Chapter 1

Boolean Connectives and Deep Learning: Three Interpretations

Miguel-Angel Mendez Lucero, Artificial Intelligence and its Applications Institute, School of Informatics, The University of Edinburgh, Edinburgh, United Kingdom

Vaishak Belle, Artificial Intelligence and its Applications Institute, School of Informatics, The University of Edinburgh , Edinburgh, United Kingdom & Turing Institute, London, United Kingdom

1.1. Introduction

In artificial intelligence, there is an inherent problem of integrating low-level perception and high-level reasoning. If solved, it has the potential to not only bring the areas of symbolic logic and learning closer together, but also provide potential benefits, such as interpretability, reasoning, safety, tractability, and data efficiency, among others. In this chapter we attempt to show several advantages of integrating high-level reasoning in the low-level perception learning process. That is, we will conduct a generalized overview of the different approaches of integrating symbolic logic to enhance deep learning methods. Then, we will concretely discuss three approaches [16, 23, 36] that are representative of these general strands. This chapter's material is drawn from the discussions and contributions of these methodologies.

We will begin this chapter by discussing the type of approaches that enhance deep learning methods to perform reasoning. Deep learning methods are widely known to dominate perception-based learning, including the domains of vision [26], speech [30], and linguistic grammar [40]. Due to this, domain-specific approaches have been developed as a way of performing symbolic reasoning [35, 13]. To illustrate the advantages of these approaches, this chapter will focus on a method which integrates logic reasoning in a visual setting [16]. In this method, we look at whether the hidden layers of neural networks can be used to represent and reason about Boolean functions via so called tractable circuits [28]. In particular, we consider Variational Autoencoders (VAEs) [27, 42] that are widely used for dimensionality reduction. With this method we were able to define a symbolic generative framework, which demonstrates the ability to perform logic reasoning using the feature layers of our deep learning structure. In this context we are able to learn relations and functions over multiple images at a time, as well as handle noisy labels.

Next, we will show how symbolic logic can be used to allow domain experts to encode prior knowledge in neural network training. Encoding expert knowledge is important for various reasons, one being enforcing safety constraints on critical systems [3, 21]. In this setting, machine learning solutions must ensure that they work within specific symbolic domain constraints defined by experts. As an example in a manufacturing learning setting, we may wish to encode that an actuator for a robotic arm does not exceed some threshold (e.g., causing the arm to move at a hazardous speed). In our contribution, called MultiplexNet [23], we provide a logic framework that is able to guarantee that prior knowledge is always satisfied during training. Our methodology represents domain knowledge as a logical formula in disjunctive normal form (DNF), which is easy to encode and to elicit from human experts. Then it compiles the formula into a differentiable circuit with a categorical latent variable which guarantees that the formula is always satisfied when applied with a neural architecture. Lastly, the learning process jointly trains the neural network and the categorical latent variable to choose over the valid space imposed by the domain knowledge. This allows for a flavor of logical reasoning over the data.

Finally, we will provide a methodology for learning functional representations of propositional formulas relying solely on deep learning methods. We start by proposing a neuron [36] based on Minsky's parallel machines [38]. This new perceptron is defined using linear combinations of analytic sinusoids to form an analytic signal representation of the function that we want to learn. The importance of this reformulated parallel mechanism is that *it can learn with a single layer any k-ary Boolean function*. This result was used as proof that there are perceptron architectures that can solve the long-standing linear separability problem [38]. In addition we show that compared to other perceptrons, and even multilayer architectures, this new perceptron achieves competitive spatial and computational complexity for logic and visual perception tasks.

1.2. Literature Survey

As mentioned in Section 1.1, this chapter will survey some key strategies that integrate symbolic logic and deep learning. In the following section, we will cover related literature and provide a more in-depth discussion of our specific contributions.

1.2.1. Logical interpretation for autoencoders

In the first part of this chapter, we introduce a novel method for performing visual reasoning considering a logical interpretation of variational autoencoders [16]. More specifically, we will show that we can re-purpose these autoencoders in a way that can perform probabilistic reasoning over the learned latent space of these mechanisms [19, 27, 42]. In addition, our methodology allows for a visualization of the learned relations by conditional sampling over the latent space. In that regard, our methodology can be thought of as a symbolic-generative framework.

Within the deep learning literature there exists multiple frameworks focused on visualizing, inspecting, and interpreting neural networks (NNs) [45, 46]. Usually such methods are used to decode regions of interest in a specific layer of a pre-trained network. In contrast, our proposed logical approach uses a symbolic generative framework to interpret NNs. Note that we do not infer the meaning of individual variables, although they can be visualized, we rather calculate their conditional probabilities.

As a symbolic logic framework our approach is based on a special case of probabilistic logical models [10, 18]. Tractable probabilistic models have emerged as an extension to data structures such as binary decision diagrams (BDDs). Our approach is based on probabilistic sentential decision diagrams (PSDDs) [28], which are a complete and canonical representation of a probabilistic distribution defined over the models of a propositional theory.

Note that recent circuit models have also attempted to solve vision problems (for example, [41, 17, 33]), and so it is possible to realize the entire image classification pipeline using these tractable probabilistic models. However, such methods solve vision problems without relying on deep learning techniques. As stated above, our approach repurposes state-of-the-art deep learning architectures by adding a symbolic framework rather than replacing it.

1.2.2. Multiplex Networks

In the second part of this chapter, we will discuss a methodology to enforce domain knowledge constraints into the deep learning model's training process [23]. This has several benefits, such as enforcing critical constraints and data efficiency. Within the deep learning literature, some solutions encode the relevant domain knowledge into a network's architecture by relying on non-trivial and/or domain-specific engineering [19]. An alternative approach is to express domain knowledge as logical constraints that can then be used to train neural networks [**?**, 14, 1]. These approaches compile the constraints into the loss function of the network by quantifying the extent to which the output of the network violates the constraints. This is appealing, as logical constraints are usually easy to elicit from experts [23, 35, 14]. However, the solution outputted by the network is designed to minimize the loss function — which combines both data and constraints — rather than to guarantee the satisfaction of the domain constraints. Thus, representing constraints in the loss function is not suitable for safety critical domains where 100% constraint satisfaction is desirable. Taking this into consideration, our work introduces a methodology to compile domain knowledge constraints directly into the output of NNs [23].This allows us to guarantee that any network output from the MultiplexNet will satisfy the domain constraints, which is crucial in safety-critical domains [3, 21].

Similarly to our line of work, there are several methods that guarantee satisfaction of logic constraints [35, 15, 24], but these come with expressive and training trade-offs when used in conjunction with deep learning techniques. For example in [24], there is no means to express rules that constrain real-valued outputs as seen in satisfiability modulo theories (SMT), like in [31]. Xsat [15] overcomes this problem, but it's methodology for solving SMT formulae is not differentiable. Finally, the DeepProblog [35] framework presents a compelling method to integrate logical constraints in the form of a ProbLog program. However, their integration with neural networks requires a complex procedure to handle the real-valued arithmetic constraints that we represent in MultiplexNet, and does not guarantee a 100% satisfaction of constraints.

1.2.3. Signal Perceptrons

Lastly, in the third part of this chapter we will discuss the signal perceptron, a neural architecture capable to learn any k-ary boolean function using only a single layer [36]. Arguably, the birth of deep learning architectures could be attributed to the need for a solution to the learning limitations of its foundational architectures called perceptrons [38]. These elementary units were proven to only be able to learn linearly separable patterns. This led to the solution of connecting multiple layers to learn separable non-linear patterns [5, 9]. These multilayer solutions called multilayer perceptrons (MLPs) came in the form of mathematical proofs which lead to a new branch of analysis of theoretical bounds on deep learning architectures [44].

Interestingly, the non-linearity problem was first introduced as a symbolic logic problem, where the functions to be learned were Boolean functions. For example, from the set of all binary Boolean functions, only 14 of the 16 possible functions are linearly separable. Additionally, increasing the number of variables exponentially increases the number of non-linear separable functions [20]. For Boolean binary functions, an MLP with sigmoid activation function needs at least one hidden layer to approximate any function from the k-ary Boolean function space with almost negligible error [25, 22]. But as the arity of the function increases, its topological boundaries (i.e. the number of learning parameters) grow exponentially, which also results in a higher computational complexity [36]. To solve these issues novel types of perceptrons have been proposed, such as complex-valued neurons [37, 2], generalized neurons [29], spiking neurons [34], and dendrite neurons [8]. However, these approaches are limited to solving only a limited set of non-linear functions [36].

Apart from the signal perceptron, the morphological-dentrite model [43] is the only mechanism capable of learning any k-ary boolean function. However, this model comes at the expense of requiring a similar amount of learning parameters as the MLP, and cannot be trained using gradient optimization methods.

1.3. Three Interpretations

Now, let us start the discussion of our three frameworks that integrate symbolic logic and deep learning. In this chapter, we only provide a high-level overview of the three approaches, and a practical example to demonstrate their capabilities. For a complete analysis on the theoretical foundations and empirical results, please refer to the main articles [16, 23, 36]. We assume that the reader is familiar with the basics of logical connectives e.g. conjunction (AND), disjunction (OR), exclusive disjunction (XOR), a general understanding of propositional logic, as well as a familiarity with the Boolean satisfiability problem (SAT), model counting #SAT problem and its extension Satisfiability Modulo Theory (SMT)–which extends the SAT problem in attempting to find assignments to usually quantifier-free formulas involving real-valued variables.

1.3.1. Logical Interpretations of Autoencoders

Firstly, we propose a framework to learn a symbolic generative model from a neural network that is trained on unstructured data [16]. A symbolic generative framework can be defined as a symbolic framework that can handle probabilistic and generative queries. In our framework, probabilistic queries refer to establishing a conditional probability over observed data. In the article, our symbolic framework answers probabilistic queries to know if some logical constraint is satisfied, given some data as evidence. As an example, one query of interest is if two images belong to the same category. In this query, the symbolic relation could be a propositional formula which will be true if indeed the two images belong to the same category. The data provided as evidence to answer this query could be an encoded version of the pair of images or some partial information/noisy representation of the images.

For the case of generative queries, the framework responses are data sampled from a predicted conditional distribution. As an analog to the previous example, a generative query may ask the framework to produce a sample of an image, given that we provide as evidence an image and know that the symbolic relation (i.e. that both images should be from the same category) is false. In this case, we expect the framework to produce an image that belongs to a different category. Now, let us start by providing a general overview of the system.

1.3.1.1. Description

Our framework can be deconstructed into two separate systems. The first is a set of autoencoders that are used to encode the unstructured data [27, 42]. The requirement in our proposal is that the encoded data (which we will refer to as the feature layer) should be a vector of discrete domain. Therefore, if X is the set that is to be encoded and $\{A_i\}$ the set of features (finite sets) that will be used to encode it, then an autoencoder of X is a function $e_X : X \to \prod_i A_i$. As an example, for the MNIST dataset [32], a set of images and labels are encoded by two separate autoencoders. Once we have encoded multiple images and labels into a set of feature layers, we can then proceed to find the relations between them.

The second part consists of designing and implementing a logic circuit that is used to represent the dependencies between the individual variables in the set of feature layers. That is, the logic circuit represents the joint probability distribution on the variables of the feature layers. In theory, any architecture (for example, sum product networks (SPNs) [41]) can be used to learn the statistical model for the joint probability distribution. In our paper, we chose to use Probabilistic Sentencial Decision Diagrams (PSDDs) [28] for two distinct reasons. First, because they have the ability to handle constraints in the learning regime. These could be "one-of" label constraints or any other kind of Boolean function over the inputs. And second, as shown by Theorem 7 of [28], the computational complexity for many probabilistic queries can be achieved in polynomial time with respect to the size of the graph. This property is commonly referred to as tractability in the literature [41, 28, 6].

Regarding the learning process, our framework is defined in two phases (see Figure 1). The first phase involves the learning of the encoder-decoder function pairs for each domain (independently and unsupervised). After completing phase one, we can use the encoders to map the data as feature representations and use them to learn the logical generative model (phase II of learning), that is, PSDD. This is done in an unsupervised manner by iteratively approximating the joint probability distribution over the data. Essentially, the variables in the set of feature layers are interpreted as logic propositions in the PSDD.

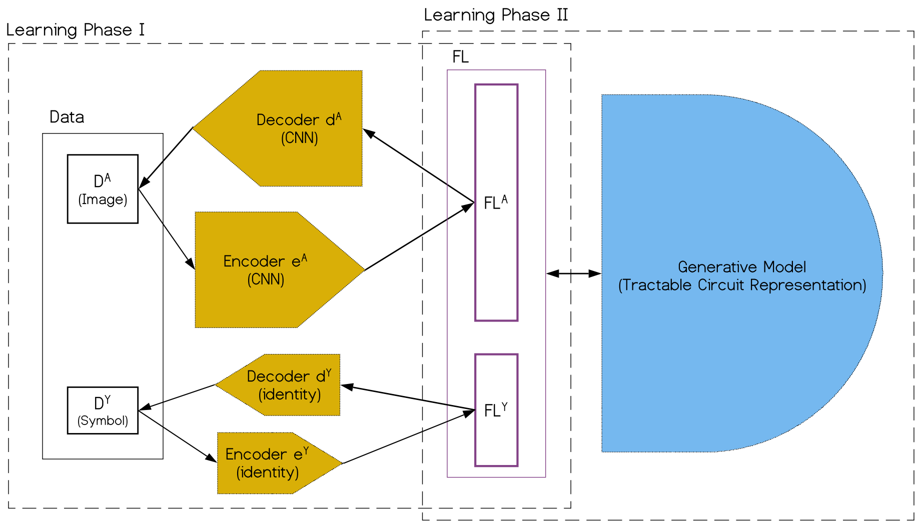


Figure 1. The learning phases of the generative symbolic framework proposed in [16].

Once the learning process is complete, we can use the logic system to answer probabilistic queries in a tractable manner, that is, to compute queries of the form: $Pr(q|v) = \frac{Pr(q \wedge v)}{Pr(v)}$ where q is the query and v is the evidence. As example of a query using the MNIST [32] dataset, consider computing the probability of an image q given that we observe that it must belong to the category labeled as v,that is, $Pr(e_X(q) \mid e_Y(v))$ where e_X, e_Y are the image and label autoencoders, respectively. Also, given evidence v we can define the task of a *generative query* as one that samples values for all encoded variables which are not assigned in the evidence. Given only a label as evidence v, PSDD can be used to sample an encoded image x_e (i.e. $x_e = generativeQuery(PSDD, e_Y(v))$) that belongs to the label. Finally, the encoded image x_e sampled from the model, can be decoded for visualization.

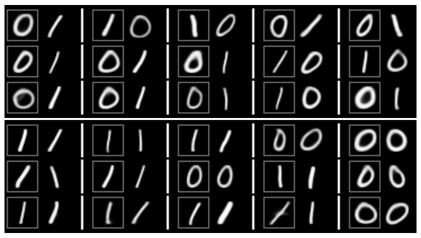
1.3.1.2. Demonstration

Let us demonstrate the frameworks' capabilities for visual reasoning by describing what is referred to as the visual XOR problem. In the visual XOR problem, there are two pairs of images that are semantically related to each other. In the paper, we consider a canonical relation to be the "categorical similarity", in which two images are related if they have the same label. For the case of the visual problem XOR, we will consider the opposite; if two images belong to the same category, we will expect the framework to learn a feature layer composed of a single Boolean variable Y_e that describes this relationship. If properly learned, this feature layer will return "0" if two images belong to the same category and "1" if they belong to a different category.[1] Conversely, we can perform a visualization of the relationship by providing a value to the logic relation $Y_e = y_e$ and the encoded version of an image $e_X(x)$. With this information, we can define a generative query that can be

[1]For ease of exposition, all our examples across the three interpretations are somewhat simple formulas. But it is worth noting that all these extend to arbitrary quantifer-free formulas.

processed by the logic framework. If trained correctly the output should be an image that belongs to the same category as x_e if $Y_e = 0$ and an image from a different category if $Y_e = 1$.

Let us briefly describe the learning process required for the visual *XOR* problem. As explained above, the first step is to train the VAE on all (e.g., MNIST, FashionMNIST) data in an unsupervised manner. Then, we create a custom training dataset for our probabilistic logic framework. For the custom dataset, each entry contains two images and the logical relationship between them. For the case of the MNIST we used the images “0” or “1” as the image pair and for Fashion-MNIST “T-shirts” and “Trousers” (the rest of the image categories can be dealt with analogously in this example). Then for the symbolic relation, we define an autoencoder that is trained using the labels of the image pair and encode them according to the Visual XOR relation. That is, for our symbolic relation we will obtain a value “0” if the labels belong to the same category and “1” if they belong to different categories. Then, we learn the PSDD by using the encoded version image pairs and the output of the symbolic relation. After training, we can visualize the symbolic relation by performing a generative query as depicted in Figure 2.


(a) *Top: (MNIST) Sampled images (with border) for $e^Y = 1$. Bottom: (MNIST) Sampled images (with border) for $e^Y = 0$.*

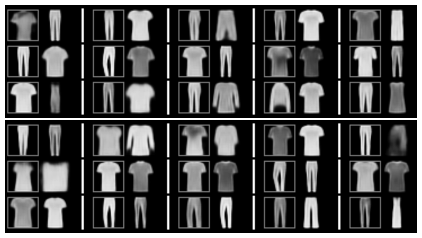

(b) *Top: (FashionMNIST) Sampled images (with border) for $e^Y = 1$. Bottom: (FashionMNIST) Sampled images (with border) for $e^Y = 0$.*

Figure 2. Visual *XOR* problem. Given a learned visual relation (XOR), we sample the image that is congruent with the symbolic relation value and the input image [16].

The generated images depicted in Figure 2 illustrates that indeed, our symbolic framework learned to generate images consistent with the logic relation. We can observe that for the MNIST dataset, if the input for the generative query was an image of the digit “0” and the symbolic relation *XOR* was set to “1”, the framework produces sampled images with “1”. Also, for the case of the fashion-MNIST, if the inputs of the generative query were “Trousers” and the *XOR* variable was set to “0”, we can observe that the generated images were also ”Trousers”. The same applies to the other categories used in the training.

1.3.2. Logical constraints with Multiplex Networks

Now that we've shown a way to use the symbolic logic framework to perform visual reasoning, we will shift our attention to a novel method for encoding expert knowledge into the training process of deep networks called Multiplex Networks (MultiplexNets). As briefly mentioned in Sections 1.1 and 1.2, there are several applications of incorporating knowledge into the training process of the deep learning architecture. One reason is safety constraints, which are rules that deep learning agents must satisfy while training. An example of this may be a self-driving car where a controller should operate within a predefined set of geometric and physical constraints (e.g., the car should always stop completely at a stop-street). Another area of application is data efficiency. Deep learning models have shown unprecedented performance on a wide variety of tasks, but these come at the cost of large data requirements. For example, OpenAI's GPT-3 [7] was trained on approximately 500 billion tokens, and ImageNet-21k, used to train the ViT network [11], consists of 14 million images. In theory, for similar tasks where domain knowledge exists, we could use it to structure the network's training to reduce the data burden placed on the learning process. With this in mind, we will show how MultiplexNets can be used to encode expert knowledge into the neural architecture for safety constraint tasks.

1.3.2.1. Description

Let us begin by providing a general specification of the problem we are trying to solve and the set of assumptions necessary for our framework to operate. Consider a data set of N i.i.d. samples from a probability density function p^* [6] i.e. we assume that the data set was generated by some random process where the probability of obtaining x is $p^*(x)$–in which there exists domain or expert knowledge. In the MultiplexNets article the expert knowledge about the random process p^* is represented in the form of a logical formula φ. We can formally summarize these assumptions as follows.

$$x \sim p^*(x) \implies x \models \varphi \tag{1}$$

In Eq. 1, the notation $x \models \varphi$, denotes that the sample x satisfies the logic formula φ [4]. For example, if $\varphi := (x > 3.5) \wedge (y > 0)$, then a sample $(x,y) = (5,2)$, satisfies that $(x,y) \models \varphi$.

The aim of MultiplexNets is to approximate $p^*(x)$ with some parametric model $p_\theta(x)$ and to incorporate the domain knowledge φ into the maximum likelihood estimation of θ, on the available data set. To achieve this, the framework proposes a reparameterization methodology that transforms the unconstrained output of the network $\tilde{x}$ into an output that is consistent with the domain knowledge $h(x) \models \varphi$. To exemplify this process, if the knowledge to be encoded is $\varphi : \forall x(cx \geq b)$, MultiplexNets will perform the following:

1. Constrain $\tilde{x}$ to be non-negative by applying an activation function g–this can be any element-wise non-negative, for example, a ReLU [39] or Softplus [12] .
2. Apply a transformation f. In this case, f can implement the transformation $f(z) = sgn(c)z + \frac{b}{c}$ where sgn is the operator that returns the sign of c.

By construction we have the following:

$$h(x) = f(g(\tilde{x})) \models \varphi \tag{2}$$

It follows that more complex conjunctions of constraints can be encoded by composing transformations of the form presented in Eq. 2. Conceptually, appending additional conjunctions to φ serves to restrict the space that the output can represent. But in many situations, domain knowledge is complex and consists of complicated formulas that go beyond mere conjunctions or inequalities. To solve this, the paper proposes a transformation of complex formulas into a disjunctive normal form (DNF). This transformation allows us to seamlessly apply the transformations to satisfy φ. Following this process, the method proceeds to the training of the neural architecture, which is now guaranteed to always satisfy the safety constraints while being trained. To illustrate the safety constraint satisfaction process, we will present the synthetic toy example called safety *XOR*.

1.3.2.2. Demonstration

In the safety *XOR* problem, a scientist wants to use deep learning methods to train the navigation system of a reusable rocket. The rocket navigation system needs to learn to predict the best pair of (x,y) coordinates for the rocket to land based on some sensory data. The scientist knows that the success of the landing is dependent on some critical areas which can be described by the logic constraint $\varphi = (x \leq 5) XOR (y > 5)$. Outside of this geometric boundary, there is complete certainty that the rocket will fail to land and be critically damaged. For this reason, before starting the training of the navigation system, the scientist proceeds to create the MultiplexNet to satisfy such a constraint.

To encode this expert knowledge during training, the framework transforms formula φ into a disjunctive normal form (DNF), that is, $\varphi = ((x \leq 5) \wedge \neg(y > 5)) \vee (\neg(x \leq 5) \wedge (y > 5))$. With this encoding, we can observe that there are two possible models that satisfy the constraint, which are $\varphi_1 = (x \leq 5) \wedge \neg(y > 5))$ and $\varphi_2 = (\neg(x \leq 5) \wedge (y > 5)))$.

Then the MultiplexNets apply a similar transformation to Equation 2, on the unconstrained output (x,y) to satisfy each formula φ_1, φ_2. This results in two transformations $h_1(x,y) = (-g(x)+5, -g(y)+5)$ and $h_2(x,y) = (g(x)+5, g(y)+5)$. For this example, we assume that g is the RELU activation function. Note that each $h_k \in \{h_1, h_2\}$ transformation to the output of the unconstrained system only satisfies a particular $h_k(x,y) \models \varphi_k$ which is sufficient to satisfy φ as $h_k(x,y) \models \varphi$ for all $k \in [1,\ldots,K]$. Nonetheless, the choice of formula is left for the training process by introducing a latent Categorical variable k that selects among the different terms $\varphi_k, k \in [1,\ldots,K]$ [23]. Now the modeller can start the neural network training process and guarantee that the constraints will be satisfied 100% of the time.

Figure 3 displays the samples obtained from the navigation system, which are drawn from a multivariate normal distribution. In the left side of the figure, the model's unconstrained inputs are shown; where the green dots are samples from the model that satisfy the safety *XOR*, and the red dots are samples that violate the constraint. In the right side, the unconstrained outputs are transformed by h_1, h_2, which yield two clouds of datapoints that always satisfy the constraint.

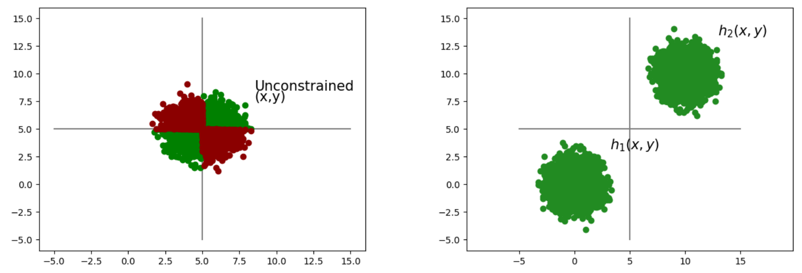


Figure 3. Safety XOR example, where red points represent samples that doesn't satisfy the safety constraint and green dots samples that satisfies it. On the left, we display the set of samples taken from the unconstrained system. On the right, we show the the same set of samples constrained by the MultiplexNet transformations h_1 and h_2 which guarantee to satisfy the formulas φ_1, φ_2 respectively.

1.3.3. Learning functional representations of propositional formulas with the Signal Perceptron

Finally, for our third contribution, we will discuss how deep learning can be used to solve pure symbolic logic problems. Specifically, we propose a neural mechanism that is capable of learning a functional representation of any n-ary propositional formula–a propositional formula that is composed with n atomic propositions–using a single layer. The functional form of an n-ary propositional formula can be defined as a mapping of the form $\varphi : \{0,1\}^n \to \{0,1\}$. Similarly to Section 1.3.1, where the values 0 and 1 correspond to the truth values true and false respectively, and for each vector $(x_1,...,x_n) \in \{0,1\}^n$, the i-th coordinate is the truth value of the i-th variable. In this section we will illustrate how this architecture can easily be implemented and trained to learn any binary logic connective such as the AND, OR and the XOR.

1.3.3.1. Description

First, let us start by providing a definition of our learning mechanism. The signal perceptron is a novel architecture that can be defined as a sum of sinusoids (sine waves). In the main article [36], there is a proposal of different versions of the architecture where the sinousoids are defined in a complex or real domain. In this section, we will provide a definition of the real domain signal perceptron in which sinousoids are cosine functions.

In its vector form, the real signal perceptron can be defined as:

$$s(\hat{x}) = \sum_{j=0}^{m^k-1} \theta_j cos(\frac{\pi}{m-1}\hat{\omega}_j \cdot \hat{x}) \tag{3}$$

where $m,n \in \mathbb{N}$, $\bar{\omega}_j \in m^n = \{(a_1,...,a_n)|a_i \in \{0,...,m-1\}\}$, $\theta_j \in \mathbb{C}$ and $\bar{\omega}_j \cdot \bar{x}$ the dot product.

As an example, if we want to learn any binary boolean function–where $m = n = 2$ then Equation 4 is:

$$s(\hat{x}) = \sum_{j=0}^{3} \theta_j cos(\pi \hat{\omega}_j \cdot \hat{x}) = \theta_1 + \theta_2 cos(\pi x_1) + \theta_3 cos(\pi x_2) + \theta_3 cos(\pi(x_1 + x_2)) \qquad (4)$$

In the article [36] we provide a formula to compute the amount of learnable parameters,given by m^n. While this parameter is of exponential magnitude, in reality, this represents only an upper bound as some functions will require less parameters to be learned [36]. Also, this upper bound is the smallest in the literature, showing that for architectures that claim to be able to learn any n-ary boolean function, the amount of parameters required is higher. Now we proceed to demonstrate the capabilities of the signal perceptron for learning Boolean functions.

1.3.3.2. Demonstration

To provide an intuition of how the signal perceptron is able to learn any n-ary Boolean function, we will show the learning process for the functional representation of any other binary logic connectives–including the XOR. First, let us remember that we are performing function learning, as opposed to function approximation. This means that as a requirement, we will need to ensure that each point in the domain X gets assigned the correct value of the codomain Y. For binary Boolean functions, the domain is quite small, as it is defined by four elements $X = \{(0,0),(0,1),(1,0),(1,1)\}$. In the article, it is shown that we can train the architecture using gradient descent, and the proof that it can learn any Boolean function is equivalent to proving that each function determines a system of linear equations that always has a solution [36].

In this section we will illustrate the gradient descent process. The domain X is passed as an input and then the signal perceptron predicts an output which is compared against the corresponding values through a loss function. Then, the loss value is used by the gradient descent algorithm to update the parameters. The process is repeated until the algorithm returns a zero loss, which tells us that the Boolean function is learned. Figure 4 shows the results of the training process using vanilla gradient descent with the signal perceptron and the one hidden layer MLP. The results clearly show that the signal perceptron requires fewer iterations to converge to the true value while the MLP takes longer. Interestingly, for the XOR function labeled function 6 and the double-implication operator labeled function 9, the MLP has the greatest difficulty to converge. Let us remember that these functions are in fact the only non-linear separable functions from the set of binary Boolean functions.

1.3.4. Conclusion

In this paper we consider three approaches for integrating symbolic logic systems and deep learning methods. We demonstrate that these techniques allow us to extend the capabilities of current deep learning methods to be able to perform visual reasoning [16], guaranteeing the satisfaction of safety constraints while training [23] and to creating deep learning models that are able to perform exact function learning [36].

1.4. Acknowledgments

This work was partly supported by a grant from the UKRI Strategic Priorities Fund to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regu-

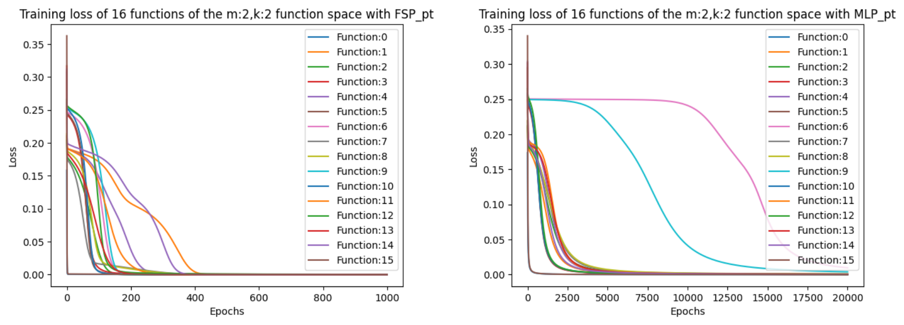


Figure 4. Plots of the loss function of all 16 binary Boolean functions using the real signal perceptron and the one hidden layer MLP. The graph on the right shows that even after 20000 epochs, the MLP cannot exactly learn some of the functions [36] .

lation (EP/V026607/1, 2020-2024). Vaishak Belle is also supported by a Royal Society University Research Fellowship. Miguel-Angel Mendez Lucero is supported by Consejo Nacional de Ciencia y Tecnología CONACYT. We also thank the authors of the disscused papers Anton Fuxjaeger, Enrique Bojorquez, Nick Hoernle, and Rafael-Michael Karampatsis for allowing us to use their material to create this chapter. Finaly, we thank Enrique Bojorquez, Ionela Mocanu and Andreas Bueff for their feedback on the article.

References

[1] C. Allen, I. Balažević, and T. Hospedales. A probabilistic framework for discriminative and neuro-symbolic semi-supervised learning. *arXiv preprint arXiv:2006.05896*, 2020.

[2] M. F. Amin, M. M. Islam, and K. Murase. Single-layered complex-valued neural networks and their ensembles for real-valued classification problems. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 2500–2506, 2008.

[3] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[4] C. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, A. Biere, M. Heule, H. van Maaren, and T. Walsh. Handbook of satisfiability. *Satisfiability modulo theories*, 185:825–885, 2009.

[5] E. B. Baum. On the capabilities of multilayer perceptrons. *J. Complex.*, 4(3):193–215, Sept. 1988.

[6] V. Belle, A. Passerini, and G. Van den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2015, pages 2770–2776, 2015.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[8] R. D. Cazé, M. Humphries, and B. Gutkin. Passive dendrites enable single neurons to compute linearly non-separable functions. *PLOS Computational Biology*, 9(2):1–15, 02 2013.

[9] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, Dec. 1989.

[10] L. De Raedt and A. Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[12] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in neural information processing systems*, pages 472–478, 2001.

[13] S. Dumancic, T. Guns, W. Meert, and H. Blockleel. Auto-encoding logic programs. In *International Conference on Machine Learning, Location: Stockholm, Sweden*, 2018.

[14] M. Fischer, M. Balunovic, D. Drachsler-Cohen, T. Gehr, C. Zhang, and M. Vechev. DL2: Training and querying neural networks with logic. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1931–1941, 2019.

[15] Z. Fu and Z. Su. XSat: A Fast Floating-Point Satisfiability Solver. In *Proceedings of the 28th International Conference on Computer Aided Verification, Part II*, pages 187–209. Springer, 2016.

[16] A. Fuxjaeger and V. Belle. Logical interpretations of autoencoders, 2019.

[17] R. Gens and P. Domingos. Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pages 3239–3247, 2012.

[18] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. Adaptive computation and machine learning. MIT Press, 2007.

[19] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[20] N. Gruzling. Linear separability of the vertices of an n-dimensional hypercube., 2007.

[21] L. Hammond and V. Belle. Learning tractable probabilistic models for moral responsibility and blame. *Data Mining and Knowledge Discovery*, 35(2):621–659, 2021.

[22] J. A. Hertz, A. S. Krogh, and R. G. Palmer. *Introduction To The Theory Of Neural Computation*. Addison-Wesley, Reading. MA. USA, 1991.

[23] N. Hoernle, R. Karampatsis, V. Belle, and K. Gal. Multiplexnet: Towards fully satisfied logical constraints in neural networks. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 5700–5709. AAAI Press, 2022.

[24] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2410–2420, 2016.

[25] G.-B. Huang and H. Babri. General approximation theorem on feedforward networks. In *Proceedings of ICICS, 1997 International Conference on Information, Communications and Signal Processing. Theme: Trends in Information Systems Engineering and Wireless Multimedia Communications (Cat.*, volume 2, pages 698–702 vol.2, 1997.

[26] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM Comput. Surv.*, 54(10s), sep 2022.

[27] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations (ICLR)*, 2014.

[28] D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *KR*, 2014.

[29] R. V. Kulkarni and G. K. Venayagamoorthy. Generalized neuron: Feedforward and recurrent architectures. *Neural Networks*, 22(7):1011–1017, 2009.

[30] A. Kumar, S. Verma, and H. Mangla. A survey of deep learning techniques in speech recognition. In *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pages 179–185, 2018.

[31] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. Smt techniques for fast predicate abstraction. In *International Conference on Computer Aided Verification*, pages 424–437. Springer, 2006.

[32] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[33] Y. Liang and G. Van den Broeck. Learning logistic circuits. *In Proceedings of the 33rd Conference on Artificial Intelligence (AAAI)*, 2019.

[34] W. Maass and M. Schmitt. On the complexity of learning for a spiking neuron (extended abstract). In *Proceedings of the Tenth Annual Conference on Computational Learning Theory*, COLT '97, page 54–61, New York, NY, USA, 1997. Association for Computing Machinery.

[35] R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, and L. De Raedt. Deepproblog: Neural probabilistic logic programming. *arXiv preprint arXiv:1805.10872*, 2018.

[36] M.-A. Mendez Lucero, R.-M. Karampatsis, E. Bojorquez Gallardo, and V. Belle. Signal perceptron: On the identifiability of boolean function spaces and beyond. *Frontiers in Artificial Intelligence*, 5, 2022.

[37] H. Michel and A. Awwal. Artificial neural networks using complex numbers and phase encoded weights.

Applied optics, 49:B71–82, 04 2010.

[38] M. Minsky. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.

[39] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, 2010.

[40] D. W. Otter, J. R. Medina, and J. K. Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2021.

[41] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.

[42] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[43] G. Ritter and G. Urcid. Lattice algebra approach to single-neuron computation. *IEEE Transactions on Neural Networks*, 14(2):282–295, 2003.

[44] D. Stathakis. How many hidden layers and nodes? *Int. J. Remote Sens.*, 30(8):2133–2147, Apr. 2009.

[45] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[46] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.