Machine Learning in Video Games: The Importance of AI Logic in Gaming

Johann Alvarez
1408 California Street, Tallahassee FL, 32304
jga09@my.fsu.edu

Abstract

Machine Learning is loosely described as "the study of systems that can learn from previously-known data." Also simply put: Machine systems can be "taught" to react to certain changes in data. For example, the internet search engine Google.com uses certain algorithms which allow the system to take previous searches given by the user, react, and give suggested results based on the user's input. In most modern video games of today's world, we can see much more advanced examples of how the AI uses previous knowledge to come up with strategies and techniques used to hinder or halt the player's progress to victory.

Introduction

Frequent players of games such as Starcraft can see how the AI responds differently depending on their previous actions in the game. For example, if the player is part of the Protoss faction, then they would need to build structures called pylons to be able to power their other structures. The AI knows this and would most likely use this knowledge to order its units to attack the pylons first.

But what if the player decided to build their pylons behind many lines of defense? The AI would not want to try and rush through the defense because this would likely mean a total wipe of its troops before they even reach the pylons. So, in learning from this new data, the AI would probably decide to wait and build stronger, more expensive units to try and destroy the lines of defense first. If the system decided to just rush in, the player wouldn't be challenged and the game would be too easy to stay enjoyable.

As you can see, it is indeed very important to implement the concept of Machine Learning into the AI systems used in modern gaming.

Background / Prior Work

In this section, we will provide a brief overview of some of the reasons as to why an AI's ability to 'learn' from an opponent's actions is key to giving the game a higher replay-ability factor, and how research in this field can help raise the success rate of games developed in the future. Software developers often need to consider the challenge an AI will provide and adjust the difficulty so as not to discourage the players from fighting against it.

Replay-ability

Currently, many video game agents use patterns in their actions that have become pretty easy to predict for human players. Because of this, these video games have lost their replay-ability, or the driving force to get players to replay games after they have already been completed or even just to get players to play them the next day. When the AI becomes too predictable, the game loses the player's focus and lowers their interest because it becomes less of a challenge.

We as humans have a natural instinct to compete, so when something becomes too easy to predict for someone, it becomes too easy to defeat and we lose the drive to fight against it. It gets to the point where the player's interest is only in completing the game and then afterwards throwing it away, because who wants to waste their time fighting something with the same predictable movements every time they are pitted against it? It becomes more of a chore than an achievement or accomplishment on our part. We as gamers strive to complete objectives and goals in a video game. This is also a part of our human nature. We want to

conquer a challenge that gives us a sense of pride in doing so.

So, what if the AI had the ability to 'learn' its opponents' actions and the ability to come up with a counter-strategy for these actions? To not just run out and use the same strategy for every situation, but to come up with better reactions to their opponents' moves and to make decisions that almost seem as if another human was making them? Obviously this would need to be balanced, or else the game would run the risk of also losing the player's interest due to a challenge being too difficult to complete (which is why we have difficulty levels). With an AI that can 'think', that can react just as a human would, almost every encounter with them would be unpredictable. Most humans learn from their mistakes and realize that making them again would most likely result in failure, so they usually would try to figure out other methods that would give them a better chance of completing the challenge given and attempt those instead. If an AI could do just the same, then the game becomes more challenging and raises the player's desire to play the game over, because every new clash with the AI would be a different experience.

Research can lead to Success

With modern video gaming gaining rise as an industry, the continued research of the game AI has become more and more critical to the success of these video games. Nowadays, most gamers tend to lead towards games that allow for online/multiplayer capabilities. The reason is not only to enjoy playing these games with others and to use them as another form of social medium, but also to quench the human urge to compete against each other. Humans have always challenged each other, be it for fame, for pride, or for power, since the era of cavemen. This natural need for competition has been passed into the gaming generation through the introduction of multiplayer gaming, because now humans have the chance to test their limits against opponents that think like them.

This is the reason why these online games have become so successful. Developers have realized this and that is why research continues to make the artificial intelligence appear smarter, more capable, and more *human*. A smart AI also allows for a game to stand out from the others, which can mean the difference between making and breaking a sale. Players do not want to buy a game in which the AI performs just like the ones from their other games. They want to have different and new challenges, all of which come from the development of an adapting AI. Developers want to explore new ideas that might take AIs to the next generation, an era in which games don't just provide an interesting opponent but one in which they can talk to the player, interact with the legions of online adventurers, and actually learn from game to game to be a more cunning and twisted opponent the next time around. Of course, these new AIs have to help make the game sell better, too. That's always the bottom line – if a game doesn't sell, then what good is the AI for?

Main Information

This section will focus primarily on the main topics of this paper. Machine Learning in game AI has become a very important part of game development. No longer is it something to just "fit" into the game if the frame-rate isn't hindered by the implementation. It's now become just as essential as the graphics or sound, if not more. Even so, with all the research being put into making smarter game AI, it still proves to be a tough chore. It's almost impossible to come up with every solution that a human might have to a situation. With every game that comes out with new revolutionary game-play, the amount of choices an AI can make becomes more overwhelming to program.

> To the human player, there might be two or three potential decisions which are "obviously" better—but what if the guy who coded the AI the Saturday night before the game's final version was sent to the publisher didn't think about those? The player sees the AI faced with a terrific decision upon which the entire fate of the game hangs—and it chooses incorrectly. Or worse than that, it chooses *stupidly*. A few instances of that and it's pop! The CD is out of the drive and the player has moved on to something else. [Buckland]

This is especially true when you play a video game with a cooperative AI. Take for example the 2013 game *The Last of Us*. The player plays from the perspective of a man named Joel who takes care of a girl named Ellie throughout a post-apocalyptic world. For almost the entire game (there are a few parts where the player switches perspectives), Ellie's actions are dictated by the AI. Fortunately for the player, the Naughty Dog studio did very well in designing the AI's reactions to certain events. If Joel is in trouble, she can help him out by throwing objects to distract the enemy. If Joel is hiding from a group of enemies, she knows that she also must hide and choose a safe location to do so while still being relatively close to the player. If Joel is running low on ammunition, she usually passes some along to him. Normally, most players would cringe at the notion of having to protect an AI, or even just having an AI partner, and with good reason. Most of these games (including *The Last of Us)* required that the AI partner stay alive in order to complete the game. If at any point the AI died, it would result in a "Game Over". This requirement

usually turned out to be quite an annoyance if the AI wasn't programmed to respond correctly to certain situations. Let's use *Resident Evil 6* as an example. I can't remember how many times I had to save Helena (one of the AI partners) from a hulking deformity because her AI couldn't seem to realize that the better thing to do would be to shoot it *from afar* rather than getting up close and allowing it to grab her, leaving me defenseless and having to try to run up and save her before she was eaten, squished, mutilated, etc.

This moment seems to be every gamer's nightmare. But developers have come up with different algorithms and practices that have evolved and have revolutionized video game AI mechanics so that these cases become rare. In the next subsections, we will go over some of these methods.

Genetic Algorithms

Genetic algorithms were formally introduced in the United States in the 1970s by John Holland at University of Michigan. Genetic algorithms are search heuristics that are designed to work in the same way as natural evolution. These algorithms generate solutions to optimization problems using techniques inspired by our world's natural evolution, such as inheritance, mutation, selection, and crossover. In a genetic algorithm, a number of candidate solutions to these optimization problems are evolved towards better solutions.

A typical genetic algorithm requires:

1. A genetic representation of the solution domain.
2. A fitness function to evaluate the solution domain.

To use a genetic algorithm in a program, the potential solutions must somehow be represented as a collection of sort of "digital" chromosomes (represented as bits) – the same way our bodies use genomes as a blueprint of our DNA. Once these solutions are encoded, a randomized population of these chromosomes is evolved over time by "breeding" the best fit solution and adding slight mutations to them so that eventually, the result is a convergence of the fittest individuals of the original population. Generic algorithms don't always have to generate a solution, however. Using these algorithms might not even generate the best solution to the problem given. But, the best thing about using a genetic algorithm is that you don't need to know how to solve a problem. All you need to do is encode it in a way in which the genetic algorithm mechanism can utilize and evolve it into a suitable solution. Below is a loop used in order to form a generic algorithm:

Loop until a solution is found:

1. Test each chromosome to see how good it is at solving the problem and assign a fitness score accordingly.
2. Select two members from the current population. The probability of being selected is proportional to the chromosome's fitness—the higher the fitness, the better the probability of being selected. A common method for this is called Roulette wheel selection.
3. Dependent on the Crossover Rate, crossover the bits from each chosen chromosome at a randomly chosen point.
4. Step through the chosen chromosome's bits and flip dependent on the Mutation Rate.
5. Repeat steps 2, 3, and 4 until a new population of one hundred members has been created.

End Loop

[Buckland]

This entire loop is known as the *epoch*. If you notice in the algorithm, the loop includes a step in which you would need to use a method such as the Roulette wheel selection. This particular method works by first making a total fitness chart for the whole of the population and representing this information as a pie chart, or roulette wheel (hence the name). Once this is made, you assign a slice of the wheel to each member of the population. However, each slice is proportional to the fitness score of the member, so if the member has a higher fitness score, then it would have a better chance of being selected. To choose a chromosome, you simply need to "spin the wheel", "throw a ball" into the wheel, and pick the member which is associated to the slice the ball ended up on. And that's it for the Roulette wheel selection A quick explanation of the Crossover Rate is the probability to which two chosen chromosomes would swap their bits to produce two new ones, and the Mutation Rate is the probability to which a bit within a chromosome would be switched or flipped from 0 to 1 or vice-versa.

Now, knowing the terminology and the algorithm, how would this method help the game AI evolve to think and to find better solutions to their situation? Let's come up with an example. Suppose we have a maze in which Pac-man has to navigate through from a start to a finish. First we would encode every direction in which Pac-man could travel: up, down, left, and right (or North, South, West, and East). Then you must make the population of random chromosomes that would give Pac-man directions to follow (a sample chromosome, 111110011011, would decode to "11, 11, 10, 01, 10, 11", or "3, 3, 2, 1, 2, 3", which if we had North = 1, South = 2, West = 3, East = 4, then the directions Pac-man would head to would be as follows: "West, West, South, North, South, West"). Setting Pac-man at the start and allowing him to follow the directions might lead him to the goal, but in a population of

hundreds of chromosomes, normally, the odds to pick from the ones that would do so would probably be unlikely. However, if we first use the genetic algorithm to test each chromosome and see how close each one would get Pac-man to the exit (if it reaches, it would be considered a solution), then we can breed the better solutions in the hopes of creating offspring chromosomes that would let Pac-man get even closer to the goal. And finally we could continue using this method until a solution is found (or until Pac-man eventually becomes stuck in a corner, which can happen). Remember, generic algorithms aren't perfect, but they do help in increasing the chances of finding the right solutions and sometimes the *best* solutions to a given situation. Using genetic algorithms, therefore, can greatly improve the game AI's logic and allow for human-like decision-making.

Neural Networks

The human brain is our biological neural network. What developers want to accomplish is to create a suitable artificial neural network for their game AI – one that operates in much the same way as our own network. There are 5 remarkable properties which our brain contains:

1. It can learn without supervision.
2. It is tolerant to damage.
3. It can process information extremely efficiently.
4. It can generalize.
5. It is conscious.

An artificial neural network attempts to mimic this amount of parallelism within the constraints of a modern computer. In doing so, it should display a number of similar properties to a biological brain. These artificial neural networks (abbreviated ANNs) are built the same way as our natural brains; where our human brains use neurons to send information, the ANN uses similar building blocks called artificial neurons.

These neurons, just like our own, take in different inputs which each have a specific floating-point weight assigned to it. These weights determine the overall activity of the neural network. If we have a positive weight, then it can exert an *excitory* influence over the input. If it's negative, it can exert an *inhibitory* influence. When the inputs are taken in, they are multiplied by the weights associated to them. Then they all convene as the nucleus of the artificial neuron, which contains an *activation function*. The function sums all these weight-adjusted input values together to get the *activation* value (which is also a negative or positive floating point number). If this final value is above a certain threshold, then the neuron would output a signal which associates to one. If it is below the threshold, the outputted signal would associate to zero. This function in particular is considered a *step function*, one of the simplest types of activation functions which are found in the nucleus of artificial neurons.

So, what can you do with these artificial neurons? Well, we obviously need to connect these together to make an artificial neural network. There are many varied ways of connecting neurons but the most widely used and easiest to understand is by connecting the neurons in layers. This type of neural network is called a *feed-forward network*. Each layer of neurons feeds their output to the next layer and so on and so forth until a final output is given. A feed-forward network consists of an input layer, one (or more) hidden layers, and an output layer. There can be any number of neurons and hidden layers in a network; however, it's desirable to keep the network as small as possible because the speed of the network decreases as more of these are added.

Artificial neural networks are usually used for pattern recognition. The reason being is because ANNs are great at mapping an input state to an output state (a pattern it's trying to recognize can be mapped to a pattern it's already been trained to recognize).

Let's take a look at handwriting recognition, for example. For each character, the network is trained to recognize many different versions of that letter. Eventually the network will not only be able to recognize the letters it has been trained with, but it will start being able to generalize. Basically, if a letter is drawn slightly differently than the letters in the training set, the network will still stand a pretty good chance of recognizing it. It's this ability to generalize that has made the neural network an invaluable tool that can be applied to a variety of applications, from face recognition and medical diagnosis to horse racing prediction and, leaning closer to the topic of this paper, AI navigation in computer games.

For example, we can use neural networks to 'train' game AI to follow certain patterns in movement and positioning in an AI-driven Ping Pong program. The training occurs by first shooting the ball from the center with a random direction and a fixed speed. The neural network is given the position and the direction of the ball and the y position of the paddle as input. The output would be a y direction in which the paddle should move in order to make contact with the ball and send it in the other direction. The weights are initially made to be random values, but as generations (or loops) pass, these weights will change to fit the situation. The network will soon learn to move the paddle in the same direction that the ball is heading. After several thousand generations of training, the network would eventually learn to play perfectly (the exact number of generations to play perfectly varies because the initial

weights are random). In this example, by varying the level of training, the computer opponent can vary from poor play to perfect play. If we train the network for some number of iterations up front (say 1000), and then train the network an additional 100 iterations every time the human player wins, eventually we would have a perfectly controlled computer opponent whose difficulty we can change by altering either the up-front iterations or the rate of learning after each win against the human.

Method of Evaluation / Experiment

In this section we will evaluate the performance of the Generic Algorithm and the Artificial Neural Network and go through a brief explanation of my experiment.

We'll look at some of the pros and cons of using a genetic algorithm. The generic algorithm uses local minima and maxima. It is only one algorithm but can have various data representations. It is stochastic, meaning that it often requires a lot of tweaking, but sometimes you can tweak it as much as you want and it would still find the same result. There are no gradients or fancy math involved with genetic algorithms. They are also easily parallelized and also easily customized as well.

However, because of this, designing an objective function can be difficult. Unfortunately, the genetic algorithm can be computationally expensive depending on the amount of chromosomes that are implemented, but they are still better than most gradient search methods and are less likely to get stuck on a local high or low because they traverse the search space using the genotype rather than the phenotype (in other words, instead of traversing using specific blueprints, they follow the general schematic).

Now we'll move on to the pros and cons of using artificial neural networks. Here are some key benefits:

- It is very easy to apply ANN to problem domains where the relationships are quite dynamic or non-linear among the input and output.
- Since ANN is capable of capturing many kinds of relationships and complex patterns among data, ANN allows user to easily model the system which otherwise is very difficult or impossible to represent through traditional modeling approaches.
- The training information is not stored in any single element but is distributed in the entire network structure. This make ANN fault tolerant and it reduces the impact of erroneous input on the result.

[Mistry]

Some of the ANN's cons have resulted in the following from experimentation:

- Neural networks are not magic hammers. Contrary to where ANNs are believed to solve any machine learning problem, sometimes they are applied indiscriminately to problems for which they are not well suited.
- Neural networks are not probabilistic. For example, a neural network might give you a continuous number as its output (e.g. a score) but translating that into a probability would be difficult.
- Neural networks are not a substitute for understanding your problem. If you are building a classifier, it's usually better to spend your time visualizing your dataset and selecting the best input features using whatever domain-specific knowledge you have available to you, rather than throwing a neural network at your data and hoping for the best.

For my experiment, I have created a simplified Five Card Stud Poker game in which the player and the AI bluff against each other to win. At the end of every round, the hand of the winning agent is shown. The AI keeps track of the ratio at which the player bluffs (a bluff is considered going all-in on a hand with a score less than a straight) using a Bluff Percentage (BP) variable. My method of AI learning is a simple Artificial Neural Network, where the input is the winning hand and the player's bet, and the output is the Bluff Percentage. If the player scored less than a straight, then the ratio of amount of times player has bluffed over the amount of total rounds changes, and this is reflected in the Bluff Percentage. The AI starts with the strategy of folding on an all-in raise from the player, but at the start of each round, if the BP has changed to be higher than 60%, then the AI would switch its strategy to try and call the player's bluff.

Results

In this section we will quickly go over the results of my experiment and we'll go into explanation of how the ANN became my choice for the AI system.

The AI had a bit of trouble with the first few tests. The programming of the ANN was a little more confusing than I had anticipated. The ANN and AI logic in my experiment is not complete, so the results of these tests cannot be taken with full merit. However, they should be enough to show how using this algorithm and not an algorithm such as, let's say, the alpha-beta pruning system used in the Dots for the AI conference paper written by Joseph Barker and Richard Korf, would provide me with better results than if I had chosen the latter. Originally I had wanted to use a similar algorithm in which the AI would use a simplified

version of the Alpha-Beta algorithm. However, this algorithm stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Since I needed an algorithm that would continue viewing all possibilities even when a better one was found, because a player's bluff percentage wouldn't guarantee that the player would be bluffing the next round, I decided that ANN would be the best choice. With an increased time allowance, there might have been a way to improve both algorithms by combining them together in a new form. Even so, the AI had not reacted as well as I thought it would (perhaps I had not given it enough iterations through the rounds to better anticipate the player's actions). Sometimes it wouldn't even go all-in with the player even if the player had an extremely high chance of bluffing (the player's BP was above 90%). However, I believe that with more preparation and with the use of more iterations to help improve the AI's learning curve, the results would become more beneficial to the research in this field.

Conclusion / Future Work

In this paper, we have gone over a couple of the methods widely used to allow for machine learning in AI, particularly in video game AI. In modern computing, the advancement of artificial intelligence has become a necessity, now that more complex events are being implemented into modern game-play situations. Because of this, research in the field has greatly increased in the last decade. New advancements have resulted in more efficient algorithms and subsequent application of these methods in modern game programming. The purpose of this paper was to enlighten and educate the reader in the value of research in AI logic in gaming, to address the importance of this study, and to promote exploration in the topic. The results from my experiment might not have been as accurate as I had hoped because of lack of preparation, but they still clearly demonstrate how the advancement of machine learning is essential to the improvement of game procedures and logistics.

Acknowledgements

I would like to thank my family for continued support in the research of this paper, my friends for giving me the ideas needed to push me towards my goal, and the professor for giving us this assignment to learn the rigors of writing an Artificial Intelligence conference paper for submission.

References

[1] Barker, Joseph K., and Korf, Richard E. 2012. *Solving Dots-And-Boxes*. AAAI Publications, Twenty-Sixth AAAI Conference on Artificial Intelligence.

[2] Buckland, Mat. 2002. *AI Techniques For Game Programming*. Cincinnati, Ohio.: Premier Press.

[3] Fabian, Nathan. 2008. *Machine Learning of Human Behavior in Interactive Games*. Albuquerque, New Mexico.: University of New Mexico.

<http://www.cs.unm.edu/~ndfabian/behaviorthesis.pdf>

[4] Wall, Matthew. *Introduction to Genetic Algorithms*. Mechanical Engineering Department.: MIT.

<http://lancet.mit.edu/mbwall/presentations/IntroToGAs/P001.html>

[5] Macri, Dean. 2011. *An Introduction to Neural Networks with an Application to Games*. Intel.

<http://software.intel.com/en-us/articles/an-introduction-to-neural-networks-with-an-application-to-games>

[6] Mistry, Kamalkumar. 2012. *Intelligent Complex Event Processing with Artificial Neural Network*. SYS-CON Media, Inc.

<http://www.sys-con.com/node/2459059>