2018

Regularization in Symbolic Regression by an Additional Fitness Objective

Ryan Grindle
University of Vermont

Regularization in Symbolic Regression by an Additional Fitness Objective

A Thesis Presented

by

Ryan Grindle

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Master of Science
Specializing in Mathematics

October, 2018

Defense Date: August 23, 2018
Dissertation Examination Committee:

Abstract

Symbolic regression is a method for discovering functions that minimize error on a given dataset. It is of interest to prevent overfitting in symbolic regression. In this work, regularization of symbolic regression is attempted by incorporating an additional fitness objective. This new fitness objective is called Worst Neighbors (WN) score, which measures differences in approximate derivatives in the form of angles. To compute the Worst Neighbors score, place partition points between each pair of adjacent data points. For each pair of data points, compute the maximum angle between the line formed by the pair of data points and the lines formed by adjacent partition points. The maximum of all these maximum angles is the Worst Neighbors score. This method differs from other attempts to regularize symbolic regression because it considers the behavior of the evolved function between data points. A high WN score indicates that the function has overfit the data. A low score could indicate either an underfit solution or a well fit solution. The error objective is used to make this distinction. Worst Neighbors can reduce overfitting in symbolic regression because it encourages functions that have a low error and a low Worst Neighbors score. The error objective helps stop the solutions from becoming underfit and the Worst Neighbors score helps stop the solutions from becoming overfit. To use Worst Neighbors for target functions of higher dimensions, select nearby points as neighbors and compute the Worst Neighbors score on the evolved function restricted to the plane formed by these neighbors and the output direction. For the one dimensional case, Worst Neighbors has promise in reducing error on unseen data when compared with Age-Fitness Pareto Optimization (AFPO). WN achieves a small decrease in testing error on several target functions compared to AFPO.

Acknowledgements

Thank you Josh Bongard and James Bagrow for your guidance and everything that you have taught me. Thank you to the members of the Morphology, Evolution and Cognition Laboratory for you interesting conversations, helpful comments, and friendly faces. Thank you friends and family for your support. And, thank you to the Mathematics Department at the University of Vermont who funded me with a Graduate Teaching Assistantship.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computers can perform straight forward computations quickly. There is also an abundance of easily accessible data in the world; in fact, 90% of data in the world was generated in the past few years [29]. Machine learning is one way to utilize both the computation speed of computers and this abundance of data. Data is used by machine learning algorithms as experience by which a computer can improve at a specific task without being explicitly programmed.

One problem that machine learning is capable of solving is symbolic regression. That is, the problem of inferring the relationship between two or more variables from a finite dataset. Unfortunately, a common problem encountered in symbolic regression is overfitting. An overfit function is one that has “learned” the dataset well, but does not generalize to unseen data. This is especially common with a noisy target function. Since a finite dataset is used to train a model that actually has infinitely many elements in its domain, it is understandable that overfitting occurs. Overfitting causes problems when actually trying to use the model. This is because symbolic regression is supposed to create solutions that accurately predict outputs outside the training data. In this thesis, a methods is presented for lessening overfitting.

In this thesis, Worst Neighbors (WN) is explained, discussed, and tested as a method for reducing overfitting in symbolic regression. In this method, error is used as one fitness objective while the second fitness objective, Worst Neighbors score, measures the differences in approximate derivatives. By using both fitness objectives, WN encourages the evolution of well fit solutions. To measure overfitting, assume that the function is approximately linear between adjacent data points. Then, compare the angles between the line formed by the adjacent data points and secant lines on the evolved function between the data points. The largest angle is the Worst Neighbor score.

Worst Neighbor solutions must achieve low error on unseen data to show promise in reducing overfitting. In this thesis, Worst Neighbors is compared with Age-Fitness Pareto Optimization (AFPO) in error on unseen data. AFPO was chosen to be compared with WN because AFPO is a popular and successful algorithm [3, 2, 16, 31]; however, AFPO was not designed specifically to solve the overfitting problem. In the future, it makes sense to compare WN with other methods designed to reduce overfitting. The results show lower error on unseen data using WN (rather than AFPO) for several target functions. The results also include experiments designed to determine if the amount of noise has any effect on Worst Neighbors. Finally, a comparison of model complexity between WN and AFPO will be performed.

Chapter 2

Background

This section provides an overview of supervised learning, regularization, overfitting, and genetic programming with a focus on symbolic regression. It provides details to guide in understanding the Worst Neighbors algorithm.

2.1 Supervised Learning

Supervised learning is a process that modifies a system based on the system's responses to inputs, which requires the knowledge of specific outputs resulting from specific inputs called training data [26]. Often, the solution to a supervised learning problem is a function. Supervised learning is a subset of machine learning and thus it is a method for solving a task without explicitly programming the solution.

The training data for a supervised learning task looks like $D = \{(\boldsymbol{x}_k, y_k)\}_{k=1}^{n}$. That is, given any $\boldsymbol{x}_k$, the system should produce y_k. The supervision in this type of learning comes from the training data and a fitness or objective function. Say the goal of the system is to discover a function f such that $f(\boldsymbol{x}_k) = y_k$. With this goal, the typical objective function is error. For example, error could be calculated as

$$e(f) = \frac{\sum_{k=1}^{n} |f(\boldsymbol{x}_k) - y_k|}{n}.$$

The objective function uses the training data to supervise the system by informing it of the ability of possible solution f to solve the given problem. In an unsupervised learning algorithm, the data would not be given labels [26]. For example, the k-means clustering algorithm is an unsupervised learning technique because its data is not initially labeled (or classified). Instead, the k-means clustering algorithm determines classification of each data point by calculating its distance from other data points [5].

2.1.1 Cross-Validation

Cross-validation is a way of estimating the accuracy of a potential solution to a supervised learning task. The most basic method of cross-validation is called holdout. It splits the dataset into two datasets: a training dataset and a testing dataset. The training dataset is used during training (as described in the previous section, 2.1) while the testing dataset is not used until training has ended. The testing dataset is used to measure the accuracy of the solution. This measurement is the error of the solution on the testing dataset rather than the training dataset [14, 26].

There is also k-fold cross-validation, which (1) creates k subsets of the dataset of similar size, (2) picks one subset as the training dataset and combines the others to form the training dataset, and (3) the accuracy is measured as in the holdout method. This is done for each of the k subsets and the final accuracy estimate is the average of all the holdout estimates [12, 14].

The validation dataset is like the testing dataset in that is not used to train the solution, but it is used before the machine learning algorithm completes. For example, the validation dataset can be used to create an early stopping criterion. The simplest version of this is to record the error on the training dataset and the error on the validation dataset at every iteration of the algorithm. When the error on the validation dataset starts to increase, stop the machine learning algorithm and use the current solution as the final one [25].

Unfortunately, the terms testing and validation are sometimes interchanged. For example, in [12], the term validation dataset corresponds to the definition of testing data described here.

2.2 Genetic Programming

Genetic programming as described in by Koza in [15] is aimed at using computers to creating solutions to tasks without explicitly programming the solution by mimicking natural selection. For each task, a virtual population of individuals (or algorithms) is generated randomly. The individuals attempt the task and receive a score reflecting their ability to solve the task. This score is called the fitness of the individual. The best individuals (those who achieve low fitness) are either combined or mutated to create new, hopefully more successful, individuals. This process is repeated for many generations. When the evolution process stops, hopefully individuals that solve the task exist in the final population.

2.2.1 Trees

The individuals in genetic programming are represented as trees. A tree is a graph with no cycles (no closed loops) and all nodes (or vertices) have at least one link (or edge) connecting them. This structure is convenient for representing algorithms, which are computer programs or mathematical functions for example.

Create trees for use by genetic programming by using elements from the functional set F and terminal set T as nodes. The functional set contains operations that can be applied to any element of the terminal set. Since the elements of the functional set do not produce anything by themselves, they must be connected to terminal nodes eventually. The terminal set can have elements such as variables, constants, and actions. The specific contents of the functional and terminal sets are chosen based on the task. For example, choose $F = \{+, -, \times, \sin\}$ and $T = \{x, y, 1, 2, 3\}$ to create mathematical functions. Figure 2.1 shows examples of trees built using these sets.

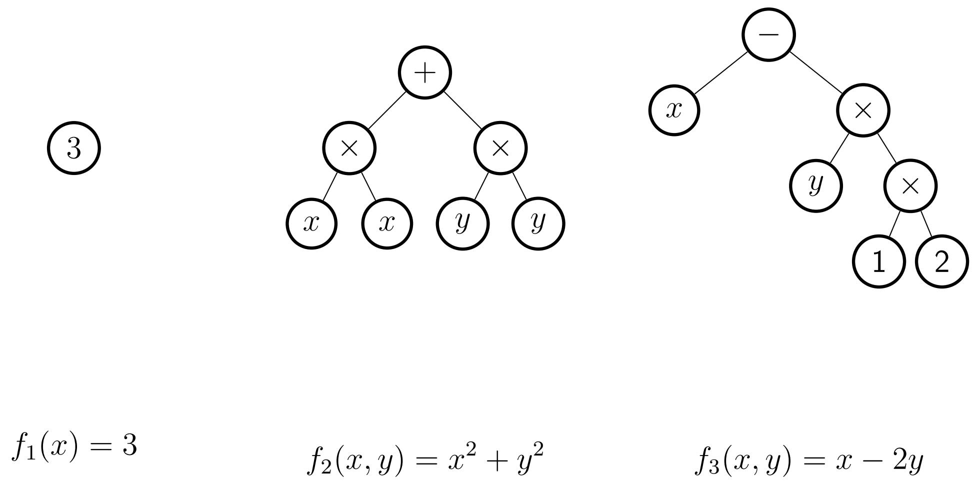

Figure 2.1: Examples of trees using $F = \{+, -, \times, \sin\}$ *and* $T = \{x, y, 1, 2, 3\}$.

From Figures 2.1, f_1 is one of the simplest function that can be made which consists of a single terminal node. Functions f_2 and f_3 are a bit more complex. The tree for f_3 is larger than necessary. This is common among evolved trees and one reason that it is easy to create large trees. When trees grow uncontrollably large it is called bloat. This is problematic because it can makes it difficult to understand the underlying solution. The solution is also more computationally expensive and more memory is required to store the solution. Finally, a large tree is more likely to be an overfit solution than a small tree.

One way to limit bloat is to pick a maximum depth. The depth of a tree is the largest number of links from the root node (top node) to any other node. For the tree that represents f_1, the depth is 0 since there are no links. For the tree representing f_2, the max depth is 2 since all the terminal nodes are separated by 2 links from the $+$ node. For the tree representing f_3, the max depth is 3 since the terminal nodes 1 and 2 are 3 steps away from the $-$ node.

Assuming all elements of the functional set require exactly two child nodes, the number of nodes in a full tree – a tree with as many nodes as possible – is the sum of the number of nodes at each depth, which is 2^d where d is the depth. Thus, the maximum number of nodes for a tree of depth d is $\sum_{k=0}^{d} 2^k$ or equivalently $2^{d+1} - 1$. Koza in [15] uses an initial max depth of 6 and from then on a max depth of 17. Using these max depths the number of nodes used is at most 127 nodes for a max depth of 6 and up to 262,143 nodes for a max depth of 17. Hopefully this upper bound will not be approached.

Generating Random Trees

Generation of random trees was discussed previously. There are two basic ways to generate random trees. In the grow method, a node is selected randomly from $F \cup T$. If the node receives an element from F, the appropriate number of nodes are created beneath using the same process. If the maximum tree depth is reached, the node is given an element from T to stop the growth. Trees created with the grow method can be any depth between 0 and the max depth.

Another method for creating trees is the full method. In this method, the nodes are chosen only from F until the max depth is reached. At which point, the nodes are chosen from T. The trees created using the full method always have depth equal to the max depth and the maximum number of nodes (if all the elements of F require the same number of child nodes).

Both these methods are used to initialize the population in the ramped half-and-half method [15, 24]. In this method, create half the individuals using the grow method and generate the other half using the full method while increasing the max depth from 2 to the initial max depth.

Mutating Trees

During evolution, the best trees in the current population are mutated. The structure of the trees created by this operation will be slightly different from the original tree; however, the semantics (or behavior) of the new tree is not necessarily different from the semantics of the old tree. To mutate a tree,

1. Select a node from the tree at random.
2. Delete the selected node and all nodes below it. Replace this node with a new subtree.

Figure 2.2 shows the process of mutating a tree. Step 1 selects the rightmost $\times$ node (highlighted in red). For step 2, the subtree, which includes the two y terminal nodes is deleted and a new subtree starting with sin is placed in the original tree. Note that it is unclear from the information given whether this new tree is an improvement over the old tree; however, the semantics of the trees are different since y^2 is not equivalent to $\sin xy$. Many other mutation are possible. This is only one example.

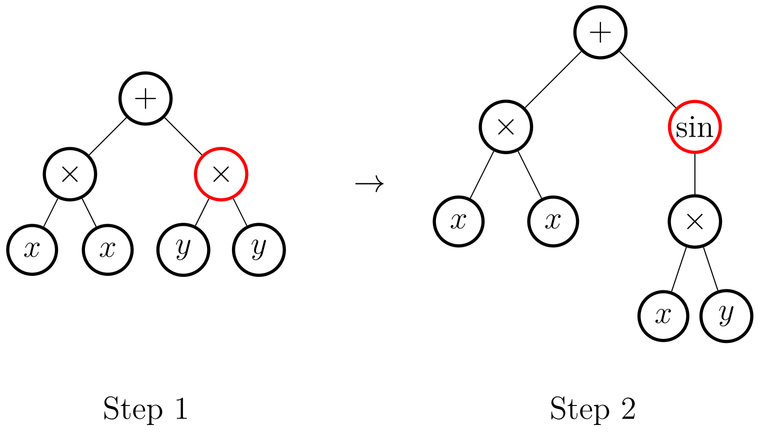

Figure 2.2: An example of mutating a tree

There are many other types of mutation that can be done. This is the standard one present by Koza in [15]. It is also possible to do crossover, which takes two individuals, picks subtrees from each, and swaps them. This type of recombination is difficult to use effectively because of the Competing Conventions Problem, which states that crossover is likely to produce deleterious results when the genomes being crossed-over represent the same solution in different forms [30]. This is certainly true of functions, which are evolved in symbolic regression. For example, the functions $f(x) = x^2$ and $g(x) = 6+x^2-6$ are equivalent functions but represented by structurally different trees.

The standard method of crossover in GP is to swap subtrees. This method does not attempt to determine at which nodes it is best to swap subtrees. For this reason, the offspring created by crossover are unlikely to capture the best aspects of their parents' genomes.

Despite the Competing Conventions Problem, crossover can be a useful tool. Ideally, crossover will combine good qualities from both parents. Using alternative crossover algorithms, the likelihood of this can be increased. One example is Looseness-Controlled Crossover (LCC) [34]. In LCC, each link is given a weight called "looseness." The higher this weight the more likely this link is to be chosen as the place to split the tree. Initially, all weights are 0. Crossover is performed as standard crossover (since all the weights are equal); however, the children created are compared with their parents. If the child is less fit than its parents, the crossover it undone and a new one is attempted. If the child is more fit than its parents, then the weight associated with crossover is held constant while that other weights increase by one. When crossover it performed again, the split point is chosen based on the weights rather than at random. In [34], LCC is shown to outperform standard GP crossover in an object classification task.

In semantic GP, the focus is moved from the structure (genotype) of trees to the semantics (phenotype) of trees. Thus, the goal in semantic crossover is to produce children with similar semantics to their parents [10].

The semantics of a tree is a vector of the output of the tree for several given inputs. The semantic distance between two trees is simply the distance between the two vectors that represent their semantics. That is, a large semantic distance between trees means that the functions behave differently. A small semantic distance means that the trees behave similarly.

There are several different types of semantic crossover such as: Semantic Aware Crossover and Semantic Similarity-based Crossover [22], Geometric Semantic Crossover [20], and a variation on Geometric Semantic Crossover [10]. In [20], a geometric crossover is defined as a map from two parents to their child where the child is guaranteed to be between its parents in semantic space. When evolving real functions, [20] defines a geometric crossover by letting the parents be $p_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, 2$ and creating their child c by $c = p_1 r + (1 - r)p_2$ where r is either a real number in $[0, 1]$ or r is a real valued function with codomain of $[0, 1]$. An unfortunate side effect of this type of crossover, is that the trees grow extremely large.

2.2.2 Symbolic Regression

Symbolic regression is a type of genetic programming and supervised learning that attempts to find a mathematical function to describe the given dataset. This function is built from the elements of the functional and terminal sets. This means that the functions produced can have many different structures.

Symbolic regression differs from classical regression because in classical regression the structure of the mathematic function is known and the regression only determines the values of the coefficients while symbolic regression makes no assumption about the structure of the underlying mathematical function.

Overfitting

An overfit function is an overly complex model [11]. In [11], it is emphasized that overfitting is a comparison, meaning that a function is overfit if there exist another function that is simpler but explains the data with an equal level of success. An overfit function has probably learned the noise in the dataset causing the function not to fit well with unseen data. Because function error (against the training dataset) only depends on a finite number of points, there are infinitely many points where error cannot be computed. Thus, functions with similarly low error can have extreme differences in behavior in areas of the domain where their is no training data. Of these functions, the ones that are more complex are the overfit functions.

Overfitting is undesirable because (1) overfit functions require additional computation time compared with well fit functions, (2) overfit function may contain unneeded variables, which requires more measurements to be taken. The primary problem is that (3) overfit solutions do not generalize well to unseen data. The aim of this thesis is to present a method in symbolic regression that will help lessen overfitting. The success of the presented algorithm will be measured by a comparison involving the error of evolved functions on unseen data.

Protected Functions

With all forms of genetic programming, it is important that all the elements of the functional set accept any element from the terminal set as inputs. This is called the closure property. However, common elements such as / and $\sqrt{\ }$ lack this property. To fix this problem, make protected versions of these functions and use them instead. The division operator only has a problem in the case where the divisor is 0. The protected division operator will return 1 in this case. To create a protected version of the square root function use the square root of the absolute value of the input. This avoids inputing negative numbers into a square root. These are by no means the only ways to create protected functions.

Multi-Objective Optimization

As mentioned before, minimizing the error of a function does not always result in the best solution. Describing the best solution to GP can be done by adding additional fitness objectives. This is called Multi-Objective Optimization [13]. Number of nodes in the function is a good example of a fitness objective. Using both error and number of nodes as objectives means that the goal of GP is now to minimize both objectives. Using number of nodes will help avoid bloat and discourage overfitting. One of the great benefits of symbolic regression, and more generally GP, is the algorithm it produces can be read. This is most easily done for small trees, which is another reason to use number of nodes as a fitness objective.

In multi-objective optimization it is necessary to determine which individuals are the best based on multiple criteria. To do this, say an individual I

is better than – or dominates – another individual J if individual I is at least as good as J in each fitness objective and better in at least one objective. This can be stated mathematically by letting the fitness objectives applied to an individual I be denoted $f_1(I), f_2(I), \cdots, f_n(I)$ for the n fitness objectives. Then, the same definition can be stated as individual I dominates individual J if $f_k(I) \leq f_k(J)$ for all k and $f_m(I) < f_m(J)$ for at least one m, assuming each fitness objective is to be minimized. To avoid keeping copies of individuals, if two individuals have the same scores for each fitness objective, the most recently created individual dominates the other. This also allows for neutral mutation that may later lead to a beneficial one.

To better understand this definition, recall the previous example where there are two fitness objectives: error and size (number of nodes). Say we have a population of size 3 and a dataset $\{(-1,1),(0,0),(1,1)\}$. Figure 2.3 shows a possible population as trees.

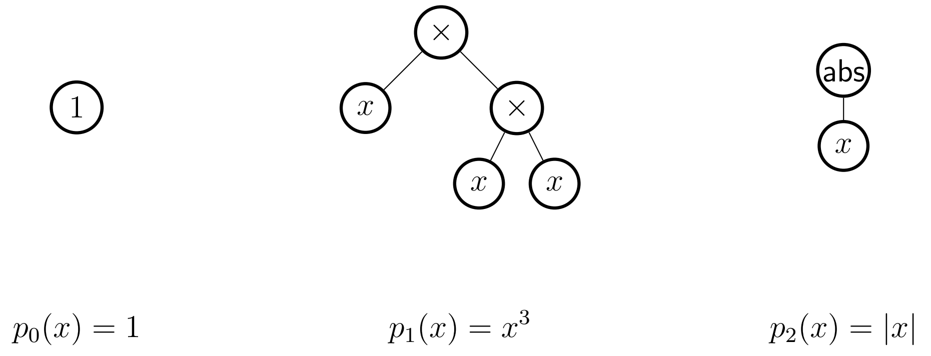

Figure 2.3: A population of three functions

The average absolute error of the trees in this population are 1/3, 2/3, and 0 from left to right. The sizes of the trees are 1, 5, and 2. At this point, the definition of the term *dominates* can be used to see that p_0 and p_2 dominate

p_1 and both p_0 and p_2 are not dominated by any individual in the population. Another way to say this is that p_0 and p_2 are non-dominated. Figure 2.4 shows this graphically by plotting size versus error.

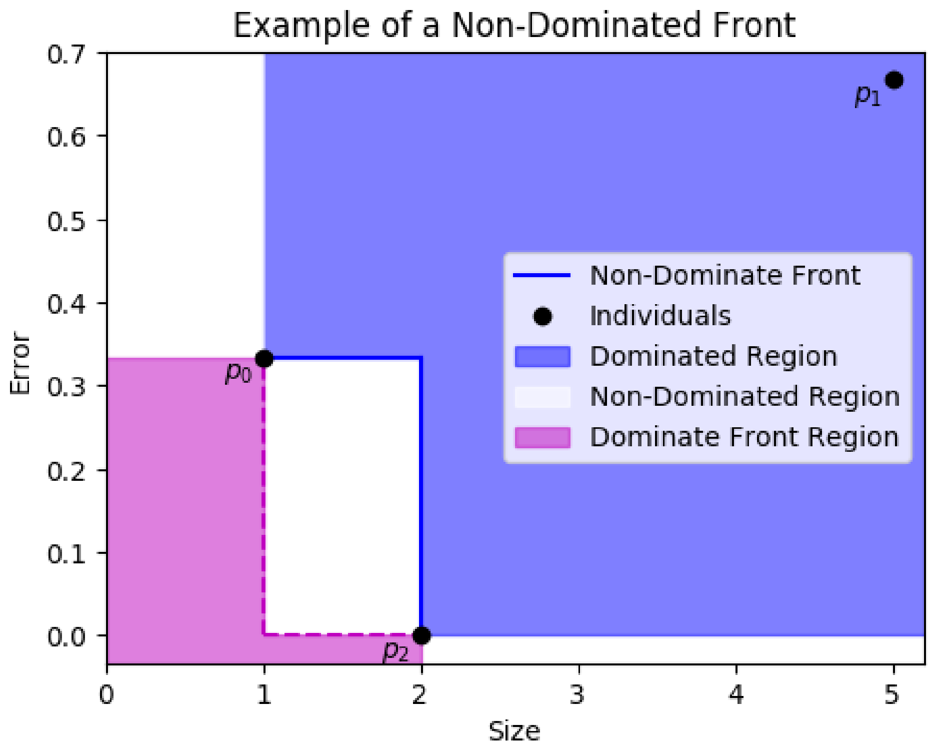

Figure 2.4: An example of a non-dominated front.

Each point, in Figure 2.4, is one of the individuals in the population and the solid, blue line segments represent the non-dominated front. Any individual that exits above and to the right of these lines (in the blue shaded region) is dominated. An individual that exists below and to the left of the dashed, purple line segments (in the purple shaded region) will dominate the entire front. Any individual that is in the white shaded regions would become part of the non-dominated front.

AFPO

AFPO stands for Age Fitness Pareto Optimization. It was designed by Lipson and Schmidt in [28]. They use age as the second fitness objective. At each generation, randomly generated individuals are added to the population with age 0. The individuals on the non-dominated front, the survivors, increase their age by one. And, the children of any of the survivors inherit their age from their parents. By considering low age better than high age, young individuals are able to explore the search space and hopefully find better solutions. The age objective implicitly encourages small trees because the randomly generated trees that are added to the population are small. As evolution progresses, some of the young individuals are able to find a better solution than the previous best (old) solution. When this happens, the old solutions are deleted and the new ones remain. It is likely the the new solutions are smaller than the old ones.

2.3 Related Work

There are many different methods that can be used to avoid overfit solutions. In this section, several methods are discussed.

One way to avoid overfit solutions is with stopping criteria. If an overfit solution has evolved then, most likely, the function was not overfit at some point during evolution. Papers like [8] look at stopping criteria in order to detect when continuing evolution will no longer be beneficial. This particular paper works with semantic GP. Its stopping criteria are called Error Deviation Variation (EDV) and Training Improvement Effectiveness (TIE). Evolution stops when either EDV or TIE drop below a given threshold. Error deviation is the standard deviation of improvements in absolute error for each datum in the training dataset. Only the the models that have improved over the previously best model in the current generation are considered when computing the percentage of models that reduce error deviation in comparison with the previously best model. This criterion is attempting to determine when GP can only improve fitness by creating higher error deviation, which indicates overfitting.

The Training Improvement Effectiveness (TIE) criterion measures the percentage of times that a semantic recombination operator creates a solution that is better than the current best [8]. This measurement is designed to determine the effectiveness of the operator at the current stage in evolution. When it becomes unlikely for the operator to be able to create improved solutions, GP is stopped.

Another completely different method for solving the overfitting problem, [27], is to assume there is noise and learn its effects on the underlying relationships by letting the algorithm choose where to place the noise in the model. In [27], an additional element is placed in the terminal set. This element represents noise as a uniform random variable. Using this method, well fit solutions can be created and the source of the noise can be discovered. Since the output of any tree containing this element is a distribution at each input value, the fitness function must be adjusted. Their goals is "to explain all variation found in the training data, and do so in the narrowest and simplest way." Thus, the fitness function becomes piecewise. If the output value is inside the distribution, the height of the distribution at the corresponding input value is the fitness. If, instead, the output value is not inside the distribution, the distance between the distribution and the output value is the fitness. This is computed for each data point and summed for the overall fitness of the function.

In [1] an additional fitness objective is used as is done in this thesis. [1] uses error and error between the derivative of the evolved function and the approximate derivative of the dataset. The derivative of the dataset is approximated by measuring the slope between adjacent data points. As with all methods discussed, the error objective encourages the evolved functions to go through the data points. The error on the first derivative encourages the evolved functions to travel in the same direction as indicated by the dataset at each data point. This algorithm uses an analytic quotient operator to ensure differentiability. The results from this paper indicate a decrease in average tree size when compared with standard GP although their was no statistical test done to confirm this. Both training error and test error are comparable.

Variance-Based Layered Learning GP attempts to solve the overfitting problem by breaking the task into a set of subtasks from simplest to most complex [9]. This is done by creating simplified versions of the original training dataset called primitive datasets. Models are evolved on the simplest primitive dataset first. At each generation, the models are evaluated on their variance (on the validation dataset). This measurement is used to determine when overfitting occurs. When this stopping criterion is met, the population from that last generation becomes the first generation for the next primitive dataset, which is at least as complex as the previous primitive dataset.

In [9], variance is a measure of overfitting, so the variance of the outputs of each model is measured when it is created. If this new individual has larger variance than a given threshold, it is deleted and another individual is created instead. This helps to maintain low variance. This same measure is also used on the output values of the datasets to determine their complexity. The primitive datasets are created by scaling the output of the original training dataset to alter the variance.

In [4], Structural Risk Minimization is applied to GP. Structural Risk Minimization uses Vapnik-Chervonenkis (VC) dimension to measure complexity. VC dimension of a classification model is the size of the largest set that can be classified (for any assignment of two classes) by the model. For example, a linear classifier of one input variable has VC dimension 3 because it can correctly classify a set of 3 elements into the correct 2 classes for any assignment of class, but it cannot correctly classify a set of 4 elements in all cases. The VC dimension can be used to estimate the risk or testing error of a model. This risk should be minimized to create good models. This is the goal of Structural Risk Minimization. Estimating the VC dimensions of GP trees is the main difficulty with using SRM in GP.

2.3.1 Regularization

Regularization is a process which incorporates additional information to avoid overfitting [26]. Regularization can be applied to different types of problems. Here the focus will be placed on regularization applied to regression. Remember that regression is the problem of finding a function f such that $y_k = f(\boldsymbol{x}_k) + \varepsilon_k$ where $(\boldsymbol{x}_k, y_k)$ are data points and ε_k is taken from a normal distribution with mean 0. Some, but not all types of regularization, are aimed at minimizing

$$M(f) = e(f) + \lambda\Omega(f)$$

where e is some error function and λ is a parameter that modifies the strength of the penalty function Ω. Equivalently, this might be stated as minimize $e(f)$ subject to the condition $\Omega(f) \leq t$ where t is a parameter that, like λ, modifies the strength of the penalty function Ω. Regularization is used if minimizing only $e(f)$ produces overfit solutions. The penalty function $\Omega(f)$ should output larger numbers for a more overfit function f. This penalty can be further adjusted by changing λ or t. For example, if $\lambda = 0$ then no penalty has been incorporated into the regression. As λ is increased, the penalty becomes increasingly persuasive. If parameter t is used, the penalty condition becomes stronger by decreasing t.

In linear regression, there is a regularization method called least absolute shrinkage and selection operator (LASSO) [32]. LASSO, minimizes the equation $\sum_{i=1}^{n}(y_i - \alpha - \sum_j \beta_j x_{i,j})^2$ subject to $\sum_j |\beta_j| \leq t$. This constraint encourages small coefficients. The goal is to "reduce the variance of the predicted values" [32]. Small coefficients are good because this gives the regression algorithm a reason to try to eliminate unneeded variables from the model. There are many ways to apply regularization methods including smoothing splines.

Smoothing Splines

Smoothing splines are a form of regularized polynomial regression. Before describing smoothing splines further, splines will be defined. A spline or polynomial spline is a piecewise polynomial. A spline f of order r is a real-valued function on $[a, b]$ such that (1) for a partition $P = \{a = t_0 < t_1 < \cdots < t_{n+1} = b\}$ of $[a, b]$ where each t_k is called a knot, f is a piecewise polynomial of degree r on $[t_k, t_{k+1})$ for $k = 0, 1, \cdots, n+1$ and (2) f has $r-2$ continuous derivatives and the $(r-1)$-th derivative is a step function with jumps at knots [33].

A smoothing spline is a spline found by minimizing

$$M(f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2 + \lambda \int_a^b (f^{(m)}(x))^2\, dx \tag{2.1}$$

as stated in [6, 33]. This always has a unique solution for any $\lambda \in [0, \infty)$ in the Sobolev space [6]. Any element in the Sobolev space $W_2^m[a, b]$ is a function with absolutely continuous $(m - 1)$-th derivatives and with the square of the m-th derivative integrable [6, 33]. The use of the Sobolev space ensures that $M(f)$ is defined for any $f \in W_2^m[a, b]$. In M, $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$ is included to minimize the error, so that the predictions made by f will be accurate. The purpose of $\int_a^b (f^{(m)}(x))^2\, dx$ is to incorporate a penalty for functions that are not smooth. There will be no penalty for an f where each polynomial is of degree m-1 or less since this term will be 0. More information on why this term measures the roughness of f can be found in [33]. The severity of the penalty term is controlled by λ. If $\lambda = 0$, there is no penalty. As λ increases, so does the severity of the penalty.

Chapter 3

Worst Neighbors

Worst Neighbors (WN) is the name of the method for regularization in symbolic regression, which is presented in this thesis. It has two fitness objectives: error and Worst Neighbors score. The WN score is an angle that describes differences in approximate derivatives associate with the evolved function. More specifically, for each interval formed by adjacent inputs in the training data a secant line is formed on the evolved function, using the end points as x-values. Between the adjacent inputs, create additional input values called partition points. Each adjacent pair of partition points forms another secant line. The original secant line is compared to each of the lines formed by the partition. The largest angle difference between these lines is the "worst neighbor" for the pair of points. This is repeated for each adjacent pair of points in the training data. Later the advantages and disadvantage of using max or average of all these "worst neighbors" will be discussed. For now, the WN score is max of all the "worst neighbors," which has a codomain of $[0, \pi]$.

The WN score is an attempt to measure the amount of overfitting in the evolved function. It is assumes that the target function behaves nearly linearly between adjacent data points and thus so should the evolved functions. In other words, the behavior of a function should not change drastically between adjacent data points. A large WN score indicates a function that has overfit the data. A small WN score indicates that the function has not overfit the data. For a small score, the function may be underfit or well-fit. It is difficult to tell which is the case with only a WN score. This is why WN has two fitness objectives. The error objective will discourage underfit solutions.

Worst Neighbors relies on a finite dataset and finite partitions (additional x-values). Unfortunately, this means that it is possible for a WN score to inaccurately measure the amount of overfitting on a function. To reduce the likelihood of this, pick non-uniform partitions. This allows the WN score to be computed with slightly different points each time. The hope is that symbolic regression will be able to recover from an incorrect score, by a change to the partition. For example, Worst Neighbors would measure the amount of overfitting inaccurately if the partition intervals where picked such that the change in x between each point was exactly the period of the function. In this case, the Worst Neighbors score would be 0 since all the y-values would be the same. Avoid this problem by picking non-uniform partitions and give the algorithm a chance to correct an inaccurate measurement by picking partitions differently at each iteration.

3.1 Picking Partitions

To compute the WN score, it is necessary to create a partition. A partition P of the interval $[a, b]$ is a real, finite sequence $\{p_0, p_1, p_2, \cdots, p_n\}$ such that $a = p_0 < p_1 < p_2 < \cdots < p_n = b$. There are many ways to pick such a partition. The following are several ways to construct a partition on the interval $[a, b]$.

1. **A Uniform Partition**

   For $k = 0, 1, 2, \cdots, n$, let

   $$q_k = a + \frac{k}{n}(b - a).$$

   Then, $Q = \{q_0, q_1, \cdots, q_n\}$ is a uniform partition. A uniform partition will not be used because a periodic function with a period equal to the step size of the partition would be given an inaccurate score. Also, a partition that is unlikely to be the same each time it is created is desired to allow WN to recover from an inaccurate score. The uniform partition is explained here to help describe the semi-uniform partition.

2. **A Semi-Uniform Partition**

   Let $p_0 = a$ and $p_n = b$. Pick all other p_k's based on the uniform partition. That is, pick a random $p_k \in [q_{k-1}, q_k)$ where q_k is the k^{th} element of the uniform partition. In other words, pick

   $$p_k \in \left[a + \frac{k-1}{n-1}(b - a), a + \frac{k}{n-1}(b - a)\right)$$

   for $k = 1, 2, \cdots, n - 1$. To avoid the possibility of selecting a for p_1, pick p_1 in the open interval (a, q_1) rather than the half open interval used elsewhere. Note that the semi-uniform partition has $n + 1$ elements but the uniform partition used to construct it only has n elements.

3. **A Random Partition**

   Let $p_0 = a$ and $p_n = b$. Pick all other p_k's such that each p_k is a randomly selected number in the interval (a, b). The partition will need to be sorted once all p_k's are picked. It is a good idea to check that no two partition elements are the same.

As stated previously, uniform partitions should be avoided. After describing the computation of WN score, the semi-uniform partition and the random partition with be compared.

3.2 Computing Worst Neighbors Score

In this method, form line segments on the evolved function (based on the x-values of the given dataset), and create partitions between these points. Then, compute the angles between the data lines and the partition lines. Large angles indicate overfitting.

3.2.1 One-Dimensional Worst Neighbors

In the one-dimensional case, the evolved function is of the form $\hat{f} : \mathbb{R} \to \mathbb{R}$ and the dataset is of the form $D = \{(t_k, y_k)\}_{k=0}^{m-1}$ where the t_k terms are inputs and the y_k terms are outputs. The data set is written as a sequence so it may be assumed that $t_k < t_{k+1}$ for all k. To compute the Worst Neighbors score, the y_k's will not be used. Instead use the t_k's to find a point on the evolved function. To map the points onto the evolved function, define

$$\boldsymbol{\phi}(x) = \Big(x, \hat{f}(x)\Big). \tag{3.1}$$

For each pair of adjacent inputs t_{k-1} and t_k, create a partition

$$P = \{t_{k-1} = p_0 < p_1 < p_2 < \cdots < p_{r-1} = t_k\}$$

of $[t_{k-1}, t_k]$ and form the vector $\boldsymbol{v}_k = \boldsymbol{\phi}(t_k) - \boldsymbol{\phi}(t_{k-1})$. For each subinterval of P, create the vector $\boldsymbol{u}_j = \boldsymbol{\phi}(p_j) - \boldsymbol{\phi}(p_{j-1})$. Now, compute the "worst neighbor" in the subinterval by

$$\theta_k = \max\left\{\cos^{-1}\left(\frac{\boldsymbol{v}_k \cdot \boldsymbol{u}_j}{\|\boldsymbol{v}_k\| \, \|\boldsymbol{u}_j\|}\right) : j \in \mathbb{N}, j < r\right\}. \tag{3.2}$$

To get the final worst neighbors score, take

$$\theta = \max\{\theta_k : k \in \mathbb{N}, k < m\}. \tag{3.3}$$

Figure 3.1 shows the process of computing one θ_k. The leftmost plot of this figure shows the dataset (which has some noise), and the actual curve (which is unknown), and an evolved curve that is overfit. The rightmost plot zooms in on the left plot on the interval $[t_1, t_2]$. Vectors are drawn as described above. The red vector is from $(t_1, f(t_1))$ to $(t_2, f(t_2))$ and the black vectors are between adjacent partition points (green x's). The bottom plot shows all the vectors moved to have the same starting point, so that the angles between them can be compared. For the interval $[t_1, t_2]$, the worst neighbor is β_0. From the notation of Equation 3.2, this angle should be called θ_2. The process will be the same for all other $[t_{k-1}, t_k]$ intervals. Note the magnitudes of the black vectors where adjusted to make the visualization easier to see.

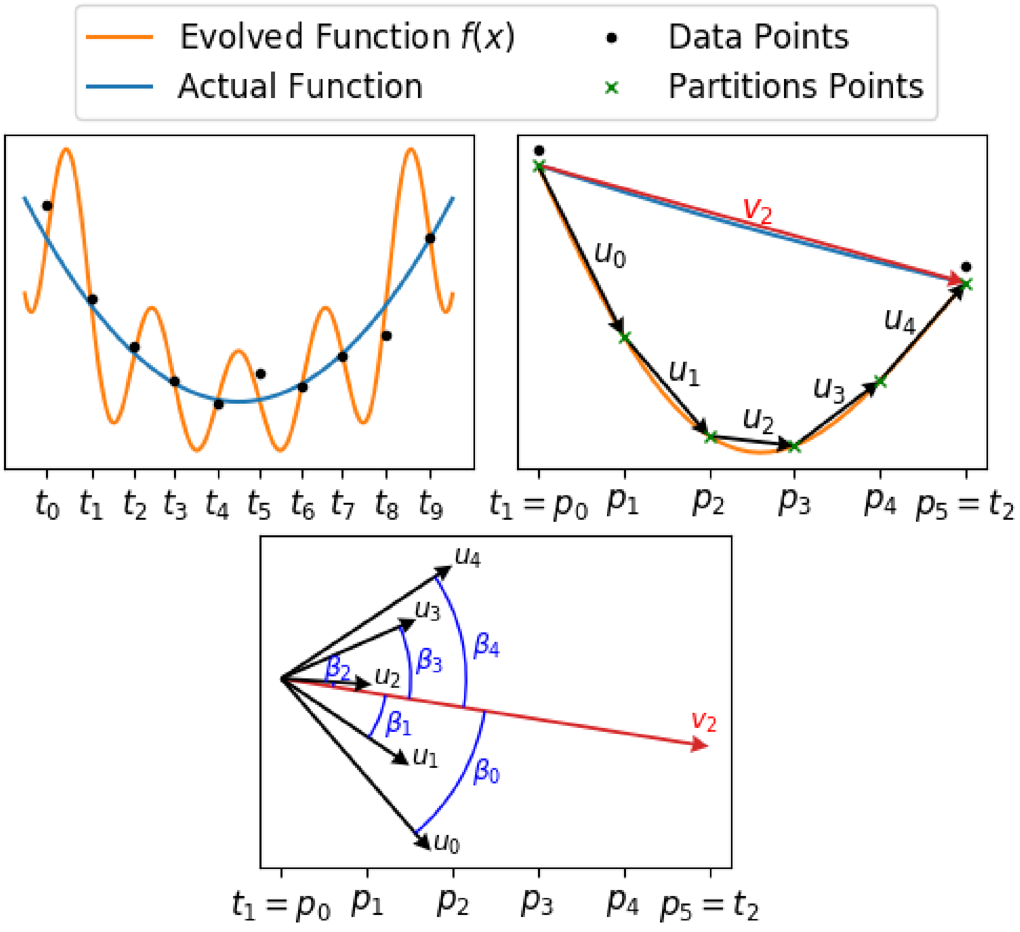

Figure 3.1: Visualization of the computation of WN score.

The underlying assumption of the Worst Neighbors algorithm is that an evolved function that is well fit to the dataset behaves nearly linearly between adjacent data points. Due to noise, it is expected that output values will not match exactly. This means it is important to create a dataset which represents the target function well enough that there are no surprises between data points. The data points and target function of Figure 3.1 provide an example of such a dataset.

Error is still an important fitness objective since the worst neighbors score will likely be low for underfit solutions. For example, any piecewise linear function will receive a worst neighbor score of 0. Some such functions are plotted in Figure 3.2.

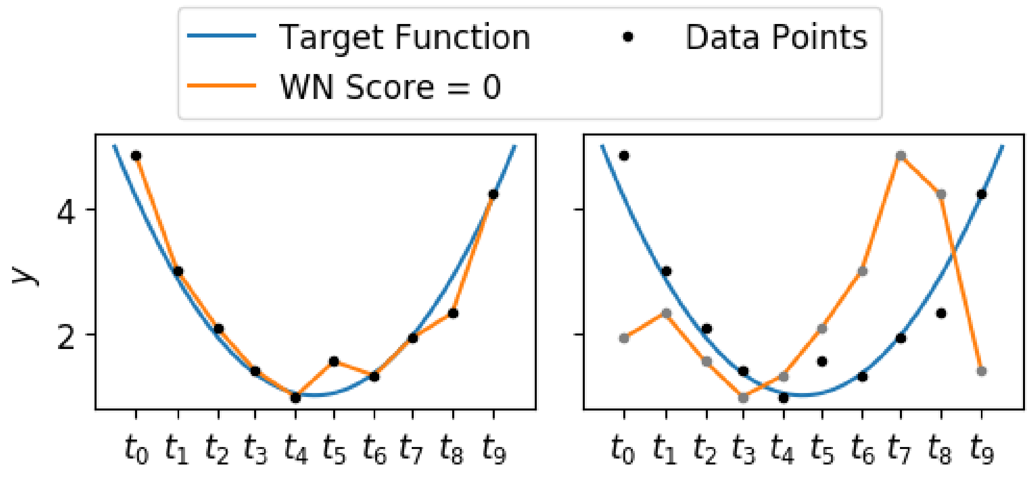

Figure 3.2: Examples of functions that receive a WN score of 0. The function with WN score of 0 (left) has much lower error than the function with WN score (right). This illustrates the importance of using an error objective with WN.

3.2.2 Multidimensional Worst Neighbors

In multiple dimensions, the evolved function has the form $\hat{f} : \mathbb{R}^n \to \mathbb{R}$ and the dataset has the form $D = \{(\boldsymbol{t}_k, y_k) : k \in \mathbb{Z}, 0 < k < m - 1\}$ where the $\boldsymbol{t}_k$ terms are inputs and the y_k terms are outputs. Now redefine, Equation 3.1 more generally as

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1, x_2, x_3, \cdots, x_n, \hat{f}(\boldsymbol{x})\right) \tag{3.4}$$

where $\boldsymbol{x} = (x_1, x_2, x_3, \cdots, x_n)$. Take a "slice" of multidimentional space to consider angles between vectors like the single dimensional version of WN score. This "slice" is a plane defined by the $n + 1$ dimensional vectors $(0, 0, 0, \cdots, 1)$ and $(\boldsymbol{t}_i - \boldsymbol{t}_j, 0)$ where $\boldsymbol{t}_i$ and $\boldsymbol{t}_j$ are considered neighbors.

First, determine which data points are neighbors. Find the a (a parameter than can be determined later) nearest neighbors of each $\boldsymbol{t}_k$ in the data set. Note that the neighbors are determined by the input values only.

For each pair of neighbors $\boldsymbol{p}$ and $\boldsymbol{q}$ construct the parametric line segment $\boldsymbol{\gamma}(s) = (1 - s)\boldsymbol{q} + s\boldsymbol{p}$ on $s \in [0, 1]$. Then, create a random partition S of the interval $[0, 1]$. For each $\boldsymbol{p}$ and $\boldsymbol{q}$, let $\boldsymbol{v}_k = \boldsymbol{\phi}(\boldsymbol{p}) - \boldsymbol{\phi}(\boldsymbol{q})$ and let $\boldsymbol{u}_j = \boldsymbol{\phi}(\boldsymbol{\gamma}(s_j)) - \boldsymbol{\phi}(\boldsymbol{\gamma}(s_{j-1}))$ where $s_j, s_{j-1} \in S$. Use Equation 3.2, to compute θ_k, which is the maximum angle between $\boldsymbol{u}$ and each $\boldsymbol{v}_j$. In this case, there are am values of θ_k because each of the m points will have a neighbors. As in Equation 3.3, compute the maximum of all θ_k to get the final Worst Neighbors score.

3.3 A Closed-Form Analysis of Worst Neighbors

WN score uses slopes in its computation. These slopes can be replaced with derivatives. The WN score is not calculated this way because it requires that the evolved function be twice differentiable, which is not guaranteed by the typical way of generating trees. There are ways of addressing this issue such as using the analytic quotient operator rather than the protected division operator, which was done in [18, 1] and originally in [23]. This section discusses a version of WN score that uses derivatives.

Equation 3.2 computes the largest angle in each interval $[t_{k-1}, t_k]$. This calculation depends on the partition used, which will be signified by writing θ_k as $\theta_k(P)$. Since the goal of Worst Neighbors score is to measure the largest differences between the evolved function and lines, the ideal output from 3.2

is

$$\theta_k^* = \sup\{\theta_k(P) : P \in \mathcal{P}([t_{k-1}, t_k])\} \tag{3.5}$$

where $\mathcal{P}([t_{k-1}, t_k])$ is the set of all partitions of $[t_{k-1}, t_k]$. This can be computed, under a few assumptions, using derivatives. To be able to compute the angle, given two derivatives (slopes), define

$$\alpha(m_1, m_2) = \cos^{-1}\left(\frac{(1, m_1)\cdot(1, m_2)}{\|(1, m_1)\|\,\|(1, m_2)\|}\right) \tag{3.6}$$

$$= \cos^{-1}\left(\frac{1 + m_1 m_2}{\sqrt{1 + {m_1}^2 + {m_2}^2 + {m_1}^2{m_2}^2}}\right). \tag{3.7}$$

This equation comes from forming vectors along the lines $y = m_i x$ from $(0,0)$ to $(1, m_i)$ where $i = 1, 2$. Now, let $g : \mathbb{R} \to \mathbb{R}$ be the piecewise linear function the connects the data points mapped onto the evolved function $\hat{f}$. This will now be stated more specifically. First, recall that the values $t_0, t_1, t_2, \cdots, t_{m-1}$ are the x-values of the dataset. The linear vertical change between adjacent t_k's is represented by

$$\ell_k(s) = \hat{f}(t_{k-1})(1 - s) + \hat{f}(t_k)(s) \text{ on } [0, 1]. \tag{3.8}$$

Then, g can be defined as

$$g(x) = \ell_k\left(\frac{x - t_{k-1}}{t_k - t_{k-1}}\right) \text{ if } x \in [t_{k-1}, t_k]. \tag{3.9}$$

Since g is piecewise linear, the derivative is a constant on each $[t_{k-1}, t_k]$, which can be written as

$${g_k}'(x) = \frac{\hat{f}(t_k) - \hat{f}(t_{k-1})}{t_k - t_{k-1}} \text{ if } x \in [t_{k-1}, t_k]. \tag{3.10}$$

To avoid defining g' two values at each t_{k-1} and t_k (except t_0 and t_{m-1}), the derivative is written as several functions ${g_k}'$ instead of the single function g'. The ideal Worst Neighbors score (given a dataset D) can be computed using derivatives as

$$\theta_k{}^* = \max\left\{\alpha\Big(\hat{f}'(x), {g_k}'(x)\Big) : x \in [t_{k-1}, t_k]\right\}. \tag{3.11}$$

By assuming that $\hat{f}'$ is differentiable and hence continuous on the compact set $[t_{k-1}, t_k]$, the maximum in Equation 3.11 exists by the Extreme Value Theorem. Furthermore, the maximum value can be determined by the chain rule since $\hat{f}'$ and ${g_k}'$ are differentiable. To find the maximum, compute

$$\frac{d}{dx}\left[\alpha\Big(\hat{f}'(x), {g_k}'(x)\Big)\right] = \frac{\partial \alpha}{\partial \hat{f}'}\hat{f}''(x) + \frac{\partial \alpha}{\partial {g_k}'}{g_k}''(x) = \frac{\partial \alpha}{\partial \hat{f}'}\hat{f}''(x). \tag{3.12}$$

The last equality of Equation 3.12 is true because ${g_k}'(x)$ is constant on $[t_{k-1}, t_k]$, and therefore, ${g_k}''(x) = 0$ on the same interval. This derivative is computable since α and $\hat{f}'$ are differentiable.

Let $A = 1 + \hat{f}'(x){g_k}'(x)$ and $B = 1 + (\hat{f}'(x))^2 + ({g_k}'(x))^2 + (\hat{f}'(x))^2({g_k}'(x))^2$, so that $\alpha(\hat{f}'(x), {g_k}'(x)) = \frac{A}{\sqrt{B}}$. Then,

$$\frac{\partial \alpha}{\partial \hat{f}'} = \frac{A\left[\hat{f}'(x) + ({g_k}'(x))^2\,\hat{f}'(x)\right] - {g_k}'(x)B}{B^{3/2}\sqrt{1-\frac{A^2}{B}}} \tag{3.13}$$

$$= \frac{\Big(\hat{f}'(x) - {g_k}'(x)\Big)\Big(1 + ({g_k}'(x))^2\Big)}{B^{3/2}\sqrt{1-\frac{A^2}{B}}}. \tag{3.14}$$

To compute the max, find the values of x for which 3.14 is zero or undefined. This happens when $\hat{f}''(x) = 0$ (or is undefined) or Equation 3.14 equals 0 (or is undefined). Next, test the value of $\alpha\Big(\hat{f}'(x), {g_k}'(x)\Big)$ at these critical numbers and at $x = t_{k-1}$ and $x = t_k$. The largest of these values is the maximum.

First, consider when Equation 3.14 is undefined. Since $B > 0$ for all x and by the Cauchy-Schwarz inequality $A^2/B \in [0, 1]$, the only time that 3.14 is undefined is when $A^2/B = 1$. This turns out to be when $\hat{f}'(x) = {g_k}'(x)$. The numerator of the derivative is equal to zero only at the same place, $\hat{f}'(x) = {g_k}'(x)$. However, when $\hat{f}'(x) = {g_k}'(x)$, the resulting angle will be 0 (a minimum). Thus, these critical numbers can be ignored.

If $\hat{f}$ is twice differentiable, compute ${\theta_k}^*$ as in Equation 3.11 by finding the x values for which $\hat{f}''(x) = 0$ or is undefined and label these $x_1, x_2, \cdots, x_n$. Then, compute

$$ {\theta_k}^* = \max \left\{ \alpha\Big(\hat{f}'(x), {g_k}'(x)\Big) : x \in \{t_{k-1}, x_1, x_2, \cdots, x_n, t_k\} \right\}. \qquad (3.15)$$

Only a $\hat{f}$ which is linear on each $[t_{k-1}, t_k]$ will receive an ideal WN score (${\theta_k}^* = 0$ for all k) because the data points get mapped onto $\hat{f}$.

3.4 Choosing Parameters

In this section, validity of Worst Neighbors with various parameters will be determined. This includes: determining if WN can detect overfit curves, determining the number of partition points to use, and determining if the maximum or average should be used to compute WN score in Equation 3.3.

3.4.1 Detection of Overfit Curves

Here a cubic function was generated. From this function, data points where picked at random with a bit of Gaussian noise added. Next, higher degree polynomials were fit to the data. This method of curve creation is designed to create overfit curves. Next, WN score was calculated with the differences stated above. Some of the curves created for this process can be seen in Figure 3.3.

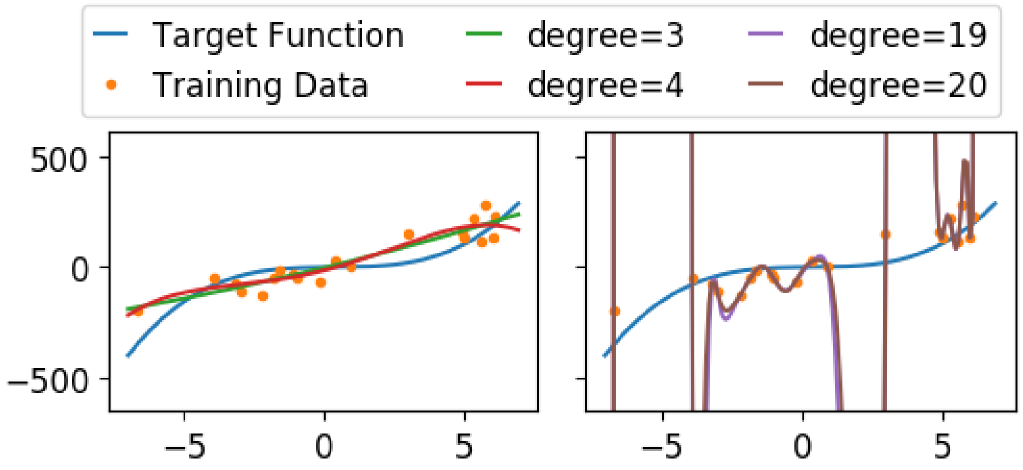

Figure 3.3: The target equation in these plots is $f(x) = x^3 - x^2 + x$. To create the training data, Gaussian noise with a standard deviation of 10% of the range of the data points was added to the y-values. As expected, the higher degree polynomials fit to the data are more overfit.

Polynomials with degree 3 through 20 were created for the following plots. In Figure 3.4, the WN score has been computed on the same 18 polynomials with variations in partition type and using both maximum and average in Equation 3.3.

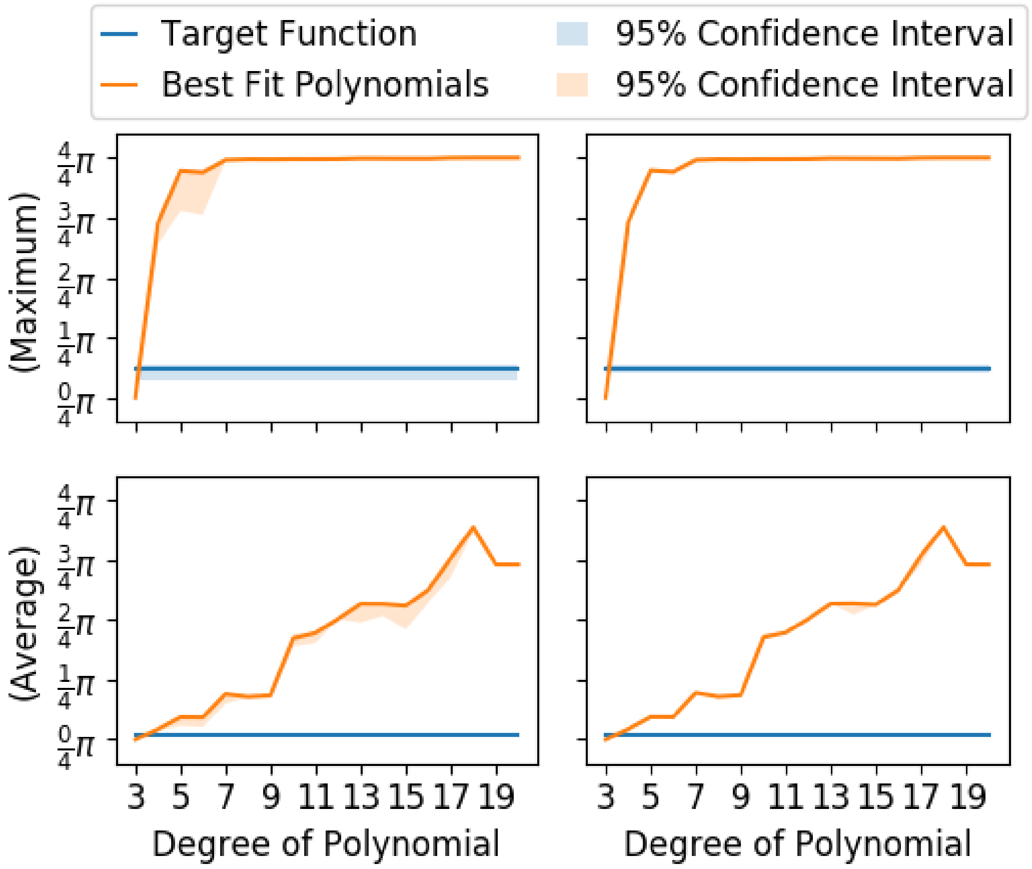

Figure 3.4: In this collection of plots, the left column uses random partitions while the right column uses semi-uniform partitions. The (Maximum) row computes WN as the maximum of all worst neighbors on the subinterval level while the (Average) row uses average instead of maximum.

From Figure 3.4 it is clear that that average versus maximum effects the WN score while the type of partition has only a small impact on the WN score. From this figure, it is concluded that it does not matter which partition type is use, so from now on random partitions will be used. It has also been concluded that there is not enough evidence to select maximum over average or vice versa. This discussion will continue in the results section.

3.4.2 Choosing the Number of Elements in Each Partition

It is easiest to see the effects of the number of partitions used in WN when looking at only two data points. With more than two data points, it is difficult to determine the contribution of each pair in the production of WN score.

In Figure 3.5, there are two functions where the number of partitions is varied. The target function and data points are consistent through all cases. The target function is a small section of $f(x) = x^2 + 1$. The evolved function is an oscillatory function on the line formed by the two data points with varying frequency. When considering the magnitude of the score, remember that the change in x is only 3/10 for the entire x-axis compare with 10 for the y-axis. Note that the these functions were not evolved, but are all cases expected to appear during evolution and WN should be able to detect them. The ideal WN score (discussed in section 3.3) is included in these plots to determine if the partitioned method of computing WN score is sufficiently accurate.

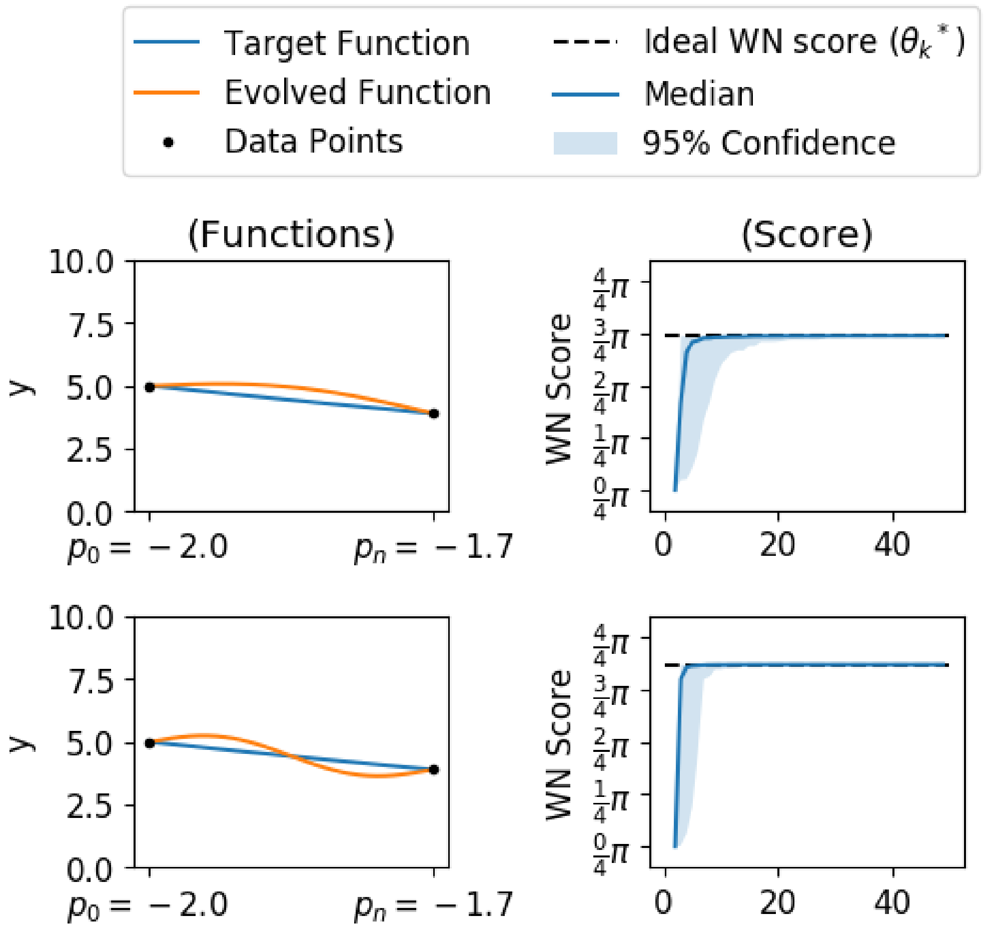

Figure 3.5: The (Function) column contains the functions on which the WN score was recorded in the (Score) column. In (Score), the WN score was computed 100 times with varying partitions. The ideal WN score (right column) is determined according to Equation 3.15.

The plots in Figure 3.6 are similar to the previous figure. This time, the amplitude is varied. Another change in frequency can be seen by comparing the last plot of Figure 3.5 to the first plot of 3.6.

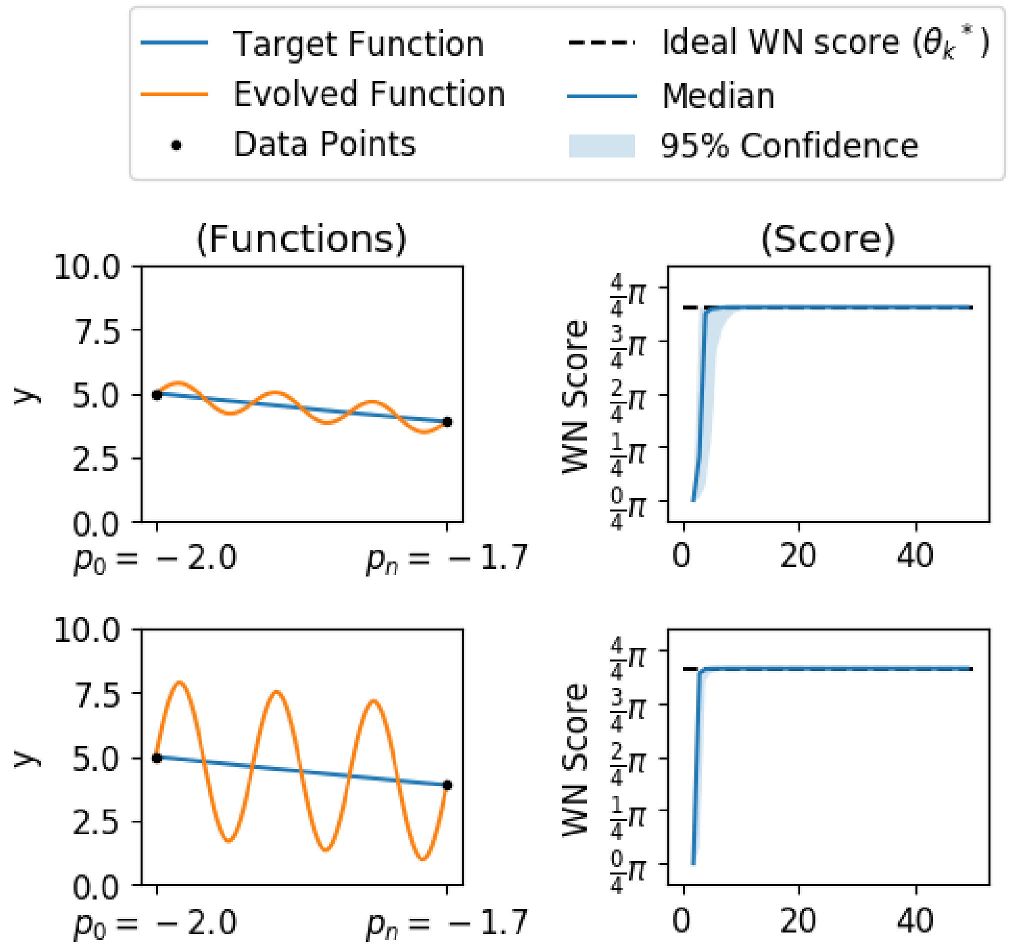

Figure 3.6: The (Function) column contains the functions on which the WN score was recorded in the (Score) column. In (Score), the WN score was computed 100 times with varying partitions.

In Figure 3.5, there are a few patterns. As the frequency of the oscillatory function increases, the evolved function rises more quickly which results in a higher WN score. The same is true of the increase in amplitude shown on the bottom row. All plots in the (Score) column flatten out quickly. This means that WN score is robust to the number of partition points. The ideal WN score is achieved by the partitioned approach with only a few partitions. This is good news since the ideal WN score can only be computed for a twice differentiable evolved function. This means, that the partitioned method of computing WN score is sufficiently accurate to justify its use.

Figure 3.7 shows one of the hardest types of overfitting to detect. This is a vertical asymptote in between the data points. This evolved function is a rational function with a denominator of $x^9 + 1.8^9$. A higher degree would make the spike narrower and harder to detect while a lower degree would make the spike wider and easier to detect. The ideal WN score is not included in this figure because the evolved function has a discontinuity and so the ideal WN score cannot be calculated.

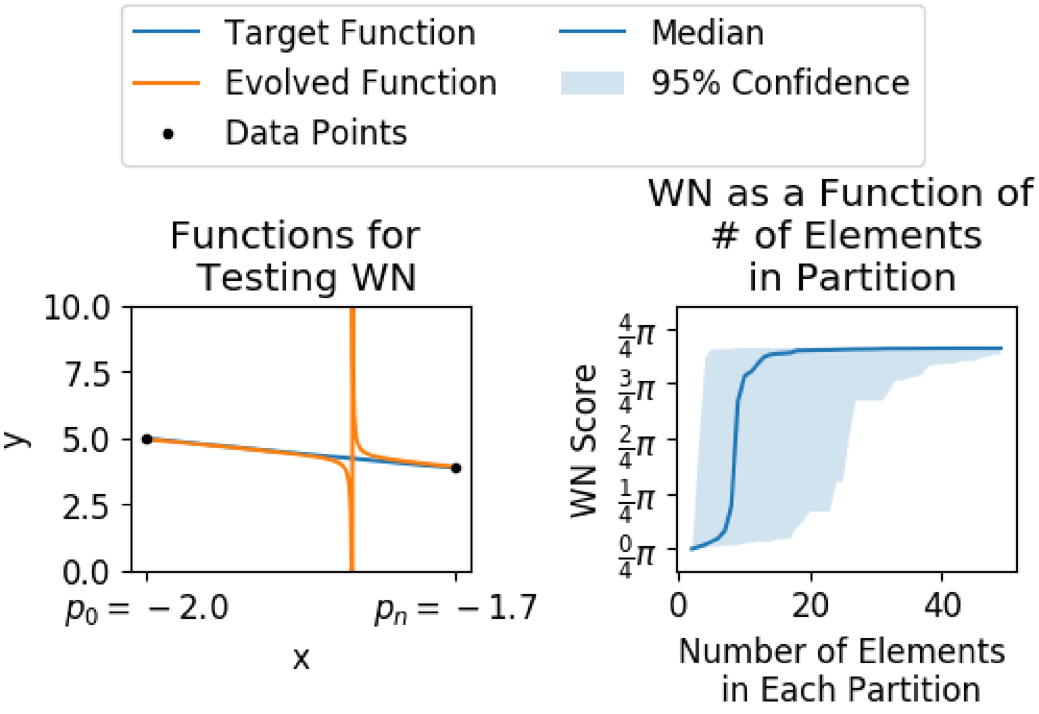

Figure 3.7: The plot on the left is the functions on which the WN score was measured for the plot on the right. The plot on the right is composed of 100 iterations of WN score on the function. The shaded area is the 95% confidence interval.

Since larger values make the algorithm more computationally expensive, the number of partitions points used should be as low as possible while still providing accurate measurements. With this in mind, we will use 10 and 25 as the number of elements in each partition. In the rest of this thesis, 10 elements per partition and 25 elements per partition will be used. Neither 10 nor 25 elements per partition is able to get a consistent score for Figure 3.7, which has a vertical asymptote. Even if WN score is not consistently measured, WN may find well fit solutions. The Results section has further comparison between this two parameter values.

Chapter 4

Experimental Setup

This section discusses how to use WN in symbolic regression and how to compare it fairly with existing algorithms. The Results section uses the outline provided by this section to produce its results.

AFPO [28] is the control case while evaluating the use of Worst Neighbors (WN). Recall that WN consists of error as the first fitness objective and WN score as the second fitness objective. Similarly, AFPO consists of error as the first fitness objective and age as the second fitness objective. This is a fair comparison because both algorithms have the same number of fitness objectives and only one of the fitness objectives is different between the two algorithms.

Since WN is designed to reduce overfitting, testing error between the two methods will be compared. Each experiment consists of 100 runs of AFPO and 100 runs of WN with initially the same set of randomly generated functions. For the one dimensional case, target functions from Table 4.1, which can also be found in [19], are used.

Table 4.1: Subset of Target Functions from [19] Used

Name	Target Function	Input Interval
Keijzer-1	$f(x) = 0.3x\sin(2\pi x)$	$[-1, 1]$
Keijzer-2	$f(x) = 0.3x\sin(2\pi x)$	$[-2, 2]$
Koza-1	$f(x) = x^4 + x^3 + x^2 + x$	$[-1, 1]$
Koza-2	$f(x) = x^5 - 2x^3 + x$	$[-1, 1]$
Koza-3	$f(x) = x^6 - 2x^4 + x^2$	$[-1, 1]$
Nguyen-3	$f(x) = x^5 + x^4 + x^3 + x^2 + x$	$[-1, 1]$
Nguyen-4	$f(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x$	$[-1, 1]$
Nguyen-5	$f(x) = \sin(x^2)\cos(x) - 1$	$[-1, 1]$
Nguyen-6	$f(x) = \sin(x) + \sin(x + x^2)$	$[-1, 1]$
Nguyen-8	$f(x) = \sqrt{x}$	$[0, 4]$

For each target function, four datasets where created. Before evolving solutions the training dataset and the validation dataset are created with 20 data points a piece in the interval given by the table (except for Keijzer-2, which has 40 data points). Once the solutions are evolved, the testing data is created, which is broken into an interpolation dataset and an extrapolation dataset. In the interpolated dataset, all the x-values are in the same interval specified in the table. In the extrapolation dataset, all the x-values are outside the given interval but within an interval twice as wide with the same center. In other words, say the interval $[a, b]$ is given for the training, validation, and interpolation datasets, then $[c - \ell, a) \cup (b, c + \ell]$ where $c = (a + b)/2$ and $\ell = b - a$ is the interval for the extrapolation dataset. Since Nguyen-8's domain is non-negative real numbers, its extrapolation dataset will be taken from only $(b, c + \ell]$.

The testing dataset is split into two datasets to determine the ability of WN to achieve low error on unseen data near the training data (interpolation dataset) and also its ability to produce solutions with low error on data outside the original training interval (extrapolation dataset). Both of these measurements will indicate overfitting if they are high. Error on the interpolation dataset is expected to be lower than error on the extrapolation dataset. This is because the interpolation dataset is closer to the training data than the extrapolation dataset. A high error on the extrapolation dataset indicates overfitting because it shows that the solution has not completely learned to underlying relationship between the independent and dependent variables. The best possible result would be to produce a solution that has the same low error on both datasets.

Noise is added to the outputs of each dataset. The amount of noise is determined by the range of the dataset. The amount of noise at each data point is pulled from a normal distribution with mean of 0 and standard deviation of 10% of the range.

To compare AFPO and WN, pick the best individual from each run. The best individual is the one, in the final generation, with the lowest error on the validation dataset. The best individuals from AFPO are compared to the best individuals of WN with the Mann-Whitney U test (also referred to as the Wilcoxon rank-sum test) [17]. The null hypothesis for this test is that there is no difference between the distributions X and Y. The one-sided alternative hypothesis is that distribution X is stochastically smaller than distribution Y. In the Mann-Whitney U test, the distributions need not be normal, but the test does assume that X and Y are independent and ordinal and their cumulative distribution functions are continuous. Given samples $x_1, \cdots, x_n$

from an experiment and samples $y_1, \cdots y_n$ from another experiment, the Mann-Whitney U test determines U, which is the number of times that a y_i proceeds an x_i when all samples are ranked. If all the y_i's are less than all the x_i's then then $U = mn$. Let T be the sum of the ranks of all y_i. Since the sum of the lowest possible ranks for the y_i's is $m(m+1)/2$, the general equation is

$$U = mn + \frac{m(m+1)}{2} - T.$$

The Mann-Whitney U test uses U to compute a p-value to determine if distribution X is stochastically greater than distribution Y.

4.1 Symbolic Regression with Worst Neighbors

AFPO and WN both have two fitness objectives. The error (absolute average) objective is the same for both algorithms. The second objective for WN is the WN score, which is described in Section 2.2. In both algorithms, a generation requires the following steps – except that in AFPO there is the added step (between step 2.2 and step 2.3) of creating a new random individual.

1. Evaluate the population on both fitness objectives.
2. Create the next generation by completing the next three substeps.
   - 2.1. Compute the non-dominated front.
   - 2.2. Remove all dominated individuals from the population.
   - 2.3. Create new individuals by mutating the individuals on the front until the population is full.

In the case of WN, the front grows in size quickly due to the WN score. This occurs because there are many low but slightly different WN scores in the population. It is possible that these slight differences occur due to changes in the partition used during calculation. If this is the case, front growth can be decreased by computing one set of partitions (one for each interval $[t_{k-1}, t_k]$) to be used by all individuals in the population at the beginning of each generation. In this work, a new partition is created for each individual. Because the front grows quickly, it is possible for the non-dominated front to become the entire population, at which point no further evolution can occur. To avoid this, a method of front reduction called binning is included between steps 2.1 and 2.2. In this method, find the individual with the highest WN score and the individual with the lowest WN score and make n bins between these two values. The parameter n is chosen to be the square roots of the population size (truncated if necessary), which makes the median front size similar to that of AFPO.

Once these bins are created, an individual is selected randomly from each bin to remain on the front. After completing this method, there will be at most n individuals on the front. This method helps to keep the front diverse in terms of the WN score. The individuals to remain on the front are not picked by lowest error because minimizing only error leads to overfitting. By picking the individuals at random, less reliance is placed on the error objective.

Binning could be performed by making n permanent bins of the interval $[0, \pi]$; however, a front whose range is not all of $[0, \pi]$, would likely lose most of its individuals. The proposed method of binning is gentler, allowing at least two individuals to remain on the front – assuming two or more individuals exist on the front before binning. It has a bin size that makes it more likely for other individuals to remain on the front.

Figure 4.1 shows the before and after images of a non-dominated front. Before binning, there tend to be many individuals with a WN score near zero. These low scores are likely underfit solutions.

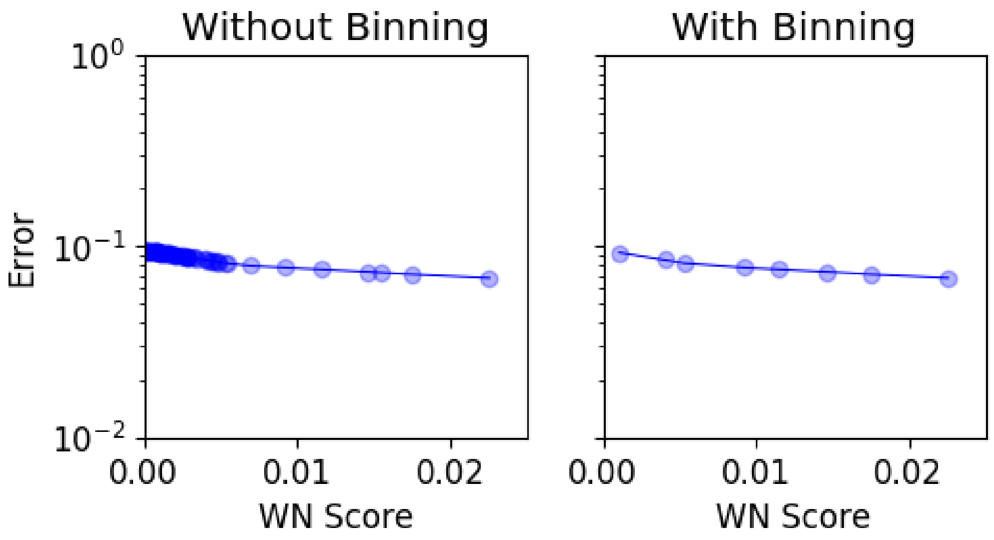

Figure 4.1: This is an example of the densely populated non-dominated front (left) and one possible reduction (using binning) of that non-dominated front (right). The dominated solutions are not drawn.

Figure 4.2 shows the size of the non-dominated front (on the 1000^{th} generation) for both AFPO and WN each with 100 repetitions.

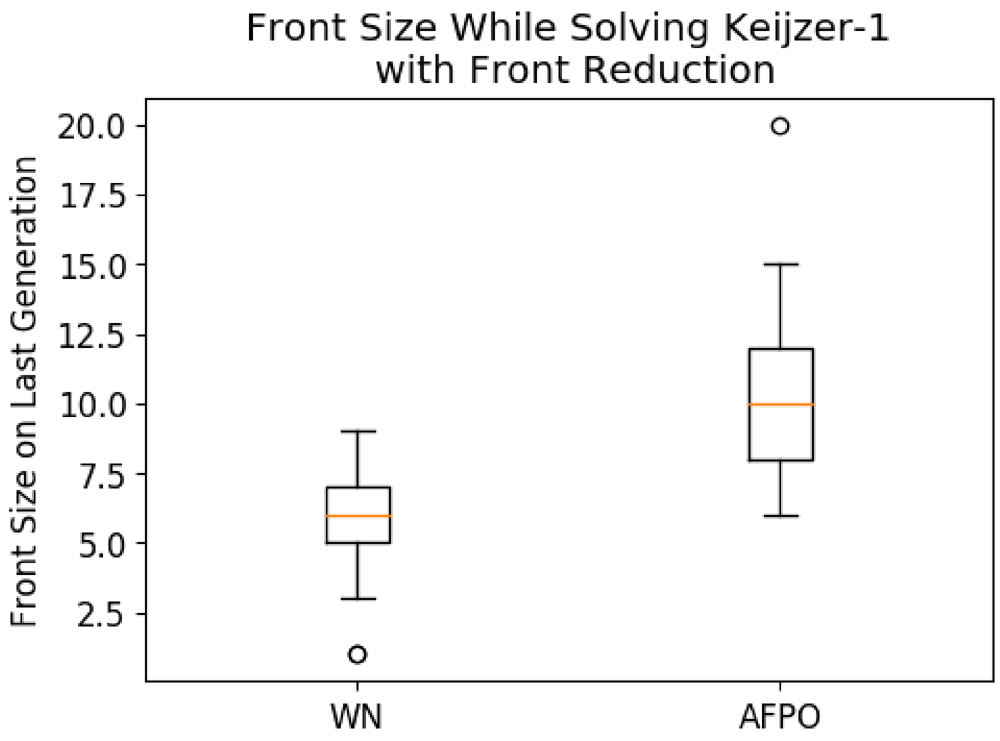

Figure 4.2: The front size on the final generation of evolution. This is a typical distribution for both AFPO and WN in all experiments.

Table 4.2 lists all parameters that are consistent through all experiments.

Table 4.2: Parameter Summary

Parameter	**Value**
Population Size	100
Number of Generations	1000
Tree Initialization	ramped half-and-half
Initial Max Tree Depth	6
Max Tree Depth	17
Number of Runs	100
Max Front Size (WN only)	$\lfloor\sqrt{\text{\# of Gens}}\rfloor = 10$
Function Set	$\{+, \times, -, /, \sin, \cos\}$
Terminal Set	$\{x, c\}$ where $c \in [-1000, 1000]$

Constants

Previously, the terminal set of $T = \{x, y, 1, 2, 3\}$ was suggested. Usually, however, the constants which will be needed are unknown when selecting the terminal set. So, instead let $T = \{x, y, c\}$ where if c is selected for a node, a random number is taken from $[-1000, 1000]$. This number is now fixed and will only change if effected by mutation. This is better than setting $T = \{x, y\} \cup [-1000, 1000]$ because it encourages the use of x and y in the evolved function. More ways of handle constants are discussed in [24].

4.2 Extra Data Points for WN

Since the WN score can be calculated without using output data, it is possible to include more data points (without output values). This is done by finding the average difference between adjacent data points and using this for a step size to create the new data. The hope is that including extra data points will improve extrapolated test error.

For example, say our target function has training data (consisting of 20 points) in the interval $[-1, 1]$, like most functions in Table 4.1. And so, the interval for the extrapolated data is $[-2, -1) \cup (1, 2]$. Let X be the set of all x-values in the training data. Then, for each $x \in X$, we have $x \in [-1, 1]$. The average difference between adjacent data points $s = (\max X - \min X)/19$ since the sum of these differences equals $\max X - \min X$. The extra data points are $X_e = \{\min X - sm : m \in \mathbb{N}\} \cup \{\max X + sn : n \in \mathbb{N}\}$ where m and n remain small enough that each element of X_e is in the interval $[-2, 2]$. In other words, the elements of X_e are in the interval for the extrapolated data or possibly the interval for the training data.

Adding more data points will increase computation time, so it is important to be sure that extra data will improve WN before using it. Extra data will be using on some comparisons against AFPO. In the Results section, it is determined that using extra data improves WN.

Chapter 5

Results

In this chapter, target functions listed in Table 4.1 are used to compare WN with AFPO. For each target function, Worst Neighbors is run with partitions of 10 elements and 25 elements and with and without extra data points. Remember that these extra data points are only x-values so the error objective remains the same and only the WN score changes. WN is compared with AFPO in interpolated and extrapolated error of the best individual in the final generation of each run – the best individual is the one with the lowest error on the validation dataset. This comparison is done with the Mann-Whitney U Test, which is described in the previous chapter.

5.1 Results for Keijzer-1

In Figure 5.1, the testing error for best individuals in each run is shown. The left column of box plots uses 10 elements per partitions and the right column uses 25 elements in each partition. The top row of box plots does not use any extra data, but the bottom row does use extra data. In the box plot labels, I stands for interpolated error and E stands for extrapolated error.

Note that that the only difference between the AFPO box plots is the initial equations created on generation 0.

The point of this figure is to show the effects of the parameters of WN. In all cases, the interpolated error is significantly less for WN than for AFPO. For extrapolated error, in all but the 10 elements per partition with no extra data case WN out performed AFPO. Note that there are typically a few large outliers for the interpolated box plots. When these outliers are visible on the plot, it is difficult to compare WN and AFPO. It is worth noting that although many of these comparisons have a statistically significant difference, the difference in magnitude is fairly small.

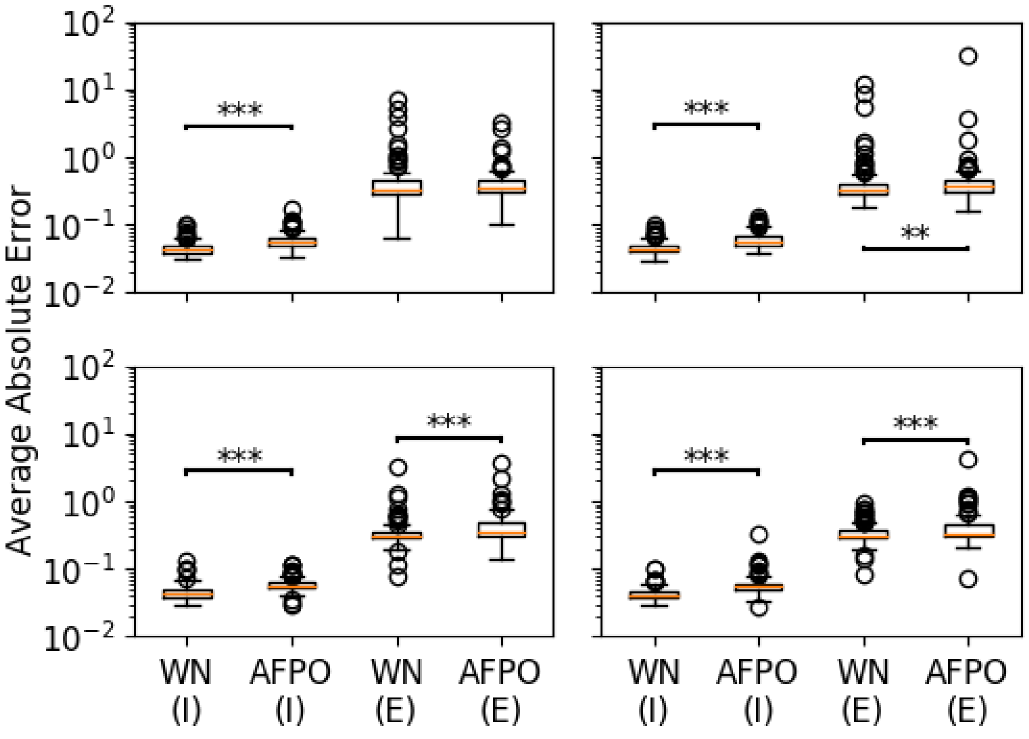

Figure 5.1: Results for Keijzer-1. The left column uses 10 elements per partition while the right column uses 25 elements per partition. The top row did not use extra data points (input only) while the bottom row did. Significance is determined using the Mann-Whitney U Test.

The next few figures are some of the best solutions whose testing error is used in the previous box plots. This will hopefully give the reader a better understanding of what sort of solutions are possible. Figures 5.2 and 5.3 shows that both AFPO and WN are able to produce well fit solutions to Keijzer-1. These solutions were picked visually and are some of the best solutions produced by AFPO and WN for Keijzer-1. Both figures have two plots. The top plot shows the solution and all types of data used. The bottom plot show the absolute difference between the target function f and the evolved function $\hat{f}$. Between the two dashed lines is the training interval. It is expected that the absolute difference is close to 0 in the interval. Since the function was not trained outside this interval, higher error is expected here. This lower plot is a helpful tool for comparing good solutions. In the following figures for example, there is a higher frequency oscillation in the APFO solution than the WN solution.

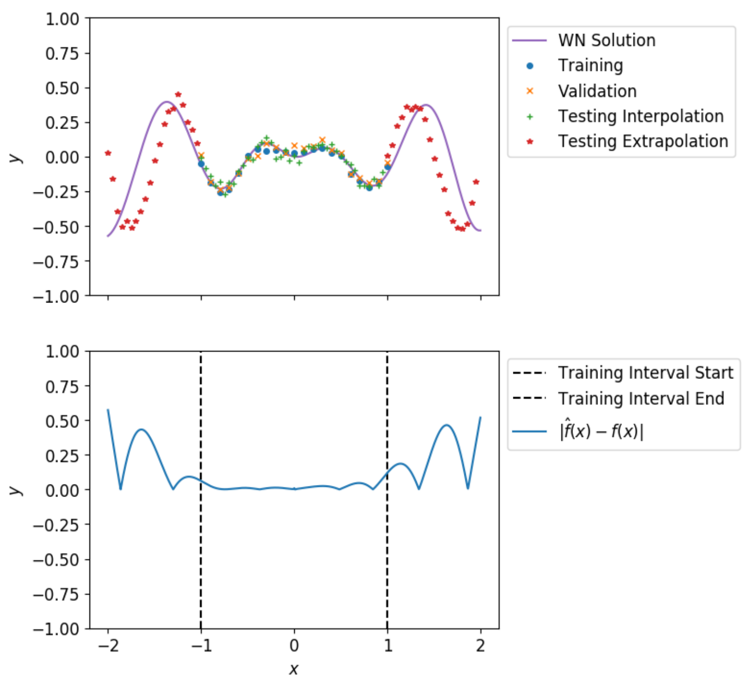

Figure 5.2: The top plot shows one of the best solutions, as determined by me, from WN. The bottom plot shows the absolute difference between the target function and the evolved function.

In Figure 5.2, the basic function behavior is appropriate. That is, the function has some oscillation and its amplitude is increasing with its distance from $x = 0$. However, the frequency of the oscillation is too small since the WN solution starts to diverge from the target function visibly in the extrapolation region. Though difficult to see in the top plot, the same frequency issue is present. It is easier to see the slight oscillations in the bottom plot.

Figure 5.3 the frequency of the AFPO solution is too large. Unlike the WN solution, this solution is not increasing in magnitude at the correct rate. This is clear by looking at the solution compared with the extrapolation data. Near $x = 0$, the WN solution has an extra turn compared with the AFPO solution. The target function has this extra turn, but due to noise and the small changes in y for this turn the AFPO solution also looks reasonable.

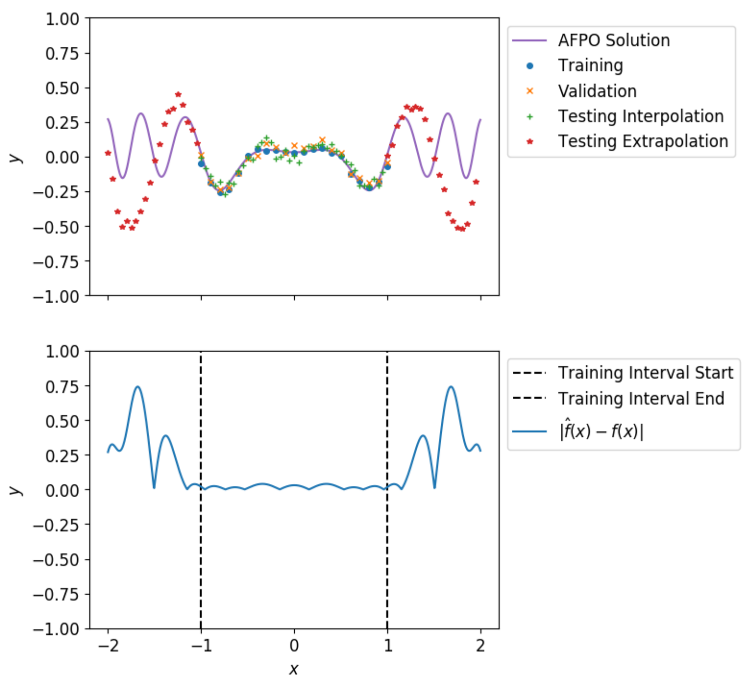

Figure 5.3: The top plot shows one of the best solutions, as determined by me, from AFPO. The bottom plot shows the absolute difference between the target function and the evolved function.

Even though WN is designed to avoid overfitting, it is still possible for the best solution in a run to be overfit. This occurs when the error objective is lower than any other solution in the population. One example of this is shown in Figure 5.4. By visually inspecting the best solutions to Keijzer-1, it becomes clear that WN is less likely to overfit than AFPO. However, this is only true on the part of the data where WN score is computed.

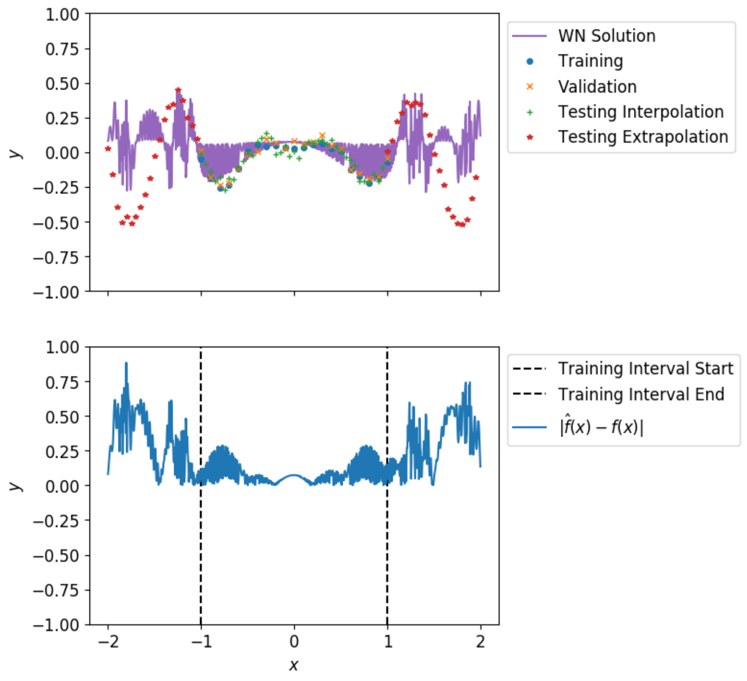

Figure 5.4: WN can still overfit.

In Figure 5.5, a WN solution is shown. On the training interval (which is also the only place where WN score was calculated), the solution has roughly the correct amount of oscillation. Outside this interval, there is a drastic increase in the frequency of oscillation. When using extra data as explained in Section 4.2, this type of solution becomes less common.

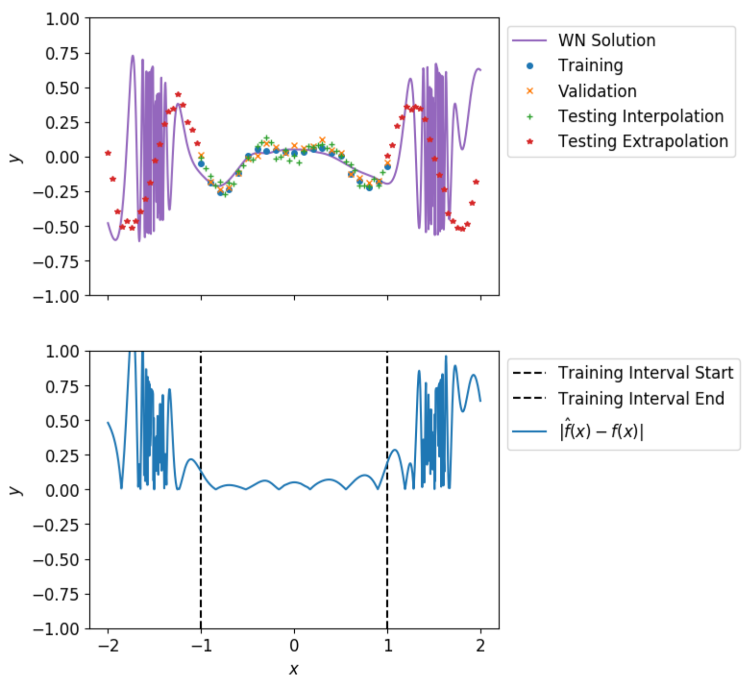

Figure 5.5: WN can produce solutions that are erratic outside the interval on which WN is computed. This is one such example.

5.2 Results for Koza-1

Koza-1 is a fourth degree polynomial from Table 4.1. Figure 5.6 shows the results for Koza-1. Just like with Keijzer-1, parameters 10 or 25 elements in each partition and with or without extra data where used in experiments. These experiments were not as successful as those for Keijzer-1, meaning there were less statistically significant results and that the difference in magnitude between distributions was smaller. WN was only able to out perform AFPO when extra data was used. Perhaps this is due to the shape of Koza-1 and the effect this has on noise. Perhaps this is due to the lack of constants in Koza-1. One possible explanation for the short box plots in interpolation error is that Koza-1 has no constants. Since GP has difficulty in constant selection, both algorithms produces solutions with more consistent interpolation error.

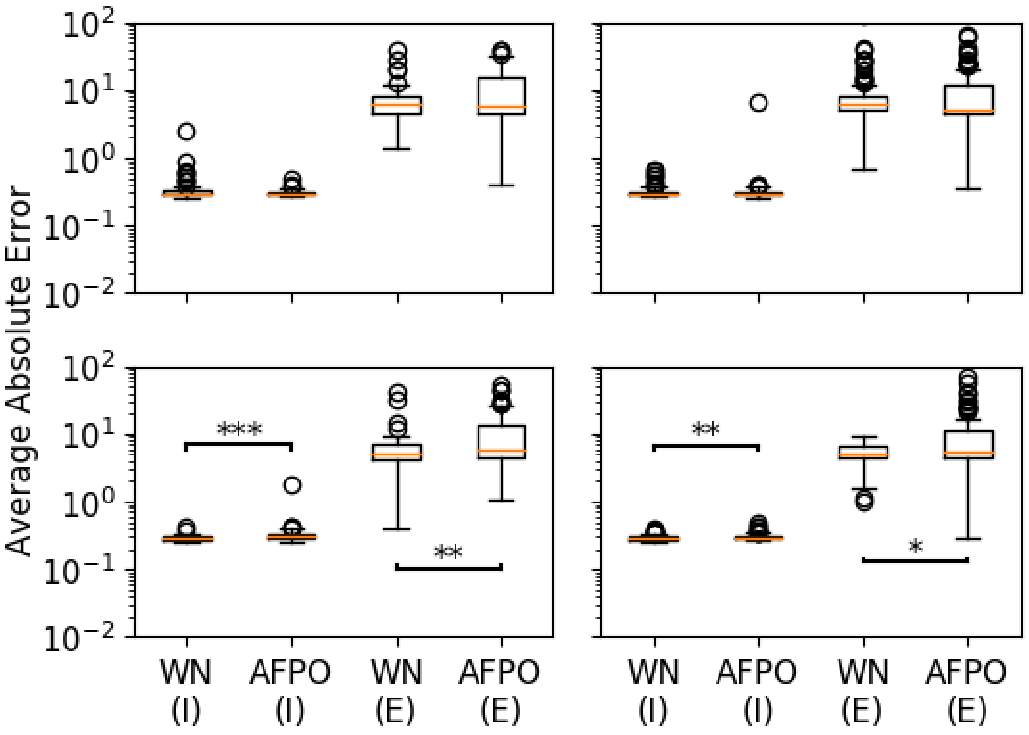

Figure 5.6: Results for Koza-1. The left column uses 10 elements per partition while the right column uses 25 elements per partition. The top row did not use extra data points (input only) while the bottom row did.

In Figure 5.6, you can see that the difference between AFPO and WN solutions is small. Figures 5.7 (WN) and 5.8 (AFPO) show typical solutions to Koza-1. On the interval for the training data, these solutions look similar as expected based on the box plots. Both these solutions are possible for the other algorithm, however, it is more typical for the spike to occur in AFPO than WN. This indicates the WN score is able to detect spikes causing the individuals that have them to not typically be on the front.

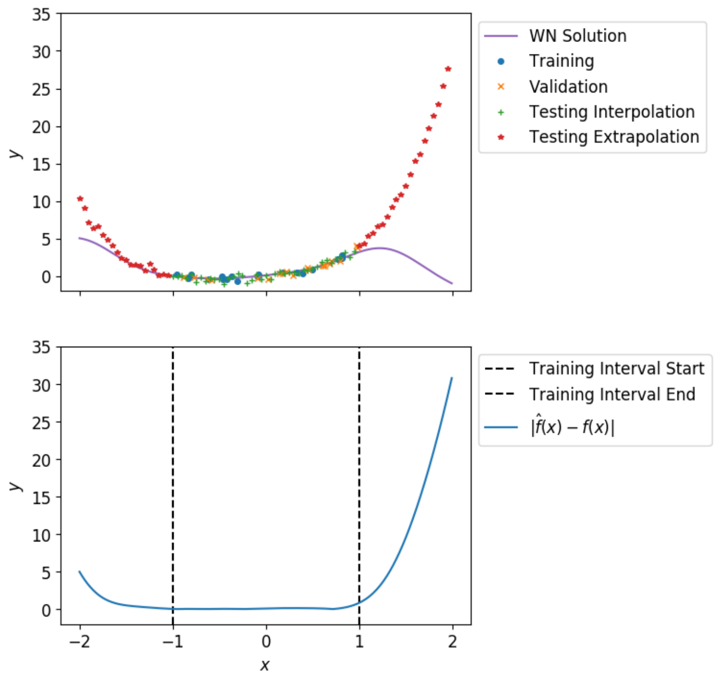

Figure 5.7: The top plot shows one of the best solutions from WN. The bottom plot shows the absolute difference between the target function and the evolved function.

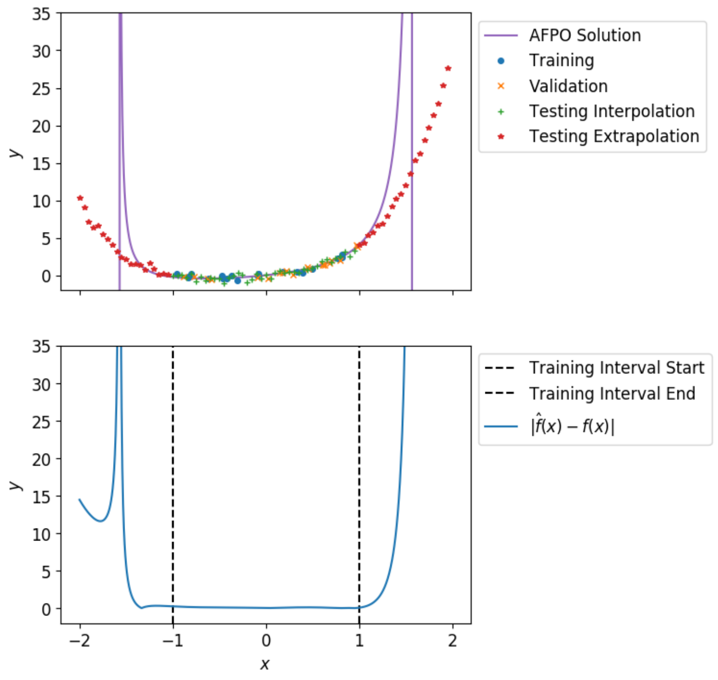

Figure 5.8: The top plot shows one of the best solutions from AFPO. The bottom plot shows the absolute difference between the target function and the evolved function.

5.3 Comparing WN and AFPO on All Functions

Since it is difficult to display 8 boxplots for each of the 10 target function to allow for easy comparison, the median error of the best individuals are located in Tables 5.1, 5.2, 5.3, and 5.4 – one table for each variation of parameters. Each table has both median interpolation error and median extrapolation error.

WN should be considered to outperform AFPO if (1) WN's interpolated error is less than (with statistical significance) AFPO's interpolated error or (2) there is no statistical difference in interpolation error, but WN achieves lower extrapolation error. This is because the primary concern is low error on the interpolated data. A solution that has decreased extrapolation error at the cost of interpolation data is not desirable.

Consistently, in all tables, WN tends to have lower extrapolated error compared with AFPO. This is interesting when extra data is not used because neither WN nor AFPO look at the evolved function in the interval for extrapolated data. Perhaps the reason for this is that discontinuities in the evolved function move as constants change. If the discontinuity appears in the interval for training data then the WN score would become higher. For high frequency oscillations, a consistent frequency is easier to construct. Thus, the oscillation would show up in both the extrapolated and interpolated regions allowing WN to discover it. Remember that this is not always the case as shown in Figure 5.5. Regardless of the reason for this it does appear that there are more statistically significant results (in extrapolated error) when using extra data than without it.

Table 5.1: Summary table of results with 10 elements in each partition and no extra data.

	Interpolated		Extrapolated	
Function	WN	AFPO	WN	AFPO
Koza-1	2.956e-01	**2.928e-01**	6.343e+00	**5.875e+00**
Koza-2	8.318e-02	**6.734e-02*** **	**4.535e+00*** **	4.698e+00
Koza-3	5.646e-02	**3.380e-02*** **	**8.066e+00**	8.068e+00
Nguyen-3	4.378e-01	**4.060e-01**	**1.380e+01*** **	1.560e+01
Nguyen-4	4.908e-01	**4.834e-01***	2.526e+01	**2.497e+01***
Nguyen-5	5.266e-02	**5.127e-02****	8.055e-01	**7.230e-01****
Nguyen-6	**3.139e-01****	3.272e-01	**1.643e+00**	1.658e+00
Nguyen-8	**1.976e-01***	2.008e-01	4.610e-01	**4.572e-01**
Keijzer-1	**4.202e-02*** **	5.772e-02	**3.198e-01**	3.444e-01
Keijzer-2	**1.645e-01***	1.767e-01	**5.747e-01***	6.018e-01

Table 5.2: Summary table of results with 10 elements in each partition and extra data.

	Interpolated		Extrapolated	
Function	WN	AFPO	WN	AFPO
Koza-1	**2.870e-01*****	3.056e-01	**5.138e+00****	5.856e+00
Koza-2	7.704e-02	**6.708e-02*****	**4.521e+00*****	4.654e+00
Koza-3	5.645e-02	**3.603e-02*****	**8.066e+00*****	8.091e+00
Nguyen-3	3.856e-01	**3.829e-01**	**1.338e+01*****	1.511e+01
Nguyen-4	**4.776e-01**	4.893e-01	**2.150e+01*****	2.769e+01
Nguyen-5	5.260e-02	**5.117e-02***	8.122e-01	**7.400e-01***
Nguyen-6	**3.176e-01*****	3.323e-01	**1.629e+00**	1.692e+00
Nguyen-8	**1.985e-01***	2.014e-01	**3.908e-01**	4.797e-01
Keijzer-1	**4.354e-02*****	5.761e-02	**3.057e-01*****	3.515e-01
Keijzer-2	2.046e-01	**1.756e-01**	**5.743e-01****	5.927e-01

Table 5.3: Summary table of results with 25 elements in each partition and no extra data.

	Interpolated		Extrapolated	
Function	WN	AFPO	WN	AFPO
Koza-1	**2.926e-01**	2.972e-01	6.283e+00	**5.240e+00**
Koza-2	8.180e-02	**6.505e-02*****	**4.540e+00*****	4.642e+00
Koza-3	5.633e-02	**3.204e-02*****	**8.066e+00*****	8.115e+00
Nguyen-3	4.147e-01	**3.906e-01**	**1.375e+01*****	1.489e+01
Nguyen-4	4.845e-01	**4.833e-01***	**2.498e+01***	2.608e+01
Nguyen-5	5.369e-02	**5.248e-02**	7.692e-01	**6.723e-01**
Nguyen-6	**3.171e-01**	3.260e-01	**1.570e+00**	1.744e+00
Nguyen-8	**1.993e-01**	1.997e-01	3.955e-01	**3.838e-01**
Keijzer-1	**4.437e-02*****	5.705e-02	**3.273e-01****	3.735e-01
Keijzer-2	**1.597e-01**	1.699e-01	**5.765e-01**	5.979e-01

Table 5.4: Summary table of results with 25 elements in each partition and extra data.

	Interpolated		Extrapolated	
Function	WN	AFPO	WN	AFPO
Koza-1	**2.876e-01****	2.969e-01	**5.263e+00***	5.479e+00
Koza-2	8.107e-02	**7.042e-02*****	**4.527e+00*****	4.652e+00
Koza-3	5.527e-02	**3.884e-02***	**8.065e+00*****	8.084e+00
Nguyen-3	**3.680e-01**	3.703e-01	**1.318e+01*****	1.446e+01
Nguyen-4	**4.734e-01**	4.743e-01	**2.187e+01*****	2.682e+01
Nguyen-5	5.108e-02	**4.975e-02**	7.713e-01	**7.010e-01**
Nguyen-6	**3.209e-01****	3.377e-01	**1.592e+00***	1.753e+00
Nguyen-8	**1.998e-01**	2.003e-01	**3.729e-01***	4.840e-01
Keijzer-1	**4.152e-02*****	5.725e-02	**3.014e-01*****	3.385e-01
Keijzer-2	1.772e-01	**1.699e-01**	**5.743e-01*****	5.934e-01

5.4 Comparing Average and Maximum

In Section 2.3.1, the use of average versus max in the computation of WN score was discussed . The discussion is continued with Figure 5.9, which compares these two possibilities by running the four variations of WN with average and with max. In interpolation error, the WN using max outperforms WN using average when using 10 elements per partition and no extra data. This happens again when using 25 elements per partition with extra data. There are no other statistically significant differences.

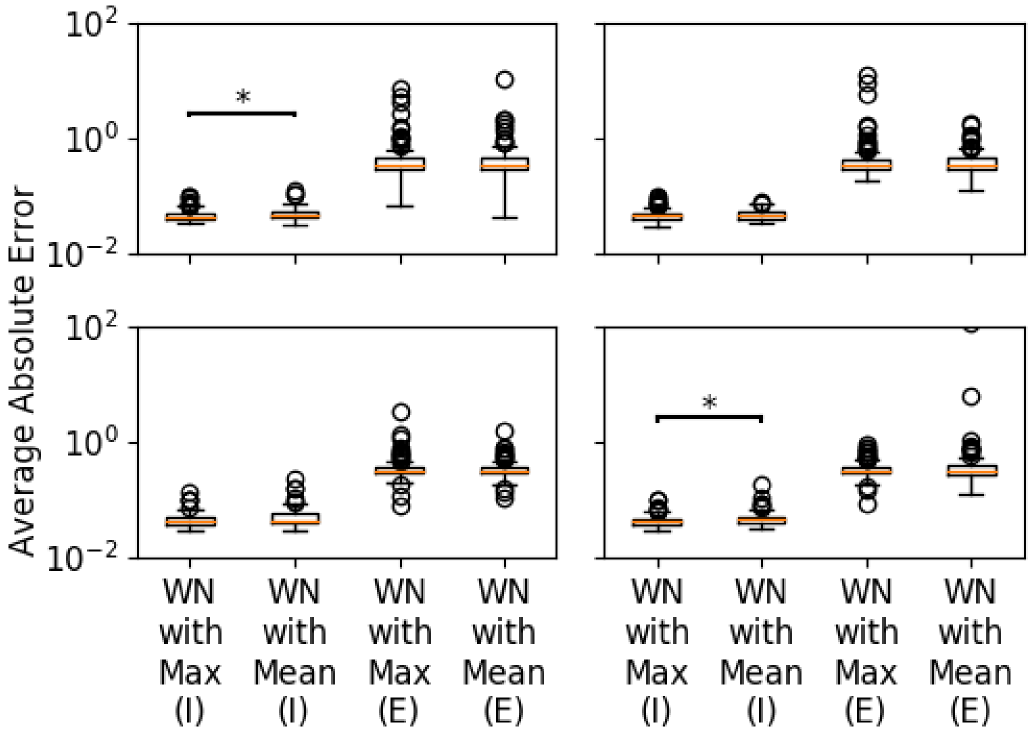

Figure 5.9: Comparison on Keijzer-1. The left column of plots use 10 elements per partition while the right column uses 25 elements per partition. The top row did not use extra data points (input only) while the bottom row did.

Table 5.5 shows these results for a few more target functions. These runs used 25 elements per partition and extra data. The only statistically significant differences are in favor of max. Thus, max will be used rather than average.

Table 5.5: Summary table comparing the use of max and average in the computation of WN score.

	Interpolated		Extrapolated	
Function	Max	Average	Max	Average
Keijzer-1	**4.124e-02***	4.528e-02	**3.006e-01**	3.095e-01
Koza-1	7.322e-01	**7.250e-01**	3.236e+00	**3.147e+00**
Nguyen-3	6.975e-01	**6.881e-01**	2.209e+00	**2.161e+00**
Nguyen-4	8.641e-01	**8.537e-01**	**5.081e+00**	5.739e+00
Nguyen-6	8.618e-01	**8.517e-01**	**2.157e+00***	2.255e+00

5.5 The Effect of Noise

Does WN produce better solutions compared with AFPO as the amount of noise on the target function increases? To see if this is true, both algorithms were run on the target functions Nguyen-5 and Koza-3 with 6 different standard deviations for the noise. The results for Nugyen-5 are in Table 5.6 and the results using Koza-3 are in Table 5.7. As before the noise was chosen by finding the range of the training dataset and taking a percent of this value to be used as the standard deviation. For each row in these tables, the dataset was recreated (likely with different x-values) and thus a different range. Even though these rows represent a change in percent, from 5% to 30% by increments of 5, the difference between the standard deviation in each row is inconsistent because the range also changes. Like the tables in the previous section, the displayed values are the median of 100 runs. Since it has been determined that the best parameters for WN are 25 elements per partition and extra data, this test uses only those parameters.

Table 5.6: *Summary table of results with different amounts of noise on Nguyen-5.*

	Interpolated		Extrapolated	
Noise	WN	AFPO	WN	AFPO
2.28e-02	**2.867e-02**	3.152e-02	6.227e-01	**4.778e-01**
4.37e-02	5.108e-02	**4.975e-02**	7.713e-01	**7.010e-01**
6.84e-02	8.866e-02	**8.358e-02*****	**7.678e-01**	7.777e-01
8.98e-02	1.235e-01	**1.223e-01***	**6.537e-01***	7.680e-01
1.13e-01	2.075e-01	**1.928e-01*****	**4.996e-01****	7.794e-01
1.37e-01	1.837e-01	**1.819e-01**	**6.282e-01*****	8.087e-01

Table 5.7: *Summary table of results with different amounts of noise on Koza-3.*

	Interpolated		Extrapolated	
Noise	WN	AFPO	WN	AFPO
7.19e-03	4.762e-02	**2.301e-02*****	**8.027e+00*****	8.094e+00
1.35e-02	5.527e-02	**3.884e-02***	**8.065e+00*****	8.084e+00
2.19e-02	6.312e-02	**5.349e-02*****	**7.999e+00*****	8.007e+00
2.94e-02	**5.464e-02***	6.003e-02	**8.061e+00*****	8.102e+00
3.70e-02	**7.319e-02**	7.429e-02	**8.041e+00*****	8.134e+00
4.10e-02	6.996e-02	**6.209e-02****	**8.002e+00*****	8.102e+00

There does not appear to a correlation between WN success against AFPO and the amount of noise applied to the target function. AFPO has more occurrences of significance in its favor for both of these target functions. This is reasonable since these functions where two of the four functions where AFPO outperformed WN in 5.4, which used 25 elements per partition and extra data. Even so, based on this data, it is doubtful that an increase in noise causes an increase in WN performance compared with AFPO.

5.6 Function Complexity

High complexity in a function can indicate an overfit solution. Thus, it is of interest to know the relative complexity of functions created by AFPO and WN. Table 5.8 shows the median complexity (number of nodes) of solutions for all 10 target functions. Again, WN is using 25 elements per partition and extra data on all entries in the table. In most of these target functions, AFPO has created less complex solutions than WN. This may be due to the fact that AFPO involves injecting new (and small) individuals into the population at every generation. This is likely contributing to AFPO's ability to keep the complexity of its solutions low. It is possible for these functions to be structured in such a way that there are more nodes than necessary. In an attempt to determine if this is the case, some simple simplification where made to the solutions before counting their nodes. The simplification where $g(x) - g(x) = 0$, $g(x) - 0 = 0$, $g(x) * 1 = g(x)$, $g(x) * 0 = 0$, and $g(x) + 0 = g(x)$ where $g(x)$ is a subtree. Also, at each node its output value was recorded for all inputs in the training data. If all values at a specific node were the same, the subtree was replace with a single constant node equal to that value. There are many more possible simplification that can be applied, so the median number of nodes displayed in Table 5.8 may still be larger than necessary. For Koza-3 WN found an underfit solution based on Table 5.8 where the median is 1 thus it must have found a constant function to the best solution in more than half the runs. To illustrate the effect of this simple form of simplification, Table 5.9 shows the medians before simplification.

Table 5.8: Summary table of complexity results with 25 elements in each partition and extra data with simplification.

Function	WN	AFPO
Koza-1	90.5	**46.0*****
Koza-2	78.0	**39.0*****
Koza-3	**1.0*****	31.0
Nguyen-3	67.5	**44.5*****
Nguyen-4	96.0	**77.5*****
Nguyen-5	68.0	**34.5*****
Nguyen-6	58.0	**29.0*****
Nguyen-8	67.0	**22.0*****
Keijzer-1	71.0	**65.5**
Keijzer-2	**35.0*****	69.0 5

Table 5.9: Summary table of complexity results with 25 elements in each partition and extra data without simplification.

Function	WN	AFPO
Koza-1	103.0	**64.0*****
Koza-2	83.0	**49.5*****
Koza-3	**1.0*****	44.5
Nguyen-3	76.0	**57.5*****
Nguyen-4	109.0	**96.0***
Nguyen-5	78.0	**44.5*****
Nguyen-6	66.0	**35.5****
Nguyen-8	73.5	**33.0*****
Keijzer-1	**80.0**	91.5
Keijzer-2	**35.0*****	86.0

Chapter 6

Conclusions

In this thesis, a fitness objective which attempts to measure the amount of overfitting of a function by measuring differences in approximate derivatives in the form of angles is presented. This fitness objective is called Worst Neighbors score. It is used with average absolute error. WN and AFPO are compared to determine which algorithm achieved lower error on unseen data. There are a few parameters used in compute WN score that are considered.

The number of elements in a partition is one parameter. This parameter effects the accuracy of the measurement; however, Figures 3.5-3.7 shows that WN score is robust to the number of partition points. Since WN score does still score all functions at 0 if using only 2 partition elements it important not to pick the lowest possible number of partitions. As more elements are added the partition, computation time increases. Using Figures 3.5-3.7, values of 10 and 25 for this parameter where determined to be worth trying.

Another parameter is the type of partition used. Uniform partitions should be avoided. Such partitions are uniform in two ways: uniform spacing between each element and a consistent partition for all generations. Uniform partitions are undesirable because a function with a period equal to the step size of the partition would receive an inaccurate score. More importantly, the partition should change every generation because no partition is perfect. Using different partitions each time WN score is computed gives WN the chance to catch a mistake in a later generation. Previously two types of partitions where discussed: random and semi-uniform partitions. Using Figure 3.4, little difference between the two partitions was detectable, so either type of partition should produce similar results. In this thesis, random partitions where used.

In the computation of WN score, it is of interest to know whether it is better to use the maximum of all the subintervals or the average. This has been explored in the Results section (Table 5.5) and Figure 3.4. The conclusion was that max should be used.

Another consideration is whether or not to use extra data. Extra data consists of input values placed at the ends of the training dataset. Since there are no outputs, this is no use to the error objective. Instead, WN score includes extra data in its calculation. The idea is to help avoid discontinuities and over oscillation outside the training data. Like with the parameter for number of elements in a partition, the downside of using this is the computational expense.

From Tables 5.1, 5.2, 5.3, and 5.4, WN using the parameter set with 25 elements per partition and extra data achieves more statistically significant results for WN than with any of the other parameter sets. However, for some of the target functions like Koza-2 and Koza-3, AFPO outperformed WN. Despite the fact that many statistically significant result where found, there is only a small difference in error. Therefore, it is concluded that the best version of WN uses 25 elements in each partition and extra data.

Table 5.4 shows that WN improves on extrapolated error when compared with AFPO. In 9 out of 10 target functions, WN has significantly lower extrapolated error. On the 10^{th} target function there is no statistically significant difference between WN and AFPO.

A couple target functions were given repeated tests with varying amounts of noise. The hypothesis was that WN would beat AFPO more consistently with higher levels of noise. No evidence was found to support this hypothesis. In fact, AFPO outperformed WN in most of the tests associated with this hypothesis; however, it is important to remember that the target functions used were those on which AFPO had already outperformed WN at the standard amount of noise. Thus, it does not appear that WN or AFPO benefit from increased amounts of noise.

WN and AFPO were also compared on complexity of solutions (number of nodes). For almost all target functions, AFPO produced simpler expressions; however, it is not clear if the solutions where simplified fully.

6.1 Issues Not Addressed

The computational effort required to run WN is much more than that required to run AFPO. Thus, WN should only be allowed to run long enough that both algorithms achieve the same computation time. The fairness of this form of comparison should be considered carefully before making this adjustment. Since overfitting is more likely to occur later in the evolutionary run, WN could be given an unfair advantage unintentionally. If WN is terminated early to perform this comparison, the comparative performance of WN at each stage in evolution should be considered.

AFPO was chosen to be compared with WN because AFPO is a popular and successful algorithm [3, 2, 16, 31]; however, AFPO was not designed specifically to solve the overfitting problem. In the future, it makes sense to compare WN with other methods designed to reduce overfitting. One reason AFPO was used is that it has two fitness objective just like WN. This makes a fair comparison easy. In [1], derivatives are used in multi-objective symbolic regression. This paper is the most similar to this thesis and should be the next algorithm to compare WN against. The algorithms differs because [1] compares approximate derivatives of the target function/dataset with derivatives of the evolved function while WN only uses approximate derivatives (average rates of change) on the evolved function.

6.2 Limitation of Methods

This work uses ephemeral constants where the range of possible constants are specified before evolution starts. Ephemeral constants tend to cause trees to fill with constants as GP works to determine the correct constants to use. By using a more effective method for constant creation such as digit concatenation [7] or co-evolution of coefficients in GP by a variant of Differential Evolution [21], WN (and AFPO) can be allowed to find solutions more easily. Another option is to remove any constants from the terminal set and only use target functions that do not require the use of constants.

6.3 Future Work

Currently, WN has only been attempted on one dimensional problems. Working with multidimensional problems will require more work. WN will involve choosing nearby points as neighbors, which introduces the parameter of number of neighbors. Additionally, the WN score will need to be computed on a plane formed by the neighbors and the output direction. In this multidimensional case, WN will require many more data points. Since the WN score will need to be computed in as many directions as possible. Otherwise, the overfit parts of a function can hide in the directions where WN was not computed.

While it good to use these benchmark problems for our initial work with WN, it is also important to see how well WN performs on real world data. This will be an important test because the noise will already be applied and the correct solution will be unknown. Both will be informative of the usefulness of WN.

Fully understanding the criteria under which WN outperforms AFPO is important. If the criteria are known, it would be easy to determine how to start the problem. One potential criterion has already been explored: the amount of noise on the target function. This potential indicator has not shown promise.

Since the individuals in symbolic regression are functions it would be valuable to use many of the tools typically associated with functions. One such tool is the derivative. This thesis has avoided the problem of differentiating a tree by approximating the derivative with average rates of change. In [23, 1], the derivative of a tree has been explored by creating an analytic quotient to ensure differentiability of trees. The trees are differentiated recursively by using the chain rule. Another tool is integration. For example, the integral of the absolute difference of the piecewise linear function formed by the dataset and the evolved function could measure the divergence of the evolved function from the piecewise linear solution.

References

[1] S. S. Mousavi Astarabadi and M. M. Ebadzadeh. Avoiding overfitting in symbolic regression using the first order derivative of gp trees. In *Proc. GECCO*, 2015.

[2] J. C. Bongard and G. S. Hornby. Combining fitness-based search and user modeling in evolutionary robotics. In *Proc. GECCO*, 2013.

[3] W. La Cava, T. Helmuth, and L. Spector. Genetic programming with epigenetic local search. In *Proc. GECCO*, 2015.

[4] A. Chen, B. Xue, L. Shang, and M. Zhang. Improving generalisation of genetic programming for symbolic regression with structural risk minimisation. In *Proc. GECCO*, 2016.

[5] A. Coates and A. Y. Ng. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*, 2012.

[6] P. Craven and G. Wahba. Smoothing noisy data with spline functions. *Numerische Mathematik*, 31(4):377–403, Dec 1978.

[7] I. Dempsey, M. O'Neill, and A. Brabazon. Constant creation and adaptation in grammatical evolution. *Foundation in Grammatical Evolution for Dynamic Environments*, pages 69–104.

[8] I. Goncalves, S. Silva, C. M. Fonseca, and M. Castelli. Unsure when to stop? ask your semantic neighbors. In *Proc. GECCO*, 2017.

[9] M. A. Haeri, M. M. Ebadzadeh, and G. Folino. Improving gp generalization: A variance-based layered learning approach. In *Genetic Programming and Evolvable Machines*, 2015.

[10] A. Hara, J. Kushida, R. Tanemura, and T. Takahama. Deterministic crossover based on target semantics in geometric semantic genetic programming. In *2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 197–202, July 2016.

[11] D. M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Science*, 44:12–44, Jan 2004.

[12] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning with Application in R.* Springer, 2013.

[13] J. D. Knowles, R. A. Watson, and D. W. Corne. Reducing local optima in single-objective problems by multi-objectivization. In *Proc. Evolutionary Multi-Criterion Optimization*, 2001.

[14] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Ijcai*, 14(2):1137–1145, 1995.

[15] J. R. Koza. *Genetic Programming: On the Programming of Computer by Means of Natural Selection.* The MIT Press, Cambridge, Massachusetts, 1992.

[16] D. L. Ly, A. Saxena, and H. Lipson. Co-evolutionary predictors for kinematic pose inference from rgbd images. In *Proc. GECCO*, 2012.

[17] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[18] J. F. B. S. Martins, L. O. V. B. Oliveira, L. F. Miranda, F. Casadei, and G. L. Pappa. Solving the exponential growth of symbolic regression trees in geometric semantic programming. In *Proc. GECCO*, 2018.

[19] J. McDermott, D. R. White, S. Luke, L. Manzoni, M. Castelli, L. Vanneschi, W. Jaskowski, K. Krawiec, R. Harper, K. De Jong, and U.-M. O'Reilly. Genetic programming needs better benchmarks. In *Proc. GECCO*, 2012.

[20] A. Moraglio, K. Krawiec, and C. G. Johnson. Geometric semantic genetic programming. In C. A. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, pages 21–31, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[21] S. Mukerjee and M. J. Eppstein.

[22] Q. U. Nguyen, X. H. Nguyen, M. O'Neill, R. I. McKay, and E. Galván-López. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2):91–119, Jun 2011.

[23] J. Ni, R. H. Drieberg, and P. I. Rockett. The use of an analytic quotient operator in genetic programming. *IEEE Transactions on Evolutionary Computation*, 17(1):146–152, Feb 2013.

[24] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`.

[25] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the Trade*, pages 55–69, London, 1998. Springer-Verlag.

[26] R. D. Reed and R. J. Marks II. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. The MIT Press, Cambridge, Massachusetts, 1999.

[27] M. Schmidt and H. Lipson. Learning noise. In *Proc. GECCO*, 2007.

[28] M. Schmidt and H. Lipson. Age-fitness pareto optimization. In *Proc. GECCO*, 2010.

[29] SINTEF. Big data, for better or worse: 90% of world's data generated over last two years., May 2013. `www.sciencedaily.com/releases/2013/05/130522085217.htm` (accessed September 3, 2018).

[30] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[31] M. Szubert, A. Kodali, S. Ganguly, K. Das, and J. C. Bongard. Reducing antagonism between behavioral diversity and fitness in semantic genetic programming. In *Proc. GECCO*, 2016.

[32] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[33] Y. Wang. *Smoothing splines: Methods and applications*. Boca Raton, FL: CRC Press, 2011.

[34] M. Zhang, X. Gao, and W. Lou. A new crossover operator in genetic programming for object classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(5):1332–1343, Oct 2007.