Using an Artificial Neural Network to detect aim assistance in Counter-Strike: Global Offensive

Kasey Maberry
New Mexico Tech
kmaberry@nmt.edu

Steven Paustian
New Mexico Tech
spaustia@nmt.edu

Sheikh Bakir
New Mexico Tech
sbakir@nmt.edu

ABSTRACT

This work presents a new approach to detecting cheating in the computer game Counter Strike: Global Offensive. With the growth in popularity of online computer games, an increasing amount of resources to detect and eliminate cheaters is needed, as playing against a cheater causes frustration and players could ultimately quit playing if they become too frustrated. Not only that, but the integrity of the professional environment of a competitve game can be ruined if a player is found to be cheating. Most current methods of detecting cheaters are done client sided, and can be circumvented. Examining the state-of-player, it was observed that research exploring the Artificial Intelligence application to this goal becomes relevant. This work shows the usage of artificial neural networks (ANN) applied in a First Person Shooter (FPS) to detect cheaters using aim assistance. In this work we apply a Multilayer Perception (MLP) and Learning Vector Quantization (LVQ) neural networks to detect players using aim-assistance through cursor movements. The network successfully detects aim-assisted kills from a testing set on the order of 98% .

Keywords

LVQ; MultiLayer Perceptron; Backpropagation; aim assistance; Learning Vector Quantization; cheat detection; Counter-Strike; Global Offensive

1. INTRODUCTION

Counter-Strike: Global Offensive (CS:GO) is an highly popular team based competitive FPS (First Person Shooter) online video game. The rules of the game are as follows: The game is played in rounds, where each team (Counter-Terrorist and Terrorists) has 5 players each and the rounds last for 30 rounds with the first team to 16 rounds won being the victor. Each round a team can either win by killing all of the enemy players or completing an objective such as planting the bomb if they are a terrorist and defusing the bomb if they are a counter-terrorist. When a player dies, they do not respawn until the round is over, then the entire team respawns and they are allowed to purchase weapons and other gear. Players earn money by winning or losing the round, completing objectives, and killing enemies. On average, each player will have around 25 or 30 kills in a game that runs for the full 30 rounds, however the minimum number of rounds in a match can be as low as 16, and in professional events a match can last an unlimited amount of rounds due to overtime if the game is tied at 30 rounds.

Cheating is an important issue in FPS games as it is technically easy to do with a high potential of reward in the professional environment. In 2014, a professional player Hovik "KQLY" Tovmassian received a Valve Anti-Cheat ban[3] while he was on one of the top professional CS:GO teams and many lower tier teams have cheat accusations constantly. This shows that there is a need for a anti-cheat that is highly effective in catching cheaters. Currently, there are four basic systems used for cheater detection. The first is a signature based anti-cheat which scans the game memory and files on a user's computer to try and detect cheat signatures corresponding to a cheat signature database. This is the most accuracte form of anti-cheat and if the anti-cheat programmer knows how the cheat works. However, the main issue with this form of anti-cheat is the most effective ones require elevated privelages on a users computer and scans every file on the computer, which is a major privacy concern. The current implmentation of this in CS:GO is known as Valve Anti-Cheat (VAC). VAC is limited to the scope of the game, and typically does not scan a users files. The most prominent and controversial client for CS:GO is known as the E-Sports Entertainment Association League (ESEA) client, which will scan all of the files located on a users computer. The next form of cheat detection is statistical, where statistical information (accuracy, headshots, reaction time, etc) is gathered about the player and compared with a predetermined threshold. If the player doesn't fit in the thresholds of a real player, they are usually kicked from the game. CS:GO currently does not include any stastical anti-cheat.

Next, there are peer review systems where players observe other players and try to decide whether or not they are cheating. Peer review is highly effective for obvious cheats as it is visually very easy to tell if a player is cheating. The only issue with this system is if it is a very good player it is possible that they are convicted as a cheater, or a very good cheater might not get banned. There are a couple of instances of professional players receiving bans from this system, such as when a player, ScreaM, known for incredible accuracy received a ban on one of his alternative accounts[4].

[redacted]

Lastly, limitations are coded into the game to prevent players from being able to exploit it, such as players do not receive information about others unless they are close enough to each other and the game calculates that they should be visible to each other.

The network that was used is capable of only catching users aim-assistance under the hypothesis that a real player will have small deviations in mouse movements, while a cheater will have a relatively straight line without deviations in the path. It may be possible to move onto cheats which allow players to see through walls, however this will be much more difficult to implement.

We will be using the Learning Vector Quantization (LVQ) and Multilayer Perceptron(MLP, also known as Backpropagation) networks as they are predicted to be some of the best networks for what we are trying to due to them genereally have a very high correct classification rate.

2. PREVIOUS WORKS

A study done by Lui and Yeung [8] showed that this type of system can have a very high detection rate, and can run on the server that the game server is running on with double the resource usage compared to just running the game server. The authors also commented on Counter-Strike, stating that the system could easily be transferable as it already collects much of the data they programmed into their server, meaning very little additional overhead. Another improvement that Lui and Yeung point out is that the signature based and statistical based anti-cheats run client side while a neural network runs on the server and has the capability to be incredibly accurate. Another study [5] also shows that using neural networks can be very efficient at detecting cheats, especially when it is testing for a specific cheat instead of multiple cheats at once.

3. HYPOTHESIS

We believe that a neural network could provide an accurate way to detect aim-assistance cheating. The network would be able to detect cheaters with a faster response time than current methods, and allow a system to detect cheaters without needing prior knowledge regarding the implementation of the cheat; something signature based and statistical based anti-cheats require. Currently, the fastest form of anti-cheat is statistical detection (Counter-Strike does not have this form of anti-cheat), followed by peer reviewed cases, and lastly signature based being the most accurate but slowest to reprimand cheaters as it needs to know exactly what the cheat modifies. When it comes to a cheat that has already been studied and implemented into the signature based anti-cheat, it will convict the user much faster than the others as it continuously scans the memory while the player is playing and instantly reprimands players once it detects an abnormality. For a newly detected cheat, sometimes VAC will flag an account for a week or two before banning an individual, that way cheaters will believe the cheat is safe and tell other cheaters to use it.

We will implement a LVQ and MLP network to detect players using aim assistance by observing their cursor movements, then the acceleration and velocity will be calculated. The input vectors will be encoded with N numbers of cursor positions before a kill, the velocity and acceleration of the cursor movements. We will also append several vectors of these kills together for a temporal analysis of the data. The final vector will be partitioned and propositionalized using random forests. [6] The neural network framework will be provided by Weka.[7]

4. IMPLEMENTATION

4.1 Data

Data points were collected from CS:GO demo files recorded on a local server and on an aim training map[2]. Demos are files which contain all of the server updates for that particular match. This was chosen so we would have a set of perfect data with only cursor movements, to test the validity of our claim. Demos of real matches were also collected, however due to an issue it was not possible to use these for testing and training. A total of four demos were collected, three of which in the medium skill were recorded by the research group, and the fourth of a high skilled player was provided by Igor ”noaki” Dobrzański. Medium skill level refers to the Master Guardian I rank (64.7 percentile) from Valve's ranking system, and the high level player corresponds to the Global Elite rank (99.4 percentile). Each demo contains about 600 kills, for a total of about 2400 data points. The demo files were broken up into the following categories:

1. Medium skill level legitimate player
2. High skill level legitimate player
3. Medium skill level obvious cheating player
4. Medium skill level subtle cheating player

Which totals up to be about 1200 legitimate player data points and 1200 cheating player data points. Splitting up the data into high level and medium level player data was chosen to test sensitivity for difference in player skill level. Both demos were recorded at the players usual setting for obtaining data as close to what would be expected from a player of that skill level. Changing the nature of the cheating player was chosen to detect how blatant of a cheater the network could catch, it was necessary to test whether or not the network would be able to detect cheaters of various levels. The demos of cheating players where recorded on an alternative computer due to the illicit nature of cheats and the possibility of the user receiving a ban on their main account or at the very worst receiving a compromised computer.

All demos were recorded using the AK-47 for the weapon with the player only going for headshots to try and mimic a player with aim-assistance. It is expected that using the AK-47 will allow the network to detect cheaters using any weapon, with the only possible exception being with a sniper rifle due to a vastly different playstyle, however it is expected that even using a sniper rifle wouldn't cause any difference as the cursor movement should still be unnatural.

4.2 Data Parsing

In order to get cursor positions from binary demo files, we:

1. Created a text file dump of the data contained in the demos using demoinfo-go, an open source tool provided by Valve Corporation.[1]

2. Created a parser to search for player deaths, get the attacker's server id, then record the last N cursor positions.

3. Calculate the velocity and acceleration of the cursor movements against the server's clock.

4. Output a vector for input to the neural networks.

The text dump created by demoinfo-go included cursor positions.[1] The cursor positions were contained in the variable an **m_angEyeAgles** where **[0]** is the x-coordinate and **[1]** is the y-coordinate.

Here is an example of the a player death being noted.

```
1  ---- CNETMsg_Tick (12 bytes) -----------------
2  tick: 13424
3  host_computationtime: 13668
4  host_computationtime_std_deviation: 1261
5  host_framestarttime_std_deviation: 1277
6  Entity Delta update: id:1, class:35, serial:763
7  Table: DT_CSPlayer
8  Field: 0, m_flSimulationTime = 26
9  Field: 1, m_nTickBase = 13427
10 Field: 18, m_angEyeAngles[0] = 355.429688
11 Field: 19, m_angEyeAngles[1] = 163.476562
12 Field: 93, m_flCycle = 0.229645
13 player_death
14 {
15  userid: Graham (id:55)
16   position: 498.487183, -94.961426, 64.031250
17   facing: pitch:355.429688, yaw:163.828125
18   team: T
19  attacker: Cheater (id:1)
20   position: 2.000000, 2.000000, 64.031250
21   facing: pitch:0.000000, yaw:348.046875
22   team: CT
23  assister: 0
24  weapon: ak47
25  weapon_itemid: 0
26  weapon_fauxitemid: 18446744069414584327
27  weapon_originalowner_xuid: 76561198262805437
28  headshot: 1
29  dominated: 0
30  revenge: 0
31  penetrated: 0
32  noreplay: 1
33 }
```

We see that line 13 indicates a player death has been recorded. The attacker has an id of 1 from line 19. Line 6 indicates the player with id 1 had x and y cursor coordinates as 355.29 and 163.47. By parsing for the previous N cursor coordinates, we can calculate the velocity and acceleration of the player‘s cursor movement milliseconds before the kill.

4.3 Encoding

We tested several vector encoding schemes for cheat detection. First, we tested N raw cursor positions before a kill. Second, we tested N raw cursor positions with the velocity and acceleration of cursor movement. Finally we tested raw cursor positions with velocity and acceleration for 3 kills appended together.

These vectors were filtered by partitioning and propositionalizing using random forests. This was accomplished through the Weka‘s supervised attribute filter *PartionMembership* using the J48 algorithm.[6][7]

5. IMPLEMENTATION

Weka was used to train and test both MLP and the LVQ networks.[7] The networks were standard and included with the package.

5.1 LVQ

The following parameters have been used for training the LVQ network.

- Learning rate: 0.3
- Training time: 10000 epochs.
- Codebook vectors: 10000
- function: Linear Decay.

5.2 MLP

The following parameters were used for training and testing the MLP network after experimentation:

- Learning Rate: 0.3 with auto decay.
- Momentum: 0.05
- Training time: 1000 epochs
- Hidden Layers: $\frac{\text{Number of attributes}-2}{2}$

6. RESULTS

The following results were obtained after training and testing the networks using the parameters defined in section 5.

We chose a 75% 25% split of our vector totals for training and testing the vectors described in section 4.3.

6.1 LVQ

N	C	C, V, A	C, V, A, 3
5	57% / 42%	79% / 20%	91% / 8%
10	60% / 39%	90% / 9%	94% / 5%
20	76% / 23%	92% / 7%	97% / 2%

Table 1: Results for LVQ. Format is % Correctly Classified / % Incorrectly Classified. N = Number of previous cursor positions, C = Raw cursor positions, V = velocity of cursor movement, A = Acceleration of cursor movement, 3 = Three vectors appended together.

6.2 MLP

N	C	C, V, A	C, V, A, 3
5	73.2% / 32.6%	86.2% / 13.8%	93.9% / 6.1%
10	71.5% / 28.5%	88.5% / 11.5%	97.1% / 2.9%
20	81.2% / 18.8%	93.8% / 6.2%	98.0% / 2.0%

Table 2: Results for MLP. Format is % Correctly Classified / % Incorrectly Classified. N = Number of previous cursor positions, C = Raw cursor positions, V = velocity of cursor movement, A = Acceleration of cursor movement, 3 = Three vectors appended together.

7. DISCUSSION

Both LVQ and MLP performed well at classifying cursor positions before a kill as aim assisted or legitimate. Classification accuracy generally increased with N, when acceleration and velocities of cursor movements were added to the vector, and when kills were stacked together.

Partitioning the vector was critical to our success. Non-partitioned vectors was generally 20% less accurate than partitioned data.

The training time for MLP networks was nearly 20 minutes for N=20 vectors. The training time for LVQ networks was about 5 minutes for N=20. Classification of vectors using MLP was approximately 2ms. Classification of vectors us LVQ was 30 ms.

At N=10 and N=20, MLP reported 0 false positives. All instances of mis-classification were false negatives. This indicates MLP would be an excellent choice for a real world classifier because in a live environment our classifier would ideally have near-zero rate of banning high level players accused of cheating.

Like MLP, partitioning the vector was very useful for LVQ. We tested by varying the parameters for LVQ. For learingin rate, we tried to use a value between 0.2 to 0.5 and we found 0.3 is the optimal value of considering the training time and accuracy. For LVQ the most important parameter is codebook vectors, it is seen that accuracy increases if we increase the value of codebook vectors. Initially the codebook vector was chosen very low (20). But for low codebook vectors the result was not very promising, so we decided to increase the value of it. One problem of increasing the codebook vector is it increases the training time.

8. CONCLUSION

MLP and LVQ neural networks are highly amenable for classifying a user‘s cursor movement as legitimate or aim assisted over time. With a properly encoded vector, and properly trained network, correct classification rates can be as high as 98%. Due to MLP‘s low classification time, it would be a good candidate as a real time cheat detection system. It is seen that LVQ could also be a good choice, and running data through both of them could help increase certainty in whether a player is cheating. Though the LVQ correct classification rate is a little bit lower than MLP (97% instead of 98%) but if we look the training time then LVQ performance is good. However, MLP and LVQ both could be used in real system based on the system requirements.

9. FUTURE WORK

Future work may include a number of things. Classification may see an improvement in both networks by adding $y = mx + b$ attributes to the vector. This may be effective because the cursor movement of aim assisted users is rarely non-linear, unlike legitimate users whose cursor movement is usually parabolic in nature.

Extending cheat detection by a neural network to wall hack detection would be a powerful new feature. This may be difficult to do, as there are no known variables in demo files that directly relate to a user's ability to see through walls.

A logical extension of this project is to implement a real time aim assist detection system which isn't reliant upon demo files in order to classify cursor movement. This can be implemented as a CS:GO server add-on and operate in real time without resource overhead to clients.

10. REFERENCES

[1] demoinfo-go. https://github.com/csgo-data/demoinfogo-linux. Accessed: 2016-05-05.

[2] Fast aim/reflex training map [sp vs bot map] / aimtraindriving. https://steamcommunity.com/sharedfiles/filedetails/?id=368026786. Accessed: 2016-03-28.

[3] Hltv. kqly handed vac ban. http://www.hltv.org/news/13636-kqly-handed-vac-ban. Accessed: 2016-05-05.

[4] Scream twitter message. https://twitter.com/g2scream/status/607502183869771776. Accessed: 2016-05-05.

[5] H. Alayed, F. Frangoudes, and C. Neuman. Behavioral-based cheating detection in online first person shooters using machine learning techniques. *Computational Intelligence in Games (CIG) (Conference)*, pages 1–8, August 2013.

[6] E. Frank and B. Pfahringer. Propositionalisation of multi-instance data using random forests. In *AI 2013: Advances in Artificial Intelligence*, pages 362–373. Springer, 2013.

[7] G. Holmes, A. Donkin, and I. H. Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE, 1994.

[8] S. F. Yeung and J. C. S. Lui. Dynamic bayesian approach for detecting cheats in multi-player online games. *Multimedia Systems*, 14(4):221–236, September 2008.