Can computers learn to lie

Surname, Name; Crosta, Tomas
14 January, 2021

Abstract: Computers are seen normally as perfect machines that follow fixed rules to get the best "secure" return, but what happens if it is impossible to get a perfect strategy with the information they have? Will machines learn to lie to compensate the lack of information?
I trained two different AI's as observers with a Q-learning algorithm and a neural network to play a simple "min-max"game, in this game there is the possibility to lie but it has a risk factor associated.
This two different AIs got different results making the Q-learning algorithm lie about 30 % of the time and the neural network lying less than 3 % of the matches.

Keywords: Neural network, Q-learning, Computer intuition.

1. Rules of the game

The game requires 2 players and a deck with 48 cards. First both players receive a different card. The match starts with a 1 point bet, if the players want, they can duplicate the bet up to 8 points but if one doesn't agree to raise the bet then the other player wins half of the points and a new match starts. If both players agreed on the bet, then they have to throw their cards (at any time both players can still raise the bet on their turn) and the player with the highest value card wins the points. In case of a tie, the player that started the game wins all the points.

The Player that gets 15 points is the winner of the game.

2. Introduction

This kind of games is hard because of the luck component. There is no perfect strategy and with the information you start is hard to decide if you are going to win. For example, if you are the second player and you have a 12, you most likely are going to win the match, so you can bet more points, but there is still over a 6 % chance that you are going to lose that match, so the evaluation of each state is hard to determine.

There are many games that follow this pattern with different rules like "Truco" or "Poker", in these games is important to make your opponent think that you always have better cards than he does. This makes two basic strategies one is always telling the truth and rely on luck, this will make you win about 50 % of the time, but if your opponent lies to you, you may lose point when you could have won the match. On the other hand, if you lie too much, the other player could notice or have a very good hand and accept your bet, that means that the strategy followed should be a balance between both.

The aim of this report is to see how different AI's balance the amount of lies, we will consider that an AI is lying if they are playing for a higher bet than 1 and have less than a 50 % chance to win with their cards, The Q-learning algorithm is going to be based on his experiences of the game and the neural network is going to try to predict the best card watching an algorithm that can see both cards.

3. Related Works

A lot of work has been done about games with "imperfect information" like **Poker Games**, in this paper for example, the researchers trained a Neural Network with data from previous games of real people and found that the AI could predict about 81 % of the plays and on 3.3 % of the cases it could predict if the opponent was going to lie.

This work also showed that the AI was capable of playing "like humans" but the main difference was that the AI was not adaptive enough so it was bad against "slow players" and couldn't exploit the opportunities.

4. Methodology

4.1. Player

There are only 4 decision possible in this game, raise the bet (input=0), throw a card (input=1), resign (input=2) and accept the bet rise (input=3). The player starts with a list containing 4 numbers that represents the value of the state giving the input. The player normalizes the values, this gives the probability of returning each input and finally using a random number returns the answer of the player.

4.2. Observer 1: Q-learning AI

Q-learning is a reinforcement learning algorithm in which for each state-action pair (s,a) the algorithm gives a value Q(s, a). In this game, the state can be passed like a vector with 6 components:

$$data = (p_0, p_1, c, t, p_m, s) \tag{1}$$

Where p_0 and p_1 are the points of the players, c is the card that the player has, t is the card that is on the table, p_m is the current bet and s is 1 if the player just raised the bet, otherwise is 0.

This makes a total of 319480 different states, each state and the evaluation for that state are saved on a matrix. To make the training faster, each state started with a value of:

$$Q(s, a) = \frac{p_1 + 1}{16(p_2 + 1)} \tag{2}$$

This way the AI gives always a value between 1 and 0 for each state and knows that moving toward the states with higher score is good.

To update the AI, when the match finishes, the algorithm receives a feedback:

$$feedback = \frac{p_1 + 1}{16(p_2 + 1)} \tag{3}$$

And the final state after the match will be:

$$Q(s, a) = Q(s, a) + \alpha(feedback - Q(s, a)) \tag{4}$$

Where α is the learning rate.

This way it doesn't mater if the player lie, Q(s, a) will grow only if they win points, but it will decrease if they lose.

Finally, the AI gives the 4 numbers to the player:

$$probability = (Q(s, 0), Q(s, 1), Q(s, 2), Q(s, 3)) \tag{5}$$

$$= (Q(s_1), Q(s_1), Q(s_2), Q(s_3))$$

Where $Q(s_i)$ is the value if in the state s is used the input i.

The name of the AI was **Ray**

4.3. Observer 2: Neural Network

The neural network called **Norman** has 6 inputs, 2 hidden layers with 6 nodes each and 4 outputs using a "relu" activation for the hidden layers and a "sigmoid" activation for the outputs.

To get the probability for the player, we input into the neural network the current state and it returns a tensor with the 4 values normalized.

The AI was trained trying to predict the "correct answer" in about 6 million matches. The "correct answer" was given by an AI that could see both hands and had specific instructions.

4.4. Experiment

First the Q-algorithm played against itself 500.000 games, all the games where saved on a .csv file with the correct answers for each game. Then 5 different neural networks were trained with an **Adam** optimizer and a **binary cross entropy** loss function. The neural networks had: The AI that could predict more of the plays, About 75 %, was the one with 6 nodes and 2 hidden layers, so the rest of the experiment was continued only with it.

hidden layers	nodes
1	4
1	6
2	4
2	6
3	4

Tabla 1: Amount of hidden layers and nodes for each neural network setup.

Then we deleted all the data and started both AI (Ray and Norman) with about the same knowledge of the game.

First Norman and Ray with no data played 100 games, it was counted the amount of lies and how many games have won each, then the Ray played against itself for 5000 games and Norman train with it. This process was repeated 100 times. Finally, it was graphed **win rate vs matches played** and **lies vs matches played**.

5. Results

We can see on **figure 1** the win rate vs matches graph, in this graph, we can appreciate that at the beginning Ray and Norman have almost the same win rate, but Norman's learning is better when the amount of matches increases so the win rate increase's. This win rate has to convert to some number (if it doesn't oscillate) because the win rate can't be bigger or smaller than 0. Let's say that exist a function that describes the win rate of both players, and we are going to approximate it with the "first" convergent term of his Lauren series 1/x, fitting the curve as seen on the **figure 1**, we found that the win rate of Ray at the "end" is about 27 % (This is the results made in a logical amount of time with the current hardware and can change if enough time is given).

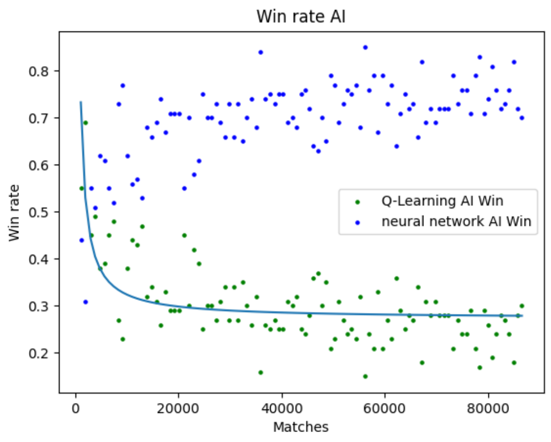

Figura 1: Graph of win rate vs matches of both AI's

On the other hand, if we graph the average lie per match against the matches played by the AI's, we obtain the **figure 2**, this shows that Ray lies much more that Norman.

The lies of Norman are quite stable, it starts lying about 5 % of the times and with time it limits the lies and in average lies 2.33 % of the time (It is important to note that with good hands the players can't lie).

On the other hand, Rays lies are different, it starts high (about 40 % of the times), then becomes lower (about 30 %) and finally it lies between 32 % and 40 % of the time, with an average of 35 % of the time.

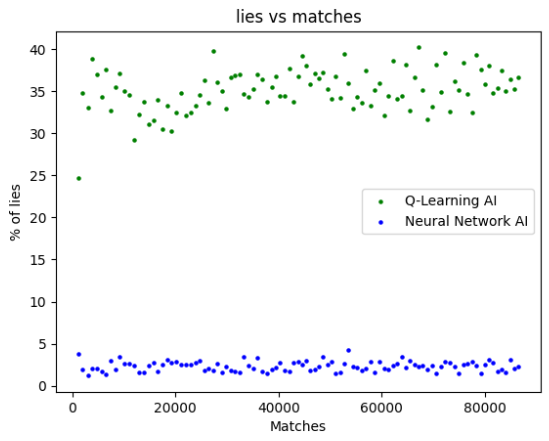

Figura 2: Percentage of lies that both AI tells during games vs matches

6. Discussion of the results

As seen on both graph, in this game Norman only lied about 2 % of the time, so it was able to see that lying in some cases may be possible, and with this strategy won about 73 % of the times. On the other hand, Ray played a more "aggressive" game but the drawback was wining only 27 % of the games.

The main difference in the strategies may come from where both AIs are learning, the neural network (Norman), learned trying to replicate a perfect game, this is not necessary the best approach because some factors like changing the aggressiveness if the game just started or if you are near losing the game.

On the other hand The Q-learning algorithm (Ray) learned just from experience, and didn't rely on perfect information at all, but the amount of possibilities and the different strategies from the opponents makes almost impossible for this AI to make better results than the neural network (Norman).

Finally, the rules of the game are very aggressive, both players can raise the bet by more than half the points needed to win the game, this makes harder trusting your cards and can be a very big feedback if the lie is good, this is probably why Ray tend to lie so much based on experience, unlike Norman that just saw the feedback of lying when it was truly possible.

7. Conclusions and future works

With this approach we saw that it is possible for computers to realize that lying is not bad in certain cases on an **imperfect information game**. We also saw that in this case the reinforcement learning algorithm tend to lie more than the one based on perfect information.

A future work that can be done, is mixing both AI, training first with perfect information a neural network and then use a reinforcement algorithm to get a better view of the game taking rewards based on the current points. Despite all of this the learning curve is not that fast and it would be hard to make the AI adapt to different players.

8. References

1-Hands-On Machine Learning with Scikit-Learn & Tensorflow. Part II, Chapters 9, 10 and 16.