1. Reptile will be used to help find the average best hyper-parameters over all tasks given. It is expected that over a single task, FFDQ will outperform its meta-learned self on the specific task it was trained on, but will falter on the task it was not trained on. This will be shown via reward loss as a comparison between both instances of FFDQ and the meta-learned instance. From initial testing, Reptile seems to converge to good hyper-parameters through the parameters of $\alpha = 1.5, \epsilon = 0.5$.

An early observation during initial testing found a trivial sense to this problem for FFDQ. At first, all agents gave a deterministic response when queried about a scrutinized agent. This allowed FFDQ to converge in minimal or close to minimal time steps at 100% accuracy. A stochastic noise element based on simulated annealing was added to the response of friendly agents. This allowed nefarious agents to continue providing doubt to the agent learner, while friendly agents seemed to show some doubt themselves. Each friendly agent is given an initial 15% chance to provide false output. This false output has a 75% chance of providing a 0 quality back, removing this agent's insight on the jury, otherwise it returns the negative of the original output. Each time step decreases the initial chance of providing false output by 0.1%.

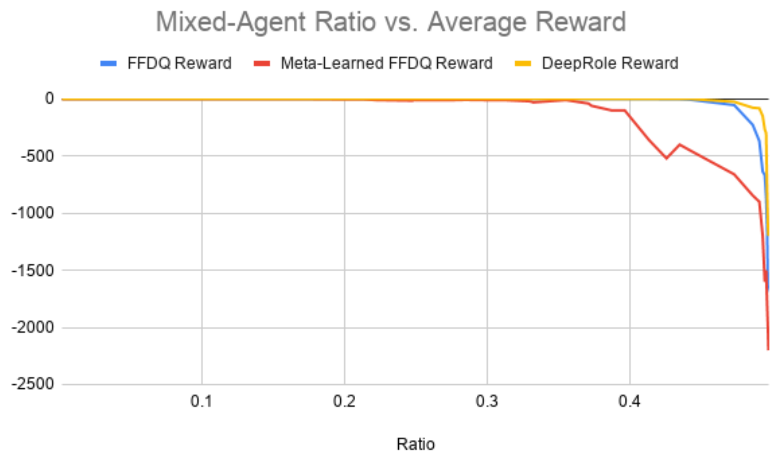

Figure 1: The average reward earned for each algorithm given a mixed-agent ratio.

6. Discussion

FFDQ turned out to be quite accurate in terms of reward loss. There is a maximum reward of 0 since there is no gain for accurately identifying an agent, only loss. This loss function was tuned to provide a small loss for a friendly agent misidentified and a large loss for a foe agent misidentified. This reward function is in line with real world scenarios, as befriending a nefarious agent can have much worse consequences than thinking a friendly agent is a foe.

Taking a look at figure 1 we can see that while the mixture of nefarious agents to friendly agents was small, all of the compared algorithms performed perfectly. All achieved maximum reward during testing. The meta-learned FFDQ started to misidentify agents when the agent pool was about 35% nefarious. From there it had a semi-linear decrease in reward. Around 48% for FFDQ and 49% for DeepRole is where these algorithms started to misidentify agents. At 50%, all algorithms took a deep dive in reward earned. This is because at this point, the agent learner does not have enough data on one side or the other to determine which agents to trust. The ambiguity is largest here since half of the agents are saying one thing, while the others are saying another. This is worsened by the fact that friendly agents have a probability to respond incorrectly, as discussed in section 5. Thus the meta-learned version, while not entirely optimized for this exact scenario since it had to generalize over two tasks, started to misidentify agents earlier than FFDQ and DeepRole.

One issue with CFRM is that because it uses an iterative inductive approach, it consequently takes longer to converge while training. FFDQ, on the other hand, assembles a jury and takes into account the trustworthiness of each juror, which influences the final update to the scrutinized agent. This allows the majority to quickly overcome the small amount of nefarious agents in the agent pool. Since the agent learner does not know how many nefarious agents there are, it tries to assemble dynamically-sized juries to find a general consensus. Figure 2 shows this happening. CFRM took an almost constant time converging, on average. We can see that FFDQ has a very erratic line, because the graph is measured by mixed-agent ratio rather than total agent count. We can deduce that the

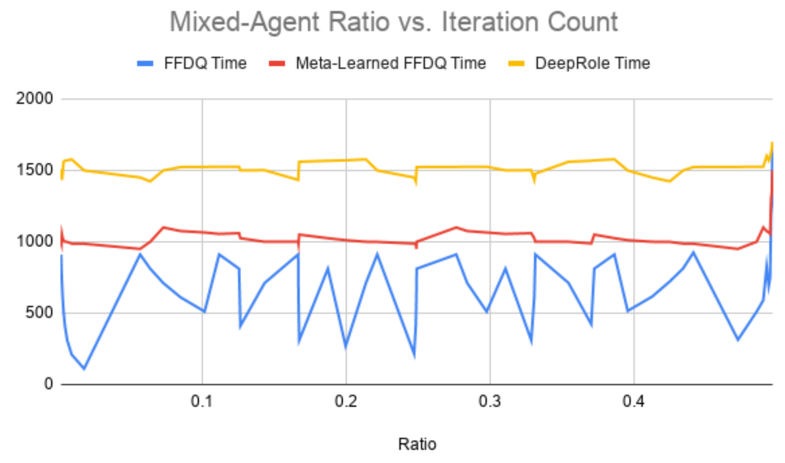

Figure 2: The average iteration count until each algorithm was in a terminal state given a mixed-agent ratio.

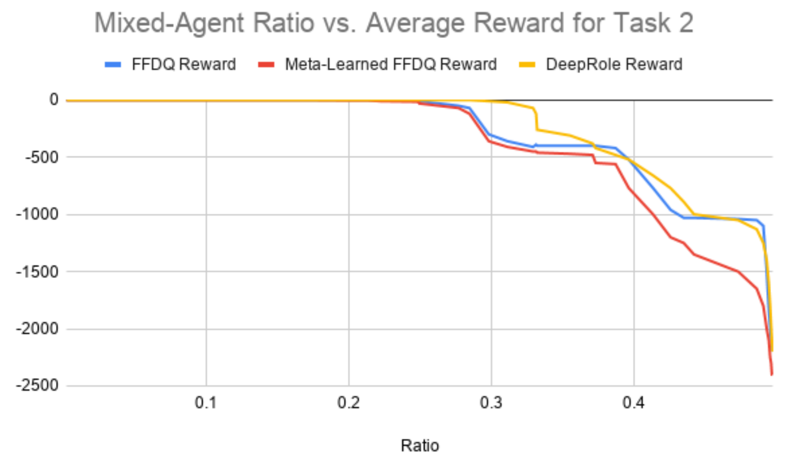

Figure 3: The average reward earned for each algorithm given a mixed-agent ratio for task 2.

erratic behavior follows the actual agent count from the ratios. Thus FFDQ converged extremely fast, regardless of agent pool size. The meta-learned FFDQ converged faster than CFRM as well but in a similar fashion, as it was not as easily influenced by the agent pool size. This was likely due to the hyper-parameters adjusting to both tasks simultaneously.

For the second task, all algorithms suffered from reward loss and an increased iteration count. This task removes a lot of certainty from the agent learner, and has to try and determine agent identification when many agents do not know of each other. Figure 3

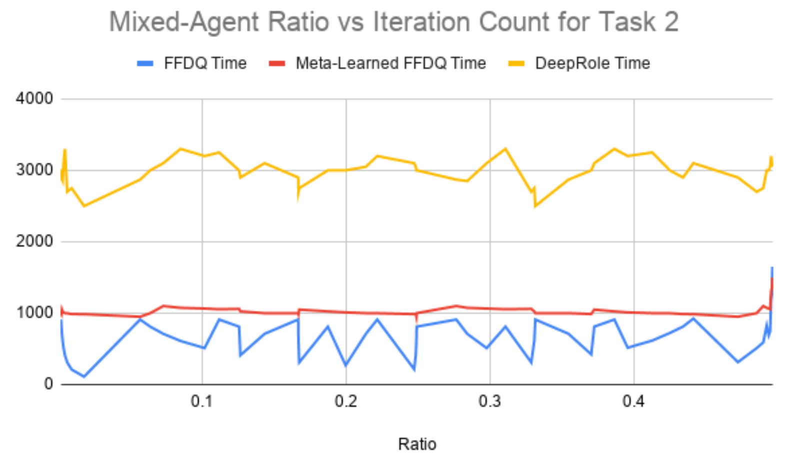

Figure 4: The average iteration count until each algorithm was in a terminal state given a mixed-agent ratio for task 2.

shows that all three algorithms began losing reward between 25% and 35% nefarious agent mixture. DeepRole had a linear falloff as it started having trouble finding the correct probability that a given agent was friend or foe. This is even further shown in figure 4, where the iteration count to converge was much higher than its task 1 run. However, here DeepRole had less of a flat graph which is the result of differing graph structures from instance to instance.

FFDQ and meta-learned FFDQ had the same convergence rate from task 1 to task 2. They both had little trouble deciding friend from foe even if it was inaccurate. Figure 3 shows that both FFDQ algorithms began suffering reward loss earlier than DeepRole. This stems from FFDQ being more certain about its environment than DeepRole which comes at a cost. However, friendly agents are at a disadvantage when it comes to actual response and knowledge about the other agents in the pool.

FFDQ was unable to perform well on the task it was not trained for. Both instances, FFDQ:1 and FFDQ:2, did well regarding reward loss on their respective trained problems, however, when they were tested on the other problem they were both unable to identify friend from foe. FFDQ:1 expected all agents to know each other and gave more weight to some agents than it should have. FFDQ:2 had the opposite problem; it undervalued certain agents due to the internal graph structure of its original problem, which allowed nefarious agents to trick FFDQ:2. The meta-learned FFDQ had a decent reward loss for both instances, as it was able to find hyper-parameters that did well on both instances. When tested, the meta-learned FFDQ was able to show that it could learn both tasks well from the learned hyper-parameters and had minimal reward loss similar to figures 1 and 3.

7. Conclusion

CFRM is a very popular technique in solving imperfect knowledge games such as friend-or-foe detection as described in this paper. By using inductive techniques, CFRM can build belief states and minimize its regret to accurately deduce the hidden knowledge of a game. However, it is slow in doing so. This paper presents an algorithm, FFDQ, that attempts to improve upon this deficiency by taking a different approach other than regret minimization. This approach, using a jury, seems to be effective as it crowdsources trust from individual members of the crowd itself. This paper also shows that by meta-learning FFDQ, it can be trained over a series of similar tasks of imperfect games and do well. FFDQ is competitive against CFRM in terms of iteration convergence yet has no significant reward loss, especially when the ratio of foes to friendly agents is small. CFRM outperforms FFDQ when the ambiguity nears the critical threshold of a 50% mixture of agents. Future research should look into how FFDQ works in general for imperfect knowledge games and if this jury technique can be extracted and then extended in other algorithms other than Q-Learning. Additionally, future researchers might look into delaying the convergence speed when the ambiguity is large to minimize reward loss. FFDQ seems to be a simpler alternative to CFRM and might be a beneficial technique to use over CFRM when the ambiguity is low or when some reward loss is acceptable.

References

Brown, N., Lerer, A., Gross, S., & Sandholm, T. (2018). Deep counterfactual regret minimization. *arXiv preprint arXiv:1811.00164*.

Cheng, H.-T. (1988). *Algorithms for partially observable Markov decision processes*. Ph.D. thesis, University of British Columbia.

Cicirello, V. A., & Smith, S. F. (2000). Modeling ga performance for control parameter optimization. In *GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 235–242.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org.

Hazan, E., & Kale, S. (2011). Beyond the regret minimization barrier: an optimal algorithm for stochastic strongly-convex optimization. In *Proceedings of the 24th Annual Conference on Learning Theory*, pp. 421–436.

Jin, P., Keutzer, K., & Levine, S. (2017). Regret minimization for partially observable deep reinforcement learning. *arXiv preprint arXiv:1710.11424*.

Kash, I. A., Sullins, M., & Hofmann, K. (2019). Combining no-regret and q-learning. *arXiv preprint arXiv:1910.03094*.

Kasperski, A., & Zielinski, P. (2006). An approximation algorithm for interval data minmax regret combinatorial optimization problems.. *Inf. Process. Lett.*, *97*(5), 177–180.

Li, H., Hu, K., Ge, Z., Jiang, T., Qi, Y., & Song, L. (2018). Double neural counterfactual regret minimization. *arXiv preprint arXiv:1812.10607*.

Littman, M. L. (2001). Friend-or-foe q-learning in general-sum games. In *ICML*, Vol. 1, pp. 322–328.

Marchiori, D., & Warglien, M. (2008). Predicting human interactive learning by regret-driven neural networks. *Science*, *319*(5866), 1111–1113.

Nguyen, K. C., Alpcan, T., & Basar, T. (2009). Stochastic games for security in networks with interdependent nodes. In *2009 International Conference on Game Theory for Networks*, pp. 697–703. IEEE.

Nichol, A., Achiam, J., & Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Nichol, A., & Schulman, J. (2018). Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, *2*, 2.

Pollack, J., & Ranganathan, P. (2018). Aviation navigation systems security: Ads-b, gps, iff. In *Proceedings of the International Conference on Security and Management (SAM)*, pp. 129–135. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Saad, E. (2010). Reinforcement learning in partially observable markov decision processes using hybrid probabilistic logic programs. *arXiv preprint arXiv:1011.5951*.

Schweighofer, N., & Doya, K. (2003). Meta-learning in reinforcement learning. *Neural Networks*, *16*(1), 5–9.

Serrino, J., Kleiman-Weiner, M., Parkes, D. C., & Tenenbaum, J. (2019). Finding friend and foe in multi-agent games. In *Advances in Neural Information Processing Systems*, pp. 1249–1259.

Steinberger, E. (2019). Single deep counterfactual regret minimization. *arXiv preprint arXiv:1901.07621*.

Syed, U. A. (2010). *Reinforcement learning without rewards*. Ph.D. thesis, Princeton University.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279–292.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Ph.D. thesis, King's College, Cambridge.

Wiering, M. A., van Hasselt, H., Pietersma, A.-D., & Schomaker, L. (2011). Reinforcement learning algorithms for solving classification problems. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 91–96. IEEE.

Yang, Y., Caluwaerts, K., Iscen, A., Tan, J., & Finn, C. (2019). Norml: No-reward meta learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 323–331. International Foundation for Autonomous Agents and Multiagent Systems.

Zhang, Z., & Ji, X. (2019). Regret minimization for reinforcement learning by evaluating the optimal bias function. In *Advances in Neural Information Processing Systems*, pp. 2823–2832.

Zhou, F., Wu, B., & Li, Z. (2018). Deep meta-learning: Learning to learn in the concept space. *arXiv preprint arXiv:1802.03596*.

Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2008). Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pp. 1729–1736.