Designing Interpretable Chess Engines Using Neurosymbolic Methods

May 12, 2022

Abstract

The problem of interpretability in DNNs is classically a hard one, as the parameterisation of perceptron layers is hard to intuit. This paper uses neurosymbolic methods, namely Logical Neural Networks (LNNs), which encode boolean values using Real Logic, to construct an interpretable model for use cases which are easily modelled by logical statements, and applies this architecture to the problem of learning Chess.

Contents

Chapter 1

Interpretability

1.1 Introduction

Research into problems in Machine Learning in recent years has focused largely into using Neural Network (NN) models to solve increasingly broad categories of problems. NNs, much like many other models, define a hypothesis class of functions which differ only in their parameterisation, and uses Stochastic Gradient Descent (SGD) or some descendant thereof, to optimise said parameters given a loss function. Classical Multi-Layer Perceptrons (MLPs) consist of a series of linear layers separated by some non-linearity, (e.g. ReLU, Sigmoid functions). Given certain conditions, it is known that MLPs can learn any continuous function to arbitrary precision, by the Universal Approximation Theorem (UAT). The effectiveness of SGD methods allows us to learn arbitrary functions tractably given sufficient data, which has resulted in a widespread adoption of the architecture in practical settings.

A common criticism of NNs, however, is that they are considered "black box" functions. NNs are difficult to interpret, making it even more difficult to diagnose issues that may arise in production. A particular node in the network may be considered as capturing a single "concept", which can further be used to determine the presence of other concepts. It is difficult, however, to interpret how these concepts are generated - taking linear combinations of features and then applying a non-linear map to the result is not a terribly human line of thinking when it comes to pattern recognition. In this way, when comparing NNs to real-world neural processes, the description of NNs capturing general human intuition, rather than any kind of conscious reasoning, is most apt.

The "black box" nature of NNs has resulted in a hesitancy for it's adoption in particular settings, most notably in the medical industry, where even small risks of misdiagnosis cannot be tolerated. One would assume that an architecture that is so widely used in so many sensitive settings should be easily examinable, but this isn't that case.

From this, the notion of "interpretability" is often discussed when developing or analysing novel neural architectures ([23], [4], [7]). An ideal interpretable model would allow the user to know precisely what the model is achieving by arriving at a particular optimal parameterisation. Interpretability is not a quantifiable metric - the user may gain an understanding through a mixture of the learnt parameters and an existing intuition over the architecture itself. This allows for a wide breadth in methods which may be used to gain an understanding of a model, and also hopefully the underlying problem.

1.2 Interpretation Methods

There are many ways we may attempt to approach the problem of interpretability. Commonly used methods take arbitrary NN architectures, and attempt to gauge how relevant a particular feature of a given input is in the overall output of the model. These are known as *Variable Importance Measures* (VIMs). Gradient-based attribution methods [4] are the classical example, where the gradient of the model with respect to its input features are used as the measure of feature importance. This makes intuitive sense, as if the output of the model is subject to large changes with small deviations in an input feature, it must naturally be fairly important. The most commonly seen setting for these methods is in image classification, where the importance of a particular pixel measures how much of an influence said pixel has on determining the category of an image. Plotting the importance of all the pixels

hopefully shows the user precisely which portions of the image contain the relevant object to classification. E.g., distinguishing between cats and dogs would largely rely on examining particular features of the face shape, so one would expect these features to be the most important by this metric.

These methods are very versatile, as they are *model-agnostic*. There are many flaws, however - what if the classification of an image relies on a combination of features, rather than just a single one? We can capture this notion by instead using VIMs over all nodes in the network, i.e. the input layers and all hidden layers, but we run into the same problem - if we determine that a node in a hidden layer is important, how do we begin to understand what this node is doing? This method captures relevance, but does not capture what concepts these features may represent - we can leave this again up to the intuition of the user, or we can apply VIMs recursively between the input features and the hidden feature. This eventually becomes somewhat unwieldy.

Another issue is that VIMs are *local* interpretation methods, as they do not describe the model as a whole - only the model given a set input. This does not give us a good understanding as to why the model's solution to a problem is best.

A solution to the problem of not capturing feature relationships is in developing *model-specific* methods of interpretability. We can design an architecture which allows for novel ways of visualising model behaviour, often by restricting the expressiveness of the model in a manner which allows the remaining hypothesis class to be easily distinguishable.

One example of such an architecture are Neural Additive Models (NAMs) [3]. Neural Additive Models are a generalisation of General Additive Models (GAMs) in that they are fully described by the equation

$$M(\mathbf{x}) = \sigma\left(\sum_i M_i(x_i)\right)$$

Where the $M_i : \mathbb{R} \mapsto \mathbb{R}$ represent univariate NNs, and σ is the *link function*.

In backpropagation, we learn the parameters of each subnetwork M_i simultaneously. Given that each subnetwork is a map $\mathbb{R} \to \mathbb{R}$, we can capture the behaviour of the model not only locally through VIMs, but globally, as we can easily plot the value of M_i over the entire domain. Simply observing this graph allows the user to speculate as to what the model has learnt.

This model, while very interpretable, is not very expressive - we cannot capture any relationships between variables that aren't described by the link function σ, as is the nature of GAMs. This is the very problem we intended to solve by discussing NAMs - we want to be able to capture not only the relevance of input features, but of learnt concepts over those features.

Again, new model architectures have been introduced to resolve this. The aptly named Explainable Neural Network (xNN) [35] is an architecture which extends NAMs with a single linear "projection layer". These are equivalent to learning a GAM over *linear combinations* of input features. They are therefore fully described by the equations

$$\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$M(\mathbf{x}) = \sigma\left(\sum_i M_i(a_i)\right)$$

Where $\mathbf{W}, \mathbf{b}$ are learnable parameters as standard. We can interpret this model very similarly, by again plotting the univariate subnetworks M_i, and simply interpreting what a concept a_i represents by directly observing the linear combination. The UAT tells us that this single added layer is enough to model any function to an arbitrary precision, but in practice this hidden layer would need to be quite wide for anything other than the most trivial problems, and the features introduced in a large hidden layer may not be terribly intuitive to understand, and may even introduce a large bias.

We could extend this further, by adding more perceptron layers to capture more complicated relations between input features, but this naturally comes at the expense of interpretability. An ideal solution to this problem would allow for the model to be extended with more layers without sacrifice.

This motivates a new architecture for layers in our NN, as perceptron layers can be considered the main obstruction to interpreting our models.

Chapter 2

Logical Neural Networks

2.1 Introduction

We will discuss an NN architecture which attempts to learn boolean functions, instead of real functions. The restricted nature of boolean functions, and their use as a mathematical description of logic, allows a intuitive way of representating causation (e.g., if for some object x). The models that are created to solve such a problem are known as *Logical Neural Networks*[1] (LNNs)[2]. The subfield of ML concerned with learning solutions to logical problems with gradient-descent based methods is called *Neurosymbolic Machine Learning*.

The goal of the LNNs we discuss here will be to learn the optimal assignments for boolean variables within a statement described by the language of first order logic. This parallels the goal of MLPs to learn the optimal assignments for real variables within linear layers. LNNs achieve this through continuous optimisation using the standard approach of applying SGD methods with backpropagation.

It is interesting to note that the most widely researched approaches within the field of Artificial Intelligence (AI) until the 1990s were symbolic-based methods. There are many well studied algorithms that, given a set of logical predicates known to be true, attempt to capture the nature of the wider system in a single (ideally simple!) logical statement. These algorithms are known as Inductive Logic Programming (ILP) systems [20]. While very useful, they were found to be computationally intractible for more complex problems, notably those in Computer Vision and Natural Language Processing. Famously, Hubert Dreyfus predicted that symbolic methods were inherently incapable of fully capturing the complexity of such problems [10], stating that these relied primarily on unconscious processes rather than conscious symbolic manipulations. It seems most apt, therefore, to understand the rise of Deep Learning methods in the new millenium as reliant on capturing such "unconscious" processes.

The aim of neurosymbolic methods are thus to be able to capture the expressiveness of Deep Learning methods, without sacrificing the interpretability afforded by understanding the model in terms of symbolic manipulations. This is a difficult task!

2.2 Real Logic

We run into the important issue of $\{\mathbf{T}, \mathbf{F}\}$, the space of boolean values, not being continuous. One way of solving this issue is by extending the domain of possible logical assignments from $\{0, 1\}$, as above, to the closed interval $[0, 1]$. This is referred to as both *real logic*, (as in [31], [34], and much of the literature focusing on differentiable applications), and *fuzzy logic* (as in most foundational literature, namely [36], [19], [14]). Using the name "fuzzy logic" emphasises the importance of properties which are not relevant to the neurosymbolic architecture we will be discussing here, so we will not be using the term widely, but many of the following constructions are drawn from literature on fuzzy logic. We will discuss approaches in extending important logical operators (`OR`, `AND`, `NOT`, ...) to this new boolean space, and ways we may begin to use it to learn in an interpretable way.

[1] also *Neural Logic Networks, Logic Tensor Networks*
[2] also NLNs, LTNs

To fully define a "real logic", that is, an extension of classical logic to the space $[0,1]$, we want to define all the operators in a manner that, ideally, preserves many of the useful properties we see in the classical definition. At the very least, we want the values over the restricted domain $\{0,1\}$ to be the same as in classical logic.

It is well known that given a finite number of variables $\{x_i \mid i \in 1,\dots,N\}$, all boolean functions (that is, functions $\phi : \{\mathbf{T},\mathbf{F}\}^N \to \{\mathbf{T},\mathbf{F}\}$) can be expressed in terms of the operators $\neg$ and $\wedge$. Explicitly,

$$\begin{aligned}
a \vee b &= \neg(\neg a \wedge \neg b) \\
a \Rightarrow b &= \neg(a \wedge \neg b) \\
a \ \texttt{XOR}\ b &= \neg(a \wedge \neg b) \wedge \neg(\neg a \wedge b) \\
a \ \texttt{XNOR}\ b &= \neg(a \wedge b) \wedge \neg(\neg a \wedge \neg b) \\
\forall x, \phi(x) &= \phi(x_1) \wedge \cdots \wedge \phi(x_N) \\
\exists x, \phi(x) &= \neg(\neg\phi(x_1) \wedge \cdots \wedge \neg\phi(x_N))
\end{aligned}$$

If we can extend the operators $\wedge$ and $\neg$ to the full domain $[0,1]$, we can therefore do so for all the above operators also. For the remainder of this section, we will specify that $\mathbf{T} \coloneqq 1$, and $\mathbf{F} \coloneqq 0$, though we will later see examples where this is not the case.

2.2.1 Real Conjunction and Negation

We will take a set of properties that apply to classical conjunction $\wedge$ and require that the extension $\wedge : [0,1]^2 \to [0,1]$ has them also. The properties are;

$$\begin{aligned}
\text{(Associativity)}\ & (a \wedge b) \wedge c = a \wedge (b \wedge c) \\
\text{(Commutativity)}\ & a \wedge b = b \wedge a \\
\text{(Monotonicity)}\ & a \leq b, c \leq d \implies a \wedge c \leq b \wedge d \\
\text{(Identity)}\ & \forall a \in [0,1], a \wedge 1 = a
\end{aligned}$$

The above are known as the *T-norm axioms* [18]. Note that they are a strict subset of the axioms required for $([0,1], \wedge, \vee)$ to be a boolean algebra, namely we are missing distributivity and idempotence. In fact, it is possible to construct a real logic that preserves these properties also, but there is precisely one such logic, and as we will see, it has flaws that make it less than ideal for gradient descent. The above axioms are, however, enough for $\wedge$ to have the correct output for the restricted domain $\{0,1\}$.

We can likewise define negation simply by $\neg : x \mapsto 1 - x$. In the fuzzy logic literature, this is known as *strong negation*, as there is an alternate formulation of negation that captures the intention of fuzzy logic more effectively.

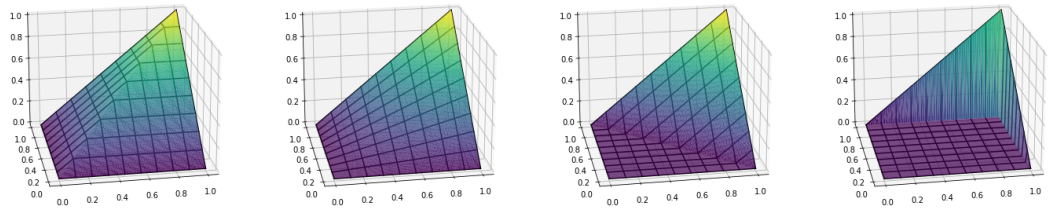
It can be shown that the axioms as we have defined them allow for an infinite family of possible real logics. Some examples are given below;

Logic	$\forall$	$\exists$
Minimum	$\min\{x_1,\dots,x_N\}$	$\max\{x_1,\dots,x_N\}$
Product	$\prod_i x_i$	$1 - \prod_i (1 - x_i)$
Łukasiewicz	$\max\{\sum_i x_i - N + 1, 0\}$	$\min\{\sum_i x_i, 1\}$
Schweizer-Sklar	$\max\{\sum_i x_i^p - N + 1, 0\}^{\frac{1}{p}}$	$\min\{N - \sum_i (1 - x_i)^p, 1\}^{\frac{1}{p}}$
Yager	$\max\{1 - (\sum_i (1 - x_i)^p)^{\frac{1}{p}}, 0\}$	$\min\{(\sum_i x_i^p)^{\frac{1}{p}}, 1\}$

The examples are expressed as conjunctions of N variables, not just 2. p in the table above is a hyperparameter. Figures 2.1 shows more visually how some of these logics behave.

2.3 General Architectures

Now that we have explicit definitions for boolean-like algebras in the domain $[0,1]$, we can begin to formulate how we may learn in this setting.



(a) Minimum (b) Product (c) Łukasiewicz (d) Drastic

Figure 2.1: Conjunction over various fuzzy logics. Note how pointwise gradients differ.

We first need to consider what problems we may want to solve. A basic problem is that of *satisfiability*, i.e., given a boolean function $\phi : \{0,1\}^K \to \{0,1\}$ which can be expressed in first-order logic, is there some element $\mathbf{x} \in \{0,1\}^K$ such that $\phi(\mathbf{x}) = 1$? If so, we want to find an example of said $\mathbf{x}$.

We can consider this as a continuous optimisation problem with $\mathbf{x}$ as a parameter. We can extend ϕ to the real domain $[0,1]$ by extending each operator in ϕ when expressed in the language of first-order logic. Then, by performing SGD as normal, we aim to minimise the loss function $\ell(\mathbf{x}) = \neg\phi(\mathbf{x})$.

To match the power of classical MLPs, we also want to be able to learn optimal *functions* ϕ. Suppose we have a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ where $\forall i, \phi(\mathbf{x}_i) = y_i$ for some formula $\phi : \{0,1\}^K \to \{0,1\}$ which need not have a known representation in classical first-order logic. Further suppose we have a function $\psi : \{0,1\}^{K+P} \to \{0,1\}$ expressed in first-order logic such that for some $\mathbf{w} \in \{0,1\}^P$, $\phi(\mathbf{x}) = \psi(\mathbf{x}, \mathbf{w})$. We consider $\mathbf{w}$ a parameter here, to be optimised. We could use a variety of losses, in the following we examine $\ell(\mathbf{x}, y; \mathbf{w}) = (y \texttt{ XOR } \psi(\mathbf{x}, \mathbf{w}))^a$, for some exponent $a \in [1, \infty)$ (as $\mathbb{E}_{\mathbf{x}}[\ell(\mathbf{x}, \phi(\mathbf{x}); \mathbf{w})] = 0 \iff \psi(\mathbf{x}, \mathbf{w}) = \phi(\mathbf{x})$), and binary cross-entropy, as is done in [32].

It is very feasible to find expressive enough ψ, as it can be shown that every boolean function ϕ can be expressed in a number of standard forms - Disjunctive Normal Form (DNF), and Conjunctive Normal Form (CNF), being two notable ones. To fit these two representations into the framework we have described, we need to be able to express how every variable x_i is represented in each normal form by appending appropriate parameter variables w_i.

In DNF, ϕ is expressed as a disjunction of conjunctions, with each conjunction described by a subset of input variables $\subseteq \{x_1, \ldots, x_K\}$, each variable being optionally negated. We can capture membership by boolean variables $m_i \in \{0,1\}$, and negation by $s_i \in \{0,1\}$. This allows us to represent a formula in DNF by

$$\begin{aligned} \phi(\mathbf{x}) &= \exists j,\ 1 \le j \le W,\ \phi_j(\mathbf{x}) \\ \text{where } \phi_j(\mathbf{x}) &= \forall i,\ m_{ij} \Rightarrow (x_i \texttt{ XOR } s_{ij}) \end{aligned}$$

for some assignment to the $M = (m_{ij}), S = (s_{ij})$. A similar form for CNF is,

$$\begin{aligned} \phi(\mathbf{x}) &= \forall j,\ 1 \le j \le W,\ \phi_j(\mathbf{x}) \\ \text{where } \phi_j(\mathbf{x}) &= \exists i,\ m_{ij} \wedge (x_i \texttt{ XOR } s_{ij}) \end{aligned}$$

In both cases, we can construct a function ψ with M and S as parameters, and optimise over these parameters as before. Here $W \in \mathbb{N}$ is a hyperparameter which specifies the number of conjunctions (or disjunctions) in the function family ψ. Naturally, the larger W is, the more expressive ψ can be, and we know that $W = 2^K$ is enough to capture all possible functions ϕ. This very closely parallels the Universal Approximation Theorem (UAT) of MLPs with one hidden layer. It also very closely mirrors the concept of "reducing induction to satisfiability" seen in many ILP systems [11].

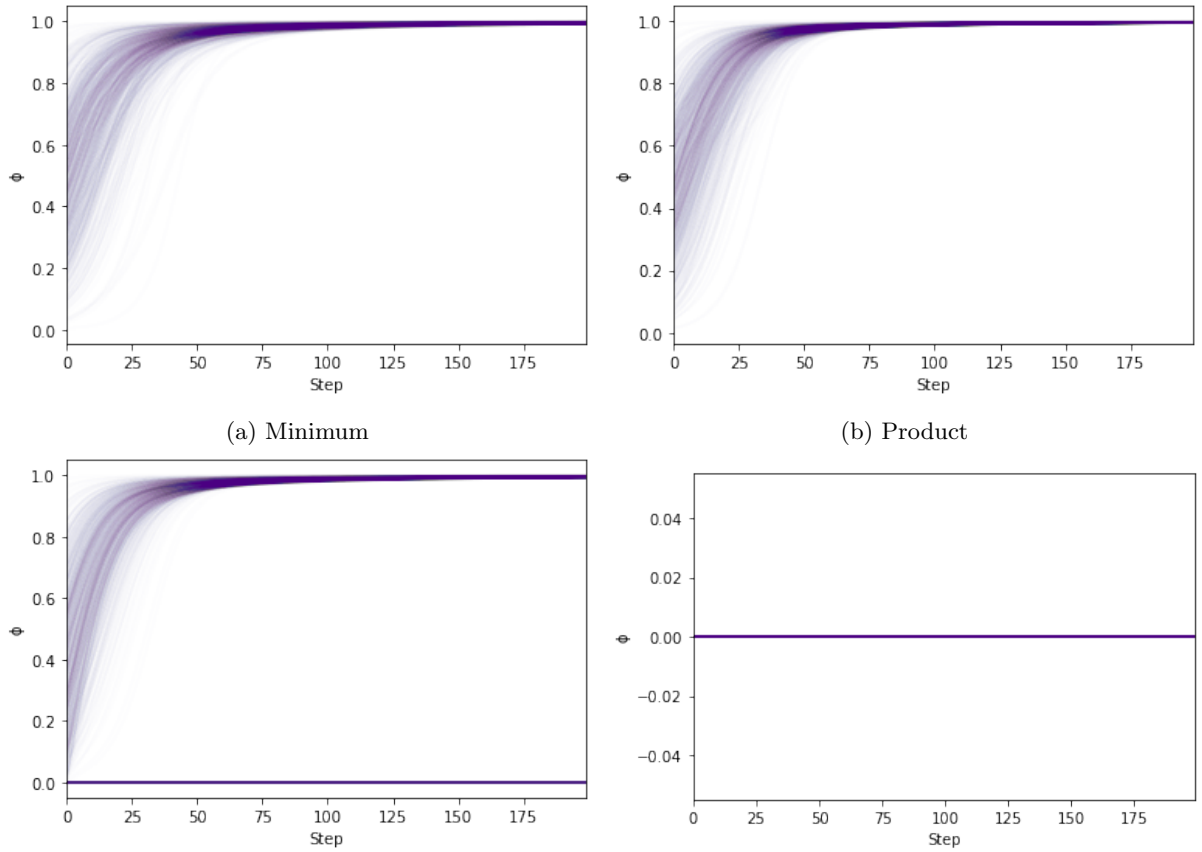
2.4 Analysis of a Toy Problem

The framework we have given is very general, and does not specify what real logic we are required to use. Indeed it does not even require that every operation need be from the same logic, only that they are correct on classical

boolean values $0, 1$. It is valuable, therefore, to compare each logic and analyse which logics are useful for which learning tasks.

In [34], the problem of satisfiability over $\phi(a, b, c) = (a \wedge b) \vee (c \wedge \neg a)$ is considered. We compare the convergence of a, b, c [3] over each logic, given randomly initialised starting parameters.

Figure 2.2 show convergence of the output of ϕ for 1000 uniformly sampled random initialisations of the parameters a, b, c in the interval $(0, 1)$, using Adam optimisation [17] with a learning rate of 10^{-1}. Convergence to 1 represents finding a satisfying assignment, whereas anything else represents a failure to do so.



(a) Minimum

(b) Product

(c) Łukasiewicz

(d) Drastic

Figure 2.2: Convergence of toy problem $\phi(a, b, c) = (a \wedge b) \vee (c \wedge \neg a)$ for various fuzzy logics.

We see that the choice of logic has a profound impact on the convergence. Minimum and Product logic both reliably find a satisfying assignment of ϕ regardless of initialisation. Łukasiewicz logic finds a satisfying assignment in many instances, but very often fails to optimise at all. Drastic logic (somewhat obviously) never finds a satisfying assignment, regardless of initialisation.

2.4.1 Vanishing Gradients

The tests show that a majority of gradient samples for Łukasiewicz logic are precisely 0. By inspection of Figure 2.1c, we can see that there are large regions (i.e. regions of non-zero measure) where the function evaluates only to 0. It is natural that some observations would not allow us to make any meaningful inferences about the optimal values of each variable, especially in cases where the proportion of satisfying inputs is incredibly small. We observe however, that different choices of logic result in different proportions of vanishing gradient observations.

To explain this, we must discuss the nature of conjunction in each logic. In Łukasiewicz logic, for $a, b \in [0, 1]$ such that $a + b - 1 < 0$, the gradient of binary conjunction $\wedge$ is precisely 0. This means that a full $\frac{1}{2}$ of all possible input

[3] In practice, we actually consider satisfiability over parameters $a, b, c \in \mathbb{R}$ mapped into $[0, 1]$ with a sigmoid function, as we want to converge to parameters in the appropriate bounds.

arguments (uniformly distributed) give no meaningful inference. Aggregating over K variables, this generalises to a proportion of $1 - \frac{1}{K!}$ inputs having vanishing gradients, which is very obviously not ideal.

Therefore, in this framework, we prefer to use logics such that $\wedge$ has non-vanishing gradient almost everywhere - these logics are known as *strict*. Minimum and Product logics satisfy this condition, along with certain parameterisations of the Schweizer–Sklar and Yager logics.

2.5 Learning Functions

There are many well known algorithms for efficiently learning classical boolean functions $\phi : \{0,1\}^K \to \{0,1\}$ given prior knowledge of about ϕ restricting it to a given family. Such algorithms often make logical inferences to determine the proper state of boolean parameters $\mathbf{w}$, with complexity guarantees that can be described within the Probably Approximately Correct (PAC) learning framework [16]. We will compare such algorithms to their equivalents in differentiable real logic, and hope for results as good, if not better, given an ideal optimisation regime.

Generalising these methods to real logic would have considerable benefits. For one, given gradient descent is by nature stochastic and (aside from current parameterisation) stateless, these methods would be inherently robust to noisy data and distribution shift in the learning dataset. A possible drawback is that correctness may only be guaranteed for data drawn similarly to the training dataset, introducing a potentially unavoidable bias.

In each test, we also train a traditional MLP model on the same data. Prior to the test, we could make the assumption that the inductive bias introduced by our model could improve the rate of convergence, but an equally convincing sentiment is that this same bias could be too restrictive on the learning process, slowing down convergence.

2.5.1 Learning Conjunctions

The problem of learning conjunctions is a classical one in Computational Learning Theory (CLT). It is well known that determining conjunctions can be done PAC-efficiently [16] and is robust to noisy data [5]. The goal of this comparison, therefore, is simply to prove the viability of real logic for this application.

The classical algorithm relies on one important observation. Suppose ϕ is a conjunction, that is - it is a function of the form

$$\phi(x_1, \ldots, x_K) = y_1 \wedge \cdots \wedge y_N$$

for $y_i \in \{x_1, \ldots, x_K, \neg x_1, \ldots, \neg x_K\}$. If for some j, we have $\neg x_j$ but ϕ returns true, then x_j is not one of the terms y_i. Similarly $x_j \wedge \phi$ implies $\neg x_j$ is not one of the terms. If we begin with ϕ a conjunction over all possible such terms, and remove terms where possible, we are eventually left only with the terms actually present in the conjunction.

As discussed previously, we can model the same thing in real logic by introducing weight and sign parameters $\mathbf{m}$ and $\mathbf{s}$,

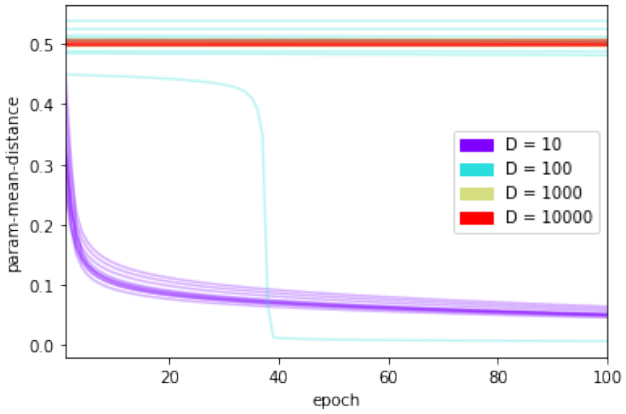
$$\psi(\mathbf{x}; \mathbf{m}, \mathbf{s}) = \forall i,\ 1 \leq i \leq K,\ m_i \Rightarrow (x_i \ \texttt{XOR} \ s_i)$$

and optimising $\mathbf{m}$ and $\mathbf{s}$. In the following, for initial simplicity, we set $\mathbf{s} = 0$ (i.e. we assume no signs $\neg$.)

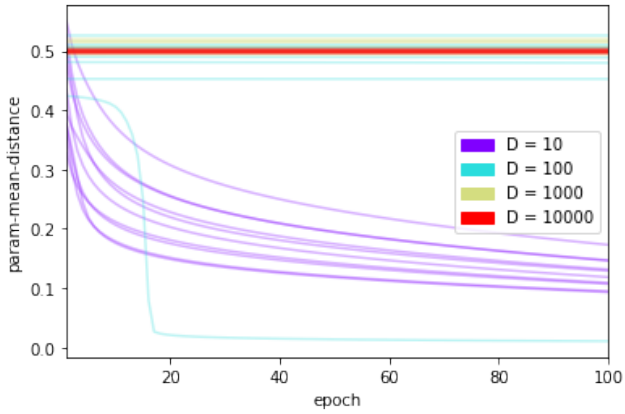
We aim to learn conjunctions by randomly initialising values $\mathbf{m}$, before minimising for some loss function, such as those given in Section 2.3. Data pairs are generated by randomly sampling bits $\{0,1\}^D$, and mapping them through a predefined goal conjunction with N terms. We vary many aspects of this model, including conjunction size D, number of present terms N, choice of fuzzy logic, optimiser, learning rate and loss exponent a.

Some results using this architecture are not promising. Figure 2.3 shows convergence of the average distance of a parameter from it's optimal (i.e. correct) value, with the `XOR` loss $\ell(\mathbf{x}, y; \mathbf{w}) = (\psi(\mathbf{x}; \mathbf{w}) \ \texttt{XOR} \ y)^a$ using an Adam optimiser with a learning rate of 10^{-2}, when learning unsigned conjunctions in the Product logic. Each test is run several times, with the convergence of each run plotted on the graph. Both training and test data are uniformly sampled random bits $\{0,1\}^D$. We see that for low dimensionality, convergence is quite fast, whereas minimal changes occur with dimensionality any higher than 100. This is of course an issue.

We see better results when using binary cross-entropy. Figure 2.4 displays the results from the same test, with differing loss function. We see that dimensionality $D = 100$ now can be learnt, but we still cannot go much higher.

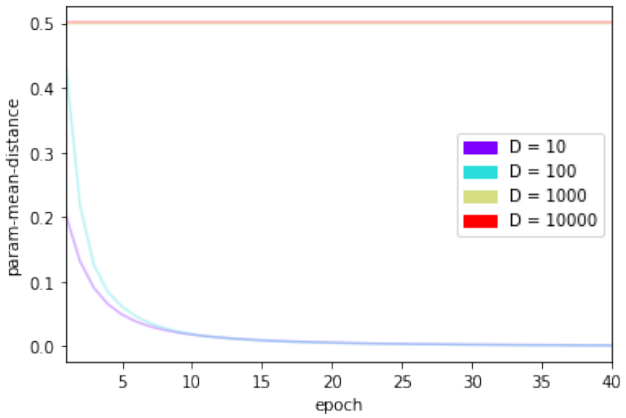


(a) 1-term conjunction

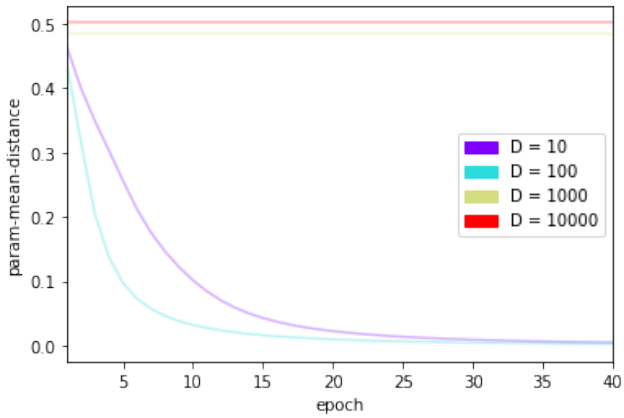


(b) 5-term conjunction

Figure 2.3: Convergence of mean distance from optimal parameters when learning conjunctions in the Product logic. Each test is run 10 times, with `XOR` loss, using exponent $a = 10$.



(a) 1-term conjunction



(b) 5-term conjunction

Figure 2.4: Convergence of mean distance from optimal parameters when learning conjunctions in the Product logic. Each test is run ? times, with exponent binary cross-entropy loss.

2.5.2 Vanishing Gradients in Strict Logics

To determine why this happens, we need only observe the induced gradient of the network. It is well known that a large family of T-norms can be constructed using what is known as an *additive generator*, that is, a function f associated with a given logic, such that $a \wedge b = f^{-1}(\min\{f(a) + f(b), f(0)\})$. This definition allows us to also say $\forall i, x_i = f^{-1}(\min\{\sum_i f(x_i), f(0)\})$. Some additive generators have $f(x) \to \infty$ as $x \to 0$, and in these cases $a \wedge b = f^{-1}(f(a) + f(b))$ for $a, b \neq 0$.

Suppose $\sum_i f(x_i) \leq f(0)$ (which is trivially satisfied if $\lim_{x \to 0} f(x) = \infty$). Then;

$$\begin{aligned} \forall i, x_i &= f^{-1}(\sum_i f(x_i)), \text{ so} \\ \frac{\partial}{\partial x_j} \forall i, x_i &= \frac{f'(x_j)}{f'(f^{-1}(\sum_i f(x_i)))} \\ &= \frac{f'(x_j)}{f'(\forall i, x_i)} \end{aligned}$$

Some examples of additive generators, as well as explicit values for the derivative above, are given in the following table. Note that if $\sum_i f(x_i) > f(0)$, all gradients are vanishing.

Logic	$f(x)$	$\frac{\partial}{\partial x_j}\forall i, x_i$
Product	$-\log x$	$\prod_{i\neq j} x_i$
Łukasiewicz	$1-x$	1
Schweizer-Sklar	$\frac{1-x^p}{p}$	$\left(\frac{x_j}{\forall i, x_i}\right)^{p-1}$
Yager	$(1-x)^p$	$\left(\frac{1-x_j}{1-\forall i, x_i}\right)^{p-1}$

The Minimum logic has no additive generator. From the table, the influence of dimensionality now becomes apparent.

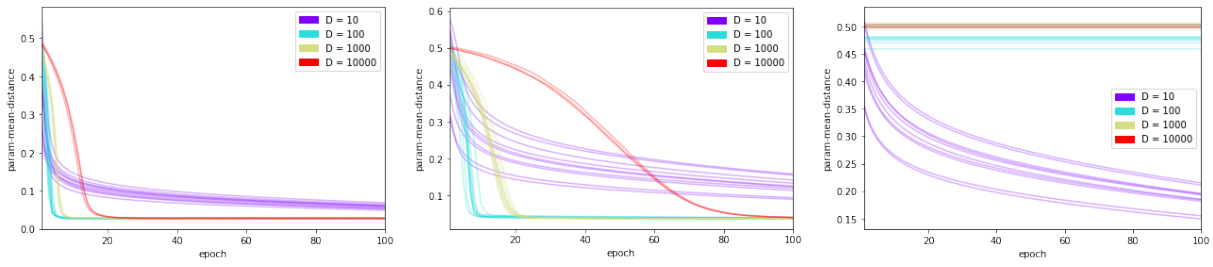
Suppose each aggregate conjunction $\forall i, x_i$ is over D variables. In the product logic, if the x_i are independently uniformly distributed in $[0,1]$, $\mathbb{E}\left[\frac{\partial}{\partial x_j}\forall i, x_i\right] = \mathbb{E}\left[\prod_{i\neq j} x_i\right] = 2^{D-1}$. Thus the total derivative $\nabla_\forall$ has expected magnitude $\mathbb{E}\left[\|\nabla_\forall\|_2\right] \leq \mathbb{E}\left[\sum_j \frac{\partial}{\partial x_j}\forall i, x_i\right] = D2^{D-1}$. With high dimensionality D, the gradients quickly degrade.

Other T-norms do not suffer this same issue. For example, Łukasiewicz logic has a gradient of 1 in every dimension when the function is non-vanishing, however we have shown previously in Section 2.4.1 that the proportion of vanishing inputs is it's own curse of dimensionality, as it is not strict. Minimum logic has constant $\|\nabla_\forall\|_2 = 1$, however we will see later that there are still issues with optimisation. In Appendix **??**, we show that for Schweizer-Sklar logic with parameter p, the magnitude of $\nabla_\forall$ is lower bounded by $D^{\frac{1}{p}-\frac{1}{2}}$. In this section, we use $p=-2$, thus making the lower bound $D^{-1} \leq \|\nabla_\forall\|_2$, which is a vast improvement on an upper bound of $D2^{-D}$ seen in the Product logic. Empirical confirmations are given in Figures **??** and **??** in the Appendix.

2.5.3 Architectural Improvements

To fix the problem described in the previous section, in [34], the authors propose a method whereby only a subset of dimensions are conjoined during training passes. Fixing the size of this subset to a number less than 100 should allow for the mitigation of any dimensionality effects on convergence, as we are only optimising this subset of parameters at any one time. This does result in a biased gradient estimator, as we are ignoring many inputs, but in practice this shows to broaden the space of problems that can be solved using this method. This method can be seen as analogous to the common "dropout" method in traditional neural networks, used to reduce overfitting during training [15].

We will refer to this method as the *subset optimisation*. Results for conjoining only 50 inputs at a time, chosen at random, are shown in Figure 2.5. We see that this extends the viability of this method to much larger dimensionalities than was previously possible, even when using the Product logic, which we know to not be ideal.



(a) 1-term conjunction (b) 5-term conjunction (c) 20-term conjunction

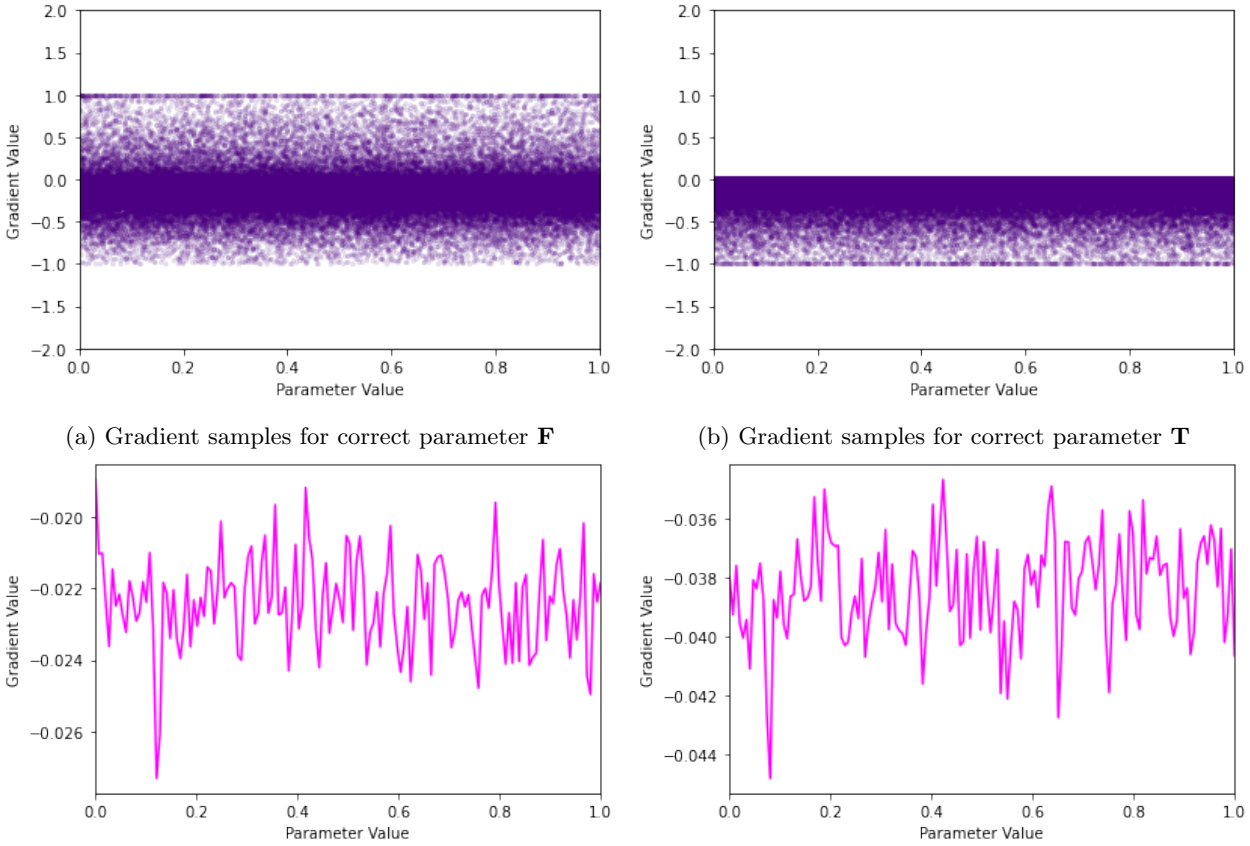
Figure 2.5: Convergence of mean distance from optimal parameters when learning conjunctions in the Product logic, with the size-50 subset optimisation.

2.5.4 Class Imbalance and Parameter Crispness

In Figure 2.5, we notably see that, whilst applying the subset optimisation makes scaling the dimension of a conjunction viable, what is not viable is increasing the size of the number of terms in the conjunction. If a boolean function is a conjunction of N terms, only a fraction of 2^{-N} of all possible inputs will be satisfying. Thus, our learning becomes exponentially worse with increasing N.

We can explicitly sample the gradients to see how this manifests more directly. Suppose we have a proposed conjunction $\phi(\mathbf{x}) = x_1 \wedge x_2 \wedge x_3$, but we learn that $x_1 = \mathbf{F}$ while $\phi = \mathbf{T}$, for some training data point. If ϕ were as we assumed, ϕ would be $\mathbf{F}$ in this case, but it is not. Thus x_1 cannot be one of the terms in ϕ. Applying this more generally is what is used in classical algorithms for learning conjunctions given noise-free data [16].

We would expect to see a similar effect in the gradient estimator that we derive through backpropagation. To test for this, we uniformly sample random initialisations of a real conjunction model, and observe the gradient estimations that result from an entire epoch of training inputs. Figure 2.6 displays the results of this procedure. A conjunction over $D = 10$ variables containing $N = 4$ present terms was used to best convey the issue in the empiric results.



(a) Gradient samples for correct parameter $\mathbf{F}$

(b) Gradient samples for correct parameter $\mathbf{T}$

(c) Mean gradient estimator for correct parameter $\mathbf{F}$

(d) Mean gradient estimator for correct parameter $\mathbf{T}$

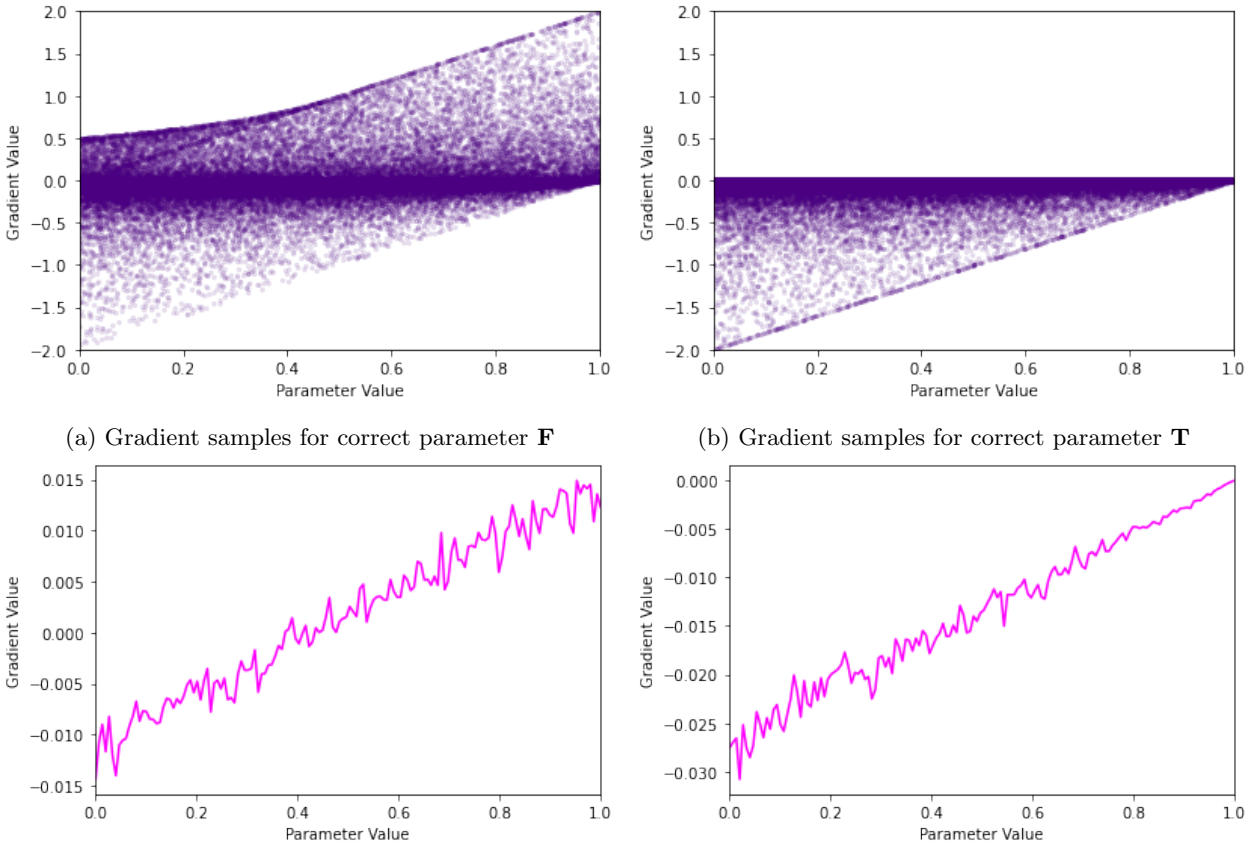
Figure 2.6: Gradient estimations over the problem of real conjunctions with exponent $a = 1$.

We do see that the gradient estimator behaves differently depending on the true valuation of a given logical parameter. If the true value is $\mathbf{F}$, any input sample with a correct output of $\mathbf{T}$ results in a positive gradient estimation, thus decreasing the parameter on the next optimisation step, as can be seen in Figure 2.6a. This is exactly the kind of behaviour we would expect, and it represents a faithful translation of logical inference to the real domain. Whenever the output is $\mathbf{F}$, the model increases all parameters, as no meaningful inference can be made, which can be seen in both Figures 2.6a and 2.6b. This can be interpreted as the model aiming to be robust to noisy data.

The samples above show that it is impossible for terms which are indeed part of the true conjunction to not be represented in the learnt model, as their parameterisations can only increase. The mean gradient estimator in Figure 2.6d reflects this, as it is firmly below 0 no matter the parameterisation. If the term is not in the true conjunction, however, Figure 2.6c shows that we unfortunately see that the gradient estimator is still negative, meaning the term appears in the learnt conjunction, regardless of initial parameterisation.

This is the result of the class imbalance problem. Despite the gradients being in the right direction at every step, the overall result is inaccurate, because inferences are not being well capitalised if they cannot be frequently made.

What is interesting is that increasing the value of the exponent a appears to resolve this issue. In all our successful tests, we use a value of $a = 10$. To display the effects of varying a here, we need only observe $a = 2$.



(a) Gradient samples for correct parameter **F**

(b) Gradient samples for correct parameter **T**

(c) Mean gradient estimator for correct parameter **F**

(d) Mean gradient estimator for correct parameter **T**

Figure 2.7: Gradient estimations over the problem of real conjunctions with exponent $a = 2$.

Figure 2.7 displays sampled gradients when setting the exponent to $a = 2$. We now see that the current value of the parameter has a more noticeable effect on the magnitude of the gradient. The positive gradient samples grow larger as the parameters get larger, and the negative samples grow larger as the parameters get smaller, acting to pushing parameters away from values that are significantly incorrect. We also see that for parameterisations > 0.4, the described inference step overpowers the class imbalance problem.

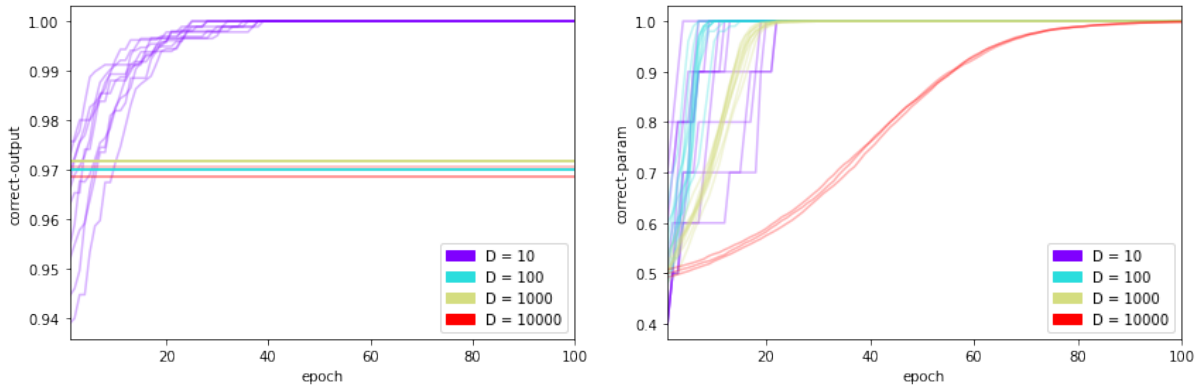
This is not a perfect solution - we still see convergence to parameters around 0.4, so our values are not very "crisp". Crispness refers to the proximity to classical boolean values $\{0, 1\}$. However, we can see how varying the gradient further can allow for more accurate inference. Appendix **??** contains more such examples, that show how varying the exponent, logic and dimension have an effect on the gradient estimator. A general observation is that with larger exponent a, the class imbalance problem becomes less apparent, however, the crispness of parameters continues to suffer, as gradient observations for uncrisp parameters have lower magnitude.

2.5.5 Optimising for Interpretability

An interesting finding is that the model is actually a very inaccurate output predictor, regardless of optimisation. Figure 2.8 shows the convergence of the proportion of outputs and parameters respectively, that when mapped to either 0 or 1 (by closest distance), are the correct classical boolean values. We see that while the proportion of correct outputs converges quite quickly, the proportion of correct outputs suffer. Of note is that, for $N \geq 100$, the proportion of correct outputs does not meaningfully differ through optimisation. The stagnant proportion is also somewhat around $1 - 2^{-5} \approx 0.968$, which suggests that the output only ever evaluates to false, as this 2^{-5} represents the proportion of true outputs for conjunctions of 5 terms.

This is not surprising, as we have just discussed how this model poorly approximates the true crisp parameterisations

$0, 1$ - suppose that we have correct parameterisations for every term, but with a distance of 0.25 from crisp values $0, 1$. In product logic, $0.25 \Rightarrow 0 = 0.75$, whereas in classical logic this would round to $0 \Rightarrow 0 = 1$. taking the product over a number of such terms that increases linearly with N gives us an upper bound of $O(0.75^N)$ for the actual output. Since randomly sampled input bits generally have a proportion of bits with value 0 close to $\frac{N}{2}$, most outputs are guaranteed to be 0.



(a) Proportion of Correct Outputs

(b) Proportion of Correct Parameters

Figure 2.8: Convergence of proportion of correct outputs and parameters, for conjunctions of 5 terms.

We can thus interpret this optimisation regime as optimising not for necessarily for output correctness, but for parameter interpretability. Collapsing the fuzzy booleans to crisp values give us a correct and interpretable solution to the problem.

2.5.6 Empiric Comparison of Logics

In the above tests, we focus only on variations of the architecture using a fixed product logic. We also see that the choice of logic, again, has a profound affect on convergence. Figure 2.9 shows the convergence plots for various metrics over various logics.

The plots show that, as in Figure 2.2, Lukasiewicz logic fails even with the discussed optimisations.

Minimum logic is not far behind - while Figure 2.9a shows the proportion of correct parameters increasing gradually for $D = 100$, this proportion does not improve whatsoever for $D = 10000$. This may be due to the fact that in Minimum logic, only one input variable to a conjunction has non-zero gradient at a time, as can be seen in Figure 2.1a. This restriction does not allow the optimisation regime to traverse the parameter space in an efficient manner. [34] refers to this problem as the *binarization* of gradients. Empirical examples of this binarization can be found in Appendix **??**.

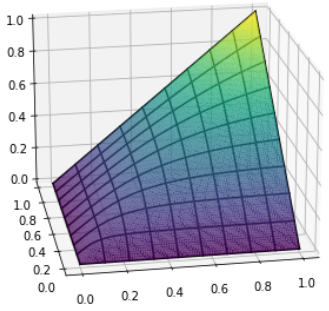
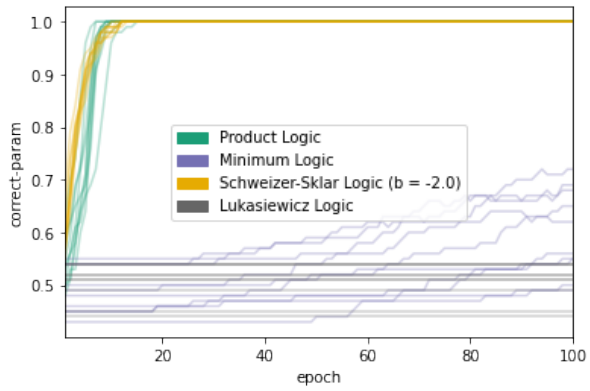


Figure 2.10: Schweizer-Sklar T-norm ($b = -2$)

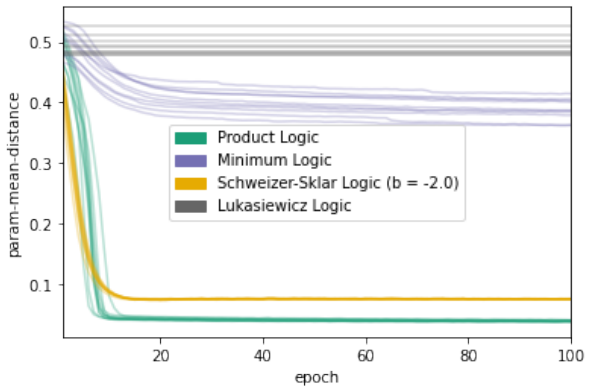
We have already mentioned in Section 2.5.2 that Product logic suffers it's own curse of dimensionality, and we can see this in how it initially has poor convergence in Figure 2.9c. However, as parameters are optimised, the all inputs to the conjunction approach 1, and thus the gradients increase in value. This results in a quickening in convergence rate around 40 epochs, before finally plateauing.

We also see the Schweizer-Sklar logic performs very efficiently, as the bounds derived in Appendix **??** predict. What is interesting to note is that, for $p = 0$, Schweizer-Sklar logic is precisely the Product logic, and for $p = -\infty$, it is precisely the Minimum logic. We can therefore also interpret Schweizer-Sklar (with a parameter $p = -2$ as used in the tests) as being a "middle ground" between the two logics.

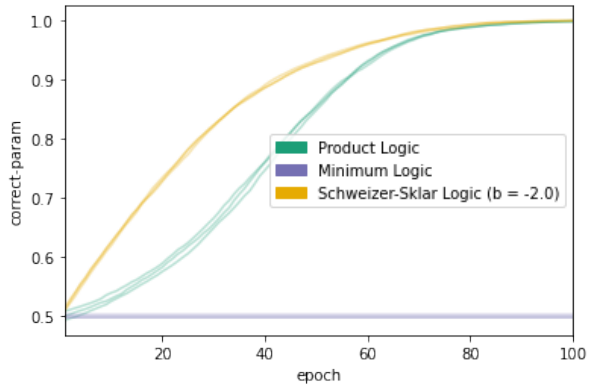
Of note is that, as we can see in Figure 2.9d, Product logic still outperforms Schweizer-Sklar logic in the "crispness" of it's learnt real parameters, as the former converges to an average distance considerably lower than that of the latter.



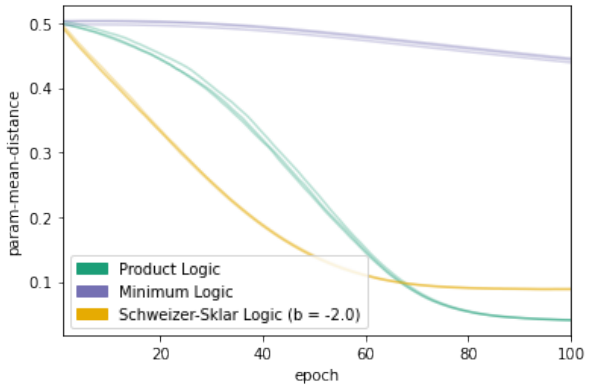
(a) Proportion of correct parameters for $D = 100$



(b) Mean distance of parameters from optimum for $D = 100$



(c) Proportion of correct parameters for $D = 10000$



(d) Mean distance of parameters from optimum for $D = 10000$

Figure 2.9: Convergence over various logics.

2.5.7 Learning Arbitrary Functions

An important result from CLT is that with the assumption that RP $\neq$ NP, it can be shown that learning boolean functions in DNF is not PAC-efficient *in general* [16]. Hopefully, relaxing boolean constraints from the discrete to the real domain allows us to overcome this issue.

Add test results here

Add analysis

It is important to note that

- Suddenly not convex (much like MLP) - Suffers greatly

2.6 Alternative Frameworks

The toy problems above show that the framework we have introduced is not adept at learning boolean functions in general. The appeal instead is that it is simple, and *extensible*. A neurosymbolic layer as we have described can be embedded within a larger model, that may also contain traditional perceptron layers. These mixed models could lend their correctness to traditional NN theory, and their interpretability to their neurosymbolic aspect. In this section, we will discuss other existing neurosymbolic methods, and judge them based on the possibility of use in this application.

One common feature of many of the architectures we have not yet discussed is that logical variables $\mathbf{x}$ explicitly describe some parent object. In the language of our real logic architecture, there exists some universe of objects O such that boolean inputs to ϕ describe an object $o \in O$ in the form of a *unary atom*, e.g. $\textsc{isGreen}(o), \textsc{isBig}(o)$. The

output of ϕ can be interpreted as the valuation of a further unary predicate. Learning ϕ is equivalent to learning some "if and only if" relationship between different properties of the object $o \in O$, if such a relationship exists.

2.6.1 Alternative Applications of Real Logic

As mentioned previously, classical symbolic approaches to AI research involve the use of ILP systems. A formal description of an ILP system is one which can solve problems of the following form. Given some background knowledge B of the world of objects O in the form of a logical statement, and new observations of *positive and negative examples* E^+ and E^-, expressed as conjunctions of *ground literals* (e.g. $\text{isGreen}(o), \neg\text{areFriends}(a, b)$), we aim to find some hypothesis statement h such that $B \wedge h$ satisfies all new observations. Formally, we require *sufficiency* $B \wedge h \vDash E^+$, and *consistency* $B \wedge h \wedge E^- \nvDash \mathbf{F}$ of the constructed h.

Our current real logic framework solves a real relaxation of the ILP problem where E^+ and E^- are all instances of the same unary predicate. There exist differentiable real logic solvers for general ILP problems [11], [26]. Such implementations improve on classical ILP methods as they are robust to noisy data while still being adept at pattern finding.

Other implementations relax real logic further by removing the requirement for all T-norm axioms to be satisfied. [29] introduces *Weighted Non-Linear Logic*, which is conceptually similar to Łukasiewicz logic with the addition of a bias parameter β which is also optimised during learning. The source paper promotes this reformulation as it removes constraints from the problem of optimisation, though the algorithm introduced in the paper as stated cannot be embedded into a mixed model.

2.6.2 Embedded Logic and Relations

Many developments is neurosymbolic learning involve embedding objects into n-dimensional real vector spaces to exploit properties of this space. Word2vec [21] is a model for learning optimal embeddings of natural language atoms in a high-dimensional vector space. Such embeddings were found to preserve relationships between words through vector arithmetic [22] (e.g. Father $-$ Man $+$ Woman $\approx$ Mother). This can be interpreted as a (relaxed) ILP system which learns over the space of *binary* atoms. Learning embeddings of objects $o \in O$ in this manner will aid in extending our current real logic model to mixed models.

In [31], the notion of a *grounding* is introduced - before applying any real logic operators, objects $o \in O$ are mapped into $\mathbb{R}^n$ through some canonical function $\mathcal{G} : O \to \mathbb{R}^n$. An m-ary predicate P is then interpreted as a function $\mathbb{R}^{n \times m} \to [0, 1]$. The value of $\mathcal{G}$ may then be learnt in a manner which best evaluates predicates P for elements of the test dataset.

Further, [13] suggests that not only should objects be embedded into a real vector space, but that boolean values should be as well. In this system, boolean values are encoded over an n-dimensional unit sphere S_{n-1}, and logical operators are required to map the encodings of $\mathbf{T}$ and $\mathbf{F}$ appropriately in this space. Introducing new dimensions may aid in improving model quality, without sacrificing interpretability as activations can be collapsed back into "real logic" through a similarity metric $\text{Sim} : S^2_{n-1} \to [0, 1]$. Cosine similarity (which corresponds to minimum distance travelling along the unit sphere) is generally used. [32] improves upon this by also learning core operators $\wedge, \vee, \neg$ as DNNs, with correctness ensured through regularisation rather than the architecture itself. This model has been used to develop high quality collaborative filtering algorithms [8].

2.6.3 Bayesian Models and Probabilistic Logic

Another approach which could be considered a continuous relaxation of classical logic is in modelling boolean values as distributions over deterministic values $\{\mathbf{T}, \mathbf{F}\}$. Bayesian methods are well studied for this application, with many efficient methods existing for learning the distributions of random variables expressed in terms of a random field. Probabilistic programming languages [12] are able to describe such problems over Bayesian Networks in an highly interpretable manner. Learning probability distributions over the parameter space $\mathbf{w}$ may prove a more interpretable solution, and further may solve some of the convergence issues seen due to many problems being non-convex. If we interpret real booleans $[0, 1]$ as parameters of a Bernoulli distribution over values $\{\mathbf{T}, \mathbf{F}\}$, we can analogize real logic models using the product logic to modelling a random field such that all parameters $\mathbf{w}$ are independent of each other. In this interpretation, we see that using general Bayesian models increases the expressivity of our model by introducing dependencies between parameters $\mathbf{w}$. Arbitrary dependencies can be modelled using Normalizing Flows [27].

Bayesian methods are very general, and many approaches exist which are specialised for uses in logic. Bayesian networks are restrictive as they model variable dependencies as a Directed Acyclic Graph (DAG), whereas many dependencies in logic are cyclic (most notably the relation $\iff$). Markov Networks describe dependencies between random variables as edges in a graph, meaning that any two variables that are not adjacent are conditionally independent. These are used to capture logical inferences in the popular Markov Logic Network (MLN) model [28], which corresponds logical atoms to vertices, and formulae to cliques in the graph. Probabilistic Soft Logic [6] further develops on this by introducing a PPL to describe instances of a model similar to MLNs, which can be used to better interpret the results of learning using such methods.

Chapter 3

Application to Complex Problems

There are some problems that cannot be easily modelled by boolean functions. This chapter will focus on applying the methods we have discussed to problems in image classification. Using the classical example of MNIST [9], an input image representing a handwritten digit from 0 to 9 is mapped to a probability distribution over all possible classifications. Deep learning proved to be very effective at this task [9]. We hope to build an architecture that would be reasonably effective, while incorporating neurosymbolic elements that would allow us to interpret the learnt model.

We have mentioned mixed models many times in the previous chapter, but so far have not introduced an example of such. The language of first-order logic cannot describe boolean functions with multiple outputs. We define some primitive functions which do have multiple outputs, that can be composed to create more complex models. The function

$$c : [0,1]^a \to [0,1]^b;\ \mathbf{x} \mapsto (c_1(\mathbf{x}), \ldots, c_b(\mathbf{x}))$$

represents a series of b conjunctions c_i over $\mathbf{x}$, each learnt independently of each other. We define disjunctions d similarly. ψ will represent some composition of c's and d's (e.g., $d \circ c$ would capture boolean functions as elements of a strict superset of DNF.)

Let m describe a classical MLP network, comprising of an alternating series of linear and non-linear layers, beginning and ending with a linear layer. We can define a mixed model MNIST classifier by

$$M(\mathbf{x}) = \text{softmax} \circ \psi \circ \sigma \circ m(\mathbf{x})$$

$\sigma \circ m_1$ isolates important features from the origin space, and "encodes" their *presence* as boolean values in real logic $[0,1]$. ψ performs some logical operations on the features generated by m_1, and outputs another set of features as elements of the space $[0,1]$. Finally, softmax uses these features to generate a probability distribution over possible categories.

What if ψ is the identity function?

Actual test

Analysis and improvements

Chapter 4

A Chess Architecture

Now that we have shown that real logic methods are adept at learning more complex problems when paired with classical MLP methods in mixed models, we want to begin tackling the problem of learning Chess.

Chess is a famous problem within AI research, often used to demonstrate the effectiveness of new methods in learning adversarial games. Classical methods iteratively descend through a game tree, applying some heuristic evaluation to the state of a game's board when computation resources no longer allow for further descent. These heuristic evaluations are then composed back up using the min-max algorithm to form more refined evaluations of the states immediately following the current, allowing the AI to take the path which it has calculated has the best future result. Stockfish [2] uses such a method with heuristics defined by expert domain knowledge. Forks also exist where the heuristic is a pre-trained neural network with an exceedingly large feature space [24]. State-of-the-art methods [33], [25] instead perform Monte Carlo Tree Search (MCTS), where random paths through game states are traversed, and the excepted result of all paths starting from a given state represents it's evaluation. Paths are weighted based on a heuristic representing the policy of both the current and opposing player, and the learning process involves altering this heuristic to maximise the expected result of the current state.

All the above methods have developed far beyond human capabilities, so any algorithm we derive in this chapter need not attempt to match this quality. The aim is to develop an algorithm which is superior to a substantial proportion of human players, built in a real logic architecture to allow for model interpretation.

4.1 Learning Procedure

We will use a mixed model similar to that of MNIST. The difference in this model will be that rather than treating the output features of ψ as a probability distribution of categories, we will simply take a weighted linear combination of said features. The overall model will look like so;

$$M(\mathbf{x}) = \sigma \circ m_2 \circ \psi \circ \sigma \circ m_1(\mathbf{x})$$

where m_2 has no hidden layers. Chess states will be encoded as a series of bitboards [30], appended with all other relevant information to game state. Notably we do not include a history of moves, which are required to determine any loss by repetition, so this may affect the quality of play in certain scenarios.

The process we use to learn Chess will not involve any kind of tree search. We instead simplify the problem by sampling evaluations from an existing model, namely Stockfish [2], and performing supervised learning over input-output pairs. It is therefore unlikely that our model will outperform Stockfish, but our goal is to relearn the output of Stockfish in an interpretable manner. Input board states are sampled from human games taken from the lichess.org open database [1].

Stockfish evaluations are not elements of $\mathbb{R}$. If the process can determine that with optimal decisions, the current player can guarantee a win (known as "mate-in-N", if the longest winning path one may have to take has length N), then evaluations are given as strings "MN". We map such strings to elements of $\mathbb{R}$ with large absolute value, so that comparisons between board states can be preserved. This proves to be problematic, however, as the distribution of outputs reflects a mixed gaussian distribution with vastly different variances. Any noise in the learnt output may

overpower values from the tighter distribution, resulting in a low signal-to-noise ratio. This is not ideal, as we will measure the effectiveness of our model by it's learnt policy, and decreased SNR may often result in move choices that do not reflect optimal strategies. To resolve this, we map the Stockfish evaluations through a sigmoid layer σ.

4.2 Results

Implement, show loss convergence

Attempt to describe learnt policy

If there is time, connect with Lichess.org and show final ELO

Chapter 5

Conclusions

Bibliography

[1] lichess.org open database. `https://database.lichess.org/`. Accessed: 2022-04-24.

[2] Stockfish - open source chess engine. `https://stockfishchess.org/blog/2022/stockfish-15/`. Accessed: 2022-04-24.

[3] Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E Hinton. Neural additive models: Interpretable machine learning with neural nets. *Advances in Neural Information Processing Systems*, 34, 2021.

[4] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. *Gradient-Based Attribution Methods*, pages 169–191. Springer International Publishing, Cham, 2019.

[5] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[6] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. 2015.

[7] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M. Rao, Troy D. Kelley, Dave Braines, Murat Sensoy, Christopher J. Willis, and Prudhvi Gurram. In *Interpretability of deep learning models: A survey of results*, 2017.

[8] Hanxiong Chen, Shaoyun Shi, Yunqi Li, and Yongfeng Zhang. Neural collaborative reasoning. In *Proceedings of the Web Conference 2021*. ACM, apr 2021.

[9] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[10] Hubert L Dreyfus. *What computers still can't do: A critique of artificial reason*. MIT press, 1992.

[11] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data, 2017.

[12] Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Future of Software Engineering Proceedings*, pages 167–181. 2014.

[13] Ramanathan Guha. Towards a model theory for distributed representations, 2014.

[14] Petr Hájek. *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media, 2013.

[15] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[16] Michael J Kearns and Umesh Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Erich Peter Klement, Radko Mesiar, and Endre Pap. *Triangular norms*, volume 8. Springer Science & Business Media, 2013.

[19] George Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic*, volume 4. Prentice hall New Jersey, 1995.

[20] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming*. 1994.

[21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[22] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013.

[23] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. In *Interpretable Machine Learning – A Brief History, State-of-the-Art and Challenges*. Springer International Publishing, 2020.

[24] Yu Nasu. Efficiently updatable neural-network-based evaluation functions for computer shogi. *The 28th World Computer Shogi Championship Appeal Document*, 2018.

[25] Pascutto, Gian-Carlo and Linscott, Gary. Leela chess zero.

[26] Ali Payani and Faramarz Fekri. Inductive logic programming via differentiable deep neural logic networks, 2019.

[27] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[28] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1):107–136, 2006.

[29] Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, et al. Logical neural networks. *arXiv preprint arXiv:2006.13155*, 2020.

[30] Pablo San Segundo, Ramon Galan, Fernando Matia, Diego Rodriguez-Losada, and Agustin Jimenez. Efficient search using bitboard models. In *2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, pages 132–138, 2006.

[31] Luciano Serafini and Artur d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge, 2016.

[32] Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. Neural logic reasoning, 2020.

[33] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[34] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302:103602, 2022.

[35] Joel Vaughan, Agus Sudjianto, Erind Brahimi, Jie Chen, and Vijayan N. Nair. Explainable neural networks based on additive index models, 2018.

[36] Lotfi A Zadeh. *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers*. World Scientific, 1996.