University of Bergen
Department of Informatics

Machine Teaching for Explainable AI: Proof of Concept

Author: Brigt Arve Toppe Håvardstun
Supervisor: Jan Arne Telle

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June, 2022

Abstract

In today's society, AI and machine learning are becoming more and more relevant. Following this, the field of Explainable AI is becoming of more relevance. The research project "Machine Teaching for Explainable AI" aims to explain black-box AIs to humans using suitable examples. In this thesis, we present a proof of concept of the basic system in the project proposal. We aim to explain a Convolutional Neural Network trained on a boolean relation of bitmaps containing letters. The thesis introduces a simple representational language to bridge the gap between the Convolutional Neural Network and a human. We look at how to measure the perceived complexity of humans and mimic their reasoning given this task.

Acknowledgements

After completing this thesis, I would like to thank myself for the work I put into this thesis. Non the less, I would not have been able to complete this thesis without the support, encouragement and guidance of many people around me. First and foremost, I would like to thank my supervisor Jan Arne Telle for his support and motivation in dire times and for always being available for enlightening discussions. As my thesis is heavily connected to the research project "Machine Teaching for Explainable AI" I have had the pleasure of discussing my thesis with a talented group. I would especially like to thank César Ferri, Jose Hernández-Orallo and Pekka Parviainen for their input in forming my thesis. Furthermore, I would also like to thank my fellow students for the positive social and academic environment and feedback on my never-ending enquiries. I would also like to thank the administration at the Department of Informatics for providing an outstanding learning environment these last five years. Last but not least, I would like to thank my dear family and friends for their support and for believing in me, without them I would not have been able to do this.

Brigt Arve Toppe Håvardtu
Tuesday 21st June, 2022

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

In our society, we interact with machine learning daily. To strengthen trust in these systems, the field of Explainable AI (XAI) is becoming ever more critical. XAI aims to help us understand these machine learning systems. In the field of XAI, there are multiple directions to achieve this understanding. One of them is example-based XAI. With example-based XAI one aims to find examples showing how the machine learning system acts in different situations. The hope is that humans can extrapolate these examples into a general behaviour and understand how the machine learning system acts. Another field working with extracting examples is Machine Teaching. In this thesis, we develop a simple proof of concept for using Machine Teaching techniques as a tool for XAI.

1.2 Our area of interest

Machine Teaching is the research area of actively selecting data sets used in teaching. In Machine Teaching, we have different learning situations; these can be human-human, human-machine, machine-machine or machine-human learning/teaching situations. The goal is to achieve effective communication of concepts.

1.2.1 Machine Teaching introduction

Machine Teaching is a field of research focusing on actively selecting information sets such that the information conveyed from a teacher T to a learner L is as large as possible while simultaneously keeping the information set simple, as described by Zhu et al. [26].

In machine teaching, as in general, the teacher and students need to agree upon the domain in which the teaching will take place. This domain could, for instance, be the natural numbers. Within machine teaching, a domain X is the set the teacher will select examples (numbers) from to teach any concept from a concept class C. These concepts will in the simplest case be some subsets of X. The concept class C could for instance be all sets of multiples of all values i, $C = \forall_{i\in\mathbb{N}^+}[n|n \in \mathbb{N}^+, n \mod i == 0]$. T conveys the concept class C to L, and this will be the framework for the learning. T then selects a concept c to be taught, for instance all multiples of 3; $c_3 = [n|n \in \mathbb{N}^+, n \mod 3 == 0]$. To teach c_3, T would select some affirming numbers from c_3 and some negations from X/c_3, such that L understands c_3. In this example, T could, for instance, select the affirmative elements $\{3, 6, 9\}$ and negative examples $\{1, 2, 5\}$.

Another example of this given by Zhu et al. "An overview of Machine Teaching" [26] follows. Imagine we want to teach an AI θ_{AI} with input space $[0, 1]$, and output space {True,False}. Let's say θ_{AI} returns True for all $x \geq \alpha$, and False for $x < \alpha$.

$$f_\alpha(x) = \left\{ \begin{array}{l} True, x \geq \alpha \\ False, x < \alpha \end{array} \right\}$$

This gives us the concept class to be taught:

$$C = [f_i|i \in (0..1)]$$

The teacher conveys this concept class C to the student as the framework of teaching.

Let us say the teacher now select the concept $c = \theta_{AI_D} = f_{\alpha=0.5}(x)$. To describe θ_{AI_D} to a learner we could show the learner some examples of what θ_{AI_D} would output given different inputs. We could select {(0.9,$True$), (0.1,$False$)} as our Teaching Set, and show the learner this. The Teaching Set chosen is **valid** in the sense that $\theta_{AI_D}(0.9) = True$ and $\theta_{AI_D}(0.1) = False$. Based on the information in C, the learner should be able to conclude that

$$\theta_{AI_D}(x) = True, x \in [0.9..1]$$

and

$$\theta_{AI_D}(y) = False, y \in [0..0.1]$$

However, this does not give the learner information in the $(0.1, 0.9)$ range.

A Teaching Set conveying more information would be (0.51,$True$), (0.49, $False$). From this information, the learner — assuming the learner to be a rational actor — would extract more information about θ_{AI_D} based on the knowledge of C.

With this technique, we can create arbitrarily good Teaching Sets of size two for any θ_{AI}. Given a maximum allowed error e, we create the Teaching Set $(\alpha + \frac{e}{2}, True), (\alpha - \frac{e}{2}, False)$, and then the only area without perfect knowledge would be of size e.

Another learning format is *passive learning* [26]. The teacher shows the learner n random data points, and the learner concludes from this. To achieve the same confidence e with passive learning, one would need $n = O(\frac{1}{e})$ examples. One average each point is then $\frac{1}{n}$ distances apart, and hence θ_{AI_D} is expected to be between two data points being $\frac{1}{n} = \frac{1}{\frac{1}{e}} = e$ distance apart.

Selecting a Teaching Set of size two could be considered a clever selection of a Teaching Set. With this clever selection, we can better communicate what we want our third part to learn. The clever selection of Teaching Sets and clear communication is the essence of machine teaching. Instead of giving massive amounts of random data instances, we aim to provide the same information with fewer data instances.

1.2.2 Formal definition

In "An overview of Machine Teaching" [26], Zhu et al. gives a generalized formal definition of Machine Teaching:

$$\min_{D,\hat{\theta}} \text{TeachingRisk}(\hat{\theta}) + \eta\text{TeachingCost}(D) \tag{1.1}$$

$$\text{s.t} \quad \hat{\theta} = \text{MachineLearning}(D)$$

In Equation 1.1 the aim is to find a Teaching Set - D, **s.t.** when D is shown to a learner, the learner produces a guessed model $\hat{\theta}$. $TeachingRisk$ is a function describing how closely the *learners* learned concept $\hat{\theta}$ matches the concept θ^* the *teacher* want to teach.

The greater the difference the greater the value of $TeachingRisk$. To achieve the minimal $TeachingRisk$ one would therefore give the *learner* as much information as possible, such that the *learner* learns the entirety of θ^*. This solution would not be interesting, nor feasible. There is too much unique input data, hence no *learner* can be shown it all.

In contrast, *TeachingCost* is a function punishing the use of too much information in conveying θ^*. The simple solution of using all possible data is not good, as this would create a huge punishment of $TeachingCost$. In principal we have to find a balance between total information - low *TeachingRisk* and high *TeachingCost* - and a simplified explanation - high *TeachingRisk* and low $TeachingCost$. Tipping this balance in either direction is controlled by the variable η. A high value of η yields a simple explanation, while a low value yields a $\hat{\theta}$ closely matching θ^*.

1.2.3 It takes two to teach

Teaching always involves two entities, one *teacher* T and one *learner* L. The *teacher* aims to teach the *learner* a given concept θ^*. The words learner and teacher generally bring associations to a human *teacher* working in a school, and $learners$ are the pupils of this teacher. However, this is not the only situation with a *teacher* and *learner* relation.

As discussed by Zhu et al. [26] we can have human-human, human-machine, machine-machine or machine-human teaching situations - (first being teacher, second being learner). Each of them comes with different difficulties and possibilities. Let us look briefly into what each of these entails.

Human-human teaching

Human-human teaching is the format of teaching that springs to mind for most people if they hear the word learning and teaching. It is the example given from the schoolroom and how we collaborate with our peers. As humans, we have gained experience in how to teach other humans new concepts and can draw from our own experiences. For instance, in human-human teaching, we often communicate with more than just our words. Body language is an essential part of effective communication [7].

Coming into the field of Machine Teaching, it is vital to remember the biases we have learned about good teaching techniques in human-human teaching situations, as these might not be the best techniques for other learning situations.

Human-machine teaching

In machine learning, humans are teaching machines the concepts of driving cars [3] or the concepts of humans, chairs and dogs in images [2]. A wildly popular strategy is supervised learning [6]. In supervised learning, humans generate large amounts of labelled examples and hope the machine can generalize the concept. This is possible as long as the problem is PAC-learnable [23]. PAC-learnable says something about what level of model error we can expect given a set of random instances from the training data drawn from a probability distribution.

However, humans can also create more "handcrafted" training data for the machine. As discussed by Zhu et al. [26] a human with the knowledge of the learning gradient of a machine would be able to predict the guessed model $\hat{\theta}$ by the machine after viewing a new instance. The human could therefore create a new instance with their knowledge of the gradient to nudge the $\hat{\theta}$ closer to the model θ^* the human wants .

Machine-machine teaching

Looking at machine-machine teaching, we could envision a teacher machine with knowledge of the gradient of a student machine selecting training data to make a Teaching Set. Having a machine with access to another machine's gradient is similar to the ideas talked about in Human-machine teaching. Having a machine select Teaching Set to nudge the gradient of another machine seems reasonable at first glance. Instead of having a human perform the laborious selection of the Teaching Set, a machine could do this. Although this sounds nice, one arrives at the problem that the teacher needs to have the model θ^* it wants to teach already understood. It can be hard to imagine a situation where we have a machine understanding θ^* and still want to teach another machine to understand the same concept θ^*.

Presented by Zhu et al. [26] we see one malicious use case, namely poising attack [10]. In a poising attack, the aim is to make minimal changes in the training data - e.g. adding/changing/deleting examples in the training data - s.t., a post condition of the trained model is satisfied.

$$\begin{aligned} \min_{D} \quad & ||D_0 - D||_p \\ \text{s.t.} \quad & \Phi(A(D)) \end{aligned} \tag{1.2}$$

The notation used in Equation 1.2 matches the notations established at this chapter's beginning for machine teaching. The measure of the post condition being satisfied, $\Phi(A(D))$, can be viewed as $TeachingRisk$. At the same time, minimizing editing on the original training set D_0 into the new training set D matches the $TeachingCost$.

Machine-human teaching

Machine teaching in a machine-human teaching environment will be the focus of this thesis. As presented by Zhu et al. [26] the most common and exciting use of machine-human teaching is to have a machine teach a human. The machine needs to have a model of the human to figure out how the humans will respond to different data instances. These models can be Generalized Context Models [15], Occam's Razor [4] or other systems used to approximate the human mind. A robust machine-human teaching system would open up many possibilities in the real world. One could, for instance, have a machine teach a human some black-box AI, ensuring the human that the AI's actions are aligned with the human's beliefs of the AI. Another example could be in the education system. Today one human teacher is teaching upwards of 15-20 pupils alone [1], and in introductory courses at the university, this can reach upwards of a few hundred students [8]. Using machine-human teaching, one could target students with individual examples best suited for them.

1.3 Problem statement for the thesis

This thesis will build on work presented in the research proposal "Machine Teaching for Explainable AI"[21]. The project aims to further develop **Machine Teaching** as a tool for **Explainable AI**, specifically example-based teaching. In the research proposal, Equation 1.3 was put forwards as a framework for the research project.

$$T(\theta_{AI}) = \underset{S:\theta_{AI}\models S}{\operatorname{argmin}}\{\delta(S) + \lambda(\theta_{AI}, \theta_M) : L_M(S) = \theta_M\} \tag{1.3}$$

$$L_M(S) = \underset{\theta_M:\theta_M\models S}{\operatorname{argmin}}\{\beta(\theta_M)\} \tag{1.4}$$

In these equations T is a teacher, aiming to teach a concept θ_{AI} to a human learner L_H. T aims to find a **Teaching Set** S such that $L_H(S) = \theta_{AI}$. To achieve automation and increase iteration speed a model L_M of L_H is used. T will therefore aim for $T(\theta_{AI}) = S$ **s.t.** $L_M(S) = \theta_{AI}$.

δ is a complexity measure of a potential Teaching Set S, punishing complex examples.

λ is a compatibility score, measuring the similarity of θ_{AI} and $L_M(S)$. λ measures how closely the guessed concept θ_M matches the concept one wants to teach θ_{AI}. In this thesis, we discuss different implementations of Equation 1.3 and aim to provide a Proof of Concept (PoC), showing the viability of this framework. In particular we discuss definitions and implementation of θ_{AI}, L_M, β, $\text{argmin}_{\theta_M:\theta_M \models S}$, λ, δ, and $\text{argmin}_{S:S \models \theta_{AI}}$.

1.3.1 Teaching Set S

In the thesis, we will be discussing different Teaching Sets S to teach the concept θ_{AI}. We therefore define a Teaching Set S to consist of multiple (labelled) examples $S = \{e_1, e_2, .., e_m\}$. One example e_i contains a data instance B_j, as well as $\theta_{AI}s$ predictions of that data instance $\theta_{AI}(B_j)$. We therefore write $e_i = (B_j, \theta_{AI}(B_j))$ to describe an example.

1.3.2 γ - Scoring function

In the Equation 1.3 we seek the minimum of $\delta(S) + \lambda(\theta_{AI}, \theta_M)$. We need to carefully balance $\delta(S)$ and $\lambda(\theta_{AI}, \theta_M)$. If δ returns many order of magnitude larger values than λ, we would always get the empty Teaching Set. If λ returned values orders of magnitude larger than δ, one would get massive Teaching Sets containing all possible data instances. To balance $\delta(S)$ and $\lambda(\theta_{AI}, \theta_M)$ one could adjust the functions δ and λ themselves to achieve a good balance. However we will be defining $\gamma(S, \theta_{AI}, \theta_M, \mu) = \delta(S) + \mu * \lambda(\theta_{AI}, \theta_M)$. This allows us to use a single value μ to express which of $\delta(S)$ and $\lambda(\theta_{AI}, \theta_M)$ are to be weighted more. The use of μ in this manner can be observed in "An Overview of Machine Teaching" [26].

1.4 Visual overview of system

We present a diagram of the system we made in the thesis. Hopefully, the reader finds this diagram helps to understand the proof of concept presented.

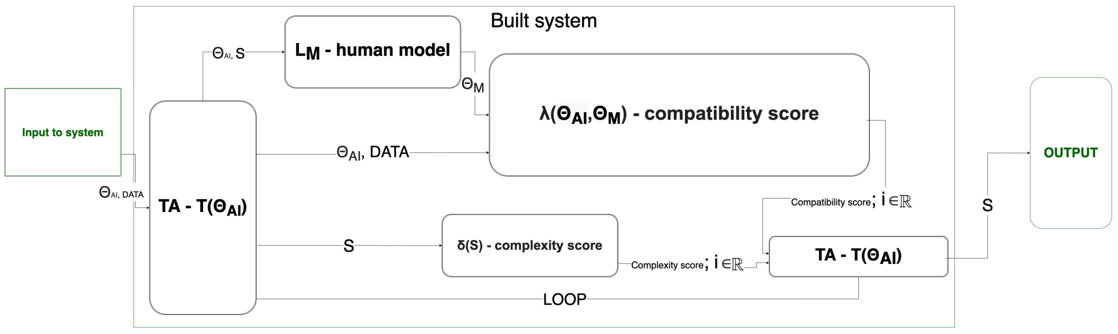


Figure 1.1: Overview of connection of subsystems. A detailed overview can be found in Appendix A.1.

In the Figure 1.1 we show how the information flows from one part of the system to another. It starts within the control system TA. TA selects a Teaching Set S to be evaluated. δ evaluates S and reports a score. Meanwhile, L_M also receives S and produces a guessed concept θ_M. θ_M is passed onward to λ where a *compatibility* score is calculated. We get a score for each Teaching Set and iterate on these to finally return the best scored Teaching Set.

1.5 Overview of chapters

We now give an overview of the work we will present in this thesis. This first chapter discussed the general aspects of Machine Teaching, and we introduced the work on which we will make a proof of concept in Equation 1.3. As we aim to work with Machine Teaching for Explainable AI, we introduce a task and an AI to solve this task in Chapter 2. This AI will be the AI we aim to explain in our PoC. In the following Chapters 3, 4 , 5 and 6, we define the system we built. In Chapter 3, we define the model of our human L_M as presented in Equation 1.4. After defining L_M, we define how we measure the compatibility of L_Ms guessed model θ_M with θ_{AI} in Chapter 4 by defining λ. We then define the δ-complexity of a Teaching Set in Chapter 5. With δ and λ defined we can compare the score of different Teaching Sets. To conclude the implementation overview, we discuss how to traverse the search space of Teaching Sets to find the optimal S by defining $\text{argmin}_{S:S\models\theta_{AI}}$ in Chapter 6. We then discuss different implementation versions, and general results of running the system in Chapter 7. To conclude the thesis, we summarise the work and potential directions for future work in Chapter 8.

Chapter 2

The AI we want to explain - θ_{AI}

As a reminder, we repeat the main formula:

$$T(\theta_{AI}) = \underset{S:\theta_{AI}\models S}{\operatorname{argmin}}\{\delta(S) + \lambda(\theta_{AI}, \theta_M) : L_M(S) = \theta_M\}$$

$$L_M(S) = \underset{\Theta_M:\Theta_M\models S}{\operatorname{argmin}}\{\beta(\Theta_M)\}$$

We need concrete implementations of all these concepts. As we are to explain some AI model θ_{AI}, we first define θ_{AI}. In this Chapter, we define a practical problem and discuss implementations of the AI leading to a concrete θ_{AI}. An diagram overview of this is shown in Figure 2.1.

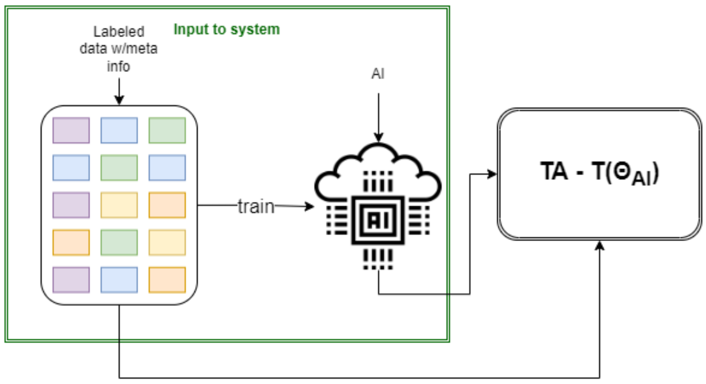


Figure 2.1: Overview of the input to our system.

2.1 Definition of θ_{AI}s task

We had a few attributes in mind when selecting the task our AI model would solve. We did not want something too complex; it had to be simple to implement and change, as we were to make a Proof Of Concept. At the same time, the problem should not be too simple, we want to check if our method will convey the AI model's errors concerning the ground truth. We also wanted to be able to test our system with different AIs. To achieve this, we wanted the ground truth to be easily changeable. An easily changeable ground truth ensures we can compare AIs trained on different ground truths.

Ultimately, we chose that θ_{AI}'s task should be to learn a Boolean function on four variables: $\phi(A, B, C, D)$. This ϕ is changeable in the system; however, we will be using the Boolean function $\phi = (A \wedge B) \vee (C \wedge D)$ in most examples.

The input to θ_{AI} will be a bitmap containing a subset of letters. The bitmaps representation of literals gives us the possibility of extensive training data to train our AI. In Figure 2.2 we show such a bitmap. We bring attention to the fact that the bitmap shown in Figure 2.2 contains the letters A and D. All input bitmaps will contain a subset of the letters in our alphabet $\Sigma = \{A, B, C, D\}$.

The output space of θ_{AI} is $\{0, 1\}$. We label an example 1 if $\phi = (A \wedge B) \vee (C \wedge D)$ evaluates to True, and 0 if ϕ evaluates to False. To check ϕ given a bitmap, we find the present letters and evaluate ϕ with present letters equal to True and the rest set to False.

For the bitmap shown in Figure 2.2 we would have $\phi(A = 1, B = 0, C = 0, D = 1) = (A \wedge B) \vee (C \wedge D) = (1 \wedge 0) \vee (0 \wedge 1) = 0$, and hence a label of 0. On the other hand the bitmap shown in in Figure 2.3 would be evaluated as $\phi(A = 0, B = 1, C = 1, D = 1) = (0 \wedge 1) \vee (1 \wedge 1) = 1$, and a label of 1. The goal is for θ_{AI} to learn this boolean relation from bitmaps to truth evaluation.

This problem fits our demands as we can easily create a new ground truth by swapping the ϕ function.

2.1.1 The ground truth Boolean function ϕ

The ground truth function used to train θ_{AI} is a Boolean function ϕ. ϕ could be any Boolean function on the literals in our alphabet $\Sigma = \{A, B, C, D\}$. All Boolean functions

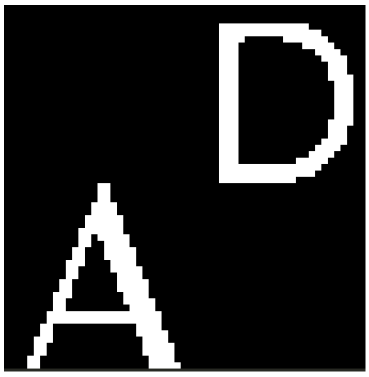


Figure 2.2: Bitmap AD

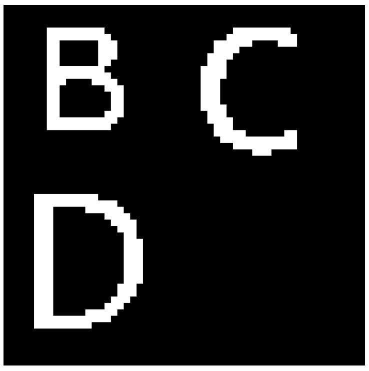


Figure 2.3: Bitmap BCD

are expressible by a truth table of size $2^{|\Sigma|}$, in our case $2^4 = 16$. As the rows in the truth table can hold either 0 or 1 as their evaluation value, the total number of possible truth tables is $2^{16} = 65536$. 65536 unique ϕ functions gives us 65536 unique θ_{AI} ground truth functions. This vast range of ground truth functions allows us to compare our system on different θ_{AI}.

Each bitmap can contain a subset a of the alphabet Σ. We will be using this subset notation quite often so we define the subsets of Σ to be Σ_C:

$$\Sigma_C = \{\#, A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD, ABCD\}$$

By $\phi(a)$, for some string $a \in \Sigma_C$, we mean ϕ evaluated with only letters present in a set to True. For example, given a = ACD, we can write this as $\phi(a) = \phi(A = 1, B = 0, C = 1, D = 1)$.

2.2 Defining the data set

Having defined the problem at large, we next define the input data. We discuss design choices in defining the data set and different parameters accessible when producing it.

2.2.1 Description of an instance

In more detail, the input to our AI will be a 64x64 bitmap. The cells in the bitmap will either have the value 0 or 1. The cells having the value 1 will together make up one or more letters from our alphabet $\Sigma = \{A, B, C, D\}$. The Figures 2.2, 2.3 depicts two examples of the bitmaps we will be giving as input to θ_{AI}.

We want to have some specific properties for each data instance. These are mostly sanity checks to ensure our AI will receive decent data. Although intuitive, it is essential to make sure our data is sane. Poor data in means poor predictions are coming out.

Our rules for the data go as follows:

- For each letter, the entirety of the letter is to be contained within the picture.
- No letter is to overlap with another letter.
- The size of the letter should not be too small. The letter shall contain enough pixels such that the characteristics of the letter are maintained, i.e. recognizable for a human.

Given our alphabet $\Sigma = \{A, B, C, D\}$ we uniformly select each image to contain one of the 16 possible combinations in Σ_C.

Because we uniformly select from Σ_C, it means $\frac{1}{16}$ of our data set will be blank images which all are identical. Having this many duplicates might not be ideal. However, this gives θ_{AI} an evenly distributed data set among all data instances. We believe these negative effects will be minor and will therefore not look more into it.

2.2.2 Variance when generating instances

When generating data, we have made some parameters to control the level of variance in the data set. These parameters control the complexity of our data. By adjusting these parameters, we can achieve a smooth increase in difficulty for our AI. Smoothly controlling the data sets' difficulty allows us to find a suitable data set for our AI model, where it performs reasonably well.

$FixedSquares$ – placement of letter

For the parameter $FixedSquares$, we divide the image into four squares –top left, top right, bottom left, and bottom right, illustrated in Figure 2.4. Each of these squares might then contain a letter. Crucially no letter is overlapping these squares. Turning this parameter on will make the data set less complex, as its counterpart is just freely placed letters randomly in the image. Without $FixedSquares$, the algorithm might find all letters at the top or bottom of the image, increasing the data set's variance. $FixedSquares$ corresponds to a less complex data set, more prone to be easily overfitted.

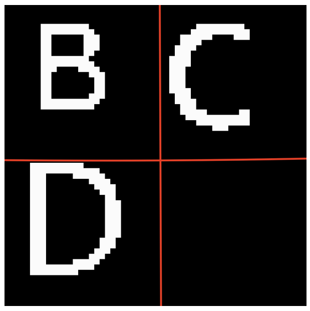


Figure 2.4: Illustration of *FixedSquares*

Rotation – **orientation of the letter**

With the parameter *Rotation*, we allow the letters to have different orientations. We have selected a subset of orientations $O = \{0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°\}$. Letting each letter take on one of these orientations increases variance and complexity in our data set. The bitmap in Figure 2.5 is an example of an image from a data set with *Rotation* allowed.

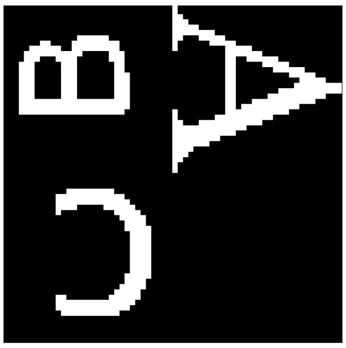


Figure 2.5: Bitmap w/rotation ABC

Scale - **size of the letter**

With *Scale*, we allow letters to have different sizes. A broader range of sizes on the images will increase the variance and complexity of our data set. Following the rules we set for the sanity of the data set, we note that no letter can be made too big, Figure 2.6, as it has to fit inside the image. Nor can it be made too small, Figure 2.7, as it needs to be recognizable. An example of bitmap with both rotation and scale is given in the Figure 2.8.

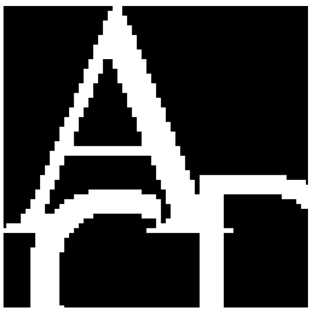


Figure 2.6: Bitmap ACD, too big. Overlapping, and out of image.

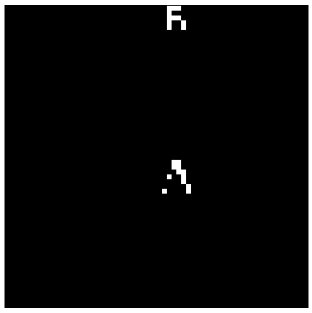


Figure 2.7: Bitmap AB, too small. A is broken up, B looks like R.

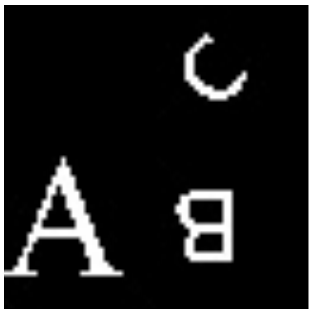


Figure 2.8: Bitmap with rotation and scaling.

2.2.3 Labels

After describing our data instance, we now describe the corresponding labels in the data set. Our labels will hold either the value 0 or 1. We use the function ϕ when generating a label. When evaluating ϕ, we set the present letters in the bitmap to True and letters not present to False. For instance given $\phi = (A \wedge B) \vee (C \wedge D)$ 2.2 would be False, and hence get label 0, while 2.3 would be True and have label 1.

2.3 Implementation of θ_{AI}

After defining the data we will be using, we needed to implement an AI system to train on this data set. As this was going to be the **black-box system** we hoped to explain, we aimed to select an AI system not easily interpreted by itself. In recent times systems using deep Convolutional Neural Network (CNN) [12, 13] have seen huge success [13, 17, 25]. However, as these networks have grown in depth to achieve better results [18], they continue to become more obscure and less interpretable. CNNs are often used in image recognition or classification because they are spatially sensitive to neighbouring values – whereas, e.g. Fully Connected Neural Networks are not. The fact that CNNs are successful at their task, while at the same time not interpretable by themselves, makes CNN a good choice for our θ_{AI}.

We implemented a CNN in python using Keras and TensorFlow. The Figure 2.9 illustrates the layers of the CNN implemented. The diagram displays each layer's size and the number of channels. The first couple of layers is a series of convolutions and max-pooling, with the size decreasing while the number of channels increases. In the

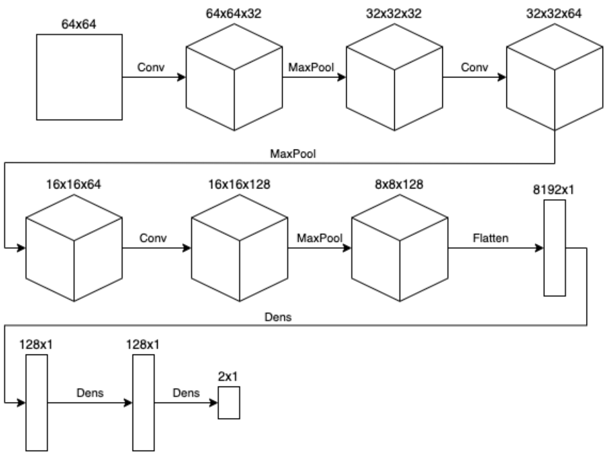


Figure 2.9: Illustration of CNN implementation. Conv = Convolutional layer. Dens = Fully dense layer

end, we flatten the layers into a 1-dimensional vector. We then have two dense layers before ending with a softmax on 0 or 1.

Although not depicted in the illustration, we use an activation function followed by a dropout layer after each convolution. The activation function used is LeakyReLU with $\alpha = 0.1$. The dropout layer has dropout rates of $[0.25, 0.25, 0.4]$.

In Listing 2.1 we see the exact code used to generate the CNN model. For convolutions, we are using kernels of size (3,3), stride (1,1) and padding "same". When MaxPooling, we are using filter size (2,2), stride (2,2) and padding "same".

```python
def _set_layers(self):
        """
        We have a set of (3,3) kernels having 'same' padding. The
            ↪ number of kernels keeps increasing.
        In the end, we have two dense layers, which then is fully
            ↪ connected.
        Activation function: LeakyReLU
        Pooling: MaxPool (2,2)
        """
        input_width = IMAGE_WIDTH
        input_height = IMAGE_HIGHT
        channels = 1

        num_classes = 2
        model = self.model
        model.add(Conv2D(32, kernel_size=(3, 3), activation='linear',
                  padding='same', input_shape=(input_width,
                     ↪ input_height, channels)))
        model.add(LeakyReLU(alpha=0.1))
        model.add(MaxPooling2D((2, 2), padding='same'))
        model.add(Dropout(0.25))
        model.add(Conv2D(64, (3, 3), activation='linear',
            ↪ padding='same'))
        model.add(LeakyReLU(alpha=0.1))
        model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
        model.add(Dropout(0.25))
        model.add(Conv2D(128, (3, 3), activation='linear',
            ↪ padding='same'))
        model.add(LeakyReLU(alpha=0.1))
        model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
        model.add(Dropout(0.4))
        model.add(Flatten())
        model.add(Dense(128, activation='linear'))
        model.add(LeakyReLU(alpha=0.1))
        model.add(Dropout(0.3))
        model.add(Dense(128, activation='linear'))
        model.add(LeakyReLU(alpha=0.1))
        model.add(Dense(num_classes, activation='softmax'))
```
Listing 2.1: Code for the structure of CNN model

2.4 Training results of θ_{AI}

In this section, we look into θ_{AI}s performance given different data sets. We begin with the least complex data set, and move on to more complex data sets. As there is little to no interest in teaching θ_{AI} if it is acting somewhat randomly, we aim for a 95% accuracy for us to consider it teaching-worthy. The accuracy of our CNN is the percentage of data instances predicated correctly:

$$accuracy = \frac{\#\ correctly\ predicted\ data\ instances}{total\ \#\ data\ instances}$$

We will change the size of the data set to give θ_{AI} enough data to learn more complex data sets. We emphasize that we continue to use the Boolean function $\phi =$ **(A and B) or (C and D)** as our ground truth for all data sets.

2.4.1 Least complex data set

We generate a data set containing 2000 images. We did not allow any *rotation* or *scaling* of the images. We bounded each letter to be placed in one of four corners with $FixedSquares$. The resulting images look like Figure 2.3.

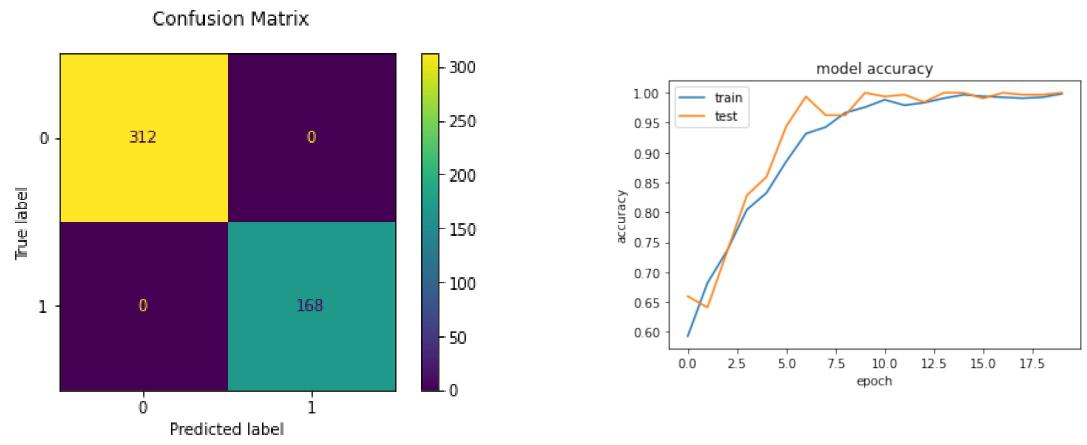


Figure 2.10: Confusion matrix and accuracy history low variance data set

As expected, we observe from the Figure 2.10 that our system can learn this data set relatively quickly as the data is of low variance and complexity.

2.4.2 Introducing rotation

After the least complex data set, we introduce more complexity. We do this by allowing *rotation* in the data set and hence allow rotation of each letter. The resulting images looks like Figure 2.5. The data set with the least amount of complexity performed well when trained on a data set containing 2000 data instances. However, this was not the case after introducing *rotation*. We therefore increased the data set size to 5000.

We observe that θ_{AI} performs well on a data set of size 5000. From the confusion matrix, we observe that the accuracy is well above 95% as shown in Figure 2.11. From the same figure we also observe that θ_{AI} makes 41 mistakes when the actual label is 1 and zero mistakes when the actual label is 0. We suspect this might happen because the data set contains an overweight of data instances with 0 as the label. Therefore, the data set incentivises θ_{AI} predicting more data to be labelled 0. Non the less, we conclude that θ_{AI} learned $\phi = (A \wedge B) \vee (C \wedge D)$ quite well.

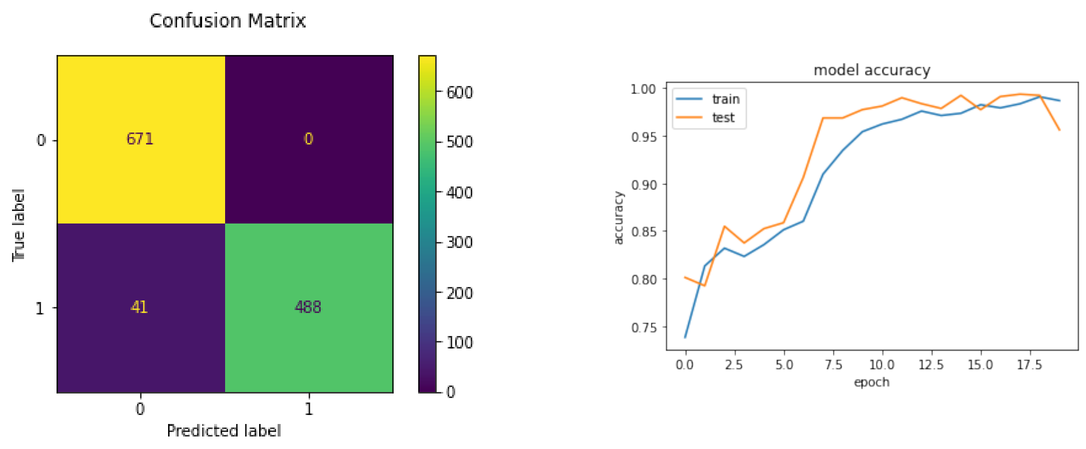


Figure 2.11: Confusion matrix and accuracy history w/rotation data set

2.4.3 Rotation and scaling

To further increase the complexity, we allow each letter to differ in size by allowing *Scale*. The bitmaps in the data set now looks like the Figure 2.8. The most enlarged letters will be 4 times the size of the smallest letters in area. Having both *Scale* and *Rotation* should further increase the variance of the data set and hence might be more challenging for θ_{AI} to learn. We again changed our data set size, now up to 10000, such that our model got a 95%+ validation accuracy score.

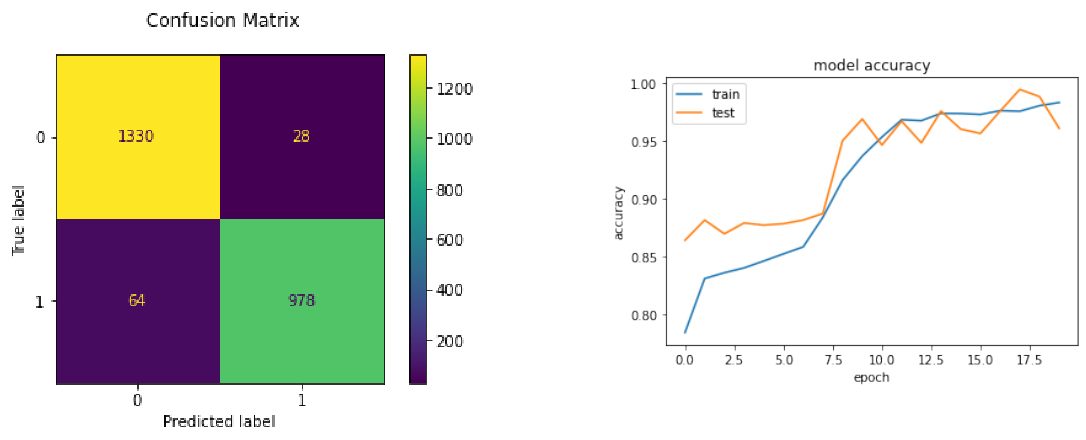


Figure 2.12: Confusion matrix and accuracy history data set with rotation and scaling

With both *rotation* and *scaling* the model θ_{AI} performs reasonably well when trained on a data set of size 10000, as seen in Figure 2.12. Generating this data and training θ_{AI} takes a couple of minutes.

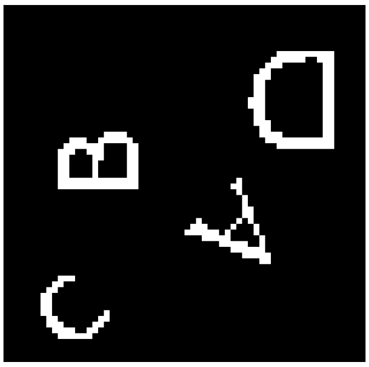


Figure 2.13: Bitmap ABCD with free placement, rotation and scale.

2.4.4 Adding free placement of letters

For the test of our most complex data set we allow *rotation*, *scaling* and $freePlacment$ of letters. The bitmaps in this data set looks like the Figure 2.13. As this is the most complex data set, we suspect θ_{AI} will need significantly large amounts of data to converge. We therefore trained it using a data set with 40000 data instances.

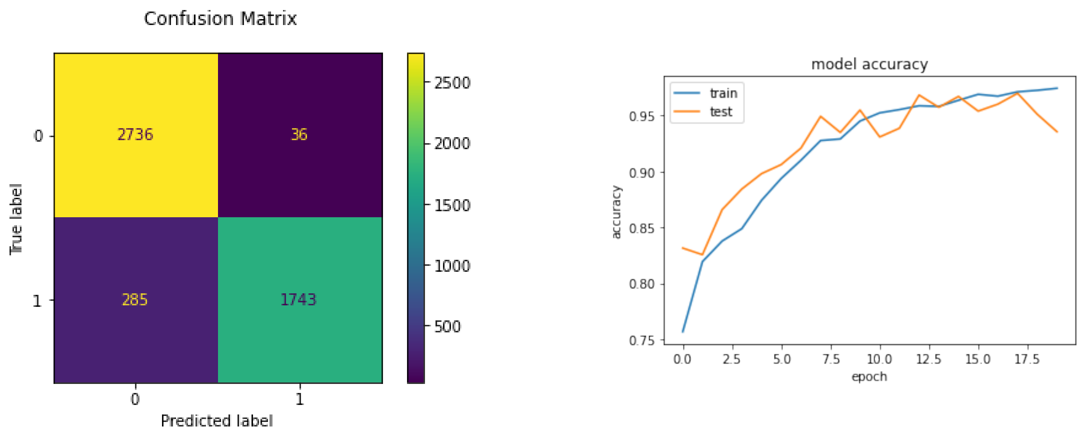
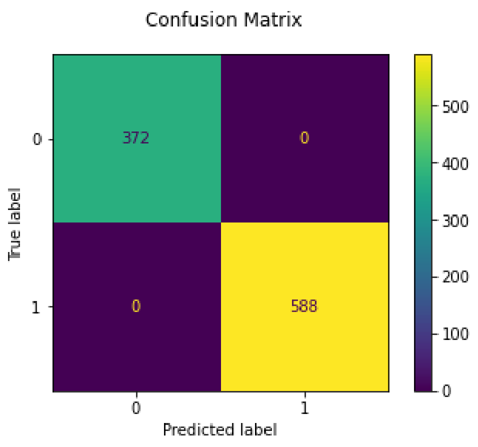


Figure 2.14: Confusion matrix and accuracy history data set with rotation, scaling and free placement

From the Figure 2.14 we observe θ_{AI} is performing reasonably well given 40000 data instances. However, to achieve this, we spent 1-2 hours generating data and then θ_{AI} spent 10-20 minutes training. Using this much time on generating data brings the iteration speed of updating to a too high level; hence we will not use this.

2.4.5 Reducing alphabet

To get θ_{AI} to perform well while increasing variance in our data set, we have had to increase our data set size from 2000 images to 40000 images. Later in the thesis, we will see that reducing the alphabet from $\Sigma = \{A, B, C, D\}$ to $\Sigma = \{A, B, C\}$ can bring a huge speedup. We therefore evaluate how well θ_{AI} does with *rotation* and *scaling* with $\Sigma = \{A, B, C\}$. We change ϕ to $\phi =$ **(A and B) or C**, and a data set containing 4000 instances.



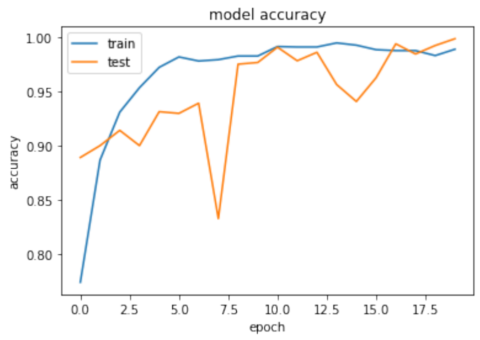


Figure 2.15: Confusion matrix and accuracy history data set with rotation, scaling and free placement

Given that our alphabet Σ has become smaller we can use a smaller data set. When we had $|\Sigma| = 4$ and a data set with *rotation* and *scale* we needed 10000 data instance to get a 95%+ accuracy. With $|\Sigma| = 3$, and 4000 data instances we get a θ_{AI} predicting perfectly in the confusion matrix as seen in Figure 2.15.

2.4.6 Conclusion

We observe that θ_{AI} performs well on data sets with *rotation* and *scaling*, and these data sets have fewer duplicates compared to not using *rotation* and *scaling*. The size of the data set needed to train θ_{AI} is also such that it takes a reasonable amount of time (minutes) to train a new model.

Although intuitive we also show how θ_{AI} needs a smaller data set if we decrease the alphabet Σ to $\Sigma = \{A, B, C\}$. Hence we will be using a data set with *rotation* and *scaling* in the rest of the work of this thesis. The alphabet will be specified to either $\{A, B, C, D\}$ or $\{A, B, C\}$ throughout.

Chapter 3

Model of the human

In the previous chapter, we defined the AI we will be teaching to a human L_H. As we are going to be using example-based teaching, the goal of our system is to find a Teaching Set S we can show to a human, such that the human infers the correct concept, i.e. L_H s.t. $L_H(S) = \theta_H = \theta_{AI}$. One way to do this is to generate a range of different Teaching Sets S and show each of them to L_H and compare L_Hs guessed model θ_H to θ_{AI}. The Teaching Set would then be S s.t. θ_H is most similar to θ_{AI} and yet S is not too complex.

However, as discussed in the introduction, humans are incredibly slow compared to machines, and we can evaluate orders of magnitude more S if we use a model of human L_M instead of an actual human L_H. We can see how this model human fits into our formula in Equation 3.1 and is further defined in Equation 3.2. We also give a diagram over L_M in Figure 3.1.

$$T(\theta_{AI}) = \underset{S:\theta_{AI}\models S}{\operatorname{argmin}}\{\delta(S) + \lambda(\theta_{AI}, \theta_M) : L_M(S) = \theta_M\} \tag{3.1}$$

$$L_M(S) = \underset{\theta_M:\theta_M\models S}{\operatorname{argmin}}\{\beta(\theta_M)\} \tag{3.2}$$

This chapter discusses the implementation of this human model L_M. [1]

[1]As an aside, let us note that given more time for this thesis, and implementation using L_M to find good teaching sets, we would test these out on humans L_H. Best on these results, we would possibly redefine L_M for a better fit of L_H, and iterate.

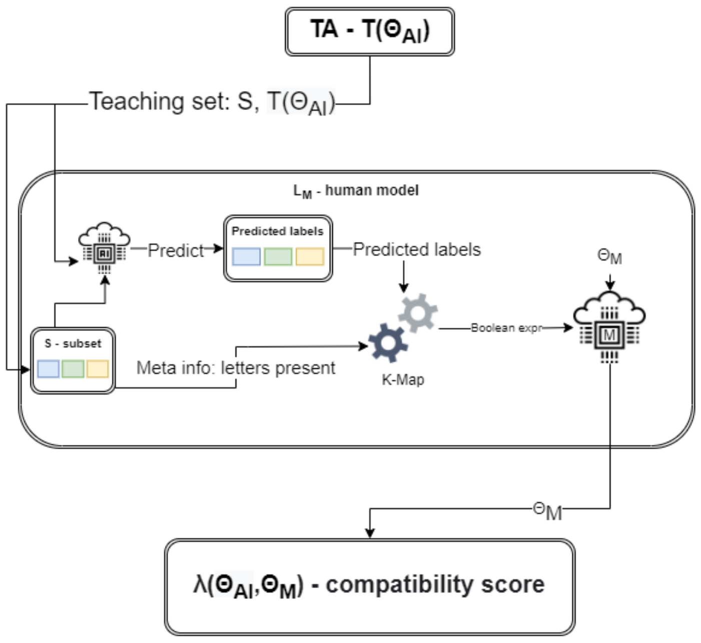


Figure 3.1: Diagram of L_M.

3.1 What humans see - Representational language

As we want to model the human, we must reflect on how a human would react to the bitmaps we present in a Teaching Set. We argue that most humans would extract the meta-information:

- There is/is not an A in this picture.
- There is/is not a B in this picture.
- There is/is not a C in this picture.
- There is/is not a D in this picture.

Moreover, we hypothesise that most people will pay the most attention to the letters present and more or less disregard other information such as rotation, size and position.

For simplicity, our L_M will not be given bitmaps as examples. Instead, it takes as input the letters present in each image. To further define the notation of present letters, we define two different representational languages.

The representational language of bitmaps R_B are examples as described thus far, bitmaps paired with θ_{AI}s predictions. We denote Teaching Set S^B and examples e^B when in R_B:

$$S^B = \{e_1^B, e_2^B\}$$

$$e^B = (B_k, \theta_{AI}(B_k)), \quad \textit{for a bitmap } B_k$$

On the other hand, we have the representational language of present letters R_L, encapsulating the idea of humans recognising letters. We denote Teaching Sets S^L and examples e^L when in R_L:

$$S^L = \{e_1^L, e_2^L, ...\}$$

$$e^L = (a, x), \quad s.t\ a \in \Sigma_C \textit{ and } x \in \{0, 1\}$$

We remind the reader that Σ_C is all subsets over the alphabet $\Sigma = \{A, B, C, D\}$.

$$\Sigma_C = \{\#, A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD, ABCD\}$$

For example, given the bitmap B_j shown in Figure 2.2, people would recognize that there is an A and D in the bitmap. The corresponding a in the representational language of present letters would be $a = AD$. We could show this as $e^B = (B_j, x)$ converted to $e^L = (AD, x)$.

The transformation from a Teaching Set S^B to a Teaching Set S^L is given by[2]:

$$S^L = L(S^B)$$

In which we transform each example.

$$L(S^B) = \{L(e_1^B), L(e_2^B), L(e_3^B), ...\} = \{e_1^L, e_2^L, e_3^L, ...\} = S^L$$

The transformation from R_B (bitmaps) to R_L (present letters) is given by:

$$L(e^B) = (L(B_k), \theta_{AI}(B_k)) = (\textit{letters present in } B_k, \theta_{AI}(B_k)) = e^L$$

[2]We note that the inverse function L^{-1} can't exist as multiple S^B can map to the same S^L.

To perform the conversion from B_j to "*present letters in* B_j", we store metadata on the bitmaps when creating them. When we make a call to $L(B_j)$, the metadata of B_j is retrieved and we use this metadata to return the "*present letters in* B_j". Without this metadata, we would have to make another system mimicking human's ability to spot present letters, e.g. CNNs trained to detect letters.

3.2 Weaknesses of R_L

The representational language of present letters R_L cannot fully describe bitmap examples e^B. e^B loses some information when converted to the representational language of present letters to e^L. We list some potential features of θ_{AI} we cannot convey using R_L below.

- Orientation of letters: If θ_{AI} is significantly sensitive to the spatial orientation, e.g. upside-down compared to right-side-up, R_L does not have the expressiveness to capture this.
- Relative distance: If θ_{AI} is sensitive to the relative placement of letters, e.g. A and B being close or further apart, R_L does not have the expressiveness to capture this.
- Spacial placement preferences: If θ_{AI} is significantly sensitive to the spatial location, e.g. top-left A compared to bottom-right A, R_L does not have the expressiveness to capture this.

We also note that humans tend to "read" letters. It is, therefore, reasonable to hypothesise that humans could classify a bitmap with "AB" differently than "BA", as seen in Figure 3.2. Even though θ_{AI} might consider these features when making a prediction, we hope these features will at least not dominate. We will later experiment with how well R_L is capable of describing θ_{AI}.

3.3 Search space - Motivation for R_L

Although R_L mimics a human, one could ask whether or not it is necessary to use R_L instead of R_B? Is there another benefit of using R_L except that it mimics a human?

When selecting Teaching Sets, it might be the case that one wants to show upwards of 3-4 negative examples and 3-4 positive examples. If we were to do this in R_B, with a data set of 4000 instances, the number of possible Teaching Sets would be huge.

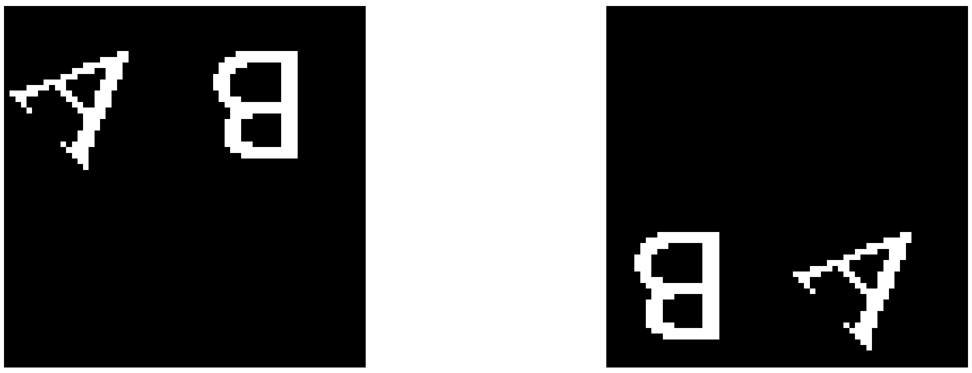


Figure 3.2: Example of similar bitmaps "AB" and "BA"

To calculate how many Teaching Sets there are, we observe that it is fair to assume that approximately all data instances are unique[3]. Combining eight examples into a Teaching Set would give approximately $\binom{4000}{8} = 1.6 * 10^{24}$ possible Teaching Sets. With $1.6 * 10^{24}$ Teaching Sets, it is not possible to evaluate a sizable fraction in reasonable time. If one instead uses R_L, there is only $|\Sigma_C| = 16$ unique examples. 16 unique examples lead to a much lower search space of $\binom{16}{8} = 12870$.

From the comparison, we can see that R_L drastically lowers the search space. However, in the above calculations, we have assumed θ_{AI} to be a R_L**-consistent-AI**. By R_L**-consistent-AI** we mean that θ_{AI} always predicts the same classification for bitmaps containing the same letters. We formalize R_L*-consistent-AI* as:

$$\forall_{(B_j,B_k)} \, [L(B_j) == L(B_k) \implies \theta_{AI}(B_j) == \theta_{AI}(B_k)]$$

If θ_{AI} is not a R_L*-consistent-AI* the number of possible Teaching Sets increases from $\binom{16}{8} = 12870$ to $\binom{32}{8} = 10518300 \approxeq 1.05 * 10^7$. This increase is due to a doubling of the number of examples, as L_M is given examples as labelled by θ_{AI}. For all $a_k \in \Sigma_C$ we could then have both $e_i^L = (a_k, 0)$ and $e_j^L = (a_k, 1)$ as possible examples.

With this we allow Teaching Sets S^L such that $(a_i, 0) = e_j^L \in S^L$ and $(a_j, 1) = e_k^L \in S^L$, s.t. $a_i == a_j$. Having two examples with the same letters and different predicted classification is confusing and not informative. We therefore limit our Teaching Sets to be R_L**-consistent-S**. With an R_L**-consistent-S** Teaching Set we only allow each letter

[3]There will be approximately $4000 * \frac{1}{16}$ empty data instance. However, the point being made still stands.

combination to appear at most once, i.e. $S^L = \{("AB", 1), ("AB", 0)\}$ would not be allowed. We formalise S^L to be R_L**-consistent-S** as:

$$\forall\{[e_j^L = (a, x), e_k^L = (b, y)] \in S^L\}\ [a == b \implies j == k]$$

Using Teaching Sets S^L that are R_L**-consistent-S**, we get $\binom{16}{8} * 2^8 = 3294720 \approxeq 3.3 * 10^6$ potential Teaching Sets. We have the same $\binom{16}{8}$ combinations of different strings $a \in \Sigma_C$, however each of them can now either be predicted as either 0 or 1. This means that for each combination of eight different $a \in \Sigma_C$ we have 2^8 possibilities for their predicted value.

The most realistic situation would be to work with R_L**-consistent-S** Teaching Sets, and a θ_{AI} which is not R_L**-consistent-AI** - a general θ_{AI} is unlikely perfectly aligned to a simple representational language. As the concept we are teaching θ_{AI} is rather simple, we can make θ_{AI} R_L**-consistent-AI** by increasing the training set. A R_L**-consistent-AI** θ_{AI} comes at the cost of longer training sessions for θ_{AI}, and hence a slow down in how often we can perform new experiments. However, as discussed, the search space is drastically lowered if θ_{AI} additionally is R_L**-consistent-AI**. We, therefore, sometimes train θ_{AI} to be R_L**-consistent-AI** when we find this suitable.[4] We will always be working with R_L**-consistent-S** Teaching Sets.

3.3.1 Changing Σ alphabet size

Up to this point, we have been working with an alphabet $\Sigma = \{A, B, C, D\}$. We will now examine how the size of Σ affects the number of possible Teaching Sets.

We will here give an analysis of the expressiveness of the language we have chosen. We will look at the alphabets of $\Sigma_3 = \{A, B, C\}, \Sigma_4 = \{A, B, C, D\}, \Sigma_5 = \{A, B, C, D, E\}$.

In the Table 3.1 we list the number of possible Teaching Sets per cardinality of the Teaching Set. From the table, we can see our language is quite expressive at four letters in the alphabet and that going to five would increase our search space many orders of magnitude. In this thesis, we will only look at the case of three-letter alphabets and four-letter alphabets, as more optimisation work on the code is needed to reach the speeds needed for five-letter alphabets.

[4]We also then lose the ability to measure how well we are able to convey the AIs when it is not aligned with the ground truth.

Teaching Set size	# Teach Sets 5 letter alphabet	# Teach Sets 4 letter alphabet	# Teach Sets 3 letter alphabet
1	64	32	16
2	1984	480	112
5	6.4E6	139776	1792
8	2.7E9	3.3E6	256
11	2.6E11	8.9E6	N/A
16	3.9E13	65536	N/A
21	2.7E14	N/A	N/A
32	4.3E9	N/A	N/A

Table 3.1: An overview of total number of Teaching Sets at some chosen Teaching Set sizes.

The values in the table was generated using the formula in Equation 3.3.

$$g(\Sigma, Ssize) = \frac{2^{Ssize} * (2^{|\Sigma|}!)}{Ssize!(2^{|\Sigma|} - Ssize)!} \tag{3.3}$$

And we would get the expressiveness of an alphabet by Equation 3.4.

$$size(\Sigma) = \sum_{i=1}^{|\Sigma|^2} g(\Sigma, i) \tag{3.4}$$

The function in Equation 3.3 is at the core the n choose k formula with no repetition. $Ssize(k)$ is the number of examples for a given Teaching Set size. n is the number of subsets of the alphabet, totaling $n = 2^{|\Sigma|} = |\Sigma_C|$. There is no information to be gained by having duplication of examples, and as all they do is increase the cognitive load of L_H, we avoid them. In the formula, we see the avoiding of duplicates as the "no repetition" part of the n choose k formula. The last part missing to be explained in Equation 3.3 is 2^{Ssize} in the numerator. This accounts for the fact that in each Teaching Set, all examples can either have the value True or False.

This term 2^{Ssize} can be avoided if we require θ_{AI} to be a R_L**-consistent-AI**. We would then get the search spaces shown in Equation 3.5 and examples shown in Table 3.2.

$$size_{R_L}(\Sigma) = \sum_{i=1}^{|\Sigma|^2} \frac{g(\Sigma, i)}{2^{Ssize}} \tag{3.5}$$

These tables show how the number of possible Teaching Sets rapidly increases with the Teaching Set's cardinality. If one limited the cardinality of the potential Teaching Sets, one would be able to increase the size of the alphabet without increasing the search space too much.

Teaching Set size	# Teach Sets 5 letter alphabet	# Teach Sets 4 letter alphabet
1	32	16
2	496	120
5	201376	4368
8	1.1E7	12870
11	1.3E8	4368
16	6.0E8	1
21	1.3E8	N/A
32	1	N/A

Table 3.2: Overview of total number of Teaching Sets. Given R_L**-consistent-AI** AI.

We end with the observation that when θ_{AI} is R_L**-consistent-AI**, the number of Teaching Sets equals the number of subsets of Σ_C. As the size of Σ_C is given by $2^{|\Sigma|}$, and the number of subsets of Σ_C is given by $2^{|\Sigma_C|}$, we get that the total number of Teaching Sets is $2^{2^{|\Sigma|}}$. We sanity check the observations:

- $size_{RL}(\{A, B, C, D, E\}) = 4294967295 = 2^{2^5} - 1$.
- $size_{RL}(\{A, B, C, D\}) = 65535 = 2^{2^4} - 1$.
- $size_{RL}(\{A, B, C\}) = 235 = 2^{2^3} - 1$

The -1 corresponds to the empty subset, which is not a valid Teaching Set[5].

3.3.2 Conveying the Teaching Set to L_H

In the end, we would want to show Teaching Sets containing bitmaps over R_B to humans. However, as our system will work in R_L and find an optimal S^L, converting an optimal S^L to S^B is not trivial. The reason for this is that for each $e_j^L = (a_k, x)$ there are multiple valid B_j s.t. $L(B_j) = a_k$ and $\theta_{AI}(B_j) = x$. One has to select which one, out of all these B_j, should be shown to the human L_H.

Solving this problem is outside the scope of this thesis, and here we restrict to output Teaching Sets in R_L.

[5]A Teaching Set must be of cardinality one or greater

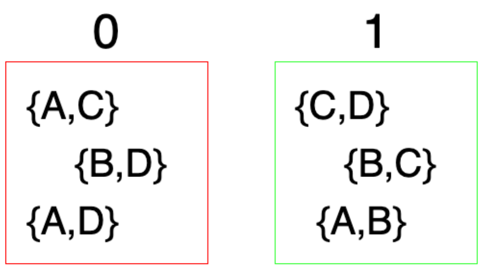


Figure 3.3: A visual representation of Teaching Set S_r^L in R_L

3.4 Boolean expressions – modelling human reasoning

We remind the reader that our goal in this chapter is to make a model L_M of a human L_H, as described in Equation 3.2. After establishing the language R_L, we now turn our attention to making a model of how a human would reason when tasked with finding a relation between bitmaps containing letters labelled 0 or 1. As we already discussed, we argue it is a reasonable assumption that most humans would mostly only pay attention to the present letters in the bitmaps. We, therefore simplify the problem to find a relation between groups of letters and either 0 or 1[6]. For example, we could task a human L_H with finding a classification rule given S_r^L shown in Figure 3.3.

The Teaching Set visualised in Figure 3.3 can be written as:

$$S_r^L = \{(AC, 0), (AD, 0), (BD, 0), (AB, 1), (BC, 1), (CD, 1)\}$$

If the Teaching Set S_r^L was shown to different humans L_H they would probably give different reasons / models θ_H to explain the classifications. For instance, some may notice that in the given S_r^L the examples labelled 1 contains consecutive letters in the alphabet, and examples labelled 0 contains letters not consecutive in the alphabet.

It is fair to assume some humans could pick up on this attribute of consecutive letters. The same people might use this as their model of the classification. This guessed model based on consecutive letter we call c^{Σ}. c^{Σ} would be part of a concept class of relations due to orders of letters in the alphabet C^{Σ}. The use of concept classes is crucial in Machine Teaching, as described in the introduction.

[6]But let us note that there could be many such relations that humans find meaningful. In the following we illustrate this.

A	B	C	D	ϕ
0	0	0	0	x
0	0	0	1	x
0	0	1	0	x
0	0	1	1	1
0	1	0	0	x
0	1	0	1	0
0	1	1	0	1
0	1	1	1	x
1	0	0	0	x
1	0	0	1	0
1	0	1	0	0
1	0	1	1	x
1	1	0	0	1
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

Table 3.3: Truth table for S^L_r. x being arbitrary value.

Given a different AI θ_{AI} and Teaching Set S^L, it is unlikely that a clear relation between examples based on the order of the alphabet exists. Therefore, one could argue it to be a poor strategy to be looking for a relation based on the order of the alphabet C^Σ. Given another Teaching Set, there might not be a relation based on the order of the alphabet at all - no *valid* $c^\Sigma \in C^\Sigma$ for the Teaching Set. We say a concept c is *valid* for S^L when $c(a) = x$ holds for all examples $(a, x) \in S^L$.

What we want is for the human to use the class C^b of Boolean functions. These Boolean functions are of the same format as ϕ, our ground truth label function. Given a Teaching Set S^L, L_M will find a valid Boolean function $\phi = c^b \in C^b$ such that the Boolean function c^b is valid for S^L. But there are several valid ϕ, which one to pick?

Given Teaching Set S^L_r, one could for instance select the Boolean function $\phi = (A \wedge B) \vee (C \wedge \neg A)$ to be the guessed concept c^b. This is a valid guessed concept as for all examples $(a, x) = e^L \in S^L_r$, $c^b(a) = x$. Contrary to the concept class of alphabet-order C^Σ, there is always a Boolean function describing S^L, as S^L is *R_L-consistent-S*. We can find such a Boolean function c^b by constructing a truth table based on S^L.

We construct such a truth table by having each row defined by a matching example[7] in S^L. The rows with no matching example arbitrarily have the values 0 or 1 (marked x). We show such a truth table in Table 3.3 for S^L_r.

For each row in Table 3.3 with a non arbitrary value, that is $\phi \neq x$, we defined the row by its matching example. For instance, given $e = (AD, x)$ we look at the row with

[7]Because S^L is R_L-consistent-S, there will at most be one example matching each row in the truth table.

	#	A	B	C	D	AB	AC	AD	BC	BD	CD	ABC	ABD	ACD	BCD	ABCD
c^{Σ}	1	1	1	1	1	1	0	0	1	0	1	1	0	0	1	1
c^b	0	0	0	1	0	1	0	0	1	0	1	1	0	0	1	1

Table 3.4: Table showing similarity of concepts: consecutive letters c^{Σ} and Boolean expression $c^b = (A \wedge B) \vee (C \wedge \neg A)$

corresponding variables set to True, $[A = 1, B = 0, C = 0, D = 1]$, and set ϕ of that row equal to x. We do this for all rows with a matching example.

As we can define a truth table, we have by extension shown that there always exists a Boolean function $\phi = c^b \in C^b$ *valid* for any Teaching Set S^L. However, it is still the cases that a human L_H might find a concept c^{Σ} based on consecutiveness in the alphabet. One might suspect it to be a problem if the human L_H aims to find a concept based on the order of the alphabet c^{Σ} when we provide a Teaching Set aimed to explain a Boolean function c^b. However, this is not necessarily the case. Even though c^b and c^{Σ} are not in the same concept class, they might evaluate most examples equally. Let us look at how the two models compare given all potential inputs $a \in \Sigma_C$.

From Table 3.4 we see that there are only four examples: $\{\#, A, B, D\}$, where c^b and c^{Σ} differs. The fact that c^b and c^{Σ} differs in only a few examples gives us hope that even if the human L_H is not even using our concept class of Boolean functions, their model c^{Σ} can still be quite well aligned with with the concept c^b we aim to teach. For the same reason, we hope that even if we find a Teaching Set S^L to teach some Boolean function c^b, and the human L_H finds some concept c from another concept class C, there is still reason to hope c and c^b is aligned.

We conclude the discussion above by stating that the task of L_M will be to find a Boolean function ϕ matching the input S^L, and hence $L_M(S) = \theta_M$ will be a Boolean function.

3.4.1 Implementing $\operatorname{argmin}_{\theta_M : \theta_M \models S}$ – Karnaugh map

Finding a partial Boolean function given a Teaching Set is rather straightforwards, and a process for this has been discussed in the beginning of this chapter. We will now look at how to select the simplest completion of this partial Boolean function, such that $L_M(S) = \theta_M$ s.t. $\beta(\theta_M)$ is minimum.

CD\AB	00	01	11	10
00	x	x	1	x
01	x	0	x	0
11	1	x	x	x
10	x	1	x	0

Figure 3.4: K-Map for S_r^L

Karnaugh Maps, by Karnaugh et al. [11], is a good match for us to use in this situation. Given a truth table (potentially incomplete), the Karnaugh Map finds a simplified / "minimal" Boolean expression matching the truth table. Importantly we will use Karnaugh Maps to find a Boolean expression in disjunctive normal form (DNF) with the minimum number of clauses. Given a Teaching Set S L_M will use a Karnaugh Map to find a minimal Boolean Expression.

Next, we discuss how Karnaugh Maps work and show examples of when multiple minimum Boolean expressions can arise.

Karnaugh map

To use Karnaugh Maps (K-Map), one needs to have a Boolean function or truth table defined. Given the Teaching Set we have been looking at previously $S_r^L = \{(AC, 0), (AD, 0), (BD, 0), (AB, 1), (BC, 1), (CD, 1)\}$, we can construct a truth table as given in Table 3.3 with some undecided values "x"[8].

To find a simplified Boolean expression from this, we construct the Karnaugh Map displayed in Figure 3.4. The Karnaugh Map has two axis. The columns denote the values of A and B, e.g. "01" indicates A =False and B = True. The rows correspond to the different assignments of variables C and D, e.g. "10" indicates C = True and D = False. One fills in the cells with "0", "1", or "x" depending on if one knows the cell to be False, True or does not care. To find a minimal Boolean Expression in a Karnaugh Map, we draw rectangles of size 2^n aiming to capture all "1"s in the rectangles without enclosing any "0".

These rectangles follow some rules. First, to find the minimal Boolean expression, one must use as few rectangles as possible. A minimum number of rectangles ensures

[8]When working with Karnaugh Maps these are called "Don't care" terms.

CD\AB	00	01	11	10
00	x	x	1	x
01	x	0	x	0
11	1	x	x	x
10	x	1	x	0

Figure 3.5: Example of not maximal rectangles.

CD\AB	00	01	11	10
00	x	x	1	x
01	x	0	x	0
11	1	x	x	x
10	x	1	x	0

Figure 3.6: K-Map rectangle giving the clause $(\neg A \land C)$.

the minimum number of clauses in the Boolean expression. Secondly, the rectangles have to be of maximum size. If a rectangle is of size 2^a, and you can make it 2^{a+1}, it is required to make it 2^{a+1}. Maximum rectangles ensure that each clause contains the fewest number of variables. If we don't follow these rules we get complex Boolean expressions like $\phi = (\neg A \land \neg B \land C \land D) \lor (\neg A \land B \land C \land \neg D) \lor (A \land B \land \neg C \land \neg D)$ as seen in Figure 3.5. Given S_r^L it would be unreasonable to decide that such a complicated function was the concept someone wanted to teach.

After constructing the K-Map, we can look at each rectangle, locate the variables with the same value for all cells inside the rectangle, and use them to construct a clause in the Boolean expression. We show this in Figure 3.6. To further understand how L_M works with K-Maps we present three of the Boolean Expression L_M could produce given the K-Map displayed in Figure 3.4. The three selected solutions are a subset of all the potential solutions giving different Boolean expressions with the same number of clauses.

The first is $\phi = (\neg A \land C) \lor (\neg C \land \neg D)$. We get this from the Figure 3.7. Here the top rectangle gives $(\neg C \land \neg D)$, and the bottom left square gives $(C \land \neg A)$. Secondly we show an example where rectangles bend around the edges in Figure 3.8. This solution would give $(B \land \neg D) \lor (C \land \neg A)$. From the third option we get $\phi = (\neg A \land C) \lor (A \land B)$. We show the rectangles giving this solution in Figure 3.9. This last possibility is the suggested Boolean function we used earlier in this chapter, e.g. in the Table 3.4.

CD\AB	00	01	11	10
00	x	x	1	x
01	x	0	x	0
11	1	x	x	x
10	x	1	x	0

Figure 3.7: K-Map giving $\phi = (\neg A \wedge C) \vee (\neg C \wedge \neg D)$

CD\AB	00	01	11	10
00	x	x	1	x
01	x	0	x	0
11	1	x	x	x
10	x	1	x	0

Figure 3.8: K-Map giving $\phi = (C \wedge D) \vee (B \wedge \neg D)$

CD\AB	00	01	11	10
00	x	x	1	x
01	x	0	x	0
11	1	x	x	x
10	x	1	x	0

Figure 3.9: K-Map giving $\phi = (A \wedge B) \vee (C \wedge \neg A)$

3.4.2 β - Simplicity of Boolean Expressions

With the Karnaugh Map we are capable of finding a complete Boolean function θ_M, however, Karnaugh Maps can return one of potentially many minimal Boolean Expressions with equally many clauses. Therefore, we are using a modified Karnaugh Map returning all Boolean expressions with equally many clauses. Given this list of Boolean expressions, we apply the β function to select the minimum. By doing this, we ensure L_M returns the same Boolean expression for each input S.

We therefore have Karnaugh Map performing the argmin in Equation 3.6 and we now define β to select the minimum:

$$L_M(S) = \underset{\theta_M : \theta_M \models S}{\operatorname{argmin}} \{\beta(\theta_M)\} \tag{3.6}$$

θ_M will be one of the potential Boolean expressions found by the Karnaugh Map. For consistency, we note our K-Map will only look at Boolean expressions in disjunctive normal form (DNF). DNF format is OR of ANDs, e.g. $\phi = (A \wedge B) \vee (C \wedge D)$. We define a Boolean expression $BE = Cl_1 \vee Cl_2 \vee ... \vee Cl_m$, to contain multiple AND-clauses $Cl_k \in BE$. A benefit of using DNF format is that it mimics human reasoning. To verify a positive instance one only need to confirm one positive clause, whereas one need to check all clauses to confirm a negative instance. The resource-heavy task of confirming a negative compared to a positive is somewhat similar to how humans are poor at negations [9]. This further strengthens L_Ms ability to mimic L_H.

For instance, given two Boolean expressions, we argue that BE_1 is simpler than BE_2:

$$BE_1 = A$$

$$BE_2 = (B \wedge C \wedge \neg D) \vee (\neg A \wedge C)$$

Although it might be simple to say that BE_1 is simpler than BE_2, we need to find some general attributes of Boolean expressions to be able to compare Boolean expression using β. We list them here:

- The number of clauses: Fewer is better.

- The number of negations: Fewer is better.
- Simplicity of minimum clause.

We then select these attributes of each clause:

- The number of letters in a clause: Fewer is better.
- The number of negations in a clause: Fewer is better.

We formalise these attributes below. When evaluating which Boolean expression is simpler we use the first rule that applies:

$$\begin{array}{lll}
1) & \beta(BE_i) < \beta(BE_j), & \textit{if \# clauses in } BE_i < \textit{ \# clauses in } BE_j. \\
2) & \beta(BE_i) < \beta(BE_j), & \textit{if \# negations in } BE_i < \textit{ \# negations in } BE_j. \\
3) & \beta(BE_i) < \beta(BE_j), & if \beta(Cl_1^i) < \beta(Cl_1^j) \\
4) & \beta(BE_i) < \beta(BE_j), & if \beta(Cl_2^i) < \beta(Cl_2^j) \\
5) & \textit{Continue for all clauses...} &
\end{array} \tag{3.7}$$

In rule 3) and onward we want to find which expression has the simplest clause. If the two simplest clauses are equal, we move on to the next clause and check them, and so on. We define that all clauses in a Boolean expressions BE are sorted by β, i.e. $\beta(Cl_i) \leq \beta(Cl_j) \iff i \leq j$. To achieve this ordering and comparison we define β for individual clauses Cl_i as follows:

$$\begin{array}{lll}
1) & \beta(Cl_i) < \beta(Cl_j), & \textit{if \# letters in } Cl_i < \textit{ \# letters in } Cl_j. \\
2) & \beta(Cl_i) < \beta(Cl_j), & \textit{if \# negations in } Cl_i < \textit{ \# negations in } Cl_j. \\
3) & \beta(Cl_i) < \beta(Cl_j), & \textit{if letters in } Cl_i \textit{ is } \textbf{lexicographically} \textit{ before letters in } Cl_j.
\end{array}$$

We want to achieve a total ordering of the Boolean expressions such that $L_M(S)$ returns the same Boolean expression for the same S each time. We therefore include the last equation step 3), discriminating letters based on lexicographical order. In this lexicographical order, negated letters are sorted after non negated letters.

Having defined β we can look back to all the different Boolean Expression our K-Map potentially could find given S_r^L. Among $\phi_1 = (\neg A \wedge C) \vee (\neg C \wedge \neg D)$, $\phi_2 = (B \wedge \neg D) \vee (C \wedge \neg A)$ and $\phi_3 = (\neg A \wedge C) \vee (A \wedge B)$ we can apply β to find the minimum to be returned by L_M. For the Teaching Set S_r^L it turns out $\phi_3 = (\neg A \wedge C) \vee (A \wedge B)$ is the minimum Boolean expression valid using our defined β.

3.5 Occam's Razor

We argue that our β function and our search for a minimum / simplistic Boolean expression is a reasonable estimation of how a human would reason based on **Occam's Razor** [4]. Occam's Razor states that when faced with two possible explanation for a phenomena, the simpler one is more likely to be correct. If you saw a tree fallen over on the road, you would assume it had fallen over by old age and wind. You would not assume someone made a fake tree, and moved it out to the road to be displayed as art. Both explanations are valid as they describe the observation, however one is vastly simpler then the other.

In our case this translates to the fact that if shown the Teaching Set $S_r^L = \{(AC, 0), (AD, 0), (BD, 0), (AB, 1), (BC, 1), (CD, 1)\}$ it is more probable that the Boolean expression used to generate S_r^L was $\phi = (\neg A \land C) \lor (A \land B)$, compared to $\phi = (\neg A \land \neg B \land C \land D) \lor (\neg A \land B \land C \land \neg D) \lor (A \land B \land \neg C \land \neg D)$. Both Boolean expressions are *valid*, however the first one is simpler and we therefore argue it is more likely by Occam's Razor to be the concept taught.

3.6 Output θ_M

The output of L_M is θ_M, which will be a Boolean function. More precisely, it is one of the minimal Boolean expressions found by our Karnaugh Map. We apply the rules in β to find the unique minimum among the minimal.

3.7 A seemingly anomalous result

This section discusses what happens if we give L_M a Teaching Set of maximal cardinality, e.g. $|S^L| = |\Sigma_C|$. Such a Teaching Set needs to contain one example for all $a \in \Sigma_C$ and associated prediction.

As S^L has one predicted value for each possible input, L_M can construct a complete truth table. With a complete truth table there are no arbitrary values, and hence we do not need to find a minimal Boolean expression to select the Boolean function θ_M. Therefore, a S^L of max cardinality corresponds to a single Boolean function θ_M.

CD/AB	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	1
10	0	0	1	0

Figure 3.10: K-Map for counter example

CD/AB	00	01	11	10
00	x	x	x	x
01	x	0	x	0
11	x	x	1	x
10	x	0	x	0

Figure 3.11: K-Map for counter example first Teaching Set

For simplicity sake we will assume θ_{AI} to be $R_L -$ *consistent-AI*. If not the Boolean function found might not be the best possible match for θ_{AI}. However, given $R_L -$ *consistent-AI*, a Teaching Set of size $|\Sigma_C|$ will perfectly describe θ_{AI}.

From this, one could assume that adding examples to a Teaching Set will continuously improve the model θ_M produced by L_M. Although we stipulate that, on average, Teaching Sets with more examples will provide better matching of θ_M and θ_{AI}, it is not always the case. A contradicting example will follow.

Let us say we have a complete Boolean function ϕ_C giving the K-Map shown in Table 3.10. From ϕ_C we can construct the Teaching Set S_c^L:

$$S_c^L = \{(ABCD, 1), (AD, 0), (BD, 0), (BC, 0)\}$$

Let us say we give this Teaching Set S_c^L to L_M. From this S_c^L, L_M constructs the K-map shown in Table 3.11. Our L_M would, given this, find the Boolean expression $\phi = A \wedge B$. This ϕ correctly matches ϕ_C for all values except one. One could argue S_c^L to be a great Teaching Set, almost perfectly teaching the concept θAI.

As S_c^L only miss evaluated the cell ACD, it might be possible to extend the Teaching Set with $e^L = (ACD, 1)$ to get a better Teaching Set:

$$S_c^L = \{(ABCD, 1), (ACD, 1), (AD, 0), (BD, 0), (BC, 0)\}$$

CD/AB	00	01	11	10
00	x	x	x	x
01	x	0	x	0
11	x	x	1	1
10	x	0	x	0

Figure 3.12: K-Map for the expended Teaching Set.

From this new Teaching Set we generate a new K-Map, shown in Figure 3.12. In this Teaching Set, L_M selects $\phi = C \wedge D$ as its guessed Boolean function. This Boolean function predicts correctly the value of $\frac{11}{16}$ cases. This is clearly now worse compared to the first Teaching Set. We have shown that a Teaching Set of size four can give better concept alignment than a Teaching Set of size five in our system, which seems anomalous at first sight.

Chapter 4

Comparing θ_{AI} and θ_M - λ

$$T(\theta_{AI}) = \underset{S:\theta_{AI}\models S}{\operatorname{argmin}}\{\delta(S) + \lambda(\theta_{AI}, \theta_M) : L_M(S) = \theta_M\} \tag{4.1}$$

$$L_M(S) = \underset{\theta_M:\theta_M\models S}{\operatorname{argmin}}\{\beta(\theta_M)\} \tag{4.2}$$

We have now defined θ_M and L_M as described in Equations 4.1 and 4.2. Next, we describe the usage of θ_M in our system. θ_M is one of two arguments given to λ. With λ, we aim to compare how equal θ_{AI} and θ_M is. To do this λ will punish a Teaching Set S if $L_M(S) = \theta_M$ is dissimilar to θ_{AI}. One could call this the $TeachingRisk$ [26], as discussed on the topic of Machine Teaching.

A diagram of the implementation of λ is given in Figure 4.1.

4.1 Comparing θ_{AI} and θ_M

When we want to compare θ_{AI} and θ_M, we immediately run into the problem that θ_{AI} and θ_M do not have matching input domains. We have one model, θ_{AI}, taking in bitmaps, i.e. in the representational language of bitmaps R_B. The other model θ_M takes as input present letters, i.e. in the representational language of present letters R_L. Because the conversion from R_B to R_L is a loss-full transformation, we will only compare some aspects of θ_{AI} with θ_M[1]. To convert θ_{AI} from R_B to R_L we make a representation of θ_{AI} in R_L.

[1]As stated earlier; we conjecture that the aspect of present letters is dominant in θ_{AI}.

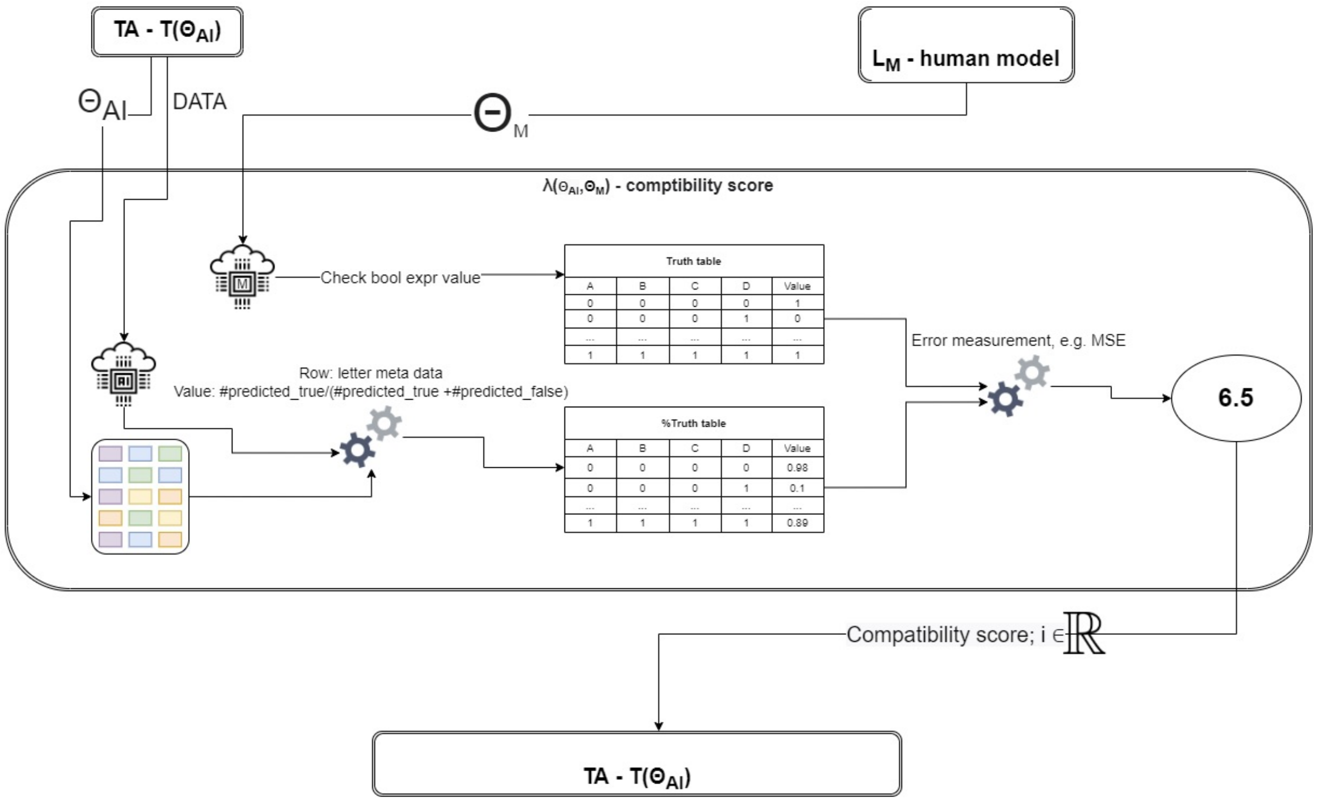

TA - T(ΘAI)
LM - human model
ΘAI
DATA
ΘM
Λ(ΘAI,ΘM) - comptibility score
Check bool expr value
Truth table
Row: letter meta data
Value: #predicted_true/(#predicted_true +#predicted_false)
%Truth table
Error measurement, e.g. MSE
6.5
Compatibility score; i ∈ ℝ
TA - T(ΘAI)

Figure 4.1: Diagram of δ.

To make this representation of θ_{AI} in R_L, we look at how θ_{AI} evaluates bitmaps grouped by present letters. For each combination of present letters, $a \in \Sigma_C$, we find all bitmaps $B_j \quad s.t. \quad L(B_j) = a$ in the data set and calculate the percentage of bitmaps θ_{AI} predicts True[2]. Doing this for all $a \in \Sigma_C$, we get a table like the one shown in Table 4.2. From the table, we observe that θ_{AI} unfailingly predicts some letter groups the same and is more undecided on other letter combinations. We conclude that R_L expresses θ_{AI} well for some letter groups and less for others.

To evaluate how well θ_M matches θ_{AI}, we create a similar table for θ_M. Since θ_M is a Boolean function on present letters, we construct the corresponding truth table from this function. Given the Boolean expression $\phi = (A \wedge B) \vee (C \wedge \neg A)$ we get the truth table shown in Table 4.3.

We can now compare the two tables using Mean Square Error (MSE) as our error measure. We look at the difference between θ_{AI} and θ_M for each row and aggregate the square difference. The MSE is visualized in Table 4.4. When adding it all up, we see that for this θ_{AI} and θ_M, we get an error score of 0.2222. To get MSE, we divide by the 16

[2]For each individual row we need 385 data instances to ensure a confidence level of 95% that the displayed value is within $\pm 5\%$ of the real distribution for that row.

Symbol	θ_{AI}'s prediction
#	0,00
A	0,00
B	0,00
C	0,95
D	0,00
AB	0,99
AC	0,02
AD	0,00
BC	0,63
BD	0,02
CD	0,91
ABC	1,00
ABD	1,00
ACD	0,04
BCD	0,74
ABCD	1,00

Figure 4.2: θ_{AI} break down in percentages.

Symbol	ϕ's evaluation
#	0
A	0
B	0
C	1
D	0
AB	1
AC	0
AD	0
BC	1
BD	0
CD	1
ABC	1
ABD	1
ACD	0
BCD	1
ABCD	1

Figure 4.3: Truth table for Boolean expression $\phi = (A \wedge B) \vee (C \wedge \neg A)$.

groups giving us a score of $\frac{0.2222}{16} \approx 0.0139$. The MSE score is the value λ returns. We note that this lambda score might seem small. However, any scaling of this value will not affect our system. Each lambda score should only be compared to another lambda score. The comparison and alignment with δ are controlled by μ in our score function.

4.1.1 λ score R_L-*consistent-AI*

We have previously introduced the concept of R_L-*consistent-AI*. If θ_{AI} is R_L-*consistent-AI* it has the interesting property that there will always exist some θ_M, for which $\lambda(\theta_{AI}, \theta_M) = 0$.

To refresh the definition of R_L-*consistent-AI*, we show it here again. A R_L-*consistent-AI* θ_{AI} is one that, for two bitmaps with the same letters in it, produces the same prediction for both.

$$\forall_{(B_j,B_k)} \, [L(B_j) == L(B_k) \implies \theta_{AI}(B_j) == \theta_{AI}(B_k)]$$

To show why R_L-*consistent-AI* leads to a λ-score of zero for some θ_M, we create a breakdown of R_L-*consistent-AI* θ_{AI}, as shown in Table 4.5.

Letters	θ_{AI}	ϕ	SE
#	0,00	0	0
A	0,00	0	0
B	0,00	0	0
C	0,95	1	0,0025
D	0,00	0	0
AB	0,99	1	0,0001
AC	0,02	0	0,0004
AD	0,00	0	0
BC	0,63	1	0,1369
BD	0,02	0	0,0004
CD	0,91	1	0,0081
ABC	1,00	1	0
ABD	1,00	1	0
ACD	0,03	0	0,0009
BCD	0,73	1	0,0729
ABCD	1,00	1	0
Mean Squared Error			0,0139

Figure 4.4: Mean Squared Error table, Col.2: θ_{AI} break down - Col.3: ϕ truth table - Col.4: Squared Error between θ_{AI} and ϕ.

Symbol	θ_{AI}'s prediction
#	0,00
A	0,00
B	0,00
C	1,00
D	0,00
AB	1,00
AC	0,00
AD	0,00
BC	1,00
BD	0,00
CD	1,00
ABC	1,00
ABD	1,00
ACD	0,00
BCD	1,00
ABCD	1,00

Figure 4.5: Breakdown of R_L-*consistent-AI* θ_{AI} in R_L

If θ_{AI} is R_L-*consistent-AI* all bitmaps with the same letters present will be evaluated equally by θ_{AI}. We will see this in the breakdown of θ_{AI} as only 0% or 100% in the rows of Table 4.5. Any other value than 0% or 100% would break the definition that all bitmaps containing the same letters are evaluated equally. As the break down of θ_{AI} is of the same format as a truth table, a Boolean function ϕ could match this breakdown of θ_{AI} and give a λ-score of zero. One could incorporate this ϕ in a Teaching Set with 16 examples, using one example to describe each row.

Given this Teaching Set, L_M will create a truth table corresponding to the Teaching Set in the same fashion discussed earlier. This new truth table will be identical to θ_{AI} breakdown. As none of the rows in the new truth table contains arbitrary values 'x', L_M will find a Boolean function matching the same truth table. As this Boolean function matches the truth table exactly, and the truth table is equal to the breakdown of θ_{AI}, we have a Boolean function perfectly matching the breakdown of θ_{AI}.

For the θ_{AI} given in Table 4.5 we can for instance create the Teaching Set shown in Equation 4.3, ensuring $L_M(S) = \theta_M$ s.t. $\lambda(\theta_{AI}, \theta_M) = 0$:

$$\begin{aligned} S^L = \{&(\#, 0), (A, 0), (B, 0), (C, 1), (D, 0), (AB, 1), (AC, 0), (AD, 0), (BC, 1), \\ &(BD, 0), (CD, 1), (ABC, 1), (ABD, 1), (ACD, 0), (BCD, 1), (ABCD, 1)\} \end{aligned} \tag{4.3}$$

We have shown that if θ_{AI} is R_L-*consistent-AI* there always exists a θ_M perfectly describing it, i.e. with a λ-score of zero. Furthermore, we can construct a Teaching Set of size Σ_C such that $L_M(S)$ is equal to this θ_M.

4.2 Speed up implementations

We wanted to run our system on as large data sets and alphabets as possible. To ensure we could run our system on large data sets, we spent time finding bottlenecks in our system and improving them. One of the significant bottlenecks at the start was due to us recalculating the representation table of θ_{AI} in R_L for each $\lambda(\theta_{AI}, \theta_M)$ call. This table can be precomputed once at the start and then reused throughout.

Chapter 5

Complexity of Teaching Set - $\delta(S)$

In the previous two chapters, we discussed how well a Teaching Set S taught θ_{AI} to L_M. In this chapter, we will give a measure $\delta(S)$ of how difficult L_H finds it to understand S.

When showing a Teaching Set S to L_H, we will be displaying S in the representation language of present letters and not bitmaps. Hence we will measure the complexity $\delta(S)$ in R_L, and therefore δ will be given Teaching Set S^L containing present letters and not bitmaps. In this chapter, we will discuss which Teaching Sets humans find difficult to understand, show four suggested δ-function implementations. Then in Chapter 7 we will compare these suggested implementations.

As in previous chapters, we repeat the main formula, in Equation 5.1, of this thesis such that the reader more easily can see the role of δ. We also give a diagram for δ in Figure 5.1.

$$T(\theta_{AI}) = \underset{S:\theta_{AI}\models S}{\operatorname{argmin}}\{\delta(S) + \lambda(\theta_{AI}, \theta_M) : L_M(S) = \theta_M\} \tag{5.1}$$

$$L_M(S) = \underset{\theta_M:\theta_M\models S}{\operatorname{argmin}}\{\beta(\theta_M)\}$$

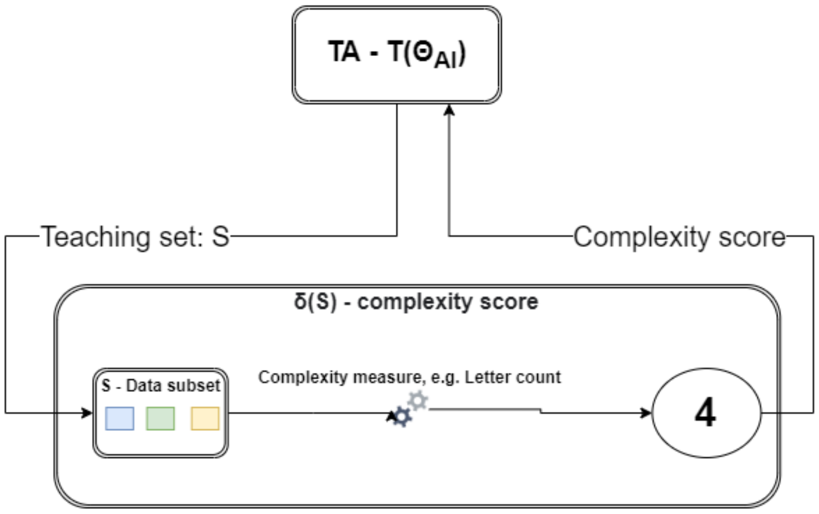


Figure 5.1: Overview of δ

5.1 Why penalise complex Teaching Sets?

One question one could ask is why even penalise Teaching Sets based on their complexity? Should we not use the Teaching Set S such that $L_M(S) = \theta_M$ is as close as possible to θ_{AI}?

To answer this, we argue there is one simplification made in L_M not discussed thus far. Machines are excellent at storing and remembering information. The larger a Teaching Set we give to L_M, the closer we expect θ_M to match θ_{AI}[1]. For $\Sigma = \{A, B, C, D\}$ a Teaching Set of size 16 would fully describe any θ_{AI}, assuming θ_{AI} to be R_L-*consistent-AI*. For any new example shown to θ_{AI} one could predict the output of θ_{AI} by finding the example in S^L containing the same letters as the new example, as θ_{AI} would predict the same as this one.

Even if θ_{AI} is not R_L-*consistent-AI* a Teaching Set of size 16 will give θ_M as closely matching θ_{AI} as possible. That is if the classification x in each example $e^L = (a, x)$ are θ_{AI}s most common classification of the letter combination a. In either case, L_H will then not be able to perfectly predict the output of θ_{AI}. However, L_H should be able to predict θ_{AI}'s output on the majority[2] of new examples.

[1] As shown, there seems to be some anomalous Teaching Set regarding this.
[2] Assuming the new examples are drawn from a similar distribution as the input data set.

Name	Description
Cardinality	{ABC,ABD} == {#,A} Baseline. Cardinality of Teaching Set.
Chunking	{A,B,C,D} > {AB,CD} Mimic human cognitive chunking.
SquaredSum	{ABC,A}>{AB,AC} Squared sum of example sizes.
SumOfExamples	{ABC, A} == {AB,AC} Sum of example sizes.

Table 5.1: Overview of considered δ functions

A computer would have no problem working with 16 or even hundreds of examples in a Teaching Set. However, a human might struggle to find the pattern among the examples for even a small Teaching Set.

In psychology, the concept of **cognitive load** [19] measures how hard a problem is for a given human. A Teaching Set containing many examples could be an example of a high cognitive load. This high cognitive load makes the human incapable of using their cognitive resources to find logical patterns. Instead, most of their mental capacity is spent on storing the images in **working memory** [19], e.g. actively remembering them. A high cognitive load is associated with poor learning of new concepts, which would mean that even if a large Teaching Set rationally contains the information to teach θ_{AI}, a human might misunderstand the information. We model this human behaviour/limited capability with $\delta(S)$.

It has often been the norm in Machine Teaching to use **cardinality** of the Teaching Set, i.e. number of examples, as the measure of complexity [26]. In this thesis, we will briefly look at some other techniques to measure complexity. We present these new ideas to better align our δ with the Teaching Sets humans find complicated.

5.2 Different δ implementations

We here present four different δ implementations. An overview of the different delta functions is given in the Table 5.1.

Cardinality

```python
def get_complexity_of_subset(self, teaching_set):
        """Compute the complexity of the examples given"""
        return len(teaching_set)
```

Cardinality measure is the standard δ used in Machine Teaching [26]. The goal of using cardinality is to find the smallest-sized Teaching Set. We have implemented *Cardinality* as a baseline and will use it primarily to compare it with other delta functions.

Chunking

```python
def get_complexity_of_subset(self, teaching_set):
        """Compute the complexity of the examples given"""
        complexity = 0
        for example in teaching_set:
            if len(example) == 0:
                complexity += 0.1
            elif 1 <= len(example) <= 3:
                complexity += len(example)*0.2 + 1
            else:
                complexity += len(example)*1.3 + 2
        return complexity
```

With *Chunking*, we intend to mimic aspects of cognitive load [20] from human psychology theory. Specifically, we want to mimic cognitive chunking [22]. Cognitive chunking is why humans find it simpler to remember grouped elements instead of singletons. An example discussed by Thalmann in "How does chunking help working memory?" [22] is something to the essence of how a human might find it hard to remember F-B-I-D-Q-B and more straightforward to remember FBI-DQB. However, this grouping technique has limitations, as humans do not find it easier to group everything, e.g. FBIDQB. In a sense, one would then end up with just singletons once again.

Chunking aims to mimic these aspects of the human mind by not heavily punishing an example containing three or fewer letters. The number three is chosen as humans have been shown to handle groups of 4 and less well before [5]. We decided to have a cut-off at three letters present to ensure that not all Teaching Sets were evaluated equally.

With this definition, we hypothesise that *Chunking* will tend to prioritise small cardinality in S while at the same time punishing having too many letters present in one example – i.e. 4 in our case.

We note that the characteristics of *chunking* would be more visible given a larger alphabet and a larger cut-of. We would then expect a distinct difference between chunking and *SquaredSum*. As mentioned earlier, the current system cannot work with $|\Sigma| > 4$, so we cannot experiment with this.

SquaredSum

```python
def get_complexity_of_subset(self, teaching_set):
        """Compute the complexity of the examples given"""
        score = sum([(len(example))**2 for example in teaching_set])
        score += sum([0.1 for example in teaching_set if len(example)
            ↪ == 0])
        return score
```

The *SquaredSum* aims to keep the total information – letters present – low while simultaneously putting a high cost on S with individually complex examples. Even if the total information is low, it might not help if one example is too big, and therefore L_H cannot comprehend it.

Later in this chapter we will put forward results where we discovered that we did not punish empty examples. Adding a line punishing empty examples was therefore added. This line ensures we do not evaluate $S^L = \{(A,1),(B,0)\}$ and $S^L = \{(\#,0),(A,1),(B,0)\}$ as equally complex Teaching Sets.

SumOfExamples

```python
 def get_complexity_of_subset(self, teaching_set):
        """Compute the complexity of the examples given"""
        score = sum([len(example) for example in teaching_set])
        score += sum([0.1 for example in teaching_set if len(example)
            ↪ == 0])
        return score
```

We present *SumOfExamples* as a reasonably straightforward δ-function that captures the idea of complete information well. With *SumOfExamples*, we do not care how the present letters in S are distributed in examples. We only care about how many total present letters there are in S. The same code line for punishing empty examples as seen in *SquaredSum* is also present in in *SumOfExample*, the reasoning is the same.

As stated in the beginning of this chapter we will compare these suggested δ in our result chapter, Chapter 7.

Chapter 6

Subset selector σ - $\text{argmin}_{S:\theta_{AI}\models S}$

After defining all of δ,λ and L_M, we can now evaluate how good a given Teaching Set is. As our goal is to find a Teaching Set explaining some θ_{AI}, being able to score Teaching Sets is a good step in the right direction. Now we only have to iterate over potential Teaching Sets and select the best one.

This iteration process of selecting a Teaching Set will be denoted σ and will be described in this chapter. Once more, we bring the readers' attention back to the formula at the centre of attention in this thesis. In Equation 6.1 we observe that the selection of subsets to be evaluated is defined by $\text{argmin}_{S:\theta_{AI}\models S}$. In this chapter we present three different implementation of this argmin and discuss their benefits and disadvantages. We call these implementations subset selectors, and denote them σ. We leave the comparison of the suggested subset selectors σ to Chapter 7.

An overview illustrating how the subset selectors fits in to the rest of the system is given in the Figure 6.1.

$$T(\theta_{AI}) = \underset{S:\theta_{AI}\models S}{\text{argmin}}\{\delta(S) + \lambda(\theta_{AI}, \theta_M) : L_M(S) = \theta_M\} \tag{6.1}$$

$$L_M(S) = \underset{\theta_M:\theta_M\models S}{\text{argmin}}\{\beta(\theta_M)\}$$

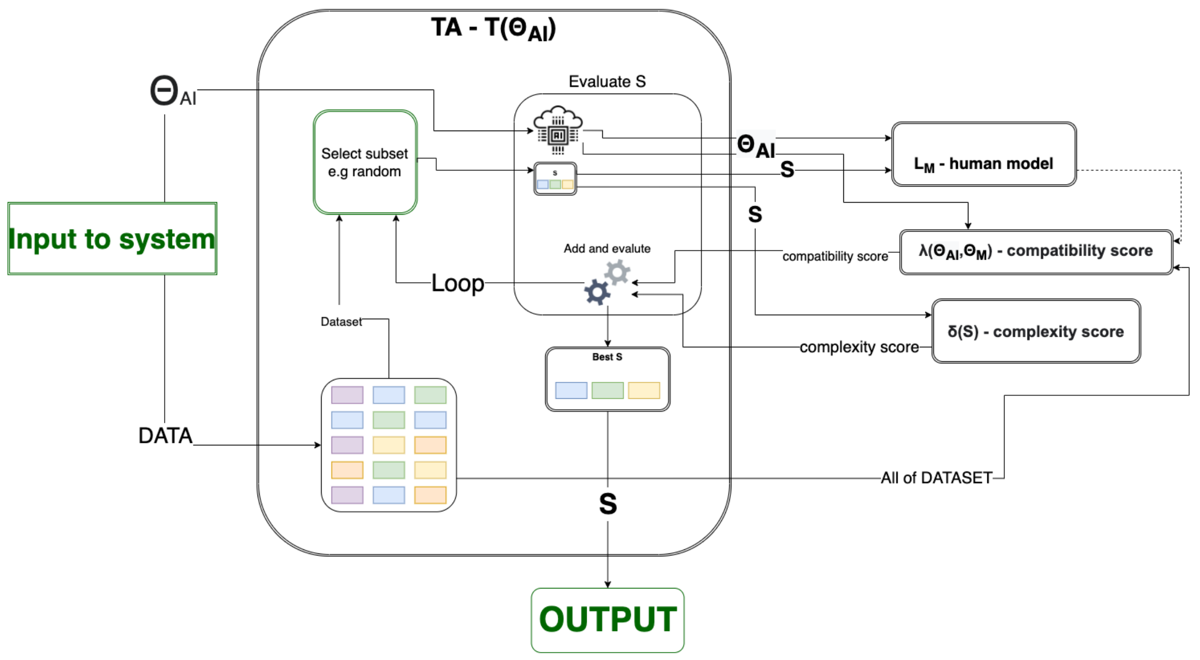


Figure 6.1: Overview of the central teaching system. The input is AI to be taught and a data set. First TA use σ to select a Teaching Set – the green box "Select Subset". Secondly TA is the control unite in the program. It make calls to L_M, λ and δ, combines the results of each sub system into a score for each Teaching Set.

6.1 Different subset selectors σ

We now introduce three different techniques to perform the selection of Teaching Set to be evaluated. Up to this point, we have used the representational language of present letters R_L quite a lot. The input of L_M is S^L, the evaluation in λ is done using present letters as R_L, and the input to δ is S^L. As all functions take as input S^L, two example bitmaps containing the same letters will be evaluated identically[1]. Therefore, all three subset selectors use R_L in their selection of a Teaching Set to avoid having "duplicate" bitmaps – same present letters – in a Teaching Set. This ensures the **R_L-consistent-S** we have been reasoning with throughout, ensuring Teaching Sets like $S^L = \{(A, 1), (A, 0)\}$ is not allowed. An overview of the three suggested subset selectors σ is given in Table 6.1.

[1]Assuming θ_{AI} predictions are the same

Subset selector	Description
RandomSelect	Baseline. Select random examples to be in Teaching Set.
RandomWHash	Random select w/ hashing to avoid repeated selecting of equal sets.
ExhaustiveSearch	Enumeration of all possible Teaching Sets, and iteratively try all.

Table 6.1: Table of all subset selector implemented in this thesis

RandomSelect

```
def get_next_subset(self, previous_score, previous_subset):
    picks = []
    possiblilities = get_all_letter_combinations()
    while(len(picks) < get_sample_size()):
        letters = choice(possiblilities)
        possiblilities.remove(label)
        to_add = choice(self.letters_to_data[letters])
        picks.append(to_add)
    picks.sort(key=lambda x: x[2])
    return picks
```

RandomSelect aims to be a simple baseline for comparison. We randomly select examples to make up the Teaching Set. To ensure R_L-*consistent-S*, we randomly select unique combinations of letters for each example and then create examples from this. Other than this, RandomSelect is a rather plain and straightforward subset selector.

RandomWHash

```
def get_next_subset(self, previous_score, previous_subset):
        while True:
            pick = self.randomSelector.get_next_subset(
                previous_score, previous_subset)

            chosen_labels = sorted([str(pL+str(pY)) for pX, pY, pL in
               ↪ pick])

            if chosen_labels in self.tried:
                # To avoid infinite spin
                if random.randint(0, 10000) < self.wait_factor:
                    self.wait_factor *= 2
                    pick.sort(key=lambda x: x[2])
                    return pick
                continue
            self.wait_factor = max(self.wait_factor//3, 2)  # never go
               ↪ below 2.
            self.tried.append(chosen_labels)
            pick.sort(key=lambda x: x[2])
            return pick
```

RandomWithHash aims to be a simple upgrade on Random select. One flaw of *RandomSelect* is that it might select the same Teaching Set multiple times. Moreover, it often can select identical Teaching Sets in R_L. To avoid evaluating identical Teaching Sets multiple times, we introduce hashing of the examples in R_L with *RandomWHash*.

RandomWHash retrieves a random example from *RandomSelect* and checks if it has seen equivalent Teaching Set before, i.e. containing the same letters with the same predictions. If *RandomWHash* finds it has seen this Teaching Set before, it asks for a new one. It repeatedly selects a new one until it gets one it has not seen before. We have implemented a stopping mechanism to avoid *RandomWHash* spinning forever. The stopping mechanism is a decreasing counter for the number of search attempts it will perform before returning any Teaching Set[2].

We expect *RandomWHash* to perform better than *RandomSelect*, especially when allowed to search for a while, and the hash functions start to reject already attempted Teaching Sets.

ExhaustiveSearch

```
def initialization:
        """
        Full code at
        https://github.com/BrigtHaavardstun/ExplainableAI/
        """

        #Pre computation
        #[...,[A,AB,C], [A,AB,D], [A,AB,AC],...]
        letter_combination = generate_all_combinations_of_letters()
        to_try=[]
        for label_combination in letter_combination:
            new_teaching_sets_to_try = []
            for all examples in label_combination:
                add combination with example = True
                add combination with example = False

            to_try.extend(new_teaching_sets_to_try)
        # ToTry holds all possible Teaching Sets
```

```
def get_next_subset(self, previous_score, previous_subset):
        return to_try.pop()
```

With *ExhaustiveSearch*, we have implemented an iterative search through the logical search space of present letters R_L. This method is the most sophisticated search algorithm presented in the thesis. It brings the possibility of exhausting the search space. By allowing *ExhaustiveSearch* to exhaust the search space, we guarantee finding the optimal S.

We have therefore used *ExhaustiveSearch* when evaluating δ-functions, as we mostly find it interesting which S each δ-function evaluates to be the minimum and not some

[2]As returning a Teaching Set already evaluated does not bring any new information one could also simply stop the search at this point.

local minima. Regardless of the comparison between the subset selectors, we find that having a subset selector capable of exhausting the search space and conclusively finding the best Teaching Set is worth any other potential downside in many use cases.

One of these downsides could, for instance, be increased memory use and pre-computation time. *ExhaustiveSearch* generates a rather huge list on the order of $O(2^{2^{\Sigma}})$[3] - as discussed in Section 3.3. It is completely infeasible to store this list all at once. *ExhaustiveSearch*, therefore, puts quite a strict limitation on the size of the alphabet. One solution to this problem would be to make a new version of *ExhaustiveSearch*, which can generate all these subsets one after the other. One would then be able to run *ExhaustiveSearch* with a larger alphabet Σ without crashing the system. However, the exhaustive search part would not terminate in any reasonable time for a moderately sized alphabet Σ.

[3]If θ_{AI} is not R_L-consistent-AI we get $O(2^{2^{|\Sigma|}} * 2^{2^{|\Sigma|}}) = O(2^{2^{|\Sigma|+1}})$

Chapter 7

Results

In this chapter, we will be comparing the different parts of our system. We will look at which of the suggested δ performs the best. Then we continue on and look at which subset selector σ is the best. Finally, we present results on how well we are able to describe different θ_{AI}.

7.1 Comparing suggested δ

In what follows, we compare the different Teaching Sets selected using different deltas. We remark that we have weighted *compatibility* λ much higher than *complexity* δ for this experiment. That is, in our score function for a Teaching Set we set our multiplier μ high, such that $\gamma(\mu = 100) = \delta(S) + \mu * \lambda(\theta_{AI}, \theta_M)$. Even for a small decrease in λ the overall score would still be better for a modest increase in $\delta(S)$. We would therefore often find a Teaching Set S, s.t. $\lambda(\theta_{AI}, \theta_M)$ is the minimum, where $L_M(S) = \theta_M$. For these experiments, we allowed the system to perform an exhaustive search, trying all logically possible combinations of examples as Teaching Sets and finding the optimal one.

Although all δ produce a score of complexity, the value they return is not comparable across different δ-functions. If *SquaredSum* evaluates one Teaching Set to have a complexity of 6.4 and *Chunking* evaluates another Teaching Set to have a complexity of 3.2, we have no information on which Teaching Set is more complex.

Therefore, we are only interested in comparing the minimum Teaching Sets each δ-function finds. First, we will look at how many equally optimal Teaching Sets each δ finds. If one δ-function finds too many equally optimal Teaching Sets, this might indicate that the δ-function is a poor discriminator. A poor discriminating δ is not ideal, as the final Teaching Set becomes random among all equal Teaching Sets.

	#Minimum Teaching Set: (A and !B and D) or (B and C and !D) or (B and !C and D)	#Minimum Teaching Set: (A and B) or (C and D)	#Minimum Teaching Set: not (C and D)
Cardinality	5	6	4
Chunking	1	4	2
SquaredSum	1	2	2
SumOfExamples	2	14	2

Table 7.1: The number of equally scored minimum Teaching Sets for different δ-functions, ran on different θ_{AI}, here defined by their ϕ-function.

Setting for these experiments

We performed an exhaustive search of all possible Teaching Sets using subset selector *ExhaustiveSearch*. The alphabet used was $\Sigma = \{A, B, C, D\}$. We test with different ϕ and corresponding θ_{AI}. θ_{AI} was always trained on a data set of size 10000 and achieved 95+% validation accuracy for all instances. $T(\theta_{AI})$ was modified to store all Teaching Sets with the minimum score instead of just one, so we could see all potential outputs of our system.

Number of equally optimal Teaching Set

When performing these experiments, all our δ-functions did not punish empty examples. One of the discoveries in this experiment is that this leads to "duplicate" minimum Teaching Sets padded with empty examples. As this was not intended, we changed this in all δ-functions as presented earlier.

Moving on to the analysis of the data. In the Table 7.1 we present how many equally good Teaching Sets S there is for different δ. Most of the time, our suggested δ-functions found between 1 and 4 different minimum Teaching Sets. Another observation we can make from the Table 7.1 is that the δ-functions rarely only find one minimum Teaching Set. At the time of the experiment, many of these functions did not punish having an extra empty example. Therefore, we hypothesised that these deltas found two Teaching Sets with only an empty example in difference. As a control, we looked at some of the Teaching Sets produced.

From Table 7.2 we found our hypothesises holds in this one case for all δ-functions expect *Cardinality*. As having an extra example increases the cognitive load, even if it is

Minimum Teaching Set	The delta(s) which produced the Teaching Set
{(ABC,1),(ABCD,0),(ABD,1)}	Cardinality
{(AC,1),(ACD,0),(AD,1)}	Cardinality
{(C,1),(CD,0),(D,1)}	Cardinality, Chunking, SumOfExamples, SquaredSum
{(#,1),(C,1),(CD,0),(D,1)}	Chunking, SumOfExamples, SquaredSum
{(BC,1),(BCD,0),(BD,1)}	Cardinality

Table 7.2: The minimum Teaching Sets for $\phi = \neg(C \wedge D)$. All minimum Teaching Sets found, and which δ-functions whom found each S to be a minimum Teaching Set.

Minimum Teaching Set	The delta(s) which guessed it
{(A,0),(AB,1),(ABC,0),(B,0),(D,1)}	Cardinality, Chunking, SumOfExamples, SquaredSum
{(A,0),(AB,1),(ABC,0),(ABCD,1),(B,0)}	Cardinality
{(A,0),(AB,1),(ABC,0),(B,0),(BCD,1)}	Cardinality
{(A,0),(AB,1),(ABC,0),(AD,1),(B,0)}	Cardinality
{(A,0),(AB,1),(ABC,0),(ACD,1),(B,0)}	Cardinality
{(A,0),(AB,1),(ABC,0),(ABC,0),(B,0),(CD,1)}	Cardinality

Table 7.3: ϕ = (A and B and not C) or D. All minimum Teaching Sets found, and which δ-functions whom found each S to be a minimum Teaching Set.

empty, we introduced a small punishment for having an empty example in our δ-functions. This punishment brought our δ-functions up to the format we presented earlier in Chapter 5. To check if this had the desired effect, we rerun our experiment for a different ϕ, now with the punishment of empty examples.

This run further strengthens our hypothesis that when punishing empty examples, most of our λ-functions effectively finds a small group (or even just one) minimum Teaching Set S. From Table 7.3 we can see that all λ-functions only has found one minimum Teaching Set S – with exception of our baseline function *Cardinality*.

From these observations, we conclude that all our delta functions are better discriminators than *cardinality*. Since all of the δ performed well in the experiment, we arbitrarily select *SquaredSum* as the δ we will use for later experiments.

7.2 Comparing the different subset selectors σ

This section presents data from runs of the proof of concept system with different subset selectors σ. We will run the system 50 times for each subset selector for a given setup. We then calculate how good the average Teaching Set is for each σ. We then slightly change the setup and do 50 more runs per σ.

When setting the setup of our system we can decide the number of Teaching Sets we are to evaluate before returning the minimum found. For each iteration of setting a new setup, we change the number of Teaching Sets the σ is allowed to check. We note that if we allow the subset selector to sample "n times", we let it select n potential Teaching Sets S from each possible cardinality size[1]. For each n of sample attempts, we calculate the average over the 50 individual runs to hopefully gather insight into the average performance of the subset selectors when given n sample attempts.

Settings for experiment

θ_{AI} was always trained to 100% validation accuracy. To achieve a lower search space, we lowered our alphabet to contain only three letters $\Sigma = \{A, B, C\}$. Doing this allows us to observe (in a reasonable time) what happens when $ExhaustiveSearch$ exhausts the search space.

Data set $\phi = (A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C)$

For the first data presented θ_{AI} was trained with $\phi = (A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C)$. We picked ϕ to be complex in this case and therefore suspect that finding an Teaching Set S having a low γ-score will require more sampling than a less complex ϕ-function. We set $\mu = 100$ in our score function $\gamma(\mu = 100) = (\delta(S) + 100 * \lambda(\theta_{AI}, \theta_M))$, indicating that we prefer *compatibility* $- \lambda$ at the cost of *complexity* $- \delta$.

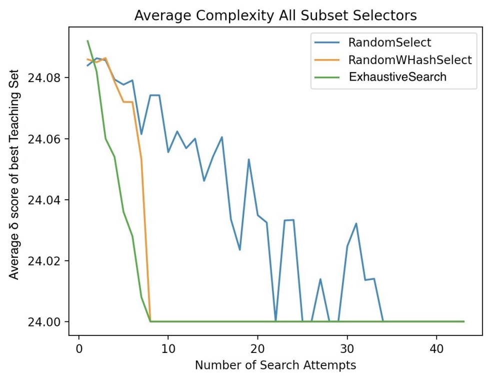


Figure 7.1: Graphing average complexity

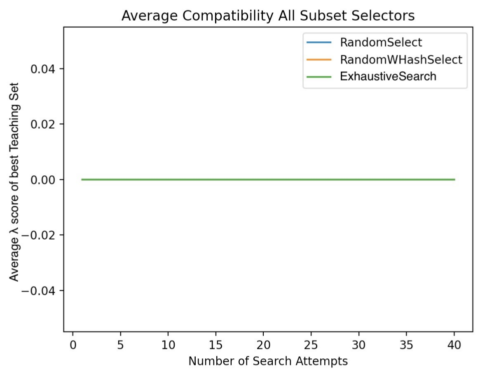


Figure 7.2: Graphing average compatibility

The first observation we would like to draw attention to in Figures 7.1, 7.2 and 7.3 is that in Figure 7.2 the *Compatibility*-score is always 0, e.g. perfect, for all σ. We

[1]This is done because of practical reasons in implementation of the Proof of Concept.

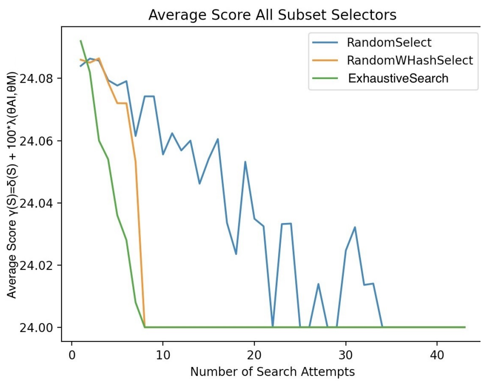


Figure 7.3: Graphing average score

hypothesise that this is a effect of our γ-function having a high $\mu = 100$. Having a high μ leads our system to select the most λ-compatible Teaching Set S, and almost disregard the complexity from δ.

In Subsection 4.1.1 we showed that if θ_{AI} is R_L**-consistent-AI** there always exist a θ_M for which $\lambda(\theta_{AI}, \theta_M) = 0$. We now argue why all our different σ finds this θ_M even for a single search attempt.

As described earlier, one search attempt allows σ to test one Teaching Set of each cardinality. Hence, even at one sample attempt, σ selects one Teaching Set of cardinality eight. A Teaching Set of size eight has to contain on example for all of $\Sigma_C = \{\#, A, B, C, AB, AC, BC, ABC\}$ to be R_L*-consistent-S*. Given θ_{AI} to be R_L*-consistent-AI*, all the examples can only be associated with one predicted value. When all examples are defined by Σ_C, and their predicted values constant, there can only be one Teaching Set of size 8. With only one Teaching Set S^L of size eight, we conclude that all our subset selectors σ find this one.

This Teaching Set is constructed similarly to the process described in Chapter 4. Therefore, we argue that this Teaching Set will have a λ-score of zero, given θ_{AI} to be R_L*-consistent-AI*. We therefore, conclude that all σ will always find a Teaching Set with λ-score zero, even for one sample attempt.

The next observation we will discuss is the steep fall in Figures 7.1 and 7.3 followed by a rather flat line after 10 search attempts. For one search attempt, we observe that the average Teaching Sets have a score of 24.08. Looking into our data, we observe that this is due to our system finding two different minimum Teaching Sets. One of them is the elementary Teaching Set of cardinality eight:

$$S^L = [(\#, 0), (A, 0), (B, 0), (C, 0), (AB, 1), (AC, 1), (BC, 1), (ABC, 0)]$$

This Teaching Set will get a λ-score of 0, and a complexity score[2] of $0.1 + 1^2 + 1^2 + 1^2 + 2^2 + 2^2 + 2^2 + 3^2 = 24.1$.

On the other hand, we have the Teaching Set of cardinality seven:

$$S^L = \{(A, 0), (B, 0), (C, 0), (AB, 1), (AC, 1), (BC, 1), (ABC, 0)\}$$

This Teaching Set will get a λ-score of 0, and a complexity score[3] of $1^2 + 1^2 + 1^2 + 2^2 + 2^2 + 2^2 + 3^2 = 24$. To find this Teaching Set, one needs to select these seven examples and drop the empty example. There are $\binom{8}{7} = 8$ unique Teaching Set of cardinality seven. To select the minimum Teaching Set of cardinality seven is, therefore, a $\frac{1}{8}$ chance. In the beginning, some of the 50 runs are lucky and find the cardinality seven Teaching Set. As σ gets more search attempts, we observe that at eight search attempts, both *ExhaustiveSearch* and *RandomWHash* converge and consistently find the Teaching Set of cardinality seven as the minimum. This is due to them not repeating Teaching Sets. Given eight search attempts, both σ will try all Teaching Sets of size seven and will therefore consistently find the best.

For the given $\phi = (A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge C)$ it turns out that the Teaching Set of cardinality seven is the optimal Teaching Set for this experiment. Therefore we do not see any more change in our graph.

It turns out that our hypothesis was somewhat wrong for this experiment. A complex ϕ did not mean we had to use more search attempts to find the optimal Teaching Set. We found the optimal Teaching Set consistently after eight search attempts. From this, we conjecture that it might be simpler to find the optimal Teaching Set if it has a high γ-score.

Data set $\phi = (A \wedge B) \vee C$

In parallel with the first data set we made another data set. For the second data set we chose to train θ_{AI} on a less complex ϕ-function. We hypothesized this data set would lead to a more rapid convergence of Teaching Set S, concerning both compatibility and complexity, and hence converge faster to the optimal. The same score function was used in the second data set analysis, $\gamma(\mu = 100) = (\textit{complexity} + 100 * \textit{compatibility})$.

[2]Using *SquaredSum* as function
[3]Using *SquaredSum* as function

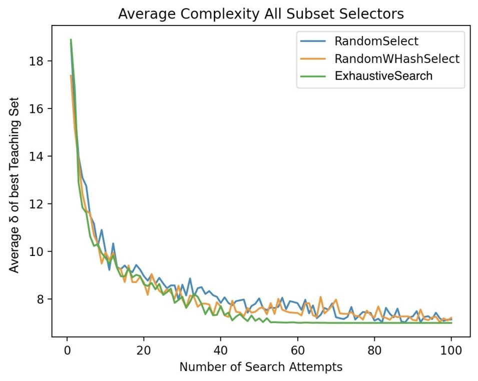


Figure 7.4: Graphing average complexity

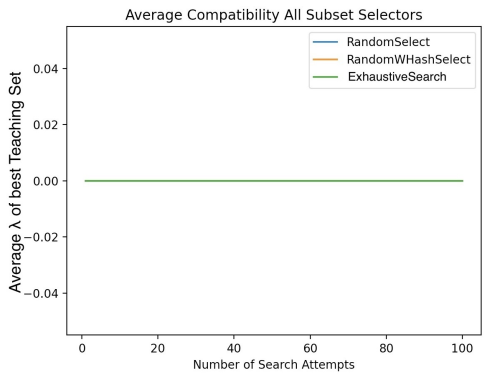


Figure 7.5: Graphing average compatibility

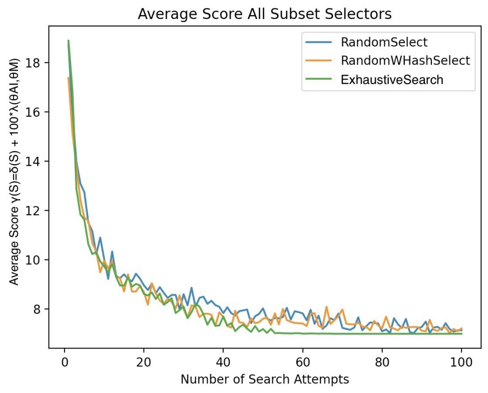


Figure 7.6: Graphing average score

We observe that these graphs have some similarities and some differences compared to Figures 7.1, 7.2 and 7.3. The first observation we make is that once again the compatibility score is perfect regardless of sample attempts, as shown with Figure 7.5. The reason for this is the same as discussed earlier – high μ value in γ and θ_{AI} is R_L*-consistent-AI*.

The next observation we would like to bring attention to is the fact that not all σ converges in Figures 7.4, compared to 7.1 where all converged. Looking at Figure 7.4 we observe that *ExhaustiveSearch* seems to be consistently finding the optimal Teaching Set given 50-60 search attempts[4]. We observe some noise in *RandomSelect* and *RandomWHash* regarding finding the optimal Teaching Set even at 100 sample attempts. This is in stark contrast to our initialize hypothesis that a simpler ϕ would lead to finding optimal Teaching Set faster.

Once again our hypothesis that a less complex ϕ would lead to faster converges to an optimal Teaching Set is flawed. This further strengthen our conjecture that it is simpler to find a optimal Teaching Set if the best Teaching Set is poor, i.e. has a high γ-score.

[4]Looking into the data we observe that *ExhaustiveSearch* exhausts the search space after 70 attempts.

Search Attempts	RandomSelect	RandomWHash	ExhaustiveSearch
5	0,24	0,21	0,29
50	0,48	0,63	0,54
100	0,76	0,99	0,83
200	1,46	2,01	1,36
500	3,23	9,45	3,03
1000	5,96	15,95	5,82
2000	12,44	24,32	10,37

Table 7.4: Table showing time used in seconds for different σ.

We conclude the discussion on subset selectors σ by stating that $ExhaustiveSearch$, as expected, seems to be the best σ. However, we have thus far not presented any time analysis comparing the different σ. In the Table 7.4 we see the average time each σ use given different search attempts. Surprisingly $ExhaustiveSearch$ comes out on top for large search attempts. The reason for this is that while $RandomSelect$ and $RandomWHash$ spend much time generating duplicates of Teaching Sets, $ExhaustiveSearch$ terminates early if it has found all Teaching Sets of a given cardinality. For instance, whereas $RandomSelect$ and $RandomWHash$ generate 2000 Teaching Sets of cardinality eight, $ExhaustiveSearch$ only generates 16 before stopping. As expected, $RandomWHash$ comes out last. It spends much time hoping to find a new Teaching Set before selecting them, and we now see the cost for the gain we observed in Figure 7.3. We conclude that $ExhaustiveSearch$ is by far the better σ, as it is both faster and finds better Teaching Sets.

7.3 Better accuracy gives better teaching set score

When aiming to explain an AI, we hypothesise it to be easier to explain an AI with high validation accuracy. We suspect an θ_{AI} with higher accuracy will tend to be better aligned with our representational language R_L. The better alignment should lead to a lower λ-score, and hence a lower total γ-score. To verify that our system works as intended and test this hypothesis, we train nine different AIs with differently sized data sets. We use differently sized training sets to achieve different level of accuracies for the AIs. All AIs are trained with the ground truth $\phi = (A \wedge B) \vee C$ and the alphabet $\Sigma = \{A, B, C\}$. The data set sizes used in the experiment are: $\{10, 50, 100, 500, 1000, 2000, 5000, 10000, 50000\}$, accordingly we denote the different AIs: $\{AI_{10}, AI_{50}, AI_{100}, AI_{500}, AI_{1000}, AI_{2000}, AI_{5000}, AI_{10000}, AI_{50000}\}$.

In Table 7.5 we display the different validation accuracies of the AIs. The most surprising result is that AI_{10} achieves the best accuracy of 100%. Looking into the data

AIs	AI_{50}	AI_{100}	AI_{500}	AI_{2000}	AI_{1000}	AI_{5000}	AI_{10000}	AI_{50000}	AI_{10}
Accuracy	75	83	84	89	90	95	99	99	100

Table 7.5: Accuracy of different AIs.

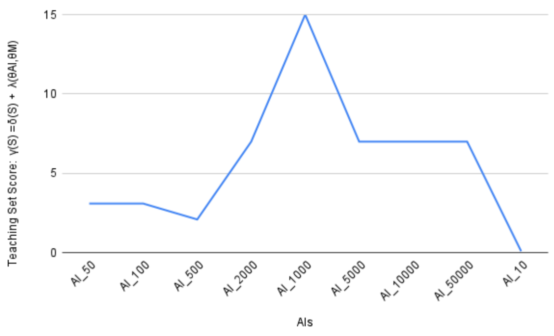


Figure 7.7: Graph of the Teaching Set scores of the AIs.

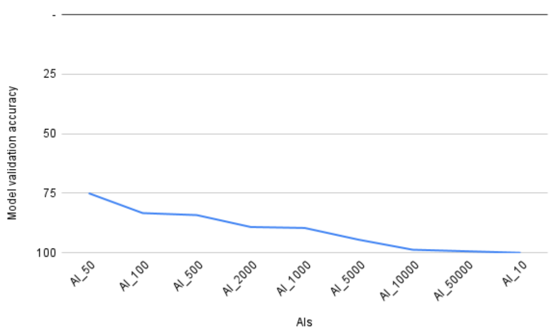


Figure 7.8: Graph of the validation accuracies of AIs.

set we used to train AI_{10} we find that all data instances in the small data set are labelled 1. Hence, AI_{10} always predicts True, regardless of the input. It was not our intention to have AI_{10} always predict True, however it will be interesting to see if we can explain this AI_{10}.

To help gather insight into this data we also visualize all of the data in Figures 7.7, 7.8, 7.9, 7.10. One interesting observation we can make from this is that it seems the δ scores and λ scores are inversely related. When δ is high, λ seems to be low, and low δ seems to be correlated with a high λ. We will not look more into this as our goal is to look at accuracy related to score.

Based on the hypothesis that it is easier to explain an AI with high validation accuracy, we expect to find that the higher the accuracy of the AI, the lower the score of the

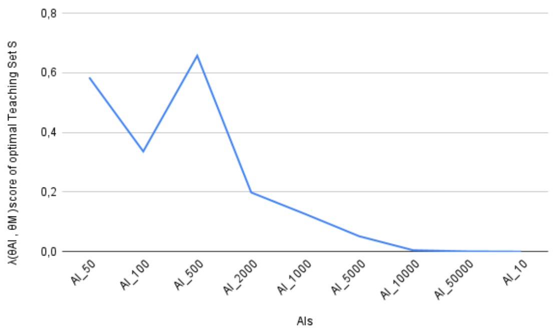


Figure 7.9: Graph of the λ scores of the Teaching Sets for the AIs.

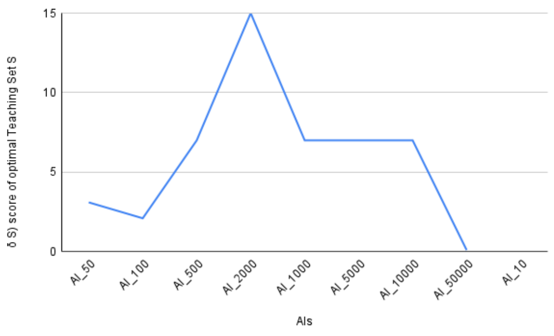


Figure 7.10: Graph of the δ-scores of the Teaching Sets for the AIs.

	AI_{50}	AI_{100}	AI_{500}	AI_{2000}	AI_{1000}	AI_{5000}	AI_{10000}	AI_{50000}	AI_{10}
Delta	3,1	3,1	2,1	7	15	7	7	7	0,1
Lambda	0,5842	0,3362	0,6573	0,1985	0,1263	0,0518	0,0049	0,0009	0
Total gamma	61,52	36,72	67,83	26,85	27,63	12,18	7,49	7,09	0,1

Table 7.6: Delta, Lambda and Gamma-score of best Teaching Set for different AIs.

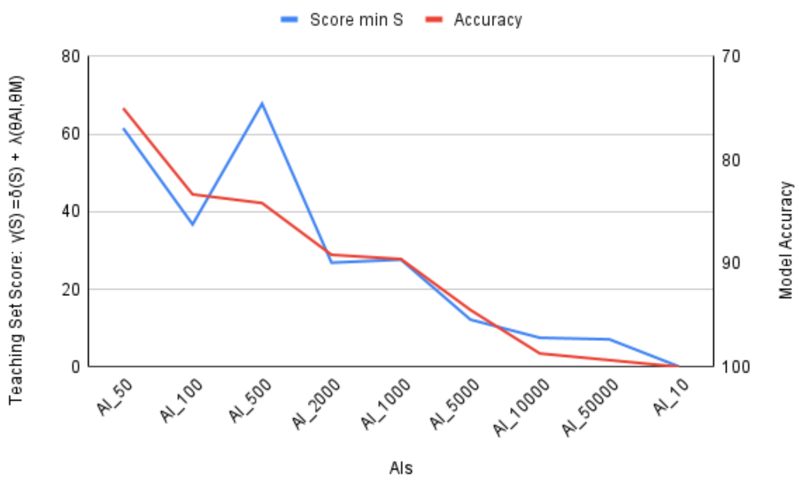


Figure 7.11: Accuracy and score of optimal Teaching Set for different AIs sorted by accuracy.

best Teaching Sets. We therefore run our system with these AI. The subset selector used is **ExhaustiveSearch**, and we allowed it to exhaust the search space in R_L – 1500 – 2000 search attempts. The data set contained 5000 data instances, and we used **FreePlacement**, **Rotation** and **Scaling**. The δ-function used is **SquaredSum**. We continue to prioritise a Teaching Set such that the guessed concept θ_M and θ_{AI} are quite similar. To do this, we set $\mu = 100$ in our score function γ. We record the γ-score, λ-score and δ-score of the best Teaching Set for all the different AIs as displayed in the Table 7.6.

Based on the information in Tables 7.5 and 7.6 we can make a graph looking at the correlation between accuracy and the γ-score of the best Teaching Set for each AI. We display this graph in Figure 7.11. From this graph, we observe that there seems to be a correlation between validation accuracy and the score of the best Teaching Set. In the graph we have moved AI_{10} to the end as the accuracy is perfect. Besides this, there is a clear deviation from the norm in the graph for AI_{500}. To better understand why this happens, we will compare AI_{500} with the AI of similar accuracy AI_{100}.

We begin the comparison by giving a breakdown of both AIs in R_L in Table 7.7. This table might give us insight into whether one of the AIs is more easily explained

AIs	AI_{100}	AI_{500}
#	0,00	0,00
A	0,71	0,49
B	0,63	0,36
C	0,65	0,49
AB	0,98	0,94
AC	0,99	0,94
BC	0,99	0,95
ABC	1,00	0,99

Table 7.7: Breakdown of AI_{100} and AI_{500} in R_L.

based on present letters than the other. In the breakdown, we show the percentage of bitmap predicted True, grouped by present letters in the bitmap. This breakdown shows that AI_{100} tends to predict bitmaps containing only A or B to be True more than AI_{500}. Since A and B alone are labelled 0, AI_{500} outperforms AI_{100} in this part of the data set, and this explains the increase in accuracy[5] seen with AI_{500}. We can see that the AI_{100} seems to mimic a concept closely matching "If there is anything in the picture, predict True". The Boolean expression $\phi = A \vee B \vee C$ quite easily explains this concept. On the other hand, AI_{500} seems to move in the direction of trying to detect "2 or more letters". The concept of "2 or more letters" is a good estimation of the ground truth $\phi = (A \wedge B) \vee C$ as it only misclassifies C[6]. To teach this concept the best Teaching Set is $S^L = \{(A, 0), (AB, 1), (AC, 1), (B, 0), (BC, 1), (C, 0)\}$, however this teaching set has the high δ-score of 15. Therefore our system instead chose to explain a concept almost identically well aligned with AI_{500}. This other concept is $\phi = (A \wedge B) \vee C$, the ground truth concept. To convey this concept our system generated the Teaching Set $S^L = \{(A, 0), (AB, 1), (B, 0), (C, 1)\}$[7]. This Teaching Set only have a δ-score of 7 and is therefore preferred.

Based on this discussion, we state that the essential feature to predict the best Teaching Set score for an AI is how well the AI is aligned with our chosen representational language. Because our ground truth function ϕ is defined in the representational language of present letters, we expected to be able to use accuracy as a good indicator for the score of the optimal Teaching Set. If an AI sufficiently matches our ground truth ϕ this relation is a given. However, this relation seems to hold even for AIs with a much poorer accuracy, and where the concept best matching the AI in our representation language is some other concept than the ground truth function.

[5]The AI AI_{100} outperforms AI_{500} in all other groups concerning the ground truth. However, the margins are small.

[6]The system chose a Teaching Set aimed at teaching $\phi = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)$ for AI_{1000}.

[7]This is the Teaching Set given for all of AI_{2000},AI_{5000},AI_{10000} and AI_{50000} as well

Chapter 8

Conclusion and future work

$$T(\theta_{AI}) = \underset{S:\theta_{AI} \models S}{\operatorname{argmin}} \{\delta(S) + \lambda(\theta_{AI}, \theta_M) : L_M(S) = \theta_M\} \tag{8.1}$$

$$L_M(S) = \underset{\theta_M:\theta_M \models S}{\operatorname{argmin}} \{\beta(\theta_M)\}$$

In this thesis, we have presented a proof of concept using Machine Teaching for explainable AI by implementing the formula repeated in Equation 8.1. We have discussed the different sub-goals of the formula and have given solutions and implementations for all of them. In the end we compared some of the different implementation variations and confirmed that our system acts as expected. We found evidence supporting that a higher accuracy leads to a lower Teaching Set score. With the overview of the work done in this thesis completed, we move on to look at some potential directions to continue this work.

8.1 Further comparing best Teaching Sets

In the thesis, we have compared the best Teaching Sets based on their accuracy. We also gained some insight into how Teaching Sets converge in δ and λ in our discussion of subset selectors σ. However, in this setting, we mostly aimed to compare how effective the σ was in finding the optimal Teaching Set. Therefore we used a R_L*-consistent-AI*, and a small alphabet $\Sigma = \{A, B, C\}$.

Future work could look into how the Teaching Sets converge given a larger alphabet and a θ_{AI} which is not R_L*-consistent-AI*. A larger alphabet comes with a larger and a more expressive representational language. From this, we might find a more gradual change in the best fitting concepts or Teaching Sets. Given a θ_{AI} which is not R_L-*consistent-AI*, our conjecture that a complex teaching set is easier to find as stated in section 7.2 might change. With a non R_L-*consistent-AI* θ_{AI} we will not only have one Teaching Set for max cardinality. Instead, we will have $2^{|\Sigma_C|}$ Teaching Sets of max cardinality.

This thesis primarily looks at Teaching Sets S with a low λ score. In future, it could be interesting to focus more on δ, and compare how this changes the Teaching Sets found. For instance, one could gradually shift the balance of λ and δ for the same θ_{AI}, and compare the different Teaching Sets one finds.

We only saw some tendencies in Section 7.3 that a Teaching Set with a low λ score might come at the cost of high δ and high delta. It would be interesting to look into how the Teaching Sets in our search gradually improves in this regard. We expect the λ score to start high, and our system then slowly finds better δ. This would be followed by a "discovery" of a better aligning concept, giving lower λ. However, the Teaching Set aimed to teach this new concept might be overly complex with a large δ. The system would then start to find a less and less complex Teaching Set for this new concept bringing δ down. This cycle of new concepts, bringing λ lower and a spike of δ, followed by a steady decline improvement of δ would be interesting to check if one would be able to detect.

8.2 Performing trials on humans

Although the system finds the minimum Teaching Set as we have defined it, one should perform trials to explore whether or not this explanation works for humans. To do this one should perform human trials. In the project proposal by Telle et al. [21] there are plans to have such trials to measure the effectiveness of the work.

There would be two main goals for such a trial. One would be to measure how closely L_M agreed with real humans, and the second would be how well the δ-function predicts which Teaching Sets humans find confusing or complicated.

When preforming such trials, a fundamental question would be whether Teaching Sets found by our system preforms any better than randomly selected Teaching Sets. It would be natural to draw experience from the work done by Yang et al. [24] where they measured the effectiveness of examples-based explanations for AI using Bayesian Teaching. Hopefully, we would be able to explain an AI if it were highly accurate and thereby aligned with our representational language, i.e. hoping for high sensitivity as described by Yang et al. [24]. It would also be of great interest to see if we were able to explain an AI with more errors with respect to the ground truth, i.e. achieve high specificity as described by Yang et al. [24] . One reason for why this might be the case is that we have observed that our Teaching Sets are able to find other concepts that better match the AI when the ground truth is a poor alignment.

8.3 On the topic of L_M

8.3.1 New representational language

The inputs we feed to our Karnaugh Map are four variables defined for our representational language of present letters. We could imagine that these four variables could represent something completely different in another representational language. For instance, we could imagine an image always containing one A and one B. However, in each image the location of the A and B changes from *high* to *low*. One could then create the representational language of the high-low location of A and B. One would have the four variables: A_H representing A being in a high location, A_L representing A being in a low location, B_H being that B is in a high location and B_L for B in a low location.

Training an AI on a function $\phi = A_H$ one would have a similar system as discussed in this thesis. The main difference would be that one now could look into partial explanations. Even if all data in the training data set contain both A and B, one might be able to create Teaching Set $S = \{(A_H, 1), (A_L, 0)\}$. When presenting this S to a human, one would show the human two images, one labelled positive with only a single A high in the picture and the other with a single A low. These images does not exist in the training data but are instead artificially created using parts of images in the training data.

This new problem would fit seamlessly into most of the work in this thesis. It would look into whether it is beneficial to create new concentrated artificial data instances as explanations.

8.3.2 Quine–McCluskey speedup

Suppose one were to increase the alphabet Σ. In that case, one should also change the boolean minimisation from Karnaugh Map to the Quine–McCluskey algorithm as described by Quine et al. [16] and McCluskey et al. [14]. The Quine–McCluskey algorithm runs faster on larger alphabets.

8.4 On the topic of δ

In the thesis, we have shown some implementations of δ based on information in the Teaching Sets. In the future, one might want to create better models of complexity and cognitive load. Inspired by Sweller et al. [19] and their use of production systems to measure cognitive load, one potential direction we suggest for future work is to take information from L_M into δ. To achieve this, we would have to redefine the formula for teaching to be:

$$T(\theta_{AI}) = \underset{S:S\models\theta_{AI}}{\operatorname{argmin}}\{\delta(S, L_M) + \lambda(\theta_{AI}, \theta_M) : L_M(S) = \theta_M\} \tag{8.2}$$

In Equation 8.2 we can measure attributes of L_M when given S to see how complex S is. When defining L_M, we use a modified Karnaugh Map to find Boolean expressions matching our Teaching Sets. If the Karnaugh Map finds only one Boolean Expression, this might be a good indicator of a simple Teaching Set.

On the other hand, the Karnaugh Maps might find multiple Boolean Expressions, e.g. $\phi_a = (A \wedge \neg B) \vee (A \wedge \neg C)$ and $\phi_b = (A \wedge B) \vee (A \wedge \neg D)$. One would then have to apply rules defined in β to find the minimum. As a reminder, we the rules defining β is given in the Equation 3.7

We can see that the number of clauses in ϕ_a and ϕ_b is the same. Hence we can not apply the first rule. Moving on to the next rule we attempt to compare the minimum clauses. We observe that the minimum clause in both expressions is the clause $(A \wedge B)$. Two identical clauses have the same β evaluation, so we move on to the next clause. This is $C_a = (A \wedge \neg C)$ and $C_b = (A \wedge \neg D)$. We see that the number of letters and negations are the same. In the end, we can differentiate them based on their lexicographical order. Using this many steps in β might indicate that the Teaching Set S is complex.

We conjecture that a L_M and a humans conclusion will differ more the more steps L_M used in β. To support the conjecture, we observe that the human is not told the rules in β. Therefore it is unlikely they will have the same rules in their mind. Even if they did, it would be harder for a human to find the one correct concept if there are a lot of similar concepts matching the Teaching Set.

8.5 On the topic of subset selector σ

On the selection of subset selector there is a huge potential for approximation and optimisation algorithms to be done. We have a clearly defined search space with adding/removing single letters from the examples in Teaching Sets, and one should be able to use this to define Local Search algorithms etc.

The benefit of continuing this work of improving subset selectors is increased speed. This opens up the possibility of a larger alphabet. With a larger alphabet, one could continue experimenting with how the different δ-functions compare and general aspects of a larger / more realistic domain to explain. If one is to achieve the benefits of a larger alphabet and search space, the improvement of subset selectors, σ is essential.

Bibliography

[1] Gi elevene nok lærere.
URL: `https://www.utdanningsforbundet.no/var-politikk/utdanningsforbundet-mener/artikler/gi-elevene-nok-larere/`.

[2] Md Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C. Van Esesn, Abdul A. S. Awwal, and Vijayan K. Asari. The history began from AlexNet: A comprehensive survey on deep learning approaches.
URL: `http://arxiv.org/abs/1803.01164`.

[3] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M. Paixão, Filipe Mutz, Lucas de Paula Veronese, Thiago Oliveira-Santos, and Alberto F. De Souza. Self-driving cars: A survey. 165:113816. ISSN 0957-4174. doi: 10.1016/j.eswa.2020.113816.
URL: `https://www.sciencedirect.com/science/article/pii/S095741742030628X`.

[4] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. 24(6):377–380. ISSN 0020-0190. doi: 10.1016/0020-0190(87)90114-1.
URL: `https://www.sciencedirect.com/science/article/pii/0020019087901141`.

[5] Nelson Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–114, February 2001. ISSN 1469-1825, 0140-525X. doi: 10.1017/S0140525X01003922.
URL: `https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/magical-number-4-in-shortterm-memory-a-reconsideration-of-mental-storage-capacity/44023F1147D4A1D44BDC0AD226838496`. Publisher: Cambridge University Press.

[6] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In Matthieu Cord and Pádraig Cunningham, editors, *Machine Learning Techniques*

for Multimedia: Case Studies on Organization and Retrieval, pages 21–49. Springer. ISBN 978-3-540-75171-7. doi: 10.1007/978-3-540-75171-7_2.
URL: `https://doi.org/10.1007/978-3-540-75171-7_2`.

[7] Shoshana Dreyfus, Susan Hood, and Maree Stenglin. *Semiotic Margins: Meaning in Multimodalities*. Bloomsbury Publishing. ISBN 978-1-4411-7016-3. Google-Books-ID: Z3M8CwAAQBAJ.

[8] Randi Eilertsen. Programmeringsemne er størst på MatNat.
URL: `https://www.uib.no/en/node/129155`.

[9] Rumjahn Hoosain. The processing of negation. *Journal of Verbal Learning and Verbal Behavior*, 12(6):618–626, December 1973. ISSN 0022-5371. doi: 10.1016/S0022-5371(73)80041-6.
URL: `https://www.sciencedirect.com/science/article/pii/S0022537173800416`.

[10] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 19–35. doi: 10.1109/SP.2018.00057. ISSN: 2375-1207.

[11] M. Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, November 1953. ISSN 2379-674X. doi: 10.1109/TCE.1953.6371932. Conference Name: Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics.

[12] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
URL: `https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html`.

[14] E. J. McCluskey. Minimization of Boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, November 1956. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1956.tb03835.x. Conference Name: The Bell System Technical Journal.

[15] Emmanuel M. Pothos and Andy J. Wills. *Formal Approaches in Categorization*. Cambridge University Press. ISBN 978-1-139-49397-0. Google-Books-ID: Z0eL3Cg5t_4C.

[16] W. V. Quine. The Problem of Simplifying Truth Functions. *The American Mathematical Monthly*, 59(8):521–531, 1952. ISSN 0002-9890. doi: 10.2307/2308219.
URL: `https://www.jstor.org/stable/2308219`. Publisher: Mathematical Association of America.

[17] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. Technical Report arXiv:1312.6229, arXiv, February 2014.
URL: `http://arxiv.org/abs/1312.6229`. arXiv:1312.6229 [cs] type: article.

[18] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. Technical Report arXiv:1409.1556, arXiv, April 2015.
URL: `http://arxiv.org/abs/1409.1556`. arXiv:1409.1556 [cs] type: article.

[19] John Sweller. Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2):257–285, April 1988. ISSN 0364-0213. doi: 10.1016/0364-0213(88)90023-7.
URL: `https://www.sciencedirect.com/science/article/pii/0364021388900237`.

[20] John Sweller. CHAPTER TWO - Cognitive Load Theory. In Jose P. Mestre and Brian H. Ross, editors, *Psychology of Learning and Motivation*, volume 55, pages 37–76. Academic Press, January 2011. doi: 10.1016/B978-0-12-387691-1.00002-8.
URL: `https://www.sciencedirect.com/science/article/pii/B9780123876911000028`.

[21] Jan Arne Telle, C´esar Ferri, Jose Hern´andez-Orallo, and Pekka Parviainen. Machine teaching for explainable ai, 2021.

[22] Mirko Thalmann, Alessandra S. Souza, and Klaus Oberauer. How does chunking help working memory? *Journal of Experimental Psychology. Learning, Memory, and Cognition*, 45(1):37–55, January 2019. ISSN 1939-1285. doi: 10.1037/xlm0000578.

[23] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, November 1984. ISSN 0001-0782, 1557-7317. doi: 10.1145/1968.1972.
URL: `https://dl.acm.org/doi/10.1145/1968.1972`.

[24] Scott Cheng-Hsin Yang, Wai Keen Vong, Ravi B. Sojitra, Tomas Folke, and Patrick Shafto. Mitigating belief projection in explainable artificial intelligence via Bayesian

teaching. *Scientific Reports*, 11(1):9863, December 2021. ISSN 2045-2322. doi: 10.1038/s41598-021-89267-4.
URL: `http://www.nature.com/articles/s41598-021-89267-4`.

[25] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. Technical Report arXiv:1311.2901, arXiv, November 2013.
URL: `http://arxiv.org/abs/1311.2901`. arXiv:1311.2901 [cs] type: article.

[26] Xiaojin Zhu, Adish Singla, Sandra Zilles, and Anna N. Rafferty. An overview of machine teaching. 2018.
URL: `http://arxiv.org/abs/1801.05927`.

Appendix A

Overview of system

This page is intentionally left blank to be able to scale up the figure to full size on the next page.

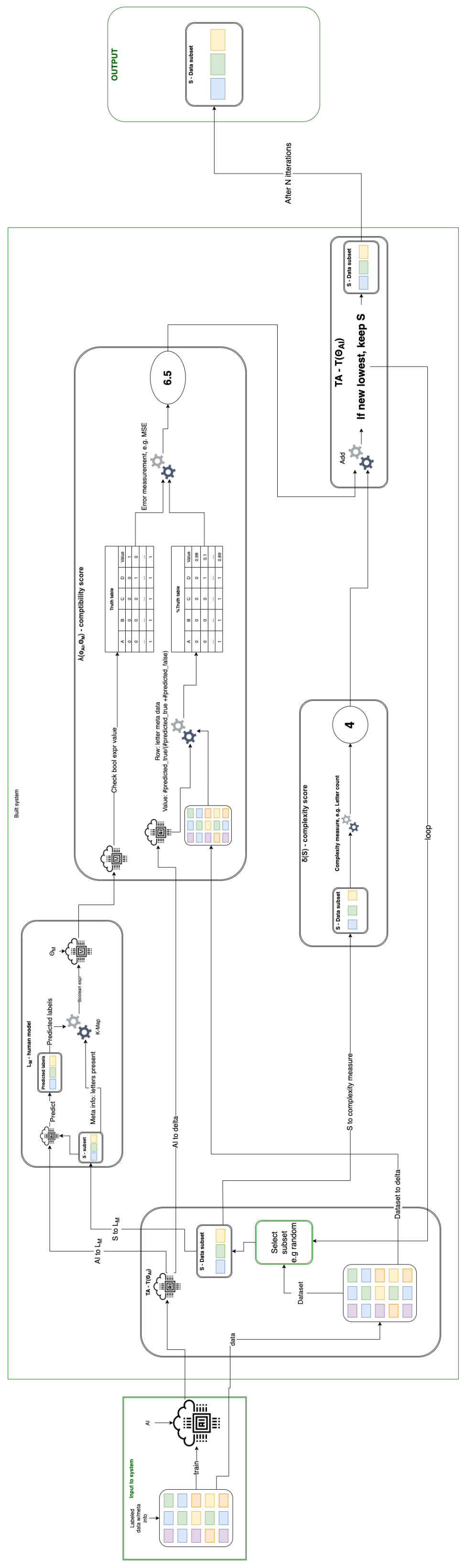


Figure A.1: Large detailed overview of system. All sub parts are presented in enlarged versions throught the thesis.