To my beloved
Benedetta

Introduction

English version

Machine learning literature is exploding in size and complexity, but most solutions found are ad hoc, there is little communication between different subfields, and there is a large research debt. Category theory can solve these problems. [SGW21].

Talk about the origins of category theory and its "rise to power" as a common language that aims to unite different fields of knowledge.

Discuss the purpose of this work: a beginner-friendly survey of categorical approaches to neural networks, causal models, and interpretability.

Italian version

Traduzione italiana dell'introduzione.

Contents

Chapter 1

Categorical Toolkit

In this chapter, we illustrate a few categorical tools that have been developed to provide compositional frameworks for machine learning. In particular, we will focus on parametric weighted optics and compositional models. Although such frameworks have been developed separately, they do interact, and they share a common philosophy: their purpose is to unify the field of machine learning and replace unreliable, non-compositional, *ad hoc* tools with reliable, compositional, elegant tools.

We are forced by space and time constraints to skip over the details of many constructions, but we hope the reader can get a good picture of how the proposed tools work. All the necessary details can be found in the cited references.

Note that this chapter is devoted to illustrating our categorical toolkit itself, not its applications. Hence, we will focus on the mathematical and categorical aspects of the presented frameworks, and we will postpone applications to the following chapters.

1.1 Parametric and differential categories

Neural networks have two main properties: they depend on parameters and information flows through them bidirectionally (forward propagation and back propagation). Any aspiring categorical model of neural networks must take these two aspects into consideration: the framework of parametric weighted optics (see [Gav24] for a thorough and mostly novel treatment of the subject and see [CGG$^+$22] for experimental results) does just that, as it offers bidirectionality using weighted optics and parametrization using the **Para** construction. In this section, we will sketch out the construction of this framework loosely following [Gav24].

1.1.1 Parametric categories

In this section, we will focus on parametric and coparametric categories. While the use case for parameters is obvious to any machine learning practitioner, one might ask why coparameters are even useful. As we will see, coparameters will come very handy in building parametric weighted optics.

Actegories

Before we can deal with (co)parametric maps, we need to find a way to glue input/output spaces to parameter spaces, so that such maps have well-defined (co)domains. One common strategy is to provide the category at hand with a monoidal structure. However, monoidal products can only combine elements within the same underlying category. Since (co)parameters are often taken from spaces that are different in nature from the input and output spaces, a more general mathematical tool is needed: namely, actegories (see the survey [CG22] for a thorough treatment of the subject).

Actegories are actions of symmetric monoidal categories on other categories. For brevity's sake, we will only give an incomplete definition (see [CG22] or [Gav24] for complete definitions).

Definition 1 (Actegory)**.** *Let $(\mathcal{M}, I, \otimes)$ be a monoidal category. A (right) $\mathcal{M}$-actegory is a tuple $(\mathcal{C}, \bullet, \eta, \mu)$, where:*

1. *$\mathcal{C}$ is a category;*
2. *$\bullet : \mathcal{C} \times \mathcal{M} \to \mathcal{C}$ is a functor;*
3. *η and μ are natural isomorphisms satisfying the diagrams in ADDIM.*

The natural isomorphisms η and μ must also satisfy coherence conditions. If η and μ are identical transformations, we say that the actegory is strict.

Thus, if we want a map $A \to B$ with parameters in P, we will find an appropriate actegory and an appropriate map $A \bullet P \to B$.

Remark 2. The definition of actegory given above technically pertains right actegories: left actegories are defined analogously.

We will also be interested in actegories that interact with the monoidal structure of the underlying category.

Definition 3 (Monoidal actegory)**.** *Let $(\mathcal{M}, I, \otimes)$ be a braided monoidal category and let $(\mathcal{C}, \bullet, \eta, \mu)$ be an actegory. Suppose $\mathcal{C}$ has a monoidal structure $(J, \boxtimes)$. Then we say that $(\mathcal{C}, \bullet)$ is monoidal if the underlying function $\bullet$ is strong monoidal and the transformation η and μ are also monoidal.*

We may also be interested in studying the interaction between actegorical structures and endofunctors. This interaction can happen owing to a natural transformation known as strength. We will not provide coherence diagrams in the definition below for the sake of brevity, but the interested reader can find more detail in [GLD+24b]. The paper also provides a definition of actegorical strong monad, which is a very similar concept.

Definition 4 (Actegorical strong functor)**.** *Let $(\mathcal{C}, \bullet)$ be an $\mathcal{M}$-actegory. A strong actegorical endofunctor on $(\mathcal{C}, \bullet)$ is a pair (F, σ) where $F : \mathcal{C} \to \mathcal{C}$ is an endofunctor and σ is a natural transformation with components $\sigma_{P,A} : P \bullet F(A) \to F(P \bullet A)$ which satisfies a few coherence conditions that we don't list here.*

The Para construction

Suppose we have an $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$. We wish to study maps in $\mathcal{C}$ which are parametrized using objects of $\mathcal{M}$. We are not just interested in the maps by themselves, but also in their compositional structure. Thus, we abstract away the details by defining a new category $\mathbf{Para}_\bullet(\mathcal{C})$ (first defined in [FST19]). Since we also want to formalize the role of reparametrization, we actually construct $\mathbf{Para}_\bullet(\mathcal{C})$ as a bicategory, so that its 0-cells can serve as input/output spaces, and its 1-cells can serve as parametric maps, and, finally, its 2-cells can serve as parameter changes.

Definition 5 ($\mathbf{Para}_\bullet(\mathcal{C})$)**.** *Let $(\mathcal{C}, \bullet)$ be a right $\mathcal{M}$-actegory. Then, we define $\mathbf{Para}_\bullet(\mathcal{C})$ as the bicategory whose components are as follows.*

- *The 0-cells are the objects of $\mathcal{C}$.*
- *The 1-cells are pairs $(P, f) : A \to B$, where $P : \mathcal{C}$ and $f : A \bullet P \to B$.*
- *The 2-cells come in the form $r : (P, f) \Rightarrow (Q, g)$, where $r : P \to Q$ is a morphism in $\mathcal{C}$. r must also satisfy a naturality condition.*
- *The 1-cell composition law is*

$$(P, f) \fatsemi (Q, g) = (P \otimes Q, f \bullet Q \fatsemi g).$$

- *The horizontal and vertical 2-cell composition laws are respectively given by parallel and sequential products in $\mathcal{M}$.*

It is quite handy to represent such cells using the string diagram notation illustrated in figure ADDIM.

It is shown in [Gav24] that $\mathbf{Para}_\bullet(\mathcal{C})$ is actually a 2-category if the underlying actegory is strict. Assuming this is the case, we can use a functor $F : \mathbf{Cat} \to \mathbf{Set}$ to quotient out the 2-categorical structure and turn $\mathbf{Para}_\bullet(\mathcal{C})$ into a 1-category $F_*(\mathbf{Para}_\bullet(\mathcal{C}))$. Here, $F_* : \mathbf{2Cat} \to \mathbf{Set}$ is the change of enrichment basis functor induced by F. We will see the significance of this construction later.

The **Para** construction has a dual **coPara** construction which is designed to represent coparametric morphisms. The main differences between the two constructions are that $(\mathcal{C}, \bullet)$ is required to be a left actegory this time, and that 1-cells $A \to B$ in $\mathbf{coPara}_\bullet(\mathcal{C})$ take the form (P, f), where $f : A \to P \bullet B$. Cells in $\mathbf{coPara}_\bullet(\mathcal{C})$ can also be represented with appropriate string diagrams (see ADDIM). The reader can find a complete definition in [Gav24].

Both $\mathbf{Para}_\bullet(\mathcal{C})$ and $\mathbf{coPara}_\bullet(\mathcal{C})$ can be given a monoidal structure if $(\mathcal{C}, \bullet)$ is a monoidal actegory. This is extremely important because it allows us to compose (co)parametric morphisms both in sequence and in parallel. Once again, more detail can be found in [Gav24].

Remark 6. Another way to parametrize morphisms is the coKleinsli construction. As noted by [Gav24], the main difference between **coKl** and **Para** is that the parametrization offered by **coKl** is global, while the parametrization offered by **Para** is local: all morphisms in $\mathbf{coKl}(X \times -)$ must take a parameter in X, while the parameter space of different morphisms of $\mathbf{Para}(\mathcal{C})$ admit different parameter spaces. Nevertheless, the two constructions are relate, and the former can be embedded into the latter.

If we take a parametrized category $\mathbf{Para}_\bullet(\mathcal{C})$ and we restrict our attention to morphisms parametrized with the monoidal identity I, we get back the original category $\mathcal{C}$. This is expressed by the following proposition ([Gav24]).

Proposition 7. *Let $(\mathcal{C}, \bullet)$ be an $\mathcal{M}$-actegory. Then, there exists an identity-on-objects pseudofunctor $\gamma : \mathcal{C} \to \mathbf{Para}_\bullet(\mathcal{C})$ that maps $f \mapsto (I, f)$. If $\mathcal{M}$ is strict, this is a 2-functor.*

1.1.2 Optics

Modelling bidirectional flows of information is not only useful in machine learning, but also in game theory, database theory, and more. As such, categorical tools for bidirectionality have been sought after for a long time: in particular, the greatest deal of efforts has been devoted to developing lens theory. Lenses have then been generalized into optics (see e.g. [Ril18]) to subsume other tools such as prisms and traversals into the same framework. Finally, there have also been various attempts at generalizing optics (see e.g. [CEG$^+$24] for a definition of mixed optics). We will introduce lenses and optics, and focus on the generalization of optics that appears (to us) to be the most versatile: weighted optics (first introduced in [Gav24]).

Lenses

As stated in [Gav24], there is no standard definition of lens, and different authors opt for different *ad hoc* definitions that best suit their purposes. We will borrow the perspective of [CGG+22] and give the following definition.

Definition 8 (Lenses)**.** *Let $\mathcal{C}$ be a Cartesian category. Then,* $\mathbf{Lens}(\mathcal{C})$ *is the category defined by the following data:*

- *an object of* $\mathbf{Lens}(\mathcal{C})$ *is a pair* $\binom{A}{A'}$ *of objects in* $\mathcal{C}$*;*
- *a* $\binom{A}{A'} \to \binom{B}{B'}$ *morphism (or lens) is a pair* $\binom{f}{f'}$ *of morphisms of* $\mathcal{C}$ *such that* $f : A \to B$ *and* $f' : A \times B' \to A'$*. f is known as the forward pass of the lens* $\binom{f}{f'}$*, whereas f' is known as the bacward pass;*
- *given* $\binom{A}{A'} : \mathbf{Lens}(\mathcal{C})$*, the associated identity lens is* $\binom{1_A}{\pi_1}$*;*
- *the composition of* $\binom{f}{f'} : \binom{A}{A'} \to \binom{B}{B'}$ *and* $\binom{g}{g'} : \binom{B}{B'} \to \binom{C}{C'}$ *is*

$$\binom{f \fatsemi g}{\langle \pi_0, \langle \pi_0 \fatsemi f, \pi_1 \rangle \fatsemi g' \rangle \fatsemi f'}.$$

Lenses are best thought of in the helpful language of the string diagrams illustrated in ADDIM. For instance, the composition formula in the definition above looks very complicated, but its graphical counterpart shown in ADDIM is extremely intuitive.

As we will soon see, in machine learning, the forward pass of a lens is used to model forward propagation, whereas the associated backward pass is tasked with modelling backpropagation. In other contexts, such as functional programming, lenses are used as data accessors for objects: the forward pass serves as a getter, while the backward pass serves as a setter (see e.g. [Ste15]).

Optics

Lenses are a powerful tool, but they cannot be used to model all situations: for instance, lenses cannot be used if we wish to be able to choose not to interact with the environment depending on the input, or if we wish like to reuse values computed in the forward pass for further computation in the backward pass.

Optics generalize lenses by weakening the link between GET maps and PUT maps and by replacing the Cartesian structure of the underlying category with a simpler symmetric monoidal structure. In an optic over $\mathcal{C}$, an object $M : \mathcal{C}$ acts as an inaccessible residual space transferring information between the upper components and the lower component. We will provide the definition given by [Ril18], but we will use the string diagram notation presented by [Gav24] for the sake of consistency.

Definition 9 (Optics)**.** *Let $(\mathcal{C}, I, \otimes, \lambda, \rho, \alpha)$ be a symmetric monoidal category (we make the unitors and associators explicit for later use). Then,* $\mathbf{Optic}(\mathcal{C})$ *is the category defined by the following data:*

- *an object of* $\mathbf{Optic}(\mathcal{C})$ *is a pair* $\binom{A}{A'}$ *of objects in* $\mathcal{C}$*;*
- *a* $\binom{A}{A'} \to \binom{B}{B'}$ *morphism (or optic) is a pair* $\binom{f}{f'}$ *of morphisms of* $\mathcal{C}$ *such that* $f : A \to M \otimes B$ *and* $f' : M \otimes B' \to A'$*, where* $M : \mathcal{C}$ *is known as residual space; such pairs* $\binom{f}{f'}$ *are also quotiented by an equivalence relation that allows for reparametrization of the residual space and effectively makes it inaccessible;*
- *the identity on* $\binom{A}{A'}$ *is the optic represented by* $(\lambda_A^{-1}, \lambda_A)$*.*

We will only show how optics compose using the string diagrams in ADDIM (see [Ril18] for the composition formula).

Remark 10. A more versatile but also more mathematically sophisticated definition of optics relies on coends. Under the coend formalism,

$$\mathbf{Optic}(\mathcal{C})\left(\binom{A}{A'}, \binom{B}{B'}\right) = \int^{M:\mathcal{C}} \mathcal{C}(A, M \times B) \times \mathcal{C}(M \otimes B', A').$$

See [Ril18] for more information about this topic.

Lenses come up as a special case of optics ([Ril18]), and optics do solve some of the issues we have with lenses. However, optics are not perfect either: for instance, [Gav24] points out that optics cannot be used in cases where we ask that the forward pass and backward pass are different kind of maps, as they are both forced to live in the same category. Thus, we need a further layer of generalization: namely, weighted optics.

Category of elements of a functor

Before we define weighted optics, we need to introduce a new tool to our toolbox: the category of elements of a functor.

Definition 11 (Elements of a functor)**.** *Let $F : \mathcal{C} \to \mathbf{Set}$ be a functor. We define* $\mathbf{El}(F)$ *as the category with the following data: (i) the objects of* $\mathbf{El}(F)$ *are pairs (C, x) where $C : \mathcal{C}$ and $x : F(C)$; (ii) the $(C, x) \to (D, y)$ morphisms in* $\mathbf{El}(F)$ *are the morphisms $f : C \to D$ in $\mathcal{C}$ such that $F(f)(x) = y$.*

[Gav24] studies $\mathcal{B}$-actegories $(\mathcal{C}, \bullet)$, which are then reparametrized so that the acting category becomes $\mathcal{E} = \mathbf{El}(W)$ for some weight functor $W : \mathcal{B}^{\mathrm{op}} \to \mathbf{Set}$ (which is to be specified). The reparametrization takes place thanks to the opposite of the forgetful functor $\pi_W : \mathcal{E} \to \mathcal{B}^{\mathrm{op}}$, which maps $(B, x) \mapsto B$. Hence, we consider the action

$$\bullet^{\pi_W^{\mathrm{op}}} = \mathbf{El}(W)^{\mathrm{op}} \times \mathcal{C} \xrightarrow{\pi_W^{\mathrm{op}} \times \mathcal{C}} \mathcal{B} \times \mathcal{C} \xrightarrow{\bullet} \mathcal{C}.$$

[Gav24] also gives the following definition, which is later instrumental in defining weighted lenses.

Definition 12 (Weighted **coPara**)**.** *If W is a weight functor as above and $(\mathcal{C}, \bullet)$ is a $\mathcal{B}$-actegory, we define*

$$\mathbf{coPara}_{\bullet}^{W}(\mathcal{C}) = \pi_{0*}(\mathbf{coPara}_{\bullet^{\pi_W^{\mathrm{op}}}}(\mathcal{C})),$$

where π_{0*} is the enrichment base change functor generated by the connected component functor $\pi_0 : \mathbf{Cat} \to \mathbf{Set}$. More explicitly, π_{0*} quotients the connections provided by reparametrizations.

Weighted optics

We are finally ready to define weighted optics.

Definition 13 (Weighted optics)**.** *Suppose $(\mathcal{C}, \bullet)$ is an $\mathcal{M}$-actegory, suppose $(\mathcal{D}, \blacksquare)$ is an $\mathcal{M}'$-actegory, and suppose $W : \mathcal{M}^{\mathrm{op}} \times \mathcal{M}' \to \mathbf{Set}$ is a lax monoidal functor. We define the category of W-weighted optics over the product actegory $(\mathcal{C} \times \mathcal{D}^{\mathrm{op}}, \binom{\bullet}{\blacksquare^{\mathrm{op}}})$ as*

$$\mathbf{Optic}_{\binom{\bullet}{\blacksquare}}^{W} = \mathbf{coPara}_{\binom{\bullet}{\blacksquare^{\mathrm{op}}}}^{W}(\mathcal{C} \times \mathcal{D}^{\mathrm{op}}).$$

The definition is very dense and deserves some explanation. Fist of all, we assume that W maps (M, M') to a set of maps $s : M \to M'$. If that's the case, a $\binom{X}{X'} \to \binom{Y}{Y'}$ map is a triplet $\left(\binom{\ }{M'}, s, \binom{f}{f'}\right)$, where M is the forward residual, M' is the backward residual, $s : M \to M'$ links the two residuals, $f : X \to M \bullet Y$ is the forward pass, and $f' : M' \bullet Y' \to X'$ is the backward pass. The triplets are also quotiented with respect to reparametrization, which makes the residual spaces effectively inaccessible (as it happens in the case of ordinary optics).

A weighted optic is a rather sophisticated piece of categorical machinery, but we can get a clear "functional" understanding of how it works looking at an associated string diagram: data from X flows through the forward map, which computes an output in Y and a forward residual in M. Such forward residual is the converted into a backward residual in M' by the map s, which is provided by the weight functor. Then, the backward residual is used to compute, together with input

from Y' a value in X'. The last step happens thanks to the backward map f'. The composition law is just as complicated as the optics, but it is not difficult to understand if considered from the point of view of string diagrams: see, for instance, ADDIM. More information can be found on [Gav24].

Remark 14. Weighted optics can also be defined using coend calculus: it is shown in [Gav24] that:

$$\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}}\begin{pmatrix} A & B \\ A', & B' \end{pmatrix} \cong \int^{(M,M'):\mathcal{M}\times\mathcal{M}'} \mathcal{C}(A, M \bullet B) \times W(M, M') \times \mathcal{D}(M' \blacksquare B', A').$$

The advantages of weighted optics over ordinary optics are clear: when dealing with weighted optics, we are no longer forced to take reverse maps from the same category as the forward maps. The action on the category of forward spaces is now separated from the action on the category of backward spaces, and the link between the two actions is provided by an external functor. Such modular approach provides a great deal of conceptual clarity and flexibility, more than regular optics or lenses can provide on their own.

As stated in [Gav24], since **coPara** can be given a monoidal structure, we can also give $\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}}$ one such structure as long as the underlying actegories are monoidal and the weight functor W is braided monoidal.

It is also shown in [Gav24] that weighted optics are indeed a generalization of optics. In particular, it is shown that the lenses in *Def. 8* are the specialized weighted optics obtained when $\mathcal{C} = \mathcal{D}$ is Cartesian and the actegories are given by the Cartesian product. More generally, the thesis claims that - to the best of the author's knowledge - all definitions of lenses currently used in the literature are subsumed by the definition of weighted optics.

Parametric weighted optics

Although weighted optics model bidirectionality very well, they cannot be used to model neural networks just yet, as they are not parametric objects. The **Para** construction can only be applied to weighted optics if we find an appropriate action on $\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}}$. As shown in [Gav24], this can be done under the assumption that $\mathcal{M}$ and $\mathcal{M}'$ are braided: if that's the case, we can just weighted optic category $\mathcal{V} = \mathbf{Optic}^{W}_{\binom{\otimes}{\otimes'}}$ generated by the self-actions of the two categories, and we can define a $\mathcal{V}$-action $*$ on $\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}}$ so that

$$\begin{pmatrix} M \\ M' \end{pmatrix} * \begin{pmatrix} X \\ X' \end{pmatrix} = \begin{pmatrix} M \bullet X \\ M' \blacksquare X' \end{pmatrix}.$$

We can then write the following definition:

Definition 15 (Parametric weighted optics)**.** *Let* $(\mathcal{M}, I, \otimes)$ *and let* $(\mathcal{M}', I', \otimes')$ *be braided monoidal categories. Let* $(\mathcal{C}, \bullet)$ *and* $(\mathcal{D}, \blacksquare)$ *be an* $\mathcal{M}$*-actegory and an* $\mathcal{M}'$*-actegory, respectively. Finally, let* $F : \mathcal{M} \to \mathcal{M}'$ *be a lax monoidal functor. Then, if* $*$ *is the action mentioned above, we define the category of parametric weighted optics as*

$$\mathbf{Para}_* \left(\mathbf{Optic}^W_{\binom{\bullet}{\blacksquare}} \right).$$

Parametric weighted optics are an even more sophisticated piece of categorical machinery than non-parametric ones, and thus the consistent use of string diagrams is extremely helpful. Refer to ADDIM to see examples of such diagrams.

1.1.3 (Co)algebras over functors and monads

We go on a short tangent about (co)algebras which will be useful later on in this chapter and also in the following one. We will only provide a few definitions and an intuition for them, but the interested reader can find more information in [JR97] and [Wis08].

Providing endofunctors with structure

It makes sense to use natural transformations to add structure to an endofunctor $F : \mathcal{C} \to \mathcal{C}$.

Definition 16 (Pointed endofunctor)**.** *A pointed endofunctor over a category* $\mathcal{C}$ *is a pair* (F, η)*, where* $F : \mathcal{C} \to \mathcal{C}$ *is a functor and* $\eta : \mathrm{id}_{\mathcal{C}} \Rightarrow F$ *is a natural transformation.*

Definition 17 (Copointed endofunctor)**.** *A copointed endofunctor over a category* $\mathcal{C}$ *is a pair* (F, ϵ)*, where* $F : \mathcal{C} \to \mathcal{C}$ *is a functor and* $\epsilon : F \Rightarrow \mathrm{id}_{\mathcal{C}}$ *is a natural transformation.*

The transformation η in the definition of pointed endofunctor looks like the identity in a categorical monoid. Thus, it makes perfect sense to extend the definition to include an associated product[1].

Definition 18 (Monad)**.** *A monad over a category* $\mathcal{C}$ *is a triplet* (F, η, μ)*, where* $F : \mathcal{C} \to \mathcal{C}$ *is a functor, and* $\eta : \mathrm{id}_{\mathcal{C}} \Rightarrow F$ *and* $\mu : F \fatsemi F \Rightarrow F$ *are natural transformations. Moreover,* η *and* μ *must satisfy the coherence diagrams in ADDIM.*

Reversing all the natural transformation arrows in the definition above and in the diagrams yields the definition of comonad.

[1] We don't claim that monads were discovered in this way. Our point of view is purely pedagogical.

Building upon the structure

Endofunctors, (co)pointed endofunctors and (co)monads provide a frame for a category $\mathcal{C}$. We can use the internal machinery of $\mathcal{C}$ to build upon such frame and harvest the benefits of having such structure.

Definition 19 (Algebra over an endofunctor)**.** *Let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor. An algebra over F is a pair (A, a) where $A : \mathcal{C}$ and $a : \mathcal{C}(F(A), A)$.*

Definition 20 (Coalgebra over an endofunctor)**.** *Let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor. An algebra over F is a pair (A, a) where $A : \mathcal{C}$ and $a : \mathcal{C}(A, F(A))$.*

Definition 21 (Coalgebra over a copointed endofunctor)**.** *An algebra over a copointed endofunctor (F, ϵ) on $\mathcal{C}$ is a pair (A, a) where $A : \mathcal{C}$, $a : \mathcal{C}(F(A), A)$ and the commutative diagram in ADDIM is satisfied.*

Definition 22 (Algebra over a monad)**.** *An algebra over a monad (F, η, μ) on $\mathcal{C}$ is a pair (A, a) where $A : \mathcal{C}$, $a : \mathcal{C}(F(A), A)$ and the commutative diagrams in ADDIM are satisfied.*

Coalgebras over monads are defined by reversing all the arrows in the definition of algebras over monads.

Remark 23 (Nomenclature)**.** Given a (co)algebra (A, a), we often refer to a as the structure of the (co)algebra and A as the carrier of such structure. The underlying endofunctor or monad is often known as the signature of the (co)algebra.

(Co)algebra homomorphisms

(Co)algebras over the same endofunctor can be given a categorical structure by providing an appropriate notion of (co)algebra homomorphism.

Definition 24 (Homomorphisms of algebras over an endofunctor)**.** *Let (A, a) and (B, b) be algebras over the same endofunctor $F : \mathcal{C} \to \mathcal{C}$. An algebra homomorphism $(A, a) \to (B, b)$ is a map $f : \mathcal{C}(A, B)$ such that the diagram in ADDIM is commutative.*

Definition 25 (Homomorphisms of coalgebras over an endofunctor)**.** *Let (A, a) and (B, b) be coalgebras over the same endofunctor $F : \mathcal{C} \to \mathcal{C}$. A coalgebra homomorphism $(A, a) \to (B, b)$ is a map $f : \mathcal{C}(A, B)$ such that the diagram in ADDIM is commutative.*

Homomorphisms of (co)algebras over monads are define in the same way.

All concepts discussed in this subsection can be extended to sphere of 2-categories, defining 2-endofunctors, 2-monads, and the associated (co)algebras and (co)algebra

homomorphisms. The only delicate point is that diagrams such as the one defining the homomorphism condition have to become lax-commutative instead of strictly commutative in the 2-categorical setting. We don't insist on this subject for lack of space, but the interested reader is referred to [GLD+24b] for further information.

1.1.4 Differential categories

Modelling gradient-based learning obviously requires a setting where differentiation can take place. Although it is tempting to directly employ Euclidean space, recent research has shown that there are tangible advantages in working with generalized differential combinators that extend the notion of derivative to polynomial circuits ([WZ22], [WZ21]), manifolds (ADDREF), complex spaces (ADDREF), and so on. Thus, it makes sense to work with an abstract notion of derivative which can then be implemented as a known differential combinator through the choice of an appropriate differential category.

The standard approach to differential categories

The first two abstract settings for differentiation that we consider are Cartesian differential categories (first introduced in [BCS06]) and Cartesian reverse differential categories (first introduced by [CCG+19]). The former allow for forward differentiation, while the latter allow for reverse differentiation. We will omit the defining axioms for the sake of brevity, but the reader can find complete definitions in [CCG+19].

Definition 26 (Cartesian differential category)**.** *A Cartesian differential category (CDR)* $\mathcal{C}$ *is a Cartesian left-additive category where a differential combinator* D *is defined. Such differential combinator must take a morphism* $f : A \to B$ *and return a morphism* $\mathrm{D}[f] : A \times A \to B$*, which is known as the derivative of* f*. The combinator* D *must satisfy a number of axioms.*

Definition 27 (Cartesian reverse differential category)**.** *A Cartesian reverse differential category (CRDC)* $\mathcal{C}$ *is a Cartesian left-additive category where a reverse differential combinator* R *is defined. Such reverse differential combinator must take a morphism* $f : A \to B$ *and return a morphism* $\mathrm{R}[f] : A \times B \to A$*, which is known as the reverse derivative of* f*. The combinator* R *must satisfy a number of axioms.*

The aforementioned axioms make sure that D and R satisfy the properties expected from derivative and reverse derivative. In particular, a differential combinator satisfies a chain rule (see figure ADDIM), whereas a reverse differential combinator satisfies a reverse chain rule (see figure ADDIM).

Example 28. Smooth is a both a CDC and a CRDC. In fact, if $\mathcal{J}_f$ is the Jacobian matrix of a smooth morphism f,

$$\mathrm{D}[f] : (x, v) \mapsto \mathcal{J}_f(x)v$$

and

$$\mathrm{R}[f] : (x, y) \mapsto \mathcal{J}_f(x)^T y$$

induce well-defined combinators D and R. This is only a partial coincidence, as shown in [CCG+19] that CRDCs are always CDCs under a canonical choice of differential combinator. The converse, however, is generally false.

As it turns out, forward differentiation tends to be less efficient when dealing with neural networks that come up in practice, so CDCs are not extremely useful when studying deep learning. CRDCs, on the other hand, have been applied to great success (see e.g. [CGG+22]). Moreover, CRDCs have been recently generalized by [Gav24] to coalgebras associated with copointed endofunctors. We will examine the generalized definition later on.

Providing categories with a differential structure

Before we move on to other topics, let us discuss how a Cartesian left-additive category can be endowed with a reverse differential structure. The most straight forward approach uses the following theorem, which we take from [WZ22].

Proposition 29. *Suppose $\mathcal{C}$ be the Cartesian left-additive category generated by* $(\mathrm{Obj}, \mathrm{Mor})$ *and a set E of equations. Suppose each $s \in \mathrm{Mor}$ is endowed with a reverse derivative* $\mathrm{R}[s]$ *and suppose such derivative well-defined with respect to the equivalence classes induced by E. Suppose such reverse derivatives also satisfy the first four axioms of CRDCs. Then,* R *defines a Cartesian reverse differential structure on $\mathcal{C}$.*

Prop. 29 allows us to give differential structures to a large family of categories. For instance, [WZ22] uses the proposition to turn Cartesian distributive categories[2] into CRDCs, which allows the authors to give a differential structure to the category of polynomial circuits.

Definition 30 (Polynomial circuits)**.** *Let S be a commutative semiring. Then,* $\mathbf{PolyCirc}_S$ *is the Cartesian distributive category generated by an object* 1*, a generating morphism $s : 0 \to 1$ for each $s \in S$ and the equations in ADDIM.*

Proposition 31. *If we define* $\mathrm{R}[s]$ *as the discard map for each $s \in S$,* $\mathbf{PolyCirc}_S$ *is a CRDC.*

[2]Cartesian distributive categories are Cartesian categories whose hom-sets have both a left-additive and a left-multiplicative structure. The second is required to distribute over the first.

The $\mathbf{Lens}_A$ construction

It is shown in [Gav24] that there is a particular class of weighted optics which is useful for reverse differentiation, being able to represent both maps (through forward passes) and the associated reverse derivatives (through backward passes). Moreover, such weighted optics can be represented as lenses in the sense of *Def 8*, which means that their inner workings can be pictured in a simple, intuitive way. Let us start by defining a new kind of Cartesian left-additive category.

Definition 32 (Additively closed Cartesian left-additive category)**.** *A Cartesian left-additive category $\mathcal{C}$ is an additively closed Cartesian left-additive category (ACCLAC) if and only if the following are true:*

- *the subcategory $\mathbf{CMon}(\mathcal{C})$ of additive maps has a closed monoidal structure $(I, \otimes)$;*
- *the embedding $\iota : \mathbf{CMon}(\mathcal{C}) \to \mathcal{C}$ is a lax monoidal funtor with respect to the aforementioned structure of $\mathbf{CMon}(\mathcal{C})$ and the Cartesian structure of $\mathcal{C}$.*

Then, we can define the category of lenses with backward passes additive in the second component.

Definition 33. *Let $\mathcal{C}$ be an ACCLAC with Cartesian structure is $(1, \times)$ and whose subcategory $\mathbf{CMon}(\mathcal{C})$ has monoidal structure $(I, \otimes)$. Then, we define*

$$\mathbf{Lens}_A(\mathcal{C}) = \mathbf{Optic}^{\mathcal{C}(-,\iota(-))}_{\binom{\times}{\otimes}}.$$

As argued in [Gav24], the symbol $\mathbf{Lens}_A$ is justified because one such optic of type $\binom{X}{X'} \to \binom{Y}{Y'}$ can be concretely represented as a lens with forward pass $f : \mathcal{C}(X, Y)$ and backward pass $f' : \mathcal{C}(X \times Y', X')$. We do not have the space to report the details of this argument, but we report the following remark.

Remark 34. Some potential expressiveness is lost when passing from weighted optic composition to concrete lens composition. In particular, if we operated with optics, we would be able to implement backpropagation without resorting to gradient checkpointing, which is not possible if we use lenses.

[Gav24] proposes the following conjecture.

Conjecture 35. *Weighted optics provide good denotational and operational semantics for differentiation.*

Since the conjecture hasn't been formally proved yet, the author abandons the weighted optics perspective in favor of the lens perspective, and so do we. As it has already been shown in [CGG+22], lenses are expressive enough to satisfyingly model gradient based learning, and thus, while weighted optics seem to offer great future potential, lens-theoretic approaches are far from being disadvantageous.

Generalized reverse differential categories

[Gav24] proposes the $\mathbf{Lens}_A$ construct as foundation for generalizing reverse differential categories. This is possible because $\mathbf{Lens}_A$ is an endofunctors, as shown by the following definitions and propositions.

Definition 36. *We defined* **CLACat** *as the category whose objects are Cartesian left-additive categories and whose morphisms are Cartesian left-additive functors (see e.g.[BCS06]).*

Proposition 37. *If* $\mathcal{C} : \mathbf{CLACat}$*, then* $\mathbf{Lens}_A(\mathcal{C}) : \mathbf{CLACat}$*.*

Proof. The Cartesian structure on $\mathbf{Lens}_A(\mathcal{C})$ is given by $\binom{X}{X'} \times \binom{Y}{Y'} = \binom{X \times Y}{X' \times Y'}$ and by the initial object $\binom{1}{1}$. The monoidal structure on each $\binom{X}{X'}$ is given by the unit $0_{\binom{X}{X'}} = \begin{pmatrix} 0_A \\ !_{1 \times A'} \end{pmatrix} : \binom{1}{1} \to \binom{X}{X'}$ and by the multiplication $+_{\binom{X}{X'}} = \begin{pmatrix} +_A \\ \pi_2 \fatsemi \Delta_{A'} \end{pmatrix} : \binom{X \times X}{X' \times X'} \to \binom{X}{X'}$. □

Proposition 38. $\mathbf{Lens}_A : \mathbf{CLACat} \to \mathbf{CLACat}$ *is a functor.*

Proof. Given a Cartesian left-additive functor $F : \mathcal{C} \to \mathcal{D}$, we can define $\mathbf{Lens}_A(F)$ as the functor that maps $\binom{X}{X'} \mapsto \begin{pmatrix} F(X) \\ F(X') \end{pmatrix}$ and maps $\begin{pmatrix} f \\ f' \end{pmatrix} \mapsto \begin{pmatrix} F(f) \\ \underline{f} \end{pmatrix}$, where $\underline{f} = F(X) \times F(Y')) \xrightarrow{\cong} F(X \times Y') \xrightarrow{F(f')} F(X')$. It can be shown that $\mathbf{Lens}_A(F)$ is also Cartesian left-additive. □

Proposition 39. $\mathbf{Lens}_A$ *has a copointed structure (Def. 17).*

Proof. It suffices to endow $\mathbf{Lens}_A$ with the natural transformation ϵ whose components are the forgetful functors $\epsilon_{\mathcal{C}} : \mathbf{Lens}_A(\mathcal{C}) \to \mathcal{C}$ which strip away the backward passes. □

Finally, [Gav24] provides generalizes CRDCs as follows.

Definition 40 (Generalized Cartesian reverse differential category)**.** *A generalized Cartesian reverse differential category is a coalgebra (Def. 21) for the pointed endofunctor* $\mathbf{Lens}_A$*.*

Explicitly, a generalized CRDC is a pair $(\mathcal{C}, \mathbf{R}_{\mathcal{C}})$ such that $\mathcal{C} : \mathbf{CLACat}$ and $\mathbf{R}_{\mathcal{C}} : \mathcal{C} \to \mathbf{Lens}_A(\mathcal{C})$ satisfies $\mathbf{R}_{\mathcal{C}} \fatsemi \epsilon_{\mathcal{C}} = \mathbf{id}_{\mathcal{C}}$. The intuition behind such definition is that $\mathbf{R}_{\mathcal{C}}$ should map $f \mapsto \begin{pmatrix} f \\ R[f] \end{pmatrix}$, where $R[f]$ is a generalized reverse derivative combinator. [Gav24] shows that such a definition of $\mathbf{R}_{\mathcal{C}}$ does indeed prove that ordinary CRDC fall into the definition of generalized CRDC.

Remark 41. The $\mathbf{R}_{\mathcal{C}}$ functor associated with a generalized CRDC $\mathcal{C}$ is extremely useful because it allows is to design architectures as compositions $f_1 \fatsemi \cdots \fatsemi f_n$ in $\mathcal{C}$ and then carry them over to $\mathbf{Lens}_A(\mathcal{C})$ as $\mathbf{R}_{\mathcal{C}}(f_1 \fatsemi \cdots \fatsemi f_n) = \mathbf{R}_{\mathcal{C}}(f_1) \fatsemi \cdots \fatsemi \mathbf{R}_{\mathcal{C}}(f_n)$. The reverse derivative of the composition is automatically 'assembled 'from the reverse derivatives of the building blocks. If we want such reverse derivative explicitly, it suffices to extract the backward pass of the lens above, which is possible thanks to the Cartesian structure of $\mathcal{C}$.

Thus, $\mathbf{R}_{\mathcal{C}}$ makes it possible for us to focus on architecture by automatically handling differentiation in a compositional and functional fashion. We shall see how important this point is in the next chapter.

Functional reverse-mode automatic differentiation

We wish to highlight the similarities between the formal theory of differential categories illustrated here and the work in [**?**]. The paper describes the Haskell implementation of a purely functional automatic differentiation library, which is able to handle both forward mode and backward mode AD without resorting to the mutable computational graphs used by most current day libraries.

Among the main insights of [**?**], it is stated that derivatives should not be treated as simple vectors, but as linear maps, or multilinear maps in the case of uncurried higher-order derivatives. Moreover, the author shows that differentiation can be made compositional by working on pairs (f, Df), which behaved very similarly to lenses. As noted by [SGW21], however, [CGG$^+$22] and other lens-theoretical perspectives do not subsume the work in [**?**] because of the latter's programming focus. See [SGW21] for more information regarding this comparison.

1.1.5 Applying Para over functors

We conclude this section discussing the relation between $\mathcal{C} \to \mathcal{D}$ functors and $\mathbf{Para}_{\bullet}(\mathcal{C}) \to \mathbf{Para}_{\bullet}(\mathcal{D})$. We don't have the space necessary to provide a full account of what is known on the subject, but we will discuss two specific cases which are relevant to this thesis. More information can be found in [Gav24] and [GLD$^+$24b].

Applying Para over $\mathbf{Lens}_A$

[Gav24] shows that, under an appropriate definition, 'morphisms of actegories induce morphisms of parametric bicategories'. As a consequence, it can be shown that, if $(\mathcal{C}, \mathbf{R}_{\mathcal{C}})$ is a generalized CRDC, $\mathbf{R}_{\mathcal{C}}$ induces a functor $\mathbf{Para}(\mathbf{R}_{\mathcal{C}}) : \mathbf{Para}_{\times}(\mathcal{C}) \to \mathbf{Para}_{\bullet}(\mathbf{Lens}_A(\mathcal{C}))$, which takes a parametric map $f : P \times A \to B$ and augments is with its reverse derivative $\mathrm{R}[f]$, forming a parametric lens. Parametric lenses behave very similarly to lenses, but we provide a separate stand-alone

definition (which we take from ADDREF[3]) for the reader's convenience.

Definition 42 (Parametric lenses)**.** *The category of parametric lenses over a Cartesian category* $(\mathcal{C}, 1, \times)$ *is* $\mathbf{Para}_\bullet(\mathbf{Lens}(\mathcal{C}))$*, where* $\bullet$ *is the action on the lenses generated by the Cartesian structure of* $\mathcal{C}$*:*

$$\begin{pmatrix} P \\ P' \end{pmatrix} \bullet \begin{pmatrix} A \\ A' \end{pmatrix} = \begin{pmatrix} P \times A \\ P' \times A' \end{pmatrix}.$$

Refer to *Fig. ADDIM* to see a string diagram that shows the inner workings of a parametric lens.

Algebras over parametric endofunctors

The following proposition (proved in [GLD$^+$24b] where it is stated as an example) shows that actegorical strong endofunctors induce 2-endofunctors over parametric categories.

Proposition 43. *Suppose* $(\mathcal{C}, \bullet)$ *is an* $\mathcal{M}$*-actegory and* $F : \mathcal{C} \to \mathcal{C}$ *is an actegorical endofunctor with strength* σ*. Then,* F *induces a* 2*-functor* $\mathbf{Para}(F) : \mathbf{Para}_\bullet(\mathcal{C}) \to \mathbf{Para}_\bullet(\mathcal{C})$*.*

Proof. Define $\mathbf{Para}(F)$ so that:

1. $\mathbf{Para}(F)$ acts like F on objects $A : \mathcal{C}$;
2. $\mathbf{Para}(F)(f) = P \bullet F(A) \xrightarrow{\sigma_{P,A}} F(P \bullet A) \xrightarrow{F(f)} F(B)$ for all $(P, f) : \mathbf{Para}_\bullet(\mathcal{C})(A, B)$;
3. $\mathbf{Para}(F)$ leaves reparametrizations unchanged.

□

1.2 Compositional models

[3] The contents of ADDREF where anticipated in a number of papers, among which ADDREF.

Chapter 2

Gradient-Based Learning

Gradient-based learning has enjoyed unparalleled success in the last few years, being applied to an ever-increasing number of fields both in the sciences and in engineering. Such revolution was made possible by a number of theoretical and practical achievements that took place in the last century. Among these we must note the development of linear algebra packages in the second half of the last century, the invention of the backpropagation algorithm (ADDREF) and other automatic differentiation (AD) algorithms, and the development of large multipurpose machine learning libraries like PyTorch or TensorFlow.

However, as stated in the introduction, the development of such tools has not been informed by a general mathematical theory of gradient-based learning (ADDREF). On the contrary, such developments have taken place mainly guided by empirical results and euristics: even the most significant achievement in the filed often rely on *ad hoc* choices which have no clear theoretical justifications. In the absence of mathematical structure, most modern frameworks implement AD using computational graphs and relying on side effects, inviting preventable bugs and making it very difficult to use parallel computing at its fullest potential (ADDREF).

A number of authors (ADDREF) have proposed category theory as a general theoretical framework for gradient-based learning. In this chapter, we will illustrate three of these approaches: optic-based learning (ADDREF), categorical deep learning (ADDREF), and functor learning (ADDREF). Let's provide a short description of each.

1. *Optic-based learning* aims to construct a general, compositional, architecture-agnostic gradient-based learning framework for deep neural networks. In such frameworks, models and loss functions are represented as parametric optics - most commonly lenses - whereas optimizers are reparametrizations. Forward passes are used to compute losses, while backward passes are used to compute gradient. Finally, optimizers use the gradient and the old parameters to

compute new parameters.

2. While optic-based learning is general and versatile, it does not impose any architectural constraints on the modelled networks. On the other hand, *categorical deep learning* does not deal with optimization, but we have chosen to include it in this chapter because we see it as complementary to optic-based learning, in that it uses (co)algebras over functors and monads to model architectures which can then be implemented using optics. Particular attention is devoted to implementing invariance and equivariance constraints: categorical deep learning is proposed as a successor to geometric deep learning in this regard.

3. In optic-based learning, as in most categorical machine learning frameworks, machine learning models are represented as morphisms in an appropriate category. On the other hand, in *functor learning*, the models are incarnated by functors, which are learned using gradient-based methods. The main advantage offered by functor learning is that functoriality between appropriate categories allows for increased expressiveness and makes it easier to impose constraints (among which invariance and equivariance).

2.1 Optic-based learning

Although the framework of parametric weighted lenses presented in *Chap. 1* is very general and offers exciting possibilities, no one (to the best of our knowledge) has applied it in its fullest potential to machine learning yet. Therefore, following [Gav24] and [CGG$^+$22], we present the framework from the more specialized point of view of parametric lenses, which are nonetheless sufficient to model gradient-based compositionally.

2.1.1 Learning with parametric lenses

Parametric lenses as modules

Gradient based learning requires forward and backward propagation: the former is used to compute the objective function, while the latter calculates its gradient and uses it to update the parameters. The position held by [CGG$^+$22] is that the most natural way to relate the opposite flows of information is to represent the components of the network as parametric lenses, where the forward pass handles forward propagation while the backward pass handles backward propagation.

If we operate in Cartesian reverse differential category $(\mathcal{C}, \mathbf{R}_\mathcal{C})$ (see *Subsec. [?]*), we can use $\mathbf{Para}(\mathbf{R}_\mathcal{C})$ to carry a parametric map (P, f) onto the parametric lens

$$\left(\begin{pmatrix} P \\ P \end{pmatrix}, \begin{pmatrix} f \\ \mathrm{R}[f] \end{pmatrix}\right) : \mathbf{Para}(\mathbf{Lens}(\mathcal{C})) \left(\begin{pmatrix} A \\ A \end{pmatrix}, \begin{pmatrix} B \\ B \end{pmatrix}\right). \tag{2.1}$$

If (P, f) represents an untrained module of neural network, the lens above can be used to encapsulate both forward propagation along the module and the associated backward propagation. A whole neural network can be assembled by composing such lenses, and the functoriality of $\mathbf{Para}(\mathbf{R}_\mathcal{C})$ guarantees that there is no difference between composing lenses and generating a single lens from a composition of parametric maps.

Hence, if $\left(\binom{P}{P}, \binom{f}{\mathrm{R}[f]}\right)$ represents a linear layer and $\left(\binom{Q}{Q}, \binom{g}{\mathrm{R}[g]}\right)$ represents an activation layer, their composition represents a fully connected layer according to the same convention.

Remark 44. [SGW21] notes an important detail: the composition of two lenses in $\mathbf{Lens}_A(\mathcal{C})$ corresponds to an implicit application of the chain rule for reverse derivatives. One can notice this by justapposing ADDIM and ADDIM and observing the similarity between the two composition laws. This key detail is what makes it possible to represent neural network modules as parametric lenses.

Supervised learning with parametric lenses

The authors of [CGG+22] and [Gav24] take this line of reasoning a step further and show that it is also meaningful to represent other components of learning - such as optimizers, loss functions, and learning rates - within the framework of parametric lenses.

For instance, we may represent a loss maps as $(P, \mathrm{loss}) : \mathbf{Para}(\mathcal{C})(B, L)$ for some object L. The intuitive meaning of this representation is the following: loss takes the output of the network of type B, compares it with the labels of type P (P and B will often coincide in practical examples), and then returns a loss of type L. The parametric lens associated with loss can thus be post-composed after the parametric lens associated with a neural network to compute the loss generated by such network given some input and the associated labels. This process is illustrated in Fig. ADDIM.

This leaves two dangling wires of type L. We can use a learning rate lens α to link the wires and allow forward-propagating information to "change direction" and go backwards. α must have domain equal to $\binom{L}{L}$ and codomain equal to $\binom{1}{1}$, where 1 is the terminal object of $\mathcal{C}$. For instance, if $\mathcal{C} = \mathbf{Smooth}$, α might just multiply the loss by some ϵ, which is what machine learning practitioners would ordinarily

call learning rate. Fig. ADDIM shows how a learning rate can be linked to the loss function and the model using post-composition.

The final element needed for the model f in Fig. ADDIM to learn is an optimizer. It is shown in [CGG+22] that optimizers can be represented as reparametrisations in $\mathbf{Para}(\mathbf{Lens}(\mathcal{C}))$. More specifically, we might see an optimizer as a lens $\binom{P}{P} \to \binom{Q}{Q}$. In gradient descent, for example, $P = Q$ and the aforementioned lens is $\binom{1_P}{+_P}$. We can plug such reparametrisation on top of the model to obtain the string diagram in Fig. ADDIM. The diagram shows how the machinery hidden by the $\mathbf{Para}(\mathbf{Lens}(\mathcal{C}))$ can take care of forward propagation, loss computation, backpropagation and parameter updating in a seamless fashion. This is the true power of the compositional mindset: abstraction hides away unwanted detail so that one can focus on high-level features of the model.

Weight tying

Both [CGG+22] and [Gav24] emphasize the essential role played by weight sharing in deep learning. Weight tying can be implemented within the parametric lens framework as a reparametrization that copies a single parameter to many parameter slots: given $(P \times P, f) : \mathbf{Para}(\mathcal{C})(X, Y)$, we can define $(P, f^{\Delta_P}) : \mathbf{Para}(\mathcal{C})(X, Y)$ so that

$$f^{\Delta_P} : X \times P \xrightarrow{X \times \Delta_P} X \times P \times P \xrightarrow{f} .$$

Weight tying can also be used for batching. For instance, we can create n different copies of our supervised learning lens (comprised of model, loss function, and learning rate) and tie the parameters to values unique across the copies. Then, it suffices to feed the n data points to the n lenses, and we can optimize across a single parameter. A string diagram representation of weight tying can be found in ADDIM. Other information regarding the learning iteration in the framework of parametric lenses can be found in [CGG+22].

Empirical evidence

Empirical evidence for the effectiveness of parametric lenses can be found in [CGG+22], where the authors go on to develop a Python libraries for parametric lenses. They use the library to develop a MNIST classifier, obtaining comparable accuracy to models developed using traditional means.

Since parametric lenses can easily be implemented functionally and without side effects, success stories as the one mentioned above foreshadow a future where popular machine learning libraries also follow elegant functional paradigms informed by category theory. Quoting [CGG+22] directly, '[the] proposed algebraic structures naturally guide programming practice'.

2.1.2 Comparisons and generalizations

Learners

One of the first compositional approaches to training neural networks in the literature can be found in the seminal paper [FST19], which spurred much research in the field, including what is presented in [Gav24] and [CGG+22]. The authors introduce a category of learners, objects which are meant to represent components of a neural network and behave similarly to parametric lenses.

Definition 45 (Category of learners)**.** *Let A and B be sets. A learner $A \to B$ is a tuple (P, I, U, r) where P is a set, and $I : P \times A \to B$, $U : P \times A \times B \to P$, and $r : P \times A \times B \to A$ are functions. P is known as parameter space, I as implement functions, U as update function, and r as request function. Two learners $(P, I, U, r) : A \to B$ and $(Q, J, V, s) : B \to C$ compose forming $(P \times Q, I * J, U * V, r * s) : A \to C$, where*

$$(I * J)(p, q, a) = J(q, I(p, a)),$$

$$(U * V)(p, q, a, c) = (U(p, a, s(q, I(p, a), c)), V(q, I(p, a), c)),$$

$$(r * s)(p, q, a, c) = r(p, a, s(q, I(p, a), c)).$$

Learners quotiented by an appropriate reparametrization relationship[1] *form a category* **Learn**.

A learner represents an instance of supervised learning: the implement function takes a parameter and implements a function and the update function updates the parameters using a data from a dataset. The request function is necessary to implement backpropagation when optimizing a composition of learners. Suppose we select a learning rate ϵ and an error function $e : \mathbb{R}^2 \to \mathbb{R}$ such that $y \mapsto \frac{\partial e}{\partial x}(x_0, y)$ is invertible for all y. It is argued in [FST19] that we can define a functor $L_{\epsilon,e} : \mathbf{Para}_{\times}(\mathbf{Smooth}) \to \mathbf{Learn}$ which takes a parametric map and yields an associated learner that implements gradient descent.

We don't have the space to talk about learners at length, but we wish to draw a short comparison between parametric weighted optics (and, in particular, parametric lenses) and the approach of [FST19], given the relevant position held by the paper in the machine learning literature. The similarities between learner-based learning and lens-based learning are evident: every learner (P, I, U, r) looks like a parametric lens, where I passes information forward, r passes information backwards and P is the parameter space. Moreover, the role of $L_{\epsilon,e}$ is very similar to

[1] As argued in [FST19], learners could be studied from a bicategorical point of view, where reparametrizations would just be 2-cells. We could then use a connected component projection to compress **Learn** into a 1-category **Learn**, as it is done for **coPara** when defining weighted optics.

the role played by $\mathbf{Para}(\mathbf{R}_{\mathcal{C}})$ in optic-based learning. Such similarities were even discussed in the original paper [FST19] and have been researched at length: it has been proved in [FJ19] that learners can be functorially and faithfully embedded in a special category of symmetric lenses (as opposed to the lenses of *Def. 8*, which are asymmetric).

Despite the similarities, there is one fundamental difference between the lens-based approach and the learner-based approach: each learner carries its own optimizer, whereas optimization of lenses is usually carried out separately. Moreover, if we compare parametric weighted optics with learners, the latter clearly win in versatility, generality, and (at least from our point of view) conceptual clarity. It is argued in [SGW21] and [CGG$^+$22] that the parametric lens framework largely subsumes the learner approach. More information regarding the comparison can also be found in [Gav24].

Exotic differential categories

We have presented the parametric weighted optic approach of [Gav24] and [CGG$^+$22] within the context of neural networks for the sake of simplicity, but the framework has been developed with generality in mind and applies to a much wider range of situations. For instance, we can easily replace **Smooth** with any other CRDC $\mathcal{C}$, yielding a full-feature compositional framework for gradient-based learning over $\mathcal{C}$.

Switching to a different CRDC is useful because different differential categories can lead to very different learning outcomes, both in terms of accuracy of the model and in terms of training computational costs (ADDREF). For instance, is argued in [WZ22] that polynomial circuits (see *Def. 30*) can be used to define and train intrinsecally discrete machine learning models. Even 'radical 'environments such as Boolean circuits - where scalars reside in $\mathbb{Z}_2$ - seem to be conductive to machine learning under the right choice of architecture and optimizer ([WZ21]).

2.2 Categorical deep learning

Optic-based learning provides a structured general-purpose compositional framework for gradient-based learning. However, such great versatility has a price: optics are unable to guide the architectural design of our models. It has been shown times and times again that a better architecture makes as much of a difference in machine learning as an algorithm with better asymptotic cost does in classical computer science. In fact, even if we ignore performance concerns, selecting the right architecture is synonymous with selecting the right inductive bias, which is essential for effective learning.

Therefore, finding a principled mathematical framework able to guide such architectural choices is of paramount importance. Designing an architecture is almost synonymous to imposing a set of constraints: for instance, convolutional layers are notoriously equivalent to translationally equivariant linear layers (ADDREF). Hence, it makes sense to inform the choice of architecture by informing the choice of constraint. Among the various attempts at developing a principled theory of machine learning constraints, geometric deep learning (see e.g. [BBCV21]) is particularly relevant. Geometric deep learning (GDL) focuses on equivariance constraints defined with respect to group actions, and has thus seen a lot of success in contexts where data transformations can be expressed in group-theoretical terms. Regrettably, this is not always the case, as many transformations we can subject the data to are either not invertible or even not compositional ([GLD+24b]). Moreover, as highlighted in [GLD+24b], although GDL is effective at describing the constraints that a model should implement, it is not always clear how such constraints can be actually be implemented.

Categorical deep learning (CDL), introduced by [GLD+24b], attempts to solve the aforementioned problems by categorical means. The main insight behind CDL is that group actions are only one instance of a much larger class of transformation with respect to which equivariance is desirable. Thus, CDL aims to subsume and generalize GDL by weakening the assumptions regarding the examined transformations. At the moment, to the best of our knowledge, [GLD+24b] is the only publicly available paper that discusses this concept. In this section, we will sketch the main ideas contained in the paper, and we will then compare CDL to other categorical approaches.

2.2.1 Generalizing geometric deep learning

Equivariance

Equivariance is a highly sought after property in machine learning for many reasons. We list some below.

1. *Dimensionality reduction.* Implementing invariance or equivariance with respect to the right transformations can help ignore superfluous information, reducing the number of parameters of a model. For instance, convolutional layers, which are translationally equivariant, require significantly fewer weights than linear layers of the same size (ADDREF).

2. *Robustness.* For example, an image classifier which is equivariant to rotations can recognize images even if their features are rotated, and is thus less vulnerable to errors or adversarial attacks (ADDREF).

3. *Pooling.* Similar data pooled from different datasets will likely display shifts and imbalanced in many covariates due to differences in context and data collection protocols. Thus, when training a neural network on a pooled dataset, it is advantageous to enforce equivariance with respect to such shifts and imbalances ([CLLS24])[2].

4. *Fairness.* Neural networks should not hold unfair biases, especially when used to inform decisions with high societal impact. Equivariance can help prevent this (ADDREF). For instance, a neural network used to screen CVs can only be considered fair if it displays equivariance with respect to sensitive attributes of the candidates, such as their age, gender, ethnicity, and so on.

It is thus extremely important to provide a good definition of equivariance. The definition used in GDL is the following.

Definition 46 (Group action equivariance and invariance)**.** *Let G be a group and let $(S, \cdot)$ and $(T, *)$ be G-actions. A function $f : S \to T$ is equivariant with respect to the aforementioned actions if $f(g \cdot s) = g * f(s)$ for all $s \in \mathcal{S}$ and for all $g \in \mathcal{G}$. We say that f is invariant if $*$ is the trivial action on T, and thus $f(g \cdot s) = f(s)$ for all s and g.*

Following the approach of GDL, once a desired constraint is expressed in terms of equivariance with respect to group actions, as in the previous definition, we can derive characterizing equations. Solving such equations usually implies tying weights, which reduces the net number of parameters in the models and provides the other advantages described above.

From geometry to category theory

As stated in [BBCV21], GDL is inspired by the *Erlangen Programme*, which unified geometry around the notion of invariant at the end of the nineteenth century. Since category theory can be seen as an extension of the *Programme* (as we already remarked in the introduction to this thesis), it is only natural to attempt to generalize GDL by categorical means. Categorical deep learning aims to subsume GDL and proses, in the words of the authors of [GLD+24b], a theory of (co)algebras over endofunctors and monads (see *Sec.* **??**) as a "theory of all architectures". Consider the following proposition to see how monad algebras can model group actions, and how the associated homomorphisms are the sought after equivariant maps.

[2]This is one of those cases where invariance alone does not suffice. Using the example studied in [CLLS24], if we are training a neural network to diagnose Alzheimer's disease, invariance to age is not an option as age is strongly correlated with the disease. On the other hand, equivariance with respect to age is desirable and prevents the model from learning biases due to imbalances in the dataset.

Proposition 47. *Let $(G, e, \cdot)$ be a group. Consider the endomorphism $G \times - :$ **Set** $\to$ **Set** which maps $S \mapsto G \times X$ and $f \mapsto G \times f$. G can be given a monad structure using the natural transformations η, with components $\eta_S : s \mapsto (e, s)$, and μ, with components $(g, h, s) \mapsto (g \times h, s)$. The monad $(G \times -, \eta, \mu)$ can serve as a signature for G-actions. The actions themselves can be recovered by considering algebras $(S, *)$ for the monad, and, given two actions $(S, *)$ and $(T, \star)$, an associated equivariant map $f : S \to T$ is a $(S, *) \to (T, \star)$ monad algebra homomorphism.*

Proof. It suffices to compare the equations that define group actions and group action invariance with the commutative diagrams in ADDIM and ADDIM. □

The proposition above shows the role that monads (or endomorphisms), the associated algebras, and the associated algebra homomorphisms play in CDL: monads (or endomorphisms) provide a signature of sorts, algebras build structure using the aforementioned signature, and, finally, algebra homomorphisms are maps that preserve the structure. The following example, taken from ([GLD+24b]), shows how the monad algebra paradigm can be used to impose equivariance by weight sharing.

Example 48 (Linear equivariant layer)**.** Consider a carrier set $S = \mathbb{R}^{\mathbb{Z}_2}$, which can be seen as a pair of pixels. Consider the translation action $(i * s)(j) = s(i - j)$ of $G = \mathbb{Z}_2$ on S, which can be seen as swapping the pixels. We want to find a linear map $f : S \to S$ which is equivariant with respect to the action. Imposing the equivariance constraints as equations on the entries of the matricial representation $W_f \in \mathbb{R}^{2\times 2}$ of the map, we can prove that f is equivariant if and only if W_f is symmetric.

Remark 49. *Ex. 48* is a simple application of geometric deep learning, but the problem is presented using the formalism of categorical deep learning. In the rest of the section, we will examine problems for which geometric deep learning alone does is not sufficient.

2.2.2 From data structures to neural networks

Lists, trees, automata

Algebras and coalgebras over functors have been of interest to computer scientists long before their application to artificial intelligence was conceived. The reason is that algebras allow us to model induction, while coalgebras are a model for coinduction. We shall not go into explicit details about the general definitions of these principles, except for saying that induction is used to define processes which are guaranteed to end, while coinduction is used to define processes that are guaranteed to be productive ([GLD+24b]). Another (largely equivalent) perspective is that induction creates while coinduction consumes. The interested reader can find

much more detail about this topic in [JR97] and [**?**]: the former discusses the link between (co)algebras and (co)induction, while the latter focuses on coinduction.

Induction and coinduction are very important in computer science because they can be used to define data structures and reason about them and the associated algorithms. We will provide a few interesting examples, all taken from [GLD+24b]. Before we do that, let us introduce a piece of notation, namely, polynomial endofunctors. Let us restrict our focus to **Set**: the category of sets and functions. **Set** has both products, in the form of Cartesian products represented by $\times$ ($\langle , \rangle$ when acting on functions), and coproducts, in the form of disjoint unions represented by $+$ ($[,]$ when acting on functions). We assume the reader is familiar with both. We can use these operations to define functors that are remindful of polynomials. Let X be an unknown representing any set (just like x represents any real number in a real polynomial). Given $n+1$ of specific sets $\{A_j\}_{j=0,\dots,n}$ (which can be compared to coefficients in a polynomial), we can define a polynomial (of sorts) in the form

$$F(X) = A_n \times X^n + \cdots + A_1 \times X^1 + A_0,$$

where X^j stands for the Cartesian product of j copies of X. The polynomial F clearly defines a $\mathrm{Ob}(\mathbf{Set}) \to \mathrm{Ob}(\mathbf{Set})$ map, which we can easily extend to a functor $F : \mathbf{Set} \to \mathbf{Set}$, since a map $f : X \to Y$ map can be easily applied across Cartesian products and disjoint unions thanks to their universal properties as categorical products and coproducts. We call these functors polynomial endofunctors (see [JR97] for more information on this topic). We are now ready for the examples.

Example 50 (Lists)**.** Let A be a set. Consider the polynomial endofunctor $1 + A \times X : \mathbf{Set} \to \mathbf{Set}$. If $\mathsf{List}(A)$ is the set of A-labeled lists, $(\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}])$ is an algebra over $1 + A \times X$. Here, $\mathsf{Nil} : 1 \to List(A)$ is the map which takes the unique object of 1 and returns the empty list, while $\mathsf{Cons} : A \times \mathsf{List}(A) \to \mathsf{List}(A)$ is the map which takes an element $a \in A$ and a list l of elements of A and returns the concatenated list $l \cup \{a\}$. This described lists in $\mathsf{List}(A)$ inductively as object formed by concatenating elements of A to other such lists. The base case is the empty list.

Example 51 (Binary trees)**.** Let A be a set. Consider the polynomial endofunctor $A + X^2 : \mathbf{Set} \to \mathbf{Set}$. Let $\mathsf{Tree}(A)$ be the set of binary tree with A-labeled leaves. Once again we can form an algebra $(\mathsf{Tree}(A), [\mathsf{Leaf}, \mathsf{Node}])$, where Leaf takes $a \in A$ and returns a leaf labeled with this element, while Node takes two trees and joins them as by creating a new node and adding the two arguments as its childer. Once again this is an inductive description of A-labeled binary trees.

Now consider another two examples where the endofunctors are not merely polynomial but are defined by expressions containing the symbol B^A, which represents the set of $A \to B$ functions[3].

[3]We facetiously suggest calling endofunctors using this notation exponential.

Example 52 (Mealy machines)**.** Now consider two sets I and O of possible inputs and outputs, respectively. Consider the endofunctor $(O \times X)^I : \mathbf{Set} \to \mathbf{Set}$. Define $\mathsf{Mealy}_{I,O}$ as the set of Mealy machines with inputs and outputs in I and O, respectively. Now we can consider the coalgebra $(\mathsf{Mealy}_{I,O}, \mathsf{Next})$, where Next is the map that takes a Mealy machine $m \in \mathsf{Mealy}_{I,O}$ and yields a function which in turn, given $i \in I$, returns the output of m at i and a new machine. This is a coinductive description of Mealy machines.

Example 53 (Moore machines)**.** Consider once again the input set I and the output set O. Consider the endofunctor $O \times X^I : \mathbf{Set} \to \mathbf{Set}$. If $\mathsf{Moore}_{I,O}$ is the set of Moore machines with input in I and output in O, we can consider the coalgebra $(\mathsf{Moore}_{I,O}, \langle \mathsf{Output}, \mathsf{Next} \rangle)$, where Output takes a Moore machine m and returns its output, while Next takes the same m and yields a function, which in turn taken an input i and produces a new Moore machine. Once again, this is a coinductive description of Moore machines.

Remark 54. Notice how the descriptions we have given of Mealy machines and Moore machines do not mention the internal states of these objects. This is a recurring aspect of coinductive descriptions: as argued in [JR97], coinduction is best interpreted as a process where an observer track the behavior of an object from the outside, with no access to its internal state. This is very useful in machine learning because the internal state of a learning model is often unknown or uninterpretable.

The true power of the (co)algebraic approach becomes manifest when we consider homomorphisms between (co)algebas, as in the examples below.

Example 55 (List folds)**.** Consider the algebra $(\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}])$ of lists from *Ex. 50*, and consider a second algebra $(Z, [r_0, r_1])$ over the same functor. A homomorphism $f : \mathsf{List}(A) \to Z$ from the former into the latter must satisfy

$$\begin{aligned} f(\mathsf{Nil}) &= r_0, \\ f(\mathsf{Cons}(a, l)) &= r_1(a, f(l)). \end{aligned}$$

Hence, f is necessarily a fold over a list with recursive components r_0 and r_1. Incidentally, this proves that f is unique, making $(\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}])$ an initial object in the category of algebras over the polynomial endofunctor $1 + A \times X$.

We can similarly define folds over binary trees and unfolds over Mealy and Moore machines. The algebra of binary trees is also initial, while the coalgebras of Mealy machines and more machines are final in their respective categories. We can see the recursion and corecursion that such homomorphisms must satisfy are structural, and can be interpreted as generalizations of equivariance constraints over group actions ([GLD+24b]).

(Co)inductive definitions for neural networks

The most significant piece of novel contribution delineated in [GLD+24b] is the use of (co)algebras and (co)algebra homomorphisms over parametric categories to define (co)inductively recurrent and recursive neural networks. (Co)algebras are used to define cells, whereas the associated homomorphisms provide the weight-sharing mechanics used to unroll them. We use two of the same examples discussed in the paper.

Example 56 (Folding recurrent neural network cell)**.** Consider the endofunctor $1 + A \times X : \mathbf{Set} \to \mathbf{Set}$ from *Ex. 50*. Consider the Cartesian action of **Set** on itself and associate the following actegorical strength to the functor: $\sigma_{P,X}(p, \text{inl}) = \text{inl}$ and $\sigma_{P,A}(p, \text{inr}(x, x')) = \text{inr}((p, x), (p, x'))$. Now that the functor is actegorical strong, we can use *Prop. 43* to construct an endofunctor $\mathbf{Para}(1 + A \times X) : \mathbf{Para}_\bullet(\mathbf{Set}) \to \mathbf{Para}_\bullet(\mathbf{Set})$. Consider an algebra $(S, (P, \mathsf{Cell}))$ for this functor. Via the isomorphism $P \times (1 + A \times X) \cong P + P \times A \times X$, we deduce that $\mathsf{Cell} = [\mathsf{Cell}_0, \mathsf{Cell}_1]$, where $\mathsf{Cell}_0 : P \to S$ and $\mathsf{Cell}_0 : P \to S$. We can interpret Cell_0 and Cell_1 as folding recurrent neural network cells: Cell_0 provides the initial state based on its parameter and Cell_1 takes in the old state, a parameter, and an input, which are then used to return a new state (ADDIM).

Example 57 (Unrolling of a folding recurrent neural network)**.** Use *Prop. 7* to embed the list algebra $(\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}])$ from *Ex. 50* as an algebra over the endofunctor $\mathbf{Para}(1 + A \times X)$ from the previous example. Now consider an algebra homomorphism $(P, f) : (\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}]) \to (S, (P, \mathsf{Cell}))$. Since we are working with algebras over a 2-endofunctor, we also need to specify a 2-cell that makes the diagram in ADDIM lax-commutative. Using the weight-tying reparameterization Δ_P, yields the lax commutative diagram in ADDIM, which uniquely identifies f as the fold function which takes a list of inputs in A and unrolls a folding recurrent neural network that reads such inputs. The weight-tying reparametrization makes sure that each cell of the unrolled network uses the same parameters (see ADDIM for a graphical representation).

The same line of reasoning of *Ex. 56* and *Ex. 57* can be applied to produce unfolding recurrent neural networks, complete recurrent neural networks, and even recursive neural networks ([GLD+24b]). In all these cases, the homomorphism map returns parametric maps (P, model), which we can then feed into the $\mathbf{R}_\mathcal{C}$ functor associated with a generalized Cartesian reverse differential category[4] to augment them with their reverse derivative. The framework of parametric lenses described in *Sec. 2.1* can then be used to train these networks. Geometric deep learning

[4]The examples illustrated in this section have been developed in **Set**, but we see no reason why they couldn't be specialized to an appropriate CRDC.

and optic-based learning are thus compatible with each other, and even serve to complement each other.

2.2.3 Related work and future directions

Machine learning and classical computer science

Categorical deep learning highlights a very interesting connection between machine learning and classical[5] computer science. For instance, the derivation in [GLD+24b] allows us to give a precise mathematical formalization to the intuitive link between lists and folding recurrent neural networks.

This is particularly relevant because other such links may not be intuitive outisde of a categorical perspective. For example, it is argued in [GLD+24b] that full recurrent neural networks (i.e. such that each cell takes an input, produces an output and also updates state) are a parametric generalizations of Mealy machines, which is a seldomly consider point of view, according to the paper. This begs the question: if Mealy machines generalize to recurrent neural networks, what do Moore machines generalize to? It is argued in the paper that they generalize to a variant of full recurrent neural networks where different cells (which share the same weight) are used to update the internal state and produce an output.

Hopefully, future work in this direction will prove insightful enough to inform the design of new architectures based on parametrizations of already known classical concepts. At the very least, connecting classical notions to neural network architectures should be a source of conceptual clarity for the field of machine learning.

2.3 Functor learning

In the parametric lens framework presented above, learning happens at the morphism level. The framework is very convenient because its categorical structure hides away the automatic differentiation machinery needed for backpropagation, and thus we can just compose the parametric lenses as modules, so that we can form any network we wish. However, parametric lenses cannot be used to abstract away computations and extract the schema of the model, as they inherently carry out computations. Moreover, the generality of the parametric lens framework means that no tools are developed to study and exploit the inherent structure of the data. Finally, parametric lenses do not offer high level tools to deal with invariance ad equivariance constraints.

[5]The word ‘classical ’here stands for ‘non-machine learning ’.

It has been suggested by [Gav19] and [SY21] that the key to link different layers of abstraction and to exploit the structure of the data is to learn functors instead of morphisms. [Gav19] and [VSP+17] also use functor learning to provide tools that implement abstract constraints on neural networks.

2.3.1 Functors to Separate Layers of Abstraction

The author of [Gav19] takes inspiration from categorical data migration - a field which is located at the intersection of category theory and database theory - to create a categorical framework for deep learning that separates the development of a machine learning process into a number of key steps. The different steps operate on different levels of abstraction and are linked by functors.

Schemas

The first step in the learning pipeline proposed by [Gav19] is to write down the bare-bones structure of the model in question. One can do that using a directed multigraph G, where nodes represent data and edges represent neural networks interacting with such data. Constraints can be added at this level in the form of a set $\mathcal{X}$ of equations that identify parallel paths (see e.g. ADDIM, whose multigraph represent a cycleGAN model).

We can now consider the most abstract categorical representation of such model: its schema.

Definition 58 (Model schema)**.** *The schema of a model represented by a multigraph G is the freely generated category* $\mathbf{Free}(G)$*.*

Such schema does not contain any data nor does it do any computation, but it encodes the bare-bones structure of the model. We can take the module $\mathbf{Free}(G)/\sim$, where $\sim$ is the path congruence relation induced by the equations in $\mathcal{X}$. Depending on the context, the word schema will also refer to such constrained category.

Architectures and models

Given the schema $\mathbf{Free}(G)$, we can choose an architecture for the model. What we mean by choosing architecture is assigning to each node a Euclidean space and to each morphism a parametric map, which represents an untrained neural network.

Definition 59 (Model architecture)**.** *Let* $\mathrm{Free}(G)$ *be a model schema. An architecture for such schema is a functor* $\mathsf{Arch} : \mathrm{Free}(G) \to \mathbf{Para}(\mathbf{Smooth})$*.*

The Euclidean spaces Arch maps objects to might be intuitively interpreted as the spaces the data will live in, but it is best to consider data outside the **Para**

machinery and in the simpler **Set** category, as this allows for better compartmentalisation. Thus, [Gav19] also defines an embedding functor, which agrees with Arch on objects but exists independently of it.

Definition 60 (Model embedding)**.** *Let* $\mathrm{Free}(G)$ *be a model schema and let* Arch *be a chosen architecture. An embedding for such schema is a functor* $E : |\mathrm{Free}(G)| \to \mathbf{Set}$ *which agrees with* Arch *on objects*[6]*.*

Consider the function $\mathfrak{p} : (P, f) \mapsto P$, which takes the parameter space out of a parametric map in $\mathbf{Para}(\mathbf{Smooth})$. We can use it to define a function

$$\mathfrak{P} : \mathsf{Arch} \mapsto \prod_{f:\mathsf{Gen}_{\mathbf{Free}(G)}} \mathfrak{p}(\mathsf{Arch}(f)),$$

where $\mathsf{Gen}_{\mathbf{Free}(G)}$ is the set of generating morphisms of the free category on the multigraph G. The function $\mathfrak{P}$ takes and architecture and returns the parameter space. Given $\mathfrak{P}$, we can define parameter specification function.

Definition 61 (Parameter specification function)**.** *Let* $\mathrm{Free}(G)$ *be a model schema and let* Arch *be a chosen architecture. A parameter specification function is a function* PSpec *which maps a pair* (Arch, p) *- comprised of an architecture* Arch *and some* $p : \mathfrak{P}(\mathsf{Arch})$ *- to a functor* $\mathsf{Model}_p : \mathbf{Free}(G) \to \mathbf{Smooth}$*. The functor* Model_p *takes the model schema and returns its implementation according to* Arch*, partially applying* p_f *to each* $\mathsf{Arch}(f)$*, so that we obtain an actual smooth map.*

The functor Model_p can be seen as a bridge between the parametric notion of untrained neural network and the notion of neural network as a smooth map, which makes most sense after training. To better understand the relationship between Arch and Model_p see ADDIM.

Datasets and concepts

Now, if we hope to train the model we have defined, we will need a dataset. [Gav19] suggests that a dataset should be represented as a subfunctor of the embedding functor:

Definition 62 (Dataset)**.** *Let* E *be a model embedding. Then, a dataset is a subfunctor* $D_E : |\mathbf{Free}(G)| \to \mathbf{Set}$ *which maps every object* A *of the discretised free category to a finite subset* $D_E(A) \subseteq E(A)$*.*

[6]The reason why the domain E is the discretised schema $|\mathrm{Free}(G)|$ instead of the original schema $\mathrm{Free}(G)$ will be clear once we define datasets.

Remark 63. The reason why the author of [Gav19] chooses to define E and D_E on discretized categories is because it often happens in practical machine learning that the available data is not paired. In these cases, it would be meaningless to provide an action on morphisms because they would end up being incomplete maps.

Given a node A of G, we have associated to A a Euclidean space $E(A)$, e.g. $\mathbb{R}^n$, and a dataset $D_E(A)$. It makes sense to define another set $\mathfrak{C}(A)$ such that $D_E(A) \subseteq \mathfrak{C}(A) \subseteq E(A)$. A dataset may be considered a collection of instances of something more specific than just vectors; for instance, if we have a finite dataset of pictures of horses, we are clearly interested in the concept of horse, i.e. in the set of all possible pictures of horses, which is much larger than our dataset but still much smaller than the vector space used to host such pictures. The set $\mathfrak{C}(A)$ is the concept represented by $D_E(A)$: in the aforementioned example, $\mathfrak{C}(A)$ might be the set of all images representing horses. Hence, we can call $\mathcal{C}$ the concept functor.

Definition 64 (Concept functor)**.** *Given a schema* $\mathbf{Free}(G)/\sim$*, an embedding* E *and a dataset* D_E*, a concept associated with this information is a functor* $\mathfrak{C} : \mathbf{Free}(G)/\sim \to \mathbf{Set}$ *such that, if* $I : |\mathbf{Free}(G)| \to \mathbf{Free}(G)/\sim$ *is the inclusion functor,* $D_E \subseteq I; \mathfrak{C} \subseteq E$*.*

As [Gav19] states, $\mathfrak{C}$ is an idealization, but it is a useful idealization as it represent the goal of the optimization process: given a dataset $D_E : |\mathbf{Free}(G)| \to \mathbf{Set}$, we wish to learn the concept functor $\mathfrak{C} : \mathbf{Free}(G)/\sim \to \mathbf{Set}$. Total achievement of such goal is clearly impossible as, even in the simplest cases such as linear regression on linearly generated data, finite arithmetics and the finite nature of the learning iteration prevent us from obtaining a perfect copy of the generating function. Nevertheless, we will hopefully design an optimization process which makes the learning iteration converge towards such ideal goal.

Optimization

Now that we know what the optimization goal is, we can define what a task is. The task formalism brings together what has been defined in this section in an integrated fashion.

Definition 65 (Task)**.** *Let* G *be a directed multigraph, let* $\sim$ *be a congruence relation on* $\mathbf{Free}(G)$ *and let* $D_E : |\mathbf{Free}(G)| \to \mathbf{Set}$ *be a dataset. Then, we call the triple* $(G, \sim, D_E)$ *a task.*

Once we are assigned a machine learning task $(G, \sim, D_E)$, we have to choose an architecture, an embedding and a concept compatible with the given multigraph, equations and dataset. Then, we specify a random initial parameter with an appropriate parameter specification function. Now we can choose and optimizer, but we

must be careful to choose an appropriate loss function. The loss function should incorporate both an architecture specific loss and a path equivalence loss. The former penalizes wrong predictions while the latter penalizes violations of the constraints embodied by $\sim$.

Definition 66 (Path equivalence loss)**.** *Let $(G, \sim, D_E)$ be a task. Let Model_p be an associated model. Then, if $f \sim g : A \to B$ in G, we define the path equivalence loss associated with f, g and Model_p as*

$$\mathcal{L}_{\sim}^{f,g} = \mathbb{E}_{a \sim D_E(A)}[\|\mathsf{Model}_p(f)(a) - \mathsf{Model}_p(g)(a)\|].$$

Definition 67 (Total loss)**.** *Let $(G, \sim, D_E)$ be a task. Let Arch be an associated architecture, let Model_p be an associated model, and let $\mathcal{L}'$ be an architecture specific loss. Suppose γ is a non-negative hyperparameter. Then, we define the total loss associated with the task, the architecture, the model, and the hyperparameter as*

$$\mathcal{L} = \mathcal{L}' + \sum_{f \sim g} \mathcal{L}_{\sim}^{f,g} \tag{2.2}$$

We can now proceed as usual, computing the loss on the dataset for a number of epochs and updating the parameter p each time. Notice that $\mathcal{L}$ implicitly depends on p because each $\mathcal{L}_{\sim}^{f,g}$. Thus, the explicit formula for the loss changes each time the parameter is updated.

Product Task

It is important to notice that, while the learning iteration employed by [Gav19] is nothing new, the functor approach is actually novel, in that the usual optimization process is used to explore a functor space instead of a simple morphism space. The main advantage offered by this procedure is that different layers of abstraction are separated, which allows greater expressive power when defining tasks and solving them.

For instance, it is shown in [Gav19] that changing the dataset functor can result in semantically different networks and tasks even if we keep the same schema. The example shown in the paper is the following: if we take the cycleGAN schema (see ADDIM) and we pair it with a cycleGAN dataset, we obtain a cycleGAN network. Here, by cycleGAN dataset, we mean a dataset where A and B contain data which is essentially isomorphic, such as pictures of horses and zebras. The semantics of the task thus the following: *learn maps that turns horses into zebras and vice versa.* However, if we select a dataset where A consists in pictures which contain two elements X and Y, and B contains separate images of X and Y, then the semantics of the task become *learn how to separate X from Y*. For example, [Gav19] shows

how to use the CelebA dataset to train a neural network able to remove glasses from pictures of faces, or even insert them (see image ADDIM).

This example is especially relevant to the present discussion because it shows how relevant the categorical structure of **Set** can be to machine learning problems. We can interpret pairs *(face, glasses)* as elements of the Cartesian product of the set of faces and the set of glasses. The set of pictures of faces with glasses can instead be considered another categorical product of the aforementioned sets. Then, one interpretation of the task is the following: *find the canonical isomorphisms between two categorical products.* This task, first introduced by [Gav19], is known as product task.

2.3.2 Categorical Representation Learning

While the functorial approach described above provides a categorical interpretation of datasets, namely as functors into **Set**, no categorical structure is given to the data itself, besides the trivial notion that the data lives in sets. It is argued in [SY21] that sometimes data can be given a categorical structure of its own, and preserving such categorical structure makes learning more efficient. This can be done in two steps: (i) functorially embed the data into appropriately defined embedding categories, (ii) learn functors between such embeddings.

Categorical Embeddings

We will illustrate this procedure with the same example employed in the original paper [SY21]: unsupervised translation of the names of chemical elements from English to Chinese. Suppose we have two datasets containing thousands of chemical formulas of various inorganic compounds; the first dataset labels the elements with English labels, while the second dataset labels them in Chinese. These datasets can be given a categorical structure with elements and functional groups being the objects, and bonds between them being the morphisms. Let $\mathcal{C}$ and $\mathcal{D}$ be the resulting categories.

Any category $\mathcal{C}$ can be functorially embedded in the Euclidean space $\mathbb{R}^n$ by considering the vector space category $\mathcal{R}$ associated to the aforementioned vector space (*Def.* **??**). Given a categorical structure to the embedding codomain, it suffices to define a $\mathcal{C} \to \mathcal{R}$ functor which maps $a \mapsto v_a$ and $f \mapsto M_f$. We will use train a neural network to carry out such mapping. This happens on two separate embedding layers: one maps words to vectors, while the other maps relations to matrices.

Remark 68. It is worth noticing that a single matrix M can be in many hom-sets of $\mathcal{R}$, and thus we can have a number of different pairs of vectors linked by the same morphism.

Concurrence Statistics

The embedding layer described above needs to be trained on the data to be effectively functorial. The authors of [SY21] suggest using concurrence statistics and negative sampling to make sure that the embedded morphisms actually represent the same relations between objects elements as the original morphisms between the original objects. It is in fact posited that concurrence encodes such relations. In the authors' words, 'concurrence does not happen for no reason'.

The training strategy used in the paper is the following: given two embedded words a and b, model the probability of concurrence as $P(a \to b) = \mathsf{sigmoid}(z(a \to b))$, where the logit $z(a \to b)$ is defined as

$$z(a \to b) = F\left(\bigoplus_f v_a^T M_f v_b\right).$$

Here, F is non-linear and $\oplus_f$ represents concatenation over all morphisms in $\mathcal{C}$. The idea behind the formula is that it is not possible, given the finite precision of a computer, to compute a matrix M_f such that $v_b = M_f v_a$ exactly. However, we can find M_f such that v_b and $M_f v_a$ are closely aligned. The alignment can be computed as $v_a^T M_f v_b$. The non-linearity F serves as an aggregator for such measurements.

Now, the concurrence probability $p(a, b)$ of two objects $a, b : \mathcal{C}$ can be computed directly from the dataset. Given a negative sampling distribution p_N on objects unrelated to a, we can implement the negative sampling loss

$$\mathcal{L} = \mathbb{E}_{(a,b)\sim p(a,b)}\left(\log P(a \to b) + \mathbb{E}_{b'\sim p(b')}\right),$$

as described in [SY21]. The embedding network can then be trained by maximizing such loss.

Training Functors as Models

Apply the procedure described above to both $\mathcal{C}$ and $\mathcal{D}$ to get categorical embeddings of the datasets. Now, consider a functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ that translates between English labels and Chinese labels. Such functor must equate chemical bonds of the same kind, e.g. if f is a covalent bond so is $\mathcal{F}(f)$. It is posited in [SY21] that the action of $\mathcal{F}$ on morphism is sufficient to deduce the action of $\mathcal{F}$ on objects.

The function $\mathcal{F}$ can be precomposed with the $\mathcal{C} \to \mathcal{R}$ embedding and post-composed with the inverse of the $\mathcal{D} \to \mathcal{R}$ to become a $\mathcal{R} \to \mathcal{R}$ functor. The authors of [SY21] argue that such functor can be represented by a matrix $V_\mathcal{F}$ so that $v_{\mathcal{F}(a)} = V_\mathcal{F} v_a$ and $M_{\mathcal{F}(f)} = V_\mathcal{F} M_f$. Such representation is only meaningful if (i) $V_\mathcal{F} M_f = M_\mathcal{F} V_\mathcal{F}$ for all f, (ii) $V_\mathcal{F} M_{\mathrm{id}_a} = M_{\mathrm{id}_{\mathcal{F}(a)}}$ for all a, and (iii) $V_\mathcal{F} M_{f\circ g} = V_\mathcal{F} M_f V_\mathcal{F} M_g$ for all f, g. This is not true for all choices of $V_\mathcal{F}$ but, if we

choose every v_a to be a unit vector[7], and if we constrain $V_{\mathcal{F}}$ to be orthogonal, (ii) and (iii) are trivially satisfied. The focus can thus be shifted on requirement (i).

Requirement (i) in the previous paragraph can be learned through the following structure loss:

$$\mathcal{L}_{\text{struc}} = \sum_f \|V_{\mathcal{F}} M_f - M_{\mathcal{F}} V_{\mathcal{F}}\|^2.$$

As the authors remark, this loss is universal, in the sense that it does not depend on any specific object, but acts on the morphisms themselves. While this approach is very elegant and does indeed return a functor, this might not be the functor we expect because $V_{\mathcal{F}}$ is not unique if the M_f happen to be singular. Thus, it is better to integrate the structure loss with a second alignment loss which introduces some supervision to the unsupervised translation task. For instance, if the value of $\mathcal{F}(a)$ is known for a set A for objects, we can define

$$\mathcal{L}_{\text{align}} = \sum_{a:A} \|V_{\mathcal{F}} v_a - v_{\mathcal{F}(a)}\|.$$

Then, the total loss can be written as a weighted sum $\mathcal{L} = \mathcal{L}_{\text{align}} + \lambda \mathcal{L}_{\text{align}}$ of structure loss and alignment loss, where λ is a hyperparameter analogous of the γ that appears in *Eq. 2.2*.

Comparison with Traditional Models

It is shown in [SY21] that the kind of functor learning illustrated in this section can lead to remarkable improvements in efficiency when compared with more traditional sequence to sequence models. In particular, the authors compared a GRU cell model of similar performance as the functorial model described in the paper, noting that the former needed 17 times more parameters than the latter.

The authors also compare their approach with the multi-head attention approach first introduced by [VSP+17] (the title of [SY21] is clearly inspired on the title of [VSP+17]). It is argued in the article that the categorical approach is an improvement over multi-head attention as the matrices M_f are essentially equivalent to the products $Q_f^T K_f$, where Q_f is the query matrix associated to f and K_f is the key matrix associated to f. Categorical representation learning differs from multi-head attention as it does not separate the M_f's into their components, which is useful to learn the matrix $V_{\mathcal{F}}$ and allows us to benefit from functoriality.

[7] The paper does not specify what strategy was used. See ADDREF for a possible example of how this might be carried out. Check that this is correct!

2.3.3 Invariance and Equivariance with Functors

Inspired in part by the formalization in [Gav19], [CLLS24] presents a categorical framework that describes neural networks as functors and uses their functorial nature to impose invariance and equivariance constraints. In particular, it is shown that such constraints are helpful in accounting for shift and imbalance in covariates when pooling medical image datasets.

Categorical Structure of Data

The first challenge in using functors to enforce invariance and equivariance constraints is giving data a categorical structure. The strategy employed by [CLLS24] consists in defining a data category $\mathcal{S}$ whose objects s are data points and whose morphisms $f : s_1 \to s_2$ represent differences in covariates.

An example considered in the paper is the following: suppose the objects s are comprised of brain scans and associated information concerning patient age and other covariates. The goal is to develop a model trained to diagnose Alzheimer's disease from the scans. An example of morphism in such data category is $f_x : s_1 \to s_2$, which indicates a difference of x years in age: $s_2.\text{age} = s_1.\text{age} + x$.

Since we are dealing with a classification task, the dataset here has labels. It is important not to include the labels in the data category, else any classifier model would just read such labels instead of learning to predict them. Use the notation $\mathbf{y}_s$ to represent the label associated to s. Since the dataset has labels while the data category must not have them, we feel justified in using the phrase data category in place of the phrase dataset category employed above.

Invariance and Equivariance

Now, if we consider another category $\mathcal{T}$, we can use a functor $F : \mathcal{S} \to \mathcal{T}$ to project $\mathcal{S}$ onto $\mathcal{T}$. Learning such functor instead of a simple map between objects is advantageous because the functoriality axioms imply that F automatically satisfies equivariance constraints: if $g : s_1 \to s_2$,

$$F(g) : F(s_1) \to F(s_2). \tag{2.3}$$

Eq. 2.3 is a categorical generalization of the more usual group theoretical notion of invariance, defined as

$$f(g \cdot s) = g \cdot f(s), \tag{2.4}$$

where $s : S$, $g : G$, G is a group, and S is a G-set. To be precise, *Eq. 2.4* is equivalent to *Eq. 2.3* if $\mathcal{S}$, $\mathcal{T}$ are Borel spaces and g, $F(g)$ are group actions, as highlighted in [CLLS24].

Hence, a functor F is automatically equivariant with respect to any change $g : s_1 \to s_2$ in covariates incarnated as a morphism in the domain category. Invariance with respect to g is not much harder to define: it suffices to impose $F(s_1) = F(s_2)$ and $F(g) = \mathrm{id}_{F(s_1)}$.

Classification Task

A functor F able to satisfy invariance and equivariance can be used as a first step to create a classifier that satisfies such constraints. The architecture proposed by [CLLS24] consists in two modules: (i) an autoencoder whose encoder is $F : \mathcal{S} \to \mathcal{T}$ and whose decoder is $F^{-1} : \mathcal{T} \to \mathcal{S}$; (ii) a functor $C : \mathcal{T} \to \mathbf{Free}(\mathbb{N})$ that actually does the classification. Thus, the whole model admits the compact representation $F \fatsemi C : \mathcal{S} \to \mathbf{Free}(\mathbb{N})$ (a diagram representing such structure can be seen in ADDIM).

In the model described above, $\mathcal{T}$ acts as a latent space. The hope is that the latent representation of the data in $\mathcal{S}$ satisfies the given equivariance and invariance constraints, so that the actual classification operated by $\mathcal{C}$ naturally satisfies the requirements as well. This same strategy is used in non-categorical approaches such as ADDREF. The main advantage to the categorical formalization presented by [CLLS24] is that an arbitrary number of covariates can be handled at once without any additional complexity. This is in stark contrast with ADDREF, where at most two covariates can be handled at once, and with ADDREF, where increasing the number of covariates requires a much more complicated training pipeline. The authors argue that the framework can also be easily adapted to regression tasks by replacing $\mathbf{Free}(\mathbb{N})$ with $\mathbf{Free}(\mathbb{R})$.

Remark 69. It only makes sense to consider F^{-1} is F is fully-faithful. In practical applications, this isn't usually a concern.

Training

What is now needed is an algorithm able to learn F and C. [CLLS24] suggests implementing $\mathcal{T}$ as a vector space category (*Def.* **??**) and to train a neural network to embed data points $s : \mathcal{S}$ as vectors $F(s) = v_s$, and covariate morphisms f as matrices w_f. It is often useful to restrict ourselves to representing morphisms using orthogonal matrices, as the latter can be inverted by transposition, which is computationally efficient. As shown in the paper, being able to invert such morphisms offers great benefits.

The embeddings and matrices can now be trained using a linear combination of three separate losses: $\mathcal{L} = \gamma_1 \mathcal{L}_r + \gamma_2 \mathcal{L}_p + \gamma_3 \mathcal{L}_s$, where γ_1, γ_2, and γ_3 are hyper-parameters. Here, $\mathcal{L}_r$ is a reconstruction loss, which makes sure that F is invertible and that its inverse accurately reconstructs the original data; $\mathcal{L}_p$ is a prediction loss, which makes sure that $F \fatsemi G$ accurately predicts the labels of the data; $\mathcal{L}_s$ a

structure loss, which makes sure that F acts as a functor and not just a map. In formulae,

$$\mathcal{L}_r = \sum_{s:\mathcal{S}} \|s - (F^{-1} \circ F)(s)\|_2^2,$$

$$\mathcal{L}_p = \sum_{s:\mathcal{S}} \mathsf{crossentropy}(\mathbf{y}_s, (C \circ F)(s)),$$

$$\mathcal{L}_s = \sum_{\substack{s_1,s_2:\mathcal{S} \\ f:s_1 \to s_2}} \|W_f F(s_1) - F(s_2)\|_2^2.$$

Experimental Results

The authors of [CLLS24] prove the validity of the proposed approach with two interesting experiments: a proof of concept trained on the MNIST dataset, and a working classifier trained on the ADNI brain imaging dataset.

The proposed MNIST model implements equivariance with respect to increments, rotations, and zooming. It is shown in the paper that representing the associated morphisms with orthogonal matrices allows such morphisms to be inverted and combined in the latent space. A subsequent application of F^{-1} shows the results of the aforementioned manipulation in human-understandable form. Such results are indeed very promising: the authors are able to combine rotations and zooming successfully, even though the network was only trained to apply them separately (see ADDIM) for an example.

The ADNI classifier model also shows very promising results which are on par with state-of-the-art models that do not use categorical tools. The comparison takes place according to accuracy of prediction, maximum mean discrepancy, and adversarial validation.

Bibliography

[BBCV21] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[BCS06] Richard F Blute, J Robin B Cockett, and Robert AG Seely. Differential categories. *Mathematical structures in computer science*, 16(6):1049–1083, 2006.

[CCG+19] Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon Plotkin, and Dorette Pronk. Reverse derivative categories. *arXiv preprint arXiv:1910.07065*, 2019.

[CEG+24] Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregian, Bartosz Milewski, Emily Pillmore, and Mario Román. Profunctor optics, a categorical update. *Compositionality*, 6, 2024.

[CG22] Matteo Capucci and Bruno Gavranović. Actegories for the working amthematician. *arXiv preprint arXiv:2203.16351*, 2022.

[CGG+22] Geoffrey SH Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical foundations of gradient-based learning. In *European Symposium on Programming*, pages 1–28. Springer International Publishing Cham, 2022.

[CLLS24] Sotirios Panagiotis Chytas, Vishnu Suresh Lokhande, Peiran Li, and Vikas Singh. Pooling image datasets with multiple covariate shift and imbalance, 2024.

[FJ19] Brendan Fong and Michael Johnson. Lenses and learners. *arXiv preprint arXiv:1903.03671*, 2019.

[FST19] Brendan Fong, David Spivak, and Rémy Tuyéras. Backprop as functor: A compositional perspective on supervised learning. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2019.

[Gav19] Bruno Gavranović. Compositional deep learning. *arXiv preprint arXiv:1907.08292*, 2019.

[Gav24] Bruno Gavranović. Fundamental components of deep learning: A category-theoretic approach, 2024.

[GLD+24a] Bruno Gavranović, Paul Lessard, Andrew Dudzik, Tamara von Glehn, João GM Araújo, and Petar Veličković. Categorical deep learning: An algebraic theory of architectures. *arXiv preprint arXiv:2402.15332*, 2024.

[GLD+24b] Bruno Gavranović, Paul Lessard, Andrew Joseph Dudzik, Tamara von Glehn, João Guilherme Madeira Araújo, and Petar Veličković. Position: Categorical deep learning is an algebraic theory of all architectures. In *Forty-first International Conference on Machine Learning*, 2024.

[JR97] Bart Jacobs and Jan Rutten. A tutorial on (co) algebras and (co) induction. *Bulletin-European Association for Theoretical Computer Science*, 62:222–259, 1997.

[Ril18] Mitchell Riley. Categories of optics. *arXiv preprint arXiv:1809.00738*, 2018.

[SGW21] Dan Shiebler, Bruno Gavranović, and Paul Wilson. Category theory in machine learning. *arXiv preprint arXiv:2106.07032*, 2021.

[Ste15] Albert Steckermeier. Lenses in functional programming. *Preprint, available at https://sinusoid.es/misc/lager/lenses.pdf*, 2015.

[SY21] Artan Sheshmani and Yi-Zhuang You. Categorical representation learning: morphism is all you need. *Machine Learning: Science and Technology*, 3(1):015016, 2021.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wis08] Robert Wisbauer. Algebras versus coalgebras. *Applied Categorical Structures*, 16(1):255–295, 2008.

[WZ21] Paul Wilson and Fabio Zanasi. Reverse derivative ascent: A categorical approach to learning boolean circuits. *arXiv preprint arXiv:2101.10488*, 2021.

[WZ22] Paul Wilson and Fabio Zanasi. Categories of differentiable polynomial circuits for machine learning. In *International Conference on Graph Transformation*, pages 77–93. Springer, 2022.

Acknowledgements

I wish to acknowledge the essential role my advisor Professor F. Zanasi has had in guiding me through the process of writing this thesis. He introduced me to applied category theory, to machine learning, and to the world of academic research, and I will be forever grateful for it. I also wish to thank my family and my friends, who supported my throughout this journey, and without whom all of this would not have been possible. To all of you, my most sincere gratitude.