Gottfried Wilhelm Leibniz Universität Hannover
Faculty of Electrical Engineering and Computer Science

Master Thesis
International Mechatronics (M.Sc.)

Knowledge Production and Control of a Black Box Using Machine Learning

Student:	Christopher W. Blake
First Supervisor:	Prof. Eirini Ntoutsi
Second Supervisor:	Prof. Viacheslav Shkodyrev
Date:	October 8, 2018

Declaration of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without due acknowledgement in this thesis. All references and verbatim extracts have been quoted, and all sources of information have been specifically acknowledged.

_____ Hanover, October 8, 2018

Christopher W. Blake

Abstract

An adaptive stream-based process is described for learning a set of vocabulary to control a black box. The method serves to accurately discretize the value space, enable tracking, and identify simple patterns. These values and patterns become the system knowledge (or vocabulary) and are used for training a reinforcement learning based decision tree. An interpretation layer enables developing higher-level knowledge with time, creating an easy-to-read policy of the black box functionality, providing control information. Tests are performed to demonstrate the effectiveness and limitations with open-loop systems. Finally, a proposal is made for further knowledge identification, closed-loop systems, and learning with less prior information.

Keywords: black box, knowledge extraction, control, machine learning, decision tree

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this work, a dynamic an adaptive method for controlling a black box is described. The work begins with an overview of the history, specifies the objective, and mentions possible applications and limitations. It then introduces related work and existing processes (Chap. 2). Next the problem definition (Chap. 3) defines the important models and nomenclature, which enable the learning process (Chap. 4), and clarifies the control method, inner processes, and their roles. Afterwards, a set of experiments are presented which test the functionality of each inner process as well as the overall control method (Sec. 6 & 7). Finally, this work concludes with a summary of the results, important limitations, and recommendations for further work (Chap. 9).

1.1 Problem Statement

Systems and products are developed on a daily basis, all of which require a control process. The development of such a control process often requires extensive analysis and requires domain-specific knowledge. An automatic or semi-automatic process for developing control systems would enable faster-to-market and more capable products.

1.2 Objective

This work plans to describe a process that emulates a system of learning to produce knowledge about a black box device. Through the processes described in chapter 3 it is possible to identify the unique pieces of knowledge, or vocabulary, and primitive functions of a black box device or previously uncertain system. As such it has the following objectives.

- Identification of the unique data experienced by a black box.
- Identification of the repeating structures within the unique data.
- Identification and modeling of the primitive functions of a black box, providing the primitive control mechanisms.

1.3 History and Overview

In all systems of life, there is a common theme: the need to experiment, collect, associate, and control. Through this ongoing and repeating process, new capabilities are discovered at the frontier of current knowledge, leading to higher level knowledge. A formal definition of “knowledge” would be “acquaintance with facts, truths, or principles, as from study or investigation.” In the scope of this work, “knowledge” will be referred to as the relationships learned in and between input signals and output responses of a black box.
At any time, the recorded input signals and output responses from a black box can be examined to infer knowledge about the inner mechanism. This is however within a limit; one can only describe new knowledge within the vocabulary of existing knowledge (lower-level knowledge). As such, these new descriptions can be considered as new knowledge at a slightly higher level. Naturally, this higher-level knowledge then acts as more vocabulary to enabled higher level experimentation. This process usually repeats until sufficient knowledge of the black box is known, and a specified threshold of productivity has been reached.

1.4 Applications

Below are three example usage scenarios for such a black box identification process. However, only the first will be explored during the development of this work.

1. **Control System Development** – A controller can automatically be determined by exploring the full range of outputs, and associating them with the inputs that cause them. Examples include simple processes such as the logic operators "And", "Or", and "Xor", which are usually easy to solve or very complex devices such as robotic arms, which need high-level mathematical and engineering knowledge.

2. **Repetitive Task Identification** - A controller can monitor a manually controlled system and learn the repeated actions, converting them into memorized tasks. These tasks could be presented to the user, labeled, and allow simplified operation.

3. **Complex Scenarios** – Virtual or real sensors could be placed in an environment and relationships between the sensors identified, producing vocabulary to describe some goal. Such an example could be the unemployment rate, where virtual sensors are defined to monitor several thousand daily parameters.

1.5 Scope and Limitations

1. **Closed loop system** - In the robotic arm example, the positions of the arms are determinable, but the path between positions are not. The current process does not provide feedback from the outputs into the controller inputs. Hence, such a dynamic system cannot yet be learned.

2. **Response time** - All real systems have a time delay. The current procedure only detects bigram relationships, potentially creating confusion or false relationships with time.

3. **Gaussian noise distribution** - A unique data point per input or output is assumed to have noise with a gaussian distribution. Data points with non-gaussian noise characteristics cannot be identified.

Chapter 2

Related Work

Machine learning approaches have become very popular for solving classification and regression problems. This is especially true in highly nonlinear control problems where traditional, usually analytical, approaches are very difficult or may simply never be possible. The topic of this work is also directly related to the field of iterative learning control (ILC) and indirectly to the idea of language learning.
The field of iterative learning control is very broad and traditional feedback control, optimal control, adaptive control, robust control, and *intelligent* control [3]. The methods attempt to optimize a task by repeating that task several times, usually in batches or trials. A typical problem definition composes four components: inputs, outputs, disturbances, and measurement noise [5]. The common control objective is usually to follow some pre-defined reference path. As such there are many algorithms and approaches in in ILC. However, even though there are supposed intelligent methods, this term can be misleading in the current state of the art. The basic goal of ILC is essentially to optimize accuracy of a single task, accounting for expected disturbances. By repeating the same task several times and comparing it to the reference path, an error is computed and used to train the algorithm. This training is limited to a specified control parameter designated by the designer. Finally, it is important to note that there are different levels of "learning" [3] and learning in ILC is always at the lowest level of the pre-specified parameter.
To generalize and remove this restriction, the area of intelligent algorithms has been significantly developed. In recent years, machine learning methods involving neural networks and genetic algorithms have been employed to try to identify existing dynamic systems and effectively model them as a combination of traditional feedback control approaches, similar to a regression task. However, again these systems usually require prior knowledge about the controlled system, such as degrees of freedom or control parameters.

Essentially, the optimization task has shifted from learning on only one parameter to learning on several parameters. Like earlier systems, many of these also have little or no generalization capabilities beyond the designated task and initial training period. In fact, they also introduce a new problem, that of making decisions with partial information.

The most similar known work uses neural networks to train a controller, again with some (less) prior knowledge. A plant (black box) is emulated using a neural network, similar to plant identification in control theory. Afterwards the controller is trained on this emulated system. An emulated plant is required because the controller needs several passes, which is impractical with the real system. The number of layers in the neural network of the emulated plant are determined by the degrees of freedom of the plant. As the controller attempts to control the emulated plant, it also performs a backpropagation through the emulated plant, providing the error information. As such, through many passes, the weights of the controller's neural network converge, minimizing an error function. Although the controller automatically learns to control the emulated plant, it does so with help of the emulated plant, which contains significant prior knowledge provided by the designer. [4]

Finally, although not directly related to control is language learning, discussed by Kirby. He theorizes that the origin of language is an inevitable outcome of the system dynamics. He claims that humans have a set of shared learning biases, and based on these biases, human makes similar assumptions when attempting to deconstruct and identify the components of another's message. Essentially, language develops by identifying shared coincidences amongst the communications of the people. It begins as simple vocabulary with limited expressiveness and coordination and then later develops to be fully covering. [2]

To overcome the problem of prior knowledge and partial information, this work describes a system where unique pieces of knowledge (vocabulary) are identified in the data streams of the inputs and outputs of the black box. Rather than computing the error between the predicted output with an actual output, associations are created using a reinforcement learning approach. This enables multiple relationships between input vocabulary and output vocabulary, similar to the discussed language learning. Additionally, by introducing vocabulary, a degree of abstraction is also possible, creating symbolism for combinations of the original input and output streams.

Chapter 3

Problem Definition

3.1 Black Box Model

A black box can be simply modeled as two data spaces and a set of functions, including an influence of time. The first data space, representing the options for influencing the black box, is defined as all the possible inputs $\mathcal{X} \in \mathbb{I}^d$, where d is the number of input parameters. Similarly, the second data space, representing the possible responses from the black box, is defined as all the possible outputs $\mathcal{Y} \in \mathbb{O}^k$, where k is the number of output sources.
Within the black box, there is a set of unknown functions $\mathcal{F} \in \mathbb{Z}^l$, where l is the unknown quantity of functions. Each function $f \in F$ uses the current input $\vec{x}$ and any internal memory of the black box to cause direct changes to one or more parameters in the output $\vec{y}$. It is important to note that each function effectively processes in parallel. Hence, the changes to $\vec{y}$ occur with different timings. Finally, an extension of this model is also possible, where the scope of the black box can change, meaning the number of input parameters d or output sources k may increase or decrease.

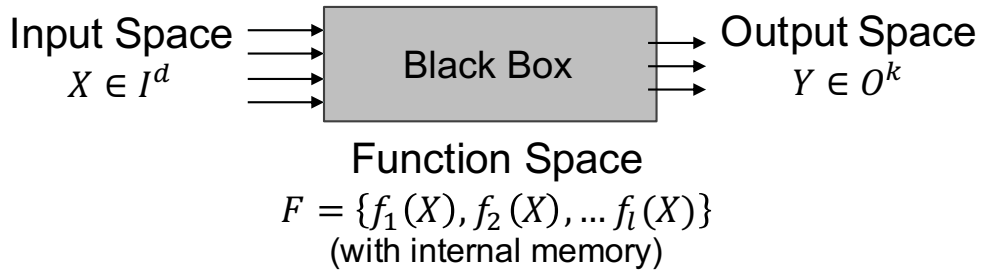


Figure 3.1: Black Box Model

3.2 Discretization Model

As mentioned in section 3.1, a black box has connections for several input and output streams, X and Y respectively. Each of these streams has an associated continuous value space $v \in \mathbb{V}$. Because the value space $\mathbb{V}$ is continuous, it is represented as a set of ranges $r \in \mathbb{R}$. Each range r is defined by a lower limit Low and upper limit $High$ and contains statistics to continuously track the the count N (3.1), sum Sum (3.2), square sum $SqSum$ (3.3), average μ (3.4) and standard deviation σ (3.5) of the previously experienced values through time t.

$$N = \sum 1_t \tag{3.1}$$

$$Sum = \sum v_t \tag{3.2}$$

$$SqSum = \sum {v_t}^2 \tag{3.3}$$

$$\mu = Sum/N \tag{3.4}$$

$$\sigma = SqSum - 2N\mu + N\mu^2 \tag{3.5}$$

The set of ranges may be affected by two actions, a split $\mathcal{S}$ or merge $\mathcal{M}$. The split action separates an existing range r into two new ranges by some intermediate value $value$ (See Alg. 3.1). The merge action combines two neighboring ranges, $rLow$ and $rHigh$, that share a common boundary value, producing one new larger range (See Alg. 3.2).

Algorithm 3.1 Split range

```
1: //Global variable
2: R ← rangesList
3: procedure SplitRange(r, value)
4:     //Create new ranges
5:     rLow ← range(r.Low, value)
6:     rHigh ← range(value, r.High)
7:     //Remove old range and add new ranges
8:     R.Remove(r)
9:     R.Add(rLow)
10:    R.Add(rHigh)
11: end procedure
```

Algorithm 3.2 Merge two ranges

```
1: //Global variable
2: R ← rangesList
3: procedure MERGETWORANGES(rLow, rHigh)
4:     //Check shared limit
5:     if (rLow.High ≠ rHigh.Low) then
6:         throw error: Ranges must share a limit.
7:     end if
8:     //Create new range
9:     rMerged ← range(rLow.Low, rHigh.High)
10:    //Remove old ranges and add new range
11:    R.Remove(rLow)
12:    R.Remove(rHigh)
13:    R.Add(rMerged)
14: end procedure
```

3.3 Knowledge Representation

An instances of knowledge $c \in \mathbb{C}$ is defined as a raw value observed by the black box or a symbol representing a combination of existing knowledge instances. Each knowledge instance should effectively represent one unique piece of information. Two representations are used for these combinations, for simultaneity and sequentiality.

Simultaneity occurs when two or more knowledge instances occur at the same time. The sub-elements of the new knowledge instance are separated by a comma, like $c_i = (c_1, c_3, c_4)$. Sequentiality occurs when two knowledge instances c^t and c^{t+1} are separated by only one time interval. The sub-elements of the new instance are separated by a semicolon, like $c_i = (c_1; c_2)$. Several examples are shown in Table 3.1.

By definition, an instance composed of other instances is referred to as higher level knowledge. For example, in Table 3.1, c_{15} is higher than c_{13}, which is higher than c_1.

All distinguishable values (Sec. 4.1.1) within a single stream of X or Y have an equivalent knowledge instance c. Hence, the entire value spaces of $\mathbb{X}$ and $\mathbb{Y}$ can be translated into $\mathbb{C}$. Naturally, an individual mapping can be written using different levels of knowledge, producing different length descriptions. The details of this mapping process are described in section 4.1.

Table 3.1: Example instances of knowledge with descriptions

Instance	Knowledge Content	Description
c_1	$(x_1 = 0.0)$	Input x_1 has a value 0.0.
c_2	$(x_1 = 1.0)$	Input x_1 has a value 1.0.
c_3	$(x_2 = 0.0)$	Input x_2 has a value 0.0.
c_4	$(x_2 = 1.0)$	Input x_2 has a value 1.0.
c_5	$(y_1 = 0.0)$	Output y_1 has a value 0.0
c_6	$(y_1 = 1.0)$	Output y_1 has a value 1.0
c_7	(c_1, c_3)	$x_1 = 0$ and x_2=0 simultaneously.
c_8	(c_1, c_4)	$x_1 = 0$ and x_2=1 simultaneously.
c_9	(c_2, c_3)	$x_1 = 1$ and x_2=0 simultaneously.
c_{10}	(c_2, c_4)	$x_1 = 1$ and x_2=1 simultaneously.
c_{11}	(c_2, c_4, c_5)	x_1=1, x_2=1, y_1=0 simultaneously.
c_{12}	(c_1, c_4, c_5)	x_1=1, x_2=1, y_1=1 simultaneously.
c_{13}	$(c_1; c_2)$	x_1 set to 1 then 0.
c_{14}	$(c_2; c_1)$	x_1 set to 0 then 1.
c_{15}	$(c_{14}; 3c_2; c_{13})$	x_1 switched from 0 to 1, held for $3t$ then switched to 0.

3.4 Prediction Model

A previously developed model [1] for reinforcement learning based decision trees is modified to discover the best set of knowledge instances from a potentially evolving stream of data. These queries are then used as the basis of a decision tree, which is used for mapping of black box inputs to black box responses. For consistency with common learning model nomenclature, inputs of the black box model are described as features, responses of the black box as as labels, and a vector of input data to the black box as an instance.

Like the black box model, the input feature space $\mathcal{X} \in \mathbb{I}^d$ is defined to be the input space with dimensionality d, which may change at any new instance. (i.e. new black box inputs may appear or disappear.)

The output label space $\mathcal{Y} \in \mathbb{O}^k$, also like the black box model, may also experience new response values (labels) at any new set of features. Clearly, there is no prior knowledge of the feature or label space before data arrives. However, because $\mathbb{X}$ and $\mathbb{Y}$ are potentially continuous, learning occurs using the discretized values (section 4.1.1). These discretized values are represented as knowledge instances $C \in \mathbb{C}$ (Sec. 3.3).
At every time step $t = \{1, 2, 3, ...\}$, an input $\vec{X}_t \in X$ and a label $\vec{y}_t \in Y$ are provided to the learner. Training then occurs by choosing between two actions, a query $\mathcal{F}_i$ or a report $\mathcal{R}_j$. A query action will introduce an additional feature into the policy, and a report will predict a class label $y \in \mathbb{Y}$. The prediction is compared to the true class label, providing a reward and ending the time step. A policy can then be created or updated by propagating the reward from the classification labels through the queries. This leads to a decision making process that is both highly accurate and uses minimal information for classification. [1]
The DT induction problem is formulated as a Markov Decision Process (MDP), again similar to [1] where transitions occur between states using actions, as defined below.

- **State** (s) A combination of features and their respective values known at that state.
- **State Space** (S) All possible states of the MDP.
- **Query Action** $(\mathcal{F})$ An action possibility at each state, which reads an additional feature's value from the training input $\vec{x}_t$, and causes transition to another state. Naturally, a state cannot query a feature that it already contains.
- **Report Action** $(\mathcal{R})$ An action possibility at each state, which predicts the classification label.
- **Action Space** (A) All possible query and report actions.

Diverging from the formulation of [1], the reward system for queries and reports is modified. The reward for a report $\mathcal{R}_j$ is based on the percentage of an experienced label at that state, modeled as a value between 0 and 1. A query action $\mathcal{F}_i$ is only used to transfer reward between states using a discount factor γ and feature importance w_f as described below.

1. **Discount Factor** $(\gamma \in [0, 1])$ The transfer rate of rewards from a state's report rewards and another state's queries, usually between 0.8 and 0.99.
2. **Feature Importance** $(w_f \in [-1, 1])$ A normalized weighting to encourage or discourage feature inclusion during training. It can also be used for offsetting class imbalance [1] and encouraged deprecation of features.
   (a) +1 indicates a more desirable feature
   (b) 0 indicates neutral importance
   (c) -1 indicates a less desirable feature

Chapter 4

Learning Process

The learning process (Fig. 4.1) is split into three main interlinked processes and two optional processes. These are listed below with a brief description. Greater detail of each process is located in the following sections. Referring again to Fig. 4.1, values are sampled from the black box, discretized, converted into primitive low-level knowledge, and then provided to the signal interpreter. The signal interpreter perceives the inputs and outputs with the current state of the knowledge and provides the interpretations to the knowledge producer and policy trainer. The knowledge producer will attempt to create higher-level knowledge. The policy trainer will attempt to map perceived inputs to the perceived outputs, producing a policy for each output. Finally, these policies become the control information for controlling the black box.

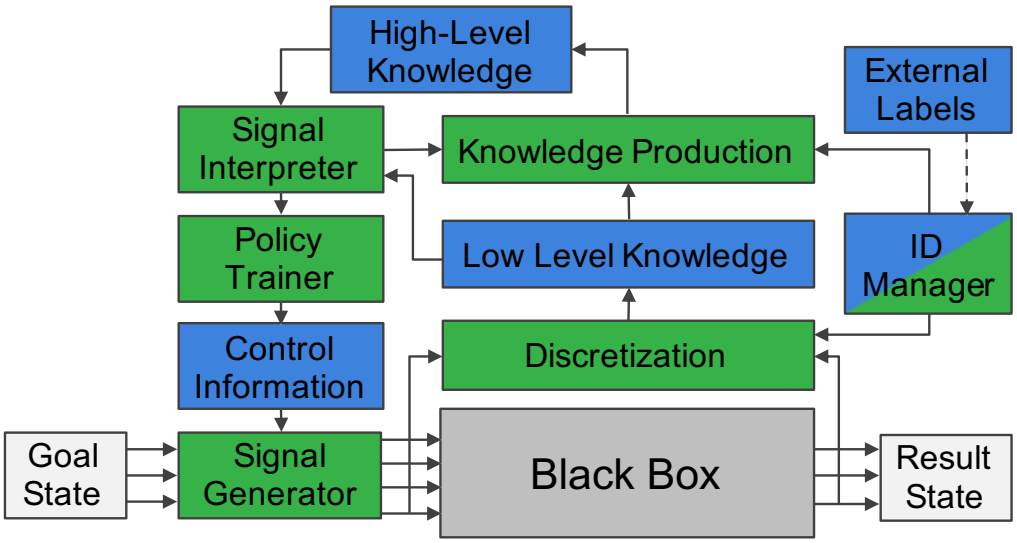


Figure 4.1: Learning Process Flowchart

1. **Discretization Process** – All values sent to the inputs and received from the outputs of the black box are continuously sampled. This data is then discretized, enabling tracking with the ID manager. As new data is experienced the feature-value space expands/contracts and existing knowledge re-associates.
2. **Knowledge Production Process** – The interpreted input and output signals are used to extend the current knowledge to higher-level knowledge. Simple patterns are identified by simultaneity, sequentiality, or other more complex methods and tracked with the ID manager.
3. **Policy Training Process** – Using the current state of the knowledge, the inputs and outputs are mapped to knowledge instances (vocabulary). These mapped values are then used for training a reinforcement learning-based decision tree.
4. **Skill Training Process** (optional) – Using the other processes, external knowledge may be introduced. The other processes allow recognizing and remembering repeated knowledge instances. As such, a user may manually repeat an activity multiple times and this activity will become its own knowledge instance.
5. **Labeling Process** (optional) – The other processes uses internal identifiers for tracking. As such they are not human-interpretable. The labeling process allows external naming to be introduced without affecting learning. As such, a user may manually assign a human-interpretable label to any generated vocabulary.

4.1 Discretization

The first step of producing knowledge is establishing distinguishability. That is, there must be the ability to distinguish a difference between the raw values within the input space and the raw values within the output space. At any time, new values may be submitted as input parameters and new responses may be produced at the outputs. Additionally, in an even more dynamic process, inputs and outputs may be added or removed. (i.e. the scope of the black box changes.)
At the beginning of learning, the values of a space may appear very binary, such as black/white, yes/no, or on/off. However, as more data is experienced, the space may transform into categorical and even continuous. For this work, true continuous space will not be considered and instead be modeled as categorical with potentially many options.

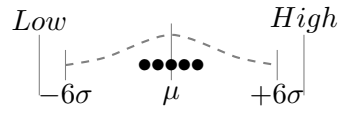
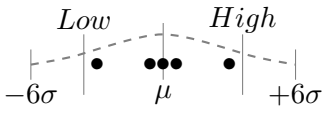
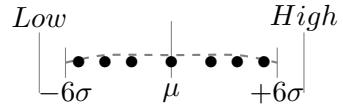
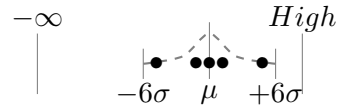
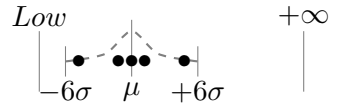
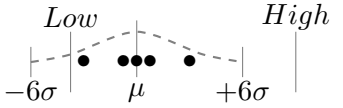
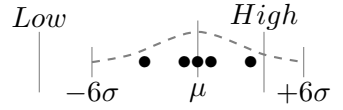
Considering these factors, the discretization process will assign unique identifiers to all raw data it considers to be at a distinguishable level. There are many successful discretization options from statistics and clustering techniques, such as histograms, bucketing, and binary splitting. Many of these processes also require prior information or multiple passes through the source data; some methods are also supportive of streams. With streams, the discretization ranges may change. As such, an important topic is enabling preservation of existing knowledge.
A typical discretization process focuses on having as few ranges as possible, such that the labels are accurately assigned. It may also be supervised or unsupervised. Unlike in this scenario, neither the value data nor the label data is known, so an unsupervised top-down approach is used. Instead, specific values are assumed to re-occur, with a certain amount of gaussian noise, indicating the similarity of the values. Considering these properties, the discretization method must contain the following capabilities.

- Learning new and significantly different values.
- Expanding dynamically for adding new ranges.
- Compressing dynamically for removing old ranges.
- Allow increased sensitivity in local regions.

4.1.1 Distinguishability

Each input x or output y data stream is tracked independently. The stream of values for a single data source are converted into a dynamic set of ranges $r \in \mathbb{R}$, dependent on the previously experienced values in that stream.
As defined in the section 3.2, each of these ranges (or bins) includes statistics for tracking the average μ and standard deviation σ. Using these statistics, the lower and upper limits of the range are compared with the statistical 6-standard-deviations range of the collected data. From this comparison, seven situations may occur, leading to a specified action as shown in table 4.1.

Table 4.1: Action scenarios during discretization

Visual	Description	Action
	Both 6σ points are within the range limits, and the values are approx. 68.1% in $\mu \pm 1\sigma$.	None
	Both 6σ points extend outside the range limits.	Split at average.
	Distribution of data is too flat. (outside of 1σ)	Split at average μ.
	The range contains data and the the lower limit is $-\infty$.	Split at -6σ.
	The range contains data and the the upper limit is $+\infty$.	Split at $+6\sigma$.
	The -6σ overlaps lower range.	Merge with lower range.
	The $+6\sigma$ overlaps higher range.	Merge with higher range.

The discretization process (Alg. 4.1) receives a value from the stream (line 5), finds the appropriate range, updates the range statistics (line 7), and then checks the distribution (line 20). If the range does not contain an enclosed gaussian distribution, splitting or merging occurs, which discovers repeating values within the stream.

During splitting (Alg. 3.1), a range is broken into two new ranges by some midpoint value. Naturally, the statistics must be purged, as the internal data locations are not known. During merging (Alg. 3.2), two ranges are combined and the old range removed. Naturally, the statistical data from these ranges can be combined.

Algorithm 4.1 Discretize stream

```
1: procedure DiscretizeStream(stream)
2:     R ← rangeList                                   ▷ globally available
3:     R.Insert(range(−∞, +∞))                         ▷ range with Low = −∞ and High = +∞
4:     while (true) do                                 ▷ forever
5:         v ← stream.GetValue()
6:         r ← FindRange(v, "value")                   ▷ See Alg. 4.2
7:         //Update range statistics.
8:         r.Count = r.Count + 1
9:         r.Sum = r.Sum + v
10:        r.SqSum = r.SqSum + v^2
11:        r.Avg = r.Sum/r.Count
12:        r.StdDev = r.SqSum − 2 * N * r.Avg + r.Count * r.Avg^2
13:        r.Neg6Sigma ← r.Avg − 6 * r.StdDev
14:        r.Pos6Sigma ← r.Avg + 6 * r.StdDev
15:        //Check percentage values within 1 Std. Dev.
16:        if (r.Neg1Sigma < v) and (v < r.Pos1Sigma) then
17:            r.Count1Sigma = r.Count1Sigma + 1
18:            r.Percent1Sigma ← r.Count1Sigma/r.Count
19:        end if
20:        //Check for end case scenario
21:        if ((r.Low < r.Neg6Sigma) and (r.Pos6Sigma < r.High) then
22:            if (r.Percent1Sigma < 0.60)) then
23:                restart loop
24:            end if
25:        end if
26:        //6 sigma ranges are outside the high and low.
27:        if (r.Neg6Sigma < r.Low) and (r.High < r.Pos6Sigma) then
28:            SplitRange(r, r.Avg)                    ▷ See Alg. 3.1
29:        end if
30:        //High or low is infinity.
31:        if (r.Low = −∞) then
32:            SplitRange(r, r.Neg6Sigma)              ▷ See Alg. 3.1
33:        else if (bin.High = +∞) then
34:            SplitRange(r, r.Pos6Sigma)              ▷ See Alg. 3.1
35:        end if
36:        //Data is evenly distributed rather than gaussian.
37:        if (r.Percent1Sigma < 0.60) then
38:            SplitRange(r, r.Avg)                    ▷ See Alg. 3.1
39:        end if
40:        //One of the 6 sigma ranges overlaps another range.
41:        if (r.Neg6Sigma < r.Low) then
42:            rLow ← FindRange(r.Low, "high")         ▷ See Alg. 4.2
43:            MergeRanges(rLow, r)                    ▷ See Alg. 3.2
44:        else if (r.High < r.Pos6Sigma) then
45:            rHigh ← FindRange(r.High, "low")        ▷ See Alg. 4.2
46:            MergeRanges(r, rHigh)                   ▷ See Alg. 3.2
47:        end if
48:    end while
49: end procedure
```

Algorithm 4.2 Find range

```
1: //Global ranges list
2: R ← rangeList
3: procedure FINDRANGE(value, property)
4:     //Search by value inside range
5:     if (property = "value") then
6:         for all r ∈ R do
7:             if (r.Low <= value) and (value < r.High) then
8:                 return r
9:             end if
10:        end for
11:        //Search by range's low property
12:    else if (property = "low") then
13:        for all r ∈ R do
14:            if (r.Low = value) then
15:                return r
16:            end if
17:        end for
18:        //Search by range's high property
19:    else if (property = "high") then
20:        for all r ∈ R do
21:            if (r.High = value) then
22:                return r
23:            end if
24:        end for
25:    end if
26: end procedure
```

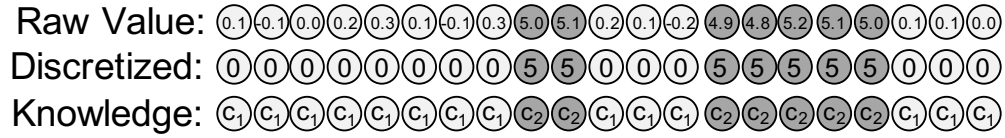


Figure 4.2: Discretization and Low-Level Information

An example of a raw stream with the discretized and knowledge layers is shown in Fig. 4.2. The raw layer shows the original stream data with noise representing a binary signal, on 5 and off 0. This raw data is discretized into the first layer of information and assigned unique identifiers like c_n according to section 3.3.

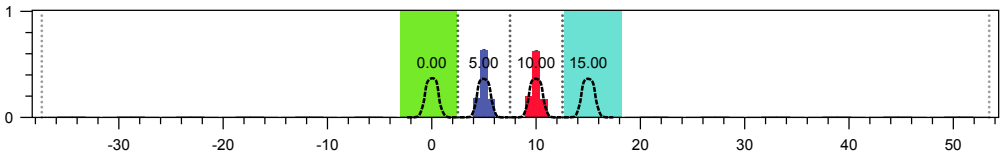


Figure 4.3: Discretized Space, 6 ranges

An example of a generated discretizer with six ranges is shown in Fig. 4.3. Four ranges represent the values 0, 5, 10, and 15. Two ranges represent $[-\infty, -37]$ and $[\ 53, +\infty]$. The range limits (low and high) of each range are represented by the vertical dotted lines. The inner data distribution within the range is represented by the bold dotted line and color bars. The average of each range is displayed as the label.

4.1.2 Knowledge Preservation

Each range is tracked with a unique identifier. As such it may be associated with other knowledge throughout higher-level systems, any systems built using this process. When a range is split, that range's identifier is purged and all knowledge associated with it could become lost. To prevent this loss of knowledge, an event is possible. This would allow higher level processes to properly handle the split/merge event. For example, existing knowledge could be duplicated during a split event, allowing learning to diverge from that point, rather than be reset.
An additional expansion of this technique is to retain old range information and related knowledge. This layering would enable fast estimations when the input and output knowledge account for different tolerances. Example: a color being perceived as blue, instead of light blue, royal blue, green-blue, etc.

4.2 Knowledge Production

As described in section 4.1 and shown in Fig. 4.2, the raw data is discretized and mapped to low-level knowledge instances. Using these lowest-level knowledge instances, simple pairs are identified by simultaneity and sequentiality, which are described in the following sections. Additionally, more complex combinations are likely directly retrievable before additional layers are required. These additional combinations and recursive layering are however delayed for future work.

4.2.1 Sequentiality

An example of a stream with additional learned objects is shown in Fig. 4.5. The stream layer, originally from Fig. 4.2, displays a signal turning on c_1 and off c_2. As data is received, the previous knowledge instance is remembered and compared with the current knowledge instance. The pairwise combinations are stored, producing the following new pieces of knowledge.

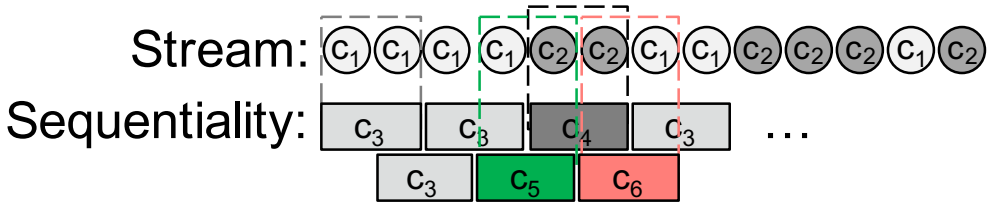


Figure 4.4: Sequentiality in Knowledge Layers

1. $c_3 = (c_1; c_1)$: Staying off.
2. $c_4 = (c_2; c_2)$: Staying on.
3. $c_5 = (c_1; c_2)$: Switch from off to on.
4. $c_6 = (c_2; c_1)$: Switch from on to off.

4.2.2 Simultaneity

An example of two input streams and a simultaneity layer of learned knowledge instances is shown in Fig. 4.5. Both streams contain two unique values and certain combinations of these inputs occur together regularly. Notice that one combination (c_2, c_3) does not occur, hence it is not learned.

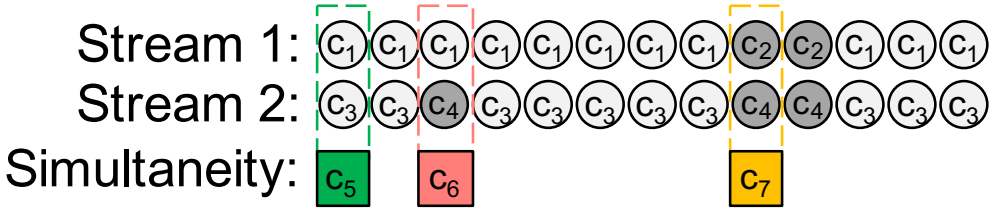


Figure 4.5: Simultaneity in Knowledge Layers

1. c_5 = (c_1, c_3): Off/Off
2. c_6 = (c_1, c_4): Off/On
3. c_7 = (c_2, c_4): On/On
4. c_7 = (c_2, c_3): On/Off (Not present)

4.2.3 Higher Knowledge Layers

An important factor of learning is the creation of higher level knowledge, most of which is only achievable through recursive layer generation. An example of one such recursive technique, while learning a triangle signal, is shown in Fig. 4.6 and further clarified with table 4.2.

Pass 1 of the triangle signal contains no previous knowledge except the low-level values. It is therefore interpreted as only low-level knowledge instances. However, it generates five new knowledge instances that can be used to further simplify the description of the triangle. Upon pass 2 of the triangle, a recursive recognition happens. The first two points are originally interpreted as c_1 and c_2, but after interpretation (Alg. 4.3) they are replaced by c_4. The next two values are again interpreted at a higher level as c_6 and then again as c_8. The same triangle signal now only requires three knowledge instances to describe it. During this pass, two more knowledge instances are generated at an even higher level. This process repeats until pass 4, where the entire triangle is finally learned as a single knowledge instance.

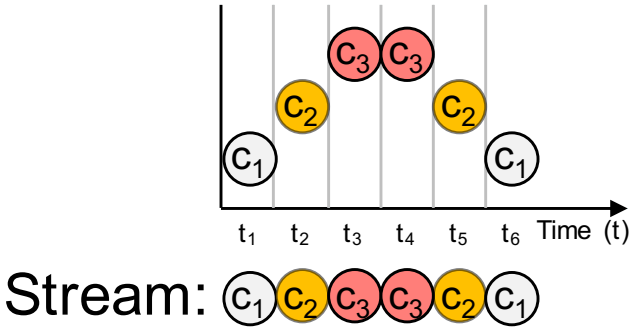


Figure 4.6: Triangle Signal

Table 4.2: Recursive know. generation

Pass	Interpretation	Generated Knowledge
1	$c_1; c_2; c_3; c_3; c_2; c_1$	$c_4 = (c_1; c_2)$ $c_5 = (c_2; c_3)$ $c_6 = (c_3; c_3)$ $c_7 = (c_3; c_2)$ $c_8 = (c_2; c_1)$
2	$c_4; c_6; c_8$	$c_9 = (c_4; c_6)$ $c_{10} = (c_6; c_8)$
3	$c_9; c_{10}$	$c_{11} = (c_9; c_{10})$
4	c_{11}	-

Algorithm 4.3 Recursive interpretation of stream

```
1: procedure RecursiveInterpretStream(stream)
2:     cPrev ← memory
3:     //Sample the stream
4:     v ← stream.GetValue()
5:
6:     //Discretize the raw value
7:     r ← FindRange(v, "value")                          ▷ See Alg. 4.2
8:
9:     //Find current knowledge instance
10:    cCurr ← FindKnowInstance(r.ID)
11:
12:    //Try to interpret the pair with higher level knowledge
13:    if (cPrev ≠ blank) then
14:        cInter ← FindKnowInstance(cPrev, cCurr)
15:        return cInter
16:    else
17:        //Set the memory as the current knowledge instance
18:        memory ← cCurr
19:        return cCurr
20:    end if
21: end procedure
```

4.3 Policy Training

A separate policy is generated for each output of the black box, using the inputs of the black box as the possible features space. However, before submission to the policy learner, they are interpreted through the existing knowledge. By doing so, the policy continuously develops through time using the highest-level knowledge, producing the simplest decision tree.

4.3.1 Action Decision

At a current state s_n, the appropriate query or report action must be determined. A query will cause transition to a different state, or a report will cause the end-training scenario, which updates the expected rewards of the reports at that state. This determination requires comparing the expected reward of all known queries and report options, leading to the action with highest expected reward.

At a given state s_n, there are likely many query options. A unique query at this state $\mathcal{F}_{s,i}$ exists for each combination of feature name f, feature value v, and classification label c as shown in (4.1). However, for a given training instance many of these queries will not be valid, and must be filtered. Similarly, multiple report

actions $R_{s,j}$ may also exist, one for each unique occurrence of a classification label as shown in (4.2).

$$\mathcal{F}_{s,i} = \mathcal{F}(f_i, v_i, c_j) \tag{4.1}$$

$$\mathcal{R}_{s,j} = \mathcal{R}(c_j) \tag{4.2}$$

Because each query and report have an associated expected reward Q at that state s, the expected rewards are determined respectively using (4.3) and (4.4).

$$Q_{\mathcal{F}_{s,i}} = Q(\mathcal{F}_{s,i}) \tag{4.3}$$

$$Q_{\mathcal{R}_{s,j}} = Q(\mathcal{R}_{s,j}) \tag{4.4}$$

4.3.2 Query Reward Calculation

At a current state s_n, upon choosing a query $F_{s,i}$ as the best action, the feature's value is obtained from the current training instance. Using this additional feature-value pair, the next state s_{n+1} is found or create. If a new state is created, all queries are given optimistic rewards of 1.0 to encourage exploration.
The expected reward of the next state $Q_{\mathcal{R}_{s+1,j}}$ is retrieved and used to update the expected reward of the current state's query $F_{s,i}$. The query's new expected reward is a function of the discount factor γ, feature importance w, and report reward $Q(R_{s+1,j})$ as shown in (4.5). If the feature's importance is set to -1, no reward is transferred, whereas if the feature's importance is set to 0, only the discount factor is relevant. For example, new training data can deprecate a feature by setting the importance nearer to -1 and also encourage inclusion of a new feature by setting the importance nearer to +1.

$$Q_{\mathcal{F}_{s,i}} = \gamma(1 + w_f)Q_{\mathcal{R}_{s+1,j}} \tag{4.5}$$

4.3.3 Report Reward Calculation

Each report $\mathcal{R}_j$ at state s is simply a percentage p_{R_j} of the witnessed labels at that state, modeled between 0 and 1. As such the total reward at any state is always equal to 1.0 according to (4.6) but distributed across all known report actions.

$$1.0 = p_{R1} + p_{R_2} + ... + p_{R_j} \tag{4.6}$$

4.3.4 Parallel Query Updates

For any given state, there exists a set of super-set states that lead to the same state, if the appropriate query is used [1]. An example of this is shown in table 4.3.

Table 4.3: Example of Parallel Query Updates

Feature	State 1	State 2	State 3	State 4	State 5
Feature 1	$E = 0.57$	w	w	w	w
Feature 2	t	$E = 0.60$	t	t	t
Feature 3	a	a	$E = 0.53$	a	a
Feature 4	f	f	f	$E = 0.48$	f
Reward	0.45	0.58	0.47	0.23	0.74

In table 4.3, state 1 will query *Feature 1* because the expected reward is 0.57, which is higher than report's reward of 0.45. States 2-4 also fit the same condition. Hence, they will all transition to state 5, retrieving the report action's reward of 0.74, and updating the query's reward appropriately. Hence, these additional queries can also be updated at each state transition, requiring less training instances for convergence.

4.3.5 Parallel Report Updates

During the end condition of a training cycle, the report actions are updated against the known class label from the training instance. It is also valid to state that all report actions of visited states during that training cycle, could also be updated, because they are all super-sets of the final state [1].

Looking again at table 4.3, state 5 is a guaranteed end-scenario because there are no more features. Hence, if this state is reached the report action will be utilized, and the classification label's reports updated. However, to get to this state, any of states 1-4 must have been visited first. Hence, the report's expected rewards at states 1-4 can also be updated, thereby again speeding up convergence.

In theory this seems appropriate, however there exists a convergence problem by introducing reward before the end condition. During training, the super states (states 1-4) of the normally expected end state (state 5) may be visited more often, thereby converging to the percentage probability earlier. Hence, a bias is formed, and training is unable to discover new classification labels deeper in the state space. This is illustrated in Fig. 4.7 and Fig. 4.8, comparing regular reward propagation and propagation with parallel report updates respectively.

Fig. 4.7 and Fig. 4.8 show four states vs time with the expected rewards of the queries and reports. Each state contains a set of features s, various queries F_i and reports R_j. For simplicity only one report's expected reward Q_R is shown per state. Fig. 4.7 shows the normal progression and eventual convergence of the expected rewards (and probabilities). Fig. 4.8 shows the progression and convergence problem when parallel report updates are utilized. Both processes are described for each time step in table 4.4.

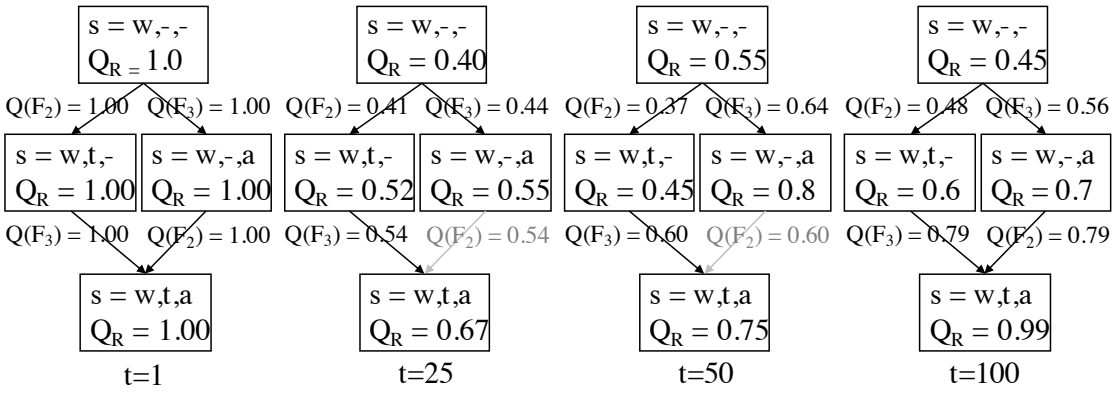


Figure 4.7: States vs Time, Regular Updates

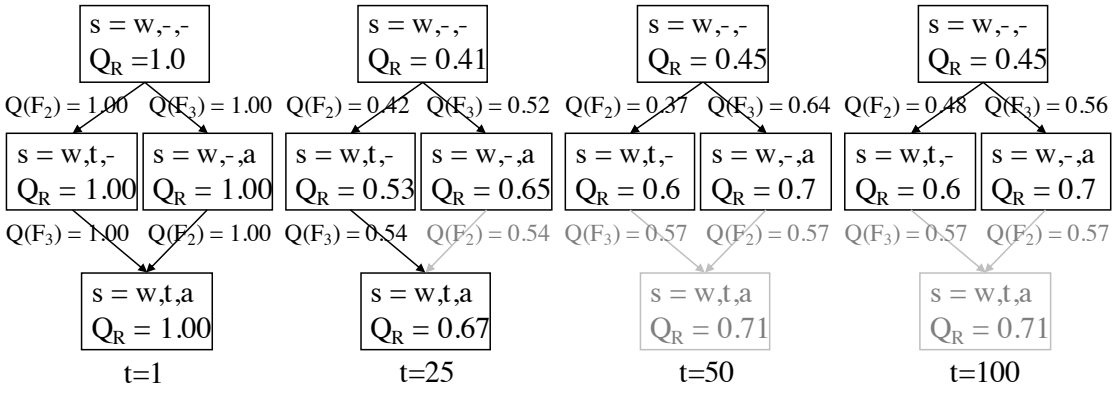


Figure 4.8: States vs Time, Parallel Report Updates

Table 4.4: Reward Propagation, Description vs Time

Time $[t]$	Regular Report Updates	Parallel Report Updates
1	The initial states created but not revisited.	The initial states created but not revisited.
25	A new classification label is encountered, splitting the reward.	The expected rewards are converging faster. A new classification label is encountered, splitting the reward.
50	Reward propagation continues from end state through the super states.	The super states have already converged, preventing exploration.
100	Convergence reached.	No changes occur.

4.3.6 Gini Impurity Index

The Gini impurity index monitors the ongoing validity of the states, preventing the policy from guessing. Each time the reports are updated, the state's Gini impurity index is also updated according to (4.9), which is normalized by the number of reports j by the maximum possible Gini impurity index using (4.8). If this normalized value exceeds 0.99, then the state has a nearly even probability for each classification label, hence the reports and expected rewards are reset, allowing for learning to restart.

$$GiniIndex = 1 - \sum_{j} {p_{R_j}}^2 \tag{4.7}$$

$$MaxGiniIndex = 1 - j(1/j)^2 = 1 - (1/j) \tag{4.8}$$

$$NormGiniIndex = GiniIndex/MaxGiniIndex \tag{4.9}$$

4.3.7 Dimensionality

A large concern of modeling the state as a combination of features of the instance, is the potentially large state space, and the heavy search requirements during state transitions. A few simple calculations can show that 10 features with 2 values has $2^{10} = 1024$ possible states. Likewise 3 options produces $3^{10} = 59049$ possible states and 4 options produces $4^{10} = 1048576$ possible states. It is clear to see that datasets with hundreds of features and several discrete values could be problematic.

However, it is important to note that this is the exact reason why reinforcement learning is a good approach, as opposed to exhaustive techniques like a lookup table. As shown in the following example results of Fig. 4.9, a synthetic dataset did not need to explore all of these combinations. It learns quickly to avoid many state space ranges, and hence only explorers a small subset. For reference, this synthetic dataset with 22 features and 4+ value options per feature (5.2 average) has 1.219E+14 state possibilities. However, training in this example only required 1465 states, which is 1.202E-9 percent of the maximum possible state space and still provided 99.7% accuracy.

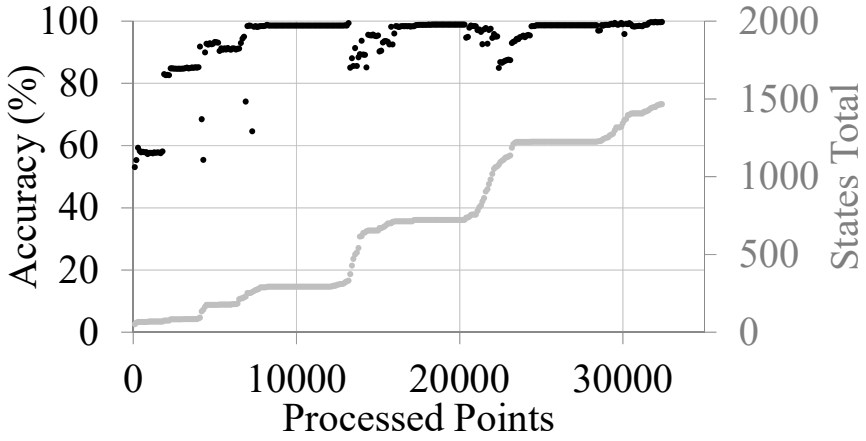


Figure 4.9: Created States vs Processed Data

It should be noted that other issues of high dimensionality still exist such as the potentially impractical search times. In a standard search, the state space may involve multiple filters to look up another state during transitions. Hence an alternative approach for locating states is necessary. To reduce search times of the state space, a hash code is generated from a state's feature-value pairs and then used as the key. In this way, a working state can be added to the state space easily and existing states can be located quickly.

4.4 Skill training

Using the other processes, external knowledge may be introduced. The other processes allow recognizing and remembering repeated knowledge instances. As such, a user may manually repeat an activity multiple times and this activity will become its own knowledge instance.

Chapter 5

Control

The final result of the learning process flowchart (Fig. 4.1) is control information about the black box. This control information, stored in the form of a Markov Decision Process with various state spaces, represents a decision tree that maps the interpreted inputs to the interpreted outputs. Therefore an open-loop system can be simply controlled by a reversal of the decision tree to determine the required input parameters of the desired output state.
There are two methods by which a new instance can be classified. The first, using the complete MDP policy according to Alg. 5.1 and Alg. 5.2. The second, summarizing the MDP policy into a simplified decision tree according to Alg. 5.3 and then using standard decision tree deduction.
Each method has its advantages and disadvantages. Using the complete MDP policy allows for large flexibility because it compares all known features at each state, enabling it to handle partial or incomplete information. Naturally, a disadvantage is that this also requires more processing and may become slow if the feature space grows very large. Conversely, the summarized decision tree is fast but sacrifices handling of partial instances, because it contains only the most important features from the MDP and classifies within that specific feature space only.
An example decision tree for the "exclusive or" logic operation is shown in Fig. 5.1. The generated knowledge instances are shown in tables 5.1 and 5.2 with discussion in section 7.3.

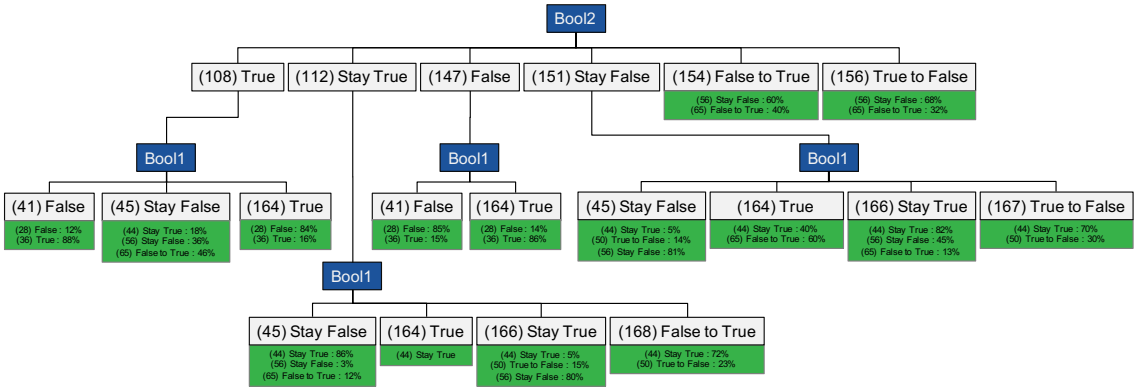


Figure 5.1: Decision Tree, 'Exclusive Or' Operation

Table 5.1: Generated knowledge inputs

Bool1		
ID	**Name**	**Content**
8		[-∞ |∞| 0.00]
41	False	[0.00 (0.00) |0.00| (0.00) 0.00]
45	Stay False	(41; 41)
164	True	[0.00 (5.00) |5.00| (5.00) 5.00]
165		[5.00 |∞| ∞]
166	Stay True	(164; 164)
167	True to False	(164; 41)
168	False to True	(41; 164)

Bool2		
ID	**Name**	**Content**
34		[5.00 |∞| ∞]
108	True	[4.47 (5.00) |5.00| (5.00) 5.00]
112	Stay True	(108; 108)
146		[-∞ |∞| 0.00]
147	False	[0.00 (0.00) |0.00| (0.00) 4.47]
151	Stay False	(147; 147)
154	False to True	(147; 108)
156	True to False	(108; 147)

Table 5.2: Gen. know. outputs

Exclusive Or		
ID	**Name**	**Content**
14		[-∞ |∞| 0.00] (∞)
28	False	[0.00 (0.00) |0.00| (0.00) 0.56]
35		[0.56 |∞| 5.00] (∞)
36	True	[5.00 (5.00) |5.00| (5.00) ∞]
44	Stay True	(36; 36)
50	True to False	(36; 28)
56	Stay False	(28; 28)
65	False to True	(28; 36)

Algorithm 5.1 Classify an instance by MDP policy

```
1: procedure ClassifyByMDP(dataVector)
2:     //Start with root node, with only queries
3:     currState ← policy.rootState
4:     while (true) do
5:         //Get query with highest reward
6:         bestQuery ← GetBestQuery(currState, dataVector)        ▷ See Alg.5.2
7:         if (bestQuery ≠ blank) then
8:             //Transition to the next state
9:             currState ← GetState(currState, bestQuery)
10:        else
11:            //No query, so retrieve the labels
12:            currLabels ← currState.Labels
13:            break loop
14:        end if
15:    end while
16:    //Pick label by percentage probability
17:    theLabel ← PickLabel(currLabels)
       return theLabel
18: end procedure
```

Algorithm 5.2 Get best query, by comparing to label

```
1: procedure GetBestQuery(currState, dataVector)
2:     //Retrieve releated queries
3:     for all (feature ∈ dataVector) do
4:         validQuery ← currState.GetQuery(feature)
5:         valQueries.Insert(validQuery)
6:     end for
7:     //Remove queries with lower reward than labels
8:     for all (query ∈ valQueries) do
9:         qryReward ← query.Reward
10:        lblReward ← Labels.Rewards.Min
11:        if (qryReward < lblReward) then
12:            valQueries.Remove(query)
13:        end if
14:    end for
15:    //If no queries are better than the labels, return blank
16:    if (valQueries.Count = 0) then return blank
17:    end if
18:    //Pick the query with highest reward
19:    bestQuery ← valQueries.Max
       return bestQuery
20: end procedure
```

Algorithm 5.3 Summarize the MDP policy to a decision tree

```
1:  procedure PolicyToTree(policy)
2:      //State with root node, with only queries
3:      rootState ← policy.rootState
4:      rootNode ← (new)TreeNode
5:      //Start recursive subtree creation
6:      PolicyToTree(rootState, rootNode)
7:      //Return the resulting tree return rootNode
8:  end procedure
9:  procedure PolicyToTree(currState, parentNode)
10:     //Get the best queries and labels
11:     bestGroupQueries = getQueriesMaxReward(currState)
12:     lblReward ← currState.Labels.Rewards.Min
13:     //Filter the the queries
14:     for all (query ∈ bestGroupQueries) do
15:         qryReward ← query.Reward
16:         if (qryReward < lblReward) then
17:             bestGroupQueries.Remove(query)
18:         end if
19:     end for
20:     //Create a subnode in the tree
21:     featureNode ← (new)TreeNode
22:     parentNode.SubNodes.Insert(featureNode)
23:     //Add queries to the node
24:     for all (query ∈ bestGroupQueries) do
25:         queryNode ← (new)TreeNode
26:         parentNode.SubNodes.Insert(queryNode)
27:         currState ← getState(currState, query)
28:         PolicyToTree(currState, queryNode)
29:     end for
30:     //Add labels as leaves to the node
31:     for all (label ∈ currState.Labels) do
32:         featureNode.Leaves.Insert(label)
33:     end for
34:     //Return the parent node return parentNode
35: end procedure
```

Chapter 6

Experiments Setup

6.1 Discretization

The discretization process (Sec. 4.1.1) is initially tested with self-generated synthetic data, to control the quality of the input. A set of crisp numeric values are defined and then a gaussian function adds noise with a predetermined maximum. Such sets include binary, categorical, and approximated continuous spaces.
Four experiments are performed to show the development of a set of ranges. Naturally the first consideration is the affect of noise. Next is the potential development or transitioning of an initially binary perspective to categorical and then finally continuous.

1. Binary - The affect of noise on a binary learning task.
2. Categorical - The affect of noise on a categorical learning task.
3. Increasing Resolution (Ranges vs Resolution) - The affect of increasing resolution, to provide enough distinguishability for a continuous system.
4. Varying Resolution - The affect of ranges that have different resolution requirements.

6.2 Knowledge Production

Given that the inputs and outputs of a black box have been successfully discretized, the unique values are now distinguishable. These identified ranges are now converted to knowledge instances (Sec. 3.3).
Two experiments with predetermined signals are submitted to the knowledge producer and knowledge is generated. The list of identified items is presented and compared agains the theoretical results.

1. On/Off - Binary Data
2. Step - Categorical

6.2.1 On/Off - Binary Data

Two binary signals with two value levels are shown in Fig. 6.1. Each signal has a different repetition rate, enabling all combinations of the two signals during testing of logical operations.

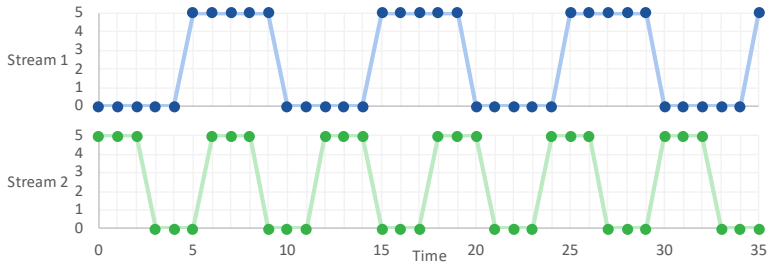


Figure 6.1: Binary Data Streams

6.2.2 Step - Categorical

A categorical signal with five value levels is shown in Fig. 6.2. Notices that each value is held for three time intervals. This is to ensure that a constant value is experienced on a given input/output regardless of the sample rate of the black box.

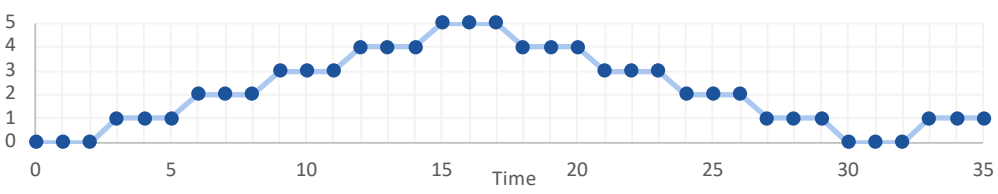


Figure 6.2: Categorical Data Streams

6.3 Policy Training

Given that the the inputs and outputs are converted into a set of vocabulary (knowledge instances), a prediction policy is created for mapping the input vocabulary to the output vocabulary (Sec. 3.4). This mapping of the generated knowledge is then used to reproduce the signal and the mean-square-error is calculated. Three experiments with one categorical, one continuous, and one virtual environment are performed. The generic process of a simulated black box is demonstrated in algorithm 6.1, which specifies a set of inputs/outputs (line 5), update process (line 9), and update interval. The update process runs once per update interval and uses the input values to calculate and set the output values.

1. Logic Operations - Categorical
2. Trigonometric Functions - Continuous
3. Robotic Arm

Algorithm 6.1 Black box simulator

```
1: procedure CREATEBLACKBOX(inputNames, outputNames, updateProcess, updateInterval)
2:     //Create template black box
3:     blackBox
4:     //Configure the inputs and outputs
5:     blackBox.inputs ← listInputs
6:     blackBox.outputs ← listInputs
7:     //Update the black box forever
8:     while true do
9:         //Calculate the outputs using the inputs blackbox.Run(updateProcess)
10:        //Wait some time before running again
11:        Wait(updateInterval)
12:    end while
13: return blackBox
14: end procedure
```

6.3.1 Logic Operators

A virtual black box of the common "and", "or" and "xor" operators (out_1, out_2, out_3 respectively) is used for testing a categorical space. The update process for the input to output mappings are shown via Fig. 6.3 and algorithm 6.2.

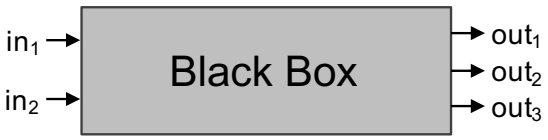


Figure 6.3: Black Box for Logic Operations

Algorithm 6.2 Black box simulation update process, logic operators

procedure LOGICBLACKBOXUPDATEPROCESS
  //and
  if $(in_1 >= 4.5)$ and $(int_2 >= 4.5)$ **then**
    $out_1 \leftarrow 5.0$
  else
    $out_1 \leftarrow 0.0$
  end if
  //or
  if $(in_1 >= 4.5)$ or $(in_2 >= 4.5)$ **then**
    $out_2 \leftarrow 5.0$
  else
    $out_2 \leftarrow 0$
  end if
  //xor
  if $((in_1 >= 4.5)$ and $(in_2 <= 4.5))$ or $((in_1 <= 4.5)$ and $(in_2 >= 4.5))$ **then**
    $out_3 \leftarrow 5.0$
  else
    $out_3 \leftarrow 0.0$
  end if
end procedure

6.3.2 Trigonometric Functions

A virtual black box of the three standard trigonometric functions "sin", "cos" and "tan" (out_1, out_2, out_3 respectively) is used for testing continuous space. The update process for the input to output mappings are shown via Fig. 6.4 and algorithm 6.3.

Synthetic data is created at an interval of 1 for a range of 0 to 180 degrees. The complete set of values was was submitted 5 times to the black black box in random order. The trained policy is then used to recreate the sin, cos, and tan signals and the error is calculated against the true values.

Figure 6.4: Black Box for Trigonometric Functions

Algorithm 6.3 Black box simulation update process, trigonometric functions

1: **procedure** TRIGBLACKBOXUPDATEPROCESS
2: $out_1 \leftarrow sin(in_1)$
3: $out_2 \leftarrow cos(in_1)$
4: $out_3 \leftarrow tan(in_1)$
5: **end procedure**

6.3.3 Robotic Arm

A virtual black box of a robotic arm with three inputs ($in_1 = motor_1[volts]$, $in_2 = motor_2[volts]$, $in_3 = motor_3[volts]$) and five outputs ($out_1 = motor_1[deg]$, $out_2 = motor_2[mm]$, $out_3 = motor_3[deg]$, $out_4 = x[mm]$, $out_5 = y[mm]$) is shown in Fig. 6.5 and Fig. 6.6. The model simulates a pivoting adjustable length arm with pivoting end factor. It contains two angular position sensors, one length sensor, and two positions sensors. The update process for the input to output mappings are shown via algorithm 6.4.

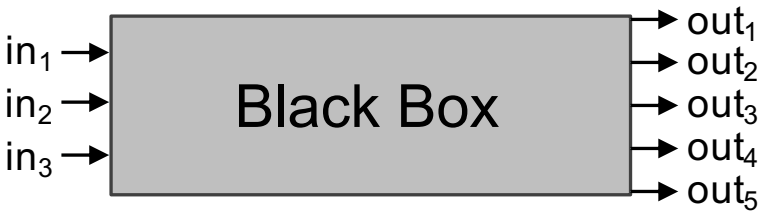


Figure 6.5: Black Box for a Robotic Arm

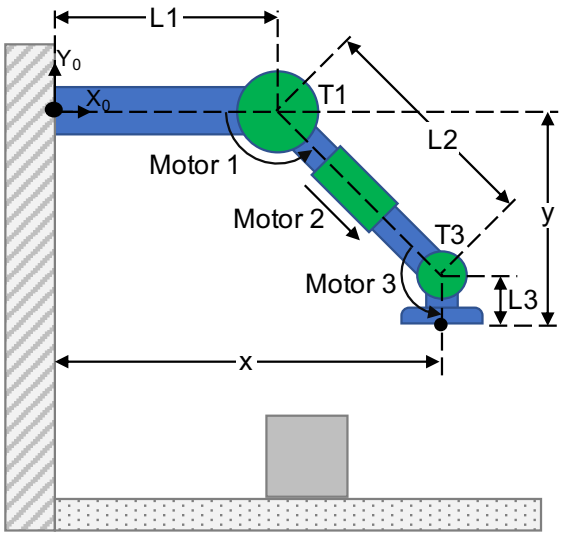


Figure 6.6: Robotic Arm

Algorithm 6.4 Black box simulation update process, robotic arm

1: //Internal memory, robot state
2: $constanttheta_1 \leftarrow 45$ ▷ deg
3: $constanttheta_3 \leftarrow 135$ ▷ deg
4: $constantLENGTH_1 \leftarrow 300$ ▷ mm
5: $length_2 \leftarrow 150$ ▷ mm
6: $constantLENGTH_3 \leftarrow 50$ ▷ mm
7: **procedure** ROBOTICARMBLACKBOXUPDATEPROCESS
8: //Read inputs
9: $voltageMotor_1 \leftarrow in_1$
10: $voltageMotor_2 \leftarrow in_2$
11: $voltageMotor_3 \leftarrow in_3$
12: //Update Motor 1 Angle
13: $theta_1 \leftarrow theta_1 + (0.1 * voltageMotor_1)$
14: **if** $(theta_1 <= 45)$ **then**
15: $theta_1 \leftarrow 45$
16: **else if** $(theta1 >= 180)$ **then**
17: $theta_1 \leftarrow 180$
18: **end if**
19: //Update Motor 2 Length
20: $length_2 \leftarrow length_2 + (0.1 * voltageMotor_2)$
21: **if** $(length_2 <= 100)$ **then**
22: $length_2 \leftarrow 100$
23: **else if** $(length_2 >= 300)$ **then**
24: $length_2 \leftarrow 300$
25: **end if**
26: //Update Motor 3 Angle
27: $theta_3 \leftarrow theta_33 + (0.1 * voltageMotor1)$
28: **if** $(theta_3 <= 90)$ **then**
29: $theta_3 \leftarrow 90$
30: **else if** $(theta_3 >= 270)$ **then**
31: $theta_3 \leftarrow 270$
32: **end if**
33: //Calculate x,y coordinates of end factor
34: $thetaAbs_1 \leftarrow 180 + theta_1$
35: $thetaAbs_3 \leftarrow thetaAbs_1 - 180 + theta_3$
36: $x \leftarrow LENGTH_1 + Cos(thetaAbs_1) * length_2 + Cos(thetaAbs_3) * LENGTH_3$
37: $y \leftarrow 0 + Sin(thetaAbs_1) * length_2 + Sin(thetaAbs_3) * LENGTH_3$
38: //Update Outputs
39: $out_1 \leftarrow theta_1$
40: $out_2 \leftarrow length_2$
41: $out_3 \leftarrow theta_3$
42: $out_4 \leftarrow x$
43: $out_5 \leftarrow y$
44: **end procedure**

Chapter 7

Results

7.1 Discretization

Reading the results

The following sections include figures similar to Fig. 7.2, but with multiple ranges, instead of just one. Each range is represented textually in the form shown by Fig.7.1. Each chart contains the following elements:

1. Average - The text directly above the dotted line.
2. Low - The light vertically dotted line.
3. High - The light vertically dotted line.
4. Distribution - The bold dotted line.
5. Inner Histogram - Data percentage per std. dev. from the average.

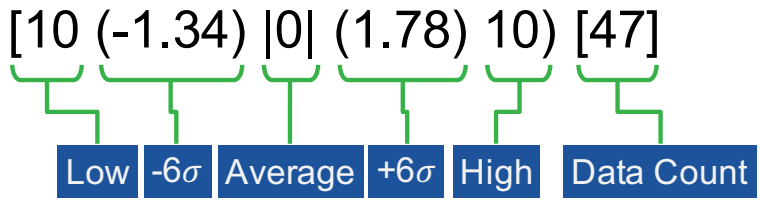


Figure 7.1: Range Nomenclature

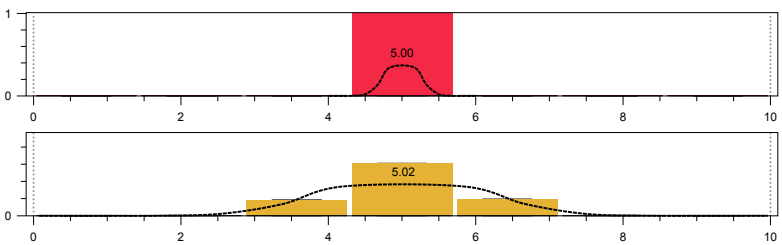


Figure 7.2: Example Ranges as Charts

7.1.1 Binary Data

Fig. 7.3 shows two bins with recurring values of 0.0 and 5.0. The test is repeated with increasing noise until the noise overlaps. It can be seen that at low noise levels, the gaussian distributions are properly determined. However, at noise levels above 2.4, the data overlaps and ranges are no longer properly generated.

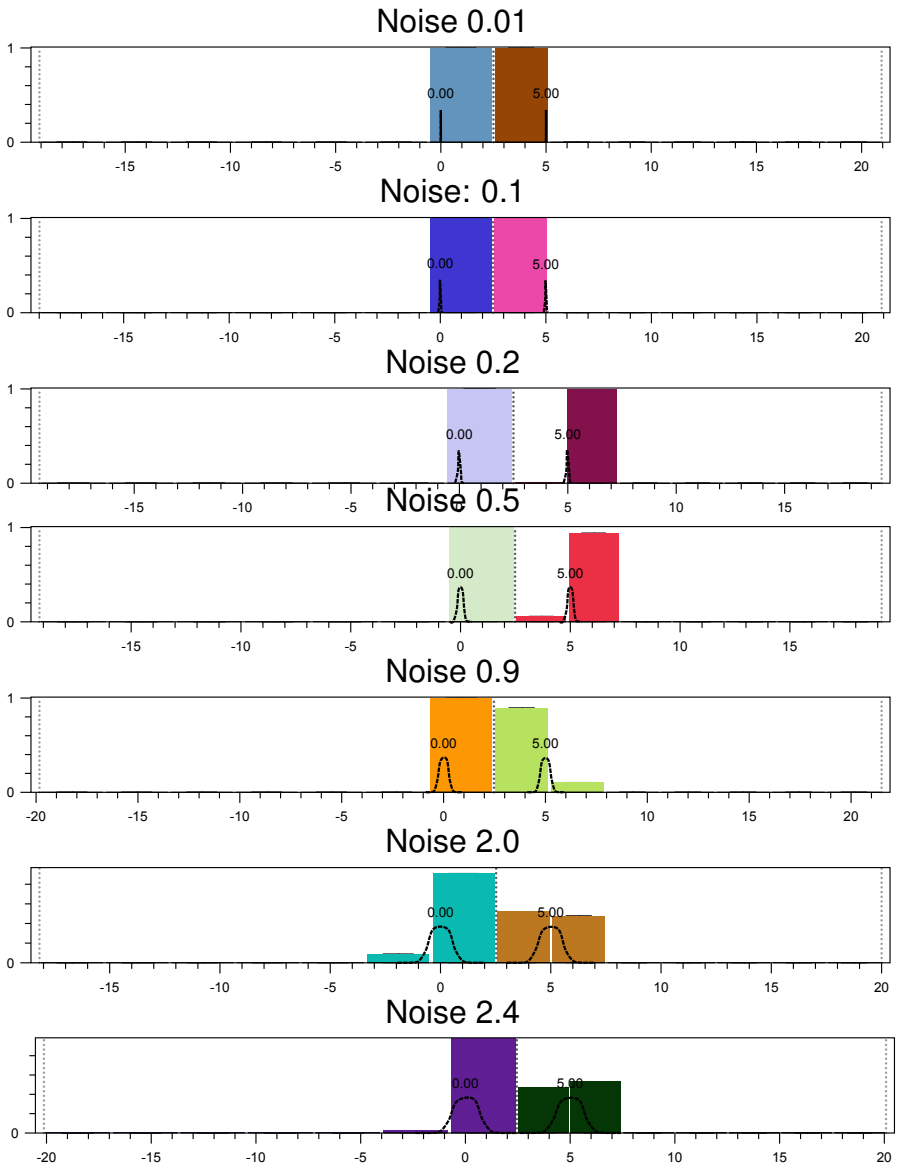


Figure 7.3: Two Ranges with Increasing Noise

7.1.2 Categorical Data

Similar to the binary data results, Fig. 7.4 also shows well-formed ranges until the noise starts overlapping. This effect can clearly be seen between the noise levels of 2.4 and 2.6.

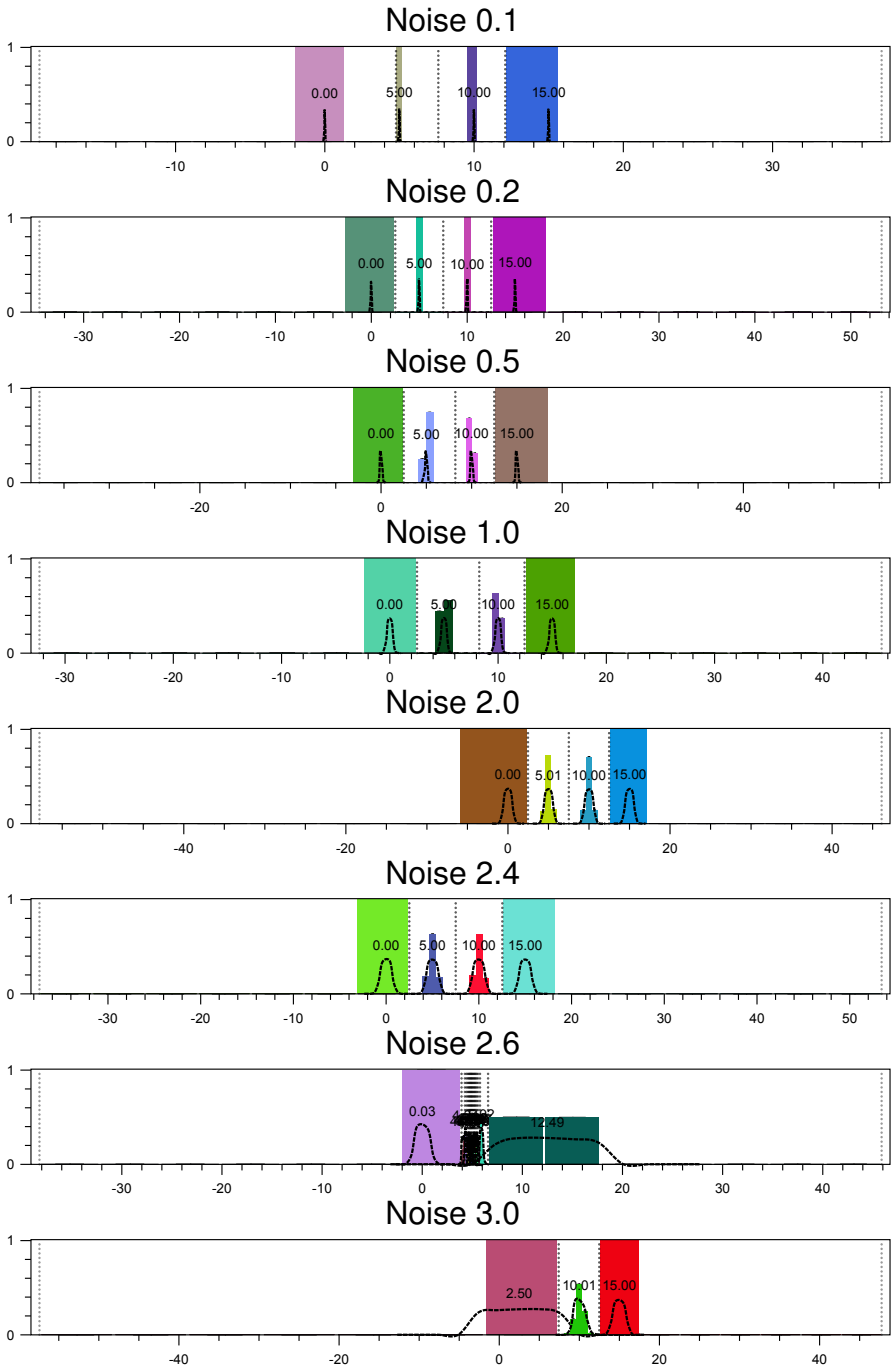


Figure 7.4: Four Ranges with Increasing Noise

7.1.3 Continuous Data

Fig. 7.5 shows a range of 0 to 10 with increasing accuracy requirements (lowering interval size). As new data in finer grainer intervals is submitted to black box, the number of ranges adapts to account for the new distributions.

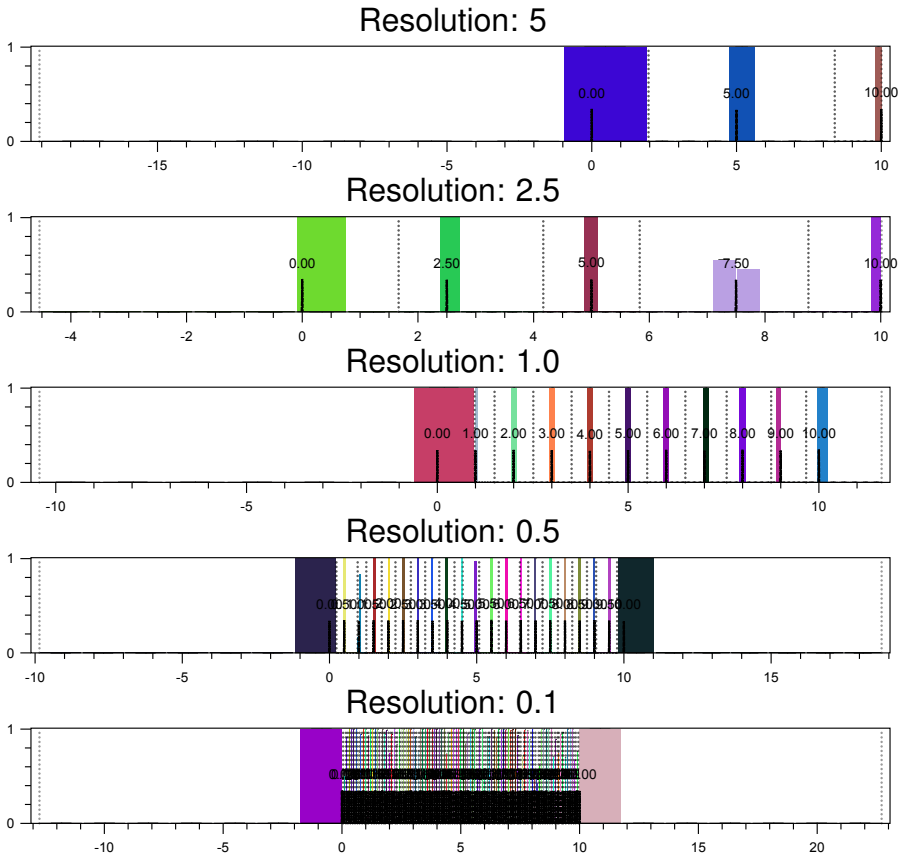


Figure 7.5: Generated Ranges vs Resolution

7.1.4 Varying Resolution

Fig. 7.6 shows a discretization process with three ranges and two different accuracy requirements. A single range represents data between -9.5 to 4.5 and another range between 11.0 to 20.5. However, there are several more ranges in between 4.5 to 11.0.

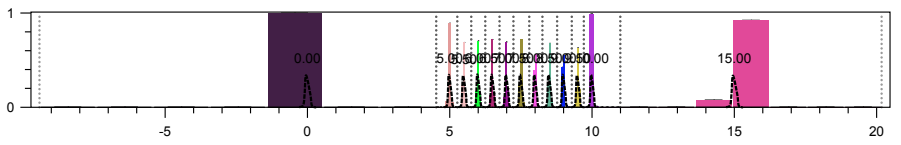


Figure 7.6: Varying Resolution

7.2 Knowledge Production

7.2.1 On-Off - Binary Data

Using the signal from Fig. 6.1, the following list of unique pieces of knowledge were identified. The 8 unique items are shown in table 7.1 and table 7.2. 2 knowledge instances represent the binary values, 2 represent $\pm\infty$, and 4 represent the transitions between values.

Table 7.1: Binary Signal 1, Gen. Knowledge

ID	Name	Content
223		(NaN) [5.00 |∞| (∞) ∞] [0]
216		(NaN) [-∞ |∞| 0.00] (∞) [0]
217	False	[0.00 (0.00) |0.00| (0.00) 3.33] [386]
226	False to True	(217; 222)
218	Stay False	(217; 217)
224	Stay True	(222; 222)
222	True	[3.33 (5.00) |5.00| (5.00) 5.00] [379]
225	True to False	(222; 217)

Table 7.2: Binary Signal 2, Gen. Knowledge

ID	Name	Content
76		(NaN) [5.00 |∞| (∞) ∞] [0]
193		(NaN) [-∞ |∞| 0.00] (∞) [0]
194	False	[0.00 (0.00) |0.00| (0.00) 1.00] [404]
201	False to True	(194; 180)
198	Stay False	(194; 194)
183	Stay True	(180; 180)
180	True	[1.00 (5.00) |5.00| (5.00) 5.00] [404]
196	True to False	(180; 194)

7.2.2 Step - Categorical

Using the signal from Fig. 6.2, the following list of 24 unique pieces of knowledge were identified (Table 7.3). 6 knowledge instances represent the ranges while 18 represent the transitions between ranges. Notice that combinations such as $0.0\ to\ 2.0\ (110; 82)$ do not exist because those values never occur after each other. Primitive knowledge instances are identified by the range details in the $content$ column.

Table 7.3: Categorical Signal Know.

ID	Name	Content
109		(NaN) [-∞ |∞| 0.00] (∞) [0]
133		(NaN) [5.00 |∞| (∞) ∞] [0]
110	0.0	[0.00 (0.00) |0.00| (0.00) 0.17] [287]
123	0.0 to 0.0	(110; 110)
111	0.0 to 1.0	(110; 81)
81	1.0	[0.17 (1.00) |1.00| (1.00) 1.20] [284]
122	1.0 to 0.0	(81; 110)
101	1.0 to 1.0	(81; 81)
112	1.0 to 2.0	(81; 82)
82	2.0	[1.20 (2.00) |2.00| (2.00) 2.00] [287]
100	2.0 to 1.0	(82; 81)
83	2.0 to 2.0	(82; 82)
113	2.0 to 3.0	(82; 85)
85	3.0	[2.00 (3.00) |3.00| (3.00) 3.11] [281]
99	3.0 to 2.0	(85; 82)
98	3.0 to 3.0	(85; 85)
114	3.0 to 4.0	(85; 89)
89	4.0	[3.11 (4.00) |4.00| (4.00) 4.60] [282]
97	4.0 to 3.0	(89; 85)
96	4.0 to 4.0	(89; 89)
135	4.0 to 5.0	(89; 132)
132	5.0	[4.60 (5.00) |5.00| (5.00) 5.00] [140]
134	5.0 to 4.0	(132; 89)
136	5.0 to 5.0	(132; 132)

7.3 Policy Training

7.3.1 Logic Operators

Figure 7.7 and table 7.4 show a quick learning of the logical operators. However, they never converge to high accuracy, especially the xor operation with a 15% error. After inspecting the decision trees, there is a theory that this is due to the more complex knowledge instances adding confusion to the policy. As such, such an investigation is mentioned for future considerations in section 8.

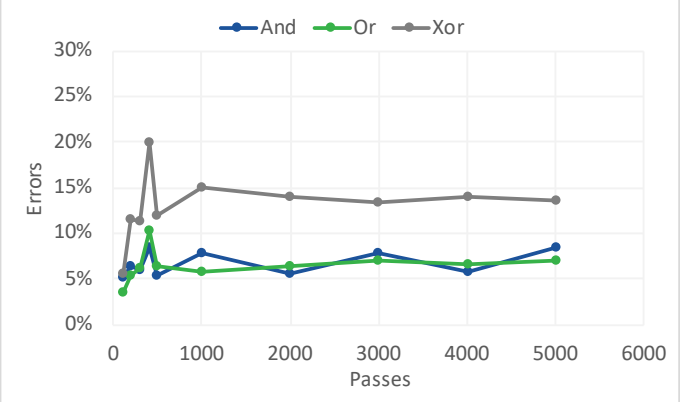


Figure 7.7: Logic Operations, Percentage Error vs Passes

Table 7.4: Logic Operations, Percentage Error

Passes	And	Or	Xor
100	0.052	0.035	0.056
200	0.063	0.054	0.116
300	0.060	0.062	0.114
400	0.085	0.102	0.199
500	0.054	0.064	0.119
1000	0.078	0.058	0.150
2000	0.055	0.064	0.141
3000	0.078	0.070	0.133
4000	0.058	0.067	0.140
5000	0.084	0.070	0.136

7.3.2 Trigonometric Functions

Figure 7.8 and table 7.5 show a decreasing MSE in the semi-continuous space until the threshold MSE of 0.1. Snapshots of the real sin, cos, and tan curves along with their predictions are shown in Fig. 7.9 at 5, 20, 30, and 35 passes. The light green, blue, and gray lines show the true value while the points show the predicted values.

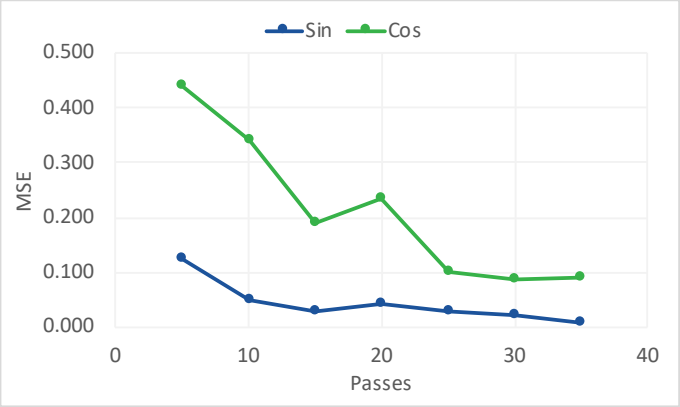


Figure 7.8: Trigonometric Functions, MSE vs Passes

Table 7.5: Trigonometric Functions, MSE vs Passes

Passes	Sin	Cos	Tan
5	0.123	0.440	840600726.029
10	0.051	0.341	88.506
15	0.030	0.191	687599294.505
20	0.042	0.234	26738111.788
25	0.030	0.102	44.134
30	0.021	0.087	29.396
35	0.007	0.090	22.547

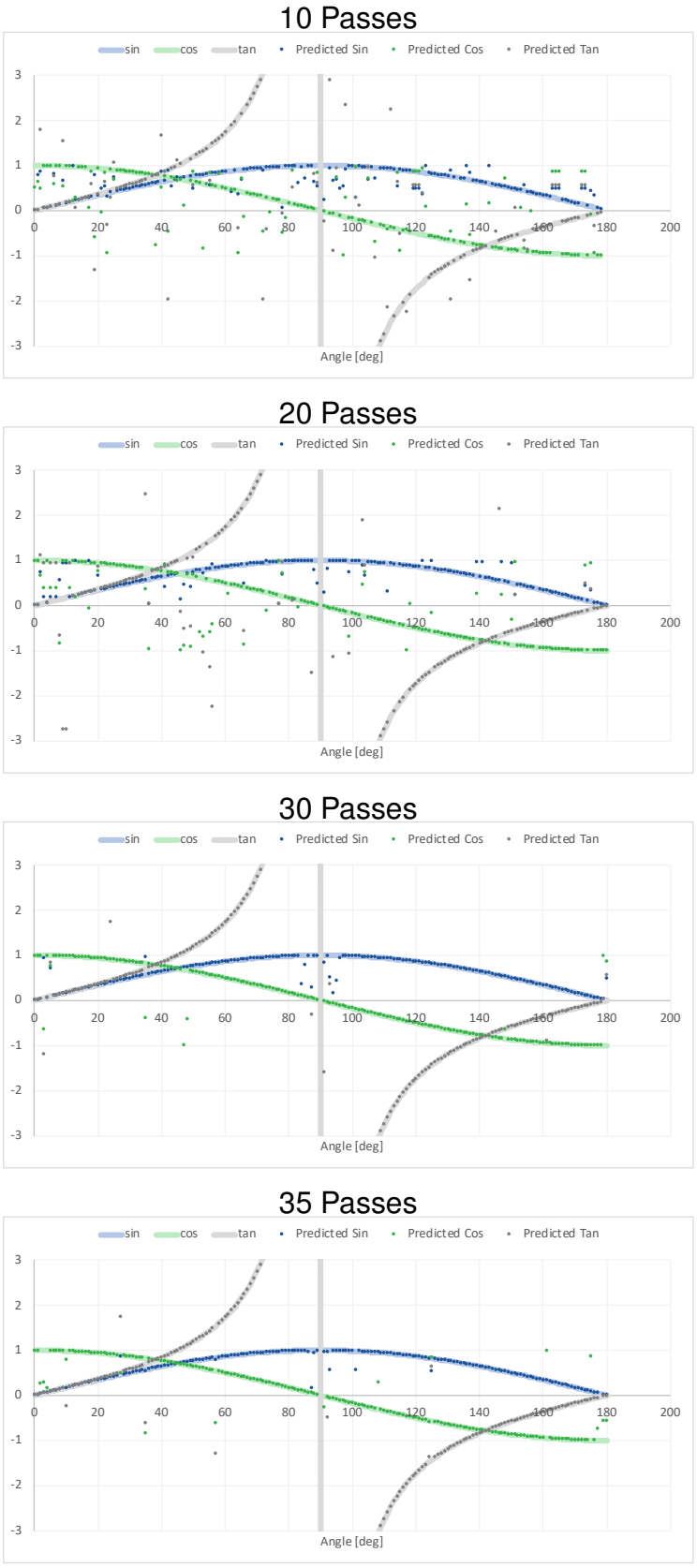


Figure 7.9: Trigonometric Functions, Actual vs Predicted

7.3.3 Robotic Arm

Although discretization of the input space occurred, discretization of the output space did not. This appears to be because of limited repetition of the values; training was only performed with 1000 cycles and required 30-60minutes. It could perhaps be solved by training for many more cycles. Nonetheless, no learning occurred during policy training.
This is however an expected result because the robotic arm black box has an inner state. Without support for a closed-loop process, control is not expected to be possible. The current learning process does not currently utilize feedback nor does the process enable comparison between knowledge instances. As such, such an extension is mentioned for future considerations in section 8.

Chapter 8

Future Considerations

8.1 Topic Points

Below is a list of topics that become apparent during development or testing of this work. They are not meant to be extensive, but rather to provide insight about any future developments. Entries in **bold** have proposed extensions in section 8.2.

1. Multiple Degrees of Freedom – Many devices require multiple inputs, so associations may be too complex for processing in a reasonable time. If possible, consider ways to decompose the complex system.

2. Prevention of Incorrect Correlations – If many devices get added to the system, it is possible that the system will confuse them. This is especially true if cross-input-output knowledge production is developed.

3. **Device Safety** – The proper input values for a given black box are not known. Some values may even be dangerous. As such prior knowledge of valid inputs is currently required, and is not ideal.

4. **Overlapping Ranges** - There exist many situations were data can be perceived as belonging to more than one class. This may be true at a low level such as at the raw signals as well.

5. Black Box Delay - Each black box system has s different delay in translating the input state to the output state. If the inputs are changed to quickly, their effect on the system can created misleading or low-confidence results in the learned policy.

6. Data Archival – The process is new and lots of data will be collected. The system may be revised later and the old data may be reusable in the new architecture.

7. Scalability – If many devices are added to the system, it will not be able to be located in one location. Processing of the various nodes will need to be distributed across multiple computers.

8. Date formats and times - The current system can only understand numeric data. However, many time and date formats are stored in a mixed text/numeric format.

9. **Closed-Loop Systems** - The current system is only capable of open-loop systems. The results of the black box outputs should be used as inputs.

10. **Comparable Knowledge** - Knowledge Instances are currently independent of each other. Information should be added allow simple greater-than and less-than operations.

8.2 Proposed Extensions

1. **Devices Safety** - The valid range of values for the inputs is not known before processing and there is a high likelihood that some values could damage the system. As such, a signal generator must be made that learns to gradually explore the unknown range, the min and max known ranges from discretization. A simple approach would be to explore these extremes until a new unique range is formed. Upon developing this new input range, continue to submit that knowledge instances through the black box until no new output knowledge is generated. However, this could be limited if locally insensitive zones exist.

2. **Closed-Loop Control** - The current system does not use the generated outputs during the decision making process. The policy learner should be adjusted to include the output and some type of heuristic function for determining distances between current output state and desired output state.

3. **Optimized Closed-Loop** - When using feedback into the system, the generated policy could be modeled as a graph search problem. The current input vector, feedback vector, and desired output vector are nodes in the graph. By finding the shortest path between them, this is the desired trajectory of inputs that should be used to control the black box towards the desired output.

4. **Expanded Knowledge Production** - The current system only considers one form of pair-wise knowledge instance generation. Other approaches using data mining or compression techniques could be introduced. This would provide alternative and potentially simplified vocabulary when deciding an action to take to achieve a desired output state.

5. **Overlapping Ranges** - A version of the discretization process may be possible with values belonging to more than one range. This would allow for different tolerances of knowledge. An example would be the simultaneous detection of colors, such as $royal-blue$, $green-blue$, and $light-blue$ (high-level knowledge) all being a type of $blue$ (low-level knowledge);

Chapter 9

Conclusions

An adaptive stream-based process has been developed for learning a set of vocabulary to control a black box. The black box is continuously sampled to monitor the state of the inputs and responses. From these samples, the value streams are discretized into defined ranges and assigned identifiers for tracking. An interpreter, combined with simple knowledge production techniques, then serves to identify unique pieces of knowledge, providing a vocabulary for describing the black box activity. Using the interpreted signals, the vocabulary is sent to a reinforcement learning-based decision tree induction process, creating a policy describing the mapping of each input to each output.

Testing was performed on three subsystems: discretization, knowledge production, and policy training. The discretization process proved to accurately identify repeated values in a stream under the condition that each unique value follows a gaussian distribution. The knowledge production process also developed the correct individual values and pair-wise values of a binary, stepped triangle wave, and pseudo-continuous sin wave, proving that binary, categorical and continuous data at a specified resolution can be learned.

Finally, the overall process was validated for an open-loop system by training on two black boxes, simulating logical operators and trigonometric functions. Using the trained policies, desired responses were produced and compared to the expected responses, enabling calculation of the error. MSE scores of less that 0.1 were achieved for the sin and cos functions. The tan function, due to its value approaching infinity at 90deg was only able to be estimated below 85deg and above 95 degrees. Interesting, the logical operators and, or, and xor were only able to achieve 85% accuracy, due to the response time of the black box. This delay in update provided misleading data to the learner. Finally, a simulation of a simple robotic arm was used to test a black box with internal state, meaning closed loop control should be required. As expected, the inputs were properly identified but the outputs were not and hence the learner was unable to produce valid policies.

Although the current iteration of this learning method is not able to develop a closed loop control process for dynamic systems, it is able to control deterministic systems and offers a framework for future development. Of the identified future considerations, device safety, closed-loop control, expanded knowledge production, and support for overlapping ranges appear to be most promising.
To support further development, all software developed during this work is available at `https://github.com/chriswblake/Knowledge-Production-and-Control-of-a-Black-Box-Using-Machine-Learning`

Bibliography

[1] Abhinav Garlapati, Aditi Raghunathan, Vaishnavh Nagarajan, and Balaraman Ravindran. A Reinforcement Learning Approach to Online Learning of Decision Trees. Technical report, Department of Computer Science, Indian Institute of Technology, Madras, 2015.

[2] Simon Kirby. Edinburgh Occasional Papers in Linguistics Language evolution without natural selection: From vocabulary to syntax in a population of learners. Technical report, 1998.

[3] Kevin L Moore. Iterative Learning Control. *Iterative Learning Control for Deterministic Systems*, pages 425–488, 1993.

[4] Derrick H. Nguyen and Bernard Widrow. Neural networks for self-learning control systems, 1991.

[5] Youqing Wang, Furong Gao, and Francis J Doyle Iii. Survey on iterative learning control, repetitive control, and run-to-run control. *Journal of Process Control*, 19:1589–1600.