Team 3 Topic

We decided to go with machine learning as our topic, and to look specifically into optimizing functional algorithms. Machine learning is an ever evolving field which requires frequent optimization. Its algorithms can make predictions and generalize based on historical data and map inputs to outputs. When it comes to functional algorithms, we are looking more into optimization rather than approximation, meaning finding the set of inputs that results in the minimum/maximum output rather than generalizing from specific examples.
Below are some relevant resources and scholarly articles that we will use to help us further define and navigate our problem.

https://scholar.google.com/scholar?q=convex+optimization+machine+learning&hl=en&as_sdt=0&as_vis=1&oi=scholart
https://people.eecs.berkeley.edu/~jordan/courses/294-fall09/lectures/optimization/slides.pdf
https://en.wikipedia.org/wiki/Machine_learning
https://web.stanford.edu/class/ee392o/cvxccv.pdf
https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/what-is-linear-regression/

Multiple strategies for games, especially games that rely on skill rather than sums (chess, tic tac toe, etc). These types of games often result in randomized movements that can be costly. Even calculated moves can be risky at times, but taking calculated risks is often the solution of choice in games like this. This is an inadequate solution to this problem as it is not optimized, and again is still risky. Though there is no way to completely eliminate risk in games, there is a way to optimize these games and the movement through them by eliminating guesswork and optimizing the game itself by making decisions based on the opponent's movements and having a calculated strategy already laid out.

Adding to Complexity: Maximizing score (with multiplier for squared variable), multiple algorithms for game, linear regression from running test (least squares, convex problem)

Report Draft

Abstract

This tutorial serves as a new modelling approach that aims to broaden the options and increase the success probabilities of the snake game. This is achieved through the optimization of path finding algorithms, which minimize processing time while maximizing success probability. Furthermore, a regression analysis is included to determine convexity as well as the validity of the relationship between the variables.

Introduction

There are many convex problems that exist within the topic of machine learning. It is an area that is consistently optimizable, as new ideas and concepts drive it to be increasingly complex. For this project, the objective is to optimize an existing problem in machine learning, specifically one that exists within functional algorithms in the common snake game. The algorithms that exist within this game make it so that game play relies on predictions, historical data, or skill rather than optimization. Randomized movements of this kind can be costly, even if risks are calculated. This is an inadequate solution to the problem, which is what prompted an optimization solution.

Snake is a game that is simple in theory but can be complex in nature. It requires the user to continuously eat apples while avoiding running into a wall or the snake's own body, which increases proportionally with the number of consumed apples. The point of the game is to get the highest possible score without dying.. This problem can be optimized, though not necessarily in a convex manner. To turn this into a convex problem, variables for regression analysis can be added. To formulate this, it is necessary to break down the component algorithms for analyzation, which upon solving should result in a higher success rate in game play.

The problem that was created to help solve this problem is not truly convex in nature. The minimization of time is in fact a convex problem, however the constraints used to get there are not convex. The maximization of success rate constraint is the main area that lack convexity. To make up for this, or rather to add to the convexity of the problem, a regression analysis helps to examine the relationship between the variables listed in the problem below. The problem's main variable in length, being that the result of the problem or algorithm changes almost proportionally with a change in length, making this a linear problem.

The problem itself is detailed below.

Minimize processing time

$$minimize \sum_{k=1}^{3} x_k t_k(L),\ with\ respect\ to\ [L,\ s]$$

Subject to the probability of success

$$\sum_{k=1}^{3} x_k p_k(L) \geq s$$

$$\sum_{k=1}^{3} x_k = 1$$

$$0 \leq x_k \leq 1\ \forall\ k$$

Where L is the length of the input argument (the length of the snake in this case), s is the minimum desired accuracy of the model, p_k is the probability of success of the algorithm for a given length, and t_k is the computational time of the algorithm for a given length.

Results

A. Algorithms and Protocols

Pathfinding algorithms are mathematical models which are computationally generated using methods derived from graph theory. In short, a pathfinding algorithm finds the shortest path between two nodes on a graph. For this project, three different algorithms were used for our assessment; the A* algorithm, the Breadth First Search algorithm, and Dijkstra's algorithm. These three algorithms all do the same thing (finding the shortest path from one node to another) but calculate their respective paths in different ways. For this project, these pathfinding algorithms were adapted to Python code and used to compute three different searching protocols for our game.

After the construction of the game "Snake" using the Pygames' library for Python, three protocols were created. All three protocols followed the same structure and the only distinguishing feature between them was what pathfinding algorithm they used to complete their computations. The protocols began by having the snake locate the shortest path between its head and the objective. If a viable path could not be found between those two points, the snake would then calculate the shortest path between its head and its tail. Regardless of which of these functions that were called, the snake does these calculations for every frame. If the snake can find no viable path between its head and either the objective or its tail, the snake assesses that it must restart with a new run.

It's important to note that the implementation of these algorithms were not ideal if the sole purpose of the models were to construct the maximum performing algorithm. This is because the purpose of the project is to explore practical functions of convex optimization theory, not to make a program that won the snake game. There are other, easier methods of playing snake that guarantees the survival of the snake until the game ends by pre-programing a list of specific directions the snake must follow. Instead, each of these protocols were constructed in a way in which they had flaws which would provide more interesting data to train the model on. It's important to note that the processing time of the algorithm was only calculated when the algorithm was making a calculation and not when the remaining code was running to ensure the primary convex optimization problem had reliable data.

When these protocols were fully implemented and the data from inside the game could be transmitted to an external .csv file, each protocol ran 1,000 times to collect data averages used to construct the regression equations. This was done by running 200 attempts at the snake game on 5 different sized boards. The variation in board size shifted the averages of the model higher so that the model is more applicable for a variety of snake game board sizes. By recording the length, processing time, board size, and currently used protocol, an accurate estimation for the probability of success and computing time could be calculated. Over 1,000,000 calculations were computed to generate over 100,000 data points which were input into the linear regression models.

B. Regression Analysis

Type of Regression

How its implemented

Convexity/Importance

While optimizing a vintage arcade game for processing time might not seem very important in itself, the importance of the model comes from the fact that it is very versatile and generalized.

Many models for playing the snake game rely on the game being played on the same sized board and would stop working if the size of the board was not what it was trained on. Regardless of the size of the board, the snake will still be able to pathfind and find the minimum processing time while maintaining a high accuracy using this model.

Conclusions

Current Problems/Future Development

Future Development of this model would include adding more input variables such as the size of the map and using other regression models such as quadratic regression. Improvements could be made to the various protocols to account for more specific situations the snake might find itself in such as what to do when the snake can't pathfind to either its tail or the objective. Additionally, increased complexity could be added to the game by introducing obstacles to the board which the snake also must avoid in addition to its body and the walls.

https://github.com/Christian-Martens-UNCC/ECGR-4115/tree/main