Creating a Poker Playing Program Using Evolutionary Computation

Simon Olsen and Rob LeGrand, Ph.D.

Abstract

Artificial intelligence is a rapidly expanding technology. We are surrounded by technology that uses AI, and areas of application are spreading. We use AI for directions (GPS), in search engines, computer translation between languages, non-playable characters in video games, and much more.

In 2001 Dr. David Fogel published impressive results by combining artificial neural networks with genetic algorithms to create a checkers playing program. We have applied the same approach to poker. The idea is to use artificial neural networks to compute bets, and then use genetic algorithms to search for the optimal artificial neural network setting. The genetic algorithm will through simulated evolution generate better versions of the poker program by playing against itself.

After evolving the agents evaluated their performance by matching them against other poker programs and human players. These agents will be evolved without domain knowledge (human knowledge) about poker, and, while evaluating them, the main focus is to see what the computer was able to learn about poker.

Poker proved to be a difficult challenge for this approach, and currently there has not been evolved a really strong agent. However, the results do provide some pointers on what steps to take for future work, and I am certain that this approach could produce really interesting results given some more time.

Introduction

Artificial intelligence (AI) is no longer exclusive to high-tech labs and science fiction movies. AI is a rapidly expanding technology and there are a lot of practical applications of AI in our world today, such as a GPS providing directions, computer translation between languages, characters in a video game, and search engines. We have just scratched the surface of AI and more research in this field is essential.

Artificial neural networks can be used to compute values based on inputs, and genetic algorithms are useful to search for optimal solutions. Combining artificial neural networks and genetic algorithms have provided exceptional results for a checkers-playing AI and we wanted to see if the same approach could be applied to poker. The plan was to see whether we could evolve a good poker-playing agent without the use of domain knowledge. While creating poker-playing agents might seem trivial by itself, development in this approach of AI might lead to computers that are able to find answers to problems that human experts have not been able to solve.

Literature Review

There are several definitions of AI. One of the early challenges in AI was the Turing test. In this test, a human would communicate with a computer, and the computer would respond. If the human could not tell if the response came from another human or from a computer, the AI would be successful. In other words, if a computer was able to imitate a human, it would be considered intelligent (Warwick, 1991). This approach to AI assumes we can make a computer act (and perhaps think) humanly. However, many people have tried to solve the Turing test without any great success. It turns out a computer's "brain" works very different than a human brain. A more modern approach to AI is to make a computer act rationally. Some scientists have defined this as "[a] rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome" (Russell & Norvig, 2010, p. 4).

One of the most challenging and intriguing problems in artificial intelligence is machine learning. How can we make a computer learn from previous iterations of solving a task? It was not long ago that computers could not learn how to perform a task by examples. Instead of learning from past mistakes, experts where required to tell the computer how to act in every possible scenario of the task. Strides in machine learning have made it possible for computers to solve complicated tasks without long and exhausting pieces of hand-code (Anderson, 1986). One very interesting method to achieve machine learning is combining genetic algorithms with artificial neural networks.

Artificial neural networks are inspired by the biological nervous system found in a brain. They compute a value by feeding inputs through a system of connected perceptrons (artificial neurons). Each input is multiplied with a connected weight before they are added together and passed through a sigmoid function. The output of this sigmoid perceptron is used as input for neurons in the next layer of the network (Azadeh, Negahban & Moghaddam, 2012).

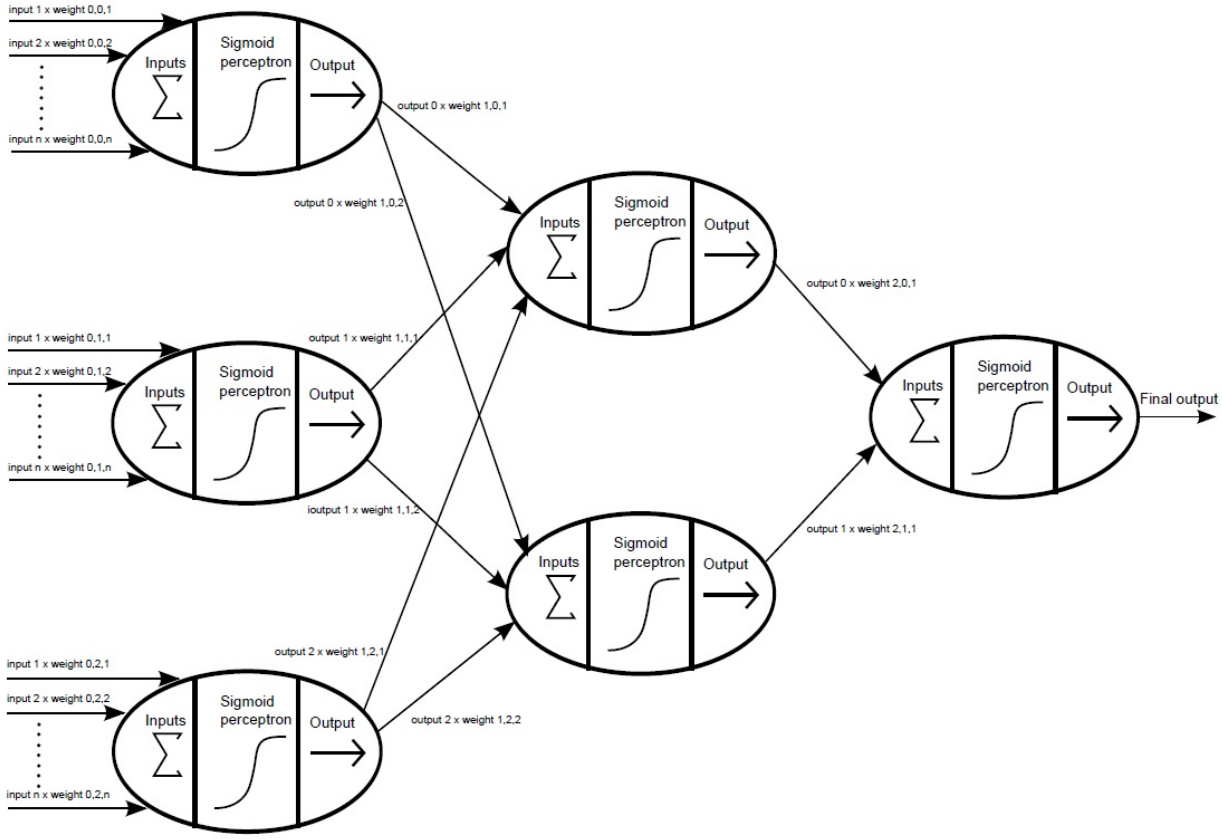

Figure 1: A sample layout of an artificial neural network

A computer can use a genetic algorithm to search for the optimal solution to a task. This is an evolutionary search algorithm and takes use of concepts such as reproduction (crossover) and mutation (Ozcelik, 2012). A set of artificial genes decides how the computer solves a given task. A computer searches for the optimal gene set by generating several random sets and, after evaluating how well each set performed, the computer discards the weakest genes and uses the strong genes for reproduction. Some mutation on genes is common as it allows for a broader search.

One of the leading computer scientists working with machine learning, and especially by using artificial neural networks and genetic algorithms, is Dr. David B. Fogel. When IBM's Deep Blue beat the world champion Garry Kasparov in chess, Dr. Fogel heard the news on the radio. The radio host proclaimed this as the moment when computers where officially more intelligent chess players than humans. While Dr. Fogel was impressed by the feat accomplished by the IBM team, he did not agree with the radio host. For him, Deep Blue was more like a really fast and expensive calculator than an actually smart computer. IBM spent millions of dollars building a computer with hardware fast enough to look at over 200 million positions a second and, based on human knowledge about chess, Deep Blue could decide which of these positions were desirable or not. Dr. Fogel wanted to design a program able to play at a really high level but without this human knowledge about the game. He started a research project called *Blondie24.* Blondie24 is an expert checkers player and is self-taught. Through evolution, Blondie24 obtained a rating of 2048, which is better than 99% of the playing population of the web site Blondie24 played on (Fogel, 2001).

A big difference between previous poker agents and the approach that we have used is the exclusion of domain knowledge. Previous AI research in poker has focused on making

computers play as optimally as possible. A research group at the University of Alberta has developed a poker program named Polaris that is already able to beat top human players in limit Texas hold 'em (Fleming, 2008). The focus on optimization and triumph over humans is much like what IBM did with Deep Blue and chess. My approach focuses on figuring out what a computer is capable of learning by itself, very similar to what Dr. Fogel did with Blondie24. However, because of random factors and probability, poker is a much more complicated game to solve for a computer than checkers.

Methods

There were three components necessary for this research. The first essential piece was the poker environment where the agents could play against each other. Here the agents played Texas hold 'em, no-limit, multiplayer poker. Each agent started out with $1,000 and the game continued until one agent controlled all the money. The number of agents at each table could vary. Since only one agent had any money left when the game was over, the agents were ranked on how long they were able to stay in the game. At the start of each round each agent was dealt two cards that only they could see. There are also five community cards that any agent could use in combination with his two cards to create the best possible five card combination. There were four rounds of betting: the first round after the two private cards have been dealt, the second after three community cards have been revealed, the third round of betting after the fourth community card has been revealed, and finally the last after the fifth and final community card.

Artificial neural networks were used to make the agents' decisions while playing poker. Information about the poker game (cards, money, amount of players in the hand, etc.) is used as inputs for the first layer of neurons in the network. Each input is multiplied with a connected weight before they are added together and passed through a sigmoid function ($\frac{1}{1+e^{-x}}$). The output of this sigmoid perceptron, which is always between 0 and 1, is being used as input for neurons in the next layer. When the inputs have been fed all the way through, the value computed by the network will be how much the agent is willing to bet. Ultimately, it is the different weights in the neural network that determine how the agent is going to play.

The genetic algorithm allows agents to evolve and become better poker players over time. Each agent was created with a random set of genes (in our case the weights in the neural network). Through playing poker, we discovered, stored, and reproduced strong genes, and discarded the weak genes. The reproduction consisted of taking genes from two strong agents, and combining them to create a new agent, while the weak agents were discarded. To broaden the search for strong genes, there was also a slight chance for gene mutation. The reproduction of genes and the games were local. 100 players (10×10 grid) all got their turn to host a game. They invited their neighbors to play, and during evolution genes were only spread to neighboring agents.

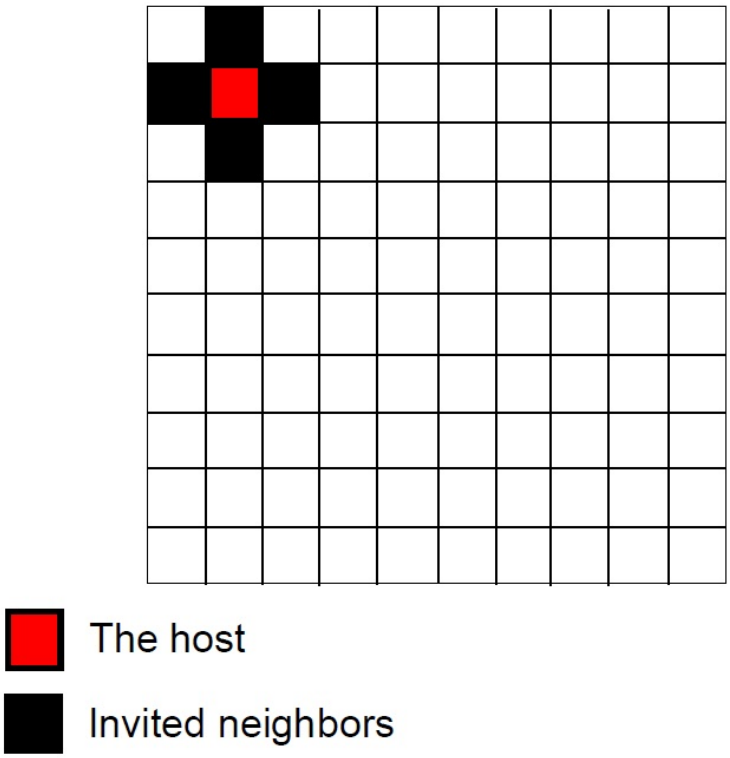

Figure 2: Layout for the population of agents

Different sizes and shapes of artificial neural network determine what an agent is capable of learning. Bigger networks allow more complex agents to emerge but also take more time to evolve. Several different neural networks were tested against each other to see which one produced the best agents within a set time-frame. The agents were then studied against human opponents to better recognize what the agents taught themselves.

Experiment

After setting up the poker environment we had to decide what information to give to the agents. We came up with 16 inputs we thought would be beneficial without giving too much information to the agents.

Input 1: Rank of 1st card in hand.
Input 2: Rank of 2nd card in hand.
Input 3: Rank of 1st community card.
Input 4: Rank of 2nd community card.
Input 5: Rank of 3rd community card.
Input 6: Rank of 4th community card.
Input 7: Rank of 5th community card.
Input 8: Money already in the pot.
Input 9: Number of pairs.
Input 10: Number of pairs of suits.
Input 11: Cost to stay in the hand.
Input 12: How many players are still in the hand.

Input 13: Agent money.
Input 14: The amount of money of the other agents.
Input 15: The amount of money the agent has already invested in the hand.
Input 16: The number of cards the agent has.
The rank is the numerical value of the card (2 – ace). All agents got the same inputs, regardless of neural network structure.

At first, in the interest of time, the agents only hosted one game in each evolution generation. They scored points based on how they placed in the games. They got 10 points for a win, 5 points for 2nd place, 3 points for 3rd place, 1 point for 4th place, and 0 points if they came in last. During evolution the genes' probability to spread was based on the agent's points. The agent with the most points would not change at all, while the four other agents would all get their genes changed. For each gene in the agents, the genes swapped to another agent's gene, based on a probability calculated by taking the score of an agent and dividing that number by the score of all agents. After the gene crossover there was a 5% chance for each gene to mutate. Gene mutation used the following formula: $Mutated\ gene = 2^x * old\ gene$, where x = a random number between −1 and 1.

The first neural network structure created was the simplest possible: just one neuron. While we did not expect this network to be able to produce any good agents, it would be interesting to see how it valued the different inputs. The final output was not sent through the sigmoid function, and would directly represent what the agent wanted to bet. To our surprise the evolution did not seem to settle, and after 10,000 generations we decided to run some tests. It seemed like it had not learned anything useful and the values of the weights were not consistent. For example, the average agent would value one of the two cards in the hand positively and the other card negatively. When we tried to play against the average agent, it ended up going all in every time. We decided to try to expand the neural network a little bit. The new network consisted of two layers, with 2 neurons in the first layer and 1 neuron in the second layer.

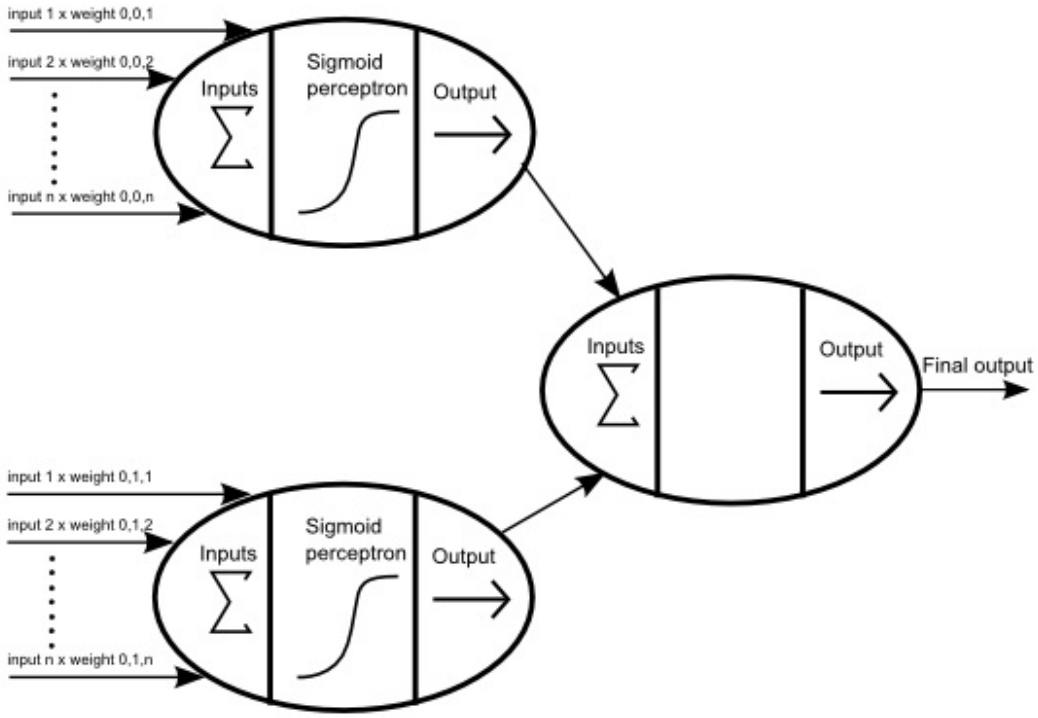

Figure 3: 2-1 Layout for the new neural network

However, this network followed the same patterns as the network with only one neuron. The weights seemed arbitrary, and it played just as aggressively. It seemed like the agents played all in all the time, which left everything to chance and luck.

We decided to make some changes before continuing to evolve agents. We had the agents host each table four times instead of just one. This increased the running time of the program, but perhaps just playing one game left too much up to luck. To stop the agents from playing all in all the time, we now used the sigmoid function even in the last neuron. Since the final output would now always be between 0 and 1, it now represented a fraction of that agent's money, so an output of 0 would indicate a decision to fold and a 1 output would go all in. To slow down the evolution and to try to make it a little bit more stable, we lowered the mutation rate. The new rate was now $Mutated\ gene = 1.5^{x} * old\ gene$. We started over with the simple one-neuron network, the 2-1 layout, and also a larger 3-2-1 network. Once again neither of the populations seemed to settle at an average agent, and while they were not playing all in like before, they seemed to make random decisions. Perhaps hosting the game four times still was not enough, so we increased this number to twenty. This significantly hurt the running time of the program, but seemed necessary to even out the luck because of all the random factors in poker.

This time we created six different neural networks, and after evolving them for a set amount of time, we were going to play them against each other to see which setup produced the best agent. There were two one-neuron networks, two 5-1 networks, and two 3-2-1 networks. Each pair had one network with a sigmoid threshold on the final output and one without this

threshold. The six different setups were allowed roughly the same evolution time, but due to different playing speeds they finished wildly different numbers of generations.

Agent number	Layers	Neuron structure	Threshold on output	Numbers of generations
1	1	1	No	599
2	1	1	Yes	499
3	2	5-1	No	299
4	2	5-1	Yes	2499
5	3	3-2-1	No	399
6	3	3-2-1	Yes	2699

Table 1: Agent information

We created a table where all six agents competed against each other; they played 10,000 rounds and ended up with these results:

Agent number	Points
1	40970
2	30629
3	44047
4	42572
5	45116
6	36656

Table 2: Agent scores

We also played all agents against each other one on one. They played 100 games where the winning agents get 1 point for a win and the losing agent gets 0.

Agent (Points)	Agent (Points)	Winning Agent
1 (0)	2 (100)	2
1 (23)	3 (77)	3
1 (0)	4 (100)	4
1 (15)	5 (85)	5
1 (0)	6 (100)	6
2 (100)	3 (0)	2
2 (59)	4 (41)	4
2 (100)	5 (0)	2
2 (43)	6 (57)	6
3 (0)	4 (100)	4
3 (15)	5 (85)	5
3 (0)	6 (100)	6
4 (100)	5 (0)	4
4 (37)	6 (63)	6
5 (0)	6 (100)	6

Table 3: Agent one-on-one scores

While the agents without threshold on the final output did a little bit better than the agents with this threshold on the six-player table, these results were extremely even, and they got crushed in the one-on-one games against agents with the threshold. Agents without this threshold were actually unable to win a single game against agents with the threshold. Based on these results, the most successful agent so far is agent number 6. It is possible that it outperformed agent number 4 because it had been able to run more generations. After seeing these results we wanted to test agent 6 against human opponents. However, the agent was still playing very aggressively, and a novice human opponent could easily beat the AI.

Conclusion

Poker proved to be a difficult challenge for this approach, and currently a really strong agent has not evolved. Perhaps the random nature of the game rewards bad decisions and punish the good decisions too often. However, the results do provide some pointers on what steps to take for future work.

The first step would be to further increase the number of games played between generations. While this would increase the running time of the program, it would help to eliminate the luck factor. It would also be interesting to see what results a larger neural network could produce. A 16-10-1 structure would require longer time to run, but could also learn more complex poker concepts. It may be interesting to see which inputs the best neural networks are using and which they are essentially ignoring. Also, currently the agents will not and cannot adapt to an opponents' play style. Having an input indicating how aggressive opponents are playing could turn out to be very useful.

In theory, in the long run, it should not matter if the final output of the neural network has a threshold or not. However, it seems to put the agents on the right path quicker, since it eliminates a lot of the all in and fold plays.

References

Anderson, J. R. (1983). *Machine learning: An Artificial Intelligence Approach*. R. S. Michalski, R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.). Tioga Pub. Co.

Azadeh, A., Negahban, A., & Moghaddam, M. (2012). A Hybrid Computer Simulation-Artificial Neural Network Algorithm for Optimisation of Dispatching Rule Selection in Stochastic Job Shop Scheduling Problems. *International Journal of Production Research*, *50*(2), 551-566.

Fleming, N. (2008). Game-playing Bots Take on Poker. *New Scientist*, *200*(2682), 28-29.

Fogel, D. B. (2001). *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann.

Ozcelik, F. (2012). A Hybrid Genetic Algorithm for the Single Row Layout Problem. *International Journal of Production Research*, *50*(20), 5872-5886.

Russell, S. J., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*, 3rd edition. Englewood Cliffs: Prentice Hall.

Warwick, K. (1991). *Applied artificial intelligence*. Institution of Electrical Engineers.