Symbolic regression using Genetic Programming leveraging Neural Information Processing

Nanna Grytzell

Master's thesis
2021:E6

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Abstract

Regression analysis conducted with traditional mathematical methods can be sub-optimal if the exact model of the observed data is unknown. Evolutionary computing (EC) and deep learning (DL) are viable alternatives, since regression performed with these methods tends to be less dependent on a particular model. EC are especially flexible, because they are capable of performing symbolic regression. A subfield of EC and DL is genetic programming (GP) and artificial neural networks (ANN), respectively. This master thesis examines the effects of giving a genetic programming system neural information processing capabilities, in order to bridge the gap between ANN and GP. The approach is to compare GP, in its standard formulation, with 1) GP that speciates using an ANN, 2) GP that extends the function set with ANNs. Two methods are used to measure the prediction error. The effect of the first approach is an increased noise in the convergence. This leads to an enlarged spread of the prediction error for one of our two error measures, and a mainly unchanged error for the other. The effects of the second approach is an increase in accuracy for one of the error measures, and a decrease in bloat.

Acknowledgements

I am truly grateful that Oxide has given me the opportunity to write this masters thesis. It has been a thrilling and worthwhile experience. I want to thank Lars Hård and Axel Stålhand for all the support, guidance and feedback that they have given me. I also like to thank Anders Brodin for help and guidelines, and all co-workers at Oxide whom took the time to read this report.

Contents

1 Introduction **1**
1.1 Oxide . . . 2
1.2 Objective . . . 2
1.3 Constraints and limitations . . . 4
1.4 Technical approach . . . 4
1.5 Related work . . . 5

2 Background **7**
2.1 Evolutionary algorithms . . . 7
2.1.1 EA lingua . . . 7
2.1.1.1 Biology and EA . . . 8
2.1.1.2 Search and solution . . . 8
2.1.2 Genetic algorithm . . . 9
2.1.2.1 Selection and replacement . . . 9
2.1.2.2 Reproduction . . . 10
2.1.2.3 Mutation . . . 11
2.1.3 Genetic programming . . . 11
2.1.3.1 Tree structure . . . 12
2.1.3.2 Stack-based representation and evaluation . . . 12
2.1.3.3 The combined set C . . . 15
2.1.3.4 Bloat . . . 16
2.1.4 Diversity . . . 17
2.1.4.1 Fitness sharing . . . 18
2.2 Artificial neural networks . . . 18
2.2.1 The structure . . . 19
2.2.2 Training . . . 22
2.3 K-means clustering . . . 23

3 Method **25**
3.1 Data . . . 25
3.2 Clustering . . . 25
3.3 Multilayer perceptrons . . . 27
3.4 Genetic algorithm . . . 29
3.4.1 Fitness function . . . 30
3.5 Genetic programming . . . 30
3.5.1 The combined set C . . . 31
3.5.1.1 Closure . . . 31
3.5.1.2 Sufficiency . . . 32

3.5.2 Population initialization 32
3.5.3 Genetic operations 33
3.5.4 Fitness function 33
3.5.4.1 Search space 34
3.5.4.2 The optimization problem 34
3.5.5 Speciation 35

4 Results 37

5 Discussion 51

6 Conclusions 52

7 Further research 53

1 Introduction

Regression analysis is about estimating a relationship between a dependent variable and one or more independent variables. The dependent variable and independent variable(s) are also known as the output and input variable(s), respectively.

Most people associate regression with classical methods such as linear regression and statistical modelling. These types of regression models are based on assumptions about the the observed process. The assumptions are often statistically oriented and infer a probability distribution from the observed process. Classical regression has good generalisation abilities and the resulting model is transparent; every model parameter can be explained, and one can tell how significant each model parameter is for the result. The drawback is that the model design requires prior knowledge about the underlying process, and that the model assumptions are not always fulfilled or static over time.

Symbolic regression is a good alternative when the observed process does not meet the model assumptions made in classical regression. This approach to regression is quite different from classical regression in that no prior knowledge is needed about the observed process. Furthermore, symbolic regression searches for model structure in addition to model parameters. The resulting model is a mathematical expression, or even a computer program, that optimizes the fit against the observed data. The freedom to decide the model structure comes at a cost: the search space is vastly increased. Also, there is no analytic way to solve the regression problem. Efficient search algorithms is thus of high importance when performing symbolic regression.

Symbolic regression using genetic programming (GP) has proven to be a highly successful approach for regression. Making customized regression in relation to specific problems feasible, without restricting the model search by prior assumptions. Another very positive side-effects is the ability to explain how it performs the computations due to the fact that it actually generates the code/math. However, genetic programming in its standard formulation (in the context of symbolic regression) lacks the ability of pattern recognition.

Artificial neural networks (ANN) are interesting constructs from a mathematical standpoint since they are capable of arbitrary function approximation. The task of classifying data or recognizing patterns in data, is very well handled by this type of model. The downside is rapid decrease of transparency/explainability as the complexity of a network increase, considering

ANNs black boxes.

This gives a good reason to explore the effect of extending GP with neural information processing capabilities.

1.1 Oxide

Oxide is a high-end R&D (Research and Development) company working in artificial intelligence (AI), machine learning and large-scale data processing. The company primarily develops a new AI engine, Polychaos, that can be applied in various information rich domains such as life science, finance, environment and space engineering. It is basically an advanced research engine built using a generative approach with evolutionary computation models performing information refinement at scale as well as detecting complex patterns in high-dimensional information spaces.

Regression is a fundamental operation in Polychaos and is used to merge great quantities of numerical data into distinctly defined signals. At the moment, symbolic regression is performed using GP (without neural information processing capabilities). Oxide aims to extend the current GP models performing regressive tasks in Polychaos with neural functions in order to improve capabilities regarding sequences/patterns and enable alternative processing based on that. It is this regression task that has set the foundation for this thesis.

1.2 Objective

Genetic programming is the center of this report and the main objective is to examine potentially positive and negative effects by 1) having GP speciate using an ANN 2) extending the function set for GP with neural information processing.

The study will include a number of related investigations of:

- the effect on the precision of the symbolic regression models evolved by the GP
- the gene frequencies within the population
- the size of evolved programs

- the general evolutionary dynamics

The study of gene frequencies may give insight in the gene interplay, displaying how the GP choose (or use) different genes from the gene pool. This gene interplay might change when different sets of functions and terminals are handed to the GP, where some genes will be favoured and others will be suppressed.

The size of evolved programs is of high relevance from a parsimonious point of view. Large programs are computationally expensive which may also slow down the evolutionary process significantly. Given the number of possible combinations of possible programs from a given set of functions and terminals, large programs extend the search space significantly, which makes size a central topic.

This thesis covers a wide range of biological onsets, where each lead to the next. The step prior to the actual GP has been put together by recursively asking:

Question: How will the GP leverage neural information processing?

Answer: By incorporating pre-trained neural networks into the GP.

Question: How will this ability be used?

Answer: Two different approaches will be tested. The first approach is speciation using a multi-class classification network. The second approach is by adding pre-trained binary classification networks to the combined set.

Needing: One multi-class classification network and several binary classification networks.

Question: In order for the ANNs to recognise classes of the data, the data has to be partitioned in some way. How?

Answer: By assigning the data points to clusters.

Needing: A clustering method.

Question: If each ANN is a binary classifier, several networks have to be trained. This sounds like a tedious task to do by hand. Is there a better way?

Answer: Yes, a GA will be implemented for this task.

Needing: A genetic algorithm.

1.3 Constraints and limitations

The following parts has to be submitted in order to complete the thesis objectives:

1. Partition the data
2. Pre-train ANNs
3. Implement a GA
4. Implement a GP

It is not possible to dig deep into each of these fields without outpace the time limit for this thesis. Hence step 1–3 has been kept simple. The pre-trained ANNs are restricted to shallow neural networks (yielding a maximum of two hidden layers), to not sacrifice to much of the GP's transparency. The partitioning of the data has by design been performed using a simple baseline method, k-means clustering. This method partition the data set into k clusters based on k cluster centers, where each data point belong to the cluster with the closest cluster center. This is not a main focus in the thesis, but rather an approach to pre-process the data into relevant patterns. The aim with the GA has been to produce a number of robust and simple trained networks for pattern recognition where networks are combined and the use optimized by the GP. High accuracy is of secondary importance in this context.

There exists many approaches to model the evolutionary process, that GP are attending. Despite that GP being the center of this thesis, comparisons to other, more complex approaches is outside the scope of this work. Making GP in its standard formulation, the single baseline in this report.

The data I use in this report consists of a single time series, hence the objective is to study the differences that may occur when leveraging neural information processing. Using several time series would yield a more generalised model, which is outside the scope of this work.

1.4 Technical approach

A large part of the work has been allocated to implementing a custom GP providing the flexibility and insights required for the research work.

The implementations has been conducted in the programming language Julia. This programming language provides easy high-level numerical computing, similar to that of Matlab and Python, and at the same time supports general programming. Furthermore, code written in Julia can have nearly as good performance as code written in a statically-typed language (such as C) [1]. Hence this approach has enabled high-speed execution in terms of compilation and multi-core parallel processing.

Also the GA and ANN I use are custom developed for the task at hand, making the workflow efficient and highly integrated.

1.5 Related work

The data used in this study is a financial time series, making literature from the intersection of computer science/math and financial forecasting relevant. Some has used ANN to make trading rules and/or predictions. One for instance is Vijh et al. [2], they trained ANNs to predict stock price and they report that it outperformed random forest. Shortly, random forest is a "forest" or ensemble of decision trees, where each decision tree models the observed data. The output of random forest is given by the average of the output from each individual tree. Vijh et al. [2] looked at historical data, such as past stock price and volume of shares traded of that particular stock.

Some papers has started to incorporate ANNs with GAs by letting the GA decide the structure and weights of the network [3, 4]. This way of combining evolutionary algorithm (EA) and ANN is called neuroevolution [5]. Yu et al. [6] wrote an review about ANNs in finance and economics forecasting. They state that using EAs, the ANNs avoid getting stuck in local optimum. Lastly, the authors conclude that "*there is no doubt that the prediction performance of neural networks is improved by integrating it with other technologies*" (including EA), [6, chap. 7].

Researchers have actively been applying GPs in forecasting, e.g. Grosan and Abraham [7] implemented two different kinds of GPs, and also an ensemble of these two. As for comparison, an ANN and a so called neural fuzzy network (NF) was also implemented. Grosan and Abraham [7] used two different stocks when testing each model and the result was that the ensemble GP outperformed the ANN and the NF on one stock, and on the other they performed equally well. The authors think that GP could play a prominent role in stock price modelling problems. Neely et al. [8] implemented a GP that made trading rules for six exchange rate series. They report that their result “*indicate that the trading rules are detecting patterns in the data that are not captured by standard statistical models*” [8]. Further, Yu et al. [9, 10] published two papers focusing on GP finding technical trading rules. In their first paper they studied international short-term capital flow between Taiwan and four foreign countries, which did not lead to any remarkable discoveries [9]. A year later Yu et al. [10] published a second paper, searching for technical trading rules based on S&P 500 index. Now, they had tweaked their GP approach a little and this time the results where more promising. Yu et al. [10] report that their GP could outperform buy-and-hold, i.e. the strategy of buying a stock and hold it for a long period of time, regardless of fluctuations in the market [11]. McDermott et al. [12] showed in their work that GP, and variants, can perform well in predicting out-of-sample over short time-horizons.

The work most similar to this thesis was made by Arshad et al. [13], but their focus is on wind power forecasting. They trained 5 different ANNs to forecast the electrical power produced at wind farms within a certain region. The trained ANNs where then assembled with a GP that performed symbolic regression. Their model outperformed each of the single ANNs.

..

2 Background

This section aim to give the theoretical background needed to understand the partitioning of the given data set and the biological onsets used in this master thesis. The first section cover evolutionary computation, including evolutionary algorithms (EA), genetic algorithms (GA) and genetic programming (GP). The second section provides background regarding artificial neural networks (ANN) and the third, and last, section explains the k-means clustering algorithm.

2.1 Evolutionary algorithms

An evolutionary algorithm is a metaheuristic optimization algorithm, meaning that it is a problem-independent strategy that guide the search for an optimal solution. This type of algorithms are inspired by nature, designed to simulate evolution. The EA consist of a set of individuals, called a population, where each individual represents a solution to the problem at hand. In every generation, i.e. iteration, the population transforms through selection, reproduction and mutation. By repeated generations, evolution of the artificial population takes place.

Two main families of evolutionary algorithms are:

- Genetic Algorithms, evolve a genome consisting of solutions to a given problem (in most cases a vector of numbers)

- Genetic Programming, enable evolution of programs and functions that are executable

The subsequent sections will first clarify the meaning of some words that is used through out this report. This will be followed by an introduction of the standard variants of GA and GP. Lastly, diversity will be presented, and explained why this is such an important subject of EA.

2.1.1 EA lingua

Terms from fields, such as biology and mathematical optimization, is commonly used in EA. This section aims to clarify the meaning of words lent from these fields.

2.1.1.1 Biology and EA

In real organisms the genome is the total genetic material coded in the nuclear DNA, i.e. the genes [14]. Genes are normally organized in certain sets, called chromosomes [15]. The genome may also be referred to as the genotype of an organism [16]. What the genome is expressing, i.e. what characteristics or traits that are observable, is called the phenotype of an organism [17].

These terms also occur in EA lingua. An organism is usually referred to as an individual. An individual has a genome, consisting of one or more chromosomes, that encode a solution to the problem at hand. The genome, or genotype, is the raw set of chromosomes (and hence genes) of an individual. The phenotype is attained by rendering the genome, such that one can see what solution the genome is encoding.

Furthermore, an EA individual is also given a fitness value based on its ability to solve the problem at hand, and this measure is used for individual comparison. In real life, Darwinian fitness (relative reproductive success) should be maximized; in EA the fitness should be optimized and depending on the problem this can mean either minimizing or maximizing the error.

2.1.1.2 Search and solution

In mathematical optimization, one have an objective function $H : A \mapsto \mathbb{R}$ that should be optimized. All points $\boldsymbol{x} \in A$ are called candidate solutions or feasible solutions [18, 19]. The domain A of H is called the search space or choice set [18].

In EA, the objective function is know as the fitness function (the function that determines the fitness value of an individual). The domain A is know as the search space and the dimension of the search space is equal to the maximum chromosome length $\ell_{\max}$ in the population. Let C denote the set of all genes available in the population, then

$$A = \{\boldsymbol{x} = (x_1, x_2, \ldots, x_{\ell_{\max}});\ x_j \in C_j \subset C,\ j = 1, \ldots, \ell_{\max}\}. \tag{1}$$

All elements $\boldsymbol{x}$ of the search space, i.e. all candidate solutions, will simply be referred to as “solutions” in this report. Further, if an individual is said to “solve” a problem, this means that this individual represents a candidate solution.

2.1.2 Genetic algorithm

The human genome consists of 46 chromosomes, or 23 chromosome pairs, meaning that we have two complete sets of chromosomes. This makes humans so called diploid organisms. An individual in a GA is usually haploid, i.e. has only one complete set of chromosomes. The genome of a GA individual in a population is typically a fixed-length arrays of binary or real numbers, but there may be other forms of representation as well. The initialization step is often fairly simple; randomly generate individuals. The steps that follows are:

- Selection — Select the individuals that will be allowed to reproduce. Often this is based on each individual's fitness value. Individuals with better fitness are more likely to be selected.
- Reproduction — Let selected individuals breed by pairing up the parents. The genes in each pair of parents are mixed by so called crossover, resulting in offspring that has inherited parts from both parents.
- Mutation — Transform the population and/or the offspring through mutation. Each individual will be mutated with a probability p_m. Sometimes the best individuals cannot be mutated.
- Replacement — Renew a part of, or the whole population with the offspring.

One iteration of the steps above is called a generation, and will be repeated until a stopping criteria is fulfilled. A typical stopping (convergence) criteria is: if the fitness of the n best individuals has not improved in the last x generations, then stop.

Mainly, there are two kinds of GA's: steady-state and generational GA. The first one mentioned, replace a subset of the population with the offspring, while the latter substitute the entire population with the offspring. The rest of this thesis focus primarily on the steady-state GA.

2.1.2.1 Selection and replacement

A basic selection and replacement method is selection of the best and replacement of the worst. Let the population size be N_{pop}. Then select the $n \leq N_{\text{pop}}/2$ individuals with highest fitness. Let them form pairs and let each pair have 2 offspring, resulting in n offspring. Let these n offspring replace the n worst individuals in the population.

A popular selection method is tournament selection [20]. Here, draw 4 individuals from the population, let the 2 individuals with highest fitness mate, resulting in 2 offspring. Replace the 2 individuals with lowest fitness. Continue in the same manner; draw 4 individuals from the population without replacement until there are no more individuals to be drawn.

2.1.2.2 Reproduction

In order to renew the population, the individuals need to reproduce. The simplest case is through single-point crossover. Suppose we have a pair of parents and they have one chromosome each. Randomly generate a break point on the chromosomes and let the subsequent bits in each chromosome be exchanged. This gives us two new offspring, see figure 1. Generating two break points on each chromosome yield two-point crossover and so on. If the chromosomes are arrays of integers or floating-point numbers it is

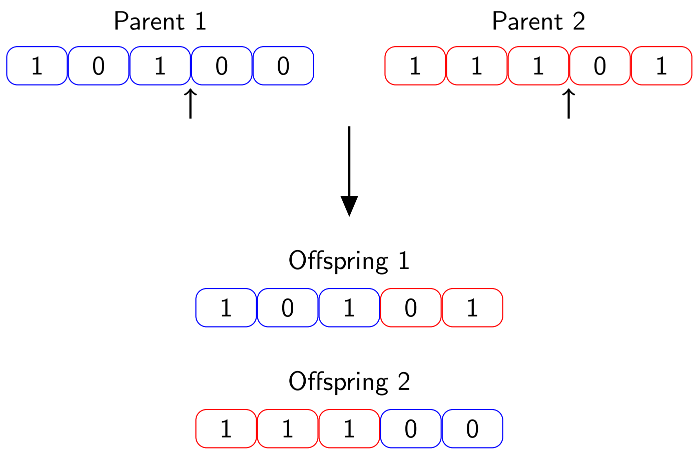


Figure 1: Single-point crossover. In this example the break point is placed after the third gene. The subsequent genes are then altered to form two new offspring.

also possible to use a method called blending crossover. Each gene is a convex combination of it's two parents' genes at the same location. Given two parents, each with one chromosome of length ℓ. Let g_j be the j'th gene in the chromosome. Then an offspring's genes would have the following formula:

$$g_{j,\text{offspring}} = r \cdot g_{j,\text{parent1}} + (1-r) \cdot g_{j,\text{parent2}}\,, \quad r \sim \mathcal{U}(0,1),\ j = 1, 2, \ldots, \ell.$$

2.1.2.3 Mutation

Mutation is ultimately the main source of variation in any living organism and in principle the same applies to evolutionary algorithms. Mutation operates by randomly modifying specific positions in otherwise preserved chromosomes rather than creating new chromosomes by crossing over. High mutation rates may refresh a population that has lost some of its genetic diversity [21, chap. 3]. For some problems, the phenotype of the top individuals may be to tender to manage mutation. In those cases one may protect the n best individuals from mutation and let the rest undergo mutation. This is called elitism [22].

One way of mutate an individual is by point mutation. Here each gene in a chromosome will be mutated with a probability p_m. If mutation occurs, the gene will be replaced with a new randomly generated gene. For example, let the chromosome `{1 1 1 0 1}` be mutated. Say that the first and fourth element will mutate and this may result in `{0 1 1 0 1}`, see also figure 2.

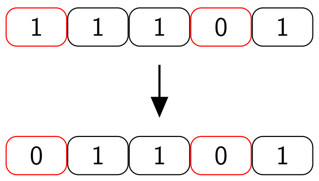


Figure 2: Here a chromosome is depicted, with the intention to explain point mutation. Each gene in the chromosome will be mutated with a probability p_m. If mutation occurs, the gene will be replaced with a new randomly generated gene. In this example the first and fourth gene is mutated and both genes are replaced with the gene `0`.

2.1.3 Genetic programming

A genetic programming system (GP) follow the same steps as a GA, but it differs at the genetic representation. A GP individual can have genomes with dynamic length, that represent a computer program solving the problem at hand. The genetic representation allows for program to be directly in-place or may require a translation from genes to an executable program. A GA will gradually tune a number of model parameters (genes), while the GP has freedom to select functions, terminal elements as well as construct the structure of the model. This freedom is accompanied by a vast growth of the search space.

Instead of a binary vector the chromosome could for example now encode a parse tree instead, such as the one seen in figure 3, that encode the expression $(8 + x) \cdot \cos(a)$.

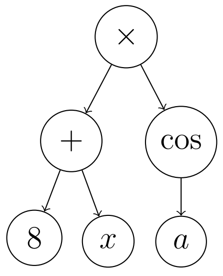


Figure 3: A expression tree forming the expression $(8 + x) \cdot \cos(a)$

The set of genes is denoted $C = F \cup T$, where F is the function set and T is the terminal set. The function set, as the name suggest, contains functions, or operators, that needs at least one input in order to produce an output value. The terminal set contains variables or constants and they serve as inputs for the previously mentioned set. An example of a function and terminal set is:

$$F = \{+, -, \times, /, \cos\}, \qquad T = \{a, x, 8, \pi\}.$$

The two following subsections describes shortly two genetic representations used in genetic programming.

2.1.3.1 Tree structure

This is the traditional genetic representation in a GP and was initially presented by Koza [21]. This structure is used in several programming languages, where LISP is convenient to use due to its easy access to the structure [21].

Each chromosome represents a tree structure where the nodes contain an operator and the leaves contain a variable or constant. As an example: an individual representing the expression $(8 + x) \cdot \cos(a)$ would have the tree seen i figure 3. It is also suitable for classification and decision making, e.g. should you take a walk? figure 4.

2.1.3.2 Stack-based representation and evaluation

A stack is a data structure that can be used for efficient genetic representation leveraging so called postfix notation (this is the structure that will be used in this thesis). The postfix notation is also known as reverse Polish notation (RPN) and were used in some of the first electronic calculators, such as

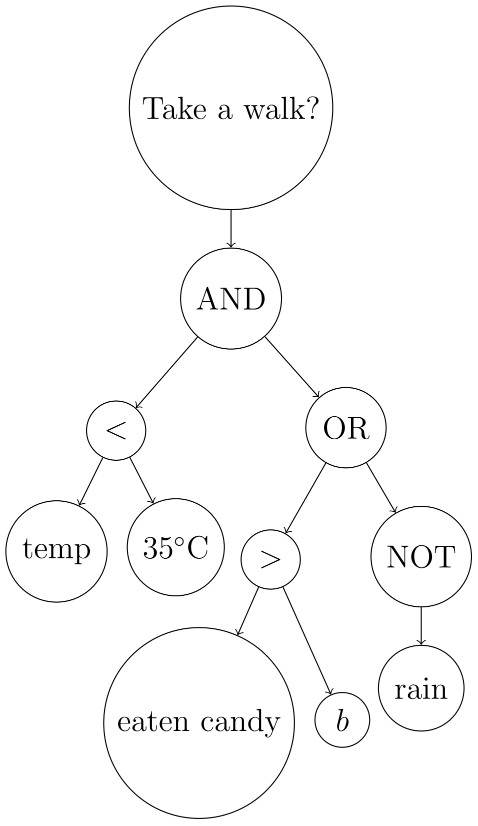


Figure 4: A decision tree, telling if you should take a walk or not.

Friden EC-130 and later by HP in their pocket-size series, named Hewlett-Packard Voyager [23, 24]. Today, RPN is used in stack-oriented programming languages such as FORTH and advanced scientific calculators. One major benefit is instruction order fully determines precedence order when evaluating an expression [25].

In order to evaluate a expression written in postfix notation, one needs a stack. For example, say we have a operator taking 2 inputs, in conventional (infix) notation this operator will have one operand on each side, like:

```
10 + 6
```

To make this suitable for a stack, this has to be rewritten in postfix notation:

```
6 10 +
```

Here, the operands are placed first in line and the operator come last. The reason for this is, when working with a stack, the operands are pushed on the stack, first 6 then 10. The operator will then pop the top two elements on the stack, namely 10 and 6, evaluate them and push the answer (16) back on the stack.

Returning to the expression $(8+x)\cdot\cos(a)$, mentioned above, this would now be written as

```
a cos x 8 + ×
```

A second example is depicted in figure 5. A merit of the postfix notation

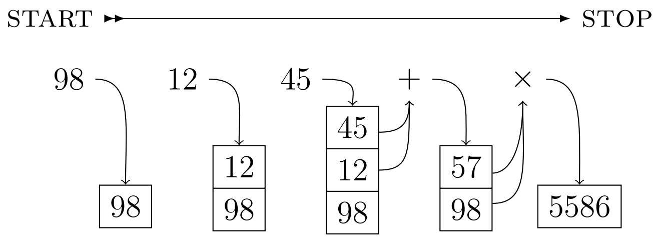


Figure 5: A stack evaluating $(45 + 12) \cdot 98$.
The postfix notation is `98 12 45 + ×` . First 98 is pushed on the stack, followed by 12 and then 45. When reaching `+` the top two elements on the stack are popped and the answer is pushed on the stack. Then `×` poppes the top two elements and pushes its answer on the stack.

is that there is no need for parentheses. As an example, the expression in figure 5, written with postfix notation, is

```
98 12 45 + ×
```

and will give the answer 5586. While writing the same expression with infix notation will need parenthesis in order to retrieve the same answer.

Using this structure, it is possible for a chromosome to have genes that are not used. Consider the chromosome `{a + b -}`. The first element will be pushed on the stack, then the second element will need two values from the stack, but there is only one value. Hence the second gene will be unexpressed. Further more, the chromosome `{a b c -}` will encode the expression $c - b$, i.e. the first gene will not be used.

2.1.3.3 The combined set C

In *Genetic Programming - On the Programming of Computers by Means of Natural Selection*, Koza [21] introduced a special new element to the combined set, which he called the ephemeral random constant $\Re$. This terminal has been widely used ever since. At the initialization of the algorithm, this terminal generates several random constants to the combined set. It is then left to the GP to find the most suitable constant (or constants) that can be achieved by combining the constants generated by the ephemeral random constant.

Mainly, there are two properties regarding the combined set C that should be considered when choosing the elements for this set: closure and sufficiency. An explanation of these two properties will follow.

Closure: Every imaginable combination of the elements in the combined set C can possibly occur during evolution of the artificial population. It is important that no combination cause an error, i.e. every function in F should be able to take as argument, any output that may be produced by any function in F and any terminal value from T. This is the property of closure and in order to fulfil this, it is normal to make adjustments to some functions.

The arithmetic operation of division is not feasible if the denominator is 0. Hence, the protected division function is used and defined as

$$\text{pdiv}(x,y) = \begin{cases} x/y, & \text{if } y \neq 0 \\ 1, & \text{if } y = 0 \end{cases} .$$

The natural logarithm is undefined when its argument is 0. Therefore, the protected natural logarithm function is used and defined as

$$\text{pln}(x) = \begin{cases} \ln(x), & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases} .$$

If the sinus function receives a number that is equal to ∞ (which is possible during program execution) the sinus function do not know what to respond. The protected sinus function is thus defined as,

$$\text{psin}(x) = \begin{cases} \sin(x), & \text{if } x \in (-\infty, \infty) \\ 0, & \text{otherwise} \end{cases} .$$

Sufficiency: Despite that any combination of elements from C may occur, this alone will not be enough if the elements in C are unable to express a solution to the problem at hand. In order to fulfil the property of sufficiency, one need to believe that the elements in C are capable of expressing a solution to the given problem.

Extending the function set significantly in order to guarantee coverage is often not an option due to the increase in search space of possible programs. The function set is often designed with a fair amount of understanding of the problem to be solved.

2.1.3.4 Bloat

Generally speaking, there is a strong preference for parsimonious explanations and one could wish that the dynamic chromosome lengths in a GP would yield parsimonious solutions. Unfortunately, this is rarely the case. As Koza [21, chap. 1] states: "*Genetic programming does not generally produce parsimonious results (unless parsimony is explicitly incorporated into the fitness measure). Like the genome of living things, the results of genetic programming are rarely the minimal structure for performing the task at hand.*".

A manifestation of this phenomenon is so called bloat. Which mean that after some generations, the individuals' genomes grow rapidly in size without also increasing their fitness. This results in unnecessary complex solutions that is computational expensive and likely over-fit, i.e. poorly generalized [26].

What causes bloat? There is no straight answer. Researchers has over the years developed several theories, trying to explain this phenomenon. Two well known theories are the crossover bias theory and the fitness-causes-bloat theory. The first mentioned, states that crossover in itself does not affect the average size of programs, but it changes the distribution of the size of programs. Small programs usually yield trivial models, while large programs are able to express more complex relations giving these a better fitness. Resulting in selection of somewhat larger programs in the selection step, thus increase the average size of programs in the next generation, and so on. The fitness-causes-bloat theory states that there are more large programs than small programs, hence the number of large programs of a given fitness is likely greater than the number of small programs with the same fitness. The programs grow in size, simply because there is more of them [27, 26].

Numerous techniques have been proposed to combat this issue. The most trivial method is to have a size limit; no offspring above a certain size will be introduced to the population, or it might be added to the population but with a very bad fitness, such that it is likely to die in the next generation. Other arrangement to control bloat is so called size fair crossover and size fair mutations, where these genetic operations will not result in a "unfairly" big chromosome. Shrink mutation is considered a rather direct approach to combat bloat, by applying mutations that shrink an individual's chromosome. Lastly, as quoted above, to control bloat, one may also incorporate minimization of it in the fitness measure. These techniques are further explained by Poli et al. [26, chap. 11].

2.1.4 Diversity

Diversity in the population is central to efficient exploration of the entire search space of possible solutions. Decreasing diversity can lead to premature convergence and make the algorithm stuck in a local optimum, i.e. the algorithm cease to, in an efficient way, explore the search space.

In genetics, diversity is counteracted by two phenomena, the concepts of fixation and genetic drift. Individuals within a population may hold different variants of a gene, i.e. alleles. If something befalls, e.g. the environment changes, the different alleles might be unequally suited for the new conditions. This may result in the effect that certain alleles will give an advantage to those individuals who hold it. Eventually, this better suited allele will increase in the population, and wipe-out the other alleles. When this better suited allele is the only variant present in the population, this gene is said to be fixed. Fixation may also happen due to chance, i.e. "sampling error" when sampling from the gene pool [28].

Fixation due to chance is an example of genetic drift. It is characterised by a random change in the allele frequency within a population. This is in contrast to natural selection that is driven by some kind of selection rule [29].

Basically, the same applies to an EA; both phenomena decrease diversity and degenerate the search for a global optimum. In order to avoid loss of diversity, mutation is of high importance. Mutation encourage exploration, but it has to be moderately applied, too much will weaken the exploitation (which is needed for fine tuning). Unfortunately, solely tuning the mutation rate will not fix the issue, which explains why basic GA/GP often are insufficient when one is facing a complex problem. Diversity is connected to all parts of an EA and to preserve it, different niching method has been proposed.

In nature, a species' niche is its role or position in a "community" (which in this case is consisting of all species in the given environment). In order for a community to be stable, species need to have different roles, i.e. niches [30]. Hence, by applying a niching method to an EA, the artificial population is encouraged to specialize on different parts of the search space. The section below cover a niching method called fitness sharing, for more methods the reader is referred to the PhD thesis by Gustafson [31].

2.1.4.1 Fitness sharing

In nature, individuals whom are close to each other compete for limited resources in a crowded area. One can mimic the competition among individuals regarding limited resources by applying a niching method called fitness sharing. It is often included to increase diversity in the evolutionary process by promoting individuals in low-density regions in the search space. Thus, preventing the population from getting stuck in a local optimum.

To measure the "closeness" of individuals, one usually chooses between genotypic or the phenotypic distance, where the first is based on the genome and the latter is based on what the genome is expressing. Next, one need a number of how many individuals that are sharing resources with an individual i. Individuals far apart, will probably not affect one another. Hence, a threshold σ is set, such that individuals with a distance greater than σ are considered to be too far apart to be competing about the same resources. On the other hand, individuals that are less than σ apart, do compete about the same resources and the effects of that are summed up in a niche count m_i,

$$m_i = \sum_{j=1}^{N} sh(d_{i,j})$$

$$sh(d_{i,j}) = \begin{cases} 1 - \left(\dfrac{d_{i,j}}{\sigma}\right)^{\alpha} & \text{if } d_{i,j} < \sigma \\ 0, & \text{if } d_{i,j} \geq \sigma \end{cases}$$

where N is the number of individuals in the population and $d_{i,j}$ is the distance between individual i and individual j. There is also a second parameter, α, which states the magnitude of sharing. For $\alpha > 1$, the effects of sharing are enhanced, while for $\alpha < 1$, sharing gets a little less intimidating.

2.2 Artificial neural networks

Artificial neural networks (ANN) is another field inspired by nature, with a layout that resembles neuron networks in biological brains, albeit highly simplified. There are various ways of connecting the nodes. If the nodes are connected in such a way that information travelling through the network only goes one way, the ANN is called a feed-forward neural network or sometimes a multilayer perceptron (MLP). Normally, a neural network consist of units of one or several nodes, called layers. By connecting several layers, one arrives at deep learning.

ANNs are capable of nonlinear modeling without prior knowledge about the relationships between input and output variables, which is particularly useful when an observed process does not meet the assumptions needed in traditional mathematical methods [6]. Further, the universal approximation theorem states that a MLP can approximate any continuous function, on a compact set of input values [32], meaning that there is no fundamental constraints built into the MLP. This also means that it can be applied in various fields. Other types of ANNs does also fulfil the universal approximation theorem, such as convolutional neural networks (very common in image analysis) [33], but not all ANNs is legitimate universal function approximators. Overall, this makes ANNs capable of solving a wide range of problems and they have been successfully applied in image analysis, pattern recognition [34], sequence recognition (such as speech [35]), social network filtering (such as the news article *Facebook Boosts AI to Block Terrorist Propaganda* [36]) and finance [2, 6].

One drawback with ANNs is that they essentially can be seen as a black box, and the model development is a computationally intensive procedure [37]. Their black box nature makes ANNs hard to use in fields where one has to explain the output of the model [38], such as in medical diagnosis [39] or credit applications in finance [40].

How a MLP is constructed and trained will be explained below.

2.2.1 The structure

Starting with a simple perceptron with one input layer and one output node, figure 6. The output node computes a weighted sum of the input values, add a bias b to the sum and then passes it through an activation function φ, that gives the output $\hat{z}$ of the perceptron. The perceptron can be seen as a mapping $F : \mathbb{R}^k \mapsto \mathbb{R}$,

$$\hat{z} = \varphi(a) = \varphi\left(b + \sum_{i=1}^{k} w_i x_i\right) = \varphi(\boldsymbol{w}^T \boldsymbol{x} + b) = F(\boldsymbol{x}, \boldsymbol{w}, b)$$

By adding one or more layers between the input and output layer, one gets

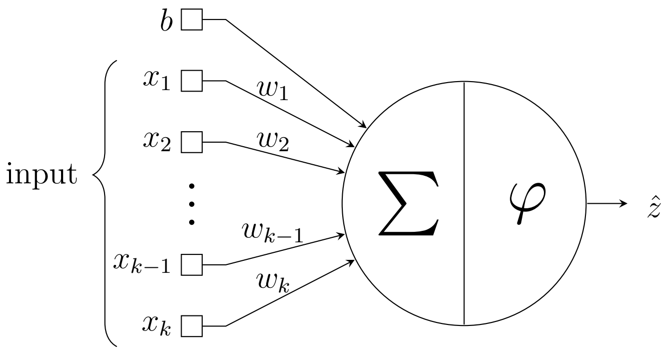


Figure 6: The simple perceptron consisting of a input layer and one output node. The output node receive input values x_1 from the input layer, compute a weighted sum of these and add a bias term b. This sum is then passed through an activation function φ, yielding the output $\hat{z}$ from the perceptron.

a MLP, see figure 7. The intermediate layers are called hidden layers, due to the fact that their output and structure is unknown when using the model, i.e. the MLP act as a black box and the hidden layers are a part of this box.

As mentioned before, looking at figure 7, one sees that information travelling through the network only goes in one direction (from left to right in this example). In the rest of this section, the MLP in figure 7 will act as a base for explaining the principles of a feed-forward neural network. The principle is the same when adding more layers.

Let $w_{j,i}$ be the weight from hidden node j to output node i and $\boldsymbol{w}_i$ be the weights from the hidden layer to output node i,

$$\boldsymbol{W} = \begin{pmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_{i'} \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1i'} \\ w_{21} & w_{22} & & w_{2i'} \\ \vdots & & \ddots & \vdots \\ w_{j'1} & w_{j'2} & \dots & w_{j'i'} \end{pmatrix}$$

be all weights to the output layer. Following, let b_i be the bias from the hidden layer to output node i and

$$\boldsymbol{b} = \begin{pmatrix} b_1 & b_2 & \cdots & b_{i'} \end{pmatrix}^T$$

Then the output $\hat{\boldsymbol{z}} = (\hat{z}_1, ..., \hat{z}_{i'})^T$ is

$$\hat{\boldsymbol{z}} = \varphi_o \circ \boldsymbol{a} = \varphi_o \circ (\boldsymbol{W}^T \boldsymbol{h} + \boldsymbol{b})$$

where φ_o is the activation function for the output layer and $\boldsymbol{h}$ is the output from the hidden layer. Declaring $\tilde{\boldsymbol{W}}$ and $\tilde{\boldsymbol{b}}$ in the same manner as $\boldsymbol{W}$ and $\boldsymbol{b}$, respectively, but from the input layer instead, results in

$$\hat{\boldsymbol{z}} = \varphi_o \circ \big(\boldsymbol{W}^T(\varphi_h \circ \tilde{\boldsymbol{a}}) + \boldsymbol{b}\big) = \varphi_o \circ \Big(\boldsymbol{W}^T\big(\varphi_h \circ (\tilde{\boldsymbol{W}}^T\boldsymbol{x} + \tilde{\boldsymbol{b}})\big) + \boldsymbol{b}\Big),$$

where φ_h is the activation function for the hidden layer.

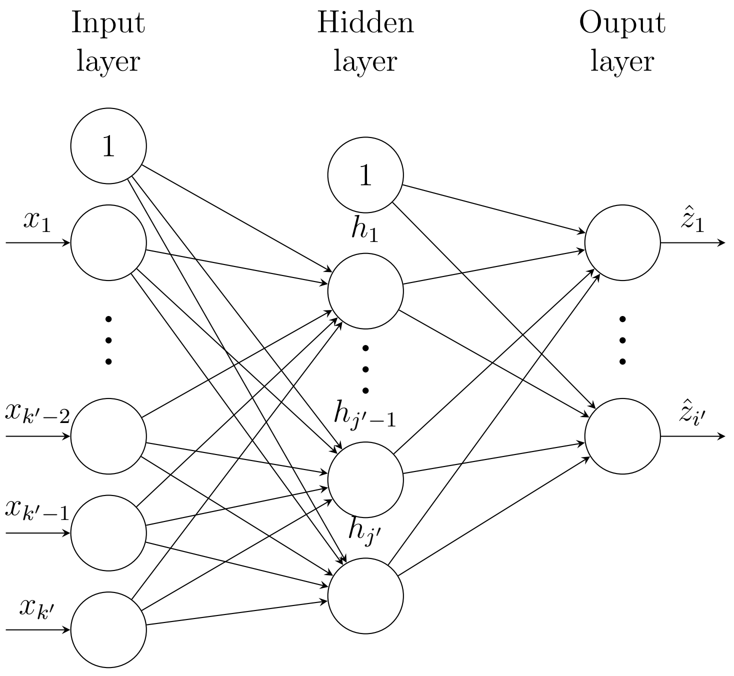


Figure 7: An MLP with three layers. The first layer, the input layer, receive the input $\boldsymbol{x}$ and pass it through to the hidden layer, which then pass it to the output layer. The nodes that contain 1 represents the bias; the input is equal to 1 and each weight from that node is the bias.

Historically speaking, the activation function for a perceptron was the Heaviside step function. Making it suitable for binary classification [41]. Due to it being a step function, it only has two states: 0 or 1 (on or off), which limit its applicability. Nowadays the perceptron and the MLP is used in a broader sense, and can refer to a feed-forward network suitable for both classification and regression. Neither is it necessary for a perceptron to have the Heaviside step function as activation function, other more common alternatives today is the rectified linear or the logistic function [42].

2.2.2 Training

In order for a network to be functional on a given problem, it need to learn how the observed process behaves. Let $\boldsymbol{D} = ((\boldsymbol{x}_1, \boldsymbol{z}_1), (\boldsymbol{x}_2, \boldsymbol{z}_2), \ldots, (\boldsymbol{x}_N, \boldsymbol{z}_N))$ be the set of all observation of a process, where each $\boldsymbol{x}_n$ correspond to experience $\boldsymbol{z}_n$. The target is to optimize the network's output $\hat{\boldsymbol{z}}$ against the observed values $\boldsymbol{z}_n$ for each input $\boldsymbol{x}_n$. To do so, one need a so called error function that measure how well the output of compete for limited resources in a crowded area.the network fit the observed data. The error function is a function of the weights and the biases in the network and is defined as,

$$Er(\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{b}}, \boldsymbol{W}, \boldsymbol{b}) = \sum_{n=1}^{N} Er_n(\tilde{\boldsymbol{W}}, \tilde{\boldsymbol{b}}, \boldsymbol{W}, \boldsymbol{b})$$

where Er_n is the error when the network is fed with the input $\boldsymbol{x}_n$. By adjusting the weights and biases in the network, the error can be minimized.

Using the gradient descent method, the weights and biases are updated according to,

$$\Delta\boldsymbol{W} = -\gamma\nabla_{\boldsymbol{W}}Er, \qquad \Delta\boldsymbol{b} = -\gamma\nabla_{\boldsymbol{b}}Er, \tag{2}$$

$$\Delta\tilde{\boldsymbol{W}} = -\gamma\nabla_{\tilde{\boldsymbol{W}}}Er, \qquad \Delta\tilde{\boldsymbol{b}} = -\gamma\nabla_{\tilde{\boldsymbol{b}}}Er, \tag{3}$$

where γ is the step size while updating the weights and biases, also known as the learning rate.

We define,

$$\begin{aligned}
\boldsymbol{\delta}_n &= \frac{\partial Er}{\partial \hat{\boldsymbol{z}}_n}, \\
\tilde{\boldsymbol{\delta}}_n &= \frac{\partial Er}{\partial \boldsymbol{h}_n} = \frac{\partial Er}{\partial \hat{\boldsymbol{z}}_n}\frac{\partial \hat{\boldsymbol{z}}_n}{\partial \boldsymbol{a}_n}\frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{h}_n} = \boldsymbol{W}^T\big(\boldsymbol{\delta}_n \odot (\varphi_o' \circ \boldsymbol{a}_n)\big).
\end{aligned}$$

where $\odot$ is element-wise (Hadamard) multiplication, $\hat{\boldsymbol{z}}_n$ is the output of the network when the input is $\boldsymbol{x}_n$ and the corresponding for $\boldsymbol{a}_n$ and $\boldsymbol{h}_n$.

The differentials in (2) can thus be expanded as,

$$\nabla_{\boldsymbol{b}} Er = \sum_{n=1}^{N} \frac{\partial Er_n}{\partial \hat{\boldsymbol{z}}_n} \frac{\partial \hat{\boldsymbol{z}}_n}{\partial \boldsymbol{a}_n} \frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{b}} = \sum_{n=1}^{N} \boldsymbol{\delta}_n \odot (\varphi_o' \circ \boldsymbol{a}_n)$$

$$\nabla_{\boldsymbol{W}} Er = \sum_{n=1}^{N} \frac{\partial Er_n}{\partial \hat{\boldsymbol{z}}_n} \frac{\partial \hat{\boldsymbol{z}}_n}{\partial \boldsymbol{a}_n} \frac{\partial \boldsymbol{a}_n}{\partial \boldsymbol{W}} = \sum_{n=1}^{N} \big(\boldsymbol{\delta}_n \odot (\varphi_o' \circ \boldsymbol{a}_n)\big) \boldsymbol{h}_n^T.$$

Further, the differentials in (3) can be expanded as,

$$\nabla_{\tilde{\boldsymbol{b}}} Er = \sum_{n=1}^{N} \frac{\partial Er_n}{\partial \boldsymbol{h}_n} \frac{\partial \boldsymbol{h}_n}{\partial \tilde{\boldsymbol{a}}_n} \frac{\partial \tilde{\boldsymbol{a}}_n}{\partial \tilde{\boldsymbol{b}}} = \sum_{n=1}^{N} \tilde{\boldsymbol{\delta}}_n \odot (\varphi_h' \circ \tilde{\boldsymbol{a}}_n)$$

$$\nabla_{\tilde{\boldsymbol{W}}} Er = \sum_{n=1}^{N} \frac{\partial Er_n}{\partial \boldsymbol{h}_n} \frac{\partial \boldsymbol{h}_n}{\partial \tilde{\boldsymbol{a}}_n} \frac{\partial \tilde{\boldsymbol{a}}_n}{\partial \tilde{\boldsymbol{b}}} = \sum_{n=1}^{N} \tilde{\boldsymbol{\delta}}_n \odot (\varphi_h' \circ \tilde{\boldsymbol{a}}_n) \boldsymbol{x}_n^T.$$

This way of updating the weights and biases is called back-propagation of error. First all the inputs are forwarded through the network, resulting in outputs $\hat{\boldsymbol{z}}_n, \forall\, n \in \{1, \ldots, N\}$. Then each $\boldsymbol{\delta}_n$ and $\tilde{\boldsymbol{\delta}}_n$ is back-propagated through the layers to update the weights and biases.

Let $\boldsymbol{W}^{(\mathrm{itr})}$ be the weight matrix at iteration itr. This gives the weight update,

$$\boldsymbol{W}^{(\mathrm{itr}+1)} = \boldsymbol{W}^{(\mathrm{itr})} - \gamma\big(\boldsymbol{\delta}^{(\mathrm{itr})} \odot (\varphi_o' \circ \boldsymbol{a}^{(\mathrm{itr})})\big)(\boldsymbol{h}^{(\mathrm{itr})})^T$$

and the corresponding for the rest of the weights and biases.

2.3 K-means clustering

This is a widely used heuristic clustering algorithm that partitions n observations into k clusters, where $k \leq n$. Each cluster C_i has a cluster centroid m_i and each observation x_p is assigned to the cluster with the nearest centroid. The objective of k-means algorithm is to minimize the Euclidean distance between the cluster centroid and its members. This are done by repeatedly refine the k cluster centroids and reassign the observations to the new clusters.

Two commonly used initialization method are Forgy and Random Partition [43]. Using the Forgy method, k randomly chosen observations are initialized as cluster centroids. The Random Partition method randomly assign each observation to a cluster, then the centroids are computed as the mean of each member in the cluster.

The algorithm proceeds in the following two steps

- Assignment of each observation x_p to the cluster with the nearest centroid, such that each cluster C_i is a set

  $$C_i = \{x_p : ||x_p - m_i||^2 \leq ||x_p - m_j||^2 \; \forall j, 1 \leq j \leq k\},$$

  where a small adjustment is made to the above formulation: all observations are assigned to exactly one cluster (even if an observation happen to fit into several clusters).

- Calculation of the new cluster centroids by taking the mean of all it's members, i.e.

  $$m_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j, \quad \text{for } i = 1, \ldots, k.$$

These two steps are repeated until the algorithm reach a stopping criterion, such as:

- The centroids have not changed during the last iterations.
- The algorithm has reached it's maximum number of iterations.

3 Method

This section cover the experimental work of this thesis. Starting by dividing the data into two parts, one for training and one for testing. The training data is then divided into smaller parts which is then clustered using k-means clustering method. The purpose of the clustering step is to prepare training data for the ANNs, since supervised learning is used when training the ANNs. Next, several binary classification networks are being trained on the clustered data using a GA, resulting in several network functions. Lastly, a GP is being trained to predict the stock price of the subsequent day. This GP has the opportunity to utilize the network functions for improving its predictions. All the steps are depicted in figure 8.

3.1 Data

The data being used is the historical Stock Quote Prices for Bureau Veritas SA (BVI.PA), Paris, France, during the period 2007-04-10 – 2017-11-10, This data was arbitrarily chosen as a basis for this work. The whole time series is normalized, meaning that it is mapped to the interval $[0, 1]$.

First, the data are divided into two parts, see figure 9. The first part is used while building the model, this will be called the training data and is denoted $\boldsymbol{Y}_{\mathrm{tr}} = (y_1, y_2, \ldots, y_{n_{\mathrm{tr}}})$. The second part, is used to evaluate how the model handles new data, i.e. the second part is only used when the model is finished and is called the test data $\boldsymbol{Y}_{\mathrm{te}}$. The training data consist of $n_{\mathrm{tr}} = 2269$ observations, which is 85% of the given data, and the test data consist of $n_{\mathrm{te}} = 400$ observations. The training data is later divided into a "net-training" set and a validation set during buildup of the MLP's.

3.2 Clustering

The training data $\boldsymbol{Y}_{\mathrm{tr}}$ is being prepared for clustering by letting a window, that is $\omega = 5$ wide, slide over the whole time series. Resulting in several subsequences, here called segments. A segment $\boldsymbol{s}_{\omega,t}$ is defined as

$$\boldsymbol{s}_{\omega,t} = \begin{pmatrix} y_{t-\omega+1} & \cdots & y_{t-1} & y_t \end{pmatrix}^T . \tag{4}$$

The idea behind segmentation is that in time t one can look back at segment $\boldsymbol{s}_{\omega,t}$ and see what has happened before. At that time t, the value y_t will be

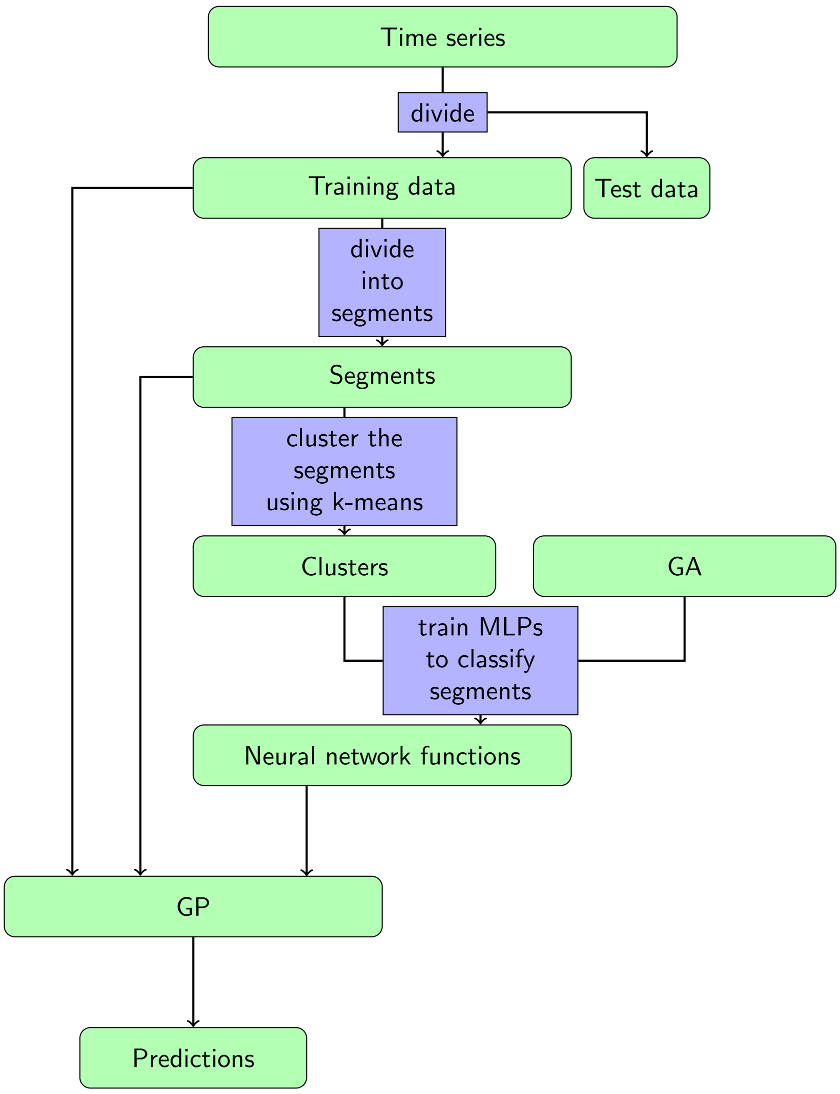


Figure 8: A flowchart for the method. The time series is divided into two parts; training data and test data. The training data is further divided into smaller subsequences, here called segments. The segments are clustered using k-means. The clusters are classified by several binary classifications networks, whom are trained using a GA. Resulting in several neural network functions. These functions, along with the training data and the segments, are used in a GP that is being trained to predict the stock price for the next day. The test data is only used when the model is finished and measure the out-of-sample performance.

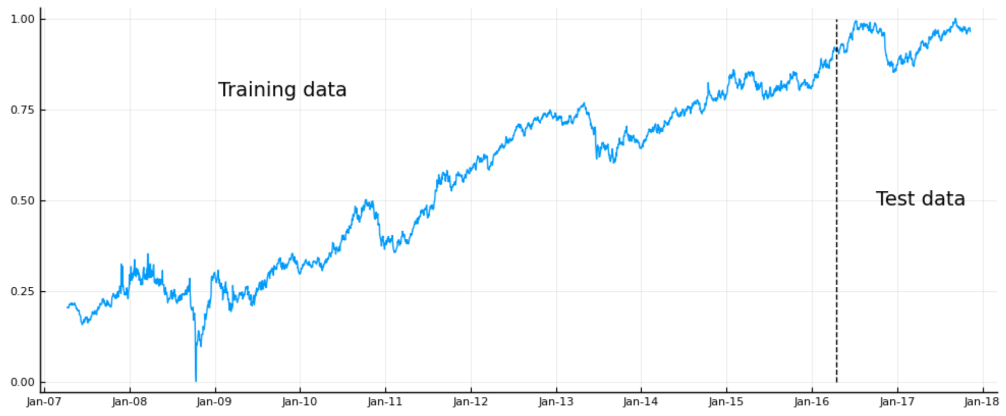


Figure 9: The time series is divided into two main parts. The first part is used for training and model designing, i.e. training data. The second part is used for evaluating the performance of the model on out-of-sample data, this part is called the test data.

the base and because of that each segment is subtracted by its last value,

$$\hat{\boldsymbol{s}}_{\omega,t} = \boldsymbol{s}_{\omega,t} - y_t \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

The log-increment in each segment is evaluated by first adding a constant ϵ to each element, such that division by 0 and the logarithm of 0 are avoided,

$$\tilde{\boldsymbol{s}}_{\omega,t} = \left(\ln\left(\frac{y_{t-\omega+2}-y_t+\epsilon}{y_{t-\omega+1}-y_t+\epsilon}\right) \quad \cdots \quad \ln\left(\frac{y_{t-1}-y_t+\epsilon}{y_{t-2}-y_t+\epsilon}\right) \quad \ln\left(\frac{y_t-y_t+\epsilon}{y_{t-1}-y_t+\epsilon}\right)\right)^T,$$

for $t = \omega, \ldots, n_{\text{tr}}$.

Following by collecting each $\tilde{\boldsymbol{s}}_{\omega_t}$ in a matrix

$$\tilde{\boldsymbol{S}}_\omega = \begin{pmatrix} \tilde{\boldsymbol{s}}_{\omega,\omega} & \tilde{\boldsymbol{s}}_{\omega,\omega+1} & \cdots & \tilde{\boldsymbol{s}}_{\omega,n_{\text{tr}}} \end{pmatrix}.$$

This matrix is then used to cluster each segment $\boldsymbol{s}_{\omega,t}$ with k-means clustering, where the number of clusters $k = 25$ and Forgy is used in the initialization step.

3.3 Multilayer perceptrons

Several MLP's are employed to classify the clustered data. For each clustering two kinds of MLP's are being trained. Firstly, one multi-class classification network is trained to classify each segment. A suitable error function for this task, is the cross entropy

$$Er(\boldsymbol{b}_1, \boldsymbol{W}_1, \dots, \boldsymbol{b}_L, \boldsymbol{W}_L) = -\sum_{n=1}^{n_{\text{tr}}} \sum_{i=1}^{k} z_{n,i} \cdot \ln(\hat{z}_{n,i}),$$

where $\boldsymbol{b}_l$ and $\boldsymbol{W}_l$ is the bias and weights for the l'th layer, for $l = 1, \dots, L$. The number of classes is the same as the number of clusters $k = 25$, n_{tr} is the number of observations, $\boldsymbol{z}_n = (z_{n,1}, z_{n,2}, \dots, z_{n,k})$ is the n'th observation, and $\hat{\boldsymbol{z}}_n$ is the estimated value of $\boldsymbol{z}_n$. Furthermore,

$$z_{n,i} = \begin{cases} 1, & \text{if } \boldsymbol{z}_n \in \text{ class } u \text{ and } i = u \\ 0, & \text{if } \boldsymbol{z}_n \in \text{ class } u \text{ and } i \neq u \end{cases}.$$

The activation function for the input and hidden layers is the logistic function. For the output layer the activation function is the softmax function,

$$\hat{z}_{n,i} = \frac{e^{a_{n,i}}}{\sum_j e^{a_{n,j}}},$$

where $a_{n,i}$ is the net input to output node i for observation n.

Secondly, given $k = 25$ clusters, $k = 25$ binary classification networks are trained to tell if a segment belongs to "their" class or not. The error function is once again the cross entropy function and here, all the nodes has the logistic function as activation function.

The chosen optimization algorithm is Adam (short for adaptive moment estimation), which was introduced by Kingma and Ba [44]. Adam combines the advantages of two popular optimization methods, namely RMSProp and AdaGrad. The implementation is straightforward and the hyper-parameters typically needs little tuning, which is very useful when having many networks to tune. Overall, it is considered to be a robust algorithm with a wide interpretability [44, 45].

The Adam algorithm store a moving average of the past gradients and also of the square of the past gradients. For each layer l these are defined as,

$$\boldsymbol{m}_{\boldsymbol{b}_l}^{(\text{itr}+1)} = \beta_1 \boldsymbol{m}_{\boldsymbol{b}_l}^{(\text{itr})} + (1 - \beta_1) \nabla_{\boldsymbol{b}_l} Er^{(\text{itr})} \tag{5}$$

$$\boldsymbol{m}_{\boldsymbol{W}_l}^{(\text{itr}+1)} = \beta_1 \boldsymbol{m}_{\boldsymbol{W}_l}^{(\text{itr})} + (1 - \beta_1) \nabla_{\boldsymbol{W}_l} Er^{(\text{itr})} \tag{6}$$

$$\boldsymbol{v}_{\boldsymbol{b}_l}^{(\text{itr}+1)} = \beta_2 \boldsymbol{v}_{\boldsymbol{b}_l}^{(\text{itr})} + (1 - \beta_2) \big(\nabla_{\boldsymbol{W}_l} Er^{(\text{itr})}\big) \odot \big(\nabla_{\boldsymbol{W}_l} Er^{(\text{itr})}\big) \tag{7}$$

$$\boldsymbol{v}_{\boldsymbol{W}_l}^{(\text{itr}+1)} = \beta_2 \boldsymbol{v}_{\boldsymbol{W}_l}^{(\text{itr})} + (1 - \beta_2) \big(\nabla_{\boldsymbol{W}_l} Er^{(\text{itr})}\big) \odot \big(\nabla_{\boldsymbol{W}_l} Er^{(\text{itr})}\big) \tag{8}$$

where $\boldsymbol{m}^{(\text{itr})}$ and $\boldsymbol{v}^{(\text{itr})}$ is the moving average at iteration itr and $Er^{(\text{itr})}$ is the error at iteration itr. Before updating the weights and biases, a correction is made to (5) – (8),

$$\hat{\boldsymbol{m}}_{\boldsymbol{b}_l} = \frac{1}{1-\beta_1}\boldsymbol{m}_{\boldsymbol{b}_l}^{(\text{itr}+1)} \qquad \hat{\boldsymbol{m}}_{\boldsymbol{W}_l} = \frac{1}{1-\beta_1}\boldsymbol{m}_{\boldsymbol{W}_l}^{(\text{itr}+1)}$$

$$\hat{\boldsymbol{v}}_{\boldsymbol{b}_l} = \frac{1}{1-\beta_2}\boldsymbol{v}_{\boldsymbol{b}_l}^{(\text{itr}+1)} \qquad \hat{\boldsymbol{v}}_{\boldsymbol{W}_l} = \frac{1}{1-\beta_2}\boldsymbol{v}_{\boldsymbol{W}_l}^{(\text{itr}+1)}.$$

Finally, the weight and bias update, for each layer l, is given by,

$$\boldsymbol{b}_l^{(\text{itr}+1)} = \boldsymbol{b}_l^{(\text{itr})} - \gamma\,\hat{\boldsymbol{m}}_{\boldsymbol{b}_l} \oslash (\hat{\boldsymbol{v}}_{\boldsymbol{b}_l}^{\circ 1/2} + \epsilon\mathbf{1})$$

$$\boldsymbol{W}_l^{(\text{itr}+1)} = \boldsymbol{W}_l^{(\text{itr})} - \gamma\,\hat{\boldsymbol{m}}_{\boldsymbol{W}_l} \oslash (\hat{\boldsymbol{v}}_{\boldsymbol{W}_l}^{\circ 1/2} + \epsilon\mathbf{1})$$

where $\epsilon > 0$ is used to avoid numerical problems if an element in $\hat{\boldsymbol{v}}_{\boldsymbol{b}_l}$ or $\hat{\boldsymbol{v}}_{\boldsymbol{W}_l}$ for any $l \in \{1, \ldots, L\}$, becomes to small.

3.4 Genetic algorithm

A GA has been implemented to train the binary classification networks. The population size is set to 16 individuals, where the 8 best are allowed to reproduce. This is a rather small population, but through testing, this showed to be sufficient for this task.

The selection and replacement is carried out by selecting the best individuals and replace the worst. For reproduction, both single-point crossover, applied on the whole genome rather than chromosomes, and blending crossover is used. Each crossover technique is used with 50% probability. The mutation method is point mutation and the mutation rate is set to 5%. This GA uses elitism, i.e. the top two individuals will not be mutated.

An individual has three chromosomes. The first chromosome has two genes, each representing the number of nodes in the first and second hidden layer. If one of the genes is equal to 0, this individual represents a network with one hidden layer. The number of hidden layers is restricted to 2 and the number of nodes in each layer belong to $\{0, 1, \ldots, 8\}$, because we want small networks (and it is sufficient for this classification task). The second chromosome holds the learning rate $\gamma \in [0.0002, 0.008]$. The third chromosome holds the number of iterations during training of the network and it's value belong to $\{100, \ldots, 300\}$. For an example, see figure 10.

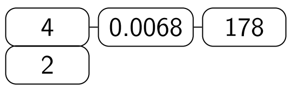


Figure 10: An example of an individual's genome in the genetic algorithm described in section 3.4. The first chromosome has two genes, each representing the number of nodes in the first and second hidden layer of the network, here the first hidden layer contains 4 nodes and the second hidden layer contain 2 nodes. The second chromosome holds the learning rate, here it is 0.0068. The third chromosome holds the number of iterations during training, here it is 178 iterations.

3.4.1 Fitness function

Here, an attempt is made to quantify what one would look for while tuning a binary classification network. Trivial is to aim for high sensitivity and specificity, both on training and validation data. Thus, the target will be to maximize the fitness.

For each class l, the data is skewed; a major part of the samples do not belong to class l. Hence, the sensitivity will get a twice as high value than the specificity. This results in the final fitness for individual i is given by,

$$f_i = 2 \cdot (\text{sens}_{\text{train}} + \text{sens}_{\text{val}}) + \text{spec}_{\text{train}} + \text{spec}_{\text{val}}.$$

3.5 Genetic programming

A stack-based genetic programming system is being implemented, as a stack based model is compatible with oxides polychaos framework. The GP has a population consisting of individuals, that evolves over generations. Each individual represents a computer program that gives a prediction of the stock index y for the next day. Let the computer program, or function, that individual i represents, be denoted as

$$\psi_i(t, t-1, \ldots, t-4) = \hat{y}_{t+1,i},$$

where $\hat{y}_{t+1,i}$ is the estimated value of y_{t+1} made by individual i.

Three slightly different versions of the GP are being made. The three versions are:

- Standard GP; no access to ANNs and randomly generated species.

- Speciation using a MLP; no access to ANNs and the species are generated using a MLP.
- ANNs in the combined set; access to ANNs and randomly generated species.

The following subsections give a more detailed explanation of the GP.

3.5.1 The combined set C

The combined set $C = F \cup T$ is

$$\begin{aligned} F &= \{+, -, \times, \text{pdiv}, \text{psin}, \exp, \text{pln}, \text{AND}, \text{OR}, \text{NOT}\} \\ T &= \{\Re, y_t, y_{t-1}, y_{t-4}, \pi\} \cup \{\text{network functions}\} \end{aligned}$$

where $\Re$ represents Koza's ephemeral random constant [21, chap. 10]. Whenever a gene is set to the ephemeral random constant during the initialization step, the gene is set to a three-digit number drawn from $\mathcal{U}(-0.5, 0.5)$. The granularity is chosen as to make the chromosomes more comprehensible. Additionally, this genes value is stored in an array E. When the initialization step is completed, E will contain all generated constants during this step and will act as a pool of constants during the rest of the algorithm. If a mutation occurs and the mutation produce a ephemeral random constant, the value will be drawn uniformly from the array E.

Further explanation of the function and terminal set will follow in the two proceeding subsections.

3.5.1.1 Closure

All functions in F are maps $f_1 : \mathbb{R} \mapsto \mathbb{R}$ or $f_2 : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$, i.e. they only take scalar input values. While each network function is a map $g : \mathbb{R}^m \mapsto \mathbb{R}$ where $m > 1$ is the segment width, i.e. only taking vector valued input. Hence, adding the network functions to the function set will cause error. Instead, the network functions are treated as variables, such that the property of closure is fulfilled.

Let net1 be the binary classification network that classifies cluster 1. Then at time t the net1-variable is

$$\text{net1}_t = \text{net1}(\boldsymbol{s}_{m,t})$$

where $\boldsymbol{s}_{m,t}$ is the segment defined in equation (4). The corresponding apply to the rest of the network functions.

3.5.1.2 Sufficiency

It should be re-emphasized that the important aim for this thesis is to *improve* the prediction of the stock index by leveraging neural information processing. It is not believed that this report will find the so long sought solution to: what will the stock price be tomorrow? Technically, the property of sufficiency is not satisfied, but the aim is to come as close as possible (within the limits of this thesis). As mentioned in the introduction, there are previous works in time series prediction where ANNs are being used [2, 7, 13]. Hence, it is believed that introducing ANNs in C will contribute to the sufficiency of the combined set.

3.5.2 Population initialization

The population is randomly generated and represents a sampling of the space of possible solutions. For each individual i, a chromosome length ℓ_i is drawn from $\mathcal{U}\{1, \ell_{\text{max}}\}$ where ℓ_{max} is the maximum chromosome length during the initialization step. Let $g_{j,i}$ be the j'th gene in the chromosome of individual i. Then each gene $g_{j,i}$ is drawn from either the function set F or the terminal set T, with equally high probability. The reason to draw from either F or T is when the ANNs are added to the combined set, the portion of terminals in C will change; resulting in a different gene frequency of terminal and functions in the population when the combined set contain ANNs and not, which might affect the gene interplay. It is believed that the comparison between the different version of the GP will be more fair when the genes are drawn this way.

There are no restrictions regarding the composition of the chromosomes. There may arise some that do not encode any expression. For example the chromosome

```
    {sin, +, pln, -}
```

will not encode any expression, but is still viable. The human genome, for instance, consist of 98% noncoding sequences [14], indicating that it does not have to be a problem.

The number of individuals in the population can have great impact in the final solution. Starting with only a few individuals will probably give a narrow covering of the search space, and may result in suboptimal solutions because the algorithm got stock in a local optimum. While having many individuals will more likely embrace a bigger area of the search space and increase the probability of finding globally optimal solutions. At the same time, many individuals also mean greater computational cost and to some extent a slower converging algorithm. To compromise these aspects, bootstrapping is being used. In this context, meaning that the population is initialized with a great number of individuals, and after some generations the population is reduced to a smaller number of individuals. This results in a broad covering of the search space, but in an inexpensive way.

3.5.3 Genetic operations

The GP is a steady-state GP. Selection and replacement is attained through tournament selection.

Crossover is performed through single-point crossover. Since there is no restriction regarding the chromosome length, a small adjustment is made to the single-point crossover described in figure 1. One crossover point is randomly generated at the first parent's chromosome and another crossover point is randomly generated at the second parent's chromosome. Enabling the chromosomes to grow beyond the maximum chromosome length $\ell_{\max}$, set during the initialization step.

To soften the growth, the break point is drawn from a normal distribution. Let ℓ_{parent1} be the chromosome length of the first parent. Then the break point bp is drawn from

$$bp \sim \mathcal{N}(\mu, \sigma), \quad \mu = \frac{\ell_{\text{parent1}} + 1}{2}, \quad \sigma = \frac{\ell_{\text{parent1}} - 1}{6}.$$

Mutation is attained through single-point mutation, in the same way that were explained in section 2.1.2.3.

3.5.4 Fitness function

First, the root mean square error (RMSE) between two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ is defined as,

$$g(\boldsymbol{u}, \boldsymbol{v}) = \frac{1}{N} \sqrt{\sum_{j=1}^{N} (v_j - u_j)^2}$$

where N is the length of the input vectors, u_j is the j'th element in vector $\boldsymbol{u}$ and the corresponding for $\boldsymbol{v}$.

The fitness is calculated in two steps and the target is to minimize it. The fitness f_i for individual i is the RMSE of the observed time series $\boldsymbol{Y}_{\text{tr}}$ and the corresponding estimated time series $\hat{\boldsymbol{y}}_i$ made by individual i,

$$f_i = f(\hat{\boldsymbol{y}}_i) = g(\boldsymbol{Y}_{\text{tr}}, \hat{\boldsymbol{y}}_i).$$

Secondly, all individuals' fitness value are being shared, as explained in section 2.1.4.1. The niche count is hence given by,

$$m_i = m(\hat{\boldsymbol{y}}_i) = \sum_{j=1}^{N} sh(d_{i,j}), \text{ where } d_{i,j} = g(\hat{\boldsymbol{y}}_i, \hat{\boldsymbol{y}}_j)$$

$$sh(d_{i,j}) = \begin{cases} 1 - \left(\dfrac{d_{i,j}}{\sigma}\right)^{\alpha} & \text{if } d_{i,j} < \sigma \\ 0, & \text{if } d_{i,j} \geq \sigma \end{cases}, \text{ where } \sigma = 0.004, \ \alpha = 1$$

where N is the number of individuals in the population. The distance $d_{i,j}$ between individual i and individual j is measured by calculating the RMSE between the estimated time series made by these two individuals.

The final fitness value f_i^* of individual i is given by

$$f_i^* = f^*(\hat{\boldsymbol{y}}_i) = f_i \cdot m_i. \tag{9}$$

3.5.4.1 Search space

Let us reconnect to the search space defined in equation (1). At the initialization step, the search space will be

$$A = \{\boldsymbol{x} = (x_1, x_2, \ldots, x_{\ell_{\max}}); \ x_j \in C, \ j = 1, \ldots, \ell_{\max}\}.$$

As the population evolve, the maximum chromosome length will grow. Let $\ell_{\max}^{(\text{itr})}$ be the maximum chromosome length in the population at generation itr; resulting in a search space which has a dynamic number of dimensions. This implies that, in generation itr, the search space is

$$A = \{\boldsymbol{x} = (x_1, x_2, \ldots, x_{\ell_{\max}^{(\text{itr})}}); \ x_j \in C, \ j = 1, \ldots, \ell_{\max}^{(\text{itr})}\}.$$

3.5.4.2 The optimization problem

Denote the candidate solution represented by individual i, as $ind_i \in A$. Then the estimated time series made by individual i is given by,

$$\hat{\boldsymbol{y}}_i = \hat{\boldsymbol{y}}(ind_i) \quad \Leftrightarrow \quad \hat{\boldsymbol{y}} : A \mapsto \mathbb{R}^{n_{\text{tr}}}.$$

Further, the final fitness function, given by equation (9), is a map $f^* : \mathbb{R}^{n_{\text{tr}}} \mapsto \mathbb{R}^+$. Resulting in the composite function $H : A \mapsto \mathbb{R}^+$ of f^* and $\hat{\boldsymbol{y}}$, whose rule is

$$H(ind_i) = (f^* \circ \hat{\boldsymbol{y}})(ind_i) = f^*\big(\hat{\boldsymbol{y}}(ind_i)\big).$$

The optimization problem can thus be written as:

Find the minimum of $(f^* \circ \hat{\boldsymbol{y}})(ind_i)$ *subject to* $ind_i \in A$.

3.5.5 Speciation

In nature, when a species is divided and isolated, different linage may occur [46]. The problem at hand is believed to be rather complex, hence it is found desirable to let the population fork and, hopefully, explore different areas of the search space. To mimic speciation, the whole population is divided into species and each species will not integrate with any other species.

Each species is allocated a unique part of the training data $\boldsymbol{Y}_{\text{tr}}$, to simulate different environments. Let $\boldsymbol{y}^{(\kappa)} \subset \boldsymbol{Y}_{\text{tr}}$ denote the data points assigned to species κ. Then

$$\boldsymbol{Y}_{\text{tr}} = \bigcup_{\kappa=1}^{\eta} \boldsymbol{y}^{(\kappa)},$$

where η is the number of species and

$$\boldsymbol{y}^{(u)} \cap \boldsymbol{y}^{(v)} = \emptyset, \quad \text{where } u \neq v, \quad \forall\, u, v \in \{1, 2, \ldots, \eta\}.$$

The allocation is in some case randomly generated, in other cases the allocation is decided by a multi-class classification network described in section 3.3.

The speciation results in several solutions working together. For the training data, the ith estimate of y_{t+1} is given by

$$\hat{y}_{t+1,i} = \begin{cases} \psi_{i^{(1)}}(t, t-1, \ldots, t-4) & \text{if } \ y_{t+1} \in \boldsymbol{y}^{(1)} \\ \psi_{i^{(2)}}(t, t-1, \ldots, t-4) & \text{if } \ y_{t+1} \in \boldsymbol{y}^{(2)} \\ \quad \vdots & \\ \psi_{i^{(\eta)}}(t, t-1, \ldots, t-4) & \text{if } \ y_{t+1} \in \boldsymbol{y}^{(\eta)} \end{cases} \tag{10}$$

where $\psi_{i^{(\kappa)}}$ is the computer program represented by individual $i^{(\kappa)}$ that belong to species κ. Also, let i denote the rank of individual $i^{(\kappa)}$ within species κ, i.e. the fitness value $f^*_{i^{(\kappa)}}$, given by equation (9), of individual $i^{(\kappa)}$ is

$$f^*_{j^{(\kappa)}} \leq f^*_{i^{(\kappa)}} \leq f^*_{k^{(\kappa)}}, \quad \text{where } j^{(\kappa)} < i^{(\kappa)} < k^{(\kappa)}, \quad \forall\, i^{(\kappa)}, j^{(\kappa)}, k^{(\kappa)} \in \{1, 2, \ldots, |S_\kappa|\},$$

where $|S_\kappa|$ is the number of individuals in species κ.

For out-of-sample data, such as the test data, the rule in equation (10) does not apply (by definition, a out-of-sample point $y_t \notin \boldsymbol{y}^{(\kappa)}$, $\forall \kappa \in \{1, 2, \ldots, \eta\}$).

For **randomly generated species**, the ith estimate of y_{t+1} is given by

$$\hat{y}_{t+1,i} = \psi_{i^{(\kappa)}}(t, t-1, \ldots, t-4), \quad \kappa \sim \mathcal{U}\{1, \eta\}. \tag{11}$$

For **species generated by a MLP**, the ith estimate of y_{t+1} is given by

$$\hat{y}_{t+1,i} = \psi_{i^{(\kappa)}}(t, t-1, \ldots, t-4), \quad \kappa = \mathrm{MLP}(\boldsymbol{s}_{5,t}), \tag{12}$$

where the MLP-function outputs the class belonging (and hence species belonging as well) of the segment $\boldsymbol{s}_{5,t}$, defined in equation (4).

4 Results

In this section, the results of this masters thesis are presented. The standard GP will be presented such that the base behaviour will be set. Next, to leverage neural information processing, the data where partitioned and then a GA trained 25 binary classification networks. The clusters and the performance of the networks will be presented. Having the pre-trained networks, the GP where extended in two ways, by using a MLP for speciation and by adding the ANNs to the combined set C. The result of this will be shown, first by examine the performance of the model in terms of accuracy against the test data. Then the gene frequencies and size of evolved programs are being presented.

In short, the results will be presented in the following order:

- Standard GP
- Cluster and multi-class classification network
- GA and binary classification networks
- The variants of the GP

A prediction obtained from equation (11) is uniformly drawn from the pool of predictions made by the ith best individuals in each species, resulting in a non-deterministic prediction. While prediction´s obtained from equation (12) is deterministic, i.e. always the same. In order to compare these results, two approaches has been made.

Single solutions are obtained from equation (11) and equation (12), since each point is estimated by one individual. Also, let $\hat{\boldsymbol{y}}_i$ and $\hat{\boldsymbol{y}}_{\text{te},i}$ be the ith best single estimate of $\boldsymbol{Y}_{\text{tr}}$ and $\boldsymbol{Y}_{\text{te}}$, respectively.

Weighted solutions are obtained by a weighted sum of the predictions made by the ith best individuals in each species. Consider the number of data points ρ_κ assigned to species κ; let this number act as a weight for the prediction made by individual $i^{(\kappa)}$. Each weight is normalized such that the weighted solution is given by,

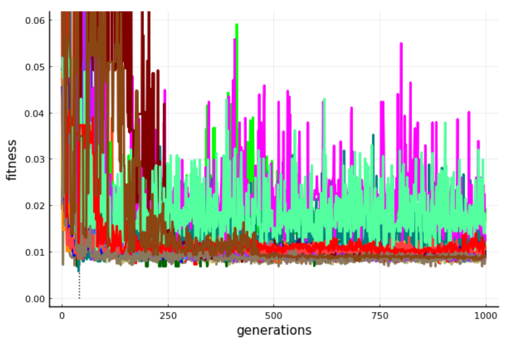
$$\tilde{y}_{t+1,i} = \frac{1}{\sum_{\kappa=1}^{\eta} \rho_\kappa} \sum_{\kappa=1}^{\eta} \rho_\kappa \psi_{i^{(\kappa)}}(t, t-1, \ldots, t-4).$$

The standard GP is presented in figure 11 and figure 12. The convergence is stable, see figure 11, but it should be mentioned that the noise can vary a little depending on the value of σ and α in the fitness sharing function. The three best single solutions are plotted against the training and test data, respectively in figure 12. Overall, the GP has adapted to follow the target curve.

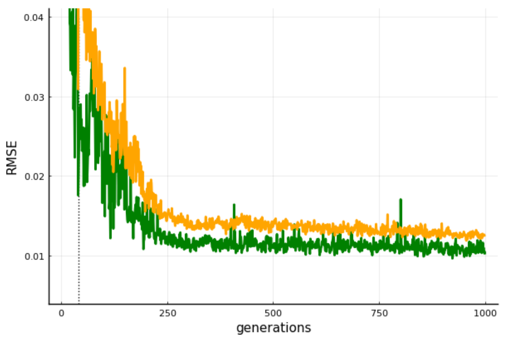
Next, let us look at the partitioning of the data. Figure 13a show in green, the cluster assignments along the training data. It is clear that some clusters have more members than others, representing patterns that appear with varying frequencies. Cluster 5, 14 and 23 seems to be very local, only appearing once in the training data and could maybe be considered as outliers. The rest of the clusters are spread out along the curve and when the curve get an unusual look, some of the sparse clusters seems to get “activated”. For instance, the dip just before $t = 1000$ and the dip after $t = 1500$ has a higher density of some low-density cluster members compared to the increase after roughly $t = 1100$ lasting until $t = 1500$.

The multi-class classification network is mostly inlined with the high-density clusters, but it has some problem with the very low-density clusters, such as cluster 5, 14, 23 and 25 which it fails to recognise, compare green and red crosses in figure 13a. These classes does not appear in the test data either, see figure 13b.

The binary classification network were trained by a GA and the result is depicted in figure 14. The blue and yellow/orange bars represent the sensitivity and specificity, respectively, of the top individuals. At the end of each run, the GA saved the top three individuals, such that the final binary classifier for each class (marked with a black square) where chosen by hand from a pool of three alternatives.



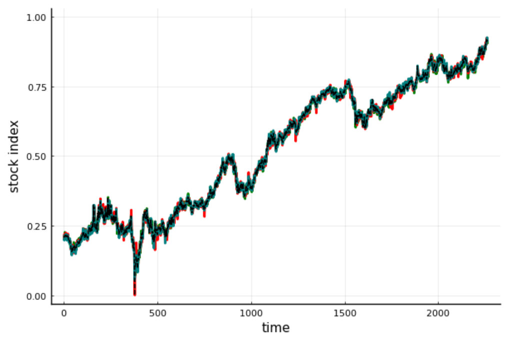
(a) The fitness values $f^*_{1(\kappa)}$



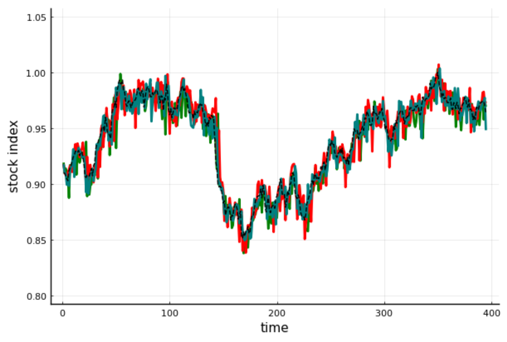
(b) The top (green) and top mean (yellow) RMSE-value for the whole population.

Figure 11: The convergence of the standard GP during one run. The fitness values $f^*_{1(\kappa)}$, for all $\kappa \in \{1, 2, \ldots, \eta\}$ are shown in plot (a). Let $\hat{\boldsymbol{y}}_i$ be the ith best estimation made on the training data. The green line in plot (b) represents the root mean square error, defined in section 3.5.4, of $\hat{\boldsymbol{y}}_1$ against the training data $\boldsymbol{Y}_{\text{tr}}$, i.e. $g(\boldsymbol{Y}_{\text{tr}}, \hat{\boldsymbol{y}}_1)$. The yellow line in plot (b) represents the RMSE mean of the 15 best estimations, i.e. $\left(\sum_{i=1}^{15} g(\boldsymbol{Y}_{\text{tr}}, \hat{\boldsymbol{y}}_i)\right)/15$.

The standard GP converge as it should. Some species appear to have a bit more noisy convergence than other and tuning the parameters α and σ of the fitness sharing function, could change the noisiness a bit.

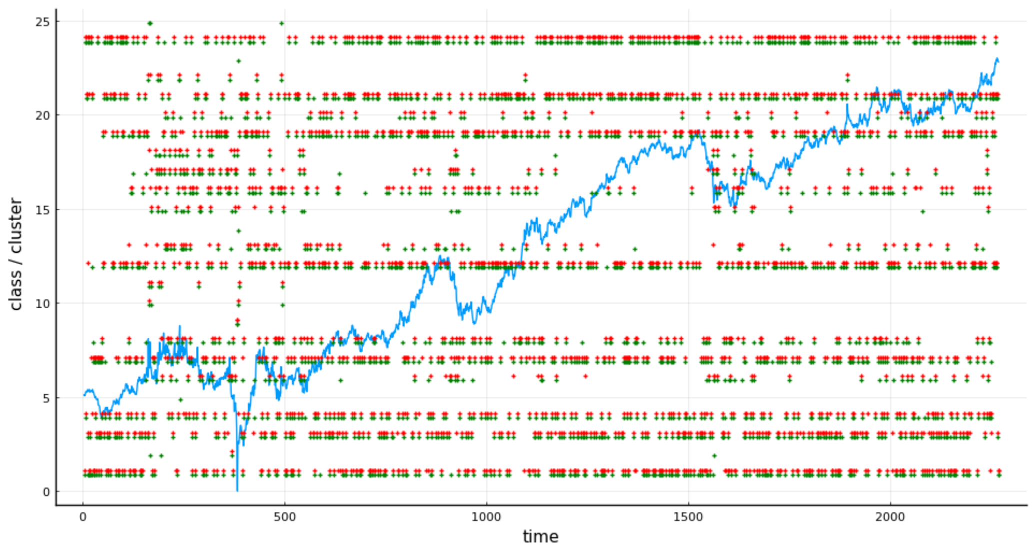


(a) Training data. $g(\boldsymbol{Y}_{\text{tr}}, \hat{\boldsymbol{y}}_1) = 0.010$
$g(\boldsymbol{Y}_{\text{tr}}, \hat{\boldsymbol{y}}_2) = 0.011$
$g(\boldsymbol{Y}_{\text{tr}}, \hat{\boldsymbol{y}}_3) = 0.011$

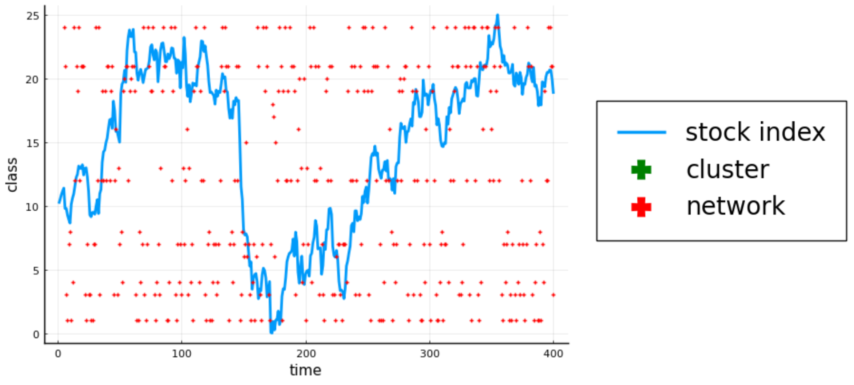


(b) Test data. $g(\boldsymbol{Y}_{\text{te}}, \hat{\boldsymbol{y}}_{\text{te},1}) = 0.009$
$g(\boldsymbol{Y}_{\text{te}}, \hat{\boldsymbol{y}}_{\text{te},2}) = 0.011$
$g(\boldsymbol{Y}_{\text{te}}, \hat{\boldsymbol{y}}_{\text{te},3}) = 0.011$

Figure 12: The precision of the standard GP from one run. Plot (a) and plot (b) show the three best single solutions along the target curve (black dashed line) for the training data and test data, respectively. The performance is equal on both data sets for all three top single solutions. Overall, the results are adequate and it is seems that the standard GP has been fitted to the target data.

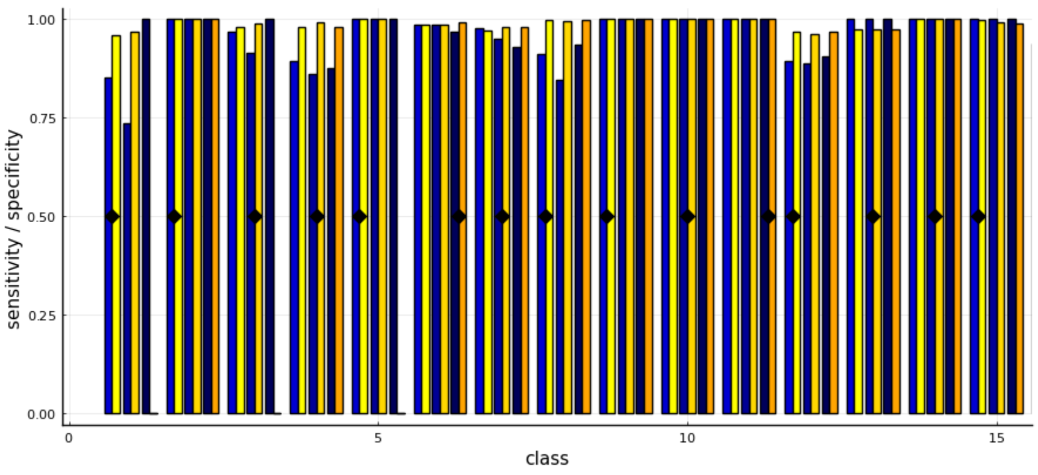


(a) Cluster belonging and network classification on the training data.

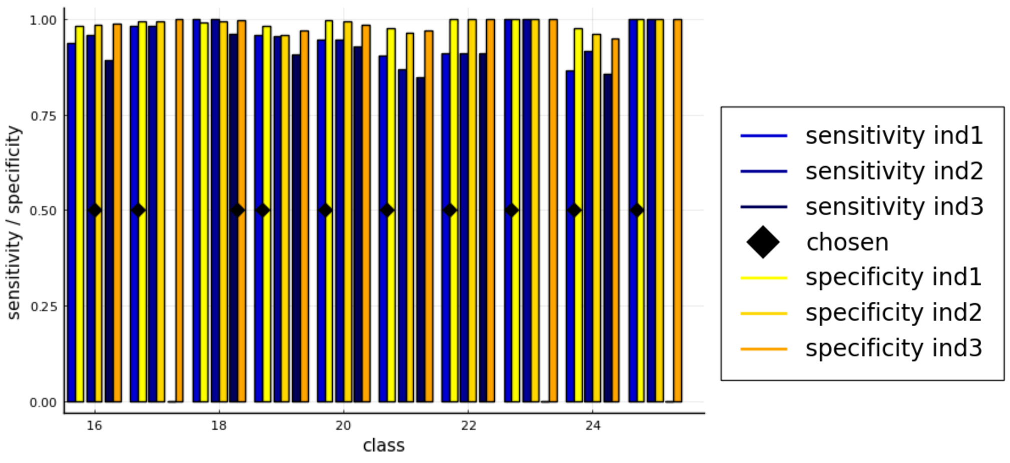


(b) Network classification on the test data.

Figure 13: Cluster belonging and network classification. To visualize the performance of the multi-class classification network, the cluster belonging of each segment of the training data (green crosses), along with the class belonging of each segment that is assigned by the multi-class classification network (red crosses) has been plotted in plot (a). Plot (b) display the class belonging of each segment in the test data, that is assigned by the multi-class classification network. The crosses are placed at the end point of the segment they are representing. Overall, the multi-class classification network does a good job, even though it misses some assignments of the low-density clusters (mainly cluster 2, 5, 9, 14, 23 and 25).



(a) Class 1–15.



(b) Class 16–25.

Figure 14: The three best individuals that where trained by the GA for each binary classifier, where "ind1" is the best individual, "ind2" second best, and so on. The blue bars represent the sensitivity of the network and the yellow bars represent the specificity of the network. The black squares mark which individual that were chosen to proceed to the GP.

Let us now compare the different versions of the GP. In all figures that follow, containing three subplots, are composed as showed in figure 15. Figure 15a depict the standard GP, figure 15b depict the GP having species generated by a MLP and figure 15c depict the GP having ANNs in the combined set.

Figure 16 and figure 17 show the precision regarding the single solutions and the weighted solutions, respectively. Figure 16 show that the median is lower when ANNs are added to the combined set, but not from speciation using a MLP. Figure 17 tell that the weighted solutions does not gain accuracy by leveraging neural information processing.

Next, figure 18 and figure 19 presents the gene frequency. It appears that the GP find the neural networks useful when they are added to the combined set, based on the decrease of use of lag-signals and constants in both figure 18c and figure 19c. Speciation using a MLP does not change the gene interplay, see figure 18b and figure 19b.

Lastly the chromosome length is being presented in figure 20 and figure 21. Here, it is seen that adding ANNs to the combined set reduces bloat.

Standard GP

Speciation: Random

$C = \{+, -, \times, \mathsf{pdiv}, \mathsf{psin}, \exp, \mathsf{pln}, \mathsf{AND}, \mathsf{OR}, \mathsf{NOT}, \Re, y_t, y_{t-1}, y_{t-4}, \pi\}$

(a)

Speciation using a MLP

Speciation: MLP

$C = \{+, -, \times, \mathsf{pdiv}, \mathsf{psin}, \exp, \mathsf{pln}, \mathsf{AND}, \mathsf{OR}, \mathsf{NOT}, \Re, y_t, y_{t-1}, y_{t-4}, \pi\}$

(b)

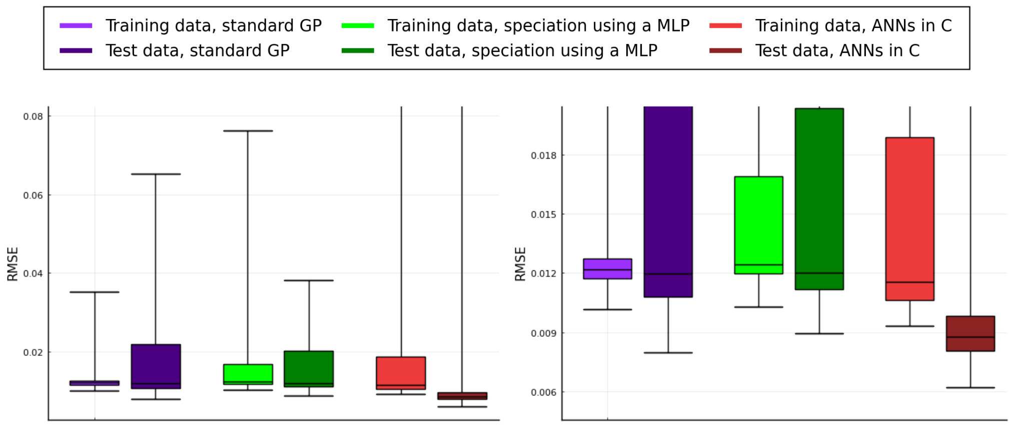
ANNs in C

Speciation: Random

$C = \{+, -, \times, \mathsf{pdiv}, \mathsf{psin}, \exp, \mathsf{pln}, \mathsf{AND}, \mathsf{OR}, \mathsf{NOT}, \Re, y_t, y_{t-1}, y_{t-4}, \pi, \mathsf{net1} - \mathsf{net25}\}$

(c)

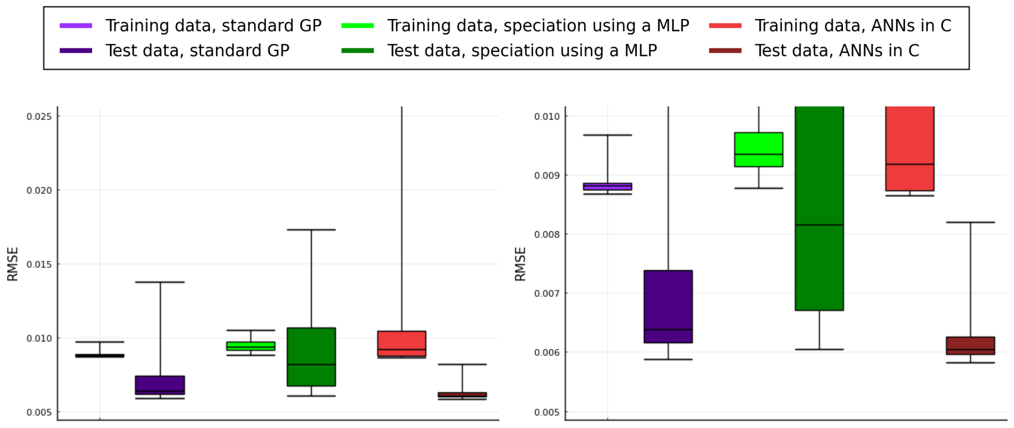
Figure 15: A figure showing how the results are being presented.



(a) All points. (b) Zoomed.

Figure 16: Boxplot of RMSE for single solutions on both on the training and test data. Due to the stochastic nature of the single solutions obtained by equation (11), the estimate $\hat{\boldsymbol{y}}_i$ and $\hat{\boldsymbol{y}}_{\text{te},i}$ of the training and test data, respectively, has been sampled 500 times, for $i = 1, \dots, 5$. For each sample, the RMSE of that sample against the observed value has been calculated. All these RMSE-values has been collected for 9 runs. The purple boxes represent the standard GP, the green boxes represent the GP with MLP generated species and the red boxes represent the GP with ANNs in the combined set. Note that the RMSE-values in the green boxes has been obtained by equation (12), hence they each consist of $5 \cdot 9 = 45$ values.

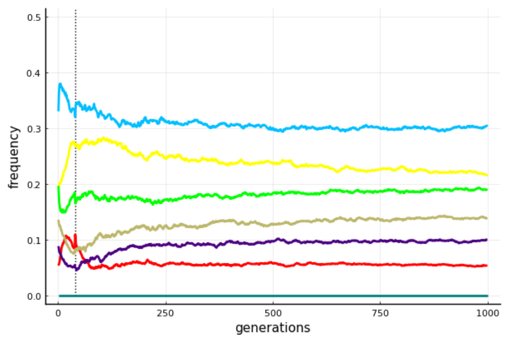
It can be observed that giving the GP neural information processing, gathers the error on unseen data, see plot (a) and also compare the size of the dark purple and the dark red box in plot (b). Further, the median is lowered when ANNs is added to the combined set and unchanged when having MLP generated species, see plot (b).



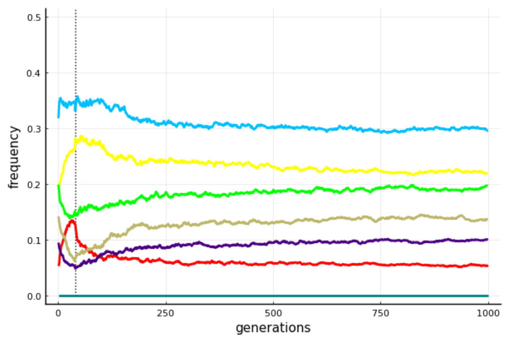
(a) All points. (b) Zoomed.

Figure 17: Boxplot of RMSE for weighted solutions both on the training and test data. The purple boxes represent the standard GP, the green boxes represent the GP with MLP generated species and the red boxes represent the GP with ANNs in the combined set.

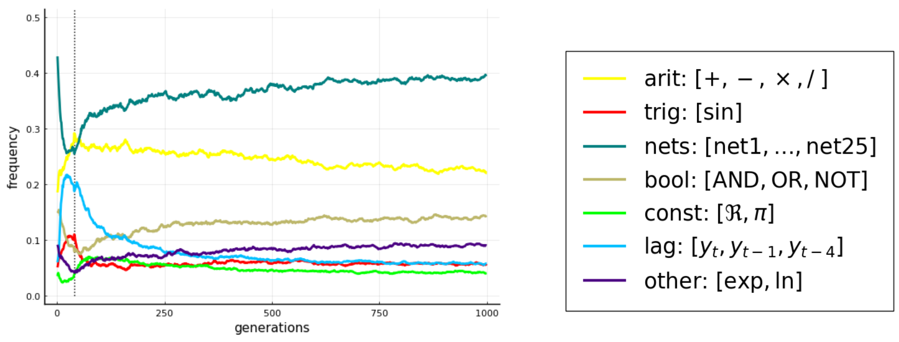
It can be observed that the weighted solutions do not gain accuracy by leveraging neural information processing. Further, speciation using a MLP enlarge the spread of the error, see plot (a).



(a) Standard GP.



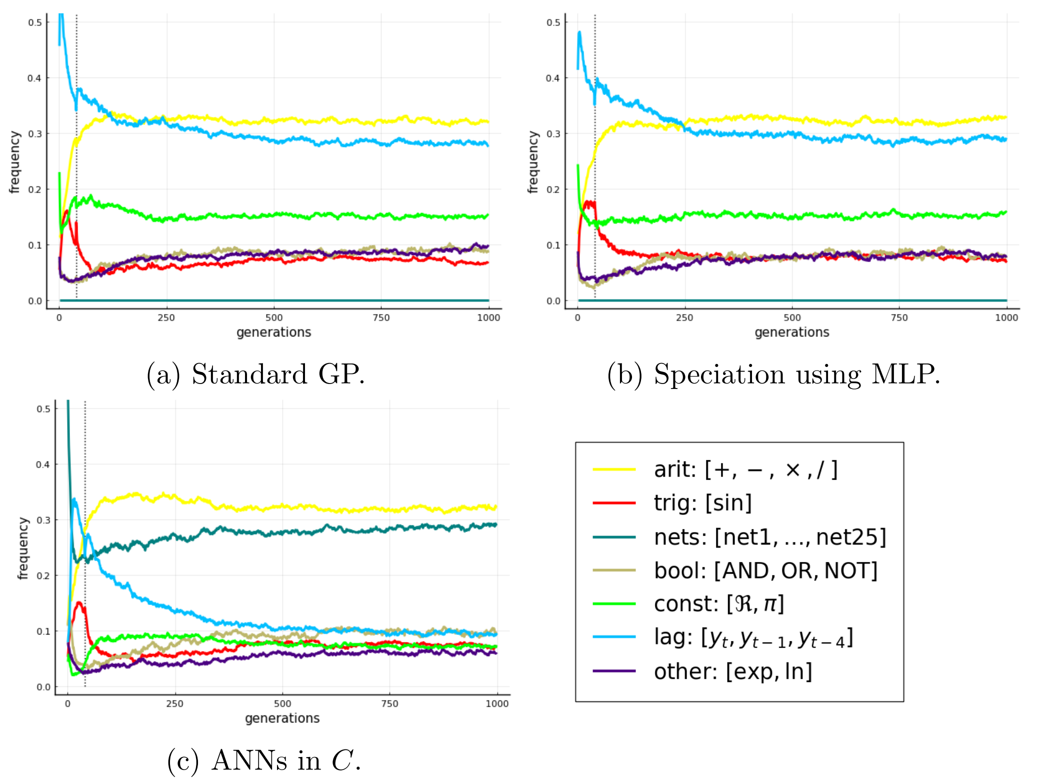
(b) Speciation using MLP.



(c) ANNs in C.

Figure 18: The total gene frequency within the population during one run. The vertical dotted line marks the bootstrap in generation 40.

It can be observed that speciation using a MLP does not change the gene frequency, see plot (b), while adding ANNs to the combined set does, see plot (c). The use of the lags (light blue) and the constants (green) decreases and the GP replace these with the ANNs (turquoise).



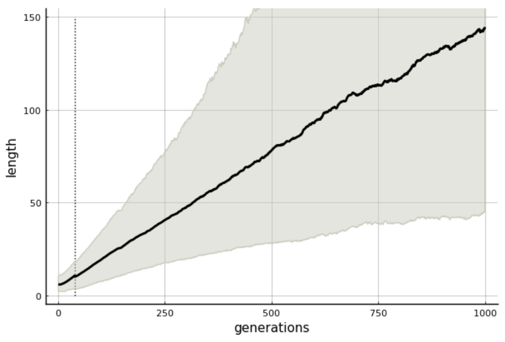
(a) Standard GP.

(b) Speciation using MLP.

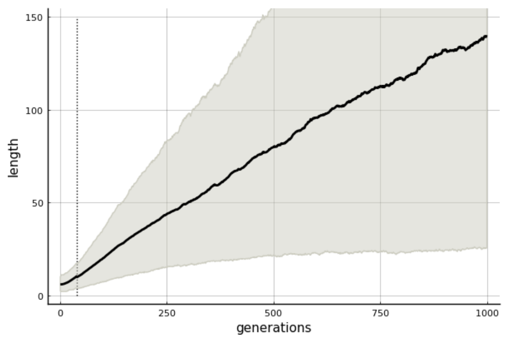
(c) ANNs in C.

Figure 19: The active gene frequency within the population during one run. The vertical dotted line marks the bootstrap in generation 40.

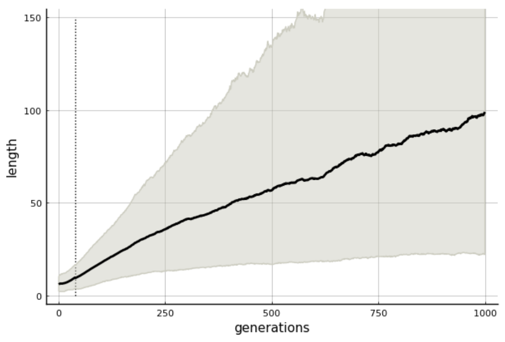
It can be observed that speciation using a MLP does not change the active gene frequency, see plot (b), while adding ANNs to the combined set does, see plot (c). The use of the lags (light blue) and the constants (green) decreases. Also the use of the exponential and the natural logarithm slightly decrease. The GP seems to replace these with the ANNs.



(a) Standard GP.



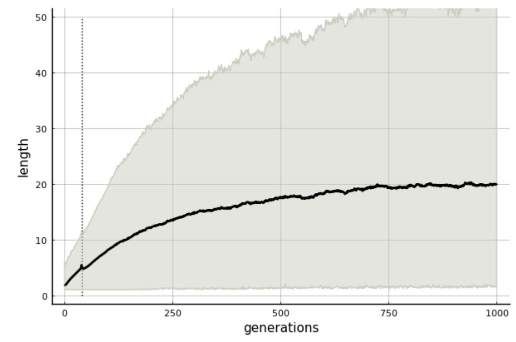
(b) Speciation using MLP.



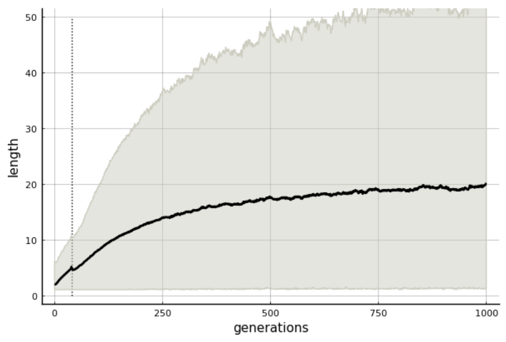
(c) ANNs in C.

Figure 20: The total chromosome length. Let $\bar{\ell}$ be the mean chromosome length of the population. Furthermore, let ℓ_5 and ℓ_{95} be the 5% and 95% quantiles of the chromosome length within the population. Then the black line represents the mean of $\bar{\ell}$ for 13 runs and the upper and lower grey line, that is the boundary of the grey area, represents the mean of ℓ_5 and ℓ_{95} for 13 runs. The vertical dotted line marks the bootstrap in generation 40.

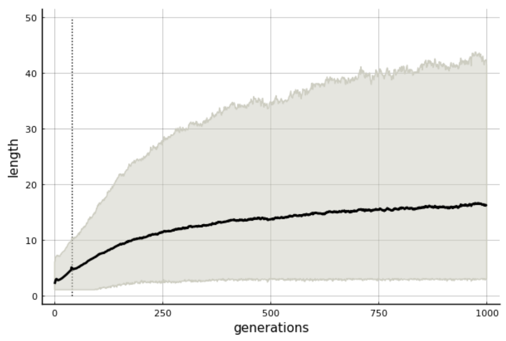
It can be observed that both speciation using a MLP and adding ANNs to the combined set lowers the 5% quantile. Speciation using a MLP does not decrease the mean chromosome length, see plot (b). While adding ANNs to the combined set clearly reduce the mean chromosome length and decrease the growth rate of the chromosomes, see plot (c).



(a) Standard GP.



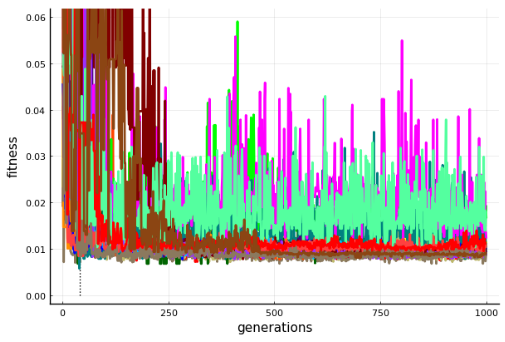
(b) Speciation using MLP.



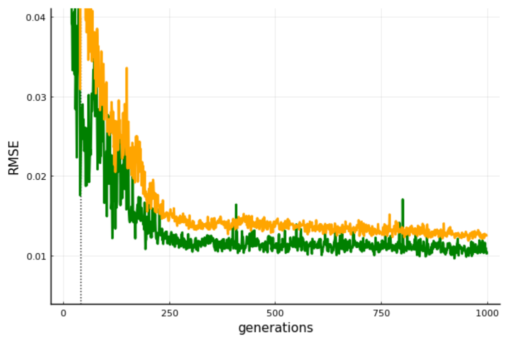
(c) ANNs in C.

Figure 21: The size of evolved programs, i.e. the active chromosome length. Let $\bar{\ell}$ be the mean active chromosome length of the population. Furthermore, let ℓ_5 and ℓ_{95} be the 5% and 95% quantiles of the active chromosome length within the population. Then the black line represents the mean of $\bar{\ell}$ for 13 runs and the upper and lower grey line, that is the boundary of the grey area, represents the mean of ℓ_5 and ℓ_{95} for 13 runs. The vertical dotted line marks the bootstrap in generation 40.

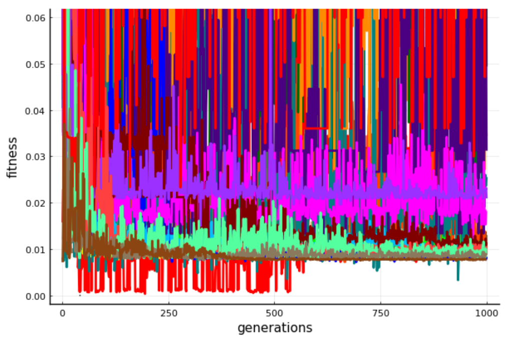
It can be observed that speciation using a MLP does not change the active chromosome length, compare plot (a) and (b). By adding ANNs to the combined set, the mean active chromosome length decreases and the 95% quantile clearly drops down, compare plot (a) and (c).



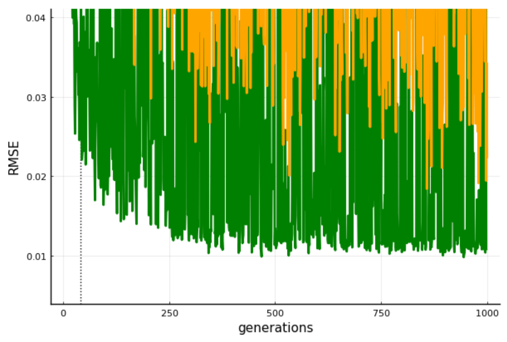
(a) Standard GP.



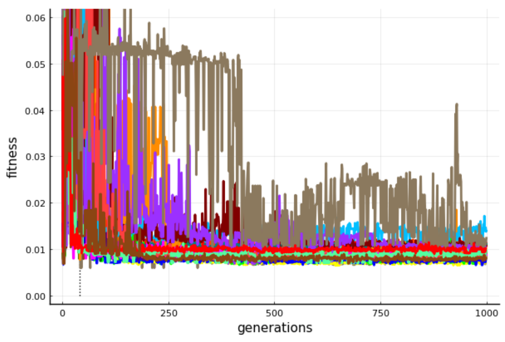
(b) Standard GP.



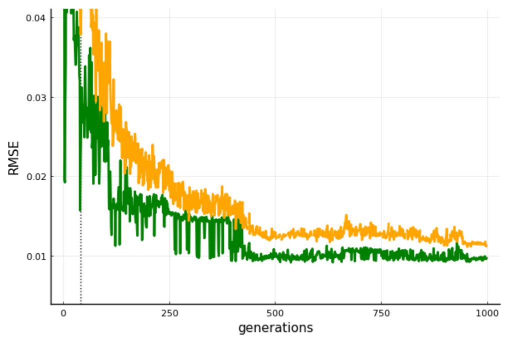
(c) Speciation using MLP.



(d) Speciation using MLP.



(e) ANNs in C.



(f) ANNs in C.

Figure 22: The convergence. To the left, the fitness values $f^*_{1(\kappa)}, \forall\, \kappa \in \{1, 2, \ldots, \eta\}$ are shown. Let $\hat{\boldsymbol{y}}_i$ be the ith best estimation made on the training data. To the right, the green line represents the root mean square error, defined in section 3.5.4, of $\hat{\boldsymbol{y}}_1$ against the training data ${}_{\text{tr}}$, i.e. $g(\boldsymbol{Y}_{\text{tr}}, \hat{\boldsymbol{y}}_1)$. The orange yellow line represents the RMSE mean of the 15 best estimations, i.e. $\left(\sum_{i=1}^{15} g(\boldsymbol{Y}_{\text{tr}}, \hat{\boldsymbol{y}}_i)\right)/15$.

It can be observed that the noisiness differs between the different versions of the GP. It should be mentioned that tuning the parameters α and σ of the fitness sharing function, may change the noisiness. By changing the α and σ value, the difference in noise between plot (a) and (e) could change, concluding that the noise rate of plot (a) and (e) is not significantly different. On the other hand, the convergence when using a MLP for speciation remained noisy, see plot (c) and (d). Plot (c) also show that there are some species that find it hard to converge.

5 Discussion

The gene frequency in figure 18c and figure 19c clearly show that the GP leverages the neural networks functions when these are added to the combined set. The lag-signals are the genes that decrease the most. The ANNs contain information about the lags (the lag-signals are inputs to the ANNs) and this might explain why such a vast part of the lags is no longer needed.

The result of figure 20 and figure 21 is that the chromosome length decreases when ANNs are added to the combined set. The neural network functions are more powerful functions than standard arithmetic, trigonometric and exponential functions, which probably explains this behavior. Instead of making a complex combination of several variables and simpler functions, the GP can achieve the same by using one of the ANNs. At the same time, the ANN is an already defined function which cannot be destroyed by crossover or mutation, while the complex combination of variables and simpler functions can.

By examine the sensitivity and specificity i figure 14, one could argue that some of the networks has been over-fit. For example, class 10 has 100% on both sensitivity and specificity for all top three individuals. This has not been considered a direct problem. First, the GP does not know what is right or wrong in terms of classification. It will use functions that it finds to be useful and suppress functions it finds to be less useful. Second, the classification is based on our partitioning of the data, which also has no distinct rule of right or wrong. Many other approaches to partitioning the data can be carried out, but in the end, it is the GP that is the magistrate of what is useful and not.

The data partitioning approach taken in this work is up for discussion as well as further improvements. An article by Lin and Keogh [47], arguing that subsequence clustering of time series (STS) provides no value. They argue that using this method while clustering gives the same result as if one would have clustered a time series of a random walk. An opposing view is presented in the paper *Making subsequence time series clustering meaningful* by Chen [48], showing that STS clustering can indeed be meaningful.

The clustering approach in this thesis may explain the increased noise in the convergence in figure 22c and figure 22d.

6 Conclusions

In this masters thesis, the effects of leveraging neural information processing has been studied. Two slightly different methods has been employed for this purpose. The first method has used a MLP for speciation and the second has incorporated ANNs in the GP's combined set.

Using a MLP for speciation did not make any improvement. The gene frequency did not change at all. The single solutions performed a little worse than the standard GP, and the weighted solutions performed worse. In fact, the most significant difference were that the convergence became more unstable, or noisy. Making it possible for the algorithm to stop when the precision where suboptimal. Resulting in a larger spread of the prediction error when examine the weighted solutions. The noise in convergence probably indicates that there is room for improvement of the partitioning of the data.

Using the ANNs in the combined set C resulted in less terminals appearing in the population. The GP filled that gap with the ANNs instead, that got the second highest gene frequency within the population.

This method appears to have a significant and positive effect, as it decreases the bloat. The mean of the total chromosome length of each individual continue to grow at about the same pace during the whole run for the other two version of the GP. While for this method, the growth rate of the mean of the total chromosome length of each individual lower, resulting in a decrease of bloat. The same trend is observed for the mean size of the evolved programs (the active chromosome length for each individual), but less significant. Overall this show that leveraging neural information processing can act as a way of controlling the bloat.

The single solutions improved when the ANNs were added to the GP, when applied on unseen data. Taking the mean of the individual predictions, i.e. the weighted solution, further improved the prediction accuracy (for all methods). Comparing the weighted solutions, the prediction accuracy were not significantly improved when the ANNs where added to the GP. Indicating that mainly single solutions benefits from leveraging neural information processing.

7 Further research

A univariate time series has been studied in this work, given the defined constraints of the study. Including more dimensions is believed to be beneficial, for prediction accuracy, such as leveraging news sentiments within the sector that BVI is at.

Speciation using ANNs did not work as expected. The idea is still believed to be useful, but the way of partitioning the data will probably benefit from further investigations. Advisedly, one could study other clustering methods and maybe using more dimensions when partitioning the data.

As mentioned in section 1.5, Yu et al. [9, 10] published two papers. In their second paper, they incorporated λ abstractions in their GP and reported this to have a positive effect [10]. Further investigations covering that area seems promising.

References

[1] Julia 1.5 documentation. *The Julia Language*. Accessed March. 16, 2021. [Online]. URL `https://docs.julialang.org/en/v1/`.

[2] Mehar Vijh, Deeksha Chandola, Vinay Anand Tikkiwal, and Arun Kumar. Stock closing price prediction using machine learning techniques. *Procedia Computer Science*, 167:599–606, 2020. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2020.03.326. URL `https://www.sciencedirect.com/science/article/pii/S1877050920307924`. International Conference on Computational Intelligence and Data Science.

[3] Mingyue Qiu and Yu Song. Predicting the direction of stock market index movement using an optimized artificial neural network model. *PLOS ONE*, 11:e0155133, 05 2016. doi: 10.1371/journal.pone.0155133.

[4] Kyoung jae Kim and Ingoo Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, 19(2):125–132, 2000. ISSN 0957-4174. doi: https://doi.org/10.1016/S0957-4174(00)00027-0. URL `https://www.sciencedirect.com/science/article/pii/S0957417400000270`.

[5] Kenneth O. Stanley. Neuroevolution: A different kind of deep learning, Jul 2017. URL `https://www.oreilly.com/radar/neuroevolution-a-different-kind-of-deep-learning/`.

[6] Lean Yu, Wei Huang, Kin Keung Lai, Yoshiteru Nakamori, and Shouyang Wang. Neural networks in finance and economics forecasting. *International Journal of Information Technology & Decision Making (IJITDM)*, 06:113–140, 03 2007. doi: 10.1142/S021962200700237X.

[7] Crina Grosan and Ajith Abraham. *Stock Market Modeling Using Genetic Programming Ensembles*, volume 13 of *Studies in Computational Intelligence*, chapter 6, pages 131–146. Springer-Verlag Berlin Heidelberg, 2006.

[8] Christopher Neely, Paul Weller, and Rob Dittmar. Is technical analysis in the foreign exchange market profitable? a genetic programming approach. *Journal of Financial and Quantitative Analysis*, 32:405–426, 10 1996. doi: 10.2307/2331231.

[9] Tina Yu, Shu-Heng Chen, and Tzu-Wen Kuo. *A genetic programming approach to model international short-term capital flow*, volume 19 of

Advances in Econometrics, pages 45–69. Emerald Group Publishing Limited, 2004. ISBN 9781849503037.

[10] Tina Yu, Shu-Heng Chen, and Tzu-Wen Kuo. *Discovering Technical Trading Rules Using λ Abstraction GP*, chapter 2, pages 215–226. Springer Science + Business Media, Inc., 2005.

[11] Brian Beers. How a buy-and-hold strategy works, Aug 2020. URL `https://www.investopedia.com/terms/b/buyandhold.asp`.

[12] James McDermott, Alexandros Agapitos, Anthony Brabazon, and Michael O'Neill. *Geometric Semantic Genetic Programming for Financial Data*, pages 215–226. Springer-Verlag Berlin Heidelberg, 2014.

[13] J. Arshad, A. Zameer, and A. Khan. Wind power prediction using genetic programming based ensemble of artificial neural networks (GPeANN). In *2014 12th International Conference on Frontiers of Information Technology*, pages 257–262. The Institute of Electrical and Electronics Engineers, Inc., 2014. doi: 10.1109/FIT.2014.55.

[14] Genome. *Wikipedia.* Accessed March. 2, 2021. [Online], . URL `https://en.wikipedia.org/wiki/Genome#Coding_sequences`.

[15] Chromosomes. *Khan Academy.* Accessed March. 18, 2021. [Online], . URL `https://www.khanacademy.org/science/high-school-biology/hs-reproduction-and-cell-division/hs-chromosome-structure-and-numbers/a/dna-and-chromosomes-article`.

[16] Genotype. *Wikipedia.* Accessed March. 4, 2021. [Online], . URL `https://en.wikipedia.org/wiki/Genotype`.

[17] Phenotype. *Wikipedia.* Accessed March. 4, 2021. [Online], . URL `https://en.wikipedia.org/wiki/Phenotype`.

[18] Mathematical optimization. *Wikipedia.* Accessed March. 29, 2021. [Online], . URL `https://en.wikipedia.org/wiki/Mathematical_optimization`.

[19] Lars-Christer Böiers. *Mathemaical Methods of Optimization.* Studentlitteratur, 2010. ISBN 978-91-44-07075-9.

[20] Yongsheng Fang and Jun li. A review of tournament selection in genetic programming. pages 181–192, 10 2010. ISBN 978-3-642-16492-7. doi: 10.1007/978-3-642-16493-4_19.

[21] John R. Koza. *Genetic Programming - On the Programming of Computers by Means of Natural Selection.* Massachusetts Institute of Technology, 1998. ISBN 0262111705.

[22] Biman Chakraborty and Probal Chaudhuri. On the use of genetic algorithm with elitism in robust and nonparametric multivariate analysis. 32, 01 2003.

[23] Friden, inc. *Wikipedia.* Accessed March. 17, 2021. [Online], . URL `https://en.wikipedia.org/wiki/Friden,_Inc.`

[24] Hewlett-packard voyager series. *Wikipedia.* Accessed March. 17, 2021. [Online], . URL `https://en.wikipedia.org/wiki/Hewlett-Packard_Voyager_series`.

[25] Rpn. *The Museum of HP Calculators.* Accessed March. 29, 2021. [Online]. URL `https://www.hpmuseum.org/rpn.htm`.

[26] Riccardo Poli, William Langdon, and Nicholas Mcphee. *A Field Guide to Genetic Programming.* 2008. ISBN 978-1-4092-0073-4.

[27] Leonardo Trujillo, Enrique Naredo, and Yuliana Martínez. Preliminary study of bloat in genetic programming with behavior-based search. In Michael Emmerich, Andre Deutz, Oliver Schuetze, Thomas Bäck, Emilia Tantar, Alexandru-Adrian Tantar, Pierre Del Moral, Pierrick Legrand, Pascal Bouvry, and Carlos A. Coello, editors, *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation IV*, pages 293–305, Heidelberg, 2013. Springer International Publishing. ISBN 978-3-319-01128-8.

[28] Fixation (popultion genetics). *Wikipedia.* Accessed March. 4, 2021. [Online], . URL `https://en.wikipedia.org/wiki/Fixation_(population_genetics)`.

[29] Genetic drift. *Khan Academy.* Accessed March. 2, 2021. [Online], . URL `https://www.khanacademy.org/science/ap-biology/natural-selection/population-genetics/a/genetic-drift-founder-bottleneck`.

[30] Niches & competition. *Khan Academy.* Accessed March. 16, 2021. [Online], . URL `https://www.khanacademy.org/science/ap-biology/ecology-ap/community-ecology/a/niches-competition`.

[31] Steven Gustafson. An analysis of diversity in genetic programming. 05 2004.

[32] Universal approximation theorem. *Wikipedia.* Accessed March. 2, 2021. [Online], . URL `https://en.wikipedia.org/wiki/Universal_approximation_theorem`.

[33] Ding-Xuan Zhou. Universality of deep convolutional neural networks, 2018.

[34] R Dhanapal and D. Bhanu. Electroencephalogram classification using various artficial neural networks. *Journal of critical reviews*, 7:891–894, 2020. doi: 10.31838/jcr.07.04.170.

[35] Alexander Waibel, Toshiyuki Hanazawa, G. Hinton, Kiyohiro Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37:328–339, 04 1989. doi: 10.1109/29.21701.

[36] Sam Schechner. Facebook boosts ai to block terrorist propaganda, Jun 2017. URL `https://www.wsj.com/articles/facebook-boosts-a-i-to-block-terrorist-propaganda-1497546000`.

[37] Jack V. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231, 1996. doi: 10.1016/S0895-4356(96)00002-9.

[38] Yavar Bathaee. The artificial intelligence black box and the failure of intent and causation. *Harvard Journal of Law & Technology*, 31:889, 2018.

[39] Thomas Grote and Philipp Berens. On the ethics of algorithmic decision-making in healthcare, Mar 2020. URL `https://jme.bmj.com/content/46/3/205`.

[40] Daniel Faggella. Ai transparency in finance - understanding the black box, Jan 2020. URL `https://emerj.com/partner-content/ai-transparency-in-finance/`.

[41] Frank Rosenblatt. The perceptron: A probabailistic model for information storage and organization in the brain, 1958. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf`.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[43] Greg Hamerly and Charles Elkan. Alternatives to the k-means algorithmthat find better clusterings. pages 600–607, 2002.

[44] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[45] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[46] Species & speciation. *Khan Academy*. Accessed Feb. 8, 2021. [Online], . URL `https://www.khanacademy.org/science/biology/her/tree-of-life/a/species-speciation`.

[47] Jessica Lin and Eamonn Keogh. *Finding or Not Finding Rules in Time Series*, volume 19 of *Advances in Econometrics*, pages 175–201. Emerald Group Publishing Limited, 2004. ISBN 9781849503037.

[48] Jason R. Chen. Making subsequence time series clustering meaningful. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–, 2005. doi: 10.1109/ICDM.2005.91.

Master’s Theses in Mathematical Sciences 2021:E6
ISSN 1404-6342

LUTFMA-3438-2021

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/