Designing Interpretable Chess Engines Using Logical Neural Networks

April 23, 2022

Abstract

The problem of interpretability in DNNs is classically a hard one, as the parameterisation of network layers is hard to intuit. This paper uses findings in Logical Neural Network architectures 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 over the past two decades has focused largely into using Neural Network (NN) models to solve an increasingly large breadth 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 derivative 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, 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 a metric for the presence of other concepts. It is difficult, however, to intuit 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 ([18], [2], [5], ...). 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 [2] are the classical example, where the gradient of the model with respect to it's 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) [1]. 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) [26] 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 [15]. 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 [7], 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 [23], [25], and much of the literature focusing on differentiable applications), and *fuzzy logic* (as in most foundational literature, namely [27], [14], [11]). 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* [13]. 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	$\wedge$	$\vee$	$\forall$	$\exists$
Minimum	$\min\{a, b\}$	$\max\{a, b\}$	$\min\{x_1, \dots, x_N\}$	$\max\{x_1, \dots, x_N\}$
Product	ab	$1 - (1 - a)(1 - b)$	$\prod_i x_i$	$1 - \prod_i (1 - x_i)$
Łukasiewicz	$\max\{a + b - 1, 0\}$	$\min\{a + b, 1\}$	$\max\{\sum_i x_i - N + 1, 0\}$	$\min\{\sum_i x_i, 1\}$

Add more logics - Drastic, Nilpotent, Schweizer–Sklar, Hamacher, Yager

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.

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, and optimise for loss $\ell(\mathbf{x}, y; \mathbf{w}) = y \texttt{ XNOR } \psi(\mathbf{x}, \mathbf{w})$. Extending ψ to real logic as standard allows us to use continuous optimisation methods to do so. $\mathbb{E}_{\mathbf{x}}[\ell(\mathbf{x}, \phi(\mathbf{x}); \mathbf{w})] = 0 \iff \psi(\mathbf{x}, \mathbf{w}) = \phi(\mathbf{x})$ for all $\mathbf{x} \in \{0,1\}^K$, so the above algorithm is correct.

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 is represented in each normal form by appending appropriate parameter variables w.

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

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

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

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

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.

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 [25], 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.

The following graphs show the convergence of loss for ? randomly generated initialisations of the parameters a, b, c. Convergence to 0 represents finding a satisfying assignment, whereas anything else represents a failure to do so.

Add reproduction of [25] for many logics

We see that the choice of logic has a very profound impact on the convergence. Add description of results

2.4.1 Incorrect Gradients

It is notable that some initialisations fail to converge at all. This was also observed in [25], and is very much a cause for concern. We would expect all initialisations to work, even if some converge slower than others. To investigate

[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.

further, we explicitly sample observations for the gradient estimator of each parameter, to see if this can help us diagnose the issue.

Add gradient sampling tests

Add gradient sampling analysis - “leaky” gradients

By explicitly deriving the gradients for a given logic, we may begin to diagnose why this may be the case.

Derivation of gradients for product logic

We see that this problem may be resolved by choosing a distribution of inputs that necessarily has the right sign for the expected gradient estimator. This, however, is not ideal. We would like to have a framework which provably converges in *all* cases, regardless of the distribution of inputs.

2.4.2 Vanishing Gradients

The tests further show that a majority of gradient samples are precisely 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. The most apt example is that of Łukasiewicz logic, as for $a, b \in [0, 1]$ such that $a + b - 1 < 0$, the pointwise gradient of binary conjugation $\wedge$ is precisely 0. This means that a full $\frac{1}{2}$ of all possible input 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. Minimum and Product logics satisfy this condition, along with certain parameterisations of the Schweizer–Sklar, Hamacher and Yager logics.

2.4.3 Other Gradient Concerns

Another consideration when comparing different logics is the phenomenon of “partial” vanishing gradients. The minimum logic is the best example of such - as over the entire domain, the partial derivative of conjugation is non-zero for precisely one input. This means that convergence is very *binarized*, meaning only one parameter is optimised at each step. This may also have an effect on convergence, as optimisation potentially only occurs for a small subset of parameters at each step, but this may also help to remedy some of the causes of the “leaky” gradient problem.

2.4.4 Possible Resolutions

SGD vs Adam analysis

2.5 Comparison with Traditional Methods

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 [12]. 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 [12] and is robust to noisy data [3]. 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 $\mathbf{s}$.

Add test results here

Add analysis, e.g. gradient stuff

2.5.2 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* [12]. 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 conjugations of *ground literals* (e.g. $\textsc{isGreen}(o), \neg\textsc{areFriends}(a, b)$), 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 [8], [19]. 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. [22] 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 [16] 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 [17] (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 [23], 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, [10] 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 $\textsc{Sim} : S_{n-1}^2 \to [0, 1]$. Cosine similarity (which corresponds to minimum distance travelling along the unit sphere) is generally used. [24] 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 [6].

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 [9] 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 [20].

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 adjacant are conditionally independent. These are used to capture logical inferences in the popular Markov Logic Network (MLN) model [21], which corresponds logical atoms to vertices, and formulae to cliques in the graph. Probabilistic Soft Logic [4] 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

Chapter 4

A Chess Architecture

Chapter 5

Conclusions

Bibliography

[1] 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.

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

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

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

[5] 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.

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

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

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

[9] 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.

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

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

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

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

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

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

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

[17] 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.

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

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

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

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

[22] 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.

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

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

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

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

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