# Fatality Bot for Resistance Game

AJAYKIRAN PALAMTHODI, ap21903@essex.ac.uk

MSc, Advanced Computer Science

University of Essex, UK

ap21903@essex.ac.uk

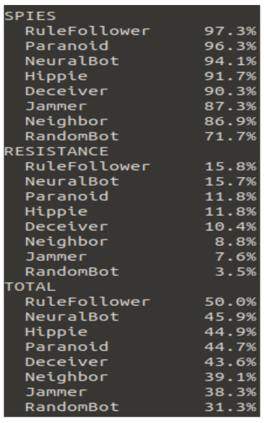
When the Neural Bot plays with beginners the results were good, but when it played against intermediates the results weren't very good. I have tried to improve the Neural Bot and make it beat all the intermediate bots. I have run the competition 10000 times and gotten promising results. In order to improve the Neural Bot, I have added in some feature inputs. (failed_mission_leadered, missions_passed_as_spy, missions_success, won_as_ress, won_as_spy)

The accuracy of the LoggerBot classifier in finding if the Logicalton Bot was a spy was 77% and I have risen it to 98.7%. I have also reduced the loss from 0.43 to around 0.029. The neural bot also works well in finding if other bots like, Bounder, Simpleton or Trickerton are spies.

## Introduction:

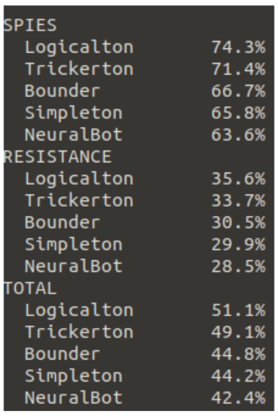
We have used neural network to play the resistance game. We have used a LoggerBot to log some data on how the spies would behave versus how a resistance player behaves. This is used as training data for supervised learning. We have trained a neural network classifier to identify resistance members from spies. We have created a NeuralBot that uses the classifier to decide who to trust in the game as a spy or a non-spy.

When the neural bot played against beginners the result was good. However, when it was put against intermediates, the result wasn't very good. The motivation is to improve the NeuralBot and make it beat the intermediates.



```
SPIES
   RuleFollower      97.3%
   Paranoid          96.3%
   NeuralBot         94.1%
   Hippie            91.7%
   Deceiver          90.3%
   Jammer            87.3%
   Neighbor          86.9%
   RandomBot         71.7%
RESISTANCE
   RuleFollower      15.8%
   NeuralBot         15.7%
   Paranoid          11.8%
   Hippie            11.8%
   Deceiver          10.4%
   Neighbor           8.8%
   Jammer             7.6%
   RandomBot          3.5%
TOTAL
   RuleFollower      50.0%
   NeuralBot         45.9%
   Hippie            44.9%
   Paranoid          44.7%
   Deceiver          43.6%
   Neighbor          39.1%
   Jammer            38.3%
   RandomBot         31.3%
```

*fig 1: NeuralBot against Beginners*



```
SPIES
  Logicalton       74.3%
  Trickerton       71.4%
  Bounder          66.7%
  Simpleton        65.8%
  NeuralBot        63.6%
RESISTANCE
  Logicalton       35.6%
  Trickerton       33.7%
  Bounder          30.5%
  Simpleton        29.9%
  NeuralBot        28.5%
TOTAL
  Logicalton       51.1%
  Trickerton       49.1%
  Bounder          44.8%
  Simpleton        44.2%
  NeuralBot        42.4%
```

*fig 2: NeuralBot against Intermediates*

## The Resistance:

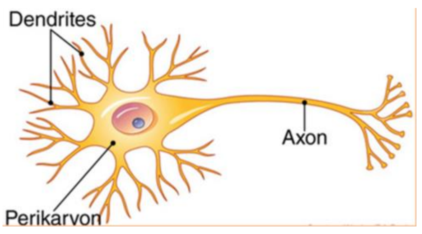
Resistance is a board and cards game. The 1st team to get 3 points win. We play as 2 Teams, Red-Spies, Blue-Resistance. Each player is given a character card, which determines if you are a spy or a resistance. Each player is also given an approve/ reject card. One player is given leader token. The leader tells everyone to close eyes. Then tells the spies to open eyes and look at each other, then close. Then tells everyone to open eyes and the game begins. The Mission leader should choose team to go on mission and everyone else get to vote on if the team should go on mission o should they select a different team. Leader gives each player he wants to go mission a gun token. If majority rejects the team, the leader token is passed on to next player clockwise. The number on board tells how many members are needed for a mission. If you can't agree on a team 5 times in a row spies win. If majority approves each member of team is given a success and a fail card. If in the resistance, players must turn in success card face down. The spies can turn in either success or fail card. Two separate players must shuffle the cards and then the leader reveals them. If there is at-least 1 fail the mission is sabotaged and a spy token is placed on the Mission circle. The goal of spies is to sabotage mission. The resistance must try to find the spies and not let them in on missions. If mission is a success a resistance token is placed on the Mission circle. When one of teams score 3 points the game ends and that team wins.

## Background:

We try to use neural network to play the resistance game. W have used a LoggerBot to collect date from playing with other bots. Then we pass these data to a neural network classifier and use the NeuralBot to identify who the spies are in the game.

In supervised learning we try to approximate a function using some labelled date and their attribute features. To evaluate the algorithm's effectiveness, it is tested on previously unseen examples. Supervised learning consists of a list of input vectors {x1, x2, x3, …}  and a list of output vectors {y1, y2, …} each pair of (xi, yi) is called a training pattern. An artificial neural network is based on the idea that the basic working of a human brain can be imitated using silicon and wires by making correct connections.

<u>The Neuron/Node</u>



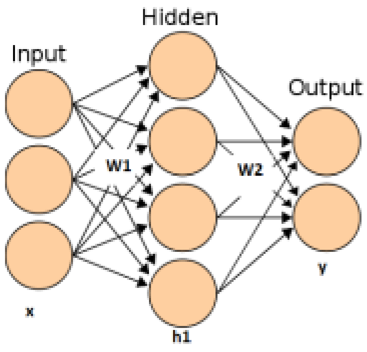


*Fig 3: Neuron*                                    *Fig 4: Neural Network with 1 ip layer, 1 op layer and weights*

A node is the basic building block of ANN. It has an Axons and Dentrites. The dentrites are the receiver of the signal and axon is the transmitter, just like the input and output signals in ANN. A network always contains input and output layers. There may also be hidden layers between them. Nodes in one layer are connected to nodes in the next layer with weights. Node takes the weighted sum of all the input signals it is getting and then it applies an activation function. Some of the common activation functions used are sigmoid, Threshold, Rectifier, tanh etc. I have used the tanh function to train the loggerbot classifier.

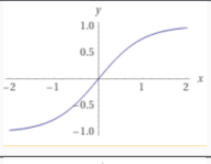
| Activation Function | Equation | TensorFlow Function | Graph | Range |
|---|---|---|---|---|
| Hyperbolic Tangent Function | $y = tanh(x)$ | tf.tanh |  | $-1 < y < 1$ |

*Fig 5: Hyperbolic Tangent Function (tanh)*

A bias constant is added to the weighted sum before sending it to the activation function so: (x1*w1+ x2*w2…xi*wi + bias) is sent to an activation function. Bias is a constant value added to the weighted sum of input signals to shift the result of the activation function to the positive of negative side.

How Neural Networks Learn?

We first need to train the NN, that is choose weights and biases that are useful. We want the NN to choose weights and biases so that the actual result and the output of the activation function are equal f(xp) ≈ yp. To obtain this we define a Loss Function that we seek to minimise. L = −log(pt) is the cross-entropy loss function. Thus, minimising L must maximise pt (probability of true category). In keras, we can specify Cross-Entropy Loss in the model.compile line with: ***loss=keras.losses.SparseCategoricalCrossentropy().*** A Softmax function is used in the final layer to make sure the outputs are all positive and adds up to 1 like probabilities should.

$$p_i = \frac{e^{y_i}}{\sum_k e^{y_k}}$$

*Fig 6: Softmax Function*

When the training set size is very large, just computing L over the whole set will take a lot of time. So, we train in mini batches. 1 epoch is one full pass through the data set which might consist of many iterations. 1 iteration is 1 application of weighted update equation, calculated for mini batch. An optimizer is used to adjust the weights and biases to speed up/slow down the result as required. Here we use The Adam optimizer. It dynamically and intelligently adjusts learning rates to handle tight corners and platues.

## Techniques Implemented:

The agent submitted works on NN to guess weather a player is a spy or not in a resistance game. I have built on the Neural bot and tried to improve the winning rates as resistance, spy and overall while playing against intermediate bots. I have used several decision trees along with the output of the NN to improve the bots winning chances.

- The bot takes in 22 input vectors out of which 4 (turn, tries, playername and playerid) are eliminated.
- The other input vectors include failed_missions_been_on, missions_been_on, 6 numbers for num_missions_voted_up_with_total_suspect_count, and another 6 numbers for num_missions_voted_down_with_total_suspect_count, won_as_res, won_as_spy, mission_success, mission_passed_as_spy.
- We use a loggerbot to log thes values by pitting it against the intermediate bots.
- We have 2 outputs, either a spy or not a spy. we choose 70% of data for training and 30% or validation.
- We define a sequential model with 3 layers. We use **10 nodes** and **tanh** function in the 1st 2 layers. We use **softmax** for the final/ output layer
- We use **keras.optimizers.Adam** with a learning rate of **0.001**
- Since this is a classification problem we use **keras.losses.SparseCategoricalCrossentropy** to find the accuracy of the model and use it on a validation set.
- **batch size = 50** and **epochs =120**
- We save the model in **bots/loggerbot_classifier**
- We get models against each of the intermediate bots stack them up together and send it to output layer with softmax function and we get probabilities of each bots being a spy or not.

- I have implemented several decision trees and nested decision trees along with the NN to increase the winning rate of the bot.
- With this I have made the bot always win against Simpleton, Trickerton and Bounder of the intermediate bots.
- It still loses to Logicalton of the intermediates, but the winning rate are very close and sometimes it even beats Logicalton.

## Experimental Study:

Playing the Neuralbot against the intermediates, the accuracy at guessing if each bot was a spy or not initially was: Logicalton-78 %, Bounder-77 %, Trickerton-70 %, Simpleton-74 %

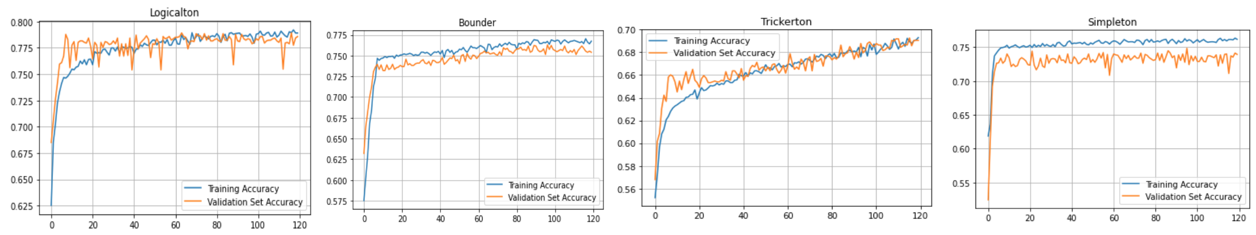


*Fig 7: Training vs Validation Accuracy at the begining*

The input features that were used earlier was (failed_missions_been_on, missions_been_on, then 6 numbers for num_missions_voted_up_with_total_suspect_count, and another 6 numbers for num_missions_voted_down_with_total_suspect_count).
**failed_missions_leadered** – This shows the number of missions that were sabotaged while the player was leader. A spy is more prone to select a team that would sabotage the mission. He would select the other spy counting on him to sabotage the mission or sabotage the mission himself. So, I thought this would help improve the accuracy of the classifier. I define this in the **onGameRevealed(…)** method of the loggerbot and increment it for each player when he is the leader of a team that fails a mission. After adding this variable and running competition for 10000 times with intermediats and the loggerbot the accuracy score for each bot was- Logicalton-80 %, Bounder- 78%, Trickerton-70%, Simpleton-74%. This was not that promising, so I did not retain this input variable.

**mission_passed** – This tells us how many missions the player has been on that were successful. I have added this variable in the **onGameRevealed (…)** method of the loggerbot and increment it for each passed missions when number of sabotages were not greater than 0 in the **onMissionCompleated(…)** method. After adding this variable and running competition for 10000 times with intermediats and the loggerbot the accuracy score for each bot was- Logicalton-88 %, Bounder- 88%, Trickerton-79%, Simpleton-84%. This was good result, but I thought I could make it better. I retained this input vector.

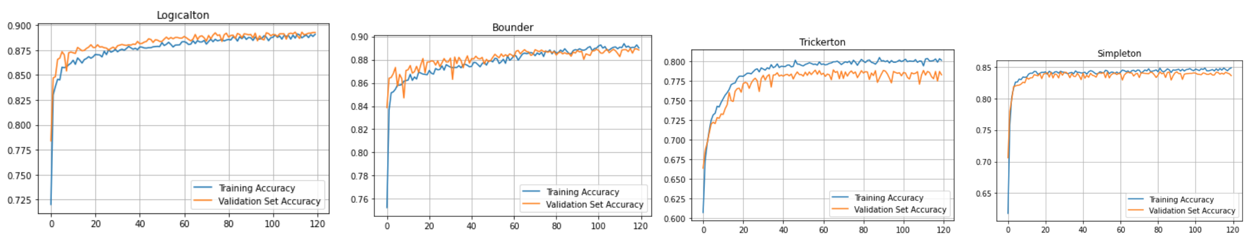


*Fig 8: Training vs Validation Accuracy after adding* ***mission_passed***

**missions_passed_as_spy** – This variable tells us about if a spy passed the mission being in the team. Spies usually don't sabotage the 1st mission as not be suspected so; this tells us about how cautious the bot is. Sometime this could backfire as the bot would just be sabotaging randomly and the variable would tell us nothing about the bot. But I thought we should give it the benefit of the doubt. I have added this variable in **onGameRevealed(…).** This variable did not have much impact on the accuracy. The accuracy scores after adding this variable was: Logicalton - 89 %, Bounder – 88 %, Trickerton – 87%, Simpleton – 84%. This variable showed improvement in accuracy only on the Trickerton bot, but I decided to keep it as it showed potential of working good on future bots that may play as spy well.

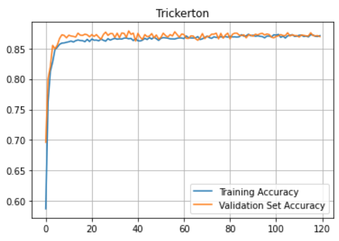


*Fig 9: Training vs Validation Accuracy of Trickerton after adding **missions_passed_as_spy***

**won_as_spy, won_as_res** – These are 2 input signals that I have added. If the player won as a spy. Won_as_spy=1 else 0. If the player wins the game as resistance won_as_res=1 else 0. I have added these in the **onGameRevealed(…)** method and update it in **onGameComplete(…).** In this method If win==True that means resistance won, else spies won. But to include these input vectors I had to move the line where we append all the input features to a list to **onGameComplete(…)** because these values are going to be always 0 if we append the input features into the list **onVoteComplete(…).** The accuracy score after adding these 2 input signals for each of the bots were: Logicalton – 98%, Bounder - 99%, Tricerton - 98%, simpleton – 98%. These were really good accuracy scores, and it makes sense because these values tell us about how much the bot is prone to win when they play as resistance or spy. It tells us if the different bots have any advantage when they are playing a specific class. I have retained these input vectors

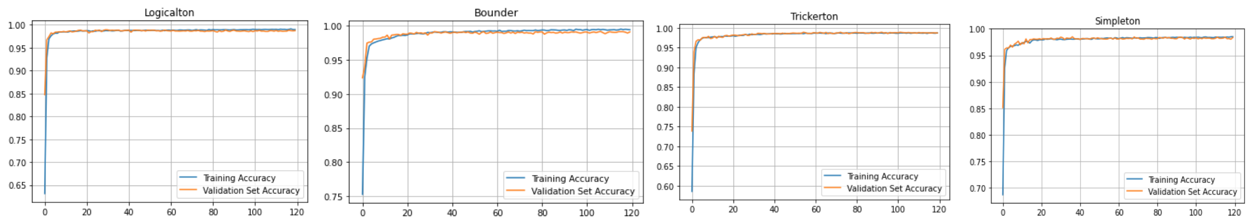
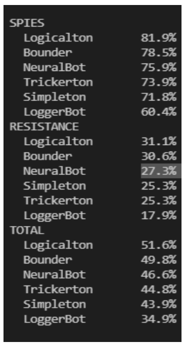


*Fig 10 - Training vs Validation Accuracy Final*

After this I decided to feed this to the neuralbot and see how it performs against the intermediate bots after the accuracy score in detrming if a bot is spy or not has been increased considerably.

I was surely getting better results with an overall win % of 46.6, win as spy = 75.9% and win as resistance = 27.3%. The Neural bot was beating 2 of the intermediate bots Simpleton and Trickerton.



```
SPIES
  Logicalton      81.9%
  Bounder         78.5%
  NeuralBot       75.9%
  Trickerton      73.9%
  Simpleton       71.8%
  LoggerBot       60.4%
RESISTANCE
  Logicalton      31.1%
  Bounder         30.6%
  NeuralBot       27.3%
  Simpleton       25.3%
  Trickerton      25.3%
  LoggerBot       17.9%
TOTAL
  Logicalton      51.6%
  Bounder         49.8%
  NeuralBot       46.6%
  Trickerton      44.8%
  Simpleton       43.9%
  LoggerBot       34.9%
```

*Fig 11 – Running Competition after getting good accuracy score on all intermediate bots*

I was not satisfied with these results as the Neural Bot got trashed by the other 2 intermediate bots. Bounder and Logicalton. So I decided to dig into what made logicalton so good and maybe implement some of those features in my bot. I quickly realized that the NN only helps in guess if the other bots are spy and it is not fullproof. It doesn't help the bot when it plays as a spy in any way as we already know the spies.

**onGameRevealed (…) –** We were just using this method from the logger bot, but we don't get access to **spies** that way. So I decided to over ride it. I also added some fields of my own as this is the 1st method that is called and so a good place to declare fields. **self.spies=spies** , **self.failedTeam = []** ( a list of failed teams), **self.otherSpy** ( to know the other spy if you are a spy), **self.downvotes** = 0 (number of times you have voted down a team), **self.teams = [] , self.obviousSpies = []**.

**vote (…) –** All that neural bot did here was vote up all the teams as spy and vote down teams with 2 highest probable spies as resistance and vote up everything else. There was room for lots of improvement here.

- If we are spy - Always vote down 5th try and try to win the game.
- We increment the **self.downvotes** if we have voted down and decrement it when we have voted up in **onVoteCompleted (…).**
- We don't want to give away that we are spy so if **self.downvotes** >=3 and if we are one of the 2 players who has the highest failed missions ( **failed_missions_been_on** from **loggerbot**) we vote the team up.
- Else we vote up teams with spies > 0
- If we are the team leader - we vote the team up.
- If we are in resistance - we vote the team up if tries are 5.
- If the length of team is 3 and we are not in it - we vote the team down.
- We discard teams or subset of teams that has been on failed missions before.
- Then we use the NN to check the trustworthiness if we haven't returned already.
- If a member of the team is one of the 2 highest in both sorted_players_by_trustworthiness and failed_missions_been_on – vote the team down as resistance.
- If a team member is in the list **self.obviousSpies** - vote the team down as resistance

**onVoteComplete (…)**

- If you have voted down a team increment self.downvotes by 1 else decrement it by 1
- If self.game.tries =5 and someone has voted down a team. Add them to obvious spies list. Only spies vote down last attempt to form team.

**sabotage (…)** – Here the neuralbot always returned True

- Always Sabotage the mission if win=2 as spy
- Always Sabotage the mission if losses=2 as spy
- As spy if the other spy is in team and it is not one of the 2 players having highest failed missions, don't sabotage the mission. Count on the other spy to sabotage it.
- As spy If you are not one of the 2 players having highest failed missions sabotage the mission
- If in resistance don't sabotage the mission.

**onMissionComplete (…)**

- If num_sabotages>0 – mission failed. Add the team to **self.failedTeam** **self.failed_missions_been_on** for the team members get incremented from loggerbot.
- Make self.downvotes=0 as we start on voting teams again
- If num_sabotages<=0 mission passed add the team to **self.teams**

**getOptimalTeam (…)**

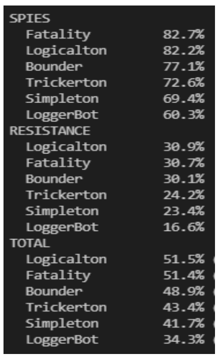
Here we try to make an optimal team with members of previously passed missions. This list can be empty

- **Suspicios_players** – players who has highest failed missions at any point
- We go through all teams in **self.teams.** if we are part of any team remove ourselves from that team.
- Remove suspicious_players from any previously successful teams.
- If there are still anybody left in the team. We add that member to another list **team**.
- If **len(team) >=count-1** we break and return **team**

**select (…)**

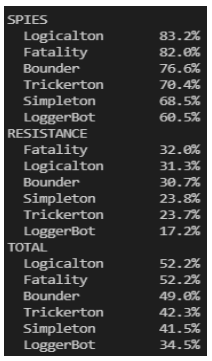
- If We are a spy, we need to select a team with yourself and 2 resistance members at random. We don't need to depend on NN to do this as we already know the spies.
- If there are any team in **self.teams** and is **self.game.team>3** we we set **team=self.getOptimalTeam(count)**
- If there are any **obviousSpies** in the team, we remove them
- If there are any **obviousSpies** in the **sorted_players_by_trustworthines** we remove them and append them at the end of the list
- If we are resistance – I length of team > count-1 we select **[self]** and **team [: count-1]**
- If we are resistance – I length of team == count-1 we select **[self]** and **team [: count]**
- Else we select **sorted_players_by_trustworthiness [: count]** if **[self]** is in **sorted_players_by_trustworthiness [: count]** else select **[self]** + **sorted_players_by_trustworthiness [: count - 1]**

After all these changes I was able to make the bot beat Bounder and Logicalton. Sometimes the bot gets beaten by Logicalton but the win percentage is very closer. I decided to call the bot **Fatality** as from the classic game **MortalKombat** and added in some monologues from the game for the bot to say at different points**.**



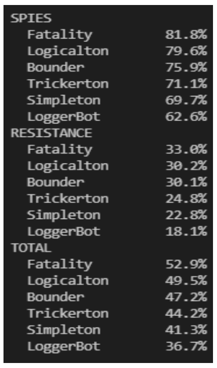
```
SPIES
  Fatality        82.7%
  Logicalton      82.2%
  Bounder         77.1%
  Trickerton      72.6%
  Simpleton       69.4%
  LoggerBot       60.3%
RESISTANCE
  Logicalton      30.9%
  Fatality        30.7%
  Bounder         30.1%
  Trickerton      24.2%
  Simpleton       23.4%
  LoggerBot       16.6%
TOTAL
  Logicalton      51.5%
  Fatality        51.4%
  Bounder         48.9%
  Trickerton      43.4%
  Simpleton       41.7%
  LoggerBot       34.3%
```

*Fig 12: Fatality lost overall by 0.1%v*



```
SPIES
  Logicalton      83.2%
  Fatality        82.0%
  Bounder         76.6%
  Trickerton      70.4%
  Simpleton       68.5%
  LoggerBot       60.5%
RESISTANCE
  Fatality        32.0%
  Logicalton      31.3%
  Bounder         30.7%
  Simpleton       23.8%
  Trickerton      23.7%
  LoggerBot       17.2%
TOTAL
  Logicalton      52.2%
  Fatality        52.2%
  Bounder         49.0%
  Trickerton      42.3%
  Simpleton       41.5%
  LoggerBot       34.5%
```

*Fig 13: Fatality ties with Logicalton*



```
SPIES
  Fatality        81.8%
  Logicalton      79.6%
  Bounder         75.9%
  Trickerton      71.1%
  Simpleton       69.7%
  LoggerBot       62.6%
RESISTANCE
  Fatality        33.0%
  Logicalton      30.2%
  Bounder         30.1%
  Trickerton      24.8%
  Simpleton       22.8%
  LoggerBot       18.1%
TOTAL
  Fatality        52.9%
  Logicalton      49.5%
  Bounder         47.2%
  Trickerton      44.2%
  Simpleton       41.3%
  LoggerBot       36.7%
```

*Fig 13: Fatality beats Logicalton*

## Analysis:

Keeping tabs on teams that failed mission or passed mission helped a lot in improving the win rate of the bot. Eliminating players who have voted down in 5th try did not help much in improving the win rate. I realize that some of the bots like Trickerton just votes randomly and Simpleton always votes True vote. This might have been a factor that caused this. **failed_missions_leadered** did not make any change in accuracy of guessing who the spy is. This might be because Simpleton always return True on sabotage and Trickerton always return True on sabotage if turn > 1 and **missions_passed_as_spy** only helped in increasing the accuracy of finding out if Trickerton is a spy. This was surprising as Trickerton is derived from Simpleton and yet it does not affect the accuracy of simpleton.
Using not just the probability of trustworthiness got from NN but a combination of it with number of failed missions' player has been on helped vote down team with potential spies and improved the winning rate of the bot a lot.

Figuring out optimal team from teams who had done successful missions and discarding teams or subset of team who had failed missions before helped improve the winning rate. Also not choosing to sabotage missions when you are spy and have another spy on team but with lower failed missions been on helped improve the winning rate of the bot.

## Overall conclusions:

Only relying on the NN which guessed the probability of a player being a spy only got the bot winning to some extent. To go all the way, I had to combine the result of NN bot with other logics. This made sense as guessing who the spy is only helps when you are in resistance. When you are a spy you already know the spies and there is no need of using NN.

To improve the accuracy in guessing the spy I have added 4 more input features - **mission_passed, mission_passed_as_spy, won_as_res** and **won_as_spy**. and made several improvements to the basic neural bot using nested decision trees to vote up or down a team, when to sabotage. How to select a team etc.

To conclude using NN along with the basic logics of how a human would think when playing the resistance game and making nested decision trees out of it helped the bot improve and come at top of all the other bots.

As there is always room for improvement maybe accuracy score on how prone a bot is to vote up a team or sabotage a game would help ease the game for the bot and may even help the bot win against experts.py. We could also add more input features or remove the ones that doesn’t impact the accuracy score much.

## References

https://www.udemy.com/course/machinelearning/learn/lecture/6760380#overview

https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da