# Research Report

**Researcher:** Samuel McCauley
**Mentoring Professor:** Dr. Daniel Lau
**Date:** November 6, 2018

## "Red Light/Green Light" Machine Learning

The goal of traditional Reinforcement Learning algorithms is to understand a specific environment and to pick the best **action** for a given scenario. For example: if a Reinforcement Learning agent is trained to understand the environment of Super Mario Bros., Mario's possible actions are walk left, walk right, run left, run right, and jump. If Mario was standing right next to a goomba, the agent would say that the *best* action for the scenario would be to jump so that he avoids touching the goomba and stomps on his head. Ultimately, it considers every possible action and guesses which action would play out the best for Mario.

The "Red Light/Green Light Method" has a similar goal, but it is more of a combination of Reinforcement Learning and Classification Models. The Red Light/Green Light Method is binary—its only goal is to decide if a potential action is a good (green) idea or a bad (red) idea. It is pure minimalism.

The inputs to the model are the environment—the coordinates of where Mario is, if/where a goomba is present, if/where a coin is present, etc. During training, Mario would be put next to a goomba. In one scenario, he jumps and avoids the goomba. That's a green light for Mario, he did good. The Mario and the goombas positions would be recorded and would be labeled as a 1 for training. In another scenario, Mario doesn't jump, the goomba hits him, and Mario dies. That's a red light for Mario, he did bad. Everything would be recorded again, but this time it would be labeled -1 for training.

That's at least the overarching idea of the Red Light/Green Light method. However, instead of testing the idea out with Mario, we tried something a little more practical…

## Training Method

In a simulated MATLAB environment, there exists two separate objects. One is a **Robot** object that has a floating point (x, y) coordinate as well as a pointing angle [0°, 360°). The other object is a **Destination** object that only has a floating point (x, y) coordinate. The goal of this is to get the Robot to navigate itself to the Destination using Red Light/Green Light Machine Learning.

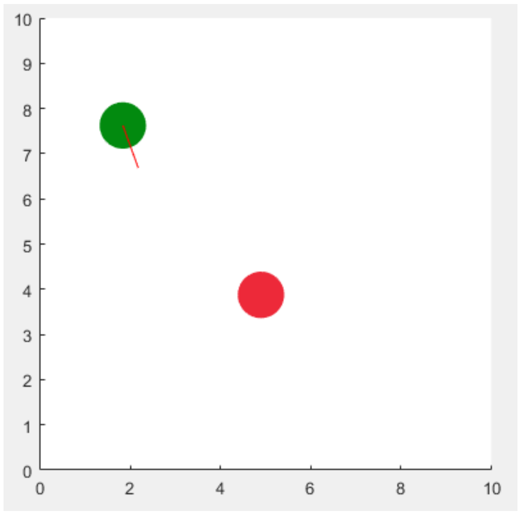

*Figure 1: Simulated MATLAB environment showing Robot (green) and Destination (red)*

The Robot has the capability to move using imaginary tank-like motors. The Robot's move function accepts two values, a left and a right motor speed. The Robot's forward and rotational speed are based off of these values. For example, if the left and right motor speeds are equal, the Robot will move in a straight line. However, if the left speed is greater than the right speed, the Robot will move clockwise. Thus, the Robot mimics the tread properties of a tank. See Figure 2 below for a depiction of these properties.

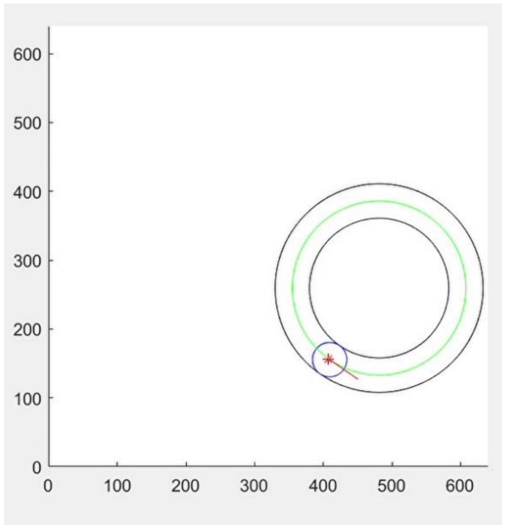

*Figure 2: Tank-like tread movement of the robot. In this example, the right speed is greater than the left. The green line is the projected path of the Robot.*

The Robot is trained by generating a list of training data. The length of this training data is dependent on a variable called TRAIN_LEN. To add data to the list, the training method starts by randomly generating an environment. A Robot is created with a random initial coordinate as well as a random initial pointing direction. A Destination object is also created with a random coordinate. Next, the Robot is given two random motor speeds, one left and one right. The movement is then simulated. If the movement gets the Robot closer to the destination *and* decreases the angle between the Robot and Destination, then the set of data is labeled as a 1—“green light”. Otherwise, it is labeled as -1—“red light”. The training data is organized as such: [dX, dY, Robot Angle, Left Motor Speed, Right Motor Speed, Label] where dX and dY are the distance between the Robot and Destination in the x and y axis.

This process is then repeated until half of the dataset is populated with “green light” labels and the other half is “red light” labels. The data is then fed to a decision tree algorithm. With the resulting model, a Robot can be told if a potential movement is good or bad by feeding the model a list of features.

## Results

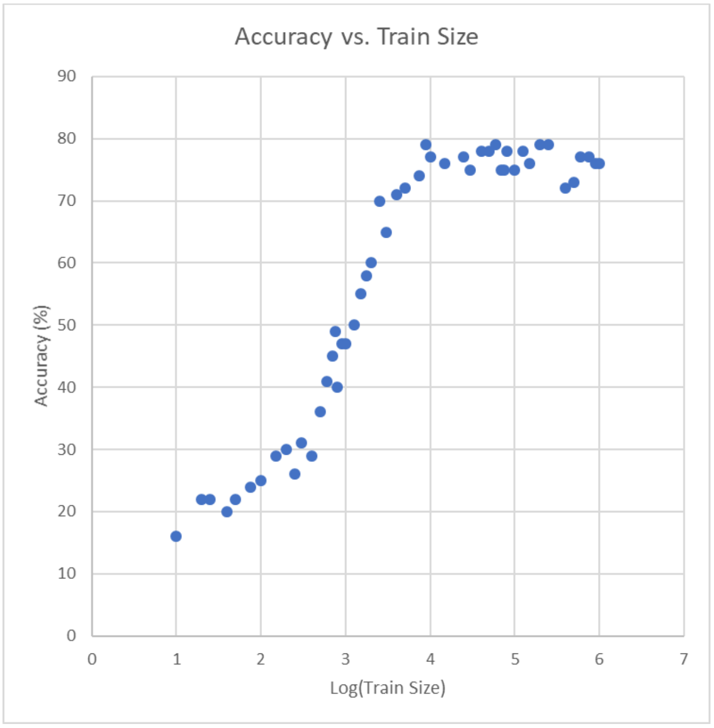

*Figure 3: Graph of Log(Training Size) vs. the Resulting Accuracy*

The results of this test were promising. The highest accuracy recorded was 79% and this first occurred at a training size of 90,000. Interesting takeaways from this data:

- Even at a training size of just 10 scenarios, accuracy is around 15%, which seems to be pretty impressive considering such a small dataset.
- After a training size of 100, the accuracy begins to rapidly increase until roughly 10,000, after which your accuracy returns by increasing training size begin to diminish.
- There appears to be an upper asymptote of around 80% accuracy. It appears this asymptote is reached around a training size of 10,000.

The largest problem with this method is certainly this limiting asymptote. When watching the results of the resulting model from large training sizes where this asymptote is reached, it seems that the model has a very hard time making the very last step of reaching the goal. In other words, all of these high training size models are incredibly good at getting close to the target, but once they are close, they start acting erratically. I can’t seem to pinpoint the exact reason for this behavior, but I think it may be that when the model is being trained, scenarios where the Robot accurately predicts a good movement to get to the target when it’s already close to the Destination are statistically rare—this case would be appearing towards the tail end of the Normally Distributed training scenario generation.

## Conclusion

In conclusion, I would say that the results were remarkable in comparison to the minimalism of the concept. I think the most important part of the experiment was the rapid growth period in the training. In my opinion, this is what makes the method interesting and potentially powerful. However, at the end of the day, I think that for a situation like this a plain regression would be a better algorithm choice.