11/21: report

A distance-based win probability model for Counter-Strike: Global Offensive

Guido Petri, advised by Prof. Claudio Silva and Peter Xenopoulos

Github link

Abstract

I created a graph distance based win probability model for the videogame Counter-Strike: Global Offensive. The model is based on both fully connected and convolutional neural network components and outperforms a baseline logistic regression, achieving a round classification AUC of 0.8268.

Introduction

In recent years, a new form of sports has emerged that is based on videogames instead of "real-life" games. This category, called *esports*, has been growing exponentially, especially due to the effects of the COVID-19 pandemic - more people are spending time at home, and during the beginning of the pandemic, most sports were canceled and only esports were available.

In this work, I approached the game of *Counter Strike: Global Offensive* (CS:GO) and attempted to create a win probability model based on inter-player distances.

CS:GO is a videogame based on two 5-player teams, one representing *terrorists* (T), who try to plant a bomb and set it off, and the other representing the *counter-terrorists* (CT), who try to prevent the Ts from planting the bomb, or try to defuse it before it explodes. Both sides can win by eliminating all players from the other side; this means that winning conditions for the Ts are 1. planting the bomb and exploding it, or 2. eliminating the CTs, while the winning conditions for CTs are 1. preventing the Ts from planting the bomb in time, 2. defusing the (planted) bomb before it explodes, or 3. eliminating all Ts.

Win probability models are common in sports, but have generally not yet been extended to esports. They are useful in several ways: using a win probability model, we can detect events that have an outsized influence on the result, we can estimate player value by how much they contribute to their team's win, and we can place odds on a team winning and compare to a betting market, to name a few examples. Peter Xenopoulos has worked a lot on both a library to parse CS:GO data, as well as a basic win probability model that takes damage events as input, but to the best of my knowledge, nobody has attempted a distance-based model for CS:GO yet.

Peter Xenopoulos also introduced a graph-based distance metric to the game. This distance metric is based on the AI's navigation meshes. In short, the AI in CS:GO uses specific files for each map to navigate the map, and these "meshes" are essentially a directed graph with each node representing a small area of the map that a player can stand on, and each edge representing a possible transition from one area to another. Since the game is based in 3D maps, asymmetric distances are possible. For instance, a player might go from a location A to a second location B by falling down a ledge; but the way from B to A may involve going up a ramp, which means $distance(A, B) \neq distance(B, A)$. This poses an interesting problem for distance-based metrics in the game.

Literature review

Not a lot of research has been done in CS:GO. While there has been some exploration of the game, none of it has focused in win probability models in particular, other than Peter Xenopoulos' work.

In the broader esports category, there has been some work done with win probability models[3][6][7][9][10], and even with distance-based models[4][5], but nobody has created a neural network that ingests distances and outputs win probabilities. This suggests that this field is still ripe for research and a lot of "quick wins" can be made by building simple models and examining them.

Finally, in the broader sports category, some work has been done with neural networks[1][2][12], either. Most of sports analytics is driven by organized league needs, and for the most part, the models need to be simple and explainable. This means that a lot of the sports analytics models that have been done are based on generalized linear models, or simple models that are well-studied and well-understood, like logistic regressions. In fact, not many organizations use more complex machine learning models like SVMs or random forests, let alone neural networks.

This being said, there is a growing interest in neural networks in this community, if only to make data annotation easier. A lot of research is being put into computer vision techniques in order to glean insight from video (e.g. via pose estimation[13]) or create data for other models (e.g. to create player and ball coordinate data).

Methodology

The intent of this project was to see how interplayer distances could affect win probability models, and whether they can inform a naïve model so that it has a better performance. The problem was set up as a binary classification problem: in particular, the problem was to correctly classify whether a round was won by the T side 0 or the CT side 1.

The main performance metric used was the area under the receiver operating characteristic curve, *AUC*. However, I also examined accuracy and the log loss of each model created. The intent here was to achieve maximal AUC while having a well-calibrated model, as measured by the log loss.

The dataset used for this was extracted from CS:GO replay files downloaded from the HLTV website and parsed using Peter Xenopoulos' `csgo` package. The data had to be cleaned and standardized, since the parser is not perfect and several issues exist in the CS:GO replay files. For example, several rounds had to be dropped due to them having a different number of players than the standard 5 per side.

Initially, I was using data from all maps. However, this proved unwieldy, since there were so many games. In order to simplify the modeling process and cut down the number of available datapoints, I elected to only examine a single map, `de_dust2`. This yielded approximately 2500 rounds.

Each row represented one player's game state at a given game frame. For training and testing, a split of 60-20-20 was used between training, validation, and testing sets, respectively. The dataset was split on a per-game basis, since there can be some correlation between rounds of the same game: a win in an earlier round can carry psychological "momentum" into the next round, and carries virtual money (used to buy in-game equipment that can improve damage output and tactics) into future rounds, as well.

The data was transformed by measuring pairwise interplayer distances in both directions based on their `xyz` coordinates. Additionally, I used a flag for whether a player was alive or not on a per-frame basis.

I used two different ways of organizing the input data. The first method was simply a 10×10 matrix with interplayer distances, along with two 1×10 columns appended to them to indicate whether a player was alive or not and which side each player belonged to, respectively. This yielded a tensor with shape $(12, 10)$ for each game frame.

The second method organized the data into 4 5x5 distance channels: one for the inter-player distances for all T players, one for all CT players, one for the TxCT interplayer distances and a final one for the CTxT distances. Two additional 5x5 channels for the players being alive or not were added, for a final tensor with shape $(6, 5, 5)$ for each game frame. The objective was to induce structure into the data and make it easier for the networks to learn from it, instead of expecting the networks to learn the relation between the player teams vector and the remainder of the data.

As a comparison base, three models with varying degrees of complexity were set up: a map-based model, a player count based model, and finally, a logistic regression that incorporated both the player counts and the interplayer distances as features.

Map-based baseline

The first model is the simplest: it just assigns each round to be won by the team that wins the most rounds in `de_dust2`. In the dataset used, approximately 52% of games were won by T. This model does not use any distance-based data, nor any player-based data. Its metrics are:

Accuracy: 0.5256
AUC score: 0.5
Log loss: 16.3849

The accuracy is precisely the number of rounds that were won by T. The AUC score is no better than random guessing because the model predicts the same result no matter what game situation it is fed. Finally, it's a particularly uncalibrated model, since it always predicts with 100% certainty that T will win. This is evident by the very high log loss.

Player count baseline

The player count baseline is a simple, yet very effective model. The idea behind it is that if there are 5 players alive for one side and only 1 alive on the other, it's very improbable that the side with only 1 player remaining will win.

This model was developed by grouping the number of players alive for each side and calculating what the average winning percentage for CTs was, for each combination of players. The results for this model are:

Accuracy: 0.6922
AUC score: 0.7838
Log loss: 0.5412

Clearly, this model would be more accurate than one that doesn't take game situation into account, like the map-based baseline. It is also evident from the above metrics that strictly knowing how many players on each side are alive can yield a very good predictor for which side will win the round. It's also surprisingly well-calibrated for a model that only uses player counts as its input.

Logistic Regression baseline

The last baseline is based on machine learning standard practice. It is usual to take a very simple machine learning model that can take the same input as the "target" model and see how it performs. As such, the data input for this logistic regression was the same as for the neural networks developed later. The only additional transformation done was scaling the features to have mean 0 and standard deviation 1. This was done due to the way the logistic regression is set up: it penalizes weights that are of large magnitude, and thus if two features have different scales, they will be penalized differently. Additionally, constant features were removed, since they only deter the logistic regression from performing efficiently.

For this baseline, the data input method mattered. The results for the first method are:

Accuracy: 0.5512
AUC score: 0.5019
Log loss:

This model clearly did not learn relationships in the data very well. It has scores that are only slightly better than the map-based baseline, and much worse than the player count baseline, despite having access to more information. The results for the second method are:

Accuracy: 0.7086
AUC score: 0.7897
Log loss: 0.5183

This model is even better than the player count based model, which is good - it has more information than the player count model, and so should have a higher performance. However, it's only slightly better, which indicates that the additional data couldn't be used by the logistic regression very efficiently.

Neural networks

Four different general architectures were used. The first is composed of strictly linear, fully-connected layers; the second, stricly convolutional layers; the third was composed of residual blocks; and the fourth was a combination of linear and fully connected layers.

In order to prevent overfitting, early stopping was used with the validation set. I used the ADAM optimizer with momentum parameters $(0.9, 0.999)$ and binary cross-entropy loss to optimize the model parameters.

The objective was to explore different architectures and different parameters to see what would work best. The search was done across different numbers of layers with different numbers of hidden units, different activation functions, and different batch sizes and learning rates. I also explored the options of including batch norm and dropout. This architecture and hyperparameter exploration was done across both versions of the dataset.

Results

The full tabular results are accessible under this link. The first dataset is the one with $(12, 10)$ shaped tensors for each frame, and the second dataset is the one with $(6, 5, 5)$ shaped tensors.

Initially, I had only trained networks based on the first dataset. This clearly was not working - the models weren't learning to predict the win probability better than random chance, if at all, as evident from the results. I then went back and created the alternate data input format, which I then used for the subsequent networks. This format worked much better than the previous one, but it was still not enough to beat the baseline.

However, it was clear that there were a few things that influenced the network's performance massively: first, the linear sections had big advantages with batch norm and dropout, and the convolutional sections had advantages with batch norm but not dropout; second, a leaky ReLU activation function worked much better than the standard ReLU; and third, the batch size could be increased to be between 50-100 and this improved performance (likely via some influence of batch norm). Even still, after the second iteration of models for the second dataset, this was not enough to beat the baseline models.

It was here that I decided to implement the combination model that used a convolutional section for the interplayer distances, and a fully connected section for the player count data. This was the third iteration of models, and it worked a lot better and achieved better overall scores than the baseline models.

In particular, for the combination model, I used dropout for the fully connected section and batch norm for both the fully connected and the convolutional heads of the network. I used leaky ReLU with the standard negative slope of 0.01 for the activation function of both heads.

Using the best combination of hyperparameters below, I trained the network on the combined training and validation sets and compared its predictions with the ground truth for the testing set. These were the results:

Hyperparameter	**Value**
Batch size	64
Hidden sizes for fully connected layers	[200, 100, 50]
Activation function	LeakyReLU
Convolutional parameters	C6k1-C6k1-C6k5
Epochs	22
Dropout	Fully connected layers only
Batch norm	Convolutional and fully connected layers
Skip connections	None
Learning rate	1e-6

For the convolutional parameters, *Cnky* means a convolution with n channels and kernel size y, followed by the activation function. No pooling was done.

Scoring function	**Value**
Accuracy	0.6951
AUC	0.8268
Log loss	0.5164

As is evident from the table above, this network outperformed all the baselines in AUC and log loss, losing the accuracy metric only to the logistic regression baseline.

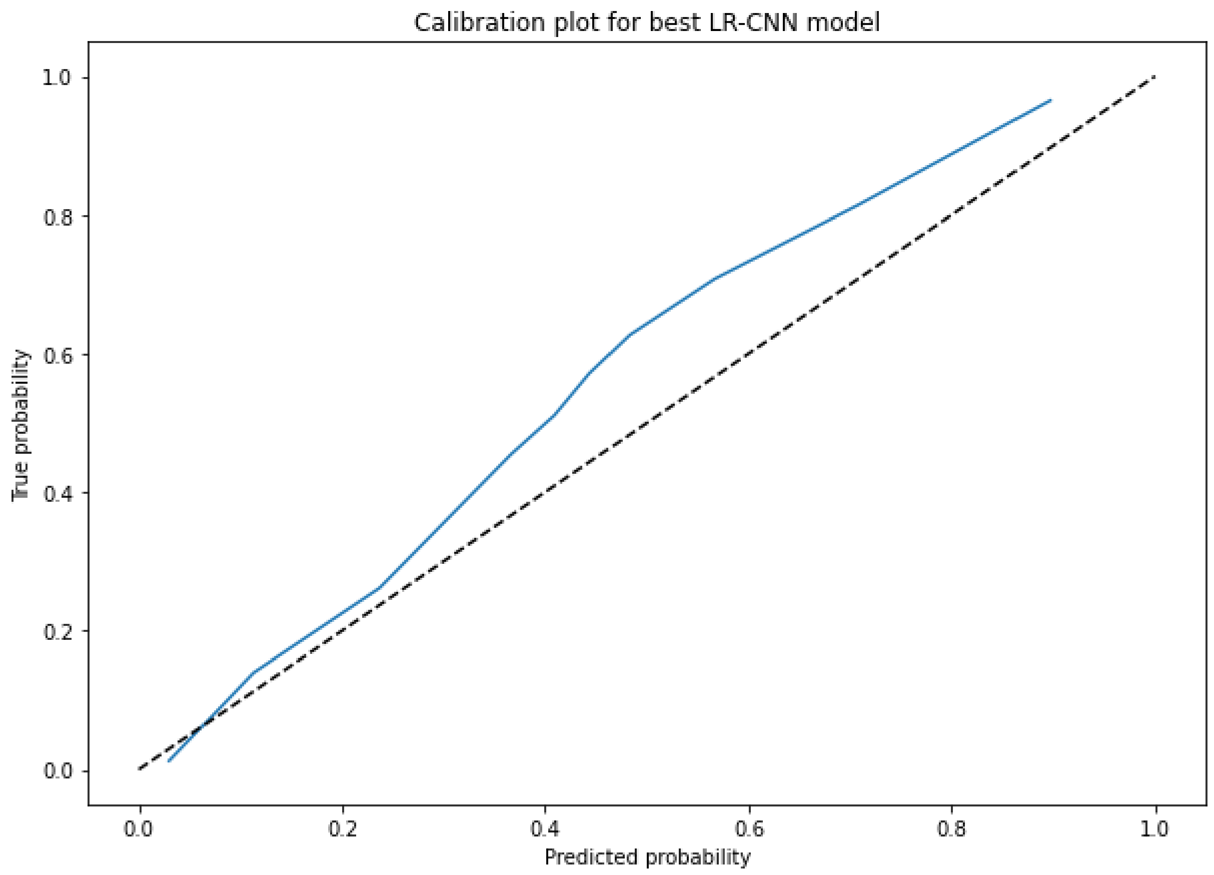
Discussion

The results of the neural network experiments and from the logistic regression baseline seem to indicate that interplayer distances indeed improve a win probability model. In order to prove that this was actually the case, I ran the best neural network found above with only the fully connected, player count based head, and with only the distance-based, convolutional head. The results are below:

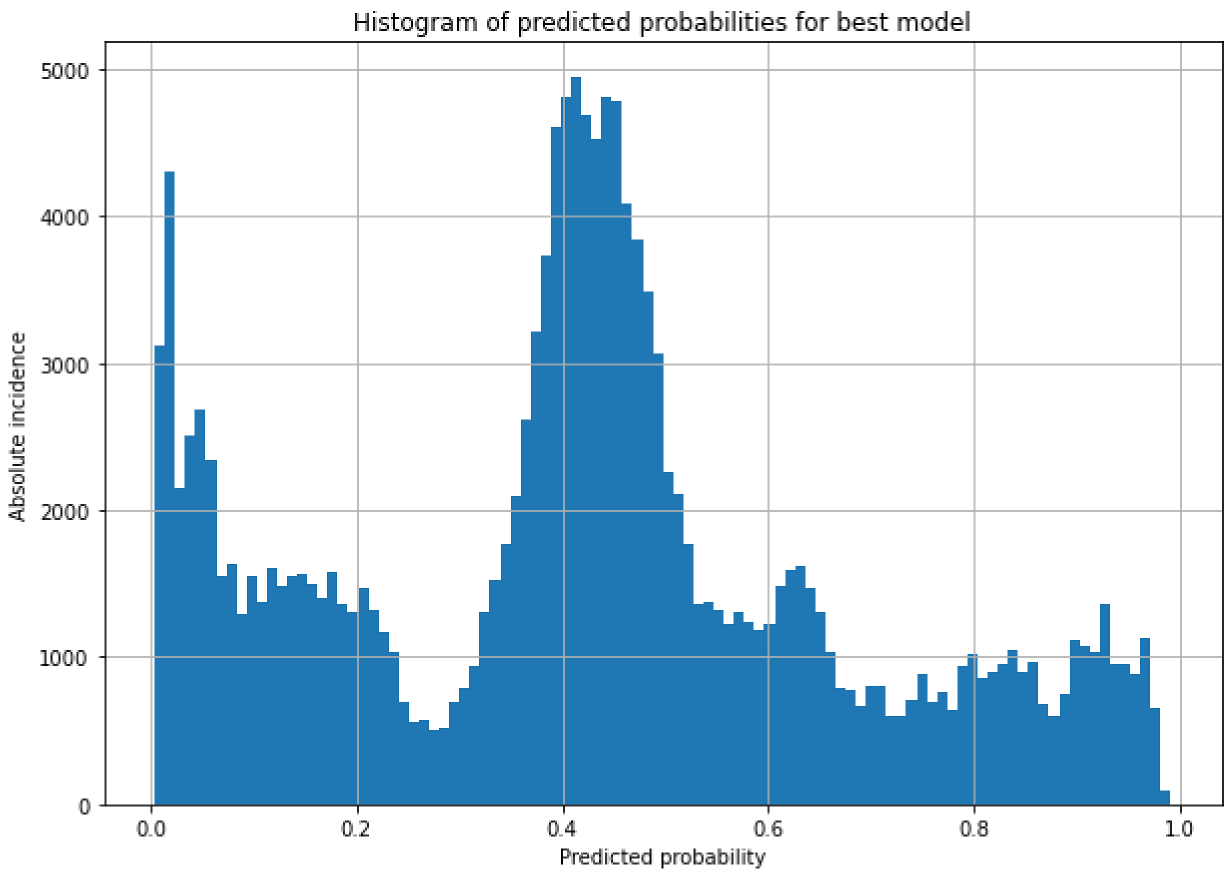
Ablation model	Accuracy	AUC	Log loss
Without distances	0.6852	0.8136	0.5093
Without player HP information	0.5407	0.5662	0.6878

From the table above, it's clear that the player HP information is very valuable for the model - with only that, it achieved a very high AUC, and was better calibrated than even the full model. The model based only on interplayer distances, on the other hand, did not perform very well. It outperformed the map baseline, meaning it was learning some representation of the game situation, but it was not enough to build a good probability model. However, the combination of both sources of information is a better model than each individual part, which is expected - if the model has more information, it should not perform worse than without that information.

Examining the probabilities output by the model can yield even further insight. First, a calibration plot:

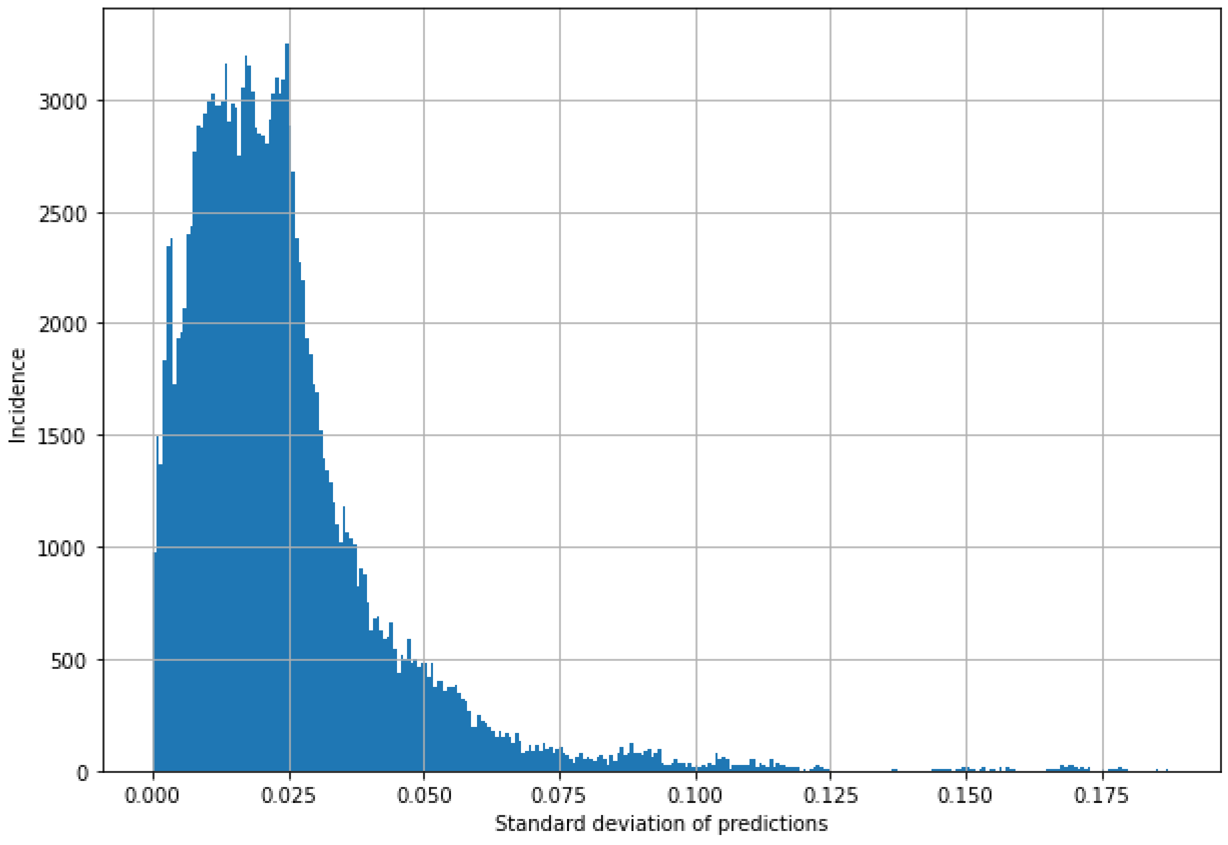


The model is very well calibrated, but it does underestimate win probabilities for situations where the game may be ambiguous, or even a clear win for CT. Looking closer at the actual network predictions as a histogram:

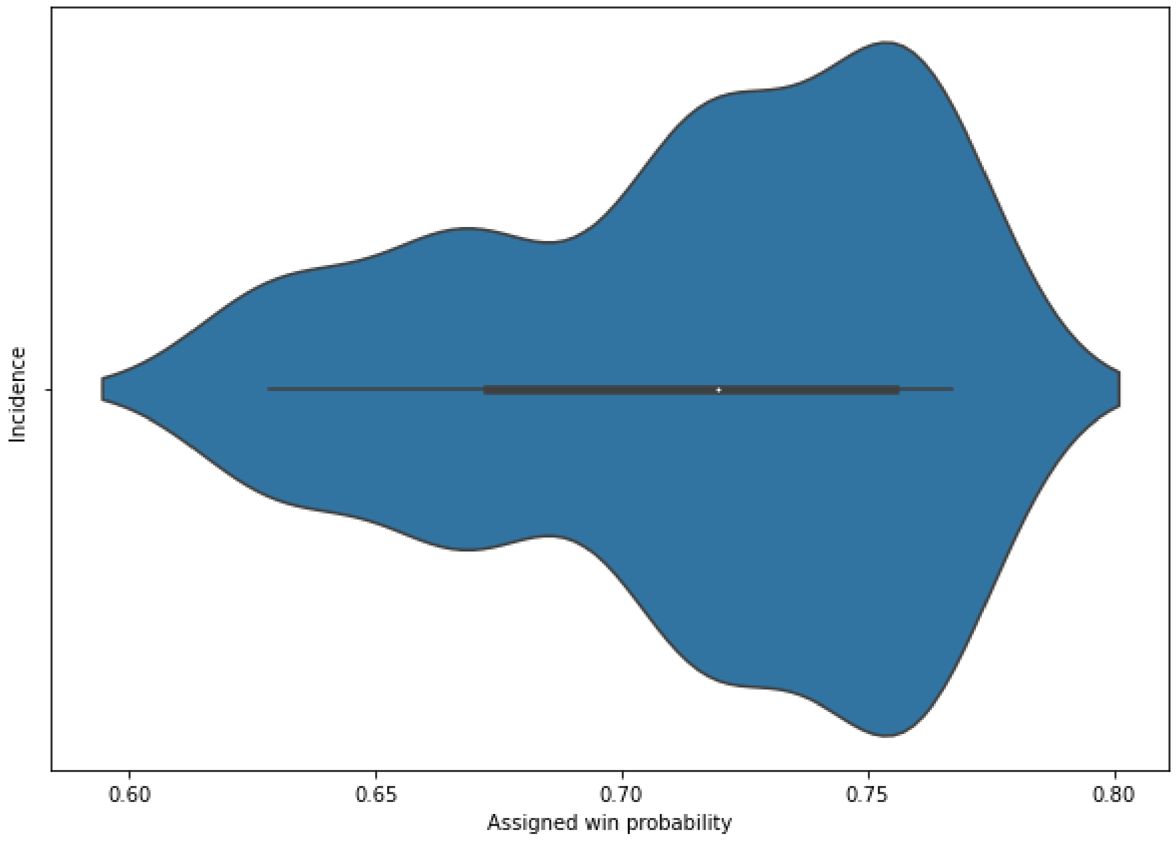


From the above graph, it seems like the model finds several situations in which it's very clear that T will win, and many situations where it's ambiguous, but not very many where CTs will win. My guess is that this is due to lacking bomb information in the dataset. Recall the winning conditions: Ts can win even when all players are dead, if the bomb has been planted and the CTs are unable to defuse it in time. This probably confused the model, since all T players were dead!

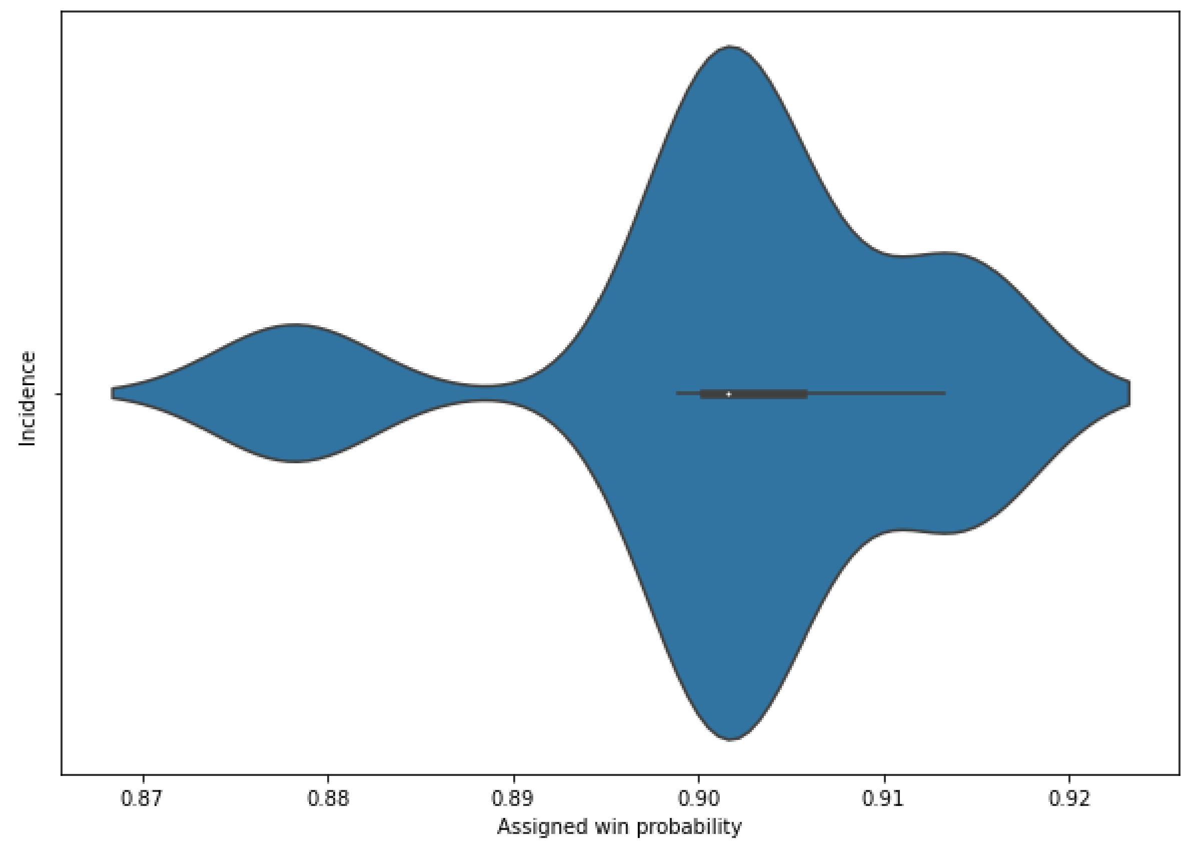
Finally, as a litmus test, we can examine the model's variance in predictions when individual players are permuted in the data. Since CS:GO does not have an inherent "player order", and any player can be first on a list of a team's players, the model should output similar predictions to data that has been modified such that individual players have been permuted. For all 120 possible permutations of T players, I fed the permuted data into the model to get the win probability for that permutation. I then calculated the standard deviations of all test inputs, across all 120 permutations for each test input. Plotting the histogram of these standard deviations:



As an example, here is the assigned win probability as a violin plot for a single test input with standard deviation across all permutations of 0.044:



The variability of the win probability estimators is indeed quite high: the win probability ranged from 60% to 80% with the exact same theoretical input, just permuted differently. Most of the test inputs did not have such a high standard deviation, however. For instance, a test input with standard deviation 0.01:



Here, the win probability varied only by approximately 5%. It's still somewhat disappointing that the neural network is not completely immune to changes in player order.

Summarizing, the neural networks outperformed the baselines set, and achieved relatively high accuracy while maintaining calibration. The interplayer distances played a role in making the network better, albeit not as much as one might have expected.

Future work

For future work, I recommend incorporating the bomb data as well as round time data into the input fed to networks. This would have an effect of improving the network's predictions around situations where several CTs are alive, but few Ts. These edge cases probably confused the networks I trained because the additional data was not available to them.

I would also suggest improving the network's resilience to permutations of the input. Perhaps one way of doing this would be via data augmentation: by randomly permuting the input data and feeding it to the network, it might learn that the player order does not matter to the win probabilities. Another alternative is to use bigger convolutional kernels or more convolutions, in the hope that a kernel of size 5 will incorporate the data better.

References

1. Loeffelholz, Bernard et al: Predicting NBA Games Using Neural Networks
2. Zamir, Evan: A Simple Neural Network for In-Game Win Probability Modeling of NCAA Basketball Games

3. Yang, Yifan et al: Real-time eSports Match Result Prediction
4. Drachen, Anders et al: Skill-based Differences in Spatio-Temporal Team Behaviour in Defence of The Ancients 2 (DotA 2)
5. Rioult, François et al: Mining Tracks of Competitive Video Games
6. Hodge, Victoria et al: Win Prediction in Esports: Mixed-Rank Match Prediction in Multi-player Online Battle Arena Games
7. Song, Kuangyan et al: Predicting the winning side of DotA2
8. Pobiedina, Nataliia et al: Ranking Factors of Team Success
9. Johansson, Filip et al: Result Prediction by Mining Replays in Dota 2
10. Makarov, Ilya et al: Predicting Winning Team and Probabilistic Ratings in “Dota 2” and “Counter-Strike: Global Offensive” Video Games
11. Malafosse, Charles: Machine Learning for Sports Betting: It’s Not a Basic Classification Problem.
12. Giuliodori, Paolo: An Artificial Neural Network-based Prediction Model for Underdog Teams in NBA Matches
13. Fani, Mehrnaz et al: Hockey Action Recognition via Integrated Stacked Hourglass Network