Designing Interpretable Chess Engines Using Logical Neural Networks

February 2, 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" as an important concept in ML

was introduced. 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 Methods* (VIMs). Gradient-based attribution methods [1] 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) [2]. 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) [3] 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 are the main obstruction to interpreting our models.

1.3 Inspiration from Non-NN Architectures

AI research is not a field that began with the development of NNs. Early research focused mainly on the nature of logical statements - put simply, if we assume facts $P_1, \ldots, P_n$, what further facts can we derive? A simple example is given by; if we know A and $A \to B$ to be true, then B immediately follows.

Given a set of background knowledge facts $P = P_1 \wedge \cdots \wedge P_n$, sets of *positive examples* $E^+ = E_1^+ \wedge \cdots$, and *negative examples* $E^- = \neg E_1^- \wedge \cdots$, it would be useful to determine a hypothesis H such that;

$$\begin{aligned}
\text{(Necessity)}\ & P \not\Longrightarrow E^+ \\
\text{(Sufficiency)}\ & P \wedge H \Longrightarrow E^+ \\
\text{(Weak Consistency)}\ & P \wedge H \not\Longrightarrow \text{False} \\
\text{(Strong Consistency)}\ & P \wedge H \wedge E^- \not\Longrightarrow \text{False}
\end{aligned}$$

Intuitively, the necessity and sufficiency conditions ensure that we can verifiably prove E^+ with H, but not without. Weak consistency forces H to not result in a contradiction (as this would entail every logical statement), and strong consistency requires H not to prove anything in E^- that we assert not to be true.

A program that can compute answers to the above problem is known as an Inductive Logic Programming (ILP) system. ILP systems are very useful - suppose we know of a particular game concept which is beneficial to the player but for which we have no logical representation. However, we may have E^+ as positive examples, and E^- as negative examples. If we can determine an H which satisfies the above conditions, that is, H is a logical statement that is consistent with our set of examples, then we can use this learnt H to describe the concept as a whole, and extrapolate our findings to game states we have not yet seen. We can therefore incorporate H into a heuristic we may use in evaluating a game state, which can go on to be used in more sophisticated algorithms such as the breadth of variants of Min-Max tree search.

A further benefit of ILP systems is that they are by nature interpretable. The hypothesis H is a single logical statement which can easily be read and verified by humans. Therefore if we do manage to devise an ILP system, and corresponding heuristic, we could learn to play a game in a highly interpretable manner. The goal of this paper is to come up with a system similar in nature to this.

Symbolic inference, like the ILP systems described above, formed the bulk of research into AI and Machine Learning from it's early inception in the 1950s up until the 1990s. This is notably no longer the case - ever since the 2000s, Machine Learning as a field has become incredibly publicly prominent through the development of statistical ML methods, and most notably NNs. NNs proved to be much more adept at learning behaviours given smaller amounts of data, where in practice, symbolic machine learning requires an incredibly large amount of data to derive anything meaningful. This lead to famous criticisms, such as

those levelled by Hubert Dreyfus, as to whether the field of symbolic machine learning would be suitable for anything other than simple toy problems.

If we want to leverage the learning power of NNs, and the interpretability of ILP systems, we need to find a way of compromising the two approaches. That is, we want to create an ILP system which learns in the same way as NNs, through backpropagation. There is immediately an obvious issue - logical hypotheses created by ILP systems are *boolean functions*, that is they are mappings $\{0,1\}^n \to \{0,1\}$. Backpropagation relies on computing derivatives of a model described as a function over a continuous domain - which $\{0,1\}^n$ is distinctly not. If we want to begin devising methods of backpropagation over boolean functions, we have to solve this issue.

Chapter 2

Logical Neural Networks

2.1 Introduction

The field of research which focuses on deriving methods of backpropagation over boolean functions is referred to as *Neurosymbolic Machine Learning*. The models that are created to solve such a problem are known as *Logical Neural Networks*[1] (LNNs)[2].

The natural solution to solving the problem of $\{0, 1\}$ being a discrete space, is by embedding values **T**, **F** in a continuous space (or more formally, a topological manifold), and computing derivatives in this space instead. This poses an obvious problem - what if we find that the optimal parameters are values that do not equal **T** or **F**? The ways we may begin to solve this problem differ depending on the particular architecture we decide to use. First, we must provide some background information as to how we may begin to do so in the first place.

2.2 A Useful Logical System

2.2.1 Notation

The notation of linear algebra is very useful in succinctly describing the nature of a classical perceptron model, but within neurosymbolic layers, we do not have this luxury. Further in this paper, we will discuss many different kinds of logical systems, and we will discuss and compare their qualities in learning. To describe our models, it's useful to phrase the layers we construct in a manner similar to linear algebra.

Given a logical system $\mathbb{B}$, we will define a matrix/vector system much like the standard one. A boolean vector of size n is an element of the set $\mathbb{B}^n$, and similarly a boolean matrix of height a and width b is an element of the set

[1] also *Neural Logic Networks*
[2] also NLNs

$\mathbb{B}^{a\times b}$. Where we now differ is in matrix algebra - in linear algebra, matrix multiplication is defined like so;

$$\mathbf{AB}_{ij} = \sum_k \mathbf{A}_{ik} \cdot \mathbf{B}_{kj}$$

We can define boolean matrix multiplication likewise;

$$\mathbf{AB}_{ij} = \bigvee_k \mathbf{A}_{ik} \wedge \mathbf{B}_{kj}$$

This allows us to motivate a meaning for "dot products" also;

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^{\mathrm{T}}\mathbf{b} = \bigvee_i a_i \wedge b_i$$

It would be useful to define element-wise operations on vectors/matrices. For any operator $\circ : \mathbb{B}^2 \to \mathbb{B}$, we write $\mathbf{a} \circ \mathbf{b}$ to mean $(\mathbf{a} \circ \mathbf{b})_i = a_i \circ b_i$. If we want to apply an operator with a constant scalar value to every element of a vector, we can write this likewise, $(\mathbf{a} \circ b)_i = a_i \circ b$, etc.

Finally, if an operator $\circ$ is both associative and commutative, we write $^{\circ}(\mathbf{a})$ to refer to $a_1 \circ a_2 \circ \cdots$.

Using the above two, we can likewise define the "dot product" by $\mathbf{a} \cdot \mathbf{b} = {}^{\vee}(\mathbf{a} \wedge \mathbf{b})$. We can use a similar notation as syntactic sugar to define matrix multiplication $\mathbf{AB} = {}^{\vee}(\mathbf{A} \wedge \mathbf{B})$. This notation allows us to use other operators also, for example standard matrix multiplication would be written by ${}^{+}(\mathbf{A} \times \mathbf{B})$.

2.2.2 Boolean Algebras

While classical perceptron layers heavily rely on linear algebra, neurosymbolic layers cannot. Instead, we must rely on *boolean algebra* to motivate our models. Boolean algebra is fundamentally different to any algebra motivated by classical arithmetic, as we do not have access to the regular operators $+$ or $\times$. Nevertheless, boolean operators behave much like arithmetic ones. A boolean algebra, formally speaking, is a mathematical object $\mathbb{B}$, which contains elements $\mathbf{T}$, $\mathbf{F}$, and for which we have binary operators $\vee$, $\wedge$, and a unary operator $\neg$, such that the following axioms always hold.

$$
\begin{aligned}
\text{(Associativity)}\quad & (a \vee b) \vee c = a \vee (b \vee c) \\
& (a \wedge b) \wedge c = a \wedge (b \wedge c) \\
\text{(Commutativity)}\quad & a \vee b = b \vee a \\
& a \wedge b = b \wedge a \\
\text{(Absorption)}\quad & a \vee (a \wedge b) = a \\
& a \wedge (a \vee b) = a \\
\text{(Identity)}\quad & a \vee \mathbf{F} = a \\
& a \wedge \mathbf{T} = a \\
\text{(Compliments)}\quad & a \vee \neg a = \mathbf{T} \\
& a \wedge \neg a = \mathbf{F} \\
\text{(Distributivity)}\quad & a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \\
& a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)
\end{aligned}
$$

We can immediately see the similarities to classical arithmetic. In fact, the above construct is a commutative ring, with some extra properties.

The above system is very intuitive, as it by design can fully motivate a propositional logic. However, we have not motivated our logical system by direct construction, but by necessitating certain behaviours. We therefore need not restrict ourselves to simply the set $\mathbb{B} = \{0, 1\}$, we can define $\mathbb{B}$ however we'd like, and exploit the unique characteristics of the $\mathbb{B}$ we have chosen in our learning.

Let's consider an alternative boolean algebra - given a set S, let $\mathbb{B} = \{U \subseteq S\}$, that is, the power set of S. It is natural to let $\wedge$ be the intersection of two sets, and $\vee$ the union. We can further say $\mathbf{T} = S$, $\mathbf{F} = \varnothing$, and $\neg : U \mapsto S \setminus U$. One can immediately see that the above axioms follow from basic facts in set theory. This construction is not useful to us, however, as we cannot differentiate over it.

In practice, we'd really like not to stray too far away from classical logic $\mathbb{B} = \{0, 1\}$, and simply approximate classical logic with our choice of $\mathbb{B}$. In this way, our goal is not to necessarily to exploit the properties of a boolean algebra once we have them, but to use the axioms of boolean algebras (and derivations thereof) to measure the quality of hypothetical boolean approximations we propose.

2.2.3 Measuring Crispness

To somehow measure how well our model is behaving like a boolean function, it would be useful to construct a distance metric within our space $\mathbb{B}$. The further away from ideal values $\mathbf{T}$, $\mathbf{F}$ we are, or from basic facts true of all boolean algebras, the worse our model may become to intuit as a logical formula.

There exist well understood ways of doing so already. A metric space X is a set endowed with a *distance metric* $d : X^2 \to R$ with the following properties;

$$\text{(Identity of Indiscernables) } d(a,b) = 0 \iff a = b$$
$$\text{(Symmetry) } d(a,b) = d(b,a)$$
$$\text{(Triangle Inequality) } d(a,b) + d(b,c) \leq d(a,c)$$

For example, over classical booleans $\mathbb{B} = \{0,1\}$, a valid metric is $d(a,b) = a +_2 b$, where $+_2$ is addition in the finite field $\mathbb{F}_2$. This can also be interpreted as the function $\texttt{XOR}(a,b) = (\neg a \wedge b) \vee (a \wedge \neg b)$. For boolean vectors $\{0,1\}^n$, a valid distance metric would be the *Hamming distance* $d(\mathbf{a},\mathbf{b}) = {}^{+}(\mathbf{a} +_2 \mathbf{b})$.

For any boolean algebra $\mathbb{B}$, given d, we can define a *vagueness metric*

$$v_d(a) = \min\{d(a,\mathbf{T}), d(a,\mathbf{F})\}$$

In cases where the choice of d is obvious, we will omit the subscript. We refer to values that have low vagueness as being *crisp*.

2.3 A General Neurosymbolic Architecture

We will see particular implementations of LNNs later. Initially, it is important to discuss some general ideas.

As mentioned previously, the UAT shows us that it is possible to model any boolean function within the framework of MLPs, but we want to restrict the architecture of MLPs such that we could begin to interpret the learnt model as a logical formula, while maintaining full expressiveness. It is well known that any boolean function can be represented in Disjunctive Normal Form (DNF). That is, a formula that takes the form;

$$\begin{aligned}\phi(x_1,\dots,x_N) = &(a_{11} \wedge a_{12} \wedge \cdots \wedge a_{1n_1}) \\ &\vee (a_{21} \wedge a_{22} \wedge \cdots \wedge a_{2n_2}) \\ &\vee \cdots \\ &\vee (a_{m1} \wedge a_{m2} \wedge \cdots \wedge a_{mn_m})\end{aligned}$$

where the $a_{ij} \in \{x_1,\dots,x_N,\neg x_1,\dots,\neg x_N\}$.

This is very convenient, as it means that any architecture we choose to build can learn this highly regular form, instead of some arbitrarily deep tree of logical operators. To transform this into the language of NNs, we want to somehow decompose the above form into a series of "logical layers".

There is a natural homomorphism between subsets $W \subseteq \{1,\cdots,N\}$ and disjunctions $D : \{0,1\}^n \to \{0,1\}$. That is, any disjunction D is fully defined by some set W, where

$$D_W(\mathbf{x}) = \bigvee_{i \in W} x_i$$

.

If we want to learn the disjunction D, it is therefore equivalent to learn the membership of the set S. Let $\mathbf{w} \in \mathbb{B}^N$, with $w_i = \mathbb{1}(i \in S) \in \mathbb{B}$. Then we need

only learn the value of the vector $\mathbf{w}$. It is important to note that, in the space $\mathbb{B} = \{0, 1\}$, we have an equivalent representation

$$\begin{aligned} D_W(\mathbf{x}) &= \left(\bigvee_{i \in W} x_i \right) \vee \left(\bigvee_{i \notin W} \mathbf{F} \right) \\ &= \bigvee_i x_i \wedge w_i \\ &= {}^{\vee}(\mathbf{w} \wedge \mathbf{x}) = \mathbf{w}^T \mathbf{x} \end{aligned}$$

To distinguish this understanding of disjunctions with the simple operation ${}^{\vee}(\mathbf{x})$, we will refer to these as *weighted disjunctions*.

This definitely justifies the notation introduced in the previous section. A similar procedure gives us a compact representation of conjunctions, though it is not as minimal;

$$\begin{aligned} C_W(\mathbf{x}) &= \bigwedge_{i \in W} x_i \\ &= \left(\bigwedge_{i \in W} x_i \right) \wedge \left(\bigwedge_{i \notin W} \mathbf{T} \right) \\ &= \bigwedge_i \neg w_i \vee x_i \\ &= {}^{\wedge}(\mathbf{w} \Rightarrow \mathbf{x}) \end{aligned}$$

The above notation allows us to represent boolean functions in DNF, as any formula in DNF is a disjunction of *weighted conjunctions* as seen above. We write this like so.

$$F(\mathbf{x}; \mathbf{W}) = {}^{\vee}({}^{\wedge}(\mathbf{W} \Rightarrow \mathbf{x} \mathbin{@} \neg \mathbf{x}))$$

Here, @ is an operator representing vector concatenation.

Suppose we are given feature valuations $x_1, \ldots, x_N$, and we are also given whether said features satisfy some set predicate $P(x_1, \ldots, x_N)$. To create a functioning ILP system, it is enough to find some logical formula ϕ which is consistent with this predicate, meaning that for all inputs $\mathbf{x}$ we have received, $\phi(\mathbf{x}) = P(\mathbf{x})$. The purpose of LNNs is to use backpropagation to find an approximation to such a solution. By nature, NNs are "forgetful" in that if the distribution and output value of incoming data changes, they are able to modify their parameterisation accordingly. They therefore aren't guaranteed to be consistent with all input data, but this actually becomes a benefit, rather than a hindrance, when we begin to embed LNN systems within classical NN architectures.

This embedding is important to note, as the aim of the architecture presented in this paper is not to find *correct* concepts, but *useful* ones. We do not necessarily aim to find hypotheses that perfectly represent known concepts, but

concepts that when incorporated into our decision making, allow the player of a game to perform effectively. In that sense, we are using LNNs in a way that is similar in construction, but different in intention to ILPs. This is an important distinction to make when comparing this architecture to existing ones in the space of Neurosymbolic ML, which we will discuss later.

2.3.1 Interpreting a Neurosymbolic Layer

We have constructed layers which represent learnable conjunctions and disjunctions, but how do we go about converting this into a human interpretable format? So far we have discussed an architecture which learns any boolean function in DNF, but is this an ideal way of communicating concepts to humans?

Let's consider what a formula in DNF is actually communicating. Each disjunction is naturally going to represent a single concept, and each conjunction within can be interpreted as being instances of said concept. Using the example of Chess, if we want to learn what it means for a bishop to attack a king, we want to iterate over all instances where a bishop is in a directly diagonal position to a king, with no obstruction. Determining each one of these instances can be done with a conjunction (e.g. Bishop on A1, empty spaces in B2, C3, ..., King on E5), and determining whether any of these conjunctions has occurred is handled by the disjunction.

In this manner, the disjunction is a set of instances of a particular concept, and the conjunctions are the elements of this set.

Is this the best architecture for interpretability? One may consider an architecture with two DNF layers - that is, concepts building on top of further concepts. This can be incredibly useful - suppose we have both a bishop and a knight attacking the king. The presence of both of these concepts, one can imagine, is greater than the sum of it's parts, so it may be useful to consider this a joint concept. We don't necessarily require a second DNF layer to discover this, as we have shown that a single layer is sufficient to fully express all boolean functions, but for the sake of interpretability, it may be advantageous to add this second layer. This is in stark comparison to conventional NNs, where adding layers generally sacrifices interpretability for the sake of expressiveness. This consideration will become important when comparing different implementations of neurosymbolic architectures in practice.

We can now begin to discuss different ways of actually implementing such an architecture.

2.4 Fuzzy Logic

An intuitive, and well researched approach to extending the space of valid boolean values is to consider truth to be "vague", in the sense that something can be "somewhat" true or "somewhat" false. To quantify this, we take boolean values in the closed interval $[0, 1]$, rather than simply $\{0, 1\}$. This approach is

known as *fuzzy logic*. From here, we will denote the traditional boolean algebra by $\mathbb{B}_2$, and fuzzy logic by $\mathbb{B}_\mathrm{f}$.

It is important to note that while this emulates the definition of a probability measure, it is distinctly not. $\mathbb{P}(x) = \frac{1}{2}$ states that x is true with a probability of $\frac{1}{2}$, which can mean either that in $\frac{1}{2}$ of cases, x appears true (the frequentist interpretation), or that we believe that x is true with $\frac{1}{2}$ certainty (the Bayesian interpretation). In fuzzy logic, $x = \frac{1}{2}$ instead states that x is "half-true", with 100% certainty. We are only using fuzzy logic to approximate classical, discrete logic, so it is not important to dwell on the philosophical implications of this. Extending the domain of boolean values means that we have to revisit the definitions of simple logical operators $\neg, \wedge$ and $\vee$.

2.4.1 Fuzzy Operators

We will begin with a generalisation for $\wedge$, as all further definitions follow from this. We want to find a function $\otimes : \mathbb{B}_\mathrm{f}^2 \to \mathbb{B}_\mathrm{f}$ which has the same value as $\wedge$ for $\mathbb{B}_2$, and maintains some natural properties of $\wedge$ also. Suppose we assume the following axioms for $\otimes$;

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

The axioms are actually already enough to ensure that $\otimes|_{\mathbb{B}_2} = \wedge$. However, they are not enough to ensure $\otimes$ takes one particular value in the set $\mathbb{B}_\mathrm{f}^2 \to \mathbb{B}_\mathrm{f}$, in fact there are an infinite family of possible functions $\otimes$. The above axioms are known as the *t-norm axioms*, and the functions which satisfy it are known as *t-norms*. Some examples of t-norms are;

$$\begin{aligned}
\text{(Product t-norm)}\ & a \otimes_\times b \coloneqq ab \\
\text{(Minimum t-norm)}\ & a \otimes_< b \coloneqq \min\{a, b\} \\
\text{(Łukasiewicz t-norm)}\ & a \otimes_\mathrm{L} b \coloneqq \max\{a + b - 1, 0\}
\end{aligned}$$

If we can generalise either $\neg$ or $\vee$, then we can generalise both. A seemingly obvious way to generalise $\neg$ is by simply declaring that $\neg x = (1 - x)$. This immediately satisfies the definition of $\neg$ in classical logic, and comes with some useful properties;

$$\begin{aligned}
\text{(Self-Invertibility)}\ & \neg(\neg a) = a \\
\text{(Monotonicity)}\ & a \leq b \implies \neg a \geq \neg b
\end{aligned}$$

From this, we can fully define generalisations for $\vee$, which we call *t-conorms*. In classical logic, we can appeal to *De Morgan's laws*, in that $a \vee b = \neg(\neg a \wedge \neg b)$. Similarly, we can say that $a \oplus b = \neg(\neg a \otimes \neg b) = 1 - (1 - a) \otimes (1 - b)$, given a particular choice of t-norm $\otimes$. We therefore have;

$$
\begin{aligned}
\text{(Product t-conorm) } a \oplus_{\times} b &\coloneqq 1 - (1-a)(1-b) \\
&= a + b - ab \\
\text{(Minimum t-conorm) } a \oplus_{<} b &\coloneqq 1 - \min\{1-a, 1-b\} \\
&= \max\{a, b\} \\
\text{(Łukasiewicz t-conorm) } a \oplus_{\mathrm{L}} b &\coloneqq 1 - \max\{(1-a) + (1-b) - 1, 0\} \\
&= \min\{a + b, 1\}
\end{aligned}
$$

Given the t-norm axioms, we immediately have some convenient properties of t-conorms also.

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

The only difference here being that the identity has value 0, rather than 1. From this, we can define every logical formula using fuzzy logic operators, allowing us to motivate an architecture for a fuzzy NN.

In the field of fuzzy logic, the above method of constructing fuzzy operators is not actually the preferred way of doing so. Fuzzy logicians instead choose to define the *residuum* $\Rightarrow$ in terms of fuzzy conjunction $\otimes$. The residuum, as the symbol suggests, is meant to generalise the implication operator $(a \Rightarrow b) = \neg(a \wedge \neg b)$. The reason this can be done only relying on the existence of the t-norm (rather than t-norm + fuzzy negation) is because it can be proven that the residuum is the *only* function that satisfies the conditions

$$
a \otimes b \leq c \iff a \leq (b \Rightarrow c)
$$

The justification for this fact comes from the understanding of fuzzy logic as measuring "confidence" - if we have confidence valuations for $a, a \Rightarrow b \in \mathbb{B}_f$, and we know that $a, a \Rightarrow b$ entails b in classical logic, then we would expect to be at least $a \otimes (a \Rightarrow b)$ confident in b, i.e. that $a \otimes (a \Rightarrow b) \leq b$. A similar construction results in the conclusion above, and allows us to uniquely determine $\Rightarrow$. From here, we can choose to define the other logical operators like so;

$$
\begin{aligned}
\neg a &= (a \Rightarrow 0) \\
a \oplus b &= (\neg a \Rightarrow b)
\end{aligned}
$$

Again, we will not dwell on this, as we are aiming to construct fuzzy operators specifically to approximate classical logical formulae in DNF. The definitions for negation, conjunction and disjunction given above are more than enough to begin doing so. In the notation introduced previously, we could write a formula for a fuzzy DNF layer like so.

$$
F(\mathbf{x}, \mathbf{W}) = {}^{\oplus}({}^{\otimes}(\mathbf{W} \Rightarrow \mathbf{x} \mathbin{@} \neg\mathbf{x}))
$$

2.4.2 Constructing Fuzzy Operators

So far we have declared the existance of operators satisfying the given axioms, but we have not discussed how we might discover new ones. It is important to be able to do so freely, as the convergence rate of an algorithm may have a strong relation to the gradient of the chosen operator. To give an example, an operator known as the *drastic t-norm* is 0 anywhere it doesn't have to be anything else, i.e.

$$a \otimes_{\mathrm{D}} b = \begin{cases} a & \text{if } b = 1 \\ b & \text{if } a = 1 \\ 0 & \text{otherwise} \end{cases}$$

The gradient of $\otimes_{\mathrm{D}}$ is 0 almost everywhere, which isn't terribly ideal for gradient descent. We want to be able to avoid cases like this, and to do so it would beneficial to come up with a method to construct t-norms reliably. Suppose we had a decreasing function $f : \mathbb{B}_{\mathrm{f}} \to [0, \infty]$ with $f(1) = 0$. Then the function $T(a, b) = f^{-1}(\min\{f(0), f(a) + f(b)\})$ is a t-norm - in fact, most common t-norms can be constructed in this way. This statement omits some details, but for our use cases this is fine.

A function f which begets a t-norm in this manner is called an *additive generator* of the t-norm. Some examples are;

$$\begin{aligned} \text{(Product t-norm)}\ & f(x) = -\log(x) \\ \text{(Łukasiewicz t-norm)}\ & f(x) = 1 - x \\ \text{(Drastic t-norm)}\ & f(x) = 2 - x \text{ for } x \in [0, 1), f(1) = 0 \end{aligned}$$

This characterisation of t-norms also allows us to explicitly define conjunctions over many variables - if $a \otimes b = f^{-1}(\min\{f(0), f(a) + f(b)\})$, then ${}^{\otimes}(\mathbf{x}) = f^{-1}(\min\{f(0), {}^{+}(f(\mathbf{x}))\})$. Thus,

$$\begin{aligned} {}^{\otimes_{\times}}(\mathbf{x}) &= {}^{\times}(\mathbf{x}) \\ {}^{\otimes_{\mathrm{L}}}(\mathbf{x}) &= \max\{0, 1 + {}^{+}(\mathbf{x} - 1)\} \end{aligned}$$

and so on. In the construction of t-norms from additive generators, we have to add a min operation, as the sum $f(a) + f(b)$ may be out of the range of f. If there exist $a, b \neq 0$ such that this is the case, we refer to the corresponding t-norm as *nilpotent*, since $a \otimes b = 0$. More accurately, this name arises due to the fact that for such $a \neq 0$, there exists $n \in \mathbb{N}$ such that $\underbrace{a \otimes \cdots \otimes a}_{n \text{ times}} = 0$.

Examples of nilpotent t-norms are the Łukasiewicz t-norm $\otimes_{\mathrm{L}}$, and the drastic t-norm $\otimes_{\mathrm{D}}$. In fact, any t-norm whose additive generator is continuous around 0, and doesnt cover the full possible range $[0, \infty]$, is nilpotent in this manner. Any nilpotent t-norm has a non-negligible region of zero gradient, so this will help inform our future discussion on constructed t-norms.

2.4.3 Defuzzification and Interpretation

Suppose we have embedded a fuzzy LNN layer into our larger neural network. How do we go about interpreting the parameterisation of the function? We have already discussed how we may approach doing so for classical DNFs, but to achieve anything at all, we would need to be able to do so for fuzzy DNFs as well.

We can avoid this issue by ensuring that we learn parameters that are as close to $\mathbb{B}_2$ as possible. To do so, we may consider applying some regularisation term to our loss function, which negatively weights parameter values that are not very crisp. We could then add a regularisation term $\lambda \sum_w v(w)$ over all parameters w, and use cross-validation to find an optimal λ for learning.

In practice, this hinders learning quite a bit. Much of the strength of fuzzy logic architectures are a direct result of maintaining vagueness - if we are close to values 0, 1, it can take a lot of data to modify this.

A more effective approach in practice seems much more naive - that is to simply clamp the value of the parameters w into the range $[0, 1]$ at every optimisation step. This allows the model to maintain a level of vagueness where it cannot learn much information, i.e. not "jumping to conclusions". When it actually can decide the nature of a concept, it can freely do so. We may then choose to terminate our learning when the parameters are sufficiently close to values $\{0, 1\}$.

In future sections, we see that using the distance metric d *can* be quite effective in regularisation, but not for directly ensuring crispness - we will use the function v to evaluate the interpretability of our chosen models, but we will not actively optimise for interpretability, we want the models to passively approach crisp parameterisations.

2.5 Parameterised Logic

So far, we have discussed learning the membership of conjunctions and disjunctions only, while keeping the actual definitions of operators constant. We have discussed the fact that some choices of operators may be more successful in learning through backpropagation than others, with the case of the drastic t-norm $\otimes_{\mathrm{D}}$ being a particularly bad choice. How may we find the *most successful* one?

It's difficult to quantify what this means, so we will not attempt to do so. Instead, we can find some way of defining parameterised *families* of logical systems, and learn the optimals parameters of these families in the same way we learn the parameters of the formulae we are trying to learn.

2.5.1 Parameterised T-Norms

In the previous section, we discussed an easy way to construct new t-norms, so it would make sense to use this technique here. We would like to find a family of t-norms that are all parameterisations of a more general construction. We can

begin by parameterising additive generators, and constructing their respective t-norms likewise.

Let $f_p(x) = \frac{1-x^p}{p}$. This satisfies all the criteria for an additive generator, since $f_p'(x) = -x^{p-1} \leq 0$, and $f_p(1) = 0$. For $p = 0$, this is not well defined, but taking the limit as p approaches 0 allows us to extend the function further. In fact this limit is well known - we can say that $f_0(x) = -\log x$, which is actually the additive generator for the product t-norm $\otimes_\times$. What is the resultant t-norm in general?

Since $f_p^{-1}(x) = (1 - px)^{\frac{1}{p}}$, we have $a \otimes_p b = \max\{0, (x^p + y^p - 1)^{\frac{1}{p}}\}$, and more generally, $^{\otimes_p}(\mathbf{x}) = \max\{0, (1 - {}^+((1 - \mathbf{x})^p))^{\frac{1}{p}}\}$. Of note is that, for $p = 1$, this is the Łukasiewicz t-norm $\otimes_\mathrm{L}$. It seems as though many t-norms we have already encountered are members of this family. Indeed, if we further extend the definition we have to $-\infty, \infty$ via continuity, we discover that $\otimes_{-\infty}$ is the minimum t-norm, and $\otimes_\infty$ is the drastic t-norm.

This is wonderful news, as it means with one parametric family of t-norms, we can capture all the examples we might like to study, and then some. These operators are known as the *Schweizer–Sklar t-norms*. Some more parameterised families of t-norms are given by the following additive generators;

$$\begin{aligned}
\text{(Yager t-norms) } & f(x) = (1 - x)^p \\
\text{(Aczél–Alsina t-norms) } & f(x) = (-\log x)^p \\
\text{(Hamacher t-norms) } & f(x) = \log \frac{p + (1 - p)x}{x}
\end{aligned}$$

2.5.2 Biased Logic

There are many properties of classical logical operators that cannot necessarily be captured by fuzzy operators. *Idempotence* is a resultant property of both $\wedge$ and $\vee$ in boolean algebras, which specifies that for all $a, b \in \mathbb{B}_2$, $(a \wedge b) \wedge b = (a \wedge b)$, and likewise for $\vee$. For most t-norms, this is specifically not the case - an obvious example is the product t-norm, where $abb = ab \iff b = 1$. It would be nice to preserve idempotence, but this is not really required. What if we can sacrifice other properties of boolean operators as well?

We will introduce a new parameterised family of boolean operators, called *weighted non-linear logic* (WNL). WNL is a generalisation of Łukasiewicz logic which introduces a bias term β, which mirrors the bias used in perceptron layers of classical NN architectures.

We will define conjunctions and disjunctions like so

2.5.3 Boolean Regularisation

2.5.4 Measuring Similarity

2.6 General Boolean Embeddings

So far we have met Fuzzy Logic, which fundamentally reinvents the common perceptron architecture seen in most NNs. What if we want to maintain an architecture which is more conventional in it's approach, while still allowing for boolean interpretations? This seems difficult at first - although we can approximate any boolean function, determining the nature of this function is difficult in conventional architectures, as we have seen.

Rather than modifying the architecture of the network, we can instead regularise an existing network architecture to act like a boolean function, while making sure that the regularisation somehow allows us to easily interpret the network's parameters. This sounds rather involved - how might we begin to approach this?

2.6.1 Booleans as Vectors

We have discussed the idea of embedding values representing **T** and **F** into a continuous space $\mathbb{B}$, and then differentiating over the space in backpropagation. In fuzzy logic, we specifically fix the set $\mathbb{B}_\mathrm{f} \coloneqq [0,1]$ to act as our continuous space, and exploit this by using continuous functions which are guaranteed to behave like logical operators. What if we instead allow for the full range of parameters $\mathbb{R}$, how may we begin to interpret this as a boolean function?

We can take inspiration from other architectures where feature embeddings are common. In natural language processing, word embeddings are maps which take words in a dictionary, and map them into a finite-dimensional vector space. The position the word is mapped to in this vector space can be used to characterise the meaning of the word, and capture relations between words. We could likewise map the boolean values **T**, **F** into a vector space $\mathbb{B}$, and measure the "crispness" of a learnt function by how similar certain values are to the boolean embeddings in this vector space.

We can define some sort of "distance metric" over $\mathbb{B}$, a convex, commutative function $d : \mathbb{B}^2 \to \mathbb{R}$ such that $d(\mathbf{a}, \mathbf{a}) = 0$ for all $\mathbf{a} \in \mathbb{B}$. This can be a formal distance metric in the topological sense, but it need not be. With this, we can say that the more distant a value $\mathbf{x} \in \mathbb{B}$ is from both **T** and **F**, the less crisp. A particular example which is effective in practice is given later in this section.

2.6.2 Learning a Boolean Operator

Suppose we want to learn the function $\texttt{XOR}(a, b) = (a \wedge \neg b) \vee (\neg a \wedge b)$. The model we will use to approximate this will be a NN with a single hidden layer, as it is well known that this cannot be done by a simple single-layer perceptron model.

The model will therefore map values between three spaces, which we will call $\mathbb{B}^2 \to X \to \mathbb{B}$, for some $\mathbb{B}, X$, which we will not yet specify.

From here, we can proceed as usual and simply backpropagate with correct input-output pairs to learn an appropriate parameterisation for `XOR`. However, we can somewhat cheat, because we know the appropriate behaviours of the function we're trying to learn, we only want to learn it over our arbitrary boolean embedding $\mathbb{B}$. We can exploit known properties of `XOR` to "push along" our learning. For instance, we know that;

$$\begin{aligned}
\text{(Associativity)}\ & \texttt{XOR}(\texttt{XOR}(\mathbf{a}, \mathbf{b}), \mathbf{c}) = \texttt{XOR}(\mathbf{a}, \texttt{XOR}(\mathbf{b}, \mathbf{c})) \\
\text{(Commutativity)}\ & \texttt{XOR}(\mathbf{a}, \mathbf{b}) = \texttt{XOR}(\mathbf{b}, \mathbf{a}) \\
\text{(Identity)}\ & \forall \mathbf{a} \in \mathbb{B}, \texttt{XOR}(\mathbf{a}, \mathbf{F}) = \mathbf{a} \\
\text{(Negation)}\ & \forall \mathbf{a} \in \mathbb{B}, \texttt{XOR}(\mathbf{a}, \mathbf{T}) = \neg \mathbf{a}
\end{aligned}$$

The meaning of $\neg\mathbf{a}$ here will be explained the next section. We will define a "`XOR`-iness loss",

$$\ell_{\texttt{XOR}}(\mathbf{x}, \mathbf{y}) = d(\texttt{XOR}(\mathbf{x}, \mathbf{y}), \texttt{XOR}(\mathbf{y}, \mathbf{x})) + d(\texttt{XOR}(\mathbf{x}, \mathbf{F}), \mathbf{x}) + d(\texttt{XOR}(\mathbf{x}, \mathbf{T}), \neg\mathbf{x})$$

We can see that a perfect parameterisation of `XOR` would minimise this loss. Thus, a complete loss function would look like so,

$$\ell(\mathbf{x}, \mathbf{y}, \mathbf{z}) = d(\texttt{XOR}(\mathbf{x}, \mathbf{y}), \mathbf{z}) + \lambda_{\texttt{XOR}} \ell_{\texttt{XOR}}(\mathbf{x}, \mathbf{y})$$

The parameters of `XOR` are hidden in this representation, as they need not be of any particular architecture.

2.6.3 Measuring Similarity

2.7 Overview of Different Architectures

Chapter 3

LNNs in Practice

Chapter 4

A Chess Architecture

Chapter 5

Interpreting Chess

Chapter 6

Conclusions

Bibliography

[1] Ancona M., Ceolini E., Öztireli C., Gross M. (2019) Gradient-Based Attribution Methods. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning.* Lecture Notes in Computer Science, vol 11700. Springer, Cham. https://doi.org/10.1007/978-3-030-28954-6_9

[2] R. Agarwal, L. Melnick, N. Frosst, X. Zhang, B. Lengerich, R. Caruana, G. Hinton. Neural Additive Models: Interpretable Machine Learning with Neural Nets. arXiv:2004.13912

[3] J. Vaughan, A. Sudjianto, E. Brahimi, J. Chen, V. Nair. Explainable Neural Networks based on Additive Index Models. arXiv:1806.01933