DETECTING PLAYERS' POSITIONS IN COMPETITIVE VIDEO GAMES USING NEURAL NETWORKS

Ngan Nguyen
Computer Science Department
Columbia University
Manhattan, New York
ntb2134@columbia.edu

Faculty Adviser: Dr. Markus Eger
Computer Science Department
California State Polytechnic University, Pomona
Ponoma, California
meger@cpp.edu

***Abstract*—Conster Strike - Global Offensive (CSGO) is a first person shooter game with two opposing teams: terrorists and counter-terrorists. In the most popular game mode, the terrorists' goal is to plant and successfully deploy a bomb, while the counter-terrorists' goal is to defend the bombsites until time runs out or through defusing the bomb. The alternative win condition is to eliminate the other side. For this project, we examined whether player's positions can be determined through network data. Network data is sent from the server and appears as a nonsensical combination of characters. If we can determine player's positions through this data, it suggests a flaw in the network data that cheaters may exploit for an unfair advantage. Our method is to create a machine learning model that can predict players' positions through network data. To test and train our data, our dependent variable is players' positions data, which we extracted through a public parser on Github. Our independent variable is network traffic data, which we extracted using Wireshark. Our resulting model can predict the relative position of the opposing team. Given the low time to kill characteristic of CSGO, knowledge of opposing players' positions is a significant advantage. In conclusion, we can possibly use this model for a proof of concept cheat that mod makers may or have already made. This proof of concept can be shown to the developers in order to fix this flaw and keep their game fair for all players.**

I. Background - Motivation

Gaming is a growing industry, it has been growing consistently and blooming in the last few years. According to Mordor Intelligence, the global gaming market is forecast to worth 314.40 billion dollars by 2026 [1]. Cheating has been a big problem in highly competitive multiplayer games. The incentive is there to cheat, as success in the game gives a sense of reward, attracts followers to streamers, or win tournaments with prizes. The black market for cheats grows with the gaming market. Some common cheats are: Exploits, Probability Prediction, Aimbot, TriggerBot, and Wallhacks. Generally, cheating in games is when user takes advantage of design flaws and advance knowledge of the game's characteristics to extract information or leverage their chances to win the game.

Fig. 1: Player's view during game

CSGO is a multiplayer first-person shooter developed by Valve and Hidden Path Entertainment. CSGO is the most played game on Steam, the gaming platform with the highest user base. According to Steam chart, the game has average of 617,397.9 players with peak of 1,028,840 players in the last 30 days. It provides players with a map but it does not show opponents' positions unless they shoot their weapons. Knowing the positions would give player a huge advantage in deciding how to approach and peak critical choke points and areas. A typical competitive game would have 10 players divided in 2 team: terrorist and counter terrorist. In the most popular mode, terrorists goal is to setup the bomb and counter terrorist are trying to guard bombsites or deactivate the bomb. The alternative win condition is to kill everyone on the opposing side. Players have 30 rounds per game with less than 2 minutes for each round. The contribution we present in this paper consists of:

1) We found correlations between network data receiving from Valve server and players' positions in game; and

2) We were able to construct a prediction model to generate preliminary results of random selections from network data bytes collected to retrieve players' positions.

Fig. 2: All players' position after parsing with Demo Analyzer program [2]

II. Related Work

Our work investigates possibility of using advantages in knowledge of games' characteristics and network data for cheating in video games. This topic has previously been explored by various authors. Following are a few examples of research that inspired our project:

Jonnalagada et al. describes how CSGO's current cheat detection works. Specifically, the Deep Neural Network is a Vision Based Cheat Detection. This cheat detector outputs probability for visual hacks, uncertainty of estimate, and a visual map with probability of cheating. They took into account the confidence of the DNN output allows avoiding false positives while keeping the cheating detection rate high. The main limitation of this cheat device is, this detector is only vision-based and in case that the players run their cheat program using a separate machine / a side-channel, then the detector will not be able to detect [3]. In fact, side-channel attacks in video games is a problem that easily go undetected. Kocher et al. presents Spectre attacks, which proves the possibility of adversaries can access private memory and contents without require any software vulnerabilities. [4] Another research also explores the link of packet-level network traffic and information local machines receives from game server. Dainotti et al. studied distributions of UDP data and use statistical analysis to examine the dynamics between network packets within a game session and its characteristics [5].

Our main focus is to find a way to predict players' positions and gain advantage in the game without attacking the game itself in anyway. Our approach is to use network data that Valve server sends to our local machines and a public CSGO parser library. Further information about our procedure is discussed in the next section.

III. Procedure

In short, our approach consists of four parts:

A. Collect replay files and extract locations of players;
B. Record network traffic received from Valve server and;
C. Align network packets to their prospective corresponding players' positions using timestamps;
D. Find a correlation between the two using a neural network.

A. Ranked Competitive. Ready. Go

For the first part of the project, we use CSGO downloadable replay feature to extract player positional data. We use a parser library to extract information from the collected replays. To be specific, the information we need are: players' ID, their x and y coordinates, and the timestamps down to milliseconds. One challenge is the available CSGO parsers required many alterations in order to work with the current version of CSGO. From Demo Analyzer [2], we can retrieve players' positions, but the parser does not give timestamps. As a tick in CSGO demo is 1/64 of a second, we get the timestamp that demo file is last modified and use the tick count provided by Demo Analyzer to trace back the exact timestamps for each tick. Missing players require special consideration. At the start of demo file, sometimes there are not enough players and we have missing values on our players' positions data. We replace these missing positions with zeroes considering this period is rather too short and does not affect the entire data set. We also normalize the values of players' positions. We use `TORCH.NN.FUNCTIONAL.NORMALIZE` library, which simply operates shift-scale on the values. We call this `Player_Position_set`

B. Sniffing your network data

The second part of the project is to collect network traffic received from Valve server for the entire duration of the game. For the project, we use Wireshark. To filter network data our local machine receives from Valve server, we use NetRange: `162.254.192.0-162.254.199.255` [6] to access network bytes, lengths of each packet and timestamps of received packet down to milliseconds. Keep in mind that although Wireshark allows users to export file as `.csv` files, however, data might not be fully exported. Instead, we export the data to `.json` files, then flatten the `.json` files and convert network bytes from each packets from hexadecimal

values to decimal values to prepare the data for the learning process, where these data need to be in numeric form to be used in tensors. We also divide each byte by 256 to normalize the data set since a byte can represent 256 individual characters.

Another challenge is that the lengths of packets received from Valve server varies. We use the same method, we find the length of the longest packet to set as standard and add zeroes to each packet's bytes' values to ensure all packets have the same length. We call this `Network_byte_set`.

C. Alignment

Our third main part is to align these two separate sets of data. Using the timestamps, we align network bytes with the corresponding players' locations. Players' positions data at t (time) should be connected to network byte data at time m that is the smaller and is the closest value to t. In the diagram in Figure 3, we demonstrate this process

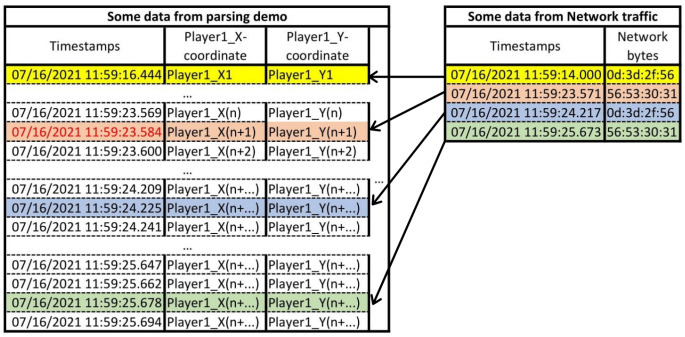

Fig. 3: Alignment data using timnestamps

D. Artificial Neural Network (ANN)

In order to analyze the correlation between the network bytes the players' positions, we use an artificial neural network.

Using the normalized data described in subsections III-A and III-B, aligned using the process outlined in subsection III-C, we then randomly split them into two subsets: 80% as training set to Neural Network model: `Train_Network_byte_set` and `Train_Player_Position_set`; and 20% is used as performance evaluation set: `Test_Network_byte_set` and `Test_Player_Position_set`. Specifically, `Train_Network_byte_set` is our sample set for training and `Train_Player_Position_set` contains the target values that theoretically, the neural net would recognize the patterns and cast predictions. In IV, we will further discuss our method of validating the generalization ability of the model using an entire separate network data set associated with a separate game match.

Following is the the architecture of the ANN we have: Since for our network traffic data, the longest packets collected have length of 1280, we started using 2 layers then increased to 4 layers to accommodate the large number of features. For the initial input layer, we have 1280 neurons, which equals the length of the longest packet. The second hidden input layer has 845 neurons, and the third hidden input layer has 558 neurons. Our output layer has 20 neurons, represents the number of target values we want to get. We improve the accuracy of predictions by using back-propagation method, where MSE losses (errors) are sent back to fine-tune the weights of the net based on loss from previous iteration to lower error rates and increase its generalization. Since our problem is regression, we have options of using Sigmoid or ReLU as the activation functions, however, Krizhevsky et al. discuss that networks with ReLU tend to show better convergence performance than Sigmoid [7]. We chose to go with Leaky ReLU, a variation of ReLU to speed up the training and avoid having dying ReLU problem in case of having too many input values below zero.

For the training process, the optimizer we use is Adaptive Moment Estimation (Adam) for its proven ability to handle sparse gradients and faster-paced convergence towards the minima [8]. We set learning rate to 0.001 and train the model with 100 epochs. We use Mean Squared Error (MSE) as loss function to evaluate algorithm's performance when modeling the dataset, the result is the mean overseen data of the squared differences between true and predicted values. Ideally, the lower the MSE score, the better the model is with predictions.

In the next section, we provide more information about the model performance and how we evaluate our predictions.

IV. Preliminary Results and Evaluation

We train the model with variations of game replays data using mainly the 2 popular maps: Mirage and Dust II. In Figure 4, on the left hand-side is real game players' positions during one full game on map Mirage and the right hand-side graph is players' positions of full game on map Dust II. Since each map from CSGO has different dimensions and variations in pathways that players can go through, we have some similar plots for matches that use the same kind of map. Another factor is density of the data, which depends on the length of the game. Although each game has 30 2-minute rounds, it would depend on players and sometimes a round may take less than the allocated time. A game also runs shorter if one side wins 16 rounds in total.

After training the model, we want to evaluate the predictions. We feed `Test_Network_byte_set` to the model to retrieve corresponding predictions for each of the map aforementioned and plot the predictions in the Figure 5.

In Figure 6, we present a snapshot of three random selections of network bytes from the test set at time 22:28:56-58 for Mirage map and at time 12:40:03-05 for Dust II map.

The results from this initial evaluation step is very promising. From Figure 6, we can see that the predicted positions are very close the real locations. This means that the model is

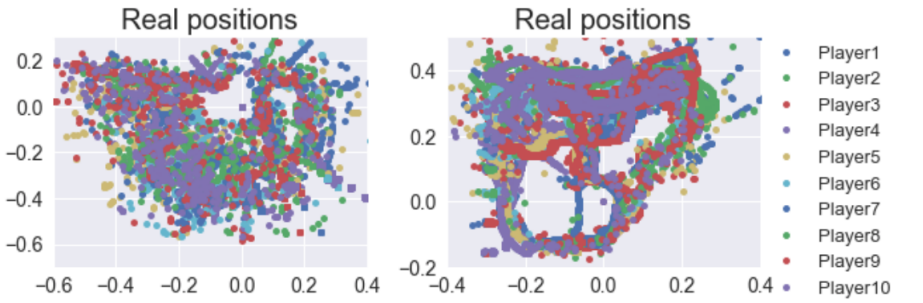

Fig. 4: Real players' movements

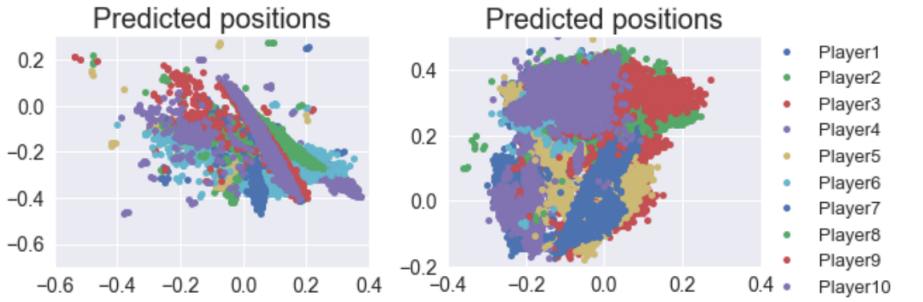

Fig. 5: Model's prediction of players' movements

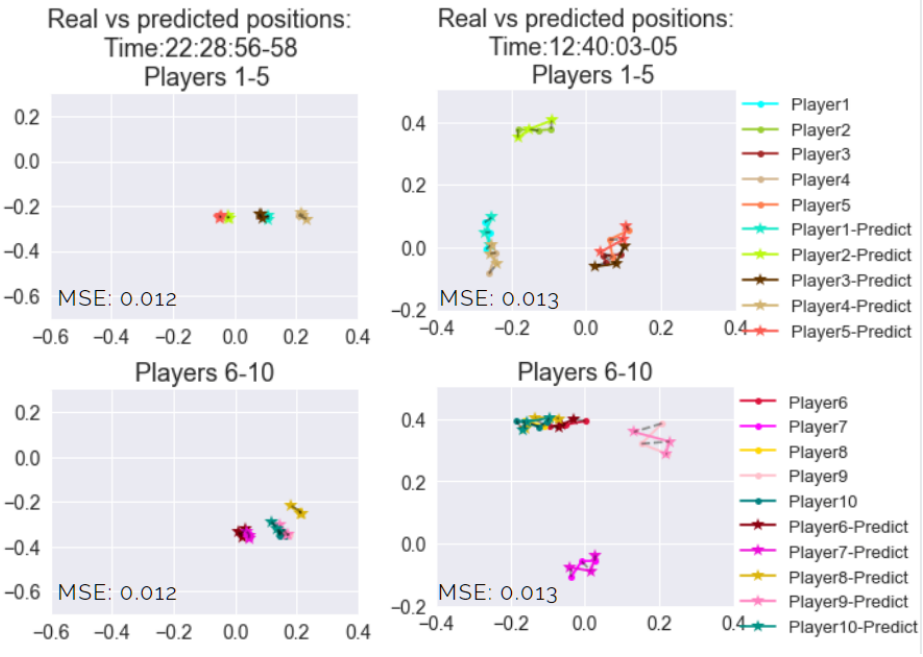

Fig. 6: Real vs predicted locations using random selection samples of 3 seconds from network data associated with same game

able to predict a general area where enemies are during gameplay. As discussed in I, knowing the general locations of the enemies would significantly advance the chance for player to initiate the approach in that situation, where they can prepare themselves for better aiming or secure their hideouts efficiently and avoid triggering their weapons unless needed. With this idea, in section V, we discuss the possibility of building a side-channel tool that reads network traffic in real time and provide a heat map that shows the general locations of enemies.

After using the 20% of the network data from the same game to evaluate the model, we receive remarkably low MSE scores, as seen in Figure 6. However, we want to test the generalization of the model generalization ability of the neural network when it is presented an unrelated set. We use a few different set of network byte to test the model and the predictions are not as good as we expected. In Figure 7, you can see that the predictions are much further away from the real positions compared to results in Figure 6. This means that

we might have overlooked a a few important features for target values and this causes the model to cast predictions further off from target values as seen in Figure 7.

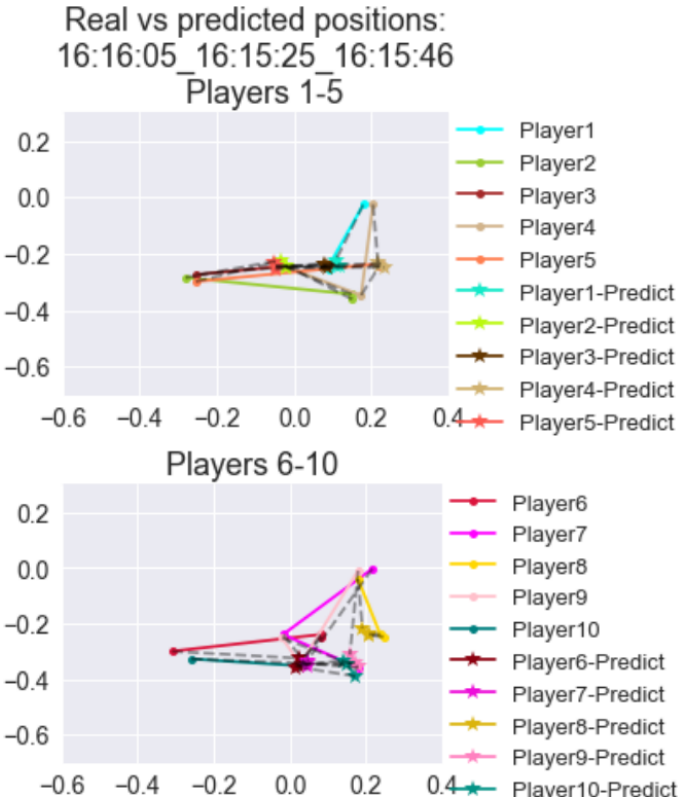

Fig. 7: Real vs predicted locations using random selection samples from network data associated with unrelated game

During training and evaluations, we have tried changing learning rates and number of hidden layers. Our highest Mean Square Error, the number represents the squared normalized distance between the predictions and the actual positions is 0.037 and the lowest score to date is 0.011, the average MSE is about 0.024. We acknowledge that when using separate games as testing sets, our MSE square tends to arrive at the range of 0.023-0.037 meanwhile when using a a subsection of dataset from the same game, we have much lower MSE ranges: 0.011-0.017. Initially, when conducting the training preparation part, we did not take into account the effect of different maps. Which results in the higher errors in predictions when the separate testing set has a different map than the map of the game we use to train the model. This is one of our main targets for future work and improvements.

V. Conclusion

In this project, we want to investigate if there is correlation between network traffic and some hidden information such as the players' positions in competitive video games. For the scope of the project, we have a few criteria when choosing the target game to investigate: 1) Hidden information would provide significant advantage; 2) The game has game demos/replays features; and 3) There is a parser library to extract the following data from demos: positions, timestamps, players' ID. Given the aforementioned requirements, we use CSGO for the project. Our methodology is straightforward and consists of 4 main parts: 1) Collecting game demos and extract players' locations: x and y coordinates and corresponding timestamps; 2) Collecting network traffic packets while playing the game; 3) Align the network packets with the players' positions from demo using timestamps; and 4) Use statistical and ANN to Analyze the correlation between the monitored network traffic vs the information extracted from parsing the replays.

Currently, we collected over 160,000 samples to train the model and the average MSE we archive so far is 0.024 and can be improved as we collect more samples for training and factor in other features such as type of map used and time where players are in launch. While investigating the impact of different maps, we noticed there are patterns in differences of predictions. We also aim to investigate effects of wait time data on model predictions. During 30 rounds of CSGO match, each rounds are less than 2 minutes. However, there is a wait time between each rounds which is 7-15 seconds where players stay static. This may cause some bias during training. Our final goal for the project is to build a tool that retrieves bytes data and displays predictions in real time during game play.

Acknowledgment

This work was supported in part by NSF grant CNS1758017 and part of Research Experience for Undergraduates in Big Data Security and Privacy program at California State Polytechnic, Pomona.

References

[1] Mordor Intelligence. Gaming market: Growth, trends, covid-19 impact, and forecasts (2021 - 2026).
[2] Aaron Parker. Demo analyzer. https://github.com/AronParker/DemoAnalyzer, 2021.
[3] Aditya Jonnalagadda, Iuri Frosio, Seth Schneider, Morgan McGuire, and Joohwan Kim. Robust vision-based cheat detection in competitive gaming. *CoRR*, abs/2103.10031, 2021.
[4] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19, 2019.
[5] A. Dainotti, Antonio Pescapè, and Giorgio Ventre. A packet-level traffic model of starcraft. pages 33 – 42, 08 2005.
[6] 162.254.193.0/24 as32590 valve corporation united states valve.net registry: Arin 256 ip addresses id: Valve-v4-6.
[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.
[8] Sebastian Bock, Josef Goppold, and Martin Georg Weiß. An improvement of the convergence proof of the adam-optimizer. *CoRR*, abs/1804.10587, 2018.