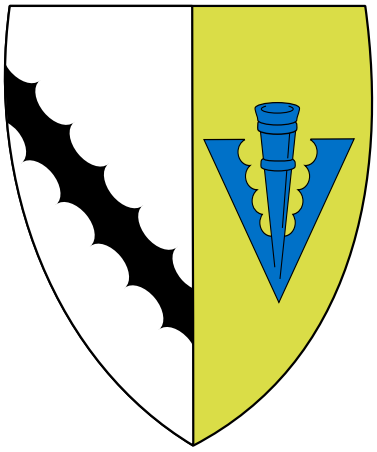
UNIVERSITY OF CAMBRIDGE
Department of Computer Science and Technology

Transforming Classical AI with Modern Computing: The BACON System for Equation Discovery from Scientific Data

Jonah Miller



Sidney Sussex College

May 2024

Submitted in partial fulfillment of the requirements for the Computer Science Tripos, Part III

Total page count: 51

Main chapters (excluding front-matter, references and appendix): 40 pages (pp 7–46)

Main chapters word count: 11396

Methodology used to generate that word count:

Overleaf word count.

Declaration

I, Jonah Miller of Sidney Sussex College, being a candidate for the Computer Science Tripos, Part III, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. In preparation of this project report I did not use text from AI-assisted platforms generating natural language answers to user queries, including but not limited to ChatGPT. I am content for my project report to be made available to the students and staff of the University.

Signed: *Jonah Miller*

Date: *28/05/2024*

Abstract

This project reconstructed Langley's `BACON` system for Computational Scientific Discovery. It proceeded to make improvements culminating in my `BACON.7` model which displayed up to 4% more noise-resilience than Langley's. It also presented higher model accuracy than state-of-the-art `PySR` when run on smaller datasets.

One of the main aims of this project was to prove that Classical AI can create an explainable and understandable solution when compared to modern neural networks. It accomplished this goal through its comparison to `PySR` and released the code open-source to encourage others to research in this area.

Acknowledgements

This project would not have been possible without the wonderful advice and persistent drive of my Part III Supervisor Dr Soumya Banerjee. It has been invaluable to have his experience guiding me throughout.

I would also like to express my gratitude to my Director of Studies Matthew Ireland. His seemingly infinite knowledge and unending time for me, has drastically improved the quality of my work this academic year and made my switch to the Computer Science Tripos seamless.

Lastly, I would like to thank Emily Hewett for offering unconditional support and compassion throughout the last 4 years - and also for proof-reading. Now we're even.

Contents

Chapter 1

Introduction

Discovering the numeric laws that shape experimental observations is a time-consuming endeavour filled with trial and error. Attempts at this are seen as early as the 16th century when astronomer Johannes Kepler deduced that planetary motion was an ellipse only after years of studiously trying to understand the observations of fellow astronomer Tycho Brahe. In the current age of computers this task can be deferred to programs which can spot patterns, relations and invariants in data, whilst performing the task faster and more accurately than any human. The initial program pioneering this field of Computational Scientific Discovery (CSD) was `BACON`, starting with `BACON.1` [Langley, 1977].

This initial work by Langley sparked a research agenda, that he and others contributed to over the next 20 years. Many programs were developed either directly inspired by, or to compete against, `BACON`. In fact, in 1997 the first neural network was designed to solve this problem [Saito and Nakano, 1997]. Fast-forward to the modern day, deep learning algorithms form the recent research in this area. Classical AI techniques like those used in `BACON` have been forgotten.

This project explores whether Classical AI techniques, represented by `BACON`, can contribute to the current field of CSD.

1.1 Motivation

Classical AI algorithms needed to be optimised and simplistic but also efficient to run on 1980s' computers. It seems reasonable that this refined nature can stand the test of time when rewritten in modern programming languages. Modern CSD techniques have made extensive use of DNNs but relatively little work has been done at applying Classical techniques. This becomes a self-perpetuating cycle. The success of modern DNNs encourages others to adapt these ideas, whilst the lack of research and reproduction of classical techniques means open-source implementations don't exist. It makes developing

and comparing with the classical programs infeasible, further hindering development in this area. This project aims to both prove that 1980s' ideas such as BACON are valid today, but also release my code open-source to encourage others to research in this area.

Modern DNNs lack the explainability that BACON offers. What a DNN outputs is often not explainable due to the black-box nature of neural networks. There's no description of how their conclusions are reached causing the systems to be less trustworthy. What is more, they often are not reproducible further reducing understandability. This is a problem synonymous with modern AI at the moment where the lack of transparency can lead to dangerous consequences as discussed in subsection 2.2. As such, this project aims to create an understandable solution that can help improve the field of CSD, whilst still being able to output useful answers.

Another issue is the time and computational complexity taken to train DNNs is immense in comparison to BACON which contrastingly is fast through its simplicity and doesn't have a training mechanism using up computational resources. In addition, initial tests with noiseless datasets indicate BACON outperforming DNN PySR in terms of speed with comparable accuracy. This implies there is potential to make BACON improvements so that it is competitive in specific environments with the state-of-the-art whilst using less power.

1.2 Technical Goals

The technical goals of this project are:

1. Deploy fully-functioning Python versions of BACON.1, BACON.3, BACON.5 and BACON.6, all different aspects of the original BACON system.
2. Make improvements on Langley's BACON aimed at improving noise-resilience and understandability. This model will be called BACON.7 and comprise of a mixture of novel techniques and the best parts of Langley's BACON.
3. Evaluate how BACON.7 compares to DNN PySR in finding the true equation of noisy datasets representing the Ideal Gas Law, Ohm's Law and Black's Law.
4. Use a Monte Carlo Tree Search (MCTS) to run many simulations of BACON.7 which iterate through the BACON tree (see Figure 2.1 for details) with different pathing, with aims of improving the accuracy of BACON.7 in noisier environments than (3).

1.3 Achievements

Every goal in this project was achieved.

1. Each individual BACON model was built, albeit slightly differently from Langley's initial design. The functionality of each system was the same (confirmed by solving

all equations correctly at 0% noise in subsection 5.4), however small changes were made to improve usability and noise-resilience.

2. `BACON.7` was formed in a combination of Langley's `BACON.1` and a novel technique I used to traverse the Space of Data. The program was also built to emphasise understandability through verbose settings, this also gave verification that `BACON.7` was working as anticipated.

3. `BACON.7` could find the correct form of the Ideal Gas Law and Ohm's Law up to the 4% relative noise aimed for prior (details in 5.1.4). Black's Law could only be found up to 1.5%. Langley's `BACON` could solve 0.5% noise on the Ideal Gas Law and 0% noise on the other two. When comparing with `PySR` on the Ideal Gas Law and Ohm's Law, `BACON.7` was more accurate. On Black's Law, `PySR` was better.

4. `BACON` was successfully integrated with MCTS but accuracy did not improve. This allowed a discussion on the ambiguity of the true equation in noisy data, and where the limits of `BACON.7` are.

In all, a successful reproduction and then improvement of the explainable `BACON`. This project proved that there is room for Classical AI in the modern day.

Chapter 2

Background

2.1 Computational Scientific Discovery and Classical AI

Computational Scientific Discovery (CSD) is about analysing datasets to find patterns, relations and invariants within. The goal of this project was to create a model that takes in a dataset of unknown size, with n independent variables and one dependent variable which could contain noise. The model must then find the best general relationship to model the variables together which strips the noise from the dependent variable. This model must follow the paradigms of Classical AI, no matter the consequences such as sacrificing goodness-of-fit from modern machine learning methods for simplicity.

Classical AI follows a set of rules rigidly. It is deterministic and reproducible. The programs are not resource intensive and the AI doesn't learn the data inputted to it. The output is interpretable and the decisions are clear based on their programmed logic. `BACON` is on the simplistic side of Classical AI. Its techniques are heuristic-based rather than grounded in mathematical truth and its algorithm is straightforward.

All implementations in this project had to obey these deterministic principles. An example of a technique that cannot be used is Gaussian Processes (GPs). GPs are built on Bayesian probability. In this project, they would have been used to build a multivariate normal distribution that models the dataset. To evaluate this model one of the multivariate distributions is randomly sampled. That is too stochastic due to the randomness in the sampled variable.

Contrast with a Monte Carlo Tree Search, which is explained in subsection 6. The aim is to search a game-tree to optimise the outcome (such as find the best move in chess). It intakes a parameter, C, which determines whether the search leans towards exploitation or exploration. There can be significantly different outcomes with small deviations in C but it is purely deterministic as the same C will always return the same outcome. This

is allowed for this project.

One of the questions this project aims to answer is if Classical AI approaches are still effective today and if their simplicity is able to yield results comparable to modern neural networks?

2.2 Explainable AI

AI is becoming increasingly prevalent in society. The applications range from AI chatbots such as ChatGPT, to medical diagnoses for brain cancer [Khalighi et al., 2024]. Many of the models behind these applications are opaque and the reasoning behind their outputs aren't even understandable to their designers [IBM, 2022]. This can lead to ethical concerns such a lack of clarity on whether an AI model is discriminatory against protected characteristics or health-concerns in the medical field where a misdiagnosis can be fatal. Explainable AI (XAI) is a field of research which aims to mitigate against these issues. It encourage developers to create understandable and explainable AI models, or alternatively make black-box models more interpretable.

An aim of this project is to apply it to real-world datasets, where relationships between variables are unknown. There is no method to verify the conclusion is correct. To be able to trust our answer we need to have an understanding of how it was formulated. Moreover, an explainable model aids in debugging and developing. Being able to pinpoint causes for spurious results allows improvements to be made directly. Comparatively – and as is shown later – black-box models such as `PySR` don't have a mechanism to explain how they work. When they don't find the correct equation on a dataset there's no solution apart from to rerun. When applied to real-world unknown data, the results are not trustworthy as the mechanism is not explainable.

There is a trade-off. Black-box models can be much more powerful than explainable models. Computers can infer patterns in ways humans can't understand and come to the correct conclusions even if there is seemingly no justification. This project aims to find out if there is an explainable model to aid in CSD that is comparable in accuracy to black-box models.

2.3 The `BACON` system

The `BACON` system was the focus of Langley's research in CSD through the 1980s'. It was coded in the now antiquated production-system language PRISM and the culmination of the project was a textbook describing an overview of the program [Langley et al., 1987b]. **This textbook formed the starting point of this project, and was the primary source of information I used to remake `BACON` in modern object-orientated Python.**

BACON was chosen because its methodology is explainable and simple even in relation to other Classical AI techniques. This is relevant, as there is no code for any of the projects from this era. To adapt one of these projects to the modern-day, it is most straightforward to take an understandable design. As displayed in subsection 3.1 BACON is also not substantially worse than any of the projects built after it in this era.

Its core functionality is based on one heuristic applied repeatedly. To find the relationship between two variables X and Y, if X increases whilst Y decreases (or vice versa), consider their product XY. If they both increase or decrease then consider the divisor $\frac{X}{Y}$. Whilst there doesn't seem to be a mathematically rigorous proof of how this leads to invariants, in the case of every example imagined it does. The best explanation is that BACON is a trend detector [Nordhausen and Langley, 1990]. It systematically analyses and ranks the trends found which allows it to uncover further patterns leading to the true relationship.

Table 2.1 demonstrates the path BACON.1 takes to uncover Kepler's 3rd law, first uncovering the trend of D and P increasing together to formulate $\frac{D}{P}$. Comparing $\frac{D}{P}$ with D leads to $\frac{D^2}{P}$. Comparing $\frac{D^2}{P}$ with P leads to $\frac{D^2}{P^2}$. Then the subsequent comparison with D leads to $\frac{D^3}{P^2}$ which the system diagnoses as invariant.

Planet	Distance (D)	Period (P)	$\frac{D}{P}$	$\frac{D^2}{P}$	$\frac{D^2}{P^2}$	$\frac{D^3}{P^2}$
A	1.0	1.0	1.0	1.0	1.0	1.0
B	4.0	8.0	0.5	2.0	0.25	1.0
C	9.0	27.0	0.333	3.0	0.111	1.0

Table 2.1: An example of the BACON.1 algorithm discovering Kepler's 3rd law from a noiseless planetary system. The program is acting on 3 different synthetic planets which obey the same laws of motion (data from [Langley et al., 1987b]).

BACON can also function with multiple variables. A demonstration of how this works is in Figure 2.1 for the Ideal Gas Law - equation $V = \frac{M(T+273)}{P}$. This Figure is used as context for the next two paragraphs. First, BACON.3 forms the datapoints into a tree (referred to as the BACON tree), with the dependent variable V at the bottom[1]. Then at the lowest level in the tree BACON iterates over all the sets. The sets are datapoints which share the same values in the variables in the layers above the current layer. In this first layer, sets are the 3 datapoints that share values in M and T with differing P and V. There are 9 of them.

Each set is fed into BACON.1 which derives the relationship between variables P and V. All 9 sets discover that PV is equal to some constant. This constant differs between sets and PV becomes the new dependent variable at the bottom layer of a new tree. This process is then repeated between PV and T. A linear relationship[2] is found with form

[1]For a dataset to be acted on by BACON it needs to have n independent variables and a single dependent variable. Each independent variable needs to take at least 3 values. This gives a total of 3^n datapoints.

[2]The variable representing the gradient in a linear relationship is always represented by a lowercase letter in the alphabet.

$PV - aT$ constant in any given set. A second tree is formed with a as the dependent variable as well as the original tree continuing with $PV - aT$. The last comparison with M discovers final equations $a = M$ and $PV - aT = 273M$. These are combined along dummy variable a to form the Ideal Gas Law.

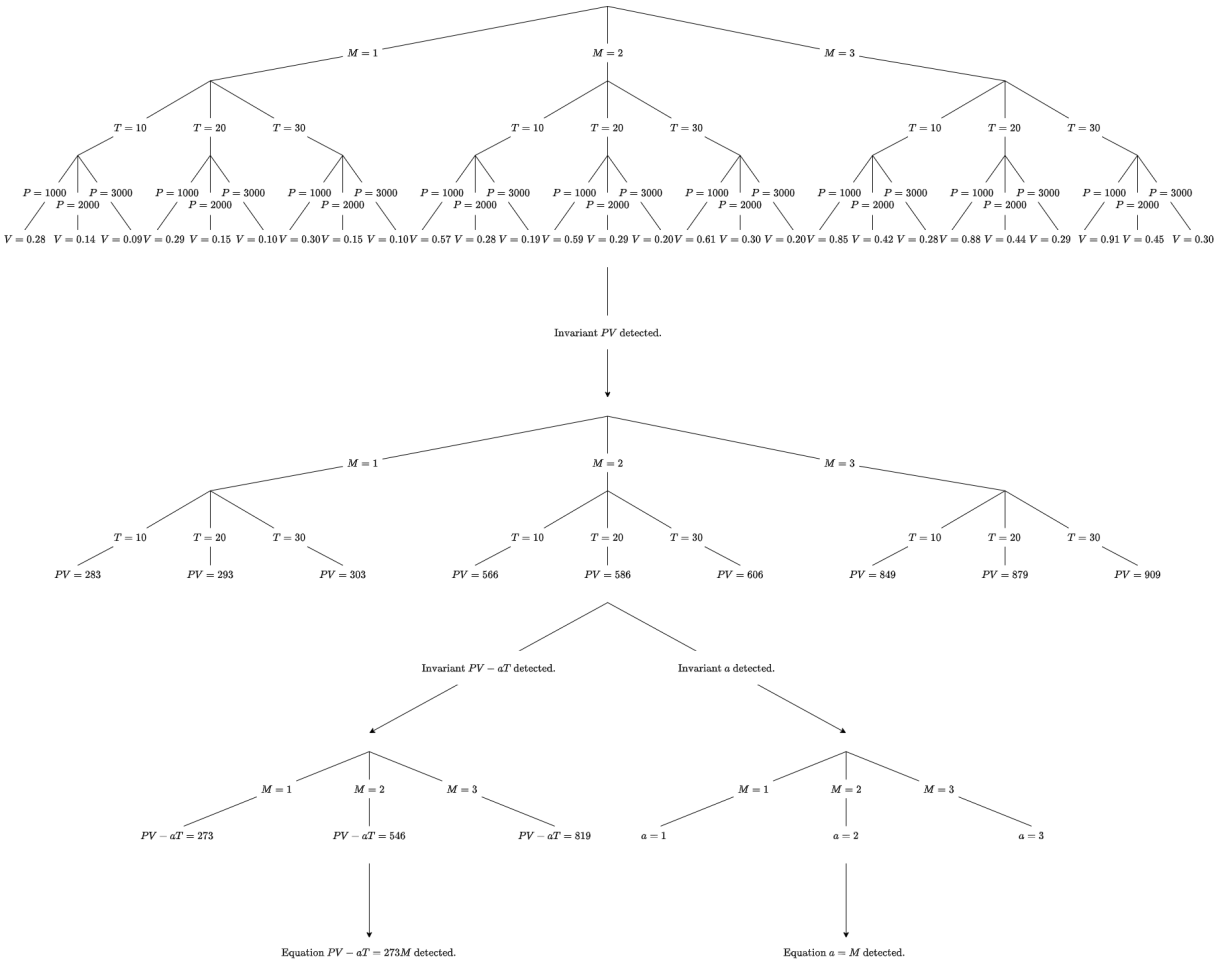


Figure 2.1: Consider the Ideal Gas Law $V = \frac{M(T+273)}{P}$. It is placed in a noiseless environment where V is a dependent variable, P, M and T are independent variables each consigned to 3 specific values, giving 27 values for V. They are arranged in the `BACON` search tree. At the lowest layers it runs the `BACON.1` heuristic between P and V. Define the sets at a layer as those that share the same value in the independent variables in the above layers. For example, the first set in the first layer, has $T = 10,\ M = 1$ and are the 3 points defined by $(P = 1000,\ V = 0.28)$, $(P = 2000,\ V = 0.14)$ and $(P = 3000,\ V = 0.09)$. All 9 sets in this layer determine invariant PV. `BACON.3` detects the agreement, then forms a new tree with PV as the dependent variable. It proceeds to run a heuristic check between PV and T. All 3 sets at this layer determine $PV - aT$ is invariant. This also introduces new dependent variable a. As both a and $PV - aT$ can be a function of M, `BACON.3` forms two new trees at this level. The last `BACON.1` heuristic check finds $a = M$, $PV - aT = 273M$. These are collated to form the Ideal Gas Law.

`BACON` went through further iterations. `BACON.5` is an improvement on `BACON.3` aimed at exploiting symmetry to reduce noise and runtime. `BACON.6` detects invariants using correlation coefficients and was built specifically with noise-resilience in mind.

`BACON.1` and `BACON.6` are algorithms designed to solve problems in the Space of Laws. They are fed variables X and Y and have to determine the law governing their relation.

BACON.3 and BACON.5 work in the Space of Data. This refers to how the BACON tree is traversed which involves the forming of the sets, determining which values to give to algorithms in the Space of Laws and then creating a new BACON tree with one less layer. This process becomes more involved once the sets in a layer present different candidate expressions. I explore this in section 4.3.

This is one significant limitation of BACON. BACON was invented in a noiseless vacuum. Effectively there was the assumption that noise didn't exist. Synthetic datasets formed testing, giving the perfect environment for BACON to perform. With noise the results are poor. Langley alludes to this by describing BACON.5 as "cannot adequately deal with noise" [Langley et al., 1981]. Fixing this is one of the main motivations of my BACON.7.

2.4 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a type of game-tree search based on user-programmable heuristics. Invented by Rémi Coulom [2006], it takes the random sampling from a Monte Carlo method intertwined with a reward function evaluated at the end of the game. This maximises the path taken at any given stage - making it a form of reinforcement learning. It has varied uses such as board games Chess [Czech et al., 2020] and Go [Couetoux et al., 2013] when layered on an artificial neural network to increase its power and efficiency. Examples also include video games [Kim et al., 2024], and more stochastic games such as applications for Poker [den Broeck et al., 2009] and Settlers of Catan [Szita et al., 2009].

Game trees are a form of heuristic-based search which falls into Classical AI territory. It feels like a natural expansion, especially once considering the tree-like nature of BACON, and the scorable reward function (determined by how close the BACON prediction is to the data inputted). It should make BACON more powerful by allowing it to search through large datasets and choose the best equation.

An interesting research direction is combining Classical AI and modern machine learning techniques. As MCTS is more modern than BACON, this forms a good litmus test about whether BACON can be the candidate Classical AI approach to perform this union.

Chapter 3

Related work

This section outlines alternative Classical AI algorithms from the 1980s' and gives more context as to why BACON was chosen. It then discusses the current deep learning approaches to Computational Scientific Discovery, and explains why PySR was selected to be the modern comparison to BACON throughout this project.

3.1 BACON inspired heuristic based search

BACON kick-started a wave of Classical AI in the 1980s exploring CSD. There were multiple projects inspired by Langley using similar heuristic based designs. Two of the most popular were Fahrenheit [Koehn and Zytkow, 1986] and Abacus [Falkenhainer and Michalski, 1986]. Fahrenheit makes minor improvements to BACON such as being able to handle irrelevant variables and re-ordering the search algorithm to deal with more terms (covered in limitations in subsection 4.3.3).

Abacus is a more interesting project, outlining similar concerns to mine about BACON. These include dealing with noise, and specifically the problem of approaches such as BACON.6 which require information about the search space (seen in section 4.2). Their solution is threefold. They introduce a more general approach to the search space, such as allowing variables X and Y to have invariant $Y = X^2$ for $X < 3$, but $Y = 12 - X$ for $X \geq 3$. This lets multiple invariants be found within a dataset - something BACON cannot do. They also develop the concept of a proportionality graph search. This uses ideas from graph theory to cycle through the possible variable relationships in multivariable systems. Lastly, they reduce the search space for the invariants by ignoring relationships which can't occur through dimensional analysis.

Fahrenheit's re-ordering of the search algorithm, and Abacus's use of dimensional analysis give fascinating concepts that may be applied to later versions of BACON.7. However, neither deal with noise. One of Fahrenheit's main contributions to deal with irrelevant variables is solved by BACON.7, whilst Abacus yields an approach irreconcilable with

BACON. It is also not able to solve a more versatile set of equations than BACON. Hence there is no justification to using either of these projects rather than BACON in attempting to make a noise-resilient Classical AI system. Lastly, no open-source implementations of either means it is hard to meticulously examine for strengths and weaknesses. It is also too time-consuming to make a coded implementation.

The era of Classical AI concludes with SDS [Washio and Motoda, 1997]. It is a mathematically rigorous approach to the same problem, stemming from two postulates by physicist Edgar Buckingham in 1914 that govern the relationship between complete equations and their associated variables scale-type. In other words, given a set of variables and knowledge about how they scale relative to other variables available, a set of possible equations can be formed. This set can be pruned based on the data (using bivariate statistical tests) until only certain equations are possible. The entire algorithm both reduces the complexity found in Langley's BACON, whilst increasing the depth of equations possible. Under the same initial assumptions as BACON, for dependent variable X from a 17 variable circuit meter, the following equations is discoverable:

$$\left(\frac{R_3 h_{fe_2}}{R_3 h_{fe_2} + h_{ie_2}} \frac{R_2 h_{fe_1}}{R_2 h_{fe_1} + h_{ie_1}} \frac{rL^2}{rL^2 + R_1}\right) V - \frac{Q}{C} - \frac{K h_{ie_3} X}{B h_{fe_3}} = 0 \tag{3.1}$$

Whilst this isn't possible in BACON, it displays an increase in flexibility rather than an explicit ability to deal with more noise. On the latter, their best contribution is a 4% relative standard deviation of the dependent variable they claim is solvable through their method - though not backed up in the paper. This becomes the baseline I'd like to get my BACON to solve. SDS is limited by both the structure the algorithm has to follow and the understandability it lacks - one of the key aspects of using BACON and why it was chosen as the Classical AI approach. Likewise to Fahrenheit and Abacus, there is no open-source project to gain accurate comparisons to my BACON.

None of the Classical AI competitors to BACON share its level of understandability whilst also being more noise-resilient. BACON is the most straightforward algorithm to adapt based on the detail from Langley's textbook.

3.2 Modern neural network approaches

Modern approaches are based around using neural networks. ConservNet proposed by Ha and Jeong [2021] trains a deep feed-forward neural network using their novel noise-variance loss function:

$$\mathcal{L} = \sum_i \mathrm{Var}\left(F_\theta\left(\mathbf{x}_{ij}\right)\right) + \left|Q - \mathrm{Var}\left(F_\theta\left(\mathbf{x}_{ij} + \boldsymbol{\epsilon}_{ij}\right)\right)\right|. \tag{3.2}$$

F_θ represents the conserved function, the left term in the sum represents reducing the intra-group variance of all data that maps to the same value via the function. The right

term is the allowed perturbations of the invariant when noise is added to the system - represented by Gaussian noise $\boldsymbol{\epsilon}_{ij}$. Combined it leads to a powerful dynamic where F_θ can't converge trivially, forcing invariants to be found in systems with well-defined Hamiltonians (a measure of total energy in the system). Q controls the scale of the variation. The program is more powerful than `BACON`. This can be seen in their ability to find the conserved quantity C in environments as varied as real, noisy double pendulum data:

$$C = L_1^2(m_1 + m_2)\omega^2 + m_2 L_2^2 w^2 + 2m_1 m_2 L_1 L_2 \omega_1 \omega_2 \cos(\theta_1 - \theta_2) \tag{3.3}$$

$$- 2gL_1(m_1 + m_2)\cos(\theta_1) - 2gm_2 L_2 \cos(\theta_2) \tag{3.4}$$

`BACON` cannot do this.

Similar to `SDS`, `ConservNet` exploits a more mathematical approach to improve results. Moreover, `ConservNet` proves how its neural model converges to appropriate answers and demonstrates substantial experiments dealing with greater noise than any effort I produce with `BACON.7`. It can also deal with multiple dependent variables which `BACON` cannot.

Its downsides come in the complexity and explainability. The loss function is novel, but when overlayed on a black-box neural network the meaning is reduced - further shown by the multiple settings needed for them to test to find the optimal hyperparameters for their best model. Additionally, with training times up to several hours, it goes against the principles of the Classical AI programs I'm enacting in this project. An interesting implementation would have been putting the trained black-box model at the Space of Laws level, however due to time constraints this was not possible - though it was done with `PySR`.

`PySR` [Cranmer, 2023] uses symbolic regression to find invariants. Symbolic regression works by imagining the search space as possible mathematical expressions. The expressions are narrowed down to the most accurate and often have a bound on complexity and simplicity in the final result. The latter often manifests itself by reducing constants from floats to their nearest integers. A hypothetical example is when considering Black's law:

$$T_f = \frac{M_1 T_1}{M_1 + M_2} + \frac{M_2 T_2}{M_1 + M_2} \tag{3.5}$$

With noisy data for independent variables M_1, M_2, T_1 and T_2, dependent T_f, `BACON` may find

$$T_{f_{\texttt{BACON}}} = 1.002\frac{M_1 T_1}{M_1 + M_2} + 0.844\frac{M_2 T_2}{M_1 + M_2} \tag{3.6}$$

whilst `PySR` finds the exact form – equation 3.5 – as it assumes the coefficients are 1. `PySR` is also a much more versatile and powerful program with customisability and the ability to run in large and complex search spaces where the only bound is the computational power. As such, it can handle invariants such as in differential equations and discontinuous

environments. BACON cannot compete.

However there are downsides. As it is trained on a deep neural network [Cranmer et al., 2020] there is no interface or explainability giving a reason for its outputs. This reduces the trustworthiness of the model.

PySR is used in this project as the main DNN comparison. The reasons why can be seen through two key components in a standard PySR implementation in Figure 3.1.

```
# ml_methods/symbolic_regression.py

model = PySRRegressor(
    niterations=15,
    maxsize=30,
    binary_operators=["+", "*", "/", "-"],
    extra_sympy_mappings={"inv": lambda x: 1 / x},
    loss="loss(prediction, target) = (prediction - target)^2",
    model_selection="accuracy"
)
```

Figure 3.1: The setup for the PySR implementation used throughout this project. Note *binary_ operators* restricting what type of equations can be found, and *niterations* determining how long PySR searches for.

The *niterations* purposes matches that of the complexity counter j in BACON.1 (detailed in subsection 4.1). Experimentally, $j = 4$ and *niterations* $= 15$ made the complexity of the forms outputted by each program similar. Also, the *binary_ operators* is set at [" + "," ∗ "," / "," − "]. This coincides with the operations that BACON.1 can output. In all, it makes the final equations yielded from my BACON.7 and PySR (in subsection 5.3) comparable allowing a balanced discussion on their strengths and weaknesses.

ConservNet's interface didn't trivially allow this level of control. ConservNet also does not have substantial documentation which PySR does, making the PySR implementation straightforward.

Chapter 4

Design and Implementation

This chapter outlines the technical details to reconstruct BACON. It concludes with the new BACON.7 linking all the components together.

In the Space of Laws, BACON.1 and BACON.6 were formed, both of which follow Langley's initial designs. BACON.1 was used as it forms the simplistic backbone that makes BACON appealing. BACON.6 was promised by Langley to be noise-resilient, I wanted to test if that was true.

In the Space of Data, I created a novel mechanism. It uses the layer-by-layer approach introduced by BACON.3 and the sense of symmetry from BACON.5. This mechanism can be transformed into BACON.3 if the appropriate parameters are set. Whilst a separate implementation of BACON.5 exists, it struggles with noise too much to be useful.

4.1 BACON.1

BACON.1 is an algorithm that deciphers the relationship between two symbols X and Y with associated datasets $d(X)$ and $d(Y)$ and no prior knowledge of how they interact. It is an iterative process, where the two symbols X and Y are steadily combined to attain the correct form. This is demonstrated in Table 2.1 for symbols D and P. I built BACON.1 due to it being the simplistic backbone of BACON.

It consists of two components. The **heuristic-layer** determines the next step in the process from symbols X and Y. It outputs either a function $f(X, Y)$ with associated dataset $f(d(X), d(Y))$, or null if X and Y don't share a valid relation. The **managing-layer** lies on top of the heuristic-layer and chooses which variables and hyperparameters to pass to it. It controls the complexity and scope of relationships that can be found.

4.1.1 Heuristic-layer

Overview

The heuristic-layer is handed two symbols, X and Y, respective datasets $d(X)$ and $d(Y)$ and hyperparameters δ, ϵ and c_{val}. The order of its run is in flowchart Figure 4.1.

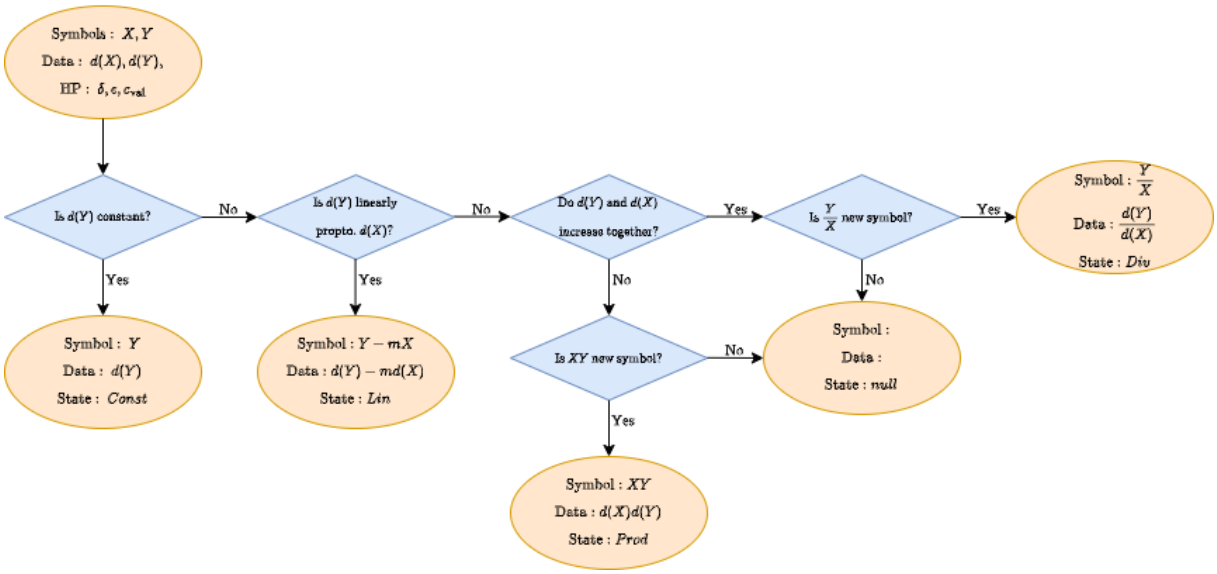


Figure 4.1: The full algorithm for the heuristic-layer. When the data is returned via $f(d(X), d(Y))$ the function is acted on element-wise. Also, the state return for a linear relationship, includes a list of the parameters of the relationship, such as the gradient and y-intercept. The consistency check determines if $d(Y)$ is constant, and the linear proportionality check determines if the data is related linearly. The novel relation check determines if a new symbols has been found. These checks implicitly use the hyperparameters the heuristic-layer is initialised with, where appropriate.

The logic behind the consistency check, linear proportionality check and novel relation check are described below.

Consistency check

For $M_Y = \text{mean}\,(d(Y))$. If for each value in $d(Y)$:

$$M_Y(1-\delta) < \text{value} < M_Y(1+\delta) \tag{4.1}$$

then Y is defined as constant. Increasing δ increases the frequency in which relationships are found. To deal with negative values, the absolute value of the datasets are used throughout. A practical example is explained through Figure 4.3.

Linear proportionality check

X and Y are compared via the correlation coefficient[1]. If $|r| \sim 1$ then X and Y are

[1]For ordered datasets $d(X) = \{x_i : i \in 1, ..., n\}$ and $d(Y) = \{y_i : i \in 1, \}$ with respective means M_X and M_Y, the correlation coefficient r is defined by:

$$r = \frac{\sum_{i=1}^{n}(x_i - M_X)(y_i - M_Y)}{\sqrt{\sum_{i=1}^{n}(x_i - M_X)^2}\sqrt{\sum_{i=1}^{n}(y_i - M_Y)^2}} \tag{4.2}$$

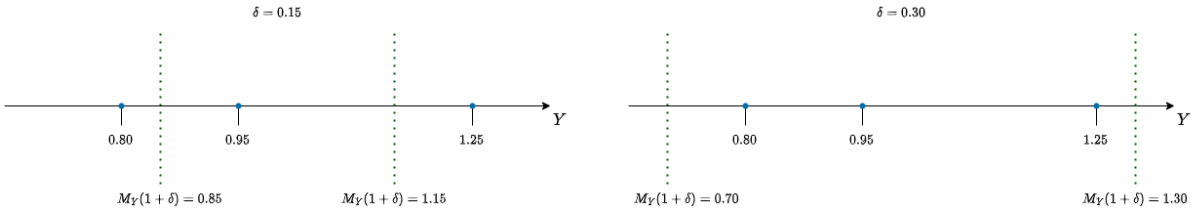


Figure 4.2: For symbol Y with $d(Y) = \{0.80, 0.95, 1.25\}$ so $M_Y = 1$, the consistency rule holds for the green dotted boundaries defined by $\delta = 0.30$, but not for $\delta = 0.15$. If the consistently rule holds, the quantity is defined to be invariant - here in the case of $\delta = 0.30$, Y would be defined as invariant with value of M_Y.

linearly proportional. $r > 0$ demonstrates that the sets increase or decrease together and they should be divided, $r < 0$ displays one increases whilst the other decreases and their product should be taken. Graphically this is shown by Figure 4.3.

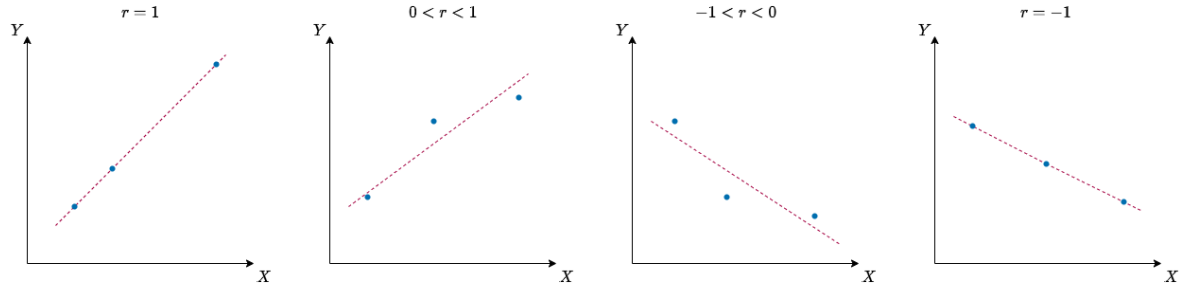


Figure 4.3: A demonstration of the different values of r for ordered datasets $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$. The pink dotted line is the line of best fit through the points. If $|r| \sim 1$, then X and Y are linearly proportional. A positive r implies they increase together, a negative r shows that one increases whilst the other decreases.

A linear relation is deduced if $1 - |r| < \epsilon$. Increasing ϵ increases the frequency linear relationships are ascertained. If a linear relationship is determined, a check is made that the y-intercept is non-zero. This is as $X + mY = 0$ is equivalent to $\frac{X}{Y} = -m$. The former means variables $X + mY$, and m have to be stored by the system whilst the latter only requires $\frac{X}{Y}$ meaning its quicker and simpler to be handled. The threshold is determined by variable c_{var}. For approximated linear relationship $d(Y) = md(X) + c$, if $|\frac{c}{\text{mean}(d(Y))}| < c_{\text{var}}$ then the y-intercept is classified as zero.

Novel relation

The iteration process discovers new forms. To make sure it doesn't recover an already found symbol, the output of the product/division step must be checked. For example, if the heuristic-layer was passed symbols X and XY by the managing-layer, and XY does not pass the consistency or linear proportionality check but has $r > 0$. The new symbol would be $\frac{XY}{X} = Y$. As Y has already been seen this is classified as an invalid symbol, and the heuristic-layer would return state null.

.

4.1.2 Managing-layer

A visualisation of the managing-layer and its interaction with the heuristic-layer is in Figure 4.4. The goal is to uncover the most accurate relation between X and Y, whilst

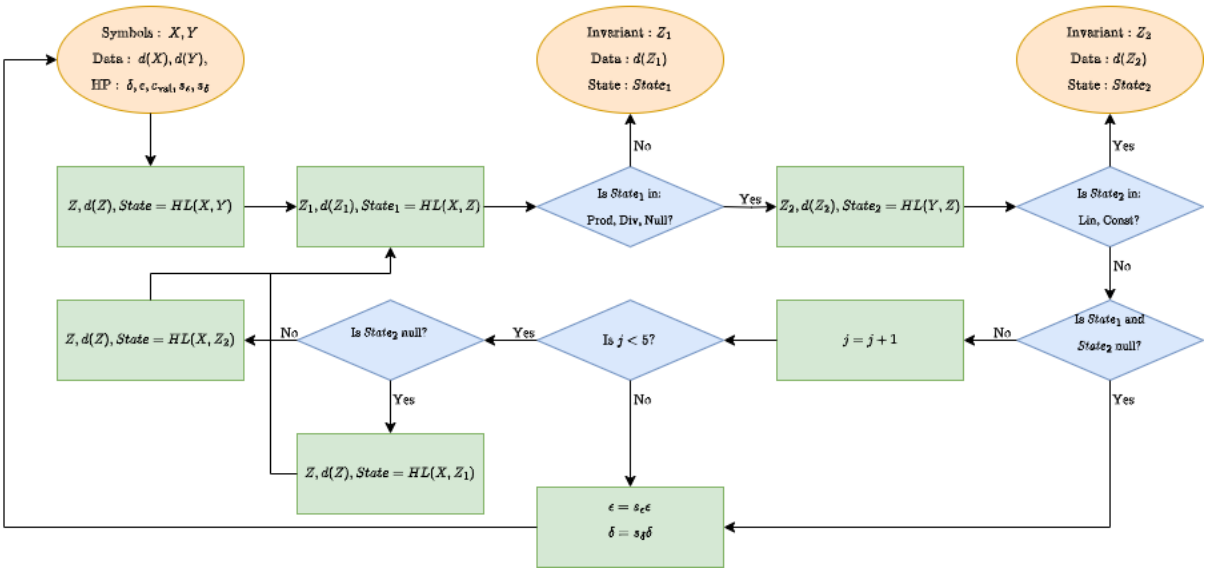


Figure 4.4: Flowchart describing the managing-layer of the `BACON.1` algorithm. The heuristic layer is run whenever HL is called. It implicitly inputs the δ, ϵ and c_{val} hyperparameters as well as the datasets of the symbols being inputted.

running the heuristic-layer the minimal amount of times. There are two key variables throughout. j is the iteration counter. It controls the complexity of the allowed equations. Variables s_ϵ and s_δ are the scale factor ϵ and δ are multiplied by, if no equation has been found once the iteration counter maxes out or no relationship is found between X and Y.

Order of events

It runs a sequential search where the new relationship - Z in Figure 4.4 - is compared first to variable X then to Y - even if X finds a relation. This is done to allow X and Y to be tested as a linear relationship with each found variable, whilst not searching exhaustively, in the aim of running the heuristic-layer as little as possible. My method works under an assumption described through the following example. If the true relation to find is $\frac{X^3}{Y}$, and the heuristic-layer run of X and $\frac{X^2}{Y}$ has outputted this symbol. By the order above, the heuristic-layer between $\frac{X^2}{Y}$ and Y now runs. My assumption is that the relationship found is a product, which is invalid as X^2 is an invalid constant as it would have already be found by X being constant. This allows the next run of the heuristic-layer to occur between $\frac{X^3}{Y}$ and X confirming $\frac{X^3}{Y}$ as the invariant. I'm yet to find a counter-example here.

Iteration Counter

j increments every-time the heuristic-layer returns a value. This adds a complexity cap to the relationships found. This is demonstrated in Figure 4.5. Every new symbol is of the form X^nY^m for $|n| + |m| \leq j + 1$ and $n, m \in \mathbb{Z}$. Each of these symbols can be in a linear relationship with X and Y.

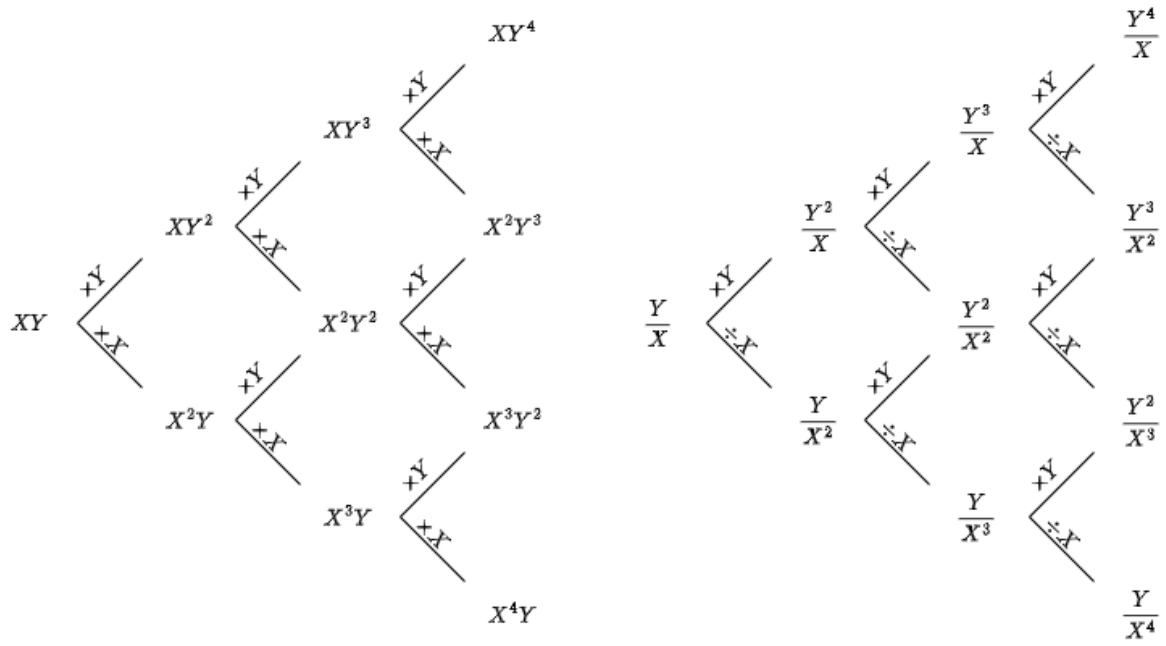


Figure 4.5: All 10 possible relations for a `BACON.1` run capped at $j = 4$. Note this is the minimum required to find Kepler's Law.

Scale Factors

The scale factors cause auto-increasing ϵ and δ if the managing-layer concludes without finding an invariant. In noisy environments the hyperparameters for error tolerance needed vary between layers in the tree. Allowing the system to auto-update itself means a relationship will inevitably be found for every set. These are governed by s_ϵ and s_δ. The updated ϵ and δ get multiplied by this scale factor if the managing-layer is rerun. Tests were done with incremental rather than multiplicative scale factors. On noisy datasets, the ϵ and δ needed at the final layer of the tree, when only 2 variables remain, are experimentally orders of magnitude larger than earlier layers. The multiplicative factor gets to these magnitudes quicker than an incremental approach. Larger s_δ will increase the programs tendency to find product/division relations, likewise for s_ϵ and linear relations.

4.1.3 Strengths of `BACON.1`

`BACON.1` is fast. Consisting of only simple checks between two sets of 3 datapoints allows a run of the managing-layer `BACON.1` to take ~ 0.03 seconds. On noisier datasets, reruns of the managing-layer are more likely. Due to the speed of `BACON.1` multiple reruns can occur without a large time deficit. Moreover, reducing this speed was the reasoning behind certain design decisions such as the complexity counter and the order of events in the managing-layer.

The alternative `BACON.6`, discussed in section 4.2, is the other method Langley devised of a similar task and it sacrifices speed for accuracy. `BACON.1` is about $4\times$ faster for a single run whilst not requiring prior knowledge of the relationship between the symbols which `BACON.6` does. It allows `BACON.1` to work in more general spaces.

This combination makes BACON.1 versatile and adaptable to different datasets no matter the noise. It thrives under modern computing which runs its algorithms quickly.

4.1.4 Weaknesses of BACON.1

BACON.1 needs at least 3 datapoints to infer any relationship. With only two datapoints, multiple relationships can be formed that fit the data equally well. For example if $X = \{2, 3\}$ and $Y = \{4, 9\}$, relationships $Y = X^2$ and $Y = 5X - 6$ are both valid, though a third datapoint would distinguish between these - or give another relation. This tautological nature is an essential limitation of BACON with no fix apart from gathering more data. Furthermore, this reasoning is why BACON can't deal with relations that are sums of three terms ie. $f(X, Y) + ag(X, Y) + bh(X, Y) =$ constant, for functions f, g, h. A quadratic function is uniquely determined by 3 points [Conroy, 2012], however terms of higher powers may hold the same relation. For example, points $X = \{1, 2, 3\}$, $Y = \{1, 8, 27\}$ satisfy both $Y = X^3$ and $Y = 6X^2 - 11X + 6$. To distinguish, an additional datapoint is needed.

The managing-layer only compares each term found in Figure 4.5 with X and Y to save time. This limits the relationships found. For instance, it couldn't find the linear relationship $X^2Y^3 - aXY$. This is changeable for a computational complexity and time trade-off. For the datasets used in this project it is not a necessary change.

4.1.5 Relation to Langley's BACON.1

Heuristic-layer

The heuristic-layer is almost identical to Langley's. Both the consistency and linear proportionality tests are in Langley's BACON.1. However, I add the hyperparameter c_{val}. This was done experimentally when I was testing on noisier data, and I discovered linear relationships were being found spuriously. Consider Ohm's Law (details in subsection 5.1.2), the first layer requires the relationship $IL - aI$ to be found. Instead, $I - aL$ appeared consistently in noisy data with few instances of $IL - aI$. Reducing ϵ stopped the few instances $IL - aI$ appearing which instead formed relation IL. However, increasing c_{val} to 3, stopped $I - aL$ appearing at all whilst not affecting $IL - aI$.

Managing-layer

Langley doesn't explicitly describe his managing-layer, but must have one in order for his heuristic-layer to function. He also makes no mention to an iteration counter or the scale increases for δ and ϵ. His program only terminated with an explicit stop command, or when a relationship was found. When he dealt with noiseless data this wasn't an issue as a relationship is almost always found. My method means I can successfully deal with noisy data, which needs different hyperparameters at different layers in the tree. This was the most crucial upgrade to BACON.1 to allow it to perform well on noisy data.

4.2 BACON.6

BACON.6 was the last iteration in the BACON family. It was described by Langley as a "hill-climbing method for dealing with noise" [Langley et al., 1987a]. BACON.6 was implemented due to Langley's belief in its noise-handling. In addition, it is the only understandable classical algorithm I've found that is adaptable to find invariants that contain functions that are trigonometric or exponential.

The difference with BACON.1 is that BACON.6 requires prior knowledge about the type of relationship the two symbols share. The algorithm is smaller than BACON.1 due to this prior knowledge.

4.2.1 Method

Given symbols X and Y with datasets $d(X)$ and $d(Y)$ and a basic understanding of their relationship, for example[2], $Y = \alpha X^2 + \beta X + \gamma$. BACON.6 iterates through a set of possible values for α and β (γ – as a constant – is ignored at this stage as it is combined with another constant after the algorithm is done), saving those with the greatest correlation coefficient. The hyperparameters are an initial N which can be any positive number, an $n_{\text{threshold}}$ which is set to 2, and a p which determines how many iterations the process performs. It is as follows:

1. Initialise 9 instances of $\alpha X^2 + \beta X$ with α and β taking values of all 9 possible pairs $\in \{-N, 0, N\}$. These are $(-NX^2 - NX), (-NX^2), (-NX^2 + NX), ...$
2. Calculate the correlation coefficient of $d(Y)$ and all 9 combinations above with $\alpha d(X)^2 + \beta d(X)$. Retain the α and β corresponding to the greatest $n_{\text{threshold}}$ of them. This measures which combinations fits the data best.
3. Initialise 9 instances of $\alpha X^2 + \beta X$ with α and β taking values of all 9 possible pairs $\in \{-\frac{N}{2}, 0, \frac{N}{2}\}$. Add the retained α and β to each pair, creating 18 combinations each an addition of a retained expression and a new combination.
4. Calculate the correlation coefficient of $d(Y)$ and all of these 18 combinations as before. Retain the best $n_{\text{threshold}}$ of them.
5. Step (3) and (4) are repeated, each time halving the N that the new combinations instantiate from. The program terminates when $N < \frac{1}{2^p}$. The output is the α and β which gives the greatest correlation coefficient.

We now have a linear relation between Y and $\alpha X^2 + \beta X$. `polyfit` from NumPy is used to perform linear regression which gives the optimal gradient m and y-intercept c to finalise the exact relationship. Including additional variable γ from earlier, would increase the algorithm time unnecessarily as it still would have combined with c at this stage.

[2]The RHS of the example can contain any type of function such as trigonometric or exponential or even those that contain Y. It is set as a polynomial here for simplicity.

4.2.2 Strengths of BACON.6

When the form is known, and the data is noiseless, BACON.6 is better on minimal datapoints than PySR. This is demonstrated through the Birthday Problem: in a year of $n \geq 1$ days, what is the minimal number of people in a room such that the probability of at least two sharing a birthday is over 50%. This is extrapolated to, assuming each person enters the room sequentially, what is the expected number of people to enter, for a birthday to first be shared. Ramanujan discovered the answer is $\lfloor Q(n) + 1 \rfloor$ [Brink, 2012], where

$$Q(n) = \sum_{k=1}^{n} \frac{n!}{(n-k)!\ n^k} \tag{4.3}$$

$$= \sqrt{\frac{\pi n}{2}} - \frac{1}{3} + \frac{1}{12}\sqrt{\frac{\pi}{2n}} - \frac{4}{135n} + \mathcal{O}\left(\frac{1}{n^{\frac{3}{2}}}\right) \tag{4.4}$$

$$\approx 1.253\sqrt{n} - 0.333 + \frac{0.104}{\sqrt{n}} - \frac{0.030}{n} \tag{4.5}$$

When given an appropriate answer form without coefficients to BACON.6[3] with datapoints for $n \in \{2, 3, 4, 5, 6, 7, 8, 9\}$ and associated $Q(n)$, it discovers that:

$$Q_{\texttt{BACON.6}} = 1.252\sqrt{n} - 0.326 + \frac{0.086}{\sqrt{n}} - \frac{0.012}{n} \tag{4.7}$$

The MSE between $Q_{\texttt{BACON.6}}$ and the real $Q(n)$ is 5.4×10^{-9}. PySR comparatively struggles. Given the same data, and the *binary_ operators* from Figure 3.1 expanded to include powers, the best result it finds is:

$$Q_{\texttt{PySR}} = n^{0.557} + \frac{0.324^{\frac{5.28}{n}}}{n} \tag{4.8}$$

This has MSE 3.8×10^{-5} to the true form - 4 orders of magnitude worse than BACON.6. A possible reason, and seen throughout Section 5.3 is that PySR doesn't function well on few datapoints. However, this demonstrates a strength by BACON.6 to be accurate in an environment the state-of-the-art blunders in, that is more complicated than the typical linear and product/division relationships seen so far. This is our first display of Classical AI techniques, outperforming the state-of-the-art.

Langley never incorporated BACON.6 in the wider BACON search tree with BACON.3 or BACON.5. He only detailed a small number of instances of BACON.6 solving examples like this. His main example was the less convoluted $Y = 3X^2 + 2X + 1$. Hence, the fact my build of BACON.6 can solve the comparatively complicated birthday problem[4] displays a faithful reconstruction of Langley's work. Moreover, I have implemented BACON.6 into

[3]The form is:

$$nQ(n) = \alpha n^{\frac{3}{2}} + \beta n + \gamma n^{\frac{1}{2}} \tag{4.6}$$

[4]I did confirm this reconstruction could solve $Y = 3X^2 + 2X + 1$. However, as this is also solved by PySR it doesn't demonstrate Classical AI outperforming a DNN like the birthday problem does.

the search tree via using a general form for BACON.6 to solve on - something Langley never did. For coefficients α, β, γ and symbols X, Y the form is:

$$Y = \alpha X + \frac{\beta}{\gamma + X} \tag{4.9}$$

However, it ultimately wasn't efficient for reasons detailed in subsection 4.2.3.

4.2.3 Weaknesses of BACON.6

Adding more variables, and increasing the generality of the form comes at an expensive time and computational complexity trade-off. Recall in the method 9 combinations of α and β are generated each step. This extrapolates to 3^n combinations being formed for n the number of variables. Completing the correlation calculations then takes exponentially longer with each additional variable, as there is $\mathcal{O}(3^n)$ computational complexity. 3 variables was found to be the maximum for BACON.6 to solve in good-time. Even then it took $4\times$ longer than BACON.1. This makes BACON too restrictive in what it can solve. For example, it couldn't solve Ohm's Law as it involves a squared term for D which corresponds to a square for Y in equation 4.9.

The reason for having to use a generalised form is symptomatic of the true issue. Having to input a predicted form of the equation goes against the premise of BACON inferring everything from the data. The user often will have no knowledge of the data they're using rather just a dataset.

Another issue is that Langley's claim that BACON.6 is especially adept at handling noise doesn't seem true. For example, even at 0.5% relative noise, the found equation from the birthday problem distorts to:

$$Q_{\texttt{BACON.6}} = 0.968\sqrt{n} + 1.307 - \frac{2.904}{\sqrt{n}} + \frac{1.732}{n} \tag{4.10}$$

A form dissimilar to equation 4.7, with an MSE of 0.002 to the noiseless Q - 7 orders of magnitude worse. Results shown in subsection 5.3 tended to only get a single order of magnitude worse with every 0.5% noise added to the dataset. This is then anomalous in how poor the performance is. I suspect this is as BACON.6 overfits to the noisy data through the use of α and β coefficients being too precise.

In all, BACON.6 is slower and limited in the forms it can find compared to BACON.1. It has strengths in noiseless environments but can't adapt to noise well enough to be used in my main model BACON.7 (in subsection 4.4). Langley's other methods didn't handle noise well, so with perspective to those - and the fact a results is always given (via the coefficients), this may have given him cause to praise BACON.6's error tolerance.

4.2.4 Relation to Langley's BACON.6

The program was implemented exactly as described in [Langley et al., 1987a] with the addition of the parameter p. Langley used a fixed threshold to stop the iterations.

4.3 The Space of Data

Langley designed both BACON.3 and BACON.5 as ways to traverse a multivariable dataset. Both were flawed when dealing with noisy datasets. As such, I've designed my own novel methodology to explore the space. It uses BACON.3's idea of traversing layer-by-layer, and BACON.5's idea of symmetry.

This section will describe how I have approached this problem, my solution and comparisons with Langley's versions as well as why they are flawed. It also discusses why BACON.5 may be the best method on large datasets.

4.3.1 Method

At each layer in the tree, a 3-step process is followed:

1. Check if the layer is sufficiently constant. If it is, terminate the tree with the symbol representing the layer. At the last layer, this check is not run.
2. If it is not sufficiently constant, calculate the expression for each set in the layer by inputting the set into BACON.1 or BACON.6. Use a layer-method to determine which expressions is chosen.
3. Average the layer to form a new layer. This is crucial in BACON's ability to determine the true equation in noisy environments as displayed in subsection 5.3.

Sufficiently constant
Similar to δ earlier, parameter Δ is introduced to prune a tree if the bottom layer is sufficiently constant. If the bottom layer has symbol Y and datapoints $d(Y)$. For $M_Y = \text{mean}(d(Y))$, if $(1-\Delta)$ proportion of the values[5] in $d\,(Y)$ satisfy

$$M_Y(1-\Delta) < \text{value} < M_Y(1+\Delta) \tag{4.11}$$

then the invariant is determined as $Y = M_Y$. The comparison between Y and other variables is skipped. Throughout the remainder of the project $\Delta = 0.01$ but is user-configurable. It saves time. For example, in Ohm's Law, the invariant $IL - aI$ is discovered in the first layer. A new tree with a uses this test to determine $a = 3$ without comparing to other variables T and D.

[5]Eg. if $\Delta = 0.02$, then 98% of the values in $d\,(Y)$ must satisfy the Δ relationship above. This allows it to deal with extremely noisy points due to variance in sampling noise from a Gaussian distribution. Without this, the test rarely passes due to these outliers.

Layer-methods

I have devised various methods to pick an expression from the list of possible expressions. It was one of the main novel contributions to improve `BACON`. They are listed in Table 4.1 with their name, an explanation and the time taken to operate on the first layer of Black's Law with 0.5% noise. This example has 81 datapoints and 27 sets leading to 2 expressions which have to be ranked.

Method	Time	Description
`bacon.3`	3×10^{-6}s	Requires the sets to all find the same expression. If this does not happen the program terminates.
`popular`	7×10^{-6}s	Selects the expression which appears most frequently - if there is a tie, defer to `min_mse`.
`min_mse`	0.37s	Apply an expression to a set to gain 3 datapoints. Normalise the datapoints. Take the mean squared error (MSE) between the normalised datapoints and their mean[6]. Sum over each set in the layer to get a total MSE for the expression. Do this for all expressions, the expression with the lowest total MSE fits the dataset best and is selected.
`gp_ranking`	0.85s	Apply an expression to all the sets. Gaussian Process (GP) Regression is performed on this new dataset to find the optimal GP that fits the data. The GP is sampled, and the signal-noise-ratio of the sample calculated. This done for each expression. The highest score implies the least noise, and best fit of the expression to the data. This expression is selected.
`user_input`	N/A	The user selects the expression they'd like to use from those found by the sets. Used for debugging.

Table 4.1: A table displaying the custom layer-methods I devised as well as and explanation of how they select an expression. The time they each take to rank a layer with 81 datapoints and 2 found expressions is also given.

`bacon.3` and `popular` are fastest as a dataset doesn't have to be reconstructed with the found expression applied to the last layer. This has to happen for `min_mse` and `gp_ranking`. `min_mse` takes less than half the time as `gp_ranking` due to the complexity in performing GP regression. They also gave near identical results in choosing the correct expression. It displays that black-box techniques aren't necessarily better than Classical techniques especially on simple tasks like calculating MSEs. `popular` and `min_mse` were used throughout the results in subsection 5.3.

[6]For found variable X with dataset $\{x_i : i \in 1, ..., n\}$ and mean M_X, the MSE is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - M_X)^2 \tag{4.12}$$

Averaging

From Figure 2.1 we see the tree gets smaller at each iteration of the tree. Once an expression has been determined via the layer-method it is applied to each set. Then in the set, there are 3 datapoints from the expression which are averaged. This becomes a new datapoint, reducing the overall number of datapoints by 3. For example, the first set in the first layer in Figure 2.1, has $T = 10,\ M = 1$ and are the 3 points defined by $(P = 1000,\ V = 0.28)$, $(P = 2000,\ V = 0.14)$ and $(P = 3000,\ V = 0.09)$. PV is determined and the datapoints found are $PV = 280,\ 280,\ 270$[7], this would be averaged to $PV = 277$. Then the new tree which has PV at the bottom layer would have $PV = 277$ for the datapoint with $T = 10, M = 1$. I hypothesise, that this averaging places a crucial role in `BACON` stripping away noise. This is elaborated on in subsection 4.4.

4.3.2 Why are multiple methods needed?

Consider the Ideal Gas Law at 3% relative noise with the hyperparameters found in args/ideal/ideal30.json. A run through the tree with the `popular` layer selected looks like:

1. In the first layer the sufficiently constant check is not passed by V. Instead expressions are found, and the 9 sets determine: PV invariant 7 times and $PV^2 - aV$ and $V - aP$ are both found once. PV is carried forwards by `popular` layer and applied universally to the dataframe. The sets are averaged reducing the dataframe size.

2. The sufficiently constant check is not passed by PV. Then, between PV and T, the 3 sets find the relation PV, $PV - aT$ and $\frac{T}{PV} - aT$. As these all appear once, we defer to the MSE calculations. These are respectively 0.0043, 0.0027, 0.0032 causing $PV - aT$ to be used. Thus this alternative MSE mechanism is needed to decide. The sets then average once again.

3. As the tree is on the final layer, the sufficiently constant test is not run. Between $PV - aT$ and M the single relationship found is $\frac{M}{PV-aT} = 0.0036$. Between a and M the relation is $\frac{M}{a} = 1.04$, altogether yielding $V = \frac{M(0.96T+273.57)}{P}$ which best fits this noisy data.

In general `min_mse` works best with little amounts of noise and as a tiebreaker for `popular`. Once the data gets noisier `min_mse` ends up getting tricked. In the example above at the first layer, the MSE for PV is 0.006, $PV^2 - aV$ is 0.004 and $V - aP$ is 0.036. $PV^2 - aV$ would have got picked, which would stop the true equation being formed. This happens as $PV^2 - aV = \text{const} \implies PV = a + \frac{\text{const}}{V}$. For V significantly larger than const, the RHS is approximately constant. This is another demonstration of the need for c_{val} from

[7] In Figure 2.1 the dataset noiseless, and so with the full decimal expansion of V, PV would be 283. However, this is a demonstration of what happens when there's noise.

subsection 4.1.1. Increasing it to 0.1 from 0.02 stops $PV^2 - aV$ becoming a valid invariant and thus retains the ability to use the `min_mse` method.

4.3.3 Weaknesses of layer-methods

Both `popular` and `min_mse` have their flaws. Experimentally, it is easy (on multiple repeats) to find a set of hyperparameters where the real equation can be determined if the correct invariants are chosen at each stage (done by `user_input`). However, once the data becomes noisier, it is often the case it is neither the most popular equation, nor satisfying `min_mse`. In addition, with this noise, different layers are solvable by different layer-method which so far has been too difficult to implement in a generalised manner.

The other issue is that the order of the layers matter. Consider Black's law again:

$$T_f = \frac{M_1 T_1}{M_1 + M_2} + \frac{M_2 T_2}{M_1 + M_2} \tag{4.13}$$

Initially between T_f and T_2 there is invariant $T_f - aT_2$. If instead the first layer was between T_f and M_1 the equation rearranges to $T_f M_1 + T_f M_2 - M_1 T_1 = M_2 T_2$, leading to invariant $T_f M_1 - aM_1 - bT_f$ which isn't solvable for reasons explained in subsection 4.1.4. However, a brute-force approach could be generated to keep varying the input order until a form that is solvable is entered. This is a potential extension to `BACON`, and similar to what `Fahrenheit` – mentioned earlier in subsection 3.1 – performed.

4.3.4 Relation to Langley's `BACON.3` and `BACON.5`

Langley's `BACON.3` required ubiquity along the expressions found by the sets. One of my layer-methods works along this premise, named `bacon.3`. With noise, the sets don't always agree, and with more noise it becomes more likely they don't agree. This limits its use.

`BACON.5` introduced symmetry. At each level only a single set is needed to form an expression which can then be applied to the sets in the layer by symmetry as they all share the same relation. This directly inspired me to gain all the expressions, with the best (determined by the layer-method) applied to the rest of the sets in the layer.

Langley's method required picking the datapoints that are used to produce the sets before any calculations were made. There are many ways to do this. For example, Figure 4.6 demonstrates possible choices for a dataset with 3 variables and 9 datapoints assuming that the set is the first 3 datapoints from the bottom left. There are 3 possible configurations for this initial set, giving a total of 27 possible index choices.

Extrapolating for an $n+1$ variable system there are $3^{n(n-2)}$ ways to do this (shown in Appendix A.2). This is an exponentially increasing choice. If made correctly it improves the likelihood of `BACON.5` leading to the correct expressions. This method also reduces

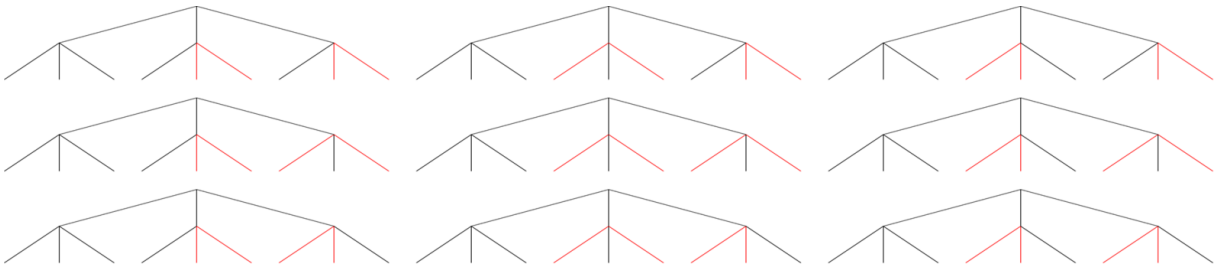


Figure 4.6: Possible ways to pick datapoints in `BACON.5` when the set needed is in the bottom left. The chosen datapoints are in black. As the initial set could also be from the centre or right branch, there's a total of 27 possible ways to choose.

the datapoints needed. Now the program only needs to consider $3 + 2(n-2) = 2n-1$ datapoints. This is significant in very large datasets as it requires small amounts of perfect data. `BACON` also performs better in smaller datasets and being able to create this environment is desired. If an optimal method could be discovered for picking branches, `BACON.5` would be effective on large datasets.

I could not find a method for this that works quickly and efficiently. Langley made this in his noiseless environment where datapoint-picking was arbitrary. With noise this is not the case. Multiple datapoints are significantly worse due to random Gaussian noise that picking them at any stage is tantamount to failure.

The only mechanism I can think of is using a Gaussian Process. We rank each datapoint by fitting a GP to the dataset with that datapoint removed, then calculating the signal-noise-ratio of a sample from that GP. The datapoints corresponding to the GPs with the largest signal-to-noise ratio are noisiest, and should not be picked. Doing this for all 3^n datapoints is time-consuming, computationally complex and goes against the principles of `BACON` as established in section 2.1. However, this does show a situation where Classical AI and modern black-box techniques could combine.

4.4 `BACON.7`

4.4.1 Overview

`BACON.7` is the name I have given for my version of `BACON`. It uses my layer-method design as described in section 4.3. It uses `BACON.1` as the Space of Laws.

At the Space of Laws level, `BACON.6` with a general equation form and an implementation of `PySR` were tested. The latter struggled consistently as 3 datapoints is not enough for it to function efficiently - a feature seen repeatably through the results in subsection 5.3. `BACON.6` tends to overfit on noise, as well as taking too long when used with a general form, both discussed in subsection 4.2.3. However, there is a question: if `BACON.1` can generate a relationship, can `BACON.6` improve the found constants in said relationship? This is tested in subsection 4.4.2.

I did create BACON.5 and the modular way I've coded the entire project allows it to be selected. However, as the initial datapoints picked are still random, there's not the reproducability or consistency offered by my layer-methods. It is not used going forwards.

The project was coded with an eye on understandability and clarity. The system had to be straightforward and reproducible. Details on how this was done are in Appendix A.1.

4.4.2 Combining BACON.1 and BACON.6

This is tested on the Ideal Gas Law. On the conclusion of the tree-search, and as elaborated on in subsection 5.1.1, the equations found are:

$$\frac{M}{a} = 1, \quad \frac{M}{PV - aT} = \frac{1}{273} \tag{4.14}$$

Reworking the RHS constants as variables α and β respectively, the following is gained:

$$V = \frac{M}{P}\left(\frac{T}{\alpha} + \frac{1}{\beta}\right) \tag{4.15}$$

This is the ideal form to input into BACON.6. Evaluating on the Ideal Gas Law with 3% noise without BACON.6 warrants the form

$$V_{\texttt{BACON.7}} = \frac{M}{P}\left(0.96T + 273.57\right) \tag{4.16}$$

with MSE of 1.6×10^{-4} to $V+n$ (the noisy data, detailed in section 5.1.4). Comparatively, BACON.6 integration gives form

$$V_{\texttt{BACON.7}} = \frac{M}{P}\left(1.02T + 280.18\right) \tag{4.17}$$

with MSE of 1.4×10^{-4} to $V + n$. When comparing instead with noiseless V, the MSE difference now becomes 3.4×10^{-7} using BACON.1 but 1.2×10^{-4} with BACON.6. Hence, BACON.1 is four orders of magnitude more accurate without BACON.6. As this is a measure against the objective true function, BACON.1 is unequivocally better at stripping noise from the equation, whereas BACON.6 overfits on the noisy data. This is a trend similarly seen with PySR in the results, subsection 5.3.

4.4.3 Weaknesses of BACON.7

BACON.7 is unable to work at variables that scale orders of magnitude apart. For example, the SIR model with variables that vary at scales of 10^{-8}, 1 and 10^{6} could not be used in BACON. This is because it causes the hyperparameters inputted into BACON.1 to vary between layers preventing the correct expressions to be found. For instance, the first layer may need $c_{\text{val}} = 1$, but the next needs $c_{\text{val}} = 1 \times 10^{-6}$. A future improvement to BACON involves devising a mechanism to vary hyperparameters between layers smartly.

Chapter 5

Evaluation and Results

5.1 Datasets

The datasets used for this evaluation are synthetically generated from three well-known equations described below. In each equation set, the values for the independent variables were based on the values Langley displayed in his book [Langley et al., 1987b].

5.1.1 The Ideal Gas Law

The Ideal Gas Law is:

$$V = \frac{M(T + 273)}{P} \tag{5.1}$$

V is volume, P is pressure, M is moles and T is temperature (in this basis the gas content R is 1). M is $\{1, 2, 3\}$, T is $\{10, 20, 30\}$ and P is $\{1000, 2000, 3000\}$. This causes V to vary between 0.09 and 0.91. The ideal gas law has a linear relationship in the numerator alongside a couple of instances of product/division meaning its a useful baseline test. When approached via consecutively introducing variables $V \rightarrow P \rightarrow T \rightarrow M$ (forming the tree in Figure 2.1) noiselessly, two equations come out of the system:

$$\frac{M}{a} = 1, \quad \frac{M}{PV - aT} = \frac{1}{273} \tag{5.2}$$

These are combined via dummy variable a to form the Ideal Gas Law.

5.1.2 Ohm's Law

Ohm's law is typically seen as $I = \frac{V}{R}$, for V voltage, I current and R internal resistance. An expanded form when considering the law applied to a bar of temperature T, diameter D and length L is:

$$I = \frac{TD^2}{2(L + 3)} \tag{5.3}$$

T is $\{100, 120, 140\}$, D is $\{0.01, 0.02, 0.03\}$, L is $\{0.5, 1, 1.5\}$ whilst 2 and 3 represent the fixed voltage and resistance in the wire respectively. I varies between 0.001 and 0.018. The law contain linear relation in the denominator as well as a square through D. It thus tests many facets of the `BACON.1` equation finding system. The system is approached via $I \to V \to D \to T$ yielding equations in a noiseless environment of

$$a = -3, \quad \frac{TD^2}{I(L-a)} = 2 \tag{5.4}$$

combining along a.

5.1.3 Black's Law

Black's law as seen is:

$$T_f = \frac{M_1 T_1}{M_1 + M_2} + \frac{M_2 T_2}{M_1 + M_2} \tag{5.5}$$

T_1 and T_2 are temperatures both taking values $\{50, 60, 70\}$ whilst M_1 and M_2 are masses taking values $\{1, 2, 3\}$. T_f is also a temperature varying between 50 and 70. This is the most involved system solvable by Langley's `BACON` with 3 linear relationships inside it when approached via $T_f \to T_2 \to T_1 \to M_2 \to M_1$. The equations found in a noiseless setup are:

$$b = 1, \quad \frac{M_1}{\frac{M_2}{a} - bM_2} = 1, \quad cM_1 = 1, \quad -cM_2 - \frac{T_1}{T_f - aT_2} = 1 \tag{5.6}$$

which are combined along variables a, b, c.

5.1.4 Noise

The experiments were run with 0.5% increments in the noise on the dependent variable until 4%. This means for dependent variable D with values $\{d_1, d_2, ..., d_n\}$, the Gaussian noise added to the scalar d_i is $\epsilon_i \sim \mathcal{N}\left(0, (0.04\,|d_i|)^2\right)$. This new noisy variable is denoted $D + n$.

For reproducibility, the noise added to the dataset is using a NumPy random seed. For repeatability all the argument files used are saved, with the location prescribed to the name of the law. Eg. the Ideal Gas Law run under 0.5% noise has its argument file stored in args/ideal/ideal05json.

5.2 Criterion for success

Langley's approach to testing prescribes that `BACON` can find an invariant if one combination of the hyperparameters inputted into the system allows the detection of the invariant. I used this approach. A similar metric is used for determining if `PySR` can find the equa-

tion; the correct form has to appear in its list of found equations. For α, β, γ constants the corrects forms to be found are:

$$V_{\texttt{BACON}} = \frac{M(\alpha T + \beta)}{P}, \quad I_{\texttt{BACON}} = \frac{\alpha T D^2}{(L + \beta)}, \quad T_{f_{\texttt{BACON}}} = \frac{\alpha M_1 T_1 + \beta M_2 T_2}{\gamma M_1 + M_2} \tag{5.7}$$

$V_{\texttt{BACON}}$ are BACON's predictions for V when fed the noisy dataset. $V_{\texttt{PySR}}$ is correspondingly used later for PySR's predictions.

When calculating the MSE between the predictions of BACON and PySR with the noisy data, the following notation is used: MSE($V_{\texttt{BACON}}$, $V + n$). This is a measure of how well the model fits the training data. Correspondingly, MSE($V_{\texttt{BACON}}$, V) is the difference between the predicted form of V and the actual noiseless V - a measure of how close BACON gets at stripping the noise away and determining the real equation. This is the most important factor in establishing if a model is successful.

5.3 Technical Results

The BACON model used for the following tests is my BACON.7 described in subsection 4.4.

The Ideal Gas Law

Figure 5.1 is a graph of the MSE differences between the equations that BACON.7 and PySR predict from inputted noisy data ($V + n$), against the true data (V) and the noisy data. There's a seemingly linear increase in the log of the MSE found as noise increases

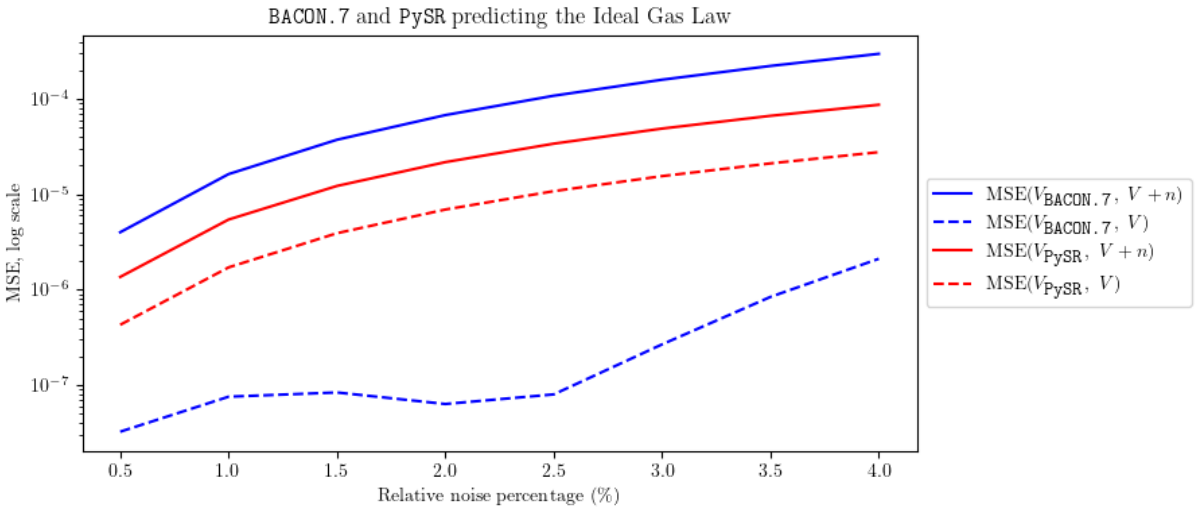


Figure 5.1: For $V = \frac{M(T+273)}{P}$ and denoting the volume with noise added as $V + n$, the graph demonstrates the MSE between predicted models from BACON.7 and PySR with $V + n$ and V. The MSE with $V + n$ is how well the model predicts the data it is trained on (solid lines). The MSE with V describes how it predicts the true data (dashed lines). In each case the model is better at predicting the true data *with* BACON.7 *creating a model an order of magnitude more accurate than* PySR . I hypothesise this is due to BACON.7's averaging through the Space of Data incidentally cancelling Gaussian noise, whereas PySR is trained to overfit on the data as it has no prior knowledge of there being noise. The latter explains as well why PySR displays a better model when compared with $V + n$ than BACON.

in all aspects. On the noisy data BACON.7 is less accurate than PySR. It implies that PySR

overfits on the dataset. This is verified when comparing $V_{\texttt{PySR}}$ with $V + n$ where its model is more accurate than BACON.7. BACON.7 does a better job of deducing the true V in a noisy environment. Moreover, it does this consistently over all the noise percentages tested on. *On this dataset* BACON.7 *outperforms* PySR *by creating a model that is an order of magnitude more accurate through its superior ability to remove the noise.*

Ohm's Law

The results on Ohm's laws are displayed in Figure 5.2. They display similar trends for BACON.7. PySR could not find the correct equation form apart from for 0% noise so instead the results from the best prediction PySR returns is shown. PySR is consistently at least

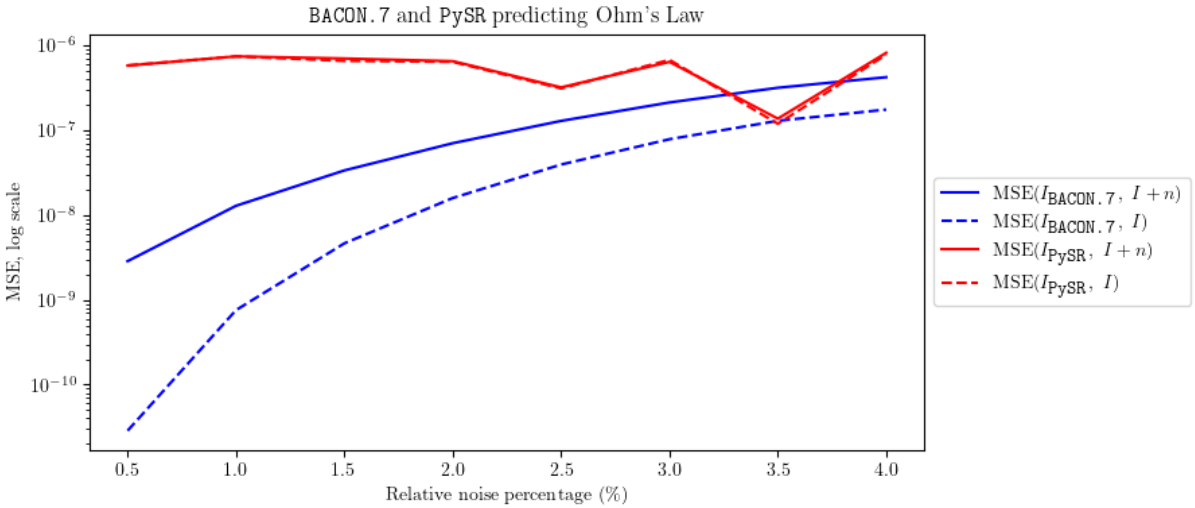


Figure 5.2: The results for Ohm's Law when using the same methodology as the Ideal Gas Law in Figure 5.1. BACON.7 again is better at predicting the I than $I + n$. Here PySR is not able to deduce the correct form of Ohm's Law so the results are taken from the best prediction it gives. In almost each case PySR*'s predictions are worse than* BACON.7*'s against both the noisy and true data* . The likely reason is that the correct form of the equation cannot be inferred by PySR from only 27 datapoints as PySR was built to function efficiently on larger, more complicated datasets. BACON.7 works best at this scale, where its strips out noise through averaging.

an order of magnitude worse in accuracy, than BACON.7 on I. Also, the difference between the noisy and noiseless forms for PySR is imperceptible. BACON.7 finds the correct form and as before, improves when run against the noiseless data. BACON.7 *again produces a model that strips away the noise more accurately than* PySR. *Moreover* PySR *cannot even deduce the correct form of the equation.*

Black's Law

Black's law varies on a non-log scale. There is exponential-like growth in the MSEs in Figure 5.3 indicating it is growing at a similar pace to before. *No results could be found to make* BACON.7 *work at 2% noise or above. Comparatively,* PySR *finds the exact form of the equation at every noise increment.* This seems to be the optimal environment for PySR; the form of the equation does help its cause. Each coefficient being 1 aligns with the simplistic aims of its final results as discussed in section 3.2.

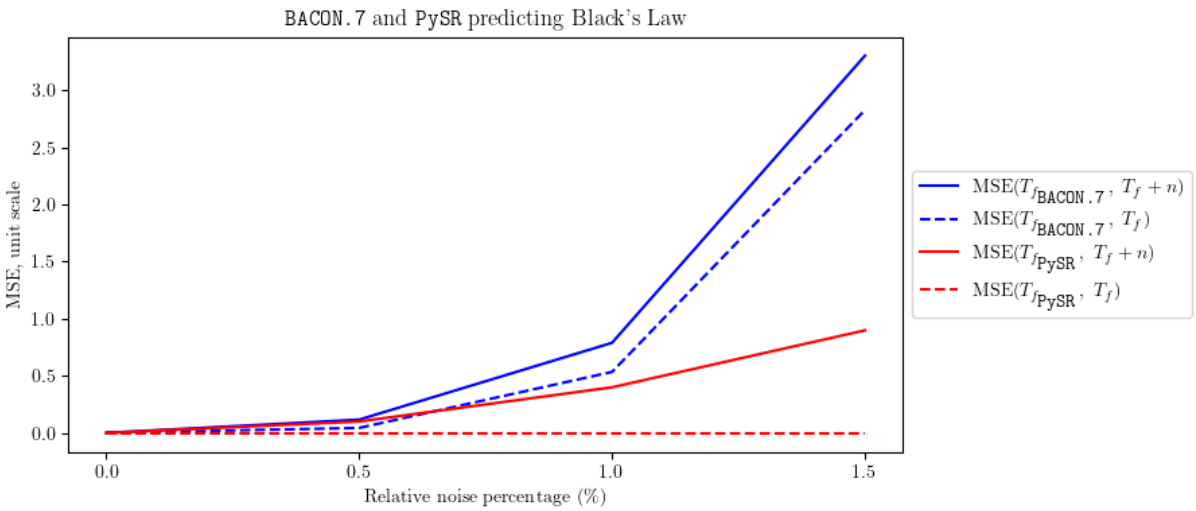


Figure 5.3: The MSE differences when the above tests are run on Black's Law. PySR outperforms BACON.7. In each case it infers the exact form of Black's Law using its simplistic biases as described in subsection 3.2. Additional testing shows it finding this up to 4% relative noise. *This demonstrates the superiority of neural network models, here* PySR*, when fed enough data.* BACON.7 can't solve greater than 1.5% due to increased noise in the dataset. With 81 values having noise applied, there's a higher likelihood of excessive noise (by random Gaussian sampling) in the datapoints. Even with averaging the noise appears in the last layer, where no combination of the hyperparameters can attain the correct relationship without contradicting previously found expressions in earlier layers.

5.4 Discussion

On the Ideal Gas Law

Analysing the equations returned allows a better of understanding of how BACON.7 outperforms PySR. For instance, at 2.5% noise, the form of BACON.7 comes from the two relations in subsection 5.1.1, the first discovers $\frac{M}{a} = 1.024$ and $\frac{M}{PV-aT} = 0.00366$. The constants come from averaging the values found in each step as discussed in subsection 4.3.1. This averaging eliminates much of the noise. Overall displaying final form $V = \frac{M(0.976T+273.488)}{P}$. Comparatively PySR only finds $V = \frac{M(T+270.74)}{P}$. This is because its simplistic aims force the T coefficient 1 - though this is a benefit in experiments on Black's Law. It is not clear why the other coefficient is 270.74. The model gives no explanation.

On Ohm's Law

I don't know why PySR can't find the correct form. For reference at 2.5% noise it finds $I_{\texttt{PySR}} = D^2\left(-L + 0.127T + \frac{0.905}{L}\right)$. As the correct form is not found for any of the noise percentages, it indicates a systematic problem rather than a fluke. Instead of reducing noise, it stays in the model and this is only known as we know the true form of I. This demonstrates the problem with picking the wrong equation in CSD. Also, the lack of explainability for PySR means I don't know what settings to tweak to fix this issue.

This environment demonstrates that BACON.7 has utility when compared to PySR at this scale. This and the Ideal Gas Law, are on smaller datasets. These are optimum conditions for BACON.7, whilst not giving enough datapoints for PySR to perform effectively. At this scale BACON.7 both strips away noise, and infers the true equation form better than PySR.

On Black's Law

BACON fails as there's a conflict in the hyperparameters to get the correct equations at the final layer in the tree. At 2% the final layer of one of the trees finds $1.74 = 0.296M_1 - cM_2 - \frac{T_1}{T_f - aT_2}$ instead of $1 = -cM_2 - \frac{T_1}{T_f - aT_2}$. To punish this excess linear relationship, ϵ needs to be reduced. However, all attempts there cause the linear relationship for c at the layer above not to form. This contradiction means there's no way of pulling the correct form out of BACON.7.

A reason for this is at previous steps you can choose between multiple expressions as there are multiple sets. As there is only one at the final layer the ability is lost and takes the only expression displayed. A potential improvement of BACON involves more variability at the final step. This is something that I input into MCTS in section 6.

Comparison to Langley's BACON

Langley's BACON – which uses bacon.3[1] as a layer-method as well as BACON.1 – could not solve any noise on any dataset, bar 0.5% on the Ideal Gas Law[2]. As Langley's BACON worked on 0% noise it demonstrated that they had been reconstructed correctly. *Thus, my* BACON.7 *has been successful in becoming more noise-resilient than Langley's* BACON.

Concluding Comments

For real-life applications the true equation of a dataset will not be known. It is then imperative to be able to trust the equation generated by the model. *For smaller datasets, these results have conclusively shown* BACON.7 *is more trustworthy than* PySR *as it is more accurate due to its superior ability to strip away noise.* PySR is best on larger, noisier datasets. In these environments, it outperforms BACON.7.

A situation has been found where classical methods outperform modern techniques. It amplifies the need to reproduce, study and understand these seemingly anachronistic mechanisms and see what lessons can be learnt. Also of note, PySR is a complete package, built in collaboration with many engineers over multiple years with well-written documentation and open-source support. BACON.7 was individually built in 7 months with only a 1980s' textbook to guide. With further development, it is possible more use-cases will be found where BACON.7 is able to compete against the state-of-the-art.

An additional note on explainability. I only managed to find the hyperparameters to run the tests up to 4% noise after much trial and error. However, it wasn't luck. I could read the output to BACON.7, understand at what stage problems occurred (such as making a linear relationship when it should have been a product) then fix the appropriate hyperparameters. *This demonstrates how explainable models help aid in development.*

[1] Whilst it was possible to test against BACON.5, Langley's never specified a mechanism for picking initial datapoints. My reconstruction does this randomly and is not reproducible. Thus it was not chosen as the baseline for Langley's BACON.

[2] Saved under args/ideal/idealbacon.json. Likewise the best attempts for Ohm's and Black's law are saved in their respective folders. These only succeed on 0% noise.

Chapter 6

Monte Carlo Tree Search

Multiple layer-methods were created as no method consistently chooses the correct expression when data is noisy. When using the `user_input` method, it seemed each expression did exist, at least until the final layer. Here, we saw from Black's Law in subsection 5.3 that the ϵ and δ needed varied from earlier layers. This section uses MCTS to optimise the search through the possible expressions and solve both these issues.

6.1 Method

To use MCTS we need the concept of a node. In chess, this is represented as the position of the pieces on a board at a given go. For `BACON` we represent it by the expressions found at a given layer. To move to the next node, the legal actions must be determined. In chess, these are the set of legal moves. In `BACON` these are the set of expressions to choose from, apart from at the final layer where it is a selection of hyperparameters to use in `BACON.1`. MCTS terminates when the game is over. In chess that would be at checkmate or stalemate, for `BACON` that would be when each layer of the tree has been searched. Lastly, scoring. In chess, it is +1 for a win, 0 for a draw or -1 for a loss. For `BACON` it is a function of MSE($V_{\texttt{BACON}}$, $V + n$) (referred to as the reward function).

The MCTS algorithm works through 4 steps after being initialised at a given node called a parent node:

1. *SELECTION*: A child node is selected to travel from those available to the parent node (through performing a legal action) based on the score of the child node. The score is weighted by a parameter C. If C is small, the parent node prefers exploitation and picks a child node based on what has returned high scores in the past. If C is larger, the parent prefers exploration and picks a child node which has not been selected much.

2. *EXPANSION*: After the parent chooses a child, this child becomes the parent node. The legal actions are run to determine the new selection of children nodes.

3. *SIMULATION*: This process is repeated until a full game has been simulated. The result of the game is given a score by the reward function.

4. *BACKPROPAGATION*: This score is backpropagated through all the nodes selected to the initial parent node, altering their S_i value.

The purpose of the MCTS is to find the node with the best score after running this process multiple times. It implies this node is the correct one to move to, to maximise winning. In `BACON` this translates to picking the best expression in a layer that maximises the score. Under an assumption that the best expression in earlier layers will always minimise the MSE (and maximise the score), MCTS is a logical continuation that can search the possible expressions generated by the sets and correctly pick the best expression at each layer. A visualisation of this on the Ideal Gas Law is in Figure 6.1.

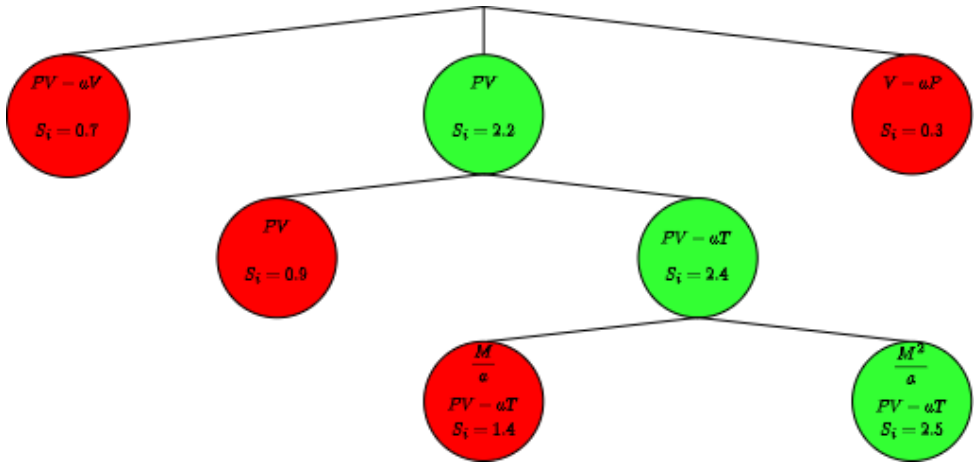


Figure 6.1: The graphical representation of the MCTS on the low C for the Ideal Gas Law in an environment with 5% noise. The scores at each stage are demonstrated with S_i, with the child node selected in green. Note the green node's S_i increases through the layers as MCTS prunes the worst nodes. For the first two layers the action is picking an equation from those discovered by the sets at that layer. The bottom layer is determined by applying a combination of the following hyperparameters $\epsilon, \delta \in \{(0.01, 0.1), (0.05, 0.5), (0.01, 0), (0, 0.1)\}$ and gaining their respective equations.

My reward function takes in multiple factors, such as how many variables make up the expression, how quick it took `BACON` to output the expression as well as the MSE. It is not a continuous function and had to be hardcoded for each equation that the MCTS was implemented on. It's hard to determine a general function due to MSE differences varying on different scales between equations.

`BACON.M` is used to refer to the model that comes from this conjunction with MCTS. It uses `BACON.1` at the Space of Laws level, and MCTS to repeatedly traverse the Space of Data.

6.2 Application

Due to time constraints, the system was only adapted for two equations: the Ideal Gas Law and Black's Law. It is simple to adapt for Ohm's Law, however from the previous

section it seems likely to perform as the Ideal Gas Law does.

6.2.1 The Ideal Gas Law

The Ideal Gas Law could handle up to 4% noise. MCTS is done on 5% noise at a low C - prioritising exploitation. MCTS outputs `BACON.M` expression:

$$V_{\texttt{BACON.M}} = \frac{M(0.502MT + 273.885)}{P} \tag{6.1}$$

This occurs from the final layer deducing $\frac{M^2}{a}$ as a constant rather than $\frac{M}{a}$. The MSE to $V + n$ is 2.9×10^{-4} and to V is 2.0×10^{-4}. Recalling the graph from Figure 5.1, this is inline with the predictions for extrapolating to 5% noise in the correct form. It displays that, at this noise level, this form and the correct form share the same MSE to V. *Hence, when there is this much noise, it is not possible to distinguish between the true expression and* `BACON.M`*'s wrong prediction by comparing to noiseless* V. If this did happen in the real-world, dimensional analysis could be used to show `BACON.M`'s prediction is not valid.

Also, the expression found in MCTS with $\frac{M}{a}$ was

$$V_{\texttt{BACON.M}} = \frac{M(0.870T + 549.756)}{P} \tag{6.2}$$

with MSE of 0.026 to $V + n$ and 0.159 to V. The 273 factor is lost - *presenting this level of noise defeats* `BACON.M`. The coefficients are not related to MCTS, suggesting the process isn't the problem and rather `BACON` just can't correctly ascertain the best coefficients in 5% noise. This is an upper-bound on what `BACON`'s simplistic mechanisms can handle based on noise not averaging out. `PySR` on 5% noise finds

$$V_{\texttt{PySR}} = \frac{M(T + 269.11)}{P} \tag{6.3}$$

with 1.4×10^{-4} MSE to $V + n$ and 3.2×10^{-5} to V. `PySR` does not have the drop in performance `BACON.M` does when noise increases.

6.2.2 Black's Law

The Ideal Gas Law failed at 5% noise, however the technique may still be valid at less. Reducing the noise for Black's Law to 2% which failed in subsection 5.3, the initial node of $T_f - aT_2$ is selected correctly, but the expression quickly derails into:

$$T_{f_{\texttt{BACON.M}}} = \frac{(28.544M_2M_1 + 0.940M_2)\left(\frac{M_1}{M_2^3} + \frac{M_1M_2}{M_2^4}\right)^{\frac{1}{3}} - 0.258M_1 - 0.242M_2 + 0.001T_1T_2}{M_1 + 0.940M_2} \tag{6.4}$$

The MSE loss is 7.704 to $T_f + n$ but 1366 to T_f. The best attempts using `BACON.7` is[1]

$$T_{f_{\texttt{BACON.7}}} = \frac{0.623(0.171M_1^2M_2T_2 - 0.603M_1^2T_1 - 0.553M_1M_2T_1 - M_1M_2T_2 - 0.615M_2^2T_2)}{0.111M_1^3 + 0.102M_1^2M_2 - 0.653M_1^2 - M_1M_2 - 0.368M_2^2} \tag{6.5}$$

with MSE loss of 13.3 to $T_f + n$ and 12.3 to T_f. *This displays that MCTS overfits on the noisy data.* If instead it was just being defeated by noise I'd expect equation 6.5 to have higher MSE to T_f than 12.3 which is inline with the exponential increase from Figure 5.3.

The reason is likely the reward function. Scoring smaller MSEs too high rewards fitting the equation to $T_f + n$ - overfitting. From equation 6.5, which is the best function I've found for fitting T_f, it has an MSE twice equation 6.4 to $T_f + n$. This is large enough to make me think that other equations can be found that have the same MSE to $T_f + n$ but higher to T_f. The reward function then wouldn't be able to differentiate based on MSE score, leading to a question of what factors it should consider. Dimensional analysis is a possible direction, but in general *this suggests that MCTS is not the correct adaptation for* `BACON` *as the right reward function is very hard to find, if it does exist at all.*

6.3 Discussion

As noise increases, the uncertainty in the true form of the equation becomes a factor. The equations best formed by `BACON.M` can cut out significant noise, and resemble the true form, whilst having extra factors. This is seen in equation 6.1. This displays a limitation in the noise `BACON` can handle. `PySR`'s attempts at the same problem show that finding the correct form is possible but `BACON.M` is not powerful enough to get there. Exterior mechanisms to `BACON.M` would have to be developed such as in a preprocessing stage to remove excess noise.

Whilst the actual implementation between MCTS and `BACON` was successful, MCTS – and likely any game tree search – is not the solution to improving `BACON`'s power. It effectively becomes a `PySR`-like attempt at overfitting on the data, but without the accuracy due to the difficulties in finding a reward function. It is a complex task as what is needed varies between noise and dataset size (seen by different reward functions between the Ideal Gas Law and Black's Law). I didn't spend too long on it for this reason, and there is potential to improve the approach. I think there are other better future directions to take with `BACON` referenced in subsection 7.5 which include modern machine learning techniques.

Lastly, MCTS approaches lacks speed. Whilst the Ideal Gas Law took $\sim$ 13 seconds, comparable with `PySR`, Black's Law took 700 seconds due to the additional layer and the branching factor associated with the possible equations in a noisy environment. It will not scale well to larger datasets. `PySR` took $\sim$ 12 seconds to exactly find Black's Law.

[1] Found under args/black/black20.json

Chapter 7

Summary and conclusions

7.1 Technical Contributions

The project achieved its initial technical goals through its successful reconstruction of `BACON`. The accomplishments were:

- A rebuild of Langley's `BACON`. Upgrades were then made to form my own model named `BACON.7` which was more noise-resilient than Langley's program which could not handle noise at all (bar 0.5% on the Ideal Gas Law).
- `BACON.7` was able to deal with the aimed for 4% noise on the Ideal Gas Law and Ohm's Law. On Black's Law it could only handle 1.5%. This failure demonstrated potential improvements to `BACON.7` by making it able to use different hyperparameters at the last stage in the search tree.
- These tests also displayed `BACON.7` outperforming state-of-the-art DNN `PySR` on smaller datasets due to its superior ability to strip away noise through its repeated averaging. On larger datasets, `PySR` was superior and demonstrated the strength of modern machine learning techniques by its ability to extract the exact version of Black's Law in up to 4% noise, whilst `BACON.7` could not.
- Successfully navigated the possible equations that `BACON.7` can generate using a MCTS. The aim was to increase the noise-handling capabilities of `BACON.7` which failed. Instead the approach showed that there is ambiguity of what the true equation is when the data is sufficiently noisy. It also displayed that game tree searches are not the right improvement due to their restrictive reward functions.

7.2 Contributions to Explainable and Classical AI

This project also successfully contributed to Explainable AI by:

- Building `BACON.7` with an eye on understandability. It contains multiple levels of

verbose settings, a straightforward output, and is also reproducable through its argument files.

- Generating a model that is explainable, yet retains the accuracy to compete against the state-of-the-art.
- Demonstrating how explainability in the model can aid development, whilst a lack of explainability hinders.

It aided the field of Classical AI. It did this through:

- Displaying how transformed 1980s' Classical AI techniques can compete, and even outperform, the current state-of-the-art.
- Releasing open-source code[1] to encourage development in this field.
- Forming suggestions on how to combine modern method with classical approaches to improve performance in both subsection 4.3.4 and in subsection 7.5.

7.3 Limitations of Classical AI and `BACON`

The problem when adapting 1980s' Classical AI techniques is that the research papers and books tend to be very dense. Without the ability to share their codebases through websites such as GitHub, the authors had to explain the code whilst also trying to discuss the research they were presenting. This makes reading the papers confusing and tedious, both due to the lack of time dedicated to explaining the important concepts, and also the code being described in an antiquated language that's not-understandable without prior knowledge which I don't have as it's outdated..

This was a problem I discovered when analysing Langley's textbook [Langley et al., 1987b] which made it hard to make consistent progress. It was also seen when trying to understand the other research in the field and explains why there seems to be a lack of enthusiasm to perform modern-day adaptions of 1980s' designs.

In general, `BACON` and Classical AI as a whole have a simplicity that means they will never have the accuracy, versatility and speed of modern approaches such as `PySR` when applied to large and noisy datasets. However, this project wasn't solely motivated by this seemingly fruitless endeavour of outperforming DNNs. Instead, the broad goal was to show that Classical AI can still be effective today. The next step is melding these two concepts to build the best model possible. Both have strengths which, when efficiently combined, could lead to refined systems which are able to analyse both large and small datasets effectively.

[1] Code is available at `https://github.com/JonahMiller/BACON`.

7.4 Limitations of Approach

As my first time properly exploring and utilising Python packages Pandas and SymPy - there's bound to be multiple instances where I have not coded the most efficient mechanisms. In addition, the iterative nature of `BACON.7` can be sped up by implementing Python's multiprocessing. Unfortunately, this didn't get completed due to time constraints. Furthermore, rewriting this project in languages that perform operations faster such as Julia, Haskell or Java would speed up `BACON.7`. However, as Python is the most preferred language for data science and one of the most understandable out there [Scarlett, 2023] , it coincides with this project's focus on explainability.

7.5 Future Directions

In terms of Classical AI based upgrades to `BACON.7`, I would add an ability to use dimensional analysis on the final found equation. If this equation was not dimensionally consistent, `BACON.7` should be made to automatically update the hyperparameters at the final layer of the tree to form a new different equation which is then validated by dimensional analysis. If this also failed, the process would repeat until a dimensionally consistent expression was found.

There is an alternative way to integrate dimensional analysis through modern machine learnig techniques. Based on what variables – and their dimensions – are left to be applied, a neural network could prune expressions that violate dimensional analysis, but in the midst of the `BACON.7` process as a layer-method, rather than on conclusion. A potential partner here is `AI Feynman` [Udrescu and Tegmark, 2020]. It is another modern mechanism aimed at CSD via symbolic regression but takes a more physical approach - detailing similar steps to `BACON` such as considering smoothness and symmetry. The initial step of their algorithm involves dimensional analysis indicating it is possible.

On a different note, there are other directions to take `BACON`. The Ramanujan machine project has gained popularity recently through their computational approach to finding continued fraction formulas for common constants [Elimelech et al., 2023]. For instance, they found:

$$\frac{4}{3\pi - 8} = 3 - \cfrac{1}{6 - \cfrac{6}{9 - \cfrac{15}{12 - \frac{28}{15 - \ldots}}}} \tag{7.1}$$

The approach is a novel combination of 4 algorithms, not using large neural networks. As such it the follows Classical AI tenets of `BACON` and feels like it should be coalesced with it.

Bibliography

David Brink. A (probably) exact solution to the Birthday Problem. In *The Ramanujan Journal*, 2012.

Matthew Conroy. Determining quadratic functions. `https://sites.math.washington.edu/~conroy/m120-general/quadraticFunctionAlgebra.pdf`, 2012.

Adrien Couetoux, Martin Müller, and Olivier Teytaud. Monte Carlo Tree Search in Go. 2013.

Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *5th International Conference on Computer and Games*, 2006.

Miles Cranmer. Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl. In *arXiv:2305.01582*, 2023.

Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

Johannes Czech, Patrick Korus, and Kristian Kersting. Monte-Carlo Graph Search for AlphaZero. In *arXiv:2012.11045*, 2020.

Guy Van den Broeck, Kurt Driessens, and Jan Ramon. Monte-Carlo Tree Search in Poker using Expected Reward Distributions. 2009.

Rotem Elimelech, Ofir David, Carlos De la Cruz Mengual, Rotem Kalisch, Wolfgang Berndt, Michael Shalyt, Mark Silberstein, Yaron Hadad, and Ido Kaminer. Algorithm-assisted discovery of an intrinsic order among mathematical constants. In *arXiv:2308.11829*, 2023.

Brian Falkenhainer and Ryszard Michalski. Integrating Quantitative and Qualitative Discovery: The ABACUS System. In *Machine Learning*, 1986.

Seungwoong Ha and Hawoong Jeong. Unraveling hidden interactions in complex systems with deep learning. In *Scientific Reports*, 2021.

IBM. What is explainable AI? `https://www.ibm.com/topics/explainable-ai/`, 2022.

Sirvan Khalighi, Kartik Reddy, Abhishek Midya, Krunal Balvantbhai Pandav, Anant Madabhushi, and Malak Abedalthagafi. Artificial intelligence in neuro-oncology: advances and challenges in brain tumor diagnosis, prognosis, and precision treatment. *npj Precision Oncology*, 2024.

Man-Je Kim, Donghyeon Lee, Jun Suk Kim, and Chang Wook Ahn. Surrogate-assisted monte carlo tree search for real-time video games. *Engineering Applications of Artificial Intelligence*, 2024.

Bruce Koehn and Jan Zytkow. Experimeting and theorizing in theory formation. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, 1986.

Patrick Langley. Bacon: A production system that discovers empirical laws. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977.

Patrick Langley, Simon Herbert, and Gary Bradshaw. Bacon.5: The discovery of conservation laws. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981.

Patrick Langley, Herbert Simon, and Gary Bradshaw. Heuristics for empirical discovery. In *L. Bolc (Ed.), Computational models of learning*, 1987a.

Patrick Langley, Herbert Simon, Gary Bradshaw, and Jan Zytkow. Scientific Discovery: Computational Explorations of the Creative Process. 1987b.

Bernd Nordhausen and Patrick Langley. A robust approach to numeric discovery. In *Proceedings of the Seventh International Conference on Machine Learning*, 1990.

Kazumi Saito and Ryohei Nakano. Law discovery using neural networks. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

Rizel Scarlett. Why Python keeps growing, explained. `https://github.blog/2023-03-02-why-python-keeps-growing-explained/`, 2023.

Istvan Szita, Guillaume Chaslot, and Pieter Spronck. Monte-Carlo Tree Search in Settlers of Catan. 2009.

Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 2020.

Takashi Washio and Hiroshi Motoda. Discovering admissable models of complex systems based on scale types and identity constraints. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

Appendix A

Additional Technical Details

A.1 Implementation

Making the program run on a system had to be straightforward and reproducible, whilst an option for a verbose mode had to be consistently available. The former was accomplished via the ability to write argument files dealing with all variables. An example file, looks like figure A.2:

```
// args/ideal/ideal30.json

{
    "layer_method": "min_mse",
    "layer_args": {"verbose": true},

    "laws_method": "bacon.1",
    "laws_args": {"epsilon": 0.025,
                  "delta": 0.04,
                  "c_val": 0.5,
                  "epsilon_scale": 1.05,
                  "delta_scale": 1.2,
                  "verbose": false},

    "data_space_args": {"Delta": 0.01,
                        "verbose": false}
}
```

Figure A.1: Typical argument file for this project. This can be found in the codebase at args/ideal/ideal30.json, as it was used for the Ideal Gas Law with 3% noise. The simplistic nature of this implementation is key for understandibility and reproducability.

The command to run on a dataset - such as the Ideal Gas Law - with 3% noise is:

```
python3 main.py --dataset ideal --noise 0.03 --args args/ideal/ideal30.json
```

The output of the above - my recommendations for the minimalist approach to understand the order of `BACON`'s implementation - can be seen in Figure A.2.

Comparatively, the output to a run of `PySR` - for the same dataset and noise - is shown in

```
Ranking layer: Expressions found with associated popularity are:
               {P*V: 9}
Ranking layer: Proceeding with P*V
Ranking layer: Expressions found with associated popularity are:
               {P*V: 2, P*V - T*a: 1}
Ranking layer: Iteratively ranking the found expressions:
               P*V has average mse 0.004335870878343652
               P*V - T*a has average mse 0.002732361384888301
Ranking layer: Proceeding with P*V - T*a
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

The constant equations found are:
1.03911145217940 = M/a
0.00365531074561991 = M/(P*V - T*a)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Final form is V = 273.574552094724*M*(0.00351772732169719*T + 1.0)/P
with loss 0.00015958807925621198.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Program took 1.65s!
```

Figure A.2: A potential output when running an instance of my BACON. This gives the best trade-off of demonstrating the process of BACON whilst remaining succinct. More details can be outputted (via changing the args file) to detail the logic behind each instance of the Space of Laws, and the overall Space of Data - rather than just each layer.

Figure A.3. It demonstrates a list of possible equations PySR builds on to present its final output. There's no explanation of how it accomplishes this. In addition, the outputs are not repeatable further reducing understandability.

```
Hall of Fame:
-----------------------------------------------------------------------------
Complexity  Loss       Score     Equation
1           5.064e-01  1.594e+01  y = -0.31842
3           2.523e-02  1.500e+00  y = 581.48/P
5           2.145e-04  2.384e+00  y = M*(290.29/P)
7           4.902e-05  7.381e-01  y = ((270.29 + T)/P)*M
9           4.702e-05  2.087e-02  y = ((307.99 - (289.71/T))/P)*M
11          4.645e-05  6.064e-03  y = ((270.29 + ((T/0.88717) - M))/P)*M
13          4.600e-05  4.866e-03  y = (((269 + T) + ((T*0.18127)/M))/P)*M
15          4.530e-05  7.727e-03  y = M*(((272.25 - ((M*(3.569/T))*3.5295)) + T)/P)
-----------------------------------------------------------------------------
```

Figure A.3: The output of PySR. It simply lists the newly found equation at each level of complexity and its associated score and loss with no justification.

A.2 BACON.5 calculations

Extrapolating for an $n+1$ variable system, with n independent variables, 1 dependent variable, and calling the total choices $c(n)$. From the first independent variable down, there are 3 possible values of this variable that the initial set can take. In the branches

of this variable coinciding with the 2 values not chosen there is 3^{n-2} possible options as any index can be picked. The equation then becomes $c(n) = 3 \times 3^{n-2} \times 3^{n-2} \times c(n-1)$. Noting $c(2) = 1$ and iterating this process, $c(n) = 3^{n-2+2(n-2+\ldots+1)} = 3^{n(n-2)}$.