CS 760 - Final Project
Reinforcement Learning using ANN and FOIL

Tushar Khot
Computer Sciences Department
University of Wisconsin, Madison, WI
tushar@cs.wisc.edu

Abstract

In this paper, we use two techniques to generalize the Q-tables in reinforcement learning. Perceptrons are useful to predict the Q-values for unknown states but may not always be accurate. FOIL derives rules for each action but sometimes may have no rules for some states. Each approach has its advantages and in this paper, we try to compare these approaches and ways to combine these approaches to make better decisions.

1 Introduction

Reinforcement Learning uses various techniques for deciding on the next move based on the current state. One technique is to score each state-action pair based on their expected reward(Q-learning). If we were to store all these Q-values in a table, then we would converge to the optimum policy, which would be to select the maximum Q-valued action each time[1]. But if the state space is too large then it would be impossible to store all the Q-values. One solution is to predict the Q-values based on the Q-values of similar states with techniques such as Perceptron. Another approach is to directly derive the policy using predicate calculus techniques such as FOIL[2]. The later two approaches help in generalizing from a comparatively smaller set of training data.

If were to use just a perceptron for predicting the next action, we wouldn't need to store the entire table but just the weights of each feature. Every time we obtain a new Q-value, we would update the weights based on this new Q-value. But this would mean, that it may tend to forget the old values. To

solve this problem, after every game we update the perceptron with all the Q-values in the Q-table currently. But even then the data may not be linear and hence the predictions may still be off.

If we were to use any rule based system such as FOIL, it would generalize well and also give human understandable results. For e.g., in the AgentWorld framework we would expect the agent to learn rules such as - "if there is food to the north and no opponent to the north, then move north". But FOIL unlike ANN can't predict the next action for states that don't satisfy any rule. Also, FOIL has no easy way to perform incremental updates, hence we use batch learning from the Q-table after every game with a small correction during the game. If an action gives a negative reward, we delete the rules that supported this action.

2 Problem definition and Algorithm

We use the AgentWorld framework to perform reinforcement learning. This world has vegetables that an agent must obtain and opponents and walls that it should avoid. There are minerals that it should avoid but can use them to hit other opponents.

2.1 Features

The AgentWorld framework provides the sensor readings for each agent. Each sensor measures the object type and distance of each object seen by it. We convert these readings into 48 boolean features. We have one boolean feature for each object type(Opponent/Mineral/Vegetable) in each direction(8 directions).Also there are three distance features(Far/Mid/Near) for each direction. There are 8 possible actions which are the 8 possible directions.

2.2 Q-Learning

Since the feature space is very huge(2^{48}), we use a sparse representation of the Q-table. Instead of storing Q-values for each state-action pair, we store only those state-action pairs that have a non-zero Q-value. Also instead of storing it as a pair, we maintain a list of action-reward pairs for each state. This prevents repetition of states. We use 10-step Q-learning for computing the Q values. Since there tend to be many empty spots in the AgentWorld(specially corners) computing rewards over longer steps helps the agent to move away form the corners. Other parameters:
$\gamma = 0.9$

α =1/#updates
Exploration probability, p = $1/\sqrt{moves}$

Update counts are also stored in the action-reward pair.

2.3 Perceptron

The Q-table is used as the training data for a perceptron. There is one perceptron for each action. The boolean features are fed to a Perceptron as input with an additional feature for threshold. At the end of each round, the entire Q-table is used to batch update the perceptrons. Also during each game, the new Q-values are sent to perceptron for one round of forward and back propagation.

2.4 FOIL

FOIL though designed for predicate calculus can be extended to propositional logic by considering addition of a proposition at each step rather than a predicate. Also the training data has Q-values instead of positive/negative samples. We assumed all actions that result in negative scores as negative samples and others as positive samples. We derived rules for each action using FOIL but this could lead to rules for different actions returning true for the same state. Hence we associated scores with each rule and used that to score the actions. The algorithm used is very similar to FOIL except the scores for the rules.

- Basic Algorithm

```
For each action, a
    RULES = {}
    POS = Positive eg's not satisfied by RULES
    NEG = Negative eg's
    While |POS| > 0 and |RULES| < 30
       Rule = a :- {}
       Neg = NEG
       Pos = POS
       While |Neg| > 0
         Take the best scoring feature, f.
         Add f to RHS of Rule.
         Neg = Neg examples that satisfy new rule.
         Pos = Pos examples that satisfy new rule.
```

```
Add Rule to RULES with score(Pos)
POS = POS - Pos
```

- Scoring Function
  score(X) = $\frac{\Sigma_{x \in X} |reward(x)|}{|X|}$
- Feature score
  Each feature score is computed using
  P0 = current positive examples satisfying Rule.
  N0 = current negative examples satisfying Rule.
  P1 = if feature added, new set of positive examples.
  N1 = if feature added, new set of negative examples.
  score(f) = $|P0| *$
  $(log \frac{score(P1)}{score(P1)+score(N1)}$ -
  $log \frac{score(P0)}{score(P0)+score(N0)})$
- Inference
  For inference, we sum the scores of the rules that are satisfied by the current action and select the best action.

2.5 Combination

We tried various approaches to combine the results of the strategies mentioned above.

1. Only-Q/FOIL/ANN : This approach just uses one strategy(Q-learning/FOIL/Perceptron) for making the next move.
2. Fallback-FOIL/ANN : If the Q-Table has no Q-value for a state or has only negative values for a state, it cannot make a well informed decision of the next step. So in such cases we use FOIL/ANN to take the next step. We assume that opposite direction of the negative reward action, is not necessarily the best direction.
3. Coll-FOIL/ANN : This is slightly more intelligent than the previous approach by not just blindly following FOIL/ANN. If it has actions in the Q-table with negative values, it will only accept results from FOIL/ANN that don't belong to the negative set.

2.6 Miscellaneous tricks

There are some minor tricks that were used for helping the agent.

- Continue Motion
  For any score returned by the scoring function, we multiply the score by a factor, if the action is the same as the last move and didn't result in a negative reward. This prevents oscillation between states.

- Hot Start
  To save time on learning with random walks which can be very slow, we had built a simple agent that just moves towards the food. This agent was run in an identical AgentWorld framework and used as the initial training data for each approach. The agents would use the logs from this simple player on startup and pretend the state,action,reward values to be obtained by actual interaction with the environment.

- Noisy Movement
  To prevent the agent from oscillating between states, we add noise to every action taken by the agent. For e.g., if the policy were to suggest to move North, we would move at an angle of 0 +/- random noise(<10).

3 Experimental Evaluation

3.1 Methodology

For comparing the various approaches, we use the AgentWorld framework with the following settings.

Minerals: 5. By keeping few minerals, we let the agents learn that the minerals can be used to hit opponents but shouldn't make finding food almost impossible. The count although is more to the conservative side.

Food: 50. By not keeping very large amount of food, we prevent just random walks resulting in good scores.

Games: 30. We noticed that the AgentWorld framework may have pathological setup of agents that can result in very low scores. For e.g., an agent may be at a corner with a random walker blocking its exit. Hence we take the scores over 30 games and plot histograms of the scores instead of plotting them as learning curves which are jagged.

Moves: 1000. This count was selected to allow for faster execution of the experiments yet provide enough time for the score to have stability. Histogram over multiple games should reduce the impact of any instability in

few games.

Players : Assassin, SamplePlayer, RandomWalker. Each game had three other agents. An assassin who attacks an opponent. A Sample Player that goes for food but is very afraid of other opponents. A RandomWalker that takes random moves.

Unless otherwise mentioned most of the scores are obtained by one Agent player in the above mentioned setting. Individual scores for various agents in this scenario are not comparable but the histograms and average scores should still be valid.

3.2 Results

1. Basic Policy

   Figure 1 shows the histogram of scores for agents using Q-table, Perceptron and FOIL. As shown in the figure, FOIL seems to perform better than the other approaches. It actually is comparable in performance to the Sample Player that moves towards the nearest food and away from an opponent.

   To be able to actually compare the results, we ran all agents in the same set of 30 games and compared the scores using a t-test.

$$\begin{aligned} \text{FOIL Vs Perceptron} & \\ \frac{|\mu_\delta|}{S_\delta} &= 4.43 > T_{95,29} \\ \text{Q-Table Vs Perceptron} & \\ \frac{|\mu_\delta|}{S_\delta} &= 2.93 > T_{95,29} \\ \text{FOIL Vs Q-Table} & \\ \frac{|\mu_\delta|}{S_\delta} &= 1.69 > T_{95,29} \end{aligned}$$

   This clearly gives the ranking that FOIL is the best approach, followed by just using Q-tables and lastly the ANN. Though these results have to be taken with a pinch of salt, because of the fact that if an agent performs really better than others, it would further reduce the score for that round for others as the food would be scarce.

   As Figure 2 shows ANN suffer a lot in the beginning when they lack sufficient training examples and hence dont approximate well to the

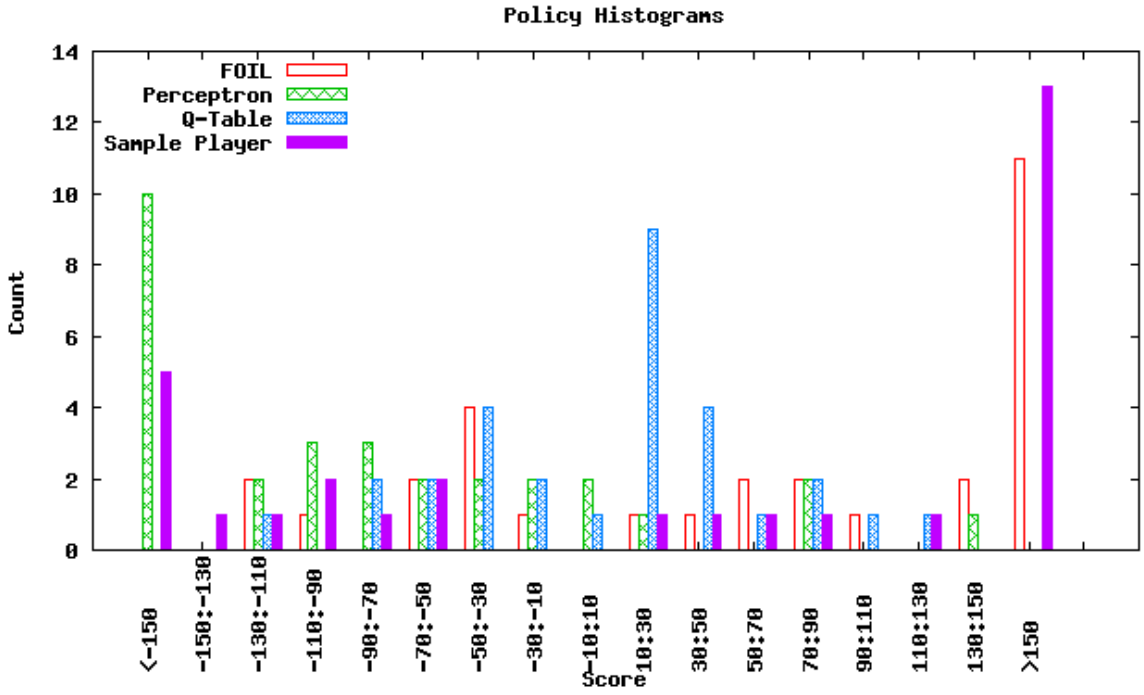

Figure 1: **Basic Policies**

actual function. The figure also shows the spikiness of the graph and hence the need to use histograms.

2. Fallback Policies
   Figure 3 shows the comparison between using either FOIL or Perceptron as the backup for cases where Q-table is not sufficient. Also we tried using Perceptron as the backup strategy for FOIL. Other configurations are not possible as ANN always would return a result even if it has never seen a state before.

   Similarly we made the three backup policies compete with each other in 30 games and compared the scores for each game.

$$\text{QT-Perceptron Vs QT-Foil}$$
$$\frac{|\mu_\delta|}{S_\delta} = 0.59 < T_{95,29}$$
$$\text{QT-Perceptron Vs FOIL-Perceptron}$$
$$\frac{|\mu_\delta|}{S_\delta} = 2.35 > T_{95,29}$$
$$\text{QT-Foil Vs FOIL-Perceptron}$$

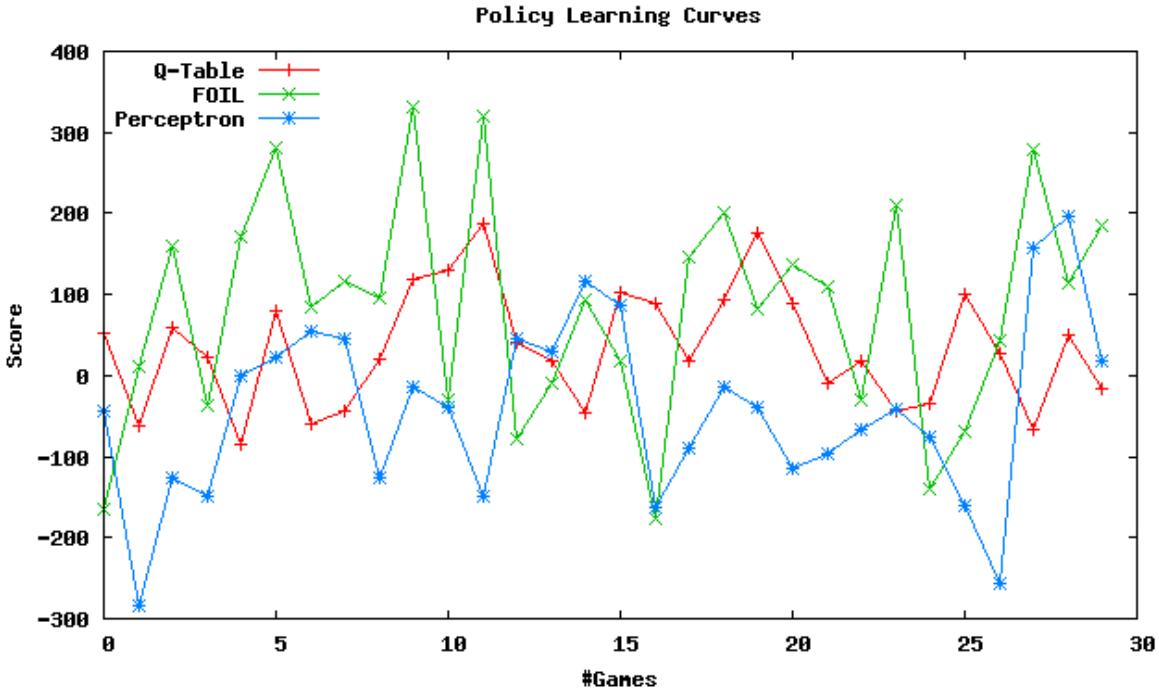

Figure 2: **Basic Policies Learning Curves**

$$\frac{|\mu_\delta|}{S_\delta} = 1.84 > T_{95,29}$$

So as the results show, QT-Perceptron and QT-Foil have almost similar performance. This could be because of the fact that Q-Tables actually donot have any result about 50% of the time(13K-15K times out of 30K moves) and hence balances the policies well. FOIL on the other hand is unable to decide actions for only 9K moves which makes it almost same as the only-FOIL approach and doesn't see any improvement.

3. Collaborative backups Figure 4 shows the comparison between the various collaborative approaches and their counter-parts.

   A t-test over 30 games with competing agents reveals:

   QT-Perceptron-Coll Vs QT-Foil-Coll

   $$\frac{|\mu_\delta|}{S_\delta} = 0.62 < T_{95,29}$$

   QT-Perceptron-Coll Vs QT-Perceptron

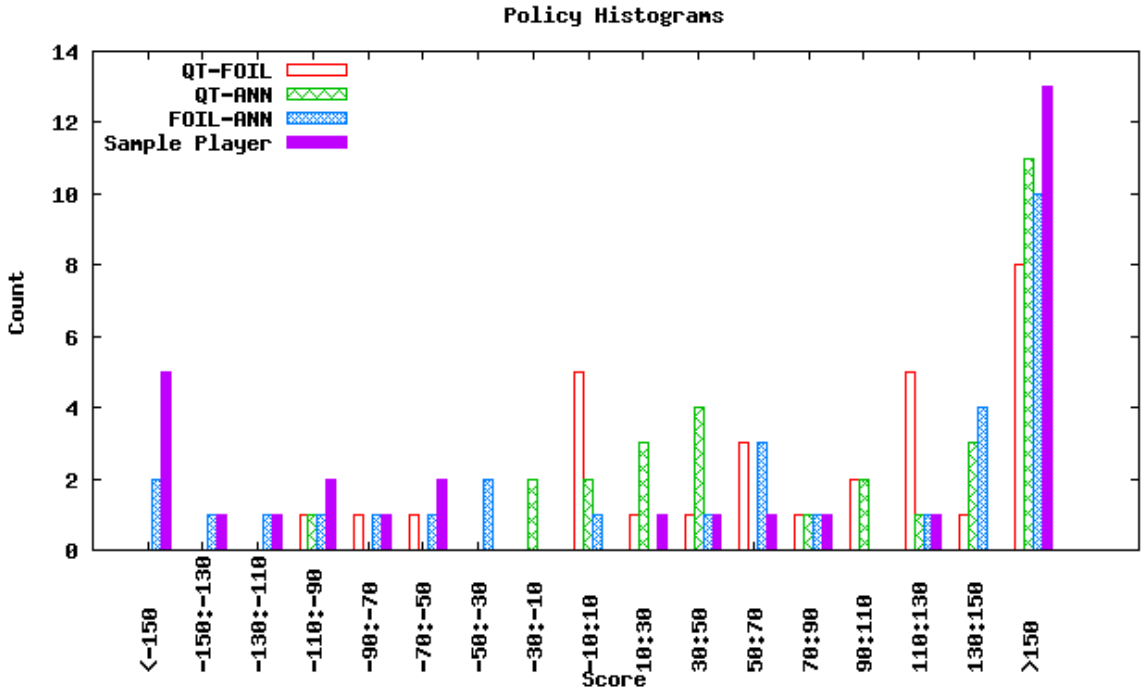

Figure 3: **Basic Fallback Policies**

$$\frac{|\mu_\delta|}{S_\delta} = 1.16 < T_{95,29}$$

QT-Foil-Coll Vs QT-Foil

$$\frac{|\mu_\delta|}{S_\delta} = 0.52 < T_{95,29}$$

QT-Perceptron-Coll Vs SamplePlayer

$$\frac{|\mu_\delta|}{S_\delta} = 1.01 < T_{95,29}$$

As the results above indicate collaboration betweeen Q-tables and Perceptrons/FOIL though seem to help the scores, the improvement is not statistically significant. This could be because of the fact that collaboration was used only on 500-1000 moves out of 30000 moves.The impact could also be less because of the fact that these policies are learnt from the same Q-table and most of the time should have the same predictions. The impact on Perceptron seems to be the only noticeable improvement possibly because of the fact that Perceptrons may have incorrectly approximated the function and suggest wrong moves,

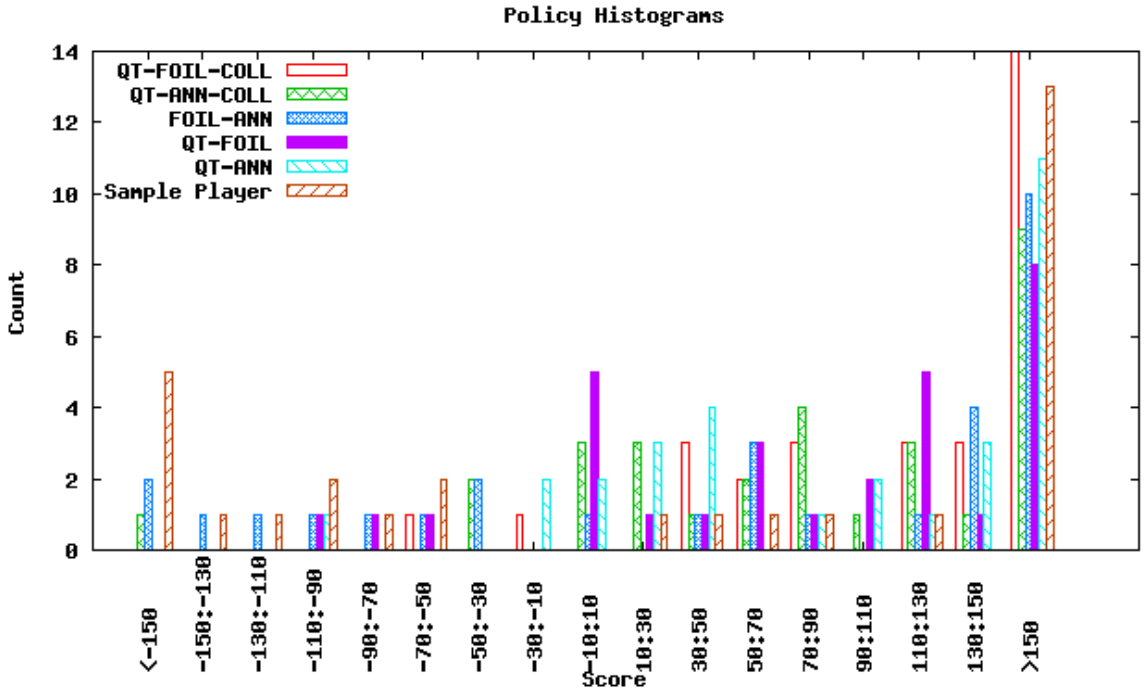

Figure 4: **Collaborative Policies**

but now some of the negative reward moves are prevented.

4. Miscellaneous experiments
   There were some experiments done for deciding on various features/parameters.

   For deciding on the number of steps for Q-learning, we ran games with different K values(#steps) and selected 10 to be the optimal choice. After deciding K, we fixed the gamma to be 0.9 in a similar fashion.

   Also we noticed that just using 4 directions as the actions was insufficient as the Sample Player would move diagonally and hence would reach the food before we did. Hence we decided to work with 8 actions even if it increased the sparsity of the representation. Experiments indicated that the average score improved with 8 directions even in the absence of the Sample Player.

3.3 Discussion

As expected the backup policies bridge the gap between FOIL and Perceptrons by using the more reliable Q-Table half the time. This was expected as when one policy has insufficient information, rather than making a random move we can make a more informed decision. Collaboration with the Q-table improved the scores further as they prevented obvious mistakes. Since Q-tables are mostly accurate, using it as a guide for the other policies is helpful.

4 Related Work

Using perceptrons to approximate the Q-table is a very common approach. Logic(MLN/Rules) on the other hand is predominantly used only as a transfer mechanism from one RL task to another[3]. This is very similar to the "Hot Start" idea to use the results from another agent to speedup learning.

The idea of using scores for each rule was derived from Markov Logic Networks where each predicate rule has a weight instead of being absolute truths [4].

5 Future Work

The current FOIL approach is not incremental and is just attempting to cut the losses. Maybe a better approach could be derived to incorporate incremental updates.

The collaboration between the various approaches can take into account the result from each algorithm and use an ensemble kind approach to decide which is the next best step to take. This would slow down each step and maybe should be used only when required. Also too many policies would also reduce the efficiency.

6 Conclusion

The experiments indicate that as a single policy, FOIL is better off than the other two approaches as it generalizes well with even limited data. With the combination with Q-tables, Perceptron avoids the early mistakes and hence is comparable to FOIL. Using collaboration has little effect since Q-Tables rarely have only actions with negative rewards. Even if they do, since the backup policies learn from the same Q-Table, they make similar decisions.

References

[1] Christopher J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, UK, 1989.

[2] J.R. Quinlan. *Learning logical definitions from relations.* Machine Learning, 5:239-266, 1990

[3] Lisa Torrey, Trevor Walker, Jude Shavlik, and Richard Maclin. *Using advice to transfer knowledge acquired in one reinforcement learning task to another.* In Proceedings of the Sixteenth European Conference on Machine Learning, 2005.

[4] Richardson, M., Domingos, P. (2004). *Markov logic networks* (Tech. Rept.). Dept. Comp. Sci. Eng., Univ. Washington, Seattle. http://www.cs.washington.edu/homes/pedrod/mln.pdf.