Project Plutus

Andrew Butler drewbutlerbb4@gmail.com
Undergraduate of Computer Science and Mathematics, UMD

Abstract

As this project continues to unfold there will be a variety of different techniques, algorithms, and ideas explored. Therefore, this abstract is dedicated more to the goals of the project, then to the specific implementations used throughout its development. This project is an attempt to implement various techniques and use them with benchmarks that are more applicative, than abstract. Using these benchmarks we will attempt to understand the benefits and detriments of various algorithms and design decisions.

Keywords

NeuroEvolution of Augmenting Topologies (NEAT), neural networks, genetic algorithms,
file organization, topological sorting, benchmark testing

1. Introduction

Project Plutus is an attempt to shed light on the value of the application of neural networks. Neural networks have seen an increase in use in recent years, partly due to the wide volume of research in the area following the AI Winter. The usefulness of neural networks is due in part to the fact that recurrent neural networks as a model are Turing Complete, but also due to the ability to easily make small and large adjustments to its structure. Neural networks then, are an adaptable model that can represent any mapping of inputs to outputs.

The question now becomes, what sort of algorithm can we use to learn models for certain tasks? This questions becomes more and more important as the volume of information available becomes larger and larger. But for some tasks, information is expensive, expensive to process, and/or incomplete. Our work then, becomes many times harder as working with incomplete information breeds incomplete models in the absence of function prediction.

Thus, our job is to create a model that predicts the mapping of various function based on the data provided. Now that our model is making predictions, it follows that sometimes our model will make incorrect predictions. A common way to make predictions is that of Occam's Razor. Occam's Razor states that the most likely answer to a question is the one that can be constructed with the fewest assumptions. However, Occam's Razor does not always make useful predictions in sufficiently complex problems.

Our problem has developed into a matching problem. Which models of prediction make accurate predictions for which type of problems? By providing benchmarks with real applications we begin to look at how our algorithms continually build up our models to make more accurate predictions. Based on this knowledge we can make predictions ourselves about what sort of problems a model will be useful for.

More specifically, however, Project Plutus focuses on models that pertain to accruing wealth. There are many advantages to this choice, some more obvious than others. As to the actual value of the money our models will attempt to accrue, I will leave for the reader to decide. However, money is a natural measurement for fitness, as oftentimes collection of money is the ultimate goal of the entity.

As a short ethical aside, there are some unsavoury side-effects to using money as a fitness determinant if not properly harnessed. For example, if our model was allowed to steal from others without repercussions our model, in its current state, would do so. Certainly, this action is unethical, but this is hardly our biggest problem. The problem becomes larger when more ambiguity is involved.

An example that applies directly to one of our benchmark tests is the problem of Wash Trading. A Wash Trade is a trade where an entity places a buy order for a financial instrument, and simultaneously places a sell order for the same financial instrument. The entity effectively buys and sells the instrument to itself. This creates misinformation that a volume of a certain instrument is being bought and sold for a certain price. Of course, we know these prices are fabricated. The entity then drives the price of the instrument in a direction artificially and can profit off of this. Our algorithm, if we are not careful, may learn this trait and use it to profit. Thus, certain ethical questions need to be considered when deciding on a fitness measurement. Just as the Greek God Plutus was blind and could not see whether he distributed wealth justly, our models will be blind to whether or not it makes wealth justly.

2. Development

During the development of this project many algorithms and benchmarks will be considered, and subsequently implemented. This document will be developed in three sections. The first section will look at implementation design, which will provide an in depth look at every design decision made. We may find multiple interesting choices at any design decision, and thus may discuss the results on the difference between the two on our benchmarks.

The second section will focus on benchmark results. We will discuss how each of our algorithms performed on the chosen benchmarks. This will include insights on to what caused periods of increase, decrease, and/or plateau. It will also include comparisons to how other algorithms performed on this benchmark

To close off there will be a section that contains an opinion piece about what the use cases of this algorithm are and what causes this algorithm to struggle. It will discuss what we would consider changing looking back and provide general thoughts on the method.

All of the implementations relevant to this paper will be written in Python, however data management and web scraping pieces may be written in Java.

3. Benchmarks
3.1 Poker
3.1.1 Game Overview

The first benchmark to be tested on will be the game of Poker. More specifically it will be eight handed, no-limit Texas Hold ‘em. This will be a five card Poker game with $1/$2 blinds and $200 buy-in for each player.

3.1.2 Information Given

Players will be given this information when it is their turn to act, as dictated by the game of Poker. Players will receive their two cards, the number of their chips in the pot, the number of chips they have not in the pot, and the cards on the board. Additionally, they have the number of chips another player has in the pot and the number of chips that same player does not have in the pot for every player in the game.

3.1.3 Actions Allowed

The players are allowed to call/check, raise, or fold. If they choose to raise they are allowed to raise anywhere from the minimum bet allowed by Poker rules at the situation and all of their chips.

3.1.4 Fitness Function

The fitness function we will use is money remaining. This could be adjusted to money per hand or money per player to normalize the results, so the models learn optimal strategies for different situations.

3.1.5 Possible Improvements

Include more information such as hand history or bet history to allow for the networks to try and predict another players strategy

3.2 London Stock Exchange
3.2.1 Game Overview

The London Stock Exchange has a long history, and thus a large archive of financial instrument data. Players will try and predict the price of the financial instrument at some point in the data based on data given before the point they are trying to predict.

3.2.2 Information Given

Financial Instruments data was taken from Yahoo! Finance. This data includes daily reports of the price of open, high, low, close, adjust close, and the volume of trades made that day. The data spans from the 1970's to August 18th, 2017. The data starts out sparse for the 1970's area and becomes more dense as the dates get more present. Portions of the data have been cleaned to remove empty data points.

3.2.3 Actions Allowed

The players will be able to predict any positive real number (limited by float precision).

3.2.4 Fitness Function

The fitness function here will be F
$F = -100 * \Delta f / f_0 + 100$
where $\Delta f = f_0 - f$
and f_0 = the actual price on the day to be predicted
and f = the predicted price on the day to be predicted

3.2.5 Possible Improvements

Predicts multiple points in the future (possibly scaled to a point every day)

4 Artificial Intelligence and Machine Learning Algorithms

4.1 NeuroEvolution of Augmenting Topologies

4.1.1 Implementation

4.1.1.1 Overview of NeuroEvolution of Augmenting Topologies

Evolving Neural Networks through Augmenting Topolgies [1] is a paper on a specific design of Topology and Weight Evolving Artificial Neural Networks (TWEANN). In it the authors, Kenneth O. Stanley and Risto Miikkulainen, outline a method called NeuroEvolution of Augmenting Topologies (NEAT). NEAT is a specific type of TWEANN with a number of interesting additions, more on that later. The methods of NEAT are best explained in the paper itself [1].

A gene is a connection from one node to another, including information like whether it is currently enabled and the weight of the connection. A collection of these genes is a genome, which is used to represent neural networks. The genome also has other auxiliary information to help with other processes. At the beginning an initial population is made with no genes, only implied input and output nodes. The initial population then goes through a round of mutation, where mutation adds random topology, bias, and weightings. This will be the first generation of genomes.

This generation then undergoes speciation, fitness scoring, culling, breeding, and mutation, in that order, to give us the next generation. First, we speciate the population. We divide the population into groupings called species, through a process called niching (more in 4.1.1.2). Fitness scoring is evaluating the networks and assigning a value to how well it did, so that we can determine how strong a network is in relation to the others. Then the culling begins, which is a process by which certain members chosen for low fitness are removed from the population to make space for new ones. These new species are produced from the breeding stage, where two genomes produce a child genome. Lastly, we mutate the genomes to add random topology, bias, and weightings.

This cycle is then repeated in order to attempt to iteratively improve the networks housed in the population. There are many intricacies to this procedure, some that I will talk about and others that I will not. The following sections provide a more clear representation of how the implementation functions and other works it pulls from.

4.1.1.2 Niching

Niching is the process of forcing splitting the population into sets of very different topologies, in order to search multiple areas of the solution space that are optimal. More formally, niching is the process of maintaining a subset of the Pareto front in an effort to find the max optima (or often minima). The form of niching used in this implementation is a technique called explicit fitness sharing [2] . This technique mandates that all genomes that share a species also share their fitness. Based on the species fitnesses, each species is then assigned a number of genomes to cull and a number of children genomes to be bred. Thus, the population removes the weak and breeds the strong.

Niching is important, because it protects innovation by allowing strong individuals to “cover” weaker individuals that are adapting new topologies that need to be further tuned. Since the number of individuals an individual can cover is proportional to its strength, areas that seem most promising receive the most resources for further research. In summary, niching forces the population to cover more of the Pareto front and promotes innovation by speciational coverage of innovation.

4.1.1.3 Fitness Proportionate Selection

Fitness proportionate selection is a method by which individuals are chosen for breeding (breeding is usually called crossover). This is a non-uniform, random method for selection. An individual’s chance of being selected is its fitness divided by the summation of the fitnesses of every individual available for selection. This means fit individuals are more likely to be chosen, but weak individuals can still participate in crossover, albeit less frequently.

4.1.1.4 Topological Ordering

In our implementation we need a way to decide which direction any connection is going. This is important, because our neural networks do not currently support recurrent neural networks. Thus, if a connection is made in the wrong direction it can break the network. We use partially ordered sets (posets) to maintain the directed acyclic graph. We choose one of the possible posets and use it as our set ordering. When we add a node to a topology we take to separate nodes add a connection from the one earlier in our topological ordering towards the node that appears later. Then we place our new node randomly between the two nodes in the topological ordering.

This preserves the directed acyclic graph and is fairly low cost to do. The larger problem is how to find the poset order when taking a genome from a saved state. The solution we employ discussed in the next section on Kahn’s algorithm.

4.1.1.5 Kahn’s Algorithm

Kahn’s Algorithm is a topological sorting algorithm for directed graphs. It is guaranteed to find a topological ordering on a directed acyclic graph. Therefore, this is the algorithm we will use to find one of the possible posets that our algorithm uses to maintain acyclicity. When restoring saved genomes from memory we must find the a poset to the graph we are restoring. We do not store this data with the saved genome, because this would be redundant data storage.

Kahn’s algorithm works by repeatedly picking a node with no incoming edges, removing its edges, adding the node to the poset, and repeating. This gives us a particular poset in the set of all posets, but this is exactly what we want.

4.1.1.6 Implicit Object Notation

Implicit Object Notation (ION) is a file format designed as a more raw form of JSON. ION is JSON with implicit attributes based on the structure of the classes. Attributes are assumed to be in the order of which they are displayed in the source code. Any attribute not to be saved in the file format should be marked with the flag # ION=FALSE. Otherwise, data is stored like in csv file with the caveat that structural information is implied with curly brackets, square brackets, and semicolons. Further documentation as to the methods used are in the works.

4.1.2 Results

Awaiting test results to analyze

4.1.3 Insight

Awaiting Results section

TODO

- Re-cite References
- Add results
- Finish 4.1.1.6

References

1. Kenneth O. Stanley, Risto Miikkulainen; *Evolving Neural Networks through Augmenting Topologies*, MIT Press Journals
   Accessed From: http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf
2. D. E. Goldberg and J. Richardson, “Genetic algorithms with sharing for multimodal function optimization,” in Proc. 2nd Int. Conf. Genetic Algorithms, J. J. Grefensette, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1987, pp. 41–49
   Accessed From: This document has been particularly elusive, so outlines of the methods were taken from other documents
3.