The Topos of Transformer Networks

Mattia Villani
Department of Informatics
King's College London
London, United Kingdom
mattia.villani@kcl.ac.uk

Peter McBurney
Department of Informatics
King's College London
London, United Kingdom
peter.mcburney@kcl.ac.uk

The transformer neural network has significantly out-shined all other neural network architectures as the engine behind large language models. We provide a theoretical analysis of the expressivity of the transformer architecture through the lens of topos theory. From this viewpoint, we show that many common neural network architectures, such as the convolutional, recurrent and graph convolutional networks, can be embedded in a pretopos of piecewise-linear functions, but that the transformer necessarily lives in its topos completion. In particular, this suggests that the two network families instantiate different fragments of logic: the former are first order, whereas transformers are higher-order reasoners. Furthermore, we draw parallels with architecture search and gradient descent, integrating our analysis in the framework of cybernetic agents.

1 Introduction

The transformer architecture [45] has achieved incredible success in a broad range of tasks, and has become the de-facto architecture for deep learning [48]. This is the neural network model that underlies the remarkable success of virtually all large language models, including ChatGPT [49] and the recent Claude [3]. To a large extent, the theoretical foundations of its success remain unexplored: there do not exist conclusive theories of why the transformer has achieved unprecedented success on a broad range of language understanding, generation and reasoning tasks.

The aim of this paper is to provide a categorical perspective on the architectural differences between traditional feedforward neural networks and transformers [45]. First, we provide a setting for categorical deep learning that is stricter than many commonly found in the literature. This ensures that any result that is true in this category can be understood to be true for a subset of commonly found neural network architectures. Secondly, we investigate what sets Transformer architectures apart from a topos theoretic perspective.

Topos theory studies the emergence of logical structures in different settings of mathematics. Through this lens, we are able to approach the question of expressivity of architecture from a logical perspective. This allows us to address for the first time the question: what fragment of logic is this network implementing? In particular, we will show that ReLU networks, with only linear and ReLU layers, and generalisations with tensor contractions, belong to a pretopos, but not necessarily a topos. The transformers will be shown to be in a co-product completion of the category, which is a topos. The significance of this is that the internal language of the transformer has a richer quality: it is higher-order. This may provide a new way to interpret the success of the architecture. Finally, we will define architecture search and backpropagation in the categorical setting, establishing a lens to reason about learners.

A burden of theorists can be that of providing prescriptive insight to practitioners; the results of this paper can have actionable consequences on the deployment of neural networks. Specifically, we expect this work to lead to empirical research towards building neural network architectures that display similar characteristics as the transformer, i.e. can be factored into **choose** and **evaluate** morphisms. Indeed, the

Submitted to:
SOS 2007

key takeaway for practitioners in our paper is that what sets the transformer network apart, through the attention mechanism, seems to be the input-dependent weights. Building layers with this design property may lead to the discovery of new and better-performing architectures.

Moreover, our theoretical insight may provide guidance on how to *explain* the networks differently. In particular, seeing that we show how transformers should be viewed *collections of models*, an explanation should address the local / contextual nature of the model.

Related Work: In recent work by [12], Category theory was called to be a unifying force in deep learning theory in analogy to how it brought together geometry in the Erlangen program. We see a first application of categories in the definition of equivariance via natural transformations in [20], and the study of expressive sheaf-based neural networks as in [10],[8] and [7]. This line of research is particularly promising: it provides a natural setting for geometric deep learning devoid of particular assumptions on the underlying mathematical structure of the data. By abstracting away from the details, the authors go beyond the heuristics of assigning architectures to data types (recurrent neural networks for time series, convolutional neural networks for images etc.) and develop a formal theory that matches architectures to known symmetries of the problem. Such theory opens the possibility of finding appropriate architectures autonomously.

On the other hand, Applied Category Theory (ACT) has witnessed growing interest for characterisations of artificial intelligence. [18] first addresses deep learning through categories, by noticing functorial properties of back propagation. Work by [15] and [14] represent neural networks through the formalisms of parametric lenses, unifying game theory with deep learning under a common language. [9] provide an alternative view, which delves into greater granularity: using model categories to define equivariance signatures of networks and realising for the first time that a neural network may have an internal language, as [41] showed independently. Indeed, [42] proceeds to develop a theory of neural networks which relies on Polynomial Functors. These works enhance the trustworthiness and verification capabilities of AI systems, as suggested by [31], insofar as they provide complete and granular frameworks for their deployment.

An underlying effort in this paper will be to select a category for neural networks that encompasses most commonly used architectures, without generalising excessively what we admit in the definition of a learner. The standard choice of **Euc** or **Smooth** as a category of neural networks, would result in having fewer or more morphisms than desired. We provide a setting suitable for deep learning in the category of piece-wise linear functions **PL**, as studied by [40]. The details of the **PL** construction can be found in [51], and [11] for a recent graphical and operadic description of the category. We argue that if the choice of base category is too coarse, including morphisms that are not generally understood to be components of neural network architecture, the categorical results cannot be taken to provide insight on the nature of deep learning. In this work, we err in prudence, only selecting a family of networks (ReLU networks), but one which captures all main characteristics of these models. Recent work by [33] discusses how the different categories of neural networks can be related via categorical constructions. Finally, [19] provide evidence of the utility of applied category theory in deep learning by developing a framework to tailor a neural architecture around a desired input type.

2 Category of Neural Networks

The main aim of this section will be to establish a setting in which we can compare architectures while striking the right balance of abstraction and concreteness. We present ReLU networks and provide a categorical characterisation of them.

Definition 1 (ReLU Network)**.** *An L-layer ReLU network is a function $\mathscr{N} : \mathbb{R}^n \to \mathbb{R}^m$ such that*

$$z^{(l)} = \sigma(W^{(l)} z^{(l-1)} + b^{(l)}), l \in [L],$$

such that, for a vector of layer widths $\mathbf{n} = [n_0, ..., n_{L+1}]$, with $n_0 = n, n_{L+1} = m$ indicating the number of neurons at each layer, we have that $z^{(l)}, b^{(l)} \in \mathbb{R}^l, W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\sigma(x_i) = x_i$ iff $x_i > 0$ and $\sigma(x_i) = 0$ otherwise, $i \in [n_l]$ for some $l \in \{1, ..., L\}$. Note that σ is called an activation function and is applied elementwise in each entry of the vector. Moreover, we have the two extremal conditions:

$$\mathscr{N}(x) = z^{(L+1)} = W^{(L+1)} z^{(L)} + b^{(L+1)}, z^{(0)} = x.$$

Broadly speaking, the following proposition will be a work of translation into category theory from [43] and [46]. Nevertheless, there is novelty in the proof that this holds for all neural networks with $n \in \mathbb{N}$ layers.

Proposition 2.1 (**ReLU Linear Layer**)**.** *Each layer of a ReLU feedforward neural network $\mathscr{N} : \mathbb{R}^n \to \mathbb{R}^m$ can be factored into the following diagram in **Set**:*

$$\begin{tikzcd}
\mathbb{R}^n \arrow[rr, "\mathscr{N}"] \arrow[dr, "s_{\mathscr{N}}"'] & & \mathbb{R}^m \\
& \mathbb{R}^n \times \{0,1\}^m \arrow[ur, "a_{\mathscr{N}}"'] &
\end{tikzcd}$$

where $s_{\mathscr{N}} = id \times \chi_{\{W^T \cdot - + b > 0\}}$, where $\chi_{\{- > 0\}}$ is a componentwise characteristic function, assessing whether a component of the preactivation is greater than zero or not (i.e. active), and $a_{\mathscr{N}} = diag(-_2) \cdot (W^T \cdot -_1 + b)$, applies the activation as a linear operation, by diagonalising the $\{0,1\}$-vector computed by $\chi_{\{W^T \cdot - + b > 0\}}$.

Proof. To show that the diagram commutes we go through the computations of $s_{\mathscr{N}}$ and $a_{\mathscr{N}}$. For a given $x \in \mathbb{R}^n$, $\chi_{\{W^T \cdot - + b > 0\}}(x)$, acting componentwise, provides a $\{0,1\}$-vector which takes values 1 exactly when the ReLU function will be active (the preactivation is greater than 0) and 0 otherwise. This means that the action of the ReLU function would be equivalent to a diagonal matrix $diag(P), P \in \{0,1\}^m$. Hence, we view the first function as selecting the appropriate linear transformation to apply in the second function. This leads us to using the vector as an input of $a_{\mathscr{N}}(P,x) = diag(P) \cdot (W^T \cdot x + b) = \sigma(W^T \cdot x + b)$ whenever $x \in \{x \in \mathbb{R}^n : \forall i \in [m], W_i^T x_i + b > 0 \iff P_i = 1\}$, which completes the proof. ☐

In other words, the function $s_{\mathscr{N}}$ computes an internal state of the layer and $a_{\mathscr{N}}$ applies the appropriate linear transformation to the network as a result. Notice how this reduces the neural network into a collection of linear models, each applicable in different parts of a partition of the input space. The set of activation patterns of the neural network encodes concepts in the input space. We can understand ReLU networks (and networks with piece-wise linear activations) as stores of linear models, each encoded by one of the patterns. With this interpretation, we show that this decomposition is valid for general feed-forward architectures.

Lemma 2.2 (**ReLU Neural Networks**)**.** *Let $\mathscr{N}^{(L)} : \mathbb{R}^n \to \mathbb{R}^m$ be a feedforward neural network, made up of the composition of L layers:*

$$\mathscr{N}^{(L)} = \bigcirc_{i \in [L]} \mathscr{N}^{(i,i+1)} = \mathscr{N}^{(L-1,L)} \circ \mathscr{N}^{(L-2,L-1)} \circ ... \circ \mathscr{N}^{(1)},$$

where the notation $\mathscr{N}^{(l,l+1)}$ refers to the $l+1$th layer.

*Then there exists a factorisation diagram $a_{\mathcal{N}} \circ s_{\mathcal{N}}$ in **Set**,*

$$\begin{tikzcd}
\mathbb{R}^n \arrow[rr, "\mathcal{N}"] \arrow[dr, "s_{\mathcal{N}}"'] & & \mathbb{R}^m \\
& \mathbb{R}^n \times \prod_{i\in[L]}\{0,1\}^{n_{i+1}} \arrow[ur, "a_{\mathcal{N}}"'] &
\end{tikzcd}$$

such that:

$$a_{\mathcal{N}} = \left(\prod_{l=0}^{L-1} diag(-_{L+1-l})W^{(L-l)}\right)W^{(1)} \cdot -_1 + \sum_{l=1}^{L}\prod_{h=1}^{L+1-l} W^{(L+1-h)}diag(-_{L-h})b^{(l-1)} + b^{(L)}$$

and,

$$s_{\mathcal{N}} = id \times \chi^{(L)} \times \chi^{(L-1)} \times ... \times \chi^{(1)}$$

where

$$\chi^{(l)} = \boldsymbol{1}_{(0,\infty)}\left(\sigma(...\sigma(\sigma(-^T \cdot W^{(1)} + b^{(1)}) \cdot W^{(2)} + b^{(2)})...) \cdot W^{(l)} + b(l))\right)$$

with $\boldsymbol{1}_{(0,\infty)}$ representing an elementwise indicator function, which outputs 1 if $x \in (0,\infty)$ and 0 otherwise.

Proof. The definition of $a_{\mathcal{N}^{(L)}}$ follows exactly from the local linear model decomposition of [43], which provides the construction of how the linear model can be derived from the weights and activation patterns of the feedforward ReLU network. It remains to show that there exists a function $s_{\mathcal{N}^{(L)}}$. We prove this by induction. The base case was the result of Proposition 2.1. The following diagram gives the proof for the inductive step. By letting $\mathcal{N}^{(L)} = \mathcal{N}^{(L-1,L)} \circ \mathcal{N}^{(L-1)}$ be the application of the Lth layer $\mathcal{N}^{(L-1,L)}$ to the $L-1$-layered network $\mathcal{N}^{(L-1)}$, we obtain the following decomposition:

$$\begin{tikzcd}
\mathbb{R}^n \arrow[rrrr, bend left, "\mathcal{N}^{(L)}"] \arrow[rr, "\mathcal{N}^{(L-1)}"] \arrow[dr, "s_{\mathcal{N}^{(L-1)}}"'] \arrow[ddrr, bend right, "s_{\mathcal{N}^{(L)}}"'] & & \mathbb{R}^k \arrow[rr, "\mathcal{N}^{(L-1,L)}"] \arrow[dr, "s_{\mathcal{N}^{(L-1,L)}}"'] & & \mathbb{R}^m \\
& \mathbb{R}^n \times \prod_{i\in[L-1]}\{0,1\}^{n_{i+1}} \arrow[ur, "a_{\mathcal{N}^{(L-1)}}"'] \arrow[dr, "id \times a_{\mathcal{N}^{(L-1)}} \circ \mathbf{1}_{(0,\infty)}"'] & & \mathbb{R}^k \times \{0,1\}^m \arrow[ur, "a_{\mathcal{N}^{(L-1,L)}}"] & \\
& & \mathbb{R}^n \times \prod_{i\in[L]}\{0,1\}^{n_{i+1}} \arrow[ur, "a_{\mathcal{N}^{(L-1)}} \times id"] \arrow[uurr, bend right, "a_{\mathcal{N}^{(L)}}"'] & &
\end{tikzcd}$$

where, in particular, we use the fact that $\chi^{(l)} = \mathbf{1}_{(0,\infty)}(a_{f_\theta^{(l)}})$ to explicitly verify:

$$\begin{aligned}
s_{\mathcal{N}^{(L)}} &= (id \times a_{\mathcal{N}^{(L-1)}} \circ \mathbf{1}_{(0,\infty)}) \circ s_{\mathcal{N}^{(L-1)}} \\
&= (id \times a_{\mathcal{N}^{(L-1)}} \circ \mathbf{1}_{(0,\infty)}) \circ (id \times \chi^{(L-1)} \times \chi^{(L-1)} \times ... \times \chi^{(1)}) \\
&= (id \times \chi^{(L)} \times \chi^{(L-1)} \times \chi^{(L-1)} \times ... \times \chi^{(1)}),
\end{aligned}$$

which shows that the case $L-1$ implies the L case. Together with Proposition 2.1, this completes the proof. □

Lemma 2.2 entails the existence of a forgetful functor from a category of feed-forward neural networks to the category of piece-wise linear functions **PL** as described in [40]. This category, will be the base setting for our categorical theory of architecture; this is motivated by the following theorem.

Proposition 2.3. *Let* ***ReLU*** *be the category of ReLU neural networks, with real spaces as objects and ReLU neural networks $\mathscr{N} : \mathbb{R}^n \to \mathbb{R}^m$ as morphisms. Then there exists a forgetful functor into the category of piece-wise linear functions:*

$$F : \textbf{\textit{ReLU}} \to \textbf{\textit{PL}}.$$

Proof. Checking functoriality we need to show that the composition of two ReLU networks is a ReLU network whose piece-wise linear function is the composition of the respective two piecewise linear functions. This is a consequence of the *select-apply* decomposition from Lemma 2.2: the select function defines a partition in the space through the fibers s_N^{-1}, on which we determine the function $a_{\mathscr{N}}$ to apply, which yields a piece-wise linear function. In particular, for two networks $\mathscr{N}, \mathscr{N}'$ with composition $\mathscr{N}'' = \mathscr{N}' \circ \mathscr{N}$, we have that,

$$F(\mathscr{N}' \circ \mathscr{N}) = F((a_{\mathscr{N}'} \circ s_{\mathscr{N}'}) \circ (a_{\mathscr{N}} \circ s_{\mathscr{N}})) = F(\mathscr{N}') \circ F(\mathscr{N}),$$

which are also equal to $F(a_{\mathscr{N}' \circ \mathscr{N}} \circ s_{\mathscr{N}' \circ \mathscr{N}}) = F(\mathscr{N}'')$. The morphism is forgetful because it does not preserve the structure of the neural networks. Many networks can yield the same piece-wise linear function and are mapped to the same $F(\mathscr{N})$. □

Several works characterise the expressive power of ReLU network [32], [4], [22]. Amongst the goals of the authors was to provide an upper bound to the depth of a neural network representing exactly a given piecewise linear function with finitely many parts. The depth has been found to be a function of the input dimensionality by [25], and is related to the representation of maximum. There are no proofs that this representation is canonical. However, [47] show that whenever weights are allowed to take infinite values, there exists a *shallow* network of depth 3 for every given finite piece-wise linear function. This is significant insofar it allows us to build a monad in the category or ReLU networks, whenever these are allowed to have infinity weights.

Theorem 2.4. *Consider the subcategory $\textbf{\textit{PL}}_{\textit{fin}}$ of* ***PL*** *generated by the piecewise linear functions with finitely many linear parts. There exists a functor net : $\textbf{\textit{PL}}_{\textit{fin}} \to \textbf{\textit{ReLU}}$ that assigns to each of these parts a functionally equivalent neural network. Moreover, $F : \textbf{\textit{ReLU}} \to \textbf{\textit{PL}}_{\textit{fin}}$ and net are an adjoint pair and form a monad on* ***ReLU****, selecting a canonical equivalent network for every network.*

Proof. The existence of such a network is in general guaranteed by [21]. However, a functor needs to map every piece-wise linear function to a *unique* network. [46] shows that there exists a shallow network for every deep ReLU network that is unique, up to renaming of neurons and the hyperplanes that generate the input regions. A lexicographic choice of the weights and biases guarantees the uniqueness of such network.

We now want to show that the functor S given by

$$\begin{aligned} S : \quad & \textbf{ReLU} \longrightarrow \textbf{ReLU} \\ & (\mathscr{N} : \mathbb{R}^n \to \mathbb{R}^m) \mapsto (\mathscr{S} : \mathbb{R}^n \to \mathbb{R}^m) \end{aligned}$$

which selects a shallow network $\mathscr{S}$ for each neural network $\mathscr{N}$ forms a monad. To do this, we show the existence of the unit and multiplication natural transformations ε, μ. The multiplication natural transformation follows from the idempotency of the operation: $S(S(\mathscr{N})) = S(\mathscr{N}) = \mathscr{S}$. The unit natural transformation is given by canonicity, mapping to every morphism in the category, the canonical shallow network gives the natural transformation $\varepsilon : 1_{\mathbf{ReLU}} \implies S$, with components in the commuting squares

$$\begin{tikzcd} 1_{\mathbf{ReLU}}(X) \arrow[r, "\mathscr{N}"] \arrow[d, "\varepsilon_X"'] & 1_{\mathbf{ReLU}}(Y) \arrow[d, "\varepsilon_Y"] \\ S(X) \arrow[r, "\mathscr{S}"'] & S(Y). \end{tikzcd}$$

where ε are identity morphisms. Indeed, we verify that $\mu \circ S\mu = \mu \circ \mu S$ and $\mu \circ \varepsilon S = \mu \circ S\varepsilon = 1_{\mathbf{ReLU}}$. □

3 Transformers

In this section, we provide a technical note on the transformer [45], a popular architecture that has grown in popularity. The transformer [45], which relies on the self-attention mechanism [30]; we study self-attention as it commonly occurs in transformer networks and present a factorisation which will be instrumental to showing how this network is distinct from other families of neural networks contraction based families.

Definition 2 (Self-Attention Mechanism and Transformer). *For a triplet of matrices $W_K \in \mathbb{R}^{n\times k'}, W_Q \in \mathbb{R}^{n\times k}, W_V \in \mathbb{R}^{m\times n}$ and input $X \in \mathbb{R}^{n\times k}$*

$$Att(X; W_K, W_Q, W_V) = softmax\left(\frac{{W_Q}^T X (W_K^T X)^T}{\sqrt{d}}\right) W_V^T X.$$

*The **transformer** is a finite composition of piece-wise linear maps that factors through the attention mechanism.*

This layer type can be understood as performing two actions: first selecting a set of parameters and, in doing so, parameterise a feed-forward neural architecture; then, using the chosen architecture to compute an output. We state this formally.

Proposition 3.1. *The self-attention mechanism can decomposed into a choice of network parameters*

$$\mathbf{choose} : \mathbb{R}^{n\times k} \to \hom(\mathbb{R}^{n\times k}, \mathbb{R}^{n\times k'}),$$

followed by the evaluation of a neural network,

$$\mathbf{eval} : \mathbb{R}^{n\times k} \times \hom(\mathbb{R}^{n\times k}, \mathbb{R}^{n\times k'}) \to \mathbb{R}^{n\times k},$$

where $\hom(\mathbb{R}^{n\times k}, \mathbb{R}^{n\times k'})$ *is the space of neural network layers with input* $\mathbb{R}^{n\times k}$ *and output* $\mathbb{R}^{n\times k'}$.

Proof. Identifying these functions in the general case proves the statement. In particular, we want to show that the self-attention mechanism is equivalent to the composition of these functions:

$$\begin{tikzcd} \mathbb{R}^{n\times k} \arrow[r, "id\times\mathbf{choose}"] \arrow[rr, bend right, "\mathscr{T}"'] & \mathbb{R}^{n\times k} \times \hom(\mathbb{R}^{n\times k}, \mathbb{R}^{n\times k'}) \arrow[r, "\mathbf{eval}"] & \mathbb{R}^{n\times k'}. \end{tikzcd}$$

This achieved by first noticing that:

$$\begin{aligned} vec(\mathscr{T}(X)) =& vec(AXW_V) \\ &= vec\left(([W_K \otimes W_Q vec(X^T X)]^{(n,m)})^\star X W_V\right) \\ &= W_V^T \otimes \left([W_K \otimes W_Q vec(X^T X)]^{(n,m)}\right)^\star vec(X) \end{aligned}$$

where $[-]^{(n,m)}$ represents the refolding of the vector into an (n,m) matrix using the vector C-like convention, where last index changes fastest, $(-)^\star$ represents the softmax function and $\otimes$ is the Kronecker product. The result shows how the transformer layer can be represented as a linear function, where the coefficients are computed as a function

$$\mathbf{choose}(x) = W_V^T \otimes \left([W_k \otimes W_Q \cdot vec(x^T x)]^{(n,m)}\right)^\star vec(-),$$

which outputs a function. Then, the canonical evaluation function is given by:

$$\mathbf{eval}(X, f) = f(X),$$

which compose together, resulting in:

$$\mathbf{eval}(X, \mathbf{choose}(X)) = \mathscr{T}(X).$$

□

The significance of this assertion is that a transformer, and any architecture involving the attention mechanism as an operator, selects an architecture and evaluates it. Indeed, it does so by determining the weights of an operator to apply on the left ('like' a graph convolution [24]) to a neural network parametrised by W_V. This sheds light on how the attention self-mechanism is distinct from other types ReLU networks. The characterisation through the composition of **choose** and **eval** morphisms to obtain the following result.

Theorem 3.2 (Expressivity of Transformer Network). *A transformer network $\mathscr{T}$ is a morphisms in the topos given by the free cocompletion Σ**PL** of **PL**.*

Proof. The free cocompletion of a Category is the Yoneda embedding into the category of presheaves $Y : \mathbf{PL} \to \mathrm{PSh}(\mathbf{PL})$, which is a topos. This category has the exponential objects of **PL**, obtained as $\hom(\mathbb{R}^n, \hom(-, \mathbb{R}^m)) \cong \hom(\mathbb{R}^n, \mathbb{R}^m)$ by Yoneda's Lemma. Hence, we extend the domain of the aforementioned inclusion from **PL** to Σ**PL**. Furthermore, there are morphisms in Σ**PL** of type $\eta : H \to \hom(\mathbb{R}^n, \mathbb{R}^m)$. In particular, there are components $\eta =$ **choose** in Σ**PL** from Proposition 3.1, which are constructed by setting H as a constant functor $H : \mathbf{PL} \ni \mathbb{R}^n \mapsto \mathbb{R}^c \in \mathbf{Set}$. This entails that $\mathscr{T}$ is in Σ**PL**. □

The decomposition suggests the need for exponential and evaluation morphisms, which are found only in the completion of **ReLU**.

4 Internal Logic of Neural Networks

[41] first introduces the idea that we can build a language on the topos of learners. However, this language is built to be able to make statements about how the network learns and is being updated. [9] suggest that a *theory* can be developed for a given network. In this section, we explore these claims and provide both a theoretical construction on how to do develop theories for neural networks and relate this to the field of explainable artificial intelligence.

The formulation of logic via categories is a vast field (see [35] for an introduction). Under relatively weak assumptions (the existence of finite products) it is possible to reflect on computing logic in a category. Using this framework, a pretopos gives birth to a propositional language, whereas a topos can allow the interpretation of a higher-order logic. Indeed, a topos has a Joyal-Kripke semantics associated to it [41], which is a construction that allows us to interpret higher-order logic with the internal language of the topos. By internal language, we mean a correspondence between categorical constructions and logical propositions. In particular, we see objects as collections of elements of a given *type*, Subobjects (ismorphism classes of monomorphisms) are viewed as propositions; in particular, for $\phi : A \to B$, B is understood to be the subobject of elements for which ϕ is true. Products and coproducts are interpreted as conjunction and disjunction respectively.

Because category **PL** of piecewise linear functions form a pretopos [40], we can see how each morphism generates an instance of propositional logic below.

Definition 3 (Language of a Network)**.** *Let $\mathscr{N} : \mathbb{R}^n \to \mathbb{R}^m$ be a neural network in* ***ReLU****. The* ***language of a network*** *is a signature Σ given by:*

- *Sorts $Sorts(\Sigma)$, given by the inputs $X_1, X_2, ..., X_n$ and the output Y,*
- *Function symbols $Fun(\Sigma)$, given by the linear functions, $f_\omega : X_1 \times X_2 \times ... \times X_n \to Y$, with associate coefficients $\alpha_\omega, \beta_\omega$,*
- *Relation symbols $Rel(\Sigma)$, with $r_{P_i^{(l)}} : R \to X_1 \times X_2 \times ... \times X_n$, for all variables, the relation symbol implement half space conditions that are represented by neurons; for example $r_{P_i^{(l)}} = 1 \iff P_i^{(l)} = 1$, the activation vector in entry $i \in \{1, .., n_l\}$.*

Consequently, we can think of networks as implementing a logical *theory* in the input space. We can think of each part of the partition as being represented by an equivalence class under the relation symbols. Therefore, statments can be of the form: for $x_1 \in X_1, ...x_n \in X_n$, if $r_{P_1^{(1)}}$ is true/fase, $r_{P_2^{(1)}}$ is true/false... $r_{P_{n_L}^{(L)}}$ is true/false such that the truth and falsity of each relation matches with the pattern for the polyhedron ω, then $\mathscr{N}$ implements f_ω. This construction enhances the explainability of the neural network. We can now think of the network as a collection of statements about the terms, which can be simplified as: whenever $x \in \mathbb{R}^n$ is in region ω the network $\mathscr{N}$ behaves as $f_\omega : x \mapsto \alpha_\omega x + \beta_\omega$.

Notice that for the transformer network, the partition changes at every point, and so does the linear function. Because the attention mechanism picks an architecture on which we evaluate the input, the logic of the theory is *dependent on the input*. From this point of view we are able to distinguish the key semantic difference between a feedforward architecture and a transformer architecture. Suppose that for a dataset $f(x) = x^2$ if $x > 0$ and $f(x) = -x$ if $x \leq 0$. The standard architecture can learn one function, defined piecewise. The transformer on the other hand, can learn two functions $f_1(x) = x^2$ and $f_2(x) = -x$, and learn a rule to select f_1 if $x > 0$, and f_2 if $x \leq 0$. We argue that this is the notion of *context* presented in [48] in the **choose** morphism. In this sense, we can interpret the choice of architecture parameters as a *bound variable*.

5 Architecture Search and Gradient Descent

In this section, we explore how transformers can be understood to be implemented gradient descent, as authors in the literature claim [48]. In short, this can be derived by viewing gradient descent as a particular type of **choose** function.

To analyse the phenomenon we use actegories. Actegories are instances of higher-categorical structures describing the action of one monoidal category on a second category. They have been used to study the behaviour of learning agents in [14].

Definition 4 (**Actegory**). *Let M be a monoidal category. The M-actegory on C is given by the action of a functor $-\bullet =: M \times C \to C$ which preserves the monoid actions on the homsets of the category C. Specifically, two natural isomorphisms $\delta : I \bullet X \cong X$,*
$\varepsilon : (K \otimes N) \bullet X \cong K \bullet (N \bullet X)$ satisfying the following commuting diagrams:

$$\begin{tikzcd}
((X \otimes Y) \otimes Z) \cdot D \arrow[r, "\delta"] \arrow[dd, "\textit{monoid associativity}"] & (X \otimes Y) \bullet (Z \bullet D) \arrow[dr, "\delta"] & \\
& & X \bullet (Y \bullet (Z \bullet D)) \\
(X \otimes (Y \otimes Z)) \bullet D \arrow[r, "\delta"] & X \bullet (Y \otimes Z) \bullet D \arrow[ur, "1\bullet\delta"] &
\end{tikzcd}$$

and,

$$\begin{tikzcd}
(I \otimes X) \bullet D \arrow[r, "\delta"] \arrow[d, "\textit{monoid identity}"'] & I \bullet (X \bullet D) \arrow[dl, "\varepsilon"] \\
X \bullet D & X \bullet D \\
(X \otimes I) \bullet D \arrow[ur, "\textit{monoid identity}"] \arrow[r, "\delta"'] & X \bullet (I \bullet D) \arrow[u, "\varepsilon"']
\end{tikzcd}$$

all commute.

Actegories are the setting in which [14] define parametric learners. In essence, the structure of the monoidal category is used as setting to define reparametrisations of the morphisms of C. Below we describe the **Para** construction on an M-actegory, which serves as a setting to reason about parametrised functions, their application and reparametrization. This is a particular instance of a bicategory, the which is a category weakly enriched in the category **Cat**: in which every hom object is a category. For a formal definition, please refer to [27].

Definition 5 (**Para Construction** [14]). *For a $(M, I, \otimes)$-actegory C, the bicategory of parametrisations on C by the action of M, $\textbf{Para}_\bullet(C)$, is given:*

- ***Objects:*** *the same objects as C;*
- ***1-Cells:*** *morphisms $\mathscr{N} : \theta \times X \to Y$, for all functions $f : X \to Y$ in C, and $\theta \in Ob(M)$;*
- ***2-Cells:*** *for a $r : \theta' \to \theta$ morphism in M we have the 2-cell $\rho : \phi \implies \phi'$ with $\phi : \theta \bullet X \to Y$ and*

$\phi' : \theta' \bullet X \to Y$ such that the following commuting diagram is respected;

$$\begin{tikzcd}
\theta \bullet X \arrow[rr, "\phi"] \arrow[drr, Rightarrow, "\rho", shorten >=20pt] & & Y \\
& & \\
\theta' \bullet X \arrow[uu, "r \bullet id"] \arrow[uurr, "\phi'"'] & &
\end{tikzcd}$$

- ***1-Composites:** for $\phi : N \bullet X \to Y$, and $\psi : K \bullet Y \to Z$ we define the composite through:*

$$\psi \circ \phi = (K \otimes N) \bullet X \to^{\delta} K \bullet (N \bullet X) \to^{id_K \bullet \phi} \to K \bullet Y \to \psi Z$$

- ***2-Composites:** $\rho_1 : \phi \implies \phi'$ and $\rho_2 : \phi' \implies \phi''$ are inherited by the composition in M: $\rho_2 \circ \rho_1 : \phi \implies \phi''$.*

$\mathbf{Para}_\bullet(\mathbf{Lens}(C))$ is the category of parametrised lenses on C, which in [14] is understood to represent *learners*. The requirements on C is that it is a reverse differential category. The overall goal of this formalism is to represent models with parameters that can be updated. Both the forward pass and the update protocol are compositional processes, in the sense that we can form a category of said learners. In the present exposition we will focus on *architectural* aspects of the parametrisation.

The authors of [14] develop a theory of cybernetic agents that unifies constructs within game theory and deep learning: by parametrising over **Smooth**, but without specifying the parametrisation. Again, we choose to parametrise over **PL**, and choose $\mathbf{Vect}_{\mathbb{R}}$ with $\otimes$, the usual tensor product, as the action monoidal actegory. It is important to note that the choice of 1-cells matters: when viewing **Para** as a functor, we should associate the $\mathscr{N}$ morphisms in a way that reflects the usual *learnability* of parameters in neural networks. In other words, we want the parameters θ to be the direct sum of the coefficients of the linear layers: if $f = f_1 \circ \mathrm{ReLU} \circ f_2 \circ ... \circ \mathrm{ReLU} \circ f_{L+1}$, with $f_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}} \in \mathrm{mor}(\mathbf{Vect}_{\mathbb{R}})$ then $\theta \in \bigoplus_{i=1,...,L} \mathbb{R}^{n_i \times n_{i+1}}$. Indeed, $\mathbf{Para}_{\otimes}(\mathbf{PL})$ is a $\mathbf{Vect}_{\mathbb{R}}$-actegory.

A corollary of Theorem 3.2 is that there is a Boolean algebra for every feedforward architecture, and a *collection* of Boolean algebras for every transformer architecture, parametrised by the input domain. See, for instance, [46], who first suggest that ReLU networks induce a Boolean algebra in their domain of definition. This can be viewed as a functor from the category of morphisms of piece-wise linear functions, into **Bool**, which we write as: $B : \mathbf{PL}^{\to} \to \mathbf{Bool}$. However, we've seen that the transformer does not live in **PL** generally, but in $\Sigma\mathbf{PL}$. Then the image of $\mathbf{choose} : \mathbb{R}^n \to \hom(\mathbb{R}^n, \mathbb{R}^m)$ induces a collection of Boolean algebras $S = \{B(\mathscr{N}) : \mathscr{N} \in \mathrm{cod}(\mathbf{choose})\}$ via lifting the B functor above to the category of morphisms in $\Sigma\mathbf{PL}$. Note that **choose** is not in **PL**. This prompts the need for the following construction.

Proposition 5.1. *Let $\iota : \mathbf{PL} \to \mathbf{Para}_{\otimes}(\mathbf{PL})$ be an inclusion functor, and $\Sigma\mathbf{PL}$ be the free cocompletion of $\mathbf{PL}$. Then there exists a left Kan extension $\mathrm{Lan}_Y \iota$; in particular, functor maps **choose** morphisms in $\Sigma\mathbf{PL}$ to choice of parameter spaces and $\hom(\mathbb{R}^n, \mathbb{R}^m)$ to reparameterisations of neural networks.*

Proof. Notice that $\mathbf{Vect}_{\mathbb{R}} \times \mathbf{PL} \hookrightarrow \mathbf{Para}_{\otimes}(\mathbf{PL})$ is a faithful functor, which follows from the definition of **Para**. Hence, we define functor $\iota : \mathbf{PL} \hookrightarrow \mathbf{Para}_{\otimes}(\mathbf{PL})$ that embeds the category into 1-cells and 1-composites in a bicategory of parametric piece-wise linear functions. The left Kan extension is visualized by the commuting diagram below.

$$\begin{tikzcd}
\mathbf{PL} \arrow[rr, "\iota"] \arrow[dr, "Y"'] & \arrow[d, Rightarrow, "\eta"] & \mathbf{Para}_{\otimes}(\mathbf{PL}) \\
& \Sigma\mathbf{PL} \arrow[ur, dashed, "\mathrm{Lan}_Y \iota"'] &
\end{tikzcd}$$

$\mathrm{Lan}_Y \iota$ is a functor that maps all $Y(\mathbf{PL})$ into $\iota(\mathbf{Para}_\otimes(\mathbf{PL}))$, meaning that for any morphisms f in $\mathbf{PL}$, we have that $\mathrm{Lan}_Y \iota(Y(f)) = \iota(f)$. There are 1-morphisms in $\mathbf{Para}_\otimes(\mathbf{PL})$ that are not in $\iota(\mathbf{PL})$; these are product morphisms of the monoid action r with the identity morphism, given by $r \times id_{\mathbb{R}^n} : \theta \times \mathbb{R}^n \to \theta' \times \mathbb{R}^n$, which can be viewed as reparametrisation of the neural network. Lan_Y maps morphisms in $\hom(\hom(\mathbb{R}^n, -), \hom(\mathbb{R}^n, -))$ into said reparametrisations, such that the following diagram commutes.

$$\begin{tikzcd}
\hom(\mathbb{R}^n, -) \arrow[r, "\mathrm{Lan}_Y \iota(-)"] \arrow[d, "\rho_{\mathbb{R}^m}"'] & \hom(\theta \times \mathbb{R}^n, -) \arrow[d, "\mathrm{Lan}_Y \iota(\rho_{\mathbb{R}^m})"] \\
\hom(\mathbb{R}^n, \mathbb{R}^m) \arrow[r, "\mathrm{Lan}_Y \iota(-)"] \arrow[d, "R"'] & \hom(\theta \times \mathbb{R}^n, \mathbb{R}^m) \arrow[d, "\mathrm{Lan}_Y \iota(R) \cong r \times id_{\mathbb{R}^n}"] \\
\hom(\mathbb{R}^n, \mathbb{R}^m) \arrow[r, "\mathrm{Lan}_Y \iota(-)"'] & \hom(\theta' \times \mathbb{R}^n, \mathbb{R}^m)
\end{tikzcd}$$

Without loss of generality, we consider the case $\theta = \mathbb{R}^k$. See for instance that $\mathbf{choose} \in \hom(\mathbb{R}^k, \hom(\mathbb{R}^n, \mathbb{R}^m))$ and the left Kan extension provides $\mathrm{Lan}_Y \iota \left(\hom(\mathbb{R}^k, \hom(\mathbb{R}^n, -)\right) \cong \hom(\mathbb{R}^n, \mathbb{R}^k \times \mathbb{R}^n)$, a mapping into choices of parameters space for a neural network in $\mathbf{Para}_\otimes(\mathbf{PL})$ and $\mathrm{Lan}_Y \iota \left(\hom(\hom(\mathbb{R}^k, \mathbb{R}^{k'}), \hom(\mathbb{R}^n, -)\right) \cong \hom\left(\hom(\mathbb{R}^k \times \mathbb{R}^n, , \mathbb{R}^{k'} \times \mathbb{R}^n)\right)$, which is the class of reparametrisations $r \times id : \mathbb{R}^k \times \mathbb{R}^n \to \mathbb{R}^{k'} \times \mathbb{R}^n$.

□

In other words, the 2-cells of this categories can be viewed both as reparametrisations, whenever $r : \theta \to \theta \in \mathrm{mor}(\mathbf{Vect}_\mathbb{R})$ is an endomorphism, and morphisms of architectures when the morphism is of general type $r : \theta \to \theta'$. This induces in the Para construction the 2-cell $\rho : \phi' \implies \phi$ for $\phi : \theta \times X \to Y$ and $\phi' : \theta' \times X \to Y$, where $X, Y, \theta, \theta' \in \mathrm{ob}(\mathbf{Para}_\otimes(\mathbf{PL}))$. This verifies the claims from [48], who argue that we can view forward passes of a transformer as gradient updates.

Ultimately, this allows us to view transformers as a parametrised collection of neural networks; the network weights are the function outputs of a $\mathbf{choose}(-)$ function, which determines entirely the architecture. This observation paves the way to questions on which networks are being selected by the transformer architecture, what do they have in common, how far they are from each other in a suitable distance, and other questions which have meaningful repercussions in the design and training of these architectures.

6 Conclusion

We have introduced a theoretical analysis of neural network architectures from the viewpoint of topos theory, noticing a previously unknwon distinction between two classes of objects: feedforward architectures and transformer architecture. In the process of proving this distinction, we have divided architectures into two classes: those that live in a pretopos and those that live in its topos completion. This distinction enabled us to relate architectures in the topos completion to backpropagation and architecture search, encoding many known neural architectures within the context of a single design space.

The categorical treatment of architectures has several advantages. Firstly, it allows to equate different families of architectures, enabling a pragmatic view on which differences are effectively influencing the expressiveness of the architectural class. Indeed, through the notion of the topos we introduce a new notion of expressivity, based on which fragment of logic the architecture class is being denominated in.

Moreover, this paper may provide insight on how to construct architectures that are more expressive in the above sense. The transformer is just an example of a function living in the completion of **PL**. The significance of this paper is that it conjectures how in practice the factorisation through **choose** and **eval** morphisms can be a successful architecture design. Further empirical research could confirm that this is an architectural choice able to generally improve the performance of architectures across tasks.

References

[1] Amina Adadi & Mohammed Berrada (2018): *Peeking inside the black-box: a survey on explainable artificial intelligence (XAI)*. *IEEE access* 6, pp. 52138–52160.

[2] Jiří Adámek & Jiří Rosickỳ (2020): *How nice are free completions of categories?* *Topology and its Applications* 273, p. 106972.

[3] Adebowale Jeremy Adetayo, Mariam Oyinda Aborisade & Basheer Abiodun Sanni (2024): *Microsoft Copilot and Anthropic Claude AI in education and library service*. *Library Hi Tech News*.

[4] Raman Arora, Amitabh Basu, Poorya Mianjy & Anirbit Mukherjee (2016): *Understanding deep neural networks with rectified linear units*. *arXiv preprint arXiv:1611.01491*.

[5] Caglar Aytekin (2022): *Neural Networks are Decision Trees*. *arXiv preprint arXiv:2210.05189*.

[6] Randall Balestriero et al. (2018): *A spline theory of deep learning*. In: *International Conference on Machine Learning*, PMLR, pp. 374–383.

[7] Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, Michael Bronstein, Petar Veličković & Pietro Liò (2022): *Sheaf Neural Networks with Connection Laplacians*. *Topological, Algebraic and Geometric Learning Workshops 2022*.

[8] Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde & Pietro Lio (2022): *Sheaf attention networks*. *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*.

[9] Jean-Claude Belfiore & Daniel Bennequin (2021): *Topos and stacks of deep neural networks*. *arXiv preprint arXiv:2106.14587*.

[10] Cristian Bodnar, Francesco Di Giovanni, Benjamin Paul Chamberlain, Pietro Liò & Michael M Bronstein (2022): *Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns*. *arXiv preprint arXiv:2202.04579*.

[11] Guillaume Boisseau & Robin Piedeleu (2022): *Graphical piecewise-linear algebra*. In: *Foundations of Software Science and Computation Structures: 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings*, Springer International Publishing Cham, pp. 101–119.

[12] Michael M Bronstein, Joan Bruna, Taco Cohen & Petar Veličković (2021): *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges*. *arXiv preprint arXiv:2104.13478*.

[13] Ruth MJ Byrne (2019): *Counterfactuals in Explainable Artificial Intelligence (XAI): Evidence from Human Reasoning*. In: *IJCAI*, pp. 6276–6282.

[14] Matteo Capucci, Bruno Gavranović, Jules Hedges & Eigil Fjeldgren Rischel (2021): *Towards foundations of categorical cybernetics*. *arXiv preprint arXiv:2105.06332*.

[15] Geoffrey SH Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson & Fabio Zanasi (2022): *Categorical foundations of gradient-based learning*. In: *Programming Languages and Systems: 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice*

of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Springer International Publishing Cham, pp. 1–28.

[16] Glenn De'Ath (2002): *Multivariate regression trees: a new technique for modeling species–environment relationships*. *Ecology* 83(4), pp. 1105–1117.

[17] Andrew Dudzik & Petar Veličković (2022): *Graph neural networks are dynamic programmers*. *arXiv preprint arXiv:2203.15544*.

[18] Brendan Fong, David Spivak & Rémy Tuyéras (2019): *Backprop as functor: A compositional perspective on supervised learning*. In: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, IEEE, pp. 1–13.

[19] Bruno Gavranović, Paul Lessard, Andrew Dudzik, Tamara von Glehn, João GM Araújo & Petar Veličković (2024): *Categorical Deep Learning: An Algebraic Theory of Architectures*. *arXiv preprint arXiv:2402.15332*.

[20] Pim de Haan, Taco S Cohen & Max Welling (2020): *Natural graph networks*. *Advances in neural information processing systems* 33, pp. 3636–3646.

[21] Juncai He, Lin Li, Jinchao Xu & Chunyue Zheng (2018): *ReLU deep neural networks and linear finite elements*. *arXiv preprint arXiv:1807.03973*.

[22] Christoph Hertrich, Amitabh Basu, Marco Di Summa & Martin Skutella (2021): *Towards lower bounds on the depth of ReLU neural networks*. *Advances in Neural Information Processing Systems* 34, pp. 3336–3348.

[23] Sepp Hochreiter & Jürgen Schmidhuber (1997): *Long short-term memory*. *Neural computation* 9(8), pp. 1735–1780.

[24] Thomas N Kipf & Max Welling (2016): *Semi-supervised classification with graph convolutional networks*. *arXiv preprint arXiv:1609.02907*.

[25] Christoph Koutschan, Bernhard Moser, Anton Ponomarchuk & Josef Schicho (2023): *Representing piecewise linear functions by functions with small arity*. *Applicable Algebra in Engineering, Communication and Computing*, pp. 1–16.

[26] Yann LeCun, Léon Bottou, Yoshua Bengio & Patrick Haffner (1998): *Gradient-based learning applied to document recognition*. *Proceedings of the IEEE* 86(11), pp. 2278–2324.

[27] Tom Leinster (2004): *Higher operads, higher categories*. 298, Cambridge University Press.

[28] Tom Leinster (2016): *Basic category theory*. *arXiv preprint arXiv:1612.09375*.

[29] Scott M Lundberg & Su-In Lee (2017): *A unified approach to interpreting model predictions*. *Advances in neural information processing systems* 30.

[30] Minh-Thang Luong, Hieu Pham & Christopher D Manning (2015): *Effective approaches to attention-based neural machine translation*. *arXiv preprint arXiv:1508.04025*.

[31] Yoshihiro Maruyama (2022): *Categorical artificial intelligence: the integration of symbolic and statistical AI for verifiable, ethical, and trustworthy AI*. In: *Artificial General Intelligence: 14th International Conference, AGI 2021, Palo Alto, CA, USA, October 15–18, 2021, Proceedings 14*, Springer, pp. 127–138.

[32] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho & Yoshua Bengio (2014): *On the number of linear regions of deep neural networks*. *Advances in neural information processing systems* 27.

[33] Michael Moy, Robert Cardona & Alan Hylton (2023): *Categories of Neural Networks*. In: *2023 IEEE Cognitive Communications for Aerospace Applications Workshop (CCAAW)*, IEEE, pp. 1–9.

[34] Ian E Nielsen, Dimah Dera, Ghulam Rasool, Ravi P Ramachandran & Nidhal Carla Bouaynaya (2022): *Robust explainability: A tutorial on gradient-based attribution methods for deep neural networks*. *IEEE Signal Processing Magazine* 39(4), pp. 73–84.

[35] Andrew M Pitts (2001): *Categorical logic*. *Handbook of logic in computer science* 5, pp. 39–128.

[36] Marco Tulio Ribeiro, Sameer Singh & Carlos Guestrin (2016): *Model-agnostic interpretability of machine learning*. *arXiv preprint arXiv:1606.05386*.

[37] David E Rumelhart, Geoffrey E Hinton & Ronald J Williams (1986): *Learning representations by back-propagating errors.* nature 323(6088), pp. 533–536.

[38] Maximilian Schlüter, Gerrit Nolte, Alnis Murtovi & Bernhard Steffen (2023): *Towards rigorous understanding of neural networks via semantics-preserving transformations.* *arXiv preprint arXiv:2301.08013.*

[39] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh & Dhruv Batra (2017): *Grad-cam: Visual explanations from deep networks via gradient-based localization.* In: Proceedings of the IEEE international conference on computer vision, pp. 618–626.

[40] Eduardo Sontag (1982): *Remarks on piecewise-linear algebra.* Pacific Journal of Mathematics 98(1), pp. 183–201.

[41] David I Spivak (2021): *Learners' Languages.* *arXiv preprint arXiv:2103.01189.*

[42] David I Spivak & Timothy Hosgood (2021): *Deep neural networks as nested dynamical systems.* *arXiv preprint arXiv:2111.01297.*

[43] Agus Sudjianto, William Knauth, Rahul Singh, Zebin Yang & Aijun Zhang (2020): *Unwrapping the black box of deep ReLU networks: interpretability, diagnostics, and simplification.* *arXiv preprint arXiv:2011.04041.*

[44] Yuandong Tian, Tina Jiang, Qucheng Gong & Ari Morcos (2019): *Luck matters: Understanding training dynamics of deep relu networks.* *arXiv preprint arXiv:1905.13405.*

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser & Illia Polosukhin (2017): *Attention is all you need.* Advances in neural information processing systems 30.

[46] Mattia Jacopo Villani & Peter McBurney (2023): *Unwrapping All ReLU Networks.* arXiv:2305.09424.

[47] Mattia Jacopo Villani & Nandi Schoots (2023): *Any Deep ReLU Network is Shallow.* *arXiv preprint arXiv:2306.11827.*

[48] Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov & Max Vladymyrov (2023): *Transformers learn in-context by gradient descent.* In: International Conference on Machine Learning, PMLR, pp. 35151–35174.

[49] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han & Yang Tang (2023): *A brief overview of ChatGPT: The history, status quo and potential future development.* IEEE/CAA Journal of Automatica Sinica 10(5), pp. 1122–1136.

[50] Shaojie Xu, Joel Vaughan, Jie Chen, Aijun Zhang & Agus Sudjianto (2022): *Traversing the Local Polytopes of ReLU Neural Networks.* *The AAAI-22 Workshop on Adversarial Machine Learning and Beyond.*

[51] Erik Christopher Zeeman (1963): *Seminar on combinatorial topology.* Institut des hautes etudes scientifiques.