To my beloved
Benedetta

Introduction

English version

Italian version

Traduzione italiana dell'introduzione.

Contents

Introduction **i**

1 Parametric Optics for Gradient-Based Learning **1**
1.1 Categorical toolkit 1
1.1.1 Actegories 2
1.1.2 The **Para** construction 2
1.1.3 Optics 4
1.1.4 Weighted optics 5
1.1.5 Differential categories 7
1.1.6 Parametric lenses 9
1.2 Supervised learning with parametric lenses 9
1.2.1 Model, loss, optimizer, learning rate 10
1.2.2 Weight tying, batching, and the learning iteration 11
1.2.3 Empirical evidence 12
1.3 Future directions and related work 12
1.3.1 Learners 12
1.3.2 Exotic differential categories 13
1.3.3 Functional reverse-mode automatic differentiation 14

2 From Classical Computer Science to Neural Networks **15**
2.1 Categorical toolkit 15
2.1.1 (Co)algebras 15
2.1.2 Integral transform 17
2.2 Categorical deep learning 19
2.2.1 From GDL to CDL 19
2.2.2 (Co)inductive definitions for RNNs 20
2.3 Algorithmic alignment: GNNs and dynamic programming 22
2.3.1 Integral transforms for GNNs and dynamic programming 22
2.3.2 Asynchronous algorithmic alignment 24
2.4 Future directions and related work 26
2.4.1 Differentiable causal computations 26
2.4.2 Sheaf neural networks 27

3 Functor Learning **31**
3.1 Using functors to separate layers of abstraction 31
3.1.1 Schemas, architectures, models, and concepts 31
3.1.2 Datasets influence the semantics of tasks 34
3.2 Categorical representation learning 35

3.2.1 Unsupervised functorial translation 35
3.2.2 Compositional distributional model for NLP 38
3.3 Equivariant neural networks as functors 40
3.3.1 An equivariant classifier to diagnose Alzheimer's disease 40
3.3.2 Transporting algebraic structure into embedding spaces 42

4 Diagrammatic Representations for Neural Networks 45
4.1 Neural circuit diagrams 45
4.1.1 Functor string diagrams 46
4.1.2 Neural circuit diagrams 46
4.2 Diagrams with universal approximators 48
4.2.1 The diagrammatic approach of [KLLWM24] 49
4.2.2 Comparing attention mechanisms 51
4.3 Conclusions and related work 52

Conclusions 55

Bibliography 61

Chapter 1

Parametric Optics for Gradient-Based Learning

Despite the unquestionable success that gradient-based deep learning has enjoyed in recent years, the field is still both young and poorly understood. As mentioned in the introduction, the lack of theoretical underpinnings means that good performance is highly dependent on *ad hoc* choices and empirical euristics leading to brittleness and poorly understood phenomena ([CGG$^+$22], [SGW21], [Gav24]). The ever-growing complexity of deep learning models poses significant challenges both in terms of optimization ([Ell18]) and architectural design ([GLD$^+$24]), and, while there are a number of general purpose deep learning libraries that automatically implement backpropagation and provide tools for designing a wide variety of neural networks, these tools often rely on inelegant machinery difficult to parallelize ([Ell18]). Given the ever-increasing role graident based learning plays in the sciences, in industry, and in every day life, solving these issues is of the outmost importance.

Hence, it would be auspicable to develop a mathematically structured framework for gradient-based learning able to act as a bridge between low-level automatic differentiation and high level architectural specifications ([Gav24]). The great number of architectures developed in recent years and the inherently modular structure of deep neural networks call for a model which is general (that is, not dependent on a specific differentiation algorithm or a specific optimizer) and compositional (that is, we should be able to predict the behavior of the entire model if the behavior of each part is known). Among the various attempts (see e.g. [FST19] or [SXM23]), a promising combination of differential categories, parametrization and optics has been recently proposed by [CCG$^+$19], [CGG$^+$22], [Gav24] and more as full-featured gradient-based framework able to challenge established tools. In this chapter, we illustrate such framework and part of its mathematical foundations.

1.1 Categorical toolkit

Learning 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 gradient-based learning must take these two aspects into consideration. A number of authors (see e.g. [Gav24], and [CGG$^+$22]) have proposed the **Para** construction as a categorical model of parameter dependence and various categories of optics as the right categorical abstraction for bidirectionality.

1.1.1 Actegories

Before we can deal with parametric maps, we need to find a way to glue input/output spaces to parameter spaces, so that such maps have well-defined 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 further information).

Definition 1 (Actegory). *Let $(\mathcal{M}, I, \otimes)$ be a strict symmetric monoidal category. A $\mathcal{M}$-actegory is a tuple $(\mathcal{C}, \bullet, \eta, \mu)$, where $\mathcal{C}$ is a category, $\bullet : \mathcal{M} \times \mathcal{C} \to \mathcal{C}$ is a functor, and η and μ are natural isomorphisms enforcing $I \bullet C \overset{\eta_C}{\cong}$ and $(M \bullet (N \bullet C)) \overset{\mu_{M,N,C}}{\cong} (M \otimes N) \bullet C$. The isomorphisms η and μ must also satisfy coherence conditions. If η and μ are identical transformations, we say that the actegory is strict.*

Remark 2. Although the requirement for strictness is somewhat restrictive, we will proceed under the assumption that the actegories we encounter are strict to streamline notation.

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 strict symmetric monoidal category and let $(\mathcal{C}, \bullet, \eta, \mu)$ be a strict actegory. Suppose $\mathcal{C}$ has a monoidal structure $(J, \boxtimes)$. Then we say that $(\mathcal{C}, \bullet)$ is monoidal if the underlying functor $\bullet$ is 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$^+$24]. 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 do not list here.*

1.1.2 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 $\mathcal{M}$, that is, maps in the form $P \bullet A \to B$. 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 introduced in simplified form 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 A can serve as input/output spaces, its 1-cells (P, f) can serve as parametric maps, and, finally, its 2-cells r can serve as reparametrizations.

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

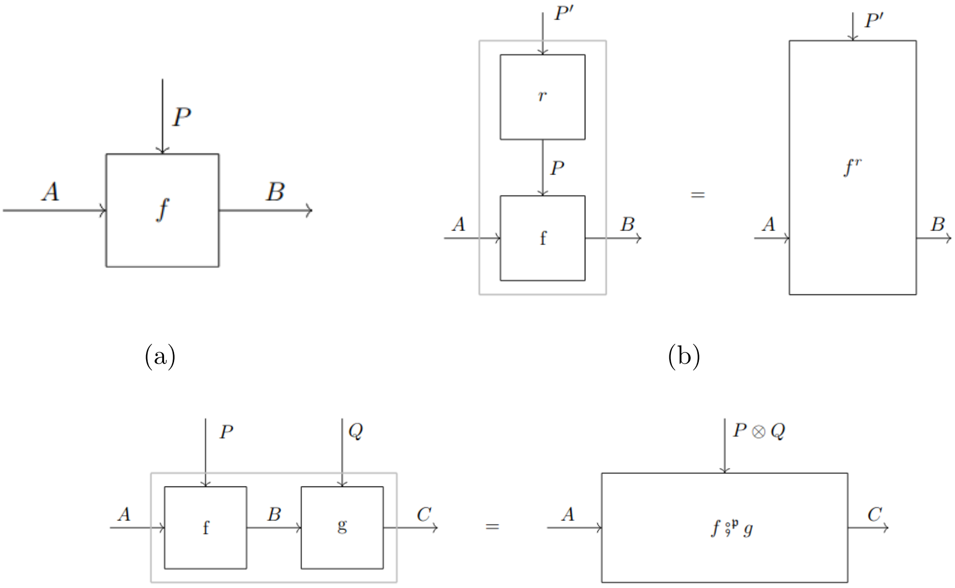
- *The 0-cells are the objects of $\mathcal{C}$.*

- *The* 1*-cells are pairs* $(P, f) : A \to B$*, where* $P : \mathcal{C}$ *and* $f : P \bullet A \to B$*.*
- *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*

$$(P, f) \mathbin{;} (Q, g) = (Q \otimes P, (Q \bullet f) \mathbin{;} g).$$

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

It is quite handy to represent the cells of $\mathbf{Para}_\bullet(\mathcal{C})$ *using the string diagram notation illustrated in Fig. 1.1. The* **Para** *construction has a dual* **coPara** *construction whose* 1*-cells* $f : \mathbf{coPara}_\bullet(\mathcal{C})(A, B)$ *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. The reader can find a complete definition in [Gav24].*



(a) (b)

(c)

Figure 1.1: String diagrams representing (a) a parametric morphism, (b) a reparametrization of a parametric morphism, (c) a composition of parametric morphisms. (Images taken from [Gav24].)

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 (as we do in this thesis), 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. This meaningfully recovers the 1-categorical perspective of [FST19].

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 related, 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.*

1.1.3 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 to generalize 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]).

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:*

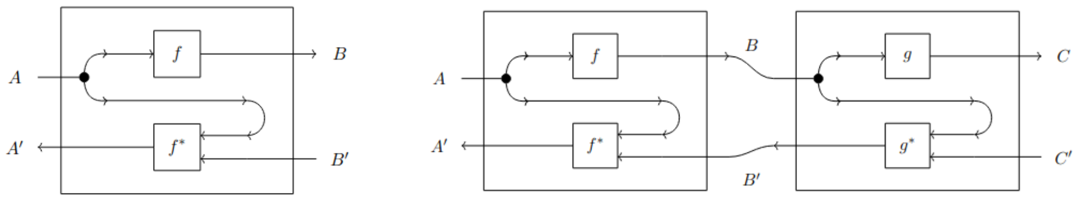
- *an object of* $\mathbf{Lens}(\mathcal{C})$ *is a pair* $\left(\begin{smallmatrix} A \\ A' \end{smallmatrix}\right)$ *of objects in* $\mathcal{C}$*;*
- *a* $\left(\begin{smallmatrix} A \\ A' \end{smallmatrix}\right) \to \left(\begin{smallmatrix} B \\ B' \end{smallmatrix}\right)$ *morphism (or lens) is a pair* $\left(\begin{smallmatrix} f \\ f' \end{smallmatrix}\right)$ *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* $\left(\begin{smallmatrix} f \\ f' \end{smallmatrix}\right)$*, whereas* f' *is known as the bacward pass;*
- *given* $\left(\begin{smallmatrix} A \\ A' \end{smallmatrix}\right) : \mathbf{Lens}(\mathcal{C})$*, the associated identity lens is* $\left(\begin{smallmatrix} 1_A \\ \pi_1 \end{smallmatrix}\right)$*;*
- *the composition of* $\left(\begin{smallmatrix} f \\ f' \end{smallmatrix}\right) : \left(\begin{smallmatrix} A \\ A' \end{smallmatrix}\right) \to \left(\begin{smallmatrix} B \\ B' \end{smallmatrix}\right)$ *and* $\left(\begin{smallmatrix} g \\ g' \end{smallmatrix}\right) : \left(\begin{smallmatrix} B \\ B' \end{smallmatrix}\right) \to \left(\begin{smallmatrix} C \\ C' \end{smallmatrix}\right)$ *is*

$$\left(\begin{smallmatrix} f ⨾ g \\ \langle \pi_0, \langle \pi_0 ⨾ f, \pi_1 \rangle ⨾ g' \rangle ⨾ f' \end{smallmatrix}\right).$$

Lenses are best thought of in the helpful language of the string diagrams illustrated in Fig. 1.2.

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 forward and bavkward passes, 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



(a) (b)

Figure 1.2: String diagrams representing (a) a lens $\begin{pmatrix} f \\ f^* \end{pmatrix}$, (b) the composition of two lenses $\begin{pmatrix} f \\ f^* \end{pmatrix}$ and $\left(\begin{smallmatrix} g \\ g^* \end{smallmatrix}\right)$. (Images taken from [CGG$^+$22].)

between the upper components and the lower component. We will provide the definition given by [Ril18][1], 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 $\left(\begin{smallmatrix} A \\ A' \end{smallmatrix}\right)$ of objects in $\mathcal{C}$;*
- *a $\left(\begin{smallmatrix} A \\ A' \end{smallmatrix}\right) \to \left(\begin{smallmatrix} B \\ B' \end{smallmatrix}\right)$ morphism (or optic) is a pair $\begin{pmatrix} f \\ f' \end{pmatrix}$ 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 $\begin{pmatrix} f \\ f' \end{pmatrix}$ are also quotiented by an equivalence relation that allows for reparametrization of the residual space and effectively makes it inaccessible;*
- *the identity on $\left(\begin{smallmatrix} A \\ A' \end{smallmatrix}\right)$ is the optic represented by $(\lambda_A^{-1}, \lambda_A)$.*

See [Ril18] for the more information about how optics compose and about the associated string diagrams.

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.

1.1.4 Weighted optics

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

Definition 10 (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

---

[1] [Ril18]also provides a more versatile (but more sophisticated) definition of optics that relies on coends. Under the coend formalism,

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

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

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

We are finally ready to define weighted optics[2].

Definition 11 (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 π_0 : **Cat** $\to$ **Set***. More explicitly, π_{0*} quotients the connections provided by reparametrizations.*

Definition 12 (Weighted optics)**.** *Suppose $(\mathcal{C}, \bullet)$ is an $\mathcal{M}$-actegory, suppose $(\mathcal{D}, \blacksquare)$ is an $\mathcal{M}'$-actegory, and suppose $W : \mathcal{M}^{\mathrm{op}} \times \mathcal{M}' \to$* **Set** *is a lax monoidal functor. We define the category of W-weighted optics over the product actegory $(\mathcal{C} \times \mathcal{D}^{\mathrm{op}}, \left(\substack{\bullet \\ \blacksquare^{\mathrm{op}}}\right))$ as*

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

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 $\left(\left({}_{M'} \right), s, \binom{f}{f'} \right)$, 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). We can get a clear "operational" understanding of how a weighted optic 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 then converted into a backward residual in M' by the map s, which is provided by the weight functor. Finally, 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'. A full account of the composition law for weighted optics can be found on [Gav24]. As stated in [Gav24], since **coPara** can be given a monoidal structure, we can also give $\mathbf{Optic}^{W}_{\left(\substack{\bullet \\ \blacksquare}\right)}$ one such structure as long as the underlying actegories are monoidal and the weight functor W is braided monoidal.

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. 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, [Gav24] 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.

[Gav24] goes on to apply the **Para** construction onto weighted optics, obtaining parametric weighted optics, which are proposed as a full-featured model for deep learning. The author conjectures that "weighted optics provide a good denotational and operational semantics for differentiation". In its full, generality, this is still an unproven conjecture. However, restricting our attention to a special class $\mathbf{Lens}_A$ of lenses with an additive backward passes yields a fully formal theory of structural backpropagation, which will be illustrated in the rest of the capter, after a short digression on differential categories.

---

[2] Weighted optics also admit a coend definition. Refer to [Gav24] for more information.

1.1.5 Differential categories

Modelling gradient-based learning obviously requires a setting where differentiation can take place. Although it is tempting to directly employ smooth functions over Euclidean spaces, 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 ([PVM$^+$21]), complex spaces ([BQL21]), and so on. Thus, it makes sense to work with an abstract notion of derivative which can then be appropriately implemented depending on the requirements at hand.

One approach to this problem involves the explicit definition of two kidns of differential categories: 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 13 (Cartesian differential category)**.** *A Cartesian differential category (CDC) $\mathcal{C}$ 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 14 (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.*

Example 15. 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 ([Ell18]), 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]). As shown in [WZ22], a large supply of CRDCs can be obtained by providing the generators of a finitely presented Cartesian left-additive category with associated reverse derivatives (as long as the choices of reverse derivative are consistent). Moreover, CRDCs have been recently generalized by [Gav24] to coalgebras associated with copointed endofunctors, which could also increase the number of known CRDCs in the future. The rest if this section is devoted to this generalization.

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.

Definition 16 (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 17. *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 $f' : \mathcal{C}(X \times Y', X')$, which is the approach we illustrate in this thesis. Nevertheless, 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]).

The generalization mentioned above is possible because $\mathbf{Lens}_A$ is an endofunctor.

Definition 18. *We defined* $\mathbf{CLACat}$ *as the category whose objects are Cartesian left-additive categories and whose morphisms are Cartesian left-additive functors (see e.g.[BCS06]).*

Proposition 19. *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 $0_{\binom{X}{X'}} = \binom{0_A}{!_{1 \times A'}} : \binom{1}{1} \to \binom{X}{X'}$ and by the multiplication $+_{\binom{X}{X'}} = \binom{+_A}{\pi_2 \fatsemi \Delta_{A'}} : \binom{X \times X}{X' \times X'} \to \binom{X}{X'}$. ☐

Proposition 20. $\mathbf{Lens}_A : \mathbf{CLACat} \to \mathbf{CLACat}$ *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 $\binom{X}{X'} \mapsto \binom{F(X)}{F(X')}$ and maps $\binom{f}{f'} \mapsto \binom{F(f)}{\underline{f}}$, 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 21. $\mathbf{Lens}_A$ *has a copointed structure*[3]*.*

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. ☐

Hence, [Gav24] defines generalized CRDCs as follows.

Definition 22 (Generalized Cartesian reverse differential category)**.** *A generalized Cartesian reverse differential category is a coalgebra for the pointed endofunctor* $\mathbf{Lens}_A$*.*

---

[3] An endofunctor $F : \mathcal{C} \to \mathcal{C}$ is copointed if it is endowed with a natural transformation $\epsilon : F(\mathcal{C}) \Rightarrow \mathrm{id}_{\mathcal{C}}$.

Explicitly, a colagebra for $\mathbf{Lens}_A$ 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 $\mathbf{R}_\mathcal{C} \fatsemi \epsilon_\mathcal{C} = \mathrm{id}_\mathcal{C}$. 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.

1.1.6 Parametric lenses

We conclude this section discussing the relation between the **Para** construction and the $\mathbf{Lens}_A$ endofunctions. [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 [CGG$^+$22]) for the reader's convenience.

Definition 23 (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. 1.3 to see a string diagram that shows the inner workings of a parametric lens.

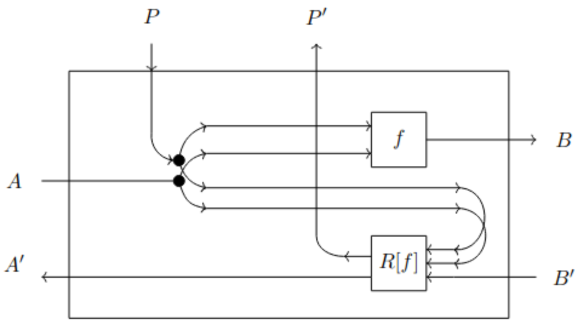


Figure 1.3: String diagrams representing the inner workings of a parametric lens. (Images taken from [CGG$^+$22].)

1.2 Supervised learning with parametric lenses

In this section, we show how parametric lenses can be used to model supervised gradient-based learning ([CGG$^+$22], [Gav24], [SGW21]). While lenses are not as general as weighted optics, it is shown in [CGG$^+$22] that they are powerful enough for most purposes and that there is some empirical evidence of their performance and applicability. The paper also discusses the use of parametric lenses in modeling unsupervised deep learning and deep dreaming, but we do not have the space to discuss them here.

1.2.1 Model, loss, optimizer, learning rate

Supervised gradient-based learning can be modeled using parametric lenses as follows:

1. we can design an architecture (P, Model) as a parametric morphism in $\mathbf{Para}_\bullet(\mathcal{C})$ for some generalized CRDC $(\mathcal{C}, \mathbf{R}_\mathcal{C})$;
2. we can use the functor $\mathbf{R}_\mathcal{C}$ to endow (P, Model) with its reverse derivative $\mathrm{R}[(P, \mathsf{Model})]$, yielding a lens in $\mathbf{Para}_\bullet(\mathbf{Lens}_A(\mathcal{C}))$;
3. we can use 2-categorical machinery of $\mathbf{Para}_\bullet(\mathbf{Lens}_A(\mathcal{C}))$ to provide a loss function, a learning rate, and optimizer, which can be assembled onto $\mathbf{R}_\mathcal{C}(P, \mathsf{Model})$ to yield a parameter update;
4. we can use copy maps from the Cartesian structure of $\mathcal{C}$ to create a learning iteration.

The theory of parametric optics and differential categories does not offer explicit insight with respect to architecture design, so we will assume a good architecture as already been designed (at least for now). Given an architecture (P, Model), it can be embedded into $\mathbf{Para}_\bullet(\mathbf{Lens}_A(\mathcal{C}))$ as a lens $\binom{P\times A}{P\times A} \to \binom{B}{B}$ by breaking it up into its basic components (such as linear layers, convolutional layers, etc.), augmenting such components with their reverse derivatives, and the composing the resulting lenses. The backward pass of the composition is the reverse derivative of its forward pass because $\mathbf{R}_\mathcal{C}$ is a functor[4]. Many examples can be found in [CGG$^+$22].

Updating the parameters based on data requires a loss function, an optimizer and a learning rate. Loss functions can be implemented as parametric lenses which take in predictions as input and labels as parameters. The output they produce can be considered the actual loss that needs to be differentiatied. Given a model parametric lens $\binom{\mathsf{Model}}{\mathrm{R}[\mathsf{Model}]} : \binom{P\times A}{P\times A} \to \binom{B}{B}$ and a loss parametric lens $\binom{\mathsf{Loss}}{\mathrm{R}[\mathsf{Loss}]} : \binom{B\times B}{B\times B} \to \binom{L}{L}$, the composition $\binom{\mathsf{Model}}{\mathrm{R}[\mathsf{Model}]} \mathbin{⨟} \binom{\mathsf{Loss}}{\mathrm{R}[\mathsf{Loss}]}$ takes in features as input and takes model parameters and labels as parameters. Then, this information is used to compute the loss associated with the model predictions. See *Fig. 1.4 (a)* for the associated string diagram.

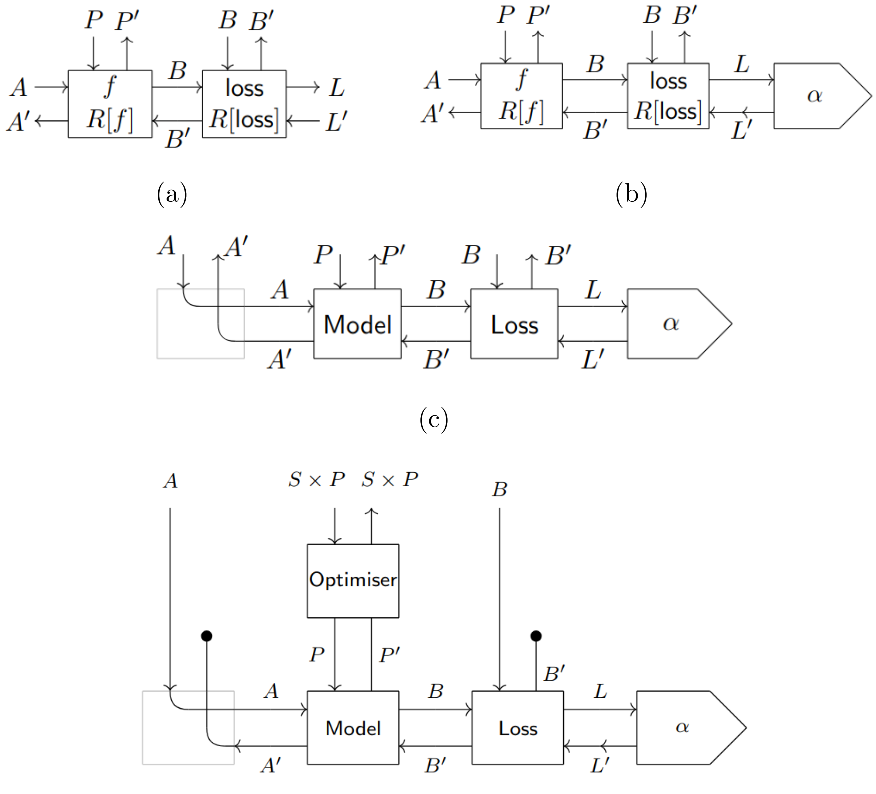
It can be helpful to think about dangling wires in the diagrams as open slots where other components can be plugged. For instance, the diagram of *Fig. 1.4 (a)* has dangling wires labeled with L on its right. We can use a learning rate lens α to link these wires and allow forward-propagating information 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. 1.4 (b)* 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 Model in *Fig. 1.4 (b)* 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. 1.4 (c)*. 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.

In the supervised learning lens in *Fig. 1.4 (c)*, there are actually has two more dangling wires we do not have any use for: namely, the second input A of the model and the second parameter B of the loss functions. These are not a problem as we can just plug them with delete maps taken from the Cartesian structure of $\mathcal{C}$. We can also bend the input wires with an appropriate lens to turn them into parameters. We are then left with a $\binom{1}{1} \to \binom{1}{1}$ parametric lens with parameter space $\binom{A}{1} \times \binom{P}{P} \times \binom{B}{1}$. This lens is pictured in *Fig. 1.4 (d)*.

---

[4] As highlighted by [SGW21], the diagram for the backward pass of the composition of two lenses looks exactly like the diagram describing the chain rule for reverse derivatives, which is what makes $\mathbf{R}_\mathcal{C}$ a well-defined functor.



(a) (b) (c) (d)

Figure 1.4: String diagrams representing (a) the composition of a model lens and an optimizer lens (b) the composition of a model lens, an optimizer lens, and a learning rate lens, (c) a composition of lenses representing a complete supervised learning algorithm, (d) the same lens of the previous point withuseless wires plugged. (Images taken from [CGG$^+$22].)

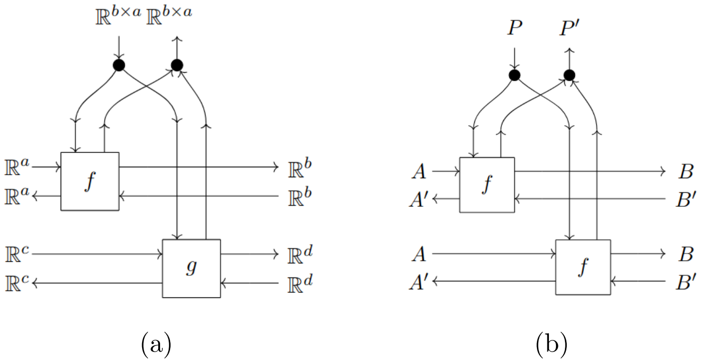
1.2.2 Weight tying, batching, and the learning iteration

Both [CGG$^+$22] and [Gav24] emphasize the essential role played by weight tying 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 (see *Fig. 1.5* (a)): 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

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

Weight tying can also be used for batching: batching is implemented by instantiating n different copies of our supervised learning lens (comprised of model, loss function, and learning rate) and tying the parameters to a unique value. Then, it suffices to feed the n data points to the n lenses, and we can optimize across a single parameter (see *Fig. 1.5* (b)).

[CGG$^+$22] introduces a possible representation for the whole learning iteration of a supervised learning model within the parametric lens framework. The paper suggests extracting the bacward pass of the lens in *Fig. 1.4 (d)* and reframing it as a $P \to P$ parametric map with parameters $A \times B$. Since this is an endomap, it can be composed n times with itself to obtain a $P \to P$ map, which is proposed as a model of the learning iteration. While this approach requires breaking lenses apart, it is markedly simple.



(a) (b)

Figure 1.5: String diagrams representing (a) weight trying and (b) batching, both implemented using the copy maps of a Cartesian category. (Images taken from [CGG$^+$22].)

1.2.3 Empirical evidence

Empirical evidence for the effectiveness of the parametric lens framework discussed in this section can be found in [CGG$^+$22], where the authors go on to implement a Python libraries for gradient-based learning based on these ideas. They use the library to develop a MNIST classifier, obtaining comparable accuracy to models developed using traditional tools.

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'.

1.3 Future directions and related work

The parametric optic framework discussed in this chapter is very promising, but there is still a lot of work that needs to be done so that it can reach its full potential. For instance, [Gav24] conjectures that weighted optics can be used in its full generality to model differentiation in cases which are not covered by lenses. For instance, lenses cannot model AD algorithms that do not use gradient checkpointing, while weighted optics are conjectured to be able to do so. [Gav24] also suggests investigating locally graded categories as potential replacements for actegories, and also investigating the applications of parametric optics to meta-learning, that is deep learning where the optimizers themselves are learned. Moreover, [CGG$^+$22] conjectures that some of the axioms of CRDC may be used to model higher order optimization algorithms. Finally, as suggested by [CGG$^+$22], hopefully future work will allow the parametric optic framework to encompass non-gradient based optimizers such as the ones used in probabilistic learning. See [SGW21] for more on this topic.

We conclude this chapter by discussing three other directions of machine learning research that are closely related to the framework of parametric optics.

1.3.1 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 24 (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[5] *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 do not 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 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].

1.3.2 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 different learning outcomes, both in terms of accuracy of the model and in terms of training computational costs ([WZ22]). For instance, is argued in [WZ22] that polynomial circuits 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]). Using such exotic differential categories may be of great advantage because they might be able to better reflect the intrinsic computational limits of computer arithmetics, leading to more efficient learning ([WZ22]).

---

[5] 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.

1.3.3 Functional reverse-mode automatic differentiation

Finally, we wish to highlight the similarities between the formal theory of differential categories illustrated here and the work in [Ell18]. 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 [Ell18], 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 [Ell18] because of the latter’s programming focus. See [SGW21] for more information regarding this comparison.

Chapter 2

From Classical Computer Science to Neural Networks

Classical computer science focuses on discovering algorithms, that is ordered sequences of steps which operate in precisely set, idealized conditions and have strong guarantees of correctness due to their exact mathematical formulations. Neural networks, on the other hand, are able to work in messy, real-world conditions, but offer very so few guarantees of correctness that their performance is often described as *unreasonably* good. Moreover, whereas algorithms generalize very well (most software engineers will only need a few dozen algorithms in their entire career), neural networks are often completely helpless when pitted against out of distribution inputs. Hence, algorithms and neural networks can be seen as complementary opposites ([VB21], [VBB$^+$22]).

Recent attempts going under the label of *neural algorithmic reasoning* (see [VB21] for a very short introduction to the subject) have tried to get the best of both worlds by training neural networks to execute algorithms (see e.g [IKP$^+$22]). The CLRS benchmark (introduced by [VBB$^+$22]) uses graphs to represent the computations associated with a few classical algorithms from the famous CLRS introductory textbook ([CLRS22]) so that graph neural networks (GNNs) can be trained to learn these algorithms. The benchmark has spurred a large amount of research in this direction, with very promising results.

More generally, linking machine learning to classical computer science may unlock interesting advances. For example, recovering neural networks as parametric versions of known algorithms may classify existing architectures in a conceptually clear manner and may even help develop new neural network architectures by taking inspiration from well-researched classical notions. In this chapter, we illustrate two lines of inquiry which use category theory to build such a bridge: *categorical deep learning* and an interesting categorical approach to *algorithmic alignment.* Before treating such topics, we will go on a short tangent categorical tangent regarding (co)algebras and the integral transform.

2.1 Categorical toolkit

2.1.1 (Co)algebras

Algebras and coalgebras are a categorical formalization of the principles of induction and coinduction. Induction and coinduction are fundamental to computer science because they allow us to give precise definitions for many data structures and to formalize recursive and corecursive algorithms on such structures. We will touch on (co)algebras very briefly but we refer interested readers to [JR97] and [Wis08] for further detail.

Definition 25 ((Co)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 $a : \mathcal{C}(F(A), A)$. A coalgebra is a pair (A, a) where $A : \mathcal{C}$ and $a : \mathcal{C}(A, F(A))$. In both cases A is known as carrier set and a as structure map.*

(Co)algebras can also be defined on monads: the only difference between (co)algebras over an endofunctor and (co)algebras over a monad is that the latter also need to be compatible with the monad structure, that is satisfy commutative diagrams that represent coherence conditions (see [GLD$^+$24]). (Co)algebras over the same functor can be given a categorical structure by using the following notion of homomorphism.

Definition 26 (Homomorphisms of (co)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 Fig. 2.1 (a) is commutative.*

Now suppose (A, a) and (B, b) are coalgebras. A homomorphism between them is a map $f : \mathcal{C}(A, B)$ such that the diagram in Fig. 2.1 (b) is commutative.

$$\begin{tikzcd}
T(A) \arrow[r, "T(f)"] \arrow[d, "a"] & T(B) \arrow[d, "b"] \\
A \arrow[r, "f"] & B
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
T(A) \arrow[r, "T(f)"] & T(B) \\
A \arrow[r, "f"] \arrow[u, "a"'] & B \arrow[u, "b"']
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
T(A) \arrow[r, "T(f)"] \arrow[d, "a"'] & T(B) \arrow[d, "b"] \arrow[dl, Rightarrow, "\kappa"'] \\
A \arrow[r, "f"'] & B
\end{tikzcd}$$

(c)

Figure 2.1: Conditions under which $f : (A, a) \to (B, b)$ is (a) an algebra homomorphism, (b) a coalgebra homomorphism, (c) a lax algebra homomorphism. (Image (c) is taken from [GLD$^+$24].)

The main intuition behind the notions of algebra and coalgebra is the following: the underlying functor defines a signature for the (co)algebraic structure; the structure of an algebra is a constructor that takes data from $F(A)$ and uses it to build data from A, whereas the structure of a coalgebra observes data from A and produces an observation in the form of data from $F(A)$; (co)algebra homomorphism are arrows that preserve the underlying structure. Consider the following clarifying examples from [GLD$^+$24].

Remark 27. In the examples below we use polynomial and exponential expressions to define endofunctors over **Set**. In this context, X is the argument of the functor, $\times$ is the Cartesian product, $+$ is the disjoint union, $\langle f, g\rangle$ is the pairing induced by $\times$, $[f, g]$ is the pairing induced by $+$, and B^A is the set of functions $A \to B$. The $\times$ operator is assumed to take precedence over the $+$ operator. Similarly, the exponential operator is assumed to take precedence over the $\times$ operator.

Example 28 (Lists)**.** Let A be a set. Consider the 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, $\mathsf{Nil} : 1 \to \mathit{List}(A)$ 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 $l \cup \{a\}$. The algebra $(\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}])$ describes lists in $\mathsf{List}(A)$ inductively as object formed by concatenating elements of A to other lists in $\mathsf{List}(A)$. The base case is the empty list.

Example 29 (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 m'. This is a coinductive description of Mealy machines.

Remark 30. Notice how the description we have given of Mealy machines does 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 tracks 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 link between (co)algebras and (co)induction does not stop at the definition level. The example below shows that an algebra homomorphism can model a recursive fold procedure. A similar corecursive unfold procedure can be defined by using a coalgebra homomorphism (see [GLD$^+$24] for further detail).

Example 31 (List folds)**.** Consider the algebra $(\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}])$ of lists from *Ex. 28*, 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 $1 + A \times X$.

The notion of (co)algebra over a functor can be generalized to the sphere 2-categories, defining the notion of (co)algebra over a 2-endofunctor. The basic concepts stay the same but the commutativity of the diagrams definining (co)algebra homomorphisms is relaxed into lax-commutativity. A square diagram of 1-cells is lax-commutative if there exists a 2-cell that carries the composition of the top and the right edge onto the composition of the left and bottom edge, as in *Fig. 2.1 (c)*. Once again we refer to [GLD$^+$24] for further information.

We conclude the section with the following proposition ([GLD$^+$24]), which shows that actegorical strong endofunctors induce 2-endofunctors over parametric categories. We will use the proposition to study parametric versions of the endofunctors of *Ex. 28* and *Ex. 29*.

Proposition 32. *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-endofunctor $\mathbf{Para}(F) : \mathbf{Para}_\bullet(\mathcal{C}) \to \mathbf{Para}_\bullet(\mathcal{C})$.*

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.

□

2.1.2 Integral transform

Remark 33. In accordance with the notation of [DV22] and [DvGPV24], we use $[A, B]$ to represent the set of $A \to B$ functions, where A and B are sets.

Suppose $(R, \oplus, \otimes)$ is a commutative semiring. An integral transform is a transformation that carries a function in $[W, R]$ to a function in $[Z, R]$ following a precise chain of steps. Integral transforms[1] have been introduced by [DV22] to provide a single formalism able to describe both dynamic programming and GNNs. Integral transforms can be encoded as polynomial spans.

Definition 34 (Polynomial span)**.** *A polynomial span is a triplet* $(i : X \to W, p : X \to Y, o : Y \to Z)$ *of morphisms in* **FinSet***, the category of finite sets and functions.* i *is known as input,* p *as process,* o *as ouput.* W *is known as input set,* X *as argument set,* Y *as message set,* Z *as ouput set. We also ask that the fibers of* p *have total orderings*[2]*. The polynomial span* (i, p, o) *can be graphically represented as the string diagram in Fig. 2.2 (a).*

Definition 35 (Integral transform)**.** *Let* $(R, \oplus, \otimes)$ *be a commutative semiring. Let* $(i : X \to W, p : X \to Y, o : Y \to Z)$ *be a polynomial span. The associated integral transform is the triplet* $(i^* : [W, R] \to [X, R], p_\otimes : [X, R] \to [Y, R], o_\oplus : [Y, R] \to [Z, R])$*, where:*

1. i^* *is the pullback mapping* $f \mapsto i \mathbin{;} f$*;*
2. $p_\otimes$ *is the argument pushforward mapping*

$$p_\otimes(a)(u) = \bigotimes_{e \in p^{-1}(u)} a(e);$$

3. $o_\otimes$ *is the message pushforward mapping*

$$o_\oplus(m)(v) = \bigoplus_{e \in o^{-1}(v)} m(e).$$

The integral transform $(i^*, p_\otimes, o_\oplus)$ *can be represented by the diagram in Fig. 2.2 (b).*

$$\begin{tikzcd}
X \arrow[r, "p"] \arrow[d, "i"] & Y \arrow[d, "o"] \\
W & Z
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
{[X, R]} \arrow[r, "p_\otimes"] & {[Y, R]} \arrow[d, "o_\oplus"] \\
{[W, R]} \arrow[u, "i^*"] & {[Z, R]}
\end{tikzcd}$$

(b)

Figure 2.2: (a) A polynomial span and (b) the associated integral transform. (Images taken from [DV22].)

Remark 36. Whereas defining i^* is quite straight-forward, defining $p_\otimes$ and $o_\otimes$ is more difficult because the arrows p and o point in the wrong direction, which implies that the underlying functions must be inverted before considering the associated pullbacks. However, inverting non-invertible functions yields functions into the powersets of the original domains. Moreover, if we want to preserve the multiplicity of arguments and messages, we have to construct inverses that go into the sets of multisets over the original domains. Hence why we need $\otimes$ and $\oplus$ to aggregate results over such multisets. The significance these steps will be clarified later on in this chapter.

---

[1] The label integral transform refers to the fact that similar ideas can be used to write categorical definitions for familiar analytical integral transforms ([Wil10]). A similar construct is also used in physics ([EPWJ80]).

[2] Neither we nor [DV22] use this requirement but, as stated in the original paper, the requirement is useful to support functions with non-commuting arguments.

2.2 Categorical deep learning

The optic-based framework we presented in the last chapter provides a structured general-purpose compositional framework for gradient-based learning, but its 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 a better asymptotic cost does in classical computer science. 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. Hence, it makes sense to inform the choice of architecture by informing the choice of constraint. Among the various attempts to develop a principled theory of machine learning constraints, *geometric deep learning* (see e.g. [BBCV21]), or GDL is particularly relevant. Categorical deep learning, or CDL, evolved as categorical generalization of GDL ([GLD$^+$24]).

2.2.1 From GDL to CDL

Geometric deep learning focuses on equivariance constraints defined with respect to group actions, according to the following definition.

Definition 37 (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.*

The GDL approach prescribes that a desired constraint should be expressed in terms of equivariance with respect to group action so that associated equations can be derived. Solving such equations usually implies tying weights, which reduces the net number of parameters in the models and provides the various advantages (see e.g. [OCV$^+$20]) afforded by equivariance.

GLD has been successfully applied in contexts where data transformations can be expressed in group-theoretical terms (see e.g. [BBCV21]). 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$^+$24]). Moreover, as highlighted in [GLD$^+$24], although GDL is effective at describing the constraints that a model should implement, it is not always clear how such constraints can be actually be implemented since solving the equations associated with a constraint might be extremely challenging. Categorical deep learning, introduced by [GLD$^+$24], attempts to solve the aforementioned problems using category theory[3] to generalize the notion of equivariance and to tie specific constraints to specific architectures.

Remark 38. At the moment, to the best of our knowledge, [GLD$^+$24] is the only publicly available paper that discusses the ideas of CDL.

The main insight of CDL is that group actions can be represented as algebras over the group action monads, and that maps that are equivariant with respect to these actions are homomorphisms between these algebras. Hence, GDL can be generalized by taking into consideration (co)algebras over other monads and endofcuntors. CDL generalizes GDL into a theory of (co)algebras over endofunctors and monads, yielding a "theory of all architectures" ([GLD$^+$24]). This profecy has not yet been fulfilled, since only a few architectures have been studied within the framework of CDL at the moment of writing this thesis. However, the results shown in [GLD$^+$24] are very promising and, hopefully, future work will widen the scope of CDL to include derivations for more architectures.

---

[3] 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.

The following proposition and the subsequent example show how exactly CDL subsumes GDL.

Proposition 39. *Let* $(G, e, \cdot)$ *be a group. The endofunctor* $G \times X : \mathbf{Set} \to \mathbf{Set}$ *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 *Fig. 2.1.* □

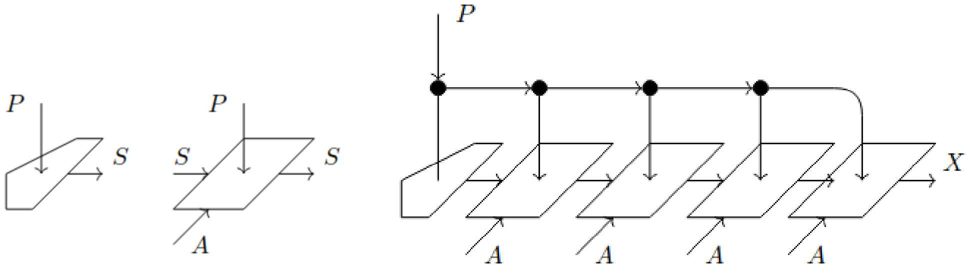
Example 40 (Linear equivariant layer)**.** Consider a carrier set $S = \mathbb{R}^{\mathbb{Z}_2}$, which can be seen as a pair of pixels. Consider the translation action $(i * s)(j) = s(i - j)$ of $G = \mathbb{Z}_2$ on S, which can be seen as swapping the pixels. We want to find a linear map $f : S \to S$ 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 ([GLD$^+$24]).

2.2.2 (Co)inductive definitions for RNNs

As seen in *Ex. 40*, the formalism of CDL subsumes the formalism of GDL, but the difference between the two is not a simple matter of notation: CDL is much more general. The most significant piece of novel contribution delineated in [GLD$^+$24] is the use of (co)algebras and (co)algebra homomorphisms over parametric categories to (co)inductively define recurrent neural networks (GNNs) and recursive neural networks (TreeRNNs). (Co)algebras are used to define cells, whereas the associated homomorphisms provide the weight-sharing mechanics used to unroll them. Let us build on *Ex. 28* and *Ex. 31*, as is done in [GLD$^+$24].

Example 41 (Folding recurrent neural network cell)**.** Consider the endofunctor $1 + A \times X : \mathbf{Set} \to \mathbf{Set}$ from *Ex. 28*. Consider the Cartesian action of **Set** on itself and associate the following actegorical strength to the functor: $\sigma_{P,X}(p, \mathrm{inl}) = \mathrm{inl}$ and $\sigma_{P,A}(p, \mathrm{inr}(x, x')) = \mathrm{inr}((p, x), (p, x'))$. Now that the functor is actegorical strong, we can use *Prop. 32* 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 $\mathsf{Cell}_1 : P \times A \times S \to S$. 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 (*Fig. 2.3 (a)*).

Example 42 (Unrolling of a folding recurrent neural network)**.** Use *Prop. 7* to embed the list algebra $(\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}])$ from *Ex. 28* as an algebra over the endofunctor $\mathbf{Para}(1 + A \times X)$ define in *Def. 41*. 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 homomorphism diagram (*Fig. 2.1 (c)*) lax-commutative. Using the weight-tying reparameterization Δ_P yields the lax commutative diagram in *Fig. 2.4*, 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 *Fig. 2.3 (b)* for a graphical representation).



(a) (b)

Figure 2.3: (a) Two kinds of folding RNN cell represeting the initial state of the network and a generic state update based on input. (b) A folding RNN unrolled using weight tying. (Images taken from [GLD$^+$24].)

$$\begin{tikzcd}
1 + A \times \mathsf{List}(A) \arrow[r, "\mathsf{Para}(1+A\times -)((P, f_{\mathsf{rcnt}}))"] \arrow[d, "\gamma([\mathsf{Nil},\mathsf{Cons}])"'] & 1 + A \times X \arrow[d, "(P, \mathsf{cell}^{\mathsf{rcnt}})"] \arrow[dl, Rightarrow, "\Delta_P"'] \\
\mathsf{List}(A) \arrow[r, "(P, f_{\mathsf{rcnt}})"'] & X
\end{tikzcd}$$

Figure 2.4: Lax-commutative diagram that makes $(P, f) : (\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}]) \to (S, (P, \mathsf{Cell}))$ an algebra homomorphisms. (Image taken from [GLD$^+$24].)

The construction in *Ex. 41* and *Ex. 42* constitutes a precise mathematical link between the classical data structure of lists and the machine learning construct of folding RNNs. Similarly, [GLD$^+$24] recovers recursive neural networks (TreeRNNs) by building upon classical binary trees and, even more interestingly, complete RNNs are recovered from the coalgebra of *Ex. 29*, which reveals an interesting link between RNNs and Mealy machines. 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 RNN where different cells (which share the same weights) are used for state update and output production. Hopefully, more work in this direction will lead to new NN architectures inspired from other classical concepts. *Fig. 2.5* shows various kinds of NN cells and the endofunctors used in their (co)algebraic definitions.

Remark 43 (CDL and optic-based learning)**.** In all the examples discussed above, the (co)algebra homomorphisms in question return parametric maps (P, model), which we can interpret as untrained NN models. We can feed these maps 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. 1.2* can then be used to train these networks. CDL and optic-based learning are thus compatible and even complementary.

---

[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.

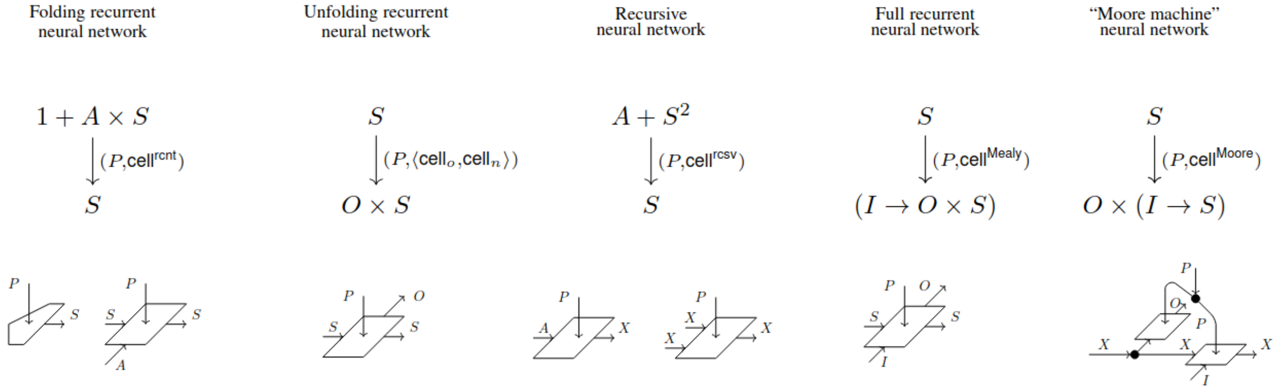

Folding recurrent neural network
Unfolding recurrent neural network
Recursive neural network
Full recurrent neural network
"Moore machine" neural network
$1 + A \times S$
$(P, \mathsf{cell}^{\mathsf{rcnt}})$
S
S
$(P, \langle \mathsf{cell}_o, \mathsf{cell}_n \rangle)$
$O \times S$
$A + S^2$
$(P, \mathsf{cell}^{\mathsf{rcsv}})$
S
S
$(P, \mathsf{cell}^{\mathsf{Mealy}})$
$(I \to O \times S)$
S
$(P, \mathsf{cell}^{\mathsf{Moore}})$
$O \times (I \to S)$
P
S
A
O
X
I

Figure 2.5: Various kinds of NN cells and the endofunctors used in their (co)algebraic definitions. (Image taken from [GLD$^+$24].)

2.3 Algorithmic alignment: GNNs and dynamic programming

One of them main tenets of neural algorithmic reasoning is *algorithmic alignment* ([XLZ$^+$19]), that is, the presence of structural similarities between the subroutines of a particular algorithm and the architecture of the neural network selected to learn such algorithm. Since [XLZ$^+$19] has shown that dynamic programming algorithms align very well with GNNs, and since dynamic programming encompasses a wide variety of techniques algorithms to many different domains, GNNs are at the forefront of neural algorithmic reasoning research ([DV22]). However, the exact link between GNNs and dynamic programming has yet to be fully formalized. In this section we present the work of [DV22], which attempts to derive such a formalization, and the work in [DvGPV24], which studies conditions under which GNNs are invariant with respect to various form of asynchrony[5], which is argued to improve algorithmic alignment in some cases.

2.3.1 Integral transforms for GNNs and dynamic programming

The main link between dynamic programming and GNNs is that dynamic programming itself can be interpreted from a graph-theoretical point of view. Dynamic programming breaks up problems into subproblems recursively until trivial base cases are reached. We can thus consider the graph with nodes corresponding to subproblems and edges (y, x) corresponding to the relationships 'y is a subproblem of x '. Then, the solutions of the subproblems are recusrively recombined to solve the original problem. This dynamic is very similar to message passing: the simpler cases are solved first, and their solutions are passed as messages along the edges so that they can be used to solve more complex cases. The architecture of a graph neural network usually implements such a message passing dynamic. However, rigorously formulating the link between the architecture of GNN and the structure an associated dynamic programming algorithm is not easy, the main obstacle being the difference in data type handled by the two mathematical processes: dynamic programming usually deals with tropical objects such as the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$, while GNNs usually deal with linear algebra over $\mathbb{R}$ ([DV22]).

[DV22] proposes the formalism of integral transforms as the common structure behind both GNNs and dynamic programming. While a full formal proof is not given, the idea is illustrated by showing that both the Bellman-Ford algorithm and the message passing neural network (MPNN) architecture can be expressed with the help of integral transforms. The difference in data type is overcome by using the weakest common hypothesis: that the data and associated operations form a semiring.

---

[5] While much of the work described in [DvGPV24] does not fall under the umbrella of applied category theory, we still mention it because on its close link with the work of [DV22] and with the idea of algorithmic alignment. Hopefully, future work will explore the intersection between this work and category theory.

Bellman-Ford algorithm

The Bellman-Ford (BF) algorithm is one of the most popular dynamic programming algorithms and is used to find the shortest paths between a single starting node and every other node in a weighted graph $G = (V, E)$. Since we can see every node of the graph as a subproblem, and since we can see the associated edges as subproblem relationships, the BF algorithm is a very good candidate for a GNN implementation. The algorithm operates within the tropical min-plus semiring $(R = \mathbb{N} \cup \{\infty\}, \oplus = \min, \otimes = +)$, and the data can be provided as a tuple (d, b, w) of three functions into R. Here, $d : V \to R$ stores the current best distances of the nodes, $b : V \to R$ stores the weights of the nodes, and $w : E \to R$ stores the weights of the edges. d is initialized as the function that maps the initial node to 0 and every other node to ∞. The values of d are updated at each step of the algorithm according to the following formula, where $\mathcal{N}_u = \{v \text{ s. t. } (v, u) \in E\}$ represent the one-hop neighborhood of a node u:

$$d_u \longleftarrow \min\left(d_u, \min_{v \in \mathcal{N}_u}(d_v) + w_{v,u}\right).$$

[DV22] propose the integral transform encoded by the polynomial span in *Fig. 2.6 (a)* as the supporting structure of the BF algorithm. The functions i, p, and o are defined as follows:

1. $i : (V + E) + (V + E) \to V + (V + E)$ acts as the identity on the first V, it maps the edges of the first E to their sources, and it acts as the identity on the second $V + E$ pair;
2. $p : (V + E) + (V + E) \to V + E$ just collapses the two copies of $V + E$;
3. o acts as the target function on the E and as the identity on V.

It is argued in the paper that the whole integral transform acts as step of the algorithm, carrying the data in (d, b, w) to the updated function d. Let's examine each step: the input pullback extracts the distances the sources of every edge; the argument pushforward computes the lengths of the one-hop extensions of the known shortest paths (the weight of each node is treated as the weight of a self-edge in this case); finally, the message pushforward selects the shortest paths to each node among the ones studied by the argument pushforward. Hence, the simple polynomial span in *Fig. 2.6 (a)* successfully encode the whole BP algorithm without any information loss or ad hoc choice.

$$\begin{tikzcd}
(V+E)+(V+E) \arrow[r, "p"] \arrow[d, "i"] & V+E \arrow[d, "o"] \\
V+(V+E) & V
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
E+(E+E)+E \arrow[r, "p"] \arrow[d, "i"] & E \arrow[d, "o"] \\
1+V+E & V
\end{tikzcd}$$

(b)

Figure 2.6: Polynomial spans used to by [DV22] to model (a) the Bellman-Ford algorithm and (b) a message pasing neural network. (Images taken from [DV22].)

Message passing neural network

A message passing neural network (MPNN) is a graph neural network described by the following equations ([GSR$^+$17]):

$$\begin{aligned} m_v^{t+1} &= \sum_{w \in \mathcal{N}_v} M_t(h_v^t, h_w^t, e_{v,w}), \\ h_v^{t+1} &= U_t(h_v^t, m_v^{t+1}), \end{aligned}$$

where t represents the time step, and M_t and U_t are learned differentiable functions.

[DV22] argues that an MPNN layer can be implemented as the integral transform associated with the polynomial span of *Fig. 2.6 (b)*, with an extra MLP. Here,

1. i sends the first E to 1, it acts as source and target on the second E and third E respectively, and it acts as the identity on the fourth E;
2. p collapses four E's into one;
3. o acts as the target function.

In the associated integral transform, i^* gathers graph, features, node features and edge features, $p_\otimes$ "projects" the such features on the edges, the MLP combines them, and, finally, $o_\oplus$ sends them to the right target.

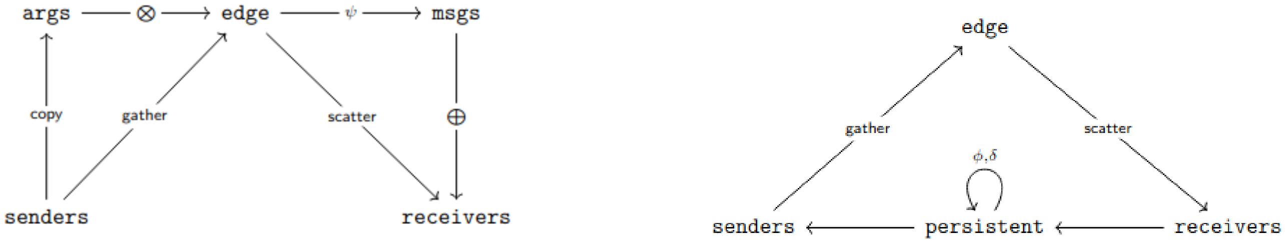
Although not a perfect representation of the target architecture (due to the extra MLP), the polynomial span in *Fig. 2.6 (b)* can be used to inform the design of new architectures which are obtained by simple manipulations of the arrows or objects in the diagram. For instance, [DV22] uses the integral transform formalism to investigate possible performance improvements on CLRS benchmark tasks ([VBB$^+$22]). They consider messages that reduce over intermediate nodes they show these architectures lead to better average performance on these tasks, likely a result of better algorithmic alignment.

2.3.2 Asynchronous algorithmic alignment

[DvGPV24] derives conditions under which synchronous GNNs are invariant under asynchronous execution. This is relevant because, as stated in the paper, in many dynamic programming tasks modeled by graphs, only small parts of the aforementioned graphs are changed at each step. A synchronously executed GNN that is trained on these tasks must learn the identity function many times over, which leads to brittleness and wasted computational resources. However, implementing many algorithms that are invariant under asynchronous execution as synchronous GNNs that are also invariant under asynchronous execution leads to better algorithmic alignment and thus more robust and efficient learning.

The authors of [DvGPV24] revise the model explored in [DV22] so that it includes a message function ψ that generates messages based on gathered arguments (see *Fig. 2.7 (a)* for the update diagram). Moreover, the authors argue that it is best to consider GNNs where every graph component that has a persistent state is elevated to the status of node, whereas transient computations are carried out along edges. The resulting GNN can be described by the diagram in *Fig. 2.7 (b)*, where ϕ is the transit function that updates the persistent state of each node, and δ is the function that computes the arguments needed to generate the next messages.

It is argued in [DvGPV24] that invariance under asynchrony can be modeled by giving both arguments and messages monoidal structures. For instance, let $(M, 1, \cdot)$ be the message monoid and let $(A, 0, +)$ be the argument monoid. Then, if S is the set of persistent states, state update and argument generation can be modeled as a function $M \times S \to S \times A$ which maps $(m, s) \mapsto (m \bullet s, \delta_m(s))$.



(a) (b)

Figure 2.7: Diagrams describing the GNN architecture described in [DvGPV24]. (Images taken from [DvGPV24].)

Invariance under asynchronous message aggregation is obtained by defining $\bullet$ as a monoidal action of M on S. However, [DvGPV24] shows that this is meaningful if and only if the argument generation function δ is compatible with the unitality and associativity equations of the action. This can only happen if $\delta : m \mapsto \delta_m$ is a 1-cocycle.

Definition 44 (1-cocycle)**.** *A map $\delta : M \to [S, A]$ is a 1-cocycle if and only if the following are satisfied:*

1. $\delta_1(s) = 0$ *for all* $s \in S$*;*
2. $\delta_{n \cdot m}(s) = \delta_n(m \cdot s) + \delta_m(s)$ *for all* $s \in S$*.*

Proposition 45. *The state update function δ described above is asynchronous with respect to message passing if and only if it is a 1-cocycle.*

[DvGPV24] also proves the following.

Proposition 46. *Under the hypotheses described above, a single-input message function ψ supports asynchronous invocation if and only if ψ is a homomorphism of monoids.*

We will not describe the whole formalism of [DvGPV24], but we will show (without proof) its implications on GNN architecture design.

Example 47. Consider the GNN architecture:

$$x'_u = \phi\left(x_u, \bigoplus_{v \in \mathcal{N}_u} \psi(x_u, x_v)\right),$$

where $\oplus$ is a message aggregator function. The authors of [DvGPV24] derive conditions under which this architecture is invariant under asynchronies in message aggregation, node update, and argument generation: the GNN is trivially invariant under asynchronous message aggregation if messages $(M, 0, \oplus)$ are given a commutative monoidal structure; invariance under asynchronies in node updates is obtained by selecting an update function ϕ which sastifies the associative law $\phi(s, m \oplus n) = \phi(\phi(s, m), n)$ for all $m, n \in M$ and for all $s \in S$; finally, invariance under argument generation is obtained if ϕ satisfies the 1-cocycle equations (*Def. 44*). [DvGPV24] also gives sufficient conditions that ensure that the 1-cocycle equations as well as the associative law are true: it suffices that $M = A = S$ and that $\phi = \oplus$ is idempotent.

2.4 Future directions and related work

In this section we provide a brief introduction to the theory of differentiable causal computations and the theory of sheaf neural networks. These two lines of work are adjacent to the main theme of this chapter - relating classical computer science to modern machine learning - and they highlight possible directions for future research into categorical deep learning and the application of integral transforms to neural networks.

2.4.1 Differentiable causal computations

[SK19] studies the differential properties of causal computations offering valuable insight into the formal properties of recurrent neural networks (RNNs). The paper focuses on sequences $\{f_k\}_{k=0,1,\dots}$ of functions $f_k : S_k \times X_k \to S_{k+1} \times Y_k$ which represent computations executed in discrete time k, where, at each tick k of the clock, f_k takes an input x_k and the previous state s_k, and uses this data to compute an output y_k and a new state s_{k+1}. In symbols, $f_k(s_k, x_k) = (s_{k+1}, y_k)$. Such sequences are given a nice compositional structure using the formalism of double categories.

Definition 48 (Category of tiles)**.** *Let $\mathcal{C}$ be a Cartesian category. Define $\mathbf{Dbl}(\mathcal{C})$ as the double category with the following data:*

1. *there is only on 0-cell, which we represent with the symbol $\cdot$;*
2. *the horizontal and vertical 1-cells are the objects of $\mathcal{C}$;*
3. *a 2-cell (tile) with horizontal source S, horizontal target S', vertical source X, and vertical target Y is a morphism $f : S \times X \to S' \times Y$ which we represent with the symbol $f : X \xrightarrow[S']{S} Y$.*

It is handy to also represent 2-cells f as the tile string diagrams in Fig. 2.8 (a). The horizontal and vertical composition laws for 2-cells are given consistently with the tile diagrams.

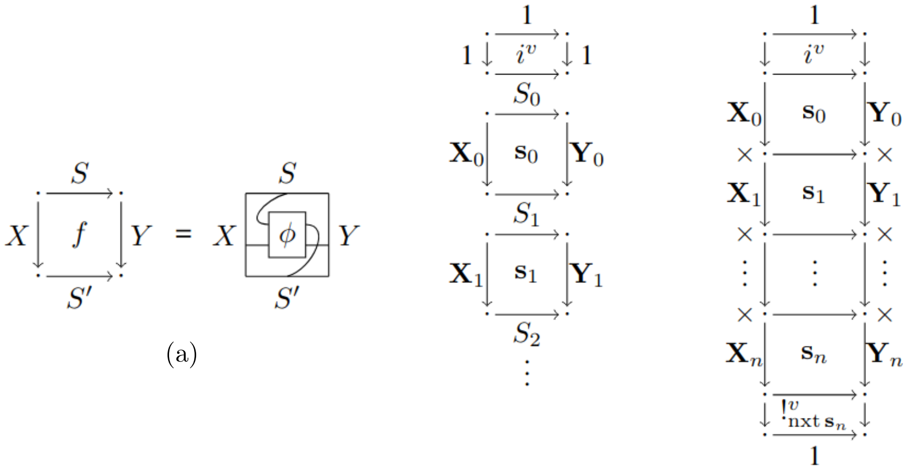
Definition 49 (Category of stateful morphism sequences)**.** *Let $\mathcal{C}$ be a Cartesian category. Define $\mathbf{St}(\mathcal{C})$ as the category with the following data:*

1. *the objects of $\mathbf{St}(\mathcal{C})$ are sequences $\mathbf{X} = \{X_k\}_{k=0,1,\dots}$ of objects of $\mathcal{C}$;*
2. *the morphisms $\mathbf{X} \to \mathbf{Y}$ are pairs $(\mathbf{f}, i)$, where $\mathbf{f} = \{f_k\}_{k=0,1,\dots}$ is a sequence of tiles in $\mathbf{Dbl}(\mathcal{C})$ such that $f_k : X_k \xrightarrow[S_{k+1}]{S_k} Y_k$, for some sequence $\mathbf{S}$ of states, and $i : 1 \to S_0$ selects an initial state.*

The morphisms of $\mathbf{St}(\mathcal{C})$ are known as stateful morphisms sequences and are represented using string diagrams as in Fig. 2.8 (b).

Stateful morphisms sequences can be truncated and unrolled as one would expect, and it is proved in [SK19] that there is a bijection between stateful morphism sequences in $\mathbf{St}(\mathbf{Set})([A], [B])$ and causal functions $A^{\mathbb{N}} \to B^{\mathbb{N}}$ (here $[A]$ is the constant sequence of objects $\{A, A, A, \dots\}$). More generally, given any $\mathbf{St}(\mathcal{C})$, we can restrict our attention to constant sequences $[A]$ and stateful sequences of morphisms in the form $([f], i)$, where $f : X \xrightarrow[S]{S} Y$ is a tile in $\mathbf{Dbl}(\mathcal{C})$. This yields a subcategory $\mathbf{St}_0(\mathcal{C})$ whose morphisms can be thought of as Mealy machines that take in an input and produce an output based on an internal state which is updated after every computation. The new state is in a sense fed back to the machine after the computation so that a new computation can take place. This is represented by the diagram in *Fig. 2.8 (b)*.

The authors of [SK19] go on to define a delayed trace operation which provides a rigorous formalization of feedback loops such as the one in *Fig. 2.9 (a)*. As stated in the paper, the delayed trace operator is closely related to the more popular trace operator ([JSV96]) and shares many of the same properties. Finally, the authors of [SK19] show how both $\mathbf{St}(\mathcal{C})$ and $\mathbf{St}_0(\mathcal{C})$ can be given the structure of a CDC (*Def. 13*), as long as $\mathcal{C}$ is itself a CDC. This differential structure is conceptually clear,



(a)

(b)

Figure 2.8: (a) String diagrams for a tile in $\mathbf{Dbl}(\mathcal{C})$. (b) String diagrams for a stateful morphism in $\mathbf{St}(\mathcal{C})$ and one of its truncation. (Images taken from [SK19].)

rigorously defined, and compatible with the dealyed trace operator. We do not have space to describe the details of these definitions, but we report the relevant string diagrams in *Fig. 2.9 (b),(c).*

The work in [SK19] provides a theoretical foundation for the technique of backpropagation through time (BPTT), which consists in computing the gradient of the k-th unrolling of a RNN in place of the gradient of the such RNN at discrete time k. Despite the alleged ad hoc nature of BPTT, [SK19] proves that the technique does not just ‘involve differentiation ’but is an actual ‘form of differentiation ’that can be reasoned about in the formalism of CDCs. Nevertheless, as stated in the paper, the differential operator of $\mathbf{St}(\mathcal{C})$ does not compute explicit gradients, and deriving the latter would from the former would be computationally intractable when there are millions of parameters.

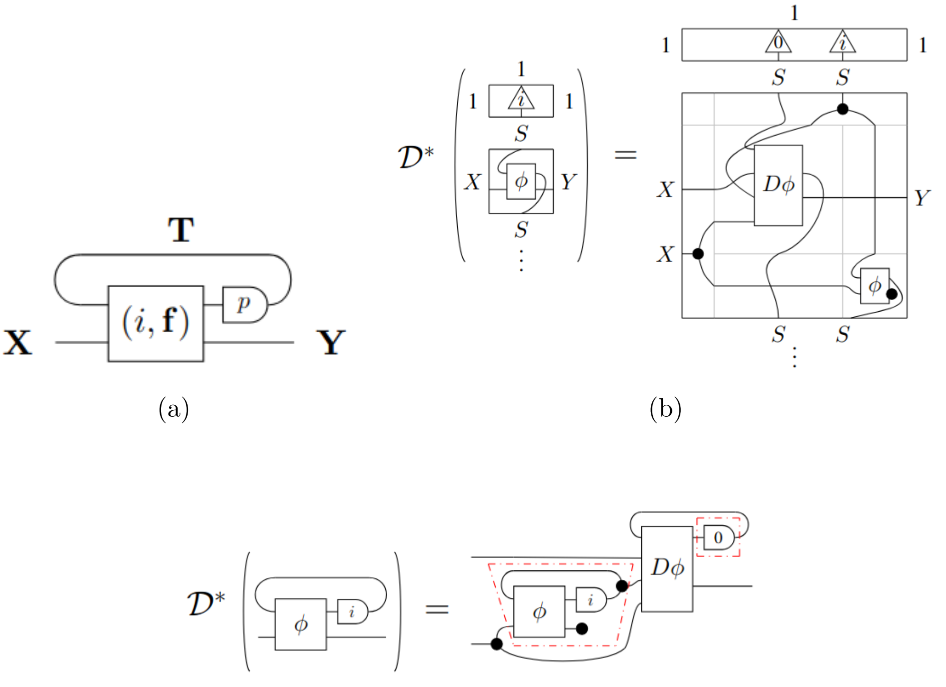
It is interesting to compare the approach of [SK19] with the framework of categorical deep learning: both CDL and the work in [SK19] synthetically describe RNN architectures, but, while CDL focuses on weight sharing mechanics and the coinductive nature of the definition, [SK19] focuses the differential properties of these architectures. However, neither categorical framework deals with the problems that come up when computing gradients of unrolled RNNs, such as vanishing or exploding gradients (see e.g. [Han18]).

2.4.2 Sheaf neural networks

The theory of sheaf neural networks ([HG20], [BDGC$^+$22], [Zag24]), or SNNs, is informed by both topology and category theory and aims to improve the GNN architecture by endowing graphs with cellular sheaf structures. In particular, SNNs are designed to solve two main issues that are encountered when training GNNs: oversmoothing, which is the tendency of deep GNNs to spread information too far in the graph to be able to effectively classify nodes, and poor performance on heterophilic graphs, that is graphs where the nodes features are diverse in structure and attributes.

Definition 50 (Cellular sheaf)**.** *A cellular sheaf $\mathcal{F}$ associated with a graph $G = (E, V)$ consists of the following data:*

1. *a vector space $\mathcal{F}(v)$ for every node $v \in V$;*
2. *a vector space $\mathcal{F}(e)$ for every edge $e \in E$;*



(a) (b)

(c)

Figure 2.9: String diagrams (a) representing a delayed trace in $\mathbf{St}(\mathcal{C})$ as a delayed feedback loop, (b) representing the derivative of a morphism in $\mathbf{St}_0(\mathcal{C})$, (c) representing the derivative of a delayed trace in $\mathbf{St}_0(\mathcal{C})$. (Images taken from [SK19].)

3. *a linear map $\mathcal{F}_{v \trianglelefteq e} : \mathcal{F}(v) \rightarrow \mathcal{F}(e)$ for each incident node-edge pair $v \trianglelefteq e$.*

The vector spaces associated to nodes and edges are known as stalks. The linear maps associated to incident node-edge pairs are known as restriction maps. The direct sum $C^0(G, \mathcal{F})$ of all node stalks is known as space of 0*-cochains, and the direct sum $C^1(G, \mathcal{F})$ of all edge stalks is known as space of* 1*-cochains.*

As stated in [Zag24], the node stalks assigned by $\mathcal{F}$ serve as spaces for node features, while the restriction maps allow the data that resides on adjacent nodes to interact on edge stalks. Given a cellular sheaf $\mathcal{F}$, we can define a coboundary map δ which measures the amount 'disagreement '[6] between nodes. The coboundary map can then be used to define a sheaf Laplacian which can be used to propagate information in the graph.

Definition 51 (Coboundary map)**.** *Let $\mathcal{F}$ be a cellular sheaf on a directed graph $G = (E, V)$. The coboundary map associated with $\mathcal{F}$ is the linear map $\delta : C^0(G, \mathcal{F}) \rightarrow C^1(G, \mathcal{F})$ that maps $\delta(\mathbf{x})_e = \mathcal{F}_{v \trianglelefteq e}(x_v) - \mathcal{F}_{u \trianglelefteq e}(x_u)$ for each edge $e : u \rightarrow v$.*

Definition 52 (Sheaf Laplacian)**.** *Let $\mathcal{F}$ be a cellular sheaf on a directed graph $G = (E, V)$ and let δ be the associated coboundary map. The sheaf Laplaciant associated with $\mathcal{F}$ is the linear map $L_{\mathcal{F}} = \delta^T \circ \delta : C^0(G, \mathcal{F}) \rightarrow C^0(G, \mathcal{F})$. The normalized sheaf Laplacian associated with the sheaf is the linear map $\Delta_{\mathcal{F}} = D^{-\frac{1}{2}} \circ L_{\mathcal{F}} \circ D^{\frac{1}{2}}$, where D is the diagonal of $L_{\mathcal{F}}$.*

---

[6] There is a close link between the theory of SNNs and the theory of opinion dynamics. See [Zag24] for further information.

Remark 53. The coboundary map and the sheaf Laplacian associated with a cellular sheaf $\mathcal{F}$ are generalizations of the more commonly known incidence matrix and Laplacian associated to a graph (see e.g. [WJL$^+$22]).

There are many kinds of SNN architectures ([HG20], [BDGC$^+$22], [Zag24]). Due to space constraints, we only give a short description of the first one to appear in the literature: the Hansen-Gebhart SNN proposed by [HG20].

Definition 54 (Sheaf neural network)**.** *Suppose $G = (E, V)$ is a directed graph and $\mathcal{F}$ is a cellular sheaf on it. Suppose the stalks of $\mathcal{F}$ are all equal to $\mathbb{R}^{f \times d}$, where f is the number of channels and d is the dimension of each feature. Then, $C^0(G, \mathcal{F})$ is isomorphic to $\mathbb{R}^{nd \times f}$, where n is the number of nodes, and its elements can be represented as matrices $\mathbf{X}$. The sheaf neural network proposed by [HG20] uses the following transition function to update this features:*

$$\mathbf{Y} = \sigma((I_{nd} - \Delta_{\mathcal{F}})(I_n \otimes W_1)\mathbf{X}W_2),$$

where σ is a non-linearity, $\otimes$ is the Kronecker product, $W_1 \in \mathbb{R}^{d \times d}$ and $W_2 \in \mathbb{R}^{f \times f}$ are weight matrices.

Remark 55. The values of f_1, f_2, n, d, and the restriction maps are all hyperparameters Choosing $W_2 \in \mathbb{R}^{f_1 \times f_2}$ allows the SNN layer described above to change the number of features from f_1 to f_2.

As observed by [BDGC$^+$22], the SNN architecture proposed by [HG20] can be seen as a discretization of the differential equation

$$\dot{\mathbf{X}}(t) = -\Delta_{\mathcal{F}}\mathbf{X}(t),$$

which is known as sheaf diffusion equation and is analogous to the heat diffusion equation used in graph convolutional networks ([BDGC$^+$22]). Studying the time limit of the sheaf diffusion equation yields important results about the diffusion of information through the graph after repeated application of the transformation in *Def. 54*. In particular, [BDGC$^+$22] argues that, in the time limit, node feature tend to values that ‘agree ’on the edges. Hence, ‘sheaf diffusion can be seen a synchronization process over the graph ’. [BDGC$^+$22] goes on to study the deiscriminative power of different classes of cellular sheaves and proposes strategies to learn the restriction maps themselves. [Zag24] extends the work of [BDGC$^+$22] by analyzing non-linear sheaf Laplacians and the associated sheaf diffusion process.

It is important to notice that SNNs can be considered a strict generalization of GNNs since the latter are nothing but instances of the former where the sheaf structure is trivial ([BDGC$^+$22]). Thus, although we are not aware of any work applying SNNs to neural algorithmic reasoning, given the success enjoyed by GNNs in this area of research, we can only image that SNNs would be even more effective at executing algorithms. To the best of our knowledge, no one has ever explicitly described SNN architectures using integral transforms either. However, a passing remark in [DvGPV24] hints that the message passing dynamic of GNNs is very similar to sheaf diffusion and thus such a generalization should be all but impossible. Hopefully, future research will shed light on these conjectures.

Chapter 3

Functor Learning

All categorical machine learning frameworks examined in the previous chapters represent machine learning models, both trained and untrained, as morphisms in some category. Morphisms capture the core idea of compositionality and are thus a very good choice in many contexts; nevertheless, there are also cases where a simple morphism is unable to capture the structure that one might want preserved. For instance, different datasets might be linked using morphisms in an appropriate category (see e.g. [Spi12], [Gav19]) or sets containing machine learning data could be given a categorical structure, where elements are objects and morphisms capture relations between such objects (see e.g. [Lam99]). In these and other cases, learning functors instead of morphisms is advantageous as it allows us to preserve the aforementioned structure during the learning process. In this chapter we examine various approaches that testify to the usefulness of this insight: we will see how functors can be used to separate different layers of abstraction in the machine learning process ([Gav19]), to embed data in vector spaces ([SY21], [CSC10],[Lew19]), to carry out unsupervised translation ([SY21]), and to impose equivariance constraints and pool data effectively ([CLLS24]). We will illustrate how functors can be learned by gradient descent, just like morphisms, as long as we have appropriate parametrizations ([Gav19]) or specialized objective functions ([SY21], [CLLS24]).

3.1 Using functors to separate layers of abstraction

The author of [Gav19] takes inspiration from the field of categorical data migration ([Spi12]) to create a categorical framework for deep learning that separates the development of a machine learning model into a number of key steps. The different steps concern different levels of abstraction and are linked by functors.

3.1.1 Schemas, architectures, models, and concepts

The first step in the learning pipeline proposed by [Gav19] is to write down the bare-bones structure of the model in question. This can be done by 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. *Fig. 3.1*).

Definition 56 (Model schema)**.** *The schema of a model represented by a multigraph G is the freely generated category* $\mathbf{Free}(G)$*.*

The schema of a model does not contain any data nor does it do any computation, but it encodes the bare-bones structure of such model. If $\sim$ is the path congruence relation induced by the equations in $\mathcal{X}$, we can take the module $\mathbf{Free}(G)/\sim$ to impose the constraints represented by $\mathcal{X}$. Depending on

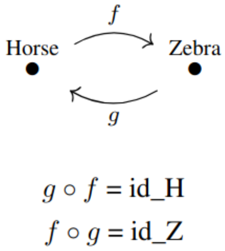


Figure 3.1: Schema for the cycleGAN architecture. (Image taken from [Gav19].)

the context, the word schema will refer either to $\mathbf{Free}(G)$ or to $\mathbf{Free}(G)/\sim$. We will see later that each equation in $\mathcal{X}$ can be associated with a specific loss function and these losses are used to teach models to abide by $\sim$.

Given a schema $\mathbf{Free}(G)$, we can choose an architecture for the model, that is, we can assign to each node a Euclidean space and to each morphism a parametric map. This procedure yields an untrained neural network.

Definition 57 (Model architecture)**.** *Let* $\mathrm{Free}(G)$ *be a model schema. An architecture for such schema is a functor* $\mathsf{Arch} : \mathrm{Free}(G) \to \mathbf{Para}_\times(\mathbf{Smooth})$ *(see Def. 5 and see Ex. 15).*

The Euclidean spaces Arch maps objects to might be intuitively interpreted as the spaces the data will live in, but it is wiser to put 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 58 (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*[1]*.*

Now that we have a model, we must find a way to assign specific values to the parameters, so that the model can be optimized by gradient descent. Consider the function $\mathfrak{p} : (P, f) \mapsto P$, which takes the parameter space out of a parametric map in $\mathbf{Para}_\times(\mathbf{Smooth})$. We can use it to define the 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 the notion of parameter specification function.

Definition 59 (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* $p \in \mathfrak{P}(\mathsf{Arch})$ *- 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 takes a schema and implements it replacing nodes with Euclidean spaces and arrows with appropriate smooth functions. Notice that, although we need to choose an architecture and specific values for its parameter in order to define Model_p, the latter acts on the schema directly and does not pass through the architecture. See *Fig. 3.2* for a srting diagram that depicts the relationship between Arch and Model_p.

---

[1] The reason why the domain E is the discretized schema $|\mathrm{Free}(G)|$ instead of the original schema $\mathrm{Free}(G)$ is clarified in *Rem. 61.*

$$\begin{tikzcd}
\mathbf{Free}(G) \arrow[r, "\mathsf{Arch}"] \arrow[dr, "\mathsf{Model}_p"] \arrow[ddr, "H_p"'] & \mathbf{Para}(\mathbf{Euc}) \\
& \mathbf{Euc} \arrow[d, "U"] \\
& \mathbf{Set}
\end{tikzcd}$$

Figure 3.2: Functors are used to link different layers of abstraction in the framework of [Gav19]. (Image taken from [Gav19].)

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 model embedding functor.

Definition 60 (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 discretized free category $|\mathbf{Free}(G)|$ to a finite subset $D_E(A) \subseteq E(A)$.*

Remark 61. The reason why [Gav19] defines 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 $E(A)$ and a dataset $D_E(A)$. 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, that is 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. It makes thus sense to define another set $\mathfrak{C}(A)$ representing such concept: this set will satisfy the inclusion relations $D_E(A) \subseteq \mathfrak{C}(A) \subseteq E(A)$. Moreover, since concepts are assumed to be complete, we can extend $\mathcal{C}$ to a functor.

Definition 62 (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, $D_E \subseteq I \mathbin{⨾} \mathfrak{C} \subseteq E$.*

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 $\mathfrak{C} : \mathbf{Free}(G)/\sim \to \mathbf{Set}$. More concretely, we want to train a model Model_p which is as close of an approximation of $\mathfrak{C}$ as possible[2]. Total achievement of such goal is clearly impossible as, even in the simplest of cases (such as linear regression on synthetic 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 the ideal goal.

---

[2] Notice that Model_p and $\mathfrak{C}$ have different domains and codomains. The difference of codomains is not an issue: we can just forget the differential structure of the codomain of Model_p, which yields a functor H_p with image in $\mathbf{Set}$ (see *Fig. 3.2*). The difference of domains, on the other hand, is by design as Model_p can only approximate the constraints imposed by $\sim$. This is not an issue in practice as we are only interested in the performance of the image of the trained model Model_p.

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

Definition 63 (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 64 (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 65 (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*

$$\mathcal{L} = \mathcal{L}' + \gamma \sum_{f \sim g} \mathcal{L}_{\sim}^{f,g} \tag{3.1}$$

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 $\mathcal{L}_{\sim}^{f,g}$ does.

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. This point of view offers two main advantages: on one hand, it separates different layers of abstraction are separated providing much needed conceptual clarity; on the other hand, it offers an explicit treatment of constraints, which are often only described implicitly and hidden away in the architecture of the model or in the loss function.

3.1.2 Datasets influence the semantics of tasks

[Gav19] centers its investigation around the powerful cycleGAN architecture ([ZPIE17]), which is described using an appropriate task consisting of the cycleGAN schema (see *Fig. 3.1*), the cycle consistency equations, and a cycleGAN dataset, that is, a dataset with two sets A and B containing data which is essentially isomorphic, such as pictures of horses and zebras. This description is particularly insightful because it can be used to prove a very important point: the choice of dataset influences the semantics of the learned task in non-trivial ways. In other words, changing the dataset functor can result in semantically different tasks even if the same schema and equations are retained.

The example shown in the paper is the [Gav19]: combining the cycleGAN schema with the cycleGAN equations and a cycleGAN dataset yields a task whose semantics can be described as *learn maps that turn horses into zebras and vice versa.* Now replace the cycleGAN dataset with a new dataset consisting of two sets A and B, where A contains pictures depicting two elements X and Y

together, and B contains separate images of X and Y. The resulting task has very different semantics: *learn how to separate X from Y*. For instance, [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.

This example is especially relevant to the present discussion because it shows how important the categorical structure of **Set** can be to machine learning. We can interpret pairs *(face, glasses)* as elements of the Cartesian product of the set of faces and the set of glasses. On the other hand, the set of pictures of faces with glasses can instead be considered another categorical product of the aforementioned sets. Since categorical products are unique, we know that there must be a unique isomorphism between the two products. Then, one interpretation of the task becomes: *find the canonical isomorphism*. [Gav19] labels this task as the product task and shows empirical evidence of the validity of the approach.

3.2 Categorical representation learning

We now discuss frameworks where the categorical structure of data is not only used to integrate data distributed among different sets (as in [Gav19]) but also to model relationships within the data. Preserving and exploiting this structure functorially can lead to marked improvements in effectiveness and efficiency of the models in question because the structure of data is often closely linked with its semantics (see e.g. [SY21]), especially in the field of natural language processing, or NLP (see e.g. [CSC10], [Lew19]).

The field of *categorical representation learning* ([CSC10], [Lew19], [SY21]) aims to learn vectorial representations of data endowed with a categorical structure mirroring the one of the original data, so that models tranined on these representations can use the structure efficiently. This is often a symbolic-to-vectorial transformation, which poses conceptual and computational challenges (see e.g. [CSC10]). We will examine two examples of categorical representation learning: the approach of [SY21], which uses the obtained representations to carry out (partially) unsupervised translation, and the approaches of [CSC10] and [Lew19], which develop a compositional distributional model of meaning for NLP.

3.2.1 Unsupervised functorial translation

We now illustrate the approach of [SY21] showing how it is applied to the same example described in the original paper: unsupervised translation of the names of chemical elements from English to Chinese. The authors are tasked with converting between two identical chemical compound datasets, one labeled in English and the other in Chinese. Both datasets can be given a categorical structure by considering elements as objects and chemical bonds as morphisms. Suppose $\mathcal{C}$ and $\mathcal{D}$ are the resulting categories. The authors of [SY21] leverage the share structure by training a model to learn a translation functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$. To this aim, they then functorially embed these categories into the vector space category defined below, which will contain representations as objects.

Definition 66 (Vector space category)**.** *Let $n \in \mathbb{N}$. Let $\mathcal{R}$ be the category whose objects are the vectors in $\mathbb{R}^n$ and such that, for all $u, v : \mathcal{R}$,*

$$\mathcal{R}(u, v) = \{M \in \mathbb{R}^{n\times n} \text{ s.t. } v = Mu\}.$$

Composition is ordinary matrix multiplication and the identity on v is $\mathrm{id}_v = \frac{vv^T}{|v|^2}$.

The embedding of $\mathcal{C}$ into $\mathcal{R}$ is $\mathcal{C} \to \mathcal{R}$ functor which maps each object a to a vector v_a and each morphism f to a matrix M_f. The actual mapping can be learned with a neural network consisting of two separate embedding layers: one mapping objects to vectors and one mapping relations to matrices. The authors of [SY21] train the embedding layers using co-occurrence statistics and negative sampling to make sure that the embedded morphisms actually represent the same relations as the original morphisms. Here, the authors of [SY21] take on a distributional point of view, positing that co-occurrence must encode such relations. In the authors' words, 'co-occurrence 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 co-occurrence 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 (or more sophisticated forms of aggregation) over all morphisms in $\mathcal{C}$. Taking on a probabilistic perspective allows us to represent non-strict relations, which is often necessary in machine learning. From this point of view, the likelihood of $f : a \to b$ is proportional to the degree of alignment between v_b and $M_f v_a$ are closely aligned. The latter can be computed as $v_a^T M_f v_b$, and the non-linearity F reads out these measurements into a single value. Now, the actual co-occurrence probability $p(a, b)$ of two objects $a, b : \mathcal{C}$ can be approximated directly from the dataset. Given a negative sampling distribution p_N on objects unrelated to a, we can implement the negative sampling objective

$$\mathcal{L} = \mathbb{E}_{(a,b)\sim p(a,b)} \left(\log P(a \to b) + \mathbb{E}_{b'\sim p(b')} \log(1 - P(a \to b'))\right).$$

The embedding network can then be trained by maximizing this objective function. The embedding of $\mathcal{D}$ into $\mathcal{R}$ can be defined and trained in the same way.

Remark 67. Strictly speaking, morphisms in category should have a specific domain and a specific codomain but, in the case of the vector space category of *Def. 66*, a single matrix M can be in many hom-sets of $\mathcal{R}$. This is mirrored by $\mathcal{C}$ and $\mathcal{D}$, where morphisms correspond to classes of chemical bond and the same morphism can link different objects. From this point of view, morphisms in $\mathcal{R}$, in $\mathcal{C}$, and in $\mathcal{D}$ behave more similarly to relations on sets than to morphisms in categories. Although this line of thinking does not strictly adhere to the definition of category, it helps our intuition and greatly reduces the size of the model by limiting the number of parameters that need to be learned. We will see a possible formal solution to this problem later on in this section, when we discuss the approach of [CSC10] and [Lew19].

The procedure described above can be been applied to both $\mathcal{C}$ and $\mathcal{D}$ to obtain meaningful vectorial representations of both datasets. An English to Chinese translation functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ can be similarly embedded as a $\mathcal{R} \to \mathcal{R}$ functor by precomposition with the $\mathcal{C} \to \mathcal{R}$ embedding and postcomposition with the inverse of the $\mathcal{D} \to \mathcal{R}$ embedding. Such functor must equate chemical bonds of the same kind, for example, if f is a covalent bond so is $\mathcal{F}(f)$. It is argued in [SY21] that this functor can be represented by a matrix $V_\mathcal{F}$ so that $v_{\mathcal{F}(a)} = V_\mathcal{F} v_a$ and $M_{\mathcal{F}(f)} = V_\mathcal{F} M_f$. This 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, and if we constrain $V_\mathcal{F}$ to be orthogonal, (ii) and (iii) are trivially satisfied. Requirement (i), on the other hand, can be learned by minimizing the following structure loss:

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

As the authors of [SY21] 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, it might not produce 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 that 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

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

Then, the total loss can be written as a weighted sum

$$\mathcal{L} = \mathcal{L}_{\text{align}} + \lambda \mathcal{L}_{\text{struct}}, \tag{3.2}$$

where λ is a hyperparameter that regulates the relative importance of the two losses.

Remark 68. It is interesting to compare the total loss in *Eq. 3.2* to the one in *Eq. 3.1*. In both cases, the total loss is obtained as a linear combination of a model specific loss and a second more abstract loss. Both abstract losses enforce equality between morphisms but the abstract loss in *Eq. 3.2* does not enforce functoriality because Model_p is already a functor by definition in the framework of [Gav19]. This highlights the main difference between the approach of [Gav19] and the approach of [SY21]: the latter requires functoriality to be learned from structure of data, while the former takes functoriality as a given.

According to [SY21], the categorical approach described above can also be strengthened by endowing the categories $\mathcal{C}$, $\mathcal{D}$, and $\mathcal{R}$ with monomial products that make combining objects into higher-level structures possible. In particular, the monomial structure $\otimes'$ of $\mathcal{R}$ can be given as

$$\begin{aligned} v_a \otimes' v_b &= \Theta(v_a \otimes v_b), \\ M_f \otimes' M_f &= \Theta(M_f \otimes M_g), \end{aligned}$$

where $\otimes$ is the Kronecker tensor product and Θ is a learned operator that sends the products back into the original spaces. This monomial structure can then be used to mine categorical structure on multiple levels: for instance, in the running example of unsupervised translation of chemicals, it can be used to derive vectorial representations for functional groups or even whole compounds. If the data has a non-obvious high-level structure, [SY21] suggests a bootstrap approach where different possible links are randomly tested and the stronger ones are selected.

It is shown in [SY21] that the categorical representation learning framework described in the paper and the associated functor learning architecture for unsupervised translation can be successfully implemented. Benchmark tests against traditional sequence-to-sequence models show that the functorial paradigm leads to marked improvements in efficiency of learning. In particular, the authors compare a GRU cell model of similar performance as the functorial model described in the paper, noting that the former needs 17 times more parameters than the latter to learn to translate element names with a similar accuracy. The authors also compare their approach with the multi-head attention approach first introduced by [VSP$^+$17], arguing that their 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. Keeping M_f united emphasizes the important role of functoriality and provides intuition concerning the nature of the vectorial representations of objects: in fact, it makes sense to interpret each M_f as a metric that distorts the space $\mathbb{R}^n$ and makes two vectors v_a and v_b closer if and only if there is a high likelihood that $a \xrightarrow{f} b$.

3.2.2 Compositional distributional model for NLP

There are two general approaches to natural language processing (NLP): the *compositional* approach and the *contextual* approach. The former aims to exploit the symbolic structure of grammar to understand text, while the latter posits that many aspects of text can be understood from distributional properties of the words ([AACFM22]). While compositionality has been very popular with theoreticians due to its elegance and its abstract structure, contextuality has been successfully applied by practitioners, yielding remarkably effective NLP models that learn the meaning of words and utterances by embedding them in vector spaces. These embeddings are mostly based on contextual information. The fact that both compositionality and contextuality are successful in their domains hints that the two approaches should be seen as complementary and that it would be auspicable to integrate them both in a single *compositional distributional framework*[3] (see e.g. [AACFM22], [Mar19]).

There have been numerous attempts to develop such a framework in the last few decades. For instance, [CP07] proposes to represent both text and the structural roles of its constituents as vectors, which can then be combines using sums and tensor products. We will focus on the approach of [CSC10], whose authors take inspiration from the ideas in [CP07] and propose a framework that uses the language of compact closed categories to unite symbolic grammar rules with concrete representation vectors. Grammar is represented as a pregroup category, whereas vectorial representations are learned on vector spaces which are then organized within **FVect**, the category of finite dimensional real vector spaces. We start by defining pregroup categories[4].

Definition 69 (Pregroup)**.** *A pregroup is a monoidal poset category P where each object a has a left adjoint a^l and a right adjoint a^r. The adjoints must be such that the following are true:*

$$\begin{aligned} a^l a &\leqslant 1 \leqslant a a^l, \\ a a^r &\leqslant 1 \leqslant a^r a, \end{aligned}$$

where 1 *is the monoidal unit of* P*.*

The use of pregroups to model grammatical structures dates back to [Lam99]. The key idea is that we can define a grammar category as the free pregroup P generated by an initial set of basic grammatical roles. Then, we can represent various parts of speech using basic objects, their adjoints, and their monomial products. The poset structure of the pregroup serves as a system of reduction rules which can be used to prove if a sentence is grammatical. Consider the following example (taken from [Lew19]).

Example 70. Suppose n represents the role of a noun and s represents the role of a sentence. Then, an English transitive verb can be represented by the product $n^r s n^l$ in the pregroup P generated by s and n. Thus, the grammatical structure of the sentence “Dragons breathe fire.” is

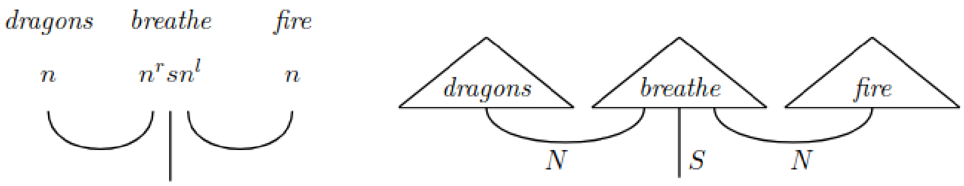
$$n(n^r s n^l)n = n n^r s n^l n = (n n^r) s (n^l n) \leqslant 1 s 1 \leqslant s. \tag{3.3}$$

The pregroup axioms show that the structure of “Dragons breathe fire.” can be reduced to the sentence type s. This is proof that the sentence is grammatical. More complex patter can be modeled by adding more grammatical types or by combining types in other ways. These kinds of computations have a nice representation in the form of string diagrams such as the ones of *Fig. 3.3.*

---

[3] Cognitive science faces a similar challenge in integrating the competing connectionists and symbolic models of the human mind. This is the central theme of [SL06], whose approach inspired [CP07] (discussed in this section). The important role of language in cognition suggests that advances in the development of a compositional distributional model of language may yield advances in cognitive science and vice versa.

[4] We work under the same assumption as [CSC10], namely, that each vector space in **FVect** is endowed with a scalar product $\langle -, - \rangle$.



(a) (b)

Figure 3.3: Two string diagram representations of the grammatical structure of the sentence "Dragons breathe fire." (Images taken from [Lew19].)

The pregroup grammar framework is useful to analyze the basic structure of language but gives no access to the actual meaning of words and sentences. The compositional distributional model of [CSC10] solves this problem by building a functor $\Phi : P \to \mathbf{FVect}$. Given a certain pregroup grammar category P, for every grammatical type $x : \mathcal{P}$, we can associate a vector space $X : \mathbf{FVect}$ containing vectorial representations of utterances of type x. Φ can be defined so that it preserves the pregroup structure of P thanks to the following proposition ([CSC10]).

Proposition 71. *Consider the category* $\mathbf{FVect}$ *of finite-dimensional real vector spaces. Suppose* $\otimes$ *is the tensor product of vector spaces. Then,* $(\mathbf{FVect}, \mathbb{R}, \otimes)$ *is a monoidal category. For all* $V : \mathbf{FVect}$*, define* $V^l = V^r = V$ *and consider the (co)unit linear maps:*

$$\eta^l_V = \eta^r_V : \mathbb{R} \to V \otimes V,\ 1 \mapsto \sum_i e_1 \otimes e_1,$$
$$\epsilon^l_V = \epsilon^r_V : V \otimes V \to \mathbb{R},\ \sum_{i,j} c_{ij} e_i \otimes e_j \mapsto \sum_{i,j} c_{ij} \langle e_i, e_j \rangle,$$

where $e_1, \dots, e_n$ *is a basis for* V*. Then,* $(\mathbf{FVect}, \mathbb{R}, \otimes, (-)^l, (-)^r, \eta^l, \eta^r, \epsilon^l, \epsilon^r)$ *is a compact closed monoidal category.*

Now, pregroups have a trivial compact-closed structure as well. Thus, the functor Φ can be defined so that it preserves the shared compact-closed structure just by mapping the poset relatioins of P to the appropriate linear maps of $\mathbf{FVect}$. In practice, Φ can be used to take a formal reduction in P and implement it in $\mathbf{FVect}$ as a series of operations on vectorial representations. The goal of this process is to take vectorial representations of single words in a sentence and obtain the vectorial representation of the meaning of the sentence. The compositional distributional model is thus compositional because it is build around the compositional structure of grammar and distributional because it allows us to encode distributional information in vectorial representations of words and sentences.

Example 72. The reduction of *Eq. 3.3* can be seen as a morphism $nn^r sn^l n \to s$. The image of this morphism through Φ is a linear map $N \otimes N \otimes S \otimes N \otimes N \to S$.

Remark 73. Differently from [SY21], [CSC10] does not prescribe a specific method for learning the representations of individual words (some possible strategies are listed in the original paper), but specifies a strategy to combine existing representations to generate representations of more complex utterances up to the level of whole sentences. It is interesting to notice that both papers suggest using tensor products in order to combine representations of lower-level structures into representations of higher-level structures.

String diagrams such as the ones in *Fig. 3.3* can be written down as binary trees thanks to the associativity of the monomial product of P and the associativity of the tensor product of **FVect**. [Lew19] proposes to exploit this fact and augment the compositional distributional model of [CSC10] by implementing the (co)unit maps of **FVect** as one or more recursive neural network (TreeRNN) cells. The paper argues that the use of TreeRNN cells should make the model easier to train and more robust. To our knowledge, such proposal has not yet been implemented, although the use of TreeRNNs in NLP has already been applied in practice (see e.g. [ASM19]).

3.3 Equivariant neural networks as functors

Warning 74. We warn the reader that the paper [CLLS24], whose contents are discussed in this section, has been *withdrawn by the authors* citing issues with the references. We decided against omitting the paper from this thesis because we think its research direction is intriguing and potentially highly valuable. Hopefully, future research will soon clarify the epistemological status of the ideas hereby discussed.

[CLLS24] presents a categorical framework that describes neural network layers as functors and uses their functorial nature to impose invariance and equivariance constraints. In particular, equivariance constraints are used to effectively transport the unary operators from the dataset into a latent embedding layer. It is shown that this enables the creation of models that account for shifts and imbalance in covariates when being trained on pooled medical image datasets. In this section, we describe the framework of [CLLS24] and we draw a short comparison with the work of [PAS24], which also deals with the transport of algebraic structure into embedding layers.

3.3.1 An equivariant classifier to diagnose Alzheimer's disease

The authors of [CLLS24] consider 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[5]: $s_2.\text{age} = s_1.\text{age} + x$. 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.

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 $g : s_1 \to s_2$,

$$F(g) : F(s_1) \to F(s_2). \tag{3.4}$$

Invariance with respect to g is not much harder to define: it suffices to impose $F(s_1) = F(s_2)$ and $F(g) = \mathrm{id}_{F(s_1)}$.

Remark 75. *Eq. 3.4* is a categorical generalization of the more usual group-theoretical notion of invariance, as defined in *Def. 37*. To be precise, *Def. 37* is equivalent to *Eq. 3.4* if $\mathcal{S}$, $\mathcal{T}$ are Borel spaces and g, $F(g)$ are group actions, as highlighted in [CLLS24].

---

[5] Notice that this kind of operator clearly possesses an algebraic structure as $f_x \mathbin{⨟} f_y = f_{x+y}$.

The authors of [CLLS24] solve the Alzheimer's disease diagnosis task using a classifier that actively exploits the equivariant nature of functors. The proposed architecture consists of two modules: an invertible functor $F : \mathcal{S} \to \mathcal{T}$ that embeds the data into a latent space, and a functor $C : \mathcal{T} \to \mathbf{Free}(\mathbb{N})$ that does the actual classification (see *Fig. 3.4* for a diagrammatic representation). Here, $\mathcal{T}$ acts as a latent space and the equivariance of F forces the representations in $\mathcal{T}$ to be robust with respect to covariate shifts and imbalances, so that the actual classification operated by $\mathcal{C}$ can be more effective at diagnosing the disease. The whole model can be compactly represented as $F \mathbin{⨟} C : \mathcal{S} \to \mathbf{Free}(\mathbb{N})$. Although the inverse of F does not appear in the formula, its existence is needed to ensure that the latent space accurately represents the data.

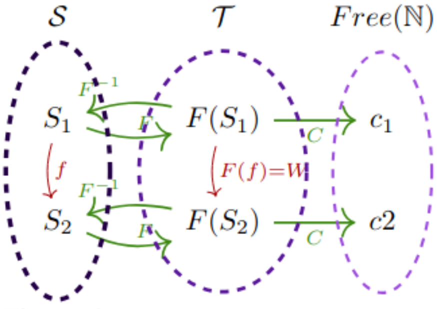


Figure 3.4: Classifier model proposed by [CLLS24] (Image taken from [CLLS24].)

The classifier described above is implemented using similar tools as the ones employed by [SY21]. The latent space $\mathcal{T}$ can be defined as a vector space category as in *Def. 66*. The functor F is realized as the encoder of an appropriate autoencoder, and the decoder portion of this autoencoder acts as the inverse F^{-1}. The maps $f : \mathcal{S}(s_1, s_2)$ can be embedded as matrices as described in the previous section. In particular, [CLLS24] suggests using orthogonal matrices $F(f) = W_f$ so that the resulting transformations can be efficiently inverted. Finally, C can be implemented as a MLP classifier.

Now that the architecture has been specifies in terms of neural networks, the resulting model can be trained using gradient descent and 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 hyperparameters. 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 $F \mathbin{⨟} C$ 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 formulas,

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

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

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

Remark 76. Notice that the amount of equivariance constraints imposed can vary with no substantial changes to the proposed architecture: it suffices to add or remove terms from $\mathcal{L}_s$. This is in stark contrast with approaches such as [LCRS22], where handling multiple covariates requires a complicate multi-stage model.

The authors of [CLLS24] test 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 scaling successfully, even though the network was only trained to apply them separately (see *Fig. 3.5*) for an example. The ADNI classifier model also shows 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.

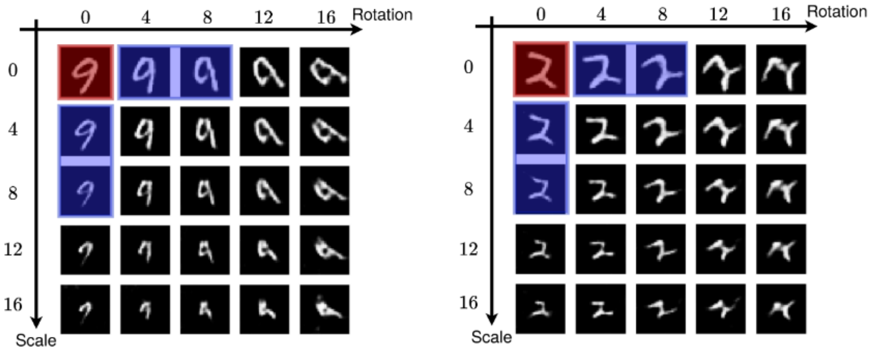


Figure 3.5: The authors of [CLLS24] applied rotations and scaling on MNIST images using the operators learned in the latent space. Despite being learned separately, the two transformations were successfully combined. (Image taken from [CLLS24].)

3.3.2 Transporting algebraic structure into embedding spaces

Although stated in terms of equivariance conditions, the work of [CLLS24] can be seen from an algebraic point of view as a framework that allows to transport unary operators from a space of data to an embedding space. This line of thinking has also been explored by [PAS24]. While [PAS24] deos not use category theory, the authors themselves state that their approach could be easily categorified using the language of categorical algebras (see *Def. 25*). The main innovation of [PAS24] is the use of *mirrored algebras* to fill the gaps between original data and the embeddings.

Consider a pre-trained autoencoder consisting of an encoder $E : \mathcal{S} \to \mathcal{L}$ and a decoder $D : \mathcal{L} \to \mathcal{S}$, where $\mathcal{S}$ is a data space endowed with an algebraic structure, such as a monomial product $\bullet^{\mathcal{S}}$, and where $\mathcal{L} = \mathbb{R}^n$ is a latent space. Suppose we want to transport this algebraic structure to $\mathbb{R}^n$ through E, that is, we want to find a monomial product $\bullet^{\mathcal{L}}$ over $\mathcal{L}$ so that E acts as a $(\mathcal{S}, \bullet^{\mathcal{S}}) \to (\mathcal{L}, \bullet^{\mathcal{L}})$ homomorphism. While finding a product $*$ that makes $(\mathcal{S}, \bullet^{\mathcal{S}})$ and $(\mathcal{L}, \bullet^{\mathcal{L}})$ homomorphic might be easy, there is no guarantee that E is an actual homomorphism with respect to this choice of algebraic structure. [CLLS24] solves this problem for unary operators by learning appropriate orthogonal matrices, but it is not clear how this approach could be scaled to operators with arities greater than 1. [PAS24] suggests an alternative approach consisting in learning a bijection ϕ between the latent space and a copy $\mathcal{M} = \mathbb{R}^n$ of this space. Endow $\mathcal{M} = \mathbb{R}^n$ with a product $\bullet^{\mathcal{M}}$ which enjoyes similar properties as $\bullet^{\mathcal{S}}$ (see *Fig.* **??**). We will say that $(\mathcal{M}, \bullet^{\mathcal{M}})$ is the mirrored algebra of $(\mathcal{S}, \bullet^{\mathcal{S}})$.

Now train ϕ so that E is $(\mathcal{S}, \bullet^{\mathcal{S}}) \to (\mathcal{L}, \bullet^{\mathcal{L}})$ homomorphism, where $\bullet^{\mathcal{L}}$ is defined by

$$l_1 \bullet^{\mathcal{L}} l_2 = \phi^{-1}(\phi(l_1) \bullet^{\mathcal{M}} \phi(l_2)).$$

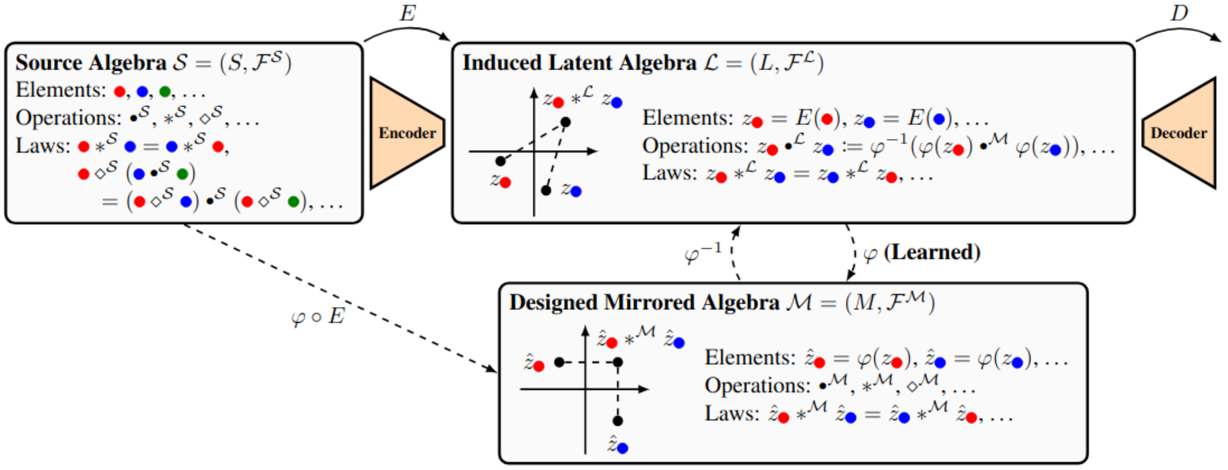

E
D
Source Algebra $\mathcal{S} = (S, \mathcal{F}^{\mathcal{S}})$
Elements: ●, ●, ●, …
Operations: $\bullet^{\mathcal{S}}, *^{\mathcal{S}}, \diamond^{\mathcal{S}}, \dots$
Laws: ● $*^{\mathcal{S}}$ ● = ● $*^{\mathcal{S}}$ ●,
● $\diamond^{\mathcal{S}}$ (● $\bullet^{\mathcal{S}}$ ●)
= (● $\diamond^{\mathcal{S}}$ ●) $\bullet^{\mathcal{S}}$ (● $\diamond^{\mathcal{S}}$ ●), …
Encoder
Induced Latent Algebra $\mathcal{L} = (L, \mathcal{F}^{\mathcal{L}})$
$z_●$ $*^{\mathcal{L}}$ $z_●$
$z_●$
$z_●$
Elements: $z_●$ = E(●), $z_●$ = E(●), …
Operations: $z_● \bullet^{\mathcal{L}} z_● := \varphi^{-1}(\varphi(z_●) \bullet^{\mathcal{M}} \varphi(z_●))$, …
Laws: $z_● *^{\mathcal{L}} z_● = z_● *^{\mathcal{L}} z_●$, …
Decoder
φ^{-1}
φ (Learned)
$\varphi \circ E$
Designed Mirrored Algebra $\mathcal{M} = (M, \mathcal{F}^{\mathcal{M}})$
$\hat{z}_● *^{\mathcal{M}} \hat{z}_●$
$\hat{z}_●$
$\hat{z}_●$
Elements: $\hat{z}_● = \varphi(z_●)$, $\hat{z}_● = \varphi(z_●)$, …
Operations: $\bullet^{\mathcal{M}}, *^{\mathcal{M}}, \diamond^{\mathcal{M}}, \dots$
Laws: $\hat{z}_● *^{\mathcal{M}} \hat{z}_● = \hat{z}_● *^{\mathcal{M}} \hat{z}_●$, …

Figure 3.6: Method proposed by [PAS24] to transport algebraic structure. (Image taken from [PAS24].)

As stated in [PAS24], the bijection ϕ can be implemented as an invertible neural network and can be trained using a loss in the form

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} d(D(p_i^{\mathcal{L}}(E(s_1), \dots, E(s_{n_i}))), p_i^{\mathcal{S}}(s_1, \dots, s_{n_i}))$$

where the $p_i^{\mathcal{S}}$ are algebraic expressions sampled from the set of expressions over $(\mathcal{S}, \bullet^{\mathcal{S}})$, the s_j are sampled from $\mathcal{S}$, $p_i^{\mathcal{L}}(t_1, \dots, t_{n_i})$ is defined as $\phi^{-1}(p_i^{\mathcal{M}}(\phi(t_1), \dots, \phi(t_{n_i})))$, $p_i^{\mathcal{M}}$ is the translation of $p_i^{\mathcal{S}}$ in terms of the algebraic structure of $(\mathcal{M}, \bullet^{\mathcal{M}})$, and d is an appropriate distance function.

We only presented a sketch of the work in [PAS24]: the original paper presents the framework in terms of any general algebraic structure, deals with laws that regulate the interaction of different operations, proves theoretical guarantees, and explores limitations of the proposed approach. It is shown in the paper that it is not always possible to find a mirrored algebra whose operations satisfy the same laws of the source algebra, which means that it is not always possible to transport this algebraic structure in its entirely. Nevertheless, the authors carry out experiments that prove that there are tangible advantages even in transporting a subset of the algebraic laws. The experiments support the following conjecture: “learned latent space operations will achieve higher performance if they are constructed to satisfy the laws of the underlying source algebra” ([PAS24]).

Chapter 4

Diagrammatic Representations for Neural Networks

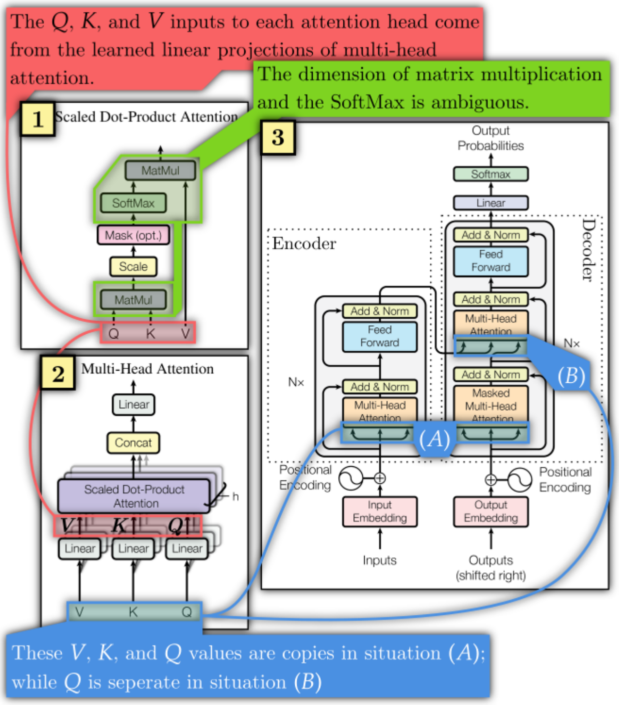

The Q, K, and V inputs to each attention head come from the learned linear projections of multi-head attention.
The dimension of matrix multiplication and the SoftMax is ambiguous.
1
Scaled Dot-Product Attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q K V
2
Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
V K Q
Linear
Linear
Linear
V K Q
3
Output Probabilities
Softmax
Linear
Encoder
Decoder
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Nx
(A)
(B)
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
These V, K, and Q values are copies in situation (A); while Q is seperate in situation (B)

Figure 4.1: Schematics for a transformer as they were presented in the original paper [VSP$^+$17]. The annotation are from the author of [Abb24]. (Image taken from [Abb23].)

4.1 Neural circuit diagrams

[Abb23] proposes to compensate for the shortcomings of monoidal string diagrams by introducing a novel kind of string diagrams: neural circuits diagrams. neural circuits diagrams are a specialization of a broader class of powerful string diagrams known as functor string diagrams, developed in order to represent objects, morphisms, functors, natural transformations, and products in a single diagram.

In this section, we provide a brief description of both classes of diagrams, and we show how neural circuits diagrams can be used to describe the sophisticated transformer architecture.

4.1.1 Functor string diagrams

Despite their unquestionable usefulness, monoidal string diagrams have are fundamentally limited by their inability to represent functors and natural transformations. [Mar14] and [Nak23] solve this relying on colors and symbols, but this only shifts the problem as their diagrams are unable to effectively represent products ([AZ24]). Functor string diagrams follow a different approach: they represent both objects and functors as wires and both morphisms and natural transformations as boxes[1]. These diagrams as built around two fundamental principles, which are already observed in traditional string diagrams. Following these principles ensures that the diagrammatic syntax stays consistent and coherent with the underlying categorical structure as notation becomes more complicated and more features as added.

Principle 77 (Vertical section decomposition)**.** *A string diagram can be divided into vertical columns. A single vertical columns must either contain only objects or only morphisms. Object columns and morphism columns must alternate each other.*

Principle 78 (Equivalent expression)**.** *Every newly introduced piece of graphical notation must be expressible using the notation already present in a compatible manner.*

In accordance with the above principles, [Abb23] introduces notation that represents a functors as wires that lie above objects wires and natural transformations as boxes that lie above morphism boxes (see *Fig. 4.2 (a), (b))*. In standard string diagrams, a wire running through a morphism column is interpreted as an identity morphism. Similarly, in the notation of [Abb23], a wire running where a natural transformation is supposed to be is interpreted as an identical natural transformation. This diagrammatic notation is already effective enough to write a graphical proof of the Yoneda lemma, and can be easily adapted to represent product bifunctors: it suffices to stack two rows, one per category envolved in the product, and seprate them using a double dashed line (as in *Fig. 4.2 (c))*. Finally, monomial products can be represented by using a single dashed separation line or no line at all (as in *Fig. 4.2 (d))*. These piece of diagrammatical notation is also in perfect compliance with the vertical section decompositionl principle and the equivalent expression principle.

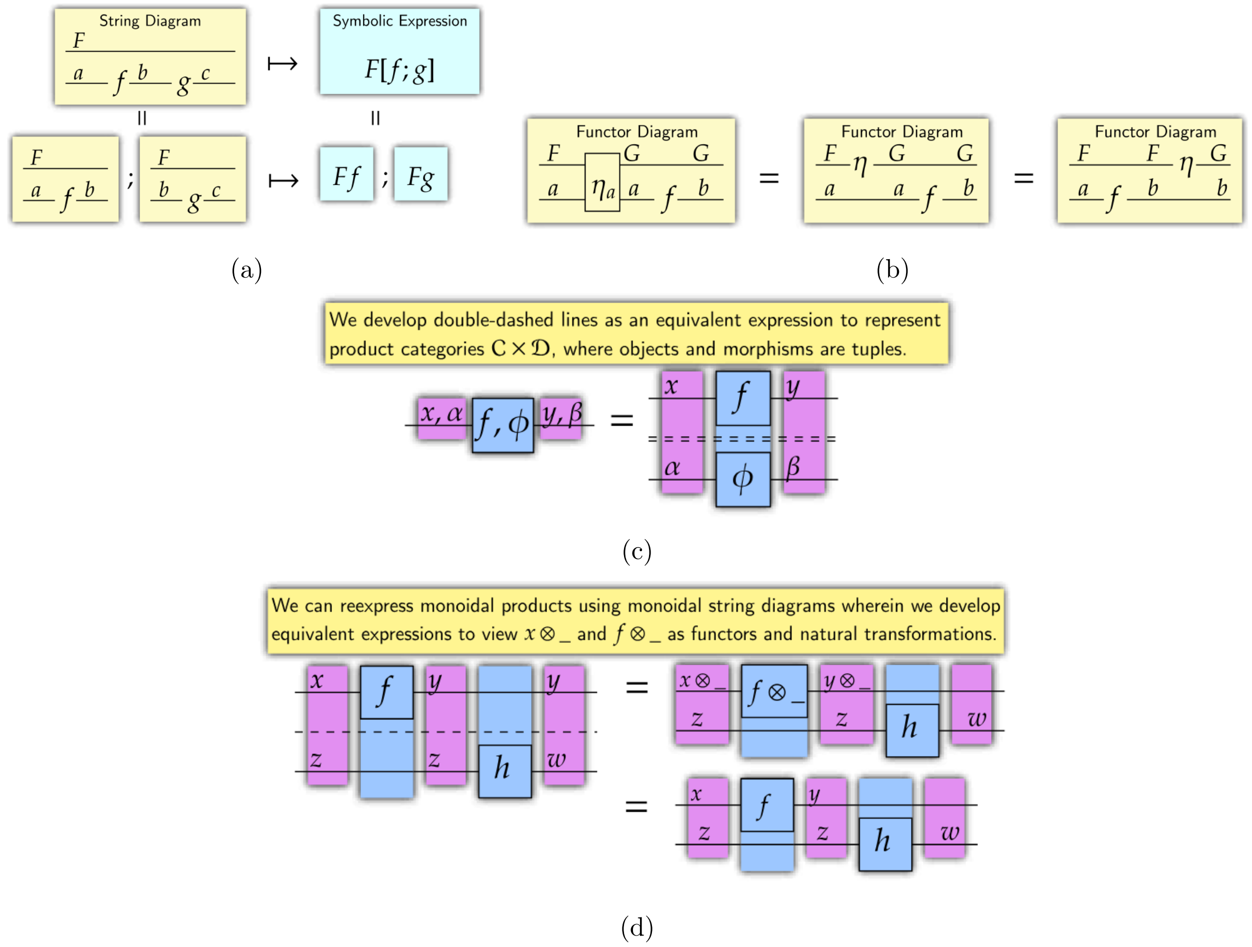
4.1.2 Neural circuit diagrams

Neural circuit diagrams are a specialization of functor string diagrams designed to represent precise schematics for deep neural networks architectures. The idea is to capture all information relevant to implement and analyze the described architecture without using specific programming frameworks. In this sense, neural circuits diagrams can be compared to very detailed pseudocode. Although neural circuit diagrams can be learned and applied without any reference to category theory (this is the perspective adopted in [Abb24]), the fact that these diagrams are just an instance of the wider class of functor string diagrams is not just a curiosity, but proof that they are categorically sound. In other words, every well-formed neural circuit diagrams corresponds to a neural networks and neural circuit diagrams can be composed intuitively.

Differently from the string diagrams seen in e.g. [Gav24] or [WZ22],neural circuit diagrams explicitly keep track of dimensionality and indexing. This allows them to represent complex architectures that involve convolutions, residuality, and so on. Crucially, neural string diagrams provide convenient

---

[1] In simple string diagrams, the actual box is often not drawn for the sake of simplicity. In these cases, the box is replaced by a single letter, but we still use the word "box" in lack of a better word.



(a) (b)

We develop double-dashed lines as an equivalent expression to represent product categories $\mathcal{C} \times \mathcal{D}$, where objects and morphisms are tuples.

(c)

We can reexpress monoidal products using monoidal string diagrams wherein we develop equivalent expressions to view $x \otimes \_$ and $f \otimes \_$ as functors and natural transformations.

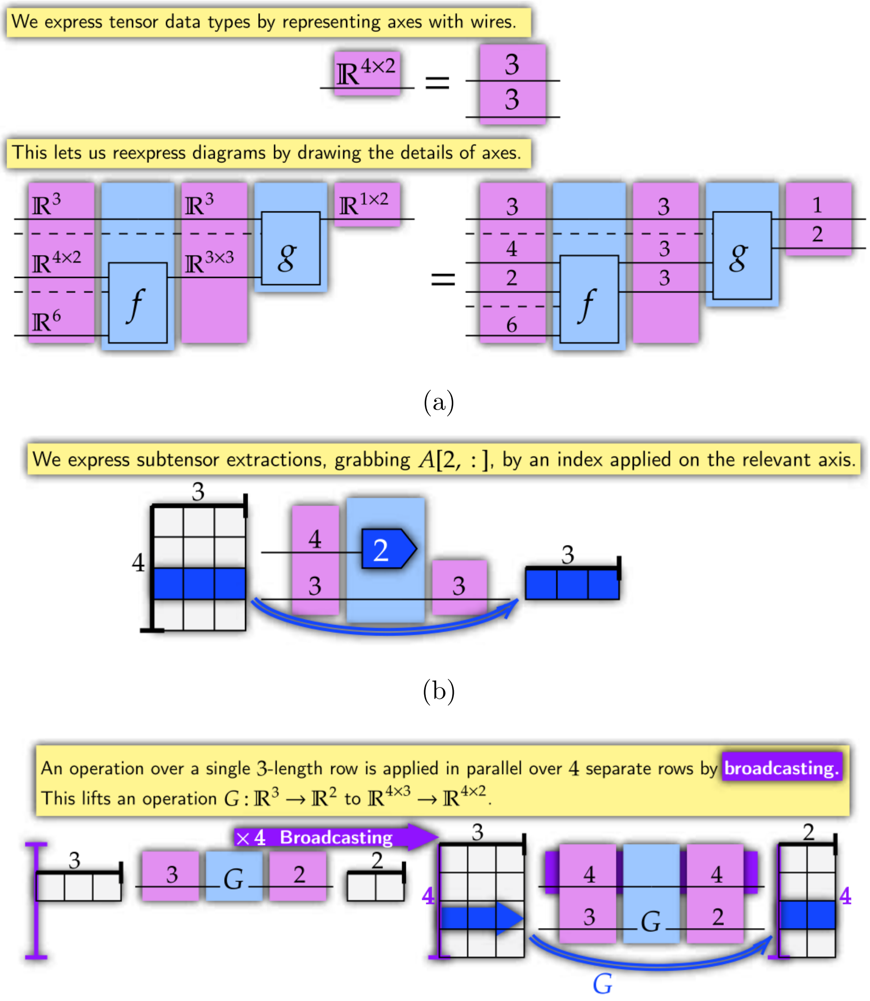
(d)

Figure 4.2: Functor string diagrams associated with (a) a functor, (b) a natural transformation, (c) a product bifunctor, (d) a monoidal product. (Images and annotations taken from [AZ24].)

notation to represent broadcasting, which is of fundamental importance in deep learning but rarely adequately represented by ad-hoc diagrams often seen in papers.

Neural circuit diagrams represent axes of tensors as parallel wires and decorate each wire with the associated number of dimensions (see *Fig. 4.3 (d)*). Indexing is implemented by assigning a specific value to the wire representing the axis with respect to which we want to index (see *Fig. 4.3 (b)*). Finally, broadcasting is represented by running a wire that represents the additional axis over the morphism that is to be broadcasted (see *Fig. 4.3 (c)*). Mathematically, tensors are constructed as hom-functors over the real line and broadcasting is just the action of such a functor on morphisms (*Fig. 4.4*). This is all representable using functor string diagram: neural circuit diagrams are just the result of omitting unnecessary detail from the latter. See [Abb24] for more information regarding neural circuit diagrams and see [Abb23] for more information regarding the relationship between neural circuit diagrams and functor string diagrams.

[Abb24] proves the usefulness of neural circuit diagrams by drawing schematics (*Fig. 4.5*) for the powerful transformer architecture proposed by [VSP$^+$17]. Comparing these schematics with the ones in *Fig. 4.1* presented in the original paper shows the wealth of detail that the former easily encode but are completely absent from the latter. For instance, *Fig. 4.1* provides no information regarding the dimensions of Q, K, and V, despite the fact that these matrices play a central role in the architecture.


We express tensor data types by representing axes with wires.

This lets us reexpress diagrams by drawing the details of axes.

We express subtensor extractions, grabbing $A[2, :]$, by an index applied on the relevant axis.

An operation over a single 3-length row is applied in parallel over 4 separate rows by **broadcasting.** This lifts an operation $G : \mathbb{R}^3 \to \mathbb{R}^2$ to $\mathbb{R}^{4\times3} \to \mathbb{R}^{4\times2}$.

(a)

(b)

(c)

Figure 4.3: Neural string diagram formalisms for (a) representation of tensors, (b) indexing of tensors, (c) broadcasting. (Images and annotations taken from [Abb24].)

Neural circuit diagrams for additional architectures can be found in [Abb24].

4.2 Diagrams with universal approximators

[KLLWM24] also develops a string-diagrammatic language capable of giving detailed unambiguous descriptions of neural network architectures. It is only natural to compare the approach of [KLLWM24] with the neural circuit diagrams of [Abb23] discussed in the previous section. Despite their unquestionable similarity, the two approaches differ on one key point: while neural circuit diagrams leave no space for abstraction and only allow equational rewrites, [KLLWM24] develops a semantics based on the notion of universal approximator that allows to abstract away details and compare different architectures based on share abstractions. In this section, we give a brief overview of the approach of [KLLWM24] and we discuss interesting experimental results presented by the paper. We do not discuss the precise categorical semantics of the diagrams introduced [KLLWM24] because that would

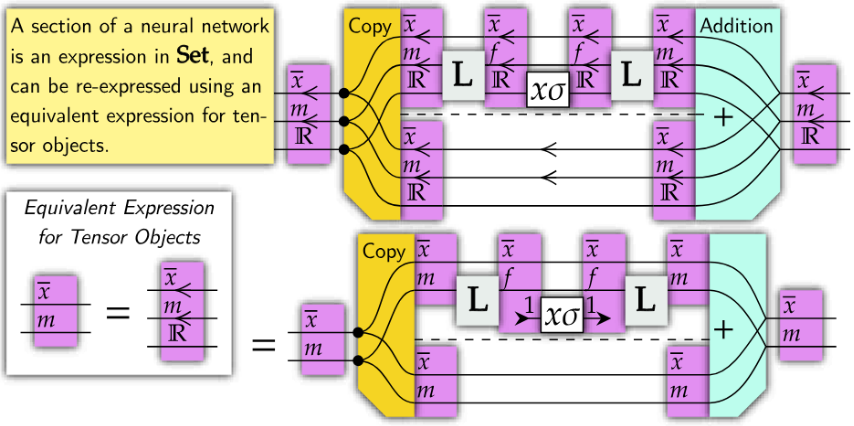

A section of a neural network is an expression in Set, and can be re-expressed using an equivalent expression for tensor objects.
Equivalent Expression for Tensor Objects
Copy
Addition
L
xσ
+
=

Figure 4.4: From functor string diagrams to neural circuit diagrams. (Image taken from [AZ24].)

require too much space, but the reader can find a full account of them in the appendices of the original paper.

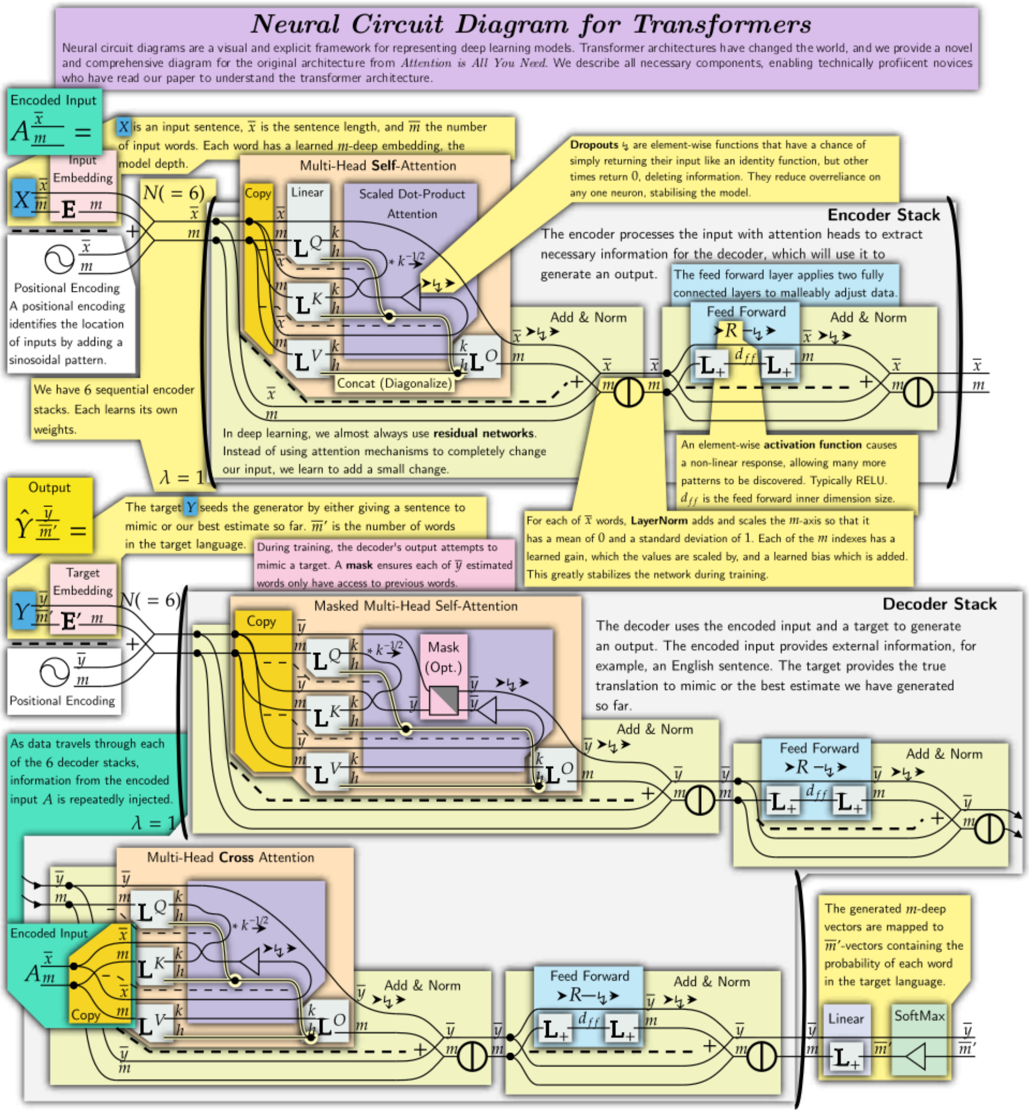
4.2.1 The diagrammatic approach of [KLLWM24]

Neural circuit diagrams effectively describe neural network architectures, but the aboundance of detail they convey also means that they can make it challenging to extract the foundamental structure of an architecture from its complete schematics. [KLLWM24] attacks this problem by drawing different diagrams for different levels of abstraction so that complete schematics like the ones encoded in neural circuit diagrams become only the end of a spectrum that begins with a purely abstract black box. The same fundamental grammar underpins all these diagrams, and appropriate rewrites can be used to seamlessly move between layers of abstraction.

Let us being by discussing the most concrete end of the aforementioned specturm. [KLLWM24] describes computation in the Cartesian category **CartSp** of Euclidean spaces and continuous functions, and follows the basic conventions of monoidal string diagrams for Cartesian categories. A sketch of the language can be found in *Fig. 4.6*. Unlike neural circuit diagrams, the diagrams of [KLLWM24] represent broadcasting using *SIMD (Same Instruction applied to Multiple Data) boxes*, as in *Fig. 4.6 (b)*, which are also used to represent tensor contractions, as in *Fig. 4.6 (c)*. Finally, syntactic sugar is offered to represent reshaping operations (see *Fig. 4.6 (c)*).

The reader can compare the upper portion of *Fig. 4.5* against *Fig. 4.8* to see how both neural circuit diagrams and the approach of [KLLWM24] can be used to describe the same transformer architecture in detail. While both formalisms shine in accuracy of representation, they also both suffer from low readibility: one might ask whether adopting such systems is even worth the trouble given that simpler diagrams such as the one in *Fig. 4.1* seem to be better at communicating basic ideas despite their lack of detail. While [Abb23] compensates for this by annotating its diagrams and by encapsulating portions of the architectures at hand into dedicated boxes, [KLLWM24] proposes to abstract away detail by replacing portions of the graph representing specific implementations with filled boxes that represent universal approximators with the same inputs and outputs.

Definition 79 (Universal approximators)**.** *A universal approximator for a class $\mathcal{C}$ a functions is a neural network architecture able to approximate any element of $\mathcal{C}$ to any desired precision.*



Neural Circuit Diagram for Transformers

Neural circuit diagrams are a visual and explicit framework for representing deep learning models. Transformer architectures have changed the world, and we provide a novel and comprehensive diagram for the original architecture from *Attention is All You Need*. We describe all necessary components, enabling technically proficent novices who have read our paper to understand the transformer architecture.

X is an input sentence, $\overline{x}$ is the sentence length, and $\overline{m}$ the number of input words. Each word has a learned m-deep embedding, the model depth.

Dropouts ↯ are element-wise functions that have a chance of simply returning their input like an identity function, but other times return 0, deleting information. They reduce overreliance on any one neuron, stabilising the model.

Encoder Stack

The encoder processes the input with attention heads to extract necessary information for the decoder, which will use it to generate an output.

The feed forward layer applies two fully connected layers to malleably adjust data.

Positional Encoding

A positional encoding identifies the location of inputs by adding a sinosoidal pattern.

We have 6 sequential encoder stacks. Each learns its own weights.

In deep learning, we almost always use **residual networks**. Instead of using attention mechanisms to completely change our input, we learn to add a small change.

An element-wise **activation function** causes a non-linear response, allowing many more patterns to be discovered. Typically RELU. d_{ff} is the feed forward inner dimension size.

The target Y seeds the generator by either giving a sentence to mimic or our best estimate so far. $\overline{m}'$ is the number of words in the target language.

During training, the decoder's output attempts to mimic a target. A **mask** ensures each of $\overline{y}$ estimated words only have access to previous words.

For each of $\overline{x}$ words, **LayerNorm** adds and scales the m-axis so that it has a mean of 0 and a standard deviation of 1. Each of the m indexes has a learned gain, which the values are scaled by, and a learned bias which is added. This greatly stabilizes the network during training.

Decoder Stack

The decoder uses the encoded input and a target to generate an output. The encoded input provides external information, for example, an English sentence. The target provides the true translation to mimic or the best estimate we have generated so far.

As data travels through each of the 6 decoder stacks, information from the encoded input A is repeatedly injected.

The generated m-deep vectors are mapped to $\overline{m}'$-vectors containing the probability of each word in the target language.

Encoded Input
A x̄ m =
Input Embedding
X x̄ m̄ E m
N(= 6)
Multi-Head Self-Attention
Copy
Linear
Scaled Dot-Product Attention
L Q
L K
L V
L O
k h
* k^-1/2
Concat (Diagonalize)
Add & Norm
Feed Forward
R
L+ d_ff L+
λ = 1
Output
Ŷ ȳ m̄′ =
Target Embedding
Y ȳ m̄′ E′ m
Positional Encoding
N(= 6)
Masked Multi-Head Self-Attention
Copy
Mask (Opt.)
Add & Norm
Feed Forward
Add & Norm
λ = 1
Multi-Head Cross Attention
Encoded Input
A x̄ m
Copy
Add & Norm
Feed Forward
Add & Norm
Linear
SoftMax
ȳ m̄′

Figure 4.5: Neural string diagram of a transformer. (Image taken from [Abb23].)

Many well known results concern the existence and characterization of such approximators (see e.g. [?]). [KLLWM24] uses universal approximators to rigorously represent “typed holes” in a diagram that need to be filled with an appropriate operator. Replacing detailed structures with abstract unversal approximators allows us to climb the ladder of abstraction. This can be combined with the omission of dimensional information that is not necessary to understrand the architecture at hand, but only to implement it. [KLLWM24] showcases this process in the diagram of *Fig. 4.9*, which highlights the fundamental structure of the attention mechanism proposed by [VSP^{+}17]. Notice how colors are used to distringuish between dimensions that are fixed as hyperparameters and dimensions that vary with the length of the sequence that is being processed.

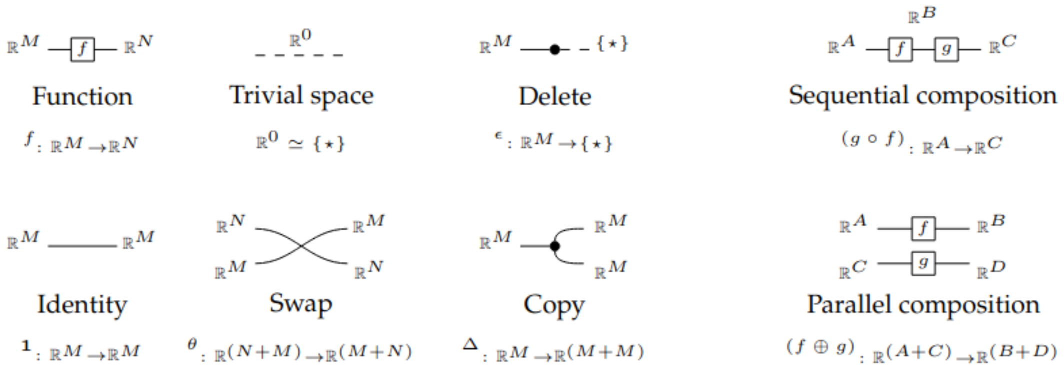


Figure 4.6: Basic elements of the diagrammatic syntax proposed by [KLLWM24]. (Image taken from [KLLWM24].)

4.2.2 Comparing attention mechanisms

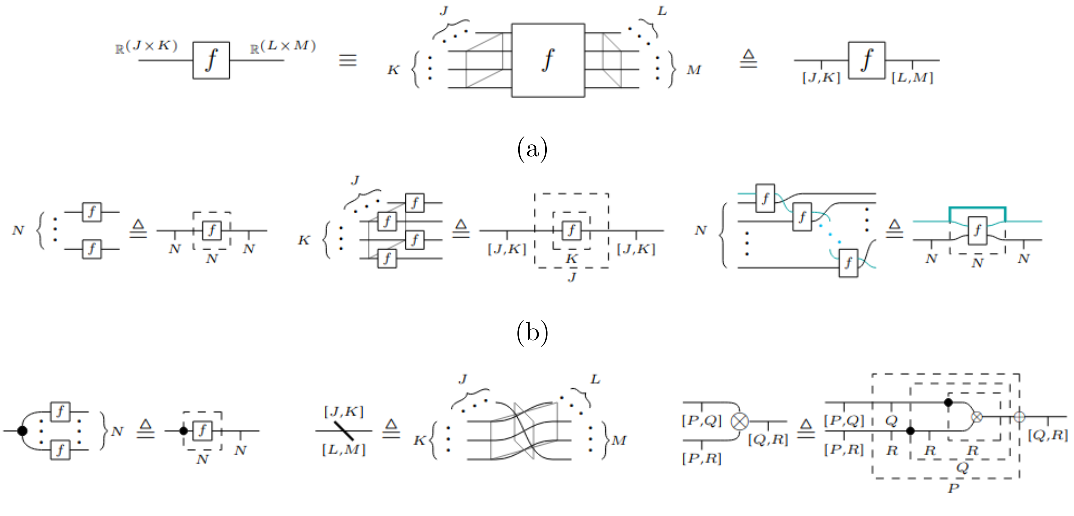
Not only does the abstraction allowed by universal approximators highlight the fundamental structure of architectures, but it also allows comparisons between architectures. Two architectures that only differ at a point can be considered as descendents of a commmon ancestor architecture, and this common link allows us to compare them. For instance, [KLLWM24] derives all known attention mechanisms as descendents of a prototype “primordial attention”, forming a sort of evolutionary tree of attention (see *Fig. 4.10*). The process of filling in a universal approximator with a more specialized structure is called by [KLLWM24] an “expressivity reduction”, as it reduces number of architectures represented by the given schematics. Expressivity reductions can be formalized as string diagrammatic rewrites. Hence, in the example discussed above, rewrites are employed to derive descendent attention mechanisms from the primordial attention.

Just like biological taxonomy classifies species based on their evolutionary history, we can use the tree in *Fig. 4.10* as a blueprint for the classification of attention mechanisms. This undertaking is analogous to the one of [GLD$^+$24], where recurrent neural network cells are classified based on the classical data structures and automata they are related to. Both taxonomies serve to systematize the scattered panorama of machine learning research, but may also inspire new architectures.

The authors of [KLLWM24] do pursue this line of thinking but they do explore a space of possible attention mechanisms freely constructed from five generators (represented as abstract diagrams). This large space is reduced in size by rewriting sections that would lead to overparameterization. This results in a selection of eleven possible attention mechanisms , that the authors of [KLLWM24] go on to test: the mechanisms are embedded in a decoder-only transformer and trained on a Penn-Treebank task. Interestingly, all the 11 mechanisms (among which there are known architectures such as the one proposed by [VSP$^+$17]) perform comparably well (with only one having a slightly better performance than the others). The authors entertain the conjecture that the aforementioned results might be a sign that the choice of attention mechanism matters little, as long as data is exchanged between tokens. While there is no uncontrovertible evidence that this is the case, these experiments prove the utility of the approach of [KLLWM24] and, more generally, of the use of diagrams in machine learning.

4.3 Conclusions and related work

The amount of implementative details conveyed by neural circuit diagrams suggests that it should be possible to find a 1-to-1 corrispondence between the latter and code. In fact, these diagrams are


$\mathbb{R}^{(J\times K)}$
f
$\mathbb{R}^{(L\times M)}$
≡
J
L
K
M
≜
[J,K]
[L,M]
N
[P,Q]
[P,R]
[Q,R]
Q
R
P

(a)

(b)

(c)

Figure 4.7: Syntax proposed by [KLLWM24] for (a) dimensionality, (b) SIMD boxes, (c) copys, reshape operators, and contractions. (Image taken from [KLLWM24].)

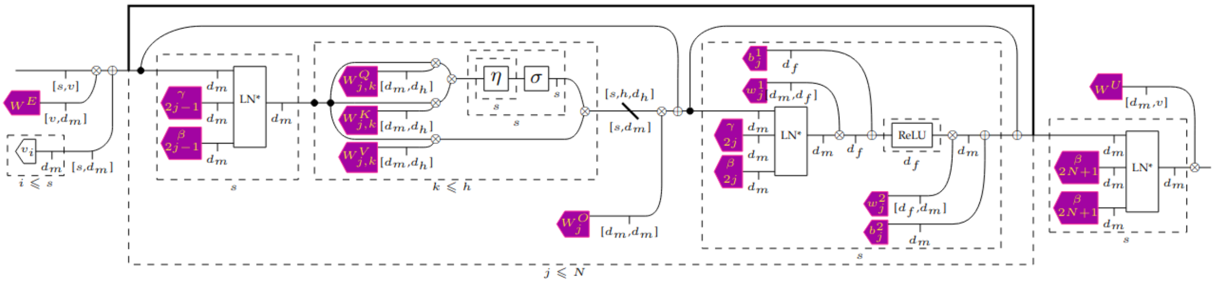

W^E
v_i
[s,v]
[v,dm]
$i \leqslant s$
LN*
η
σ
$k \leqslant h$
[s,h,dh]
[s,dm]
W_j^O
ReLU
d_f
d_m
W^U
[dm,v]
$j \leqslant N$

Figure 4.8: Blueprint of the encoder part of a transformer according to [KLLWM24]. (Image taken from [KLLWM24].)

expressive enough to be considered a high-level language with explicit memory management ([Abb23]). Hopefully, future research will lead to the creations of tools that convert neural circuit diagrams into working models, similarly to the library developed by [CGG$^+$22] that allows programmers to code using parametric lenses.

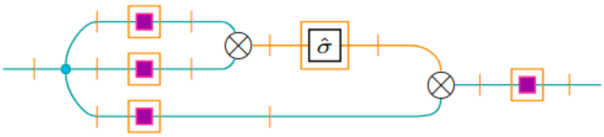
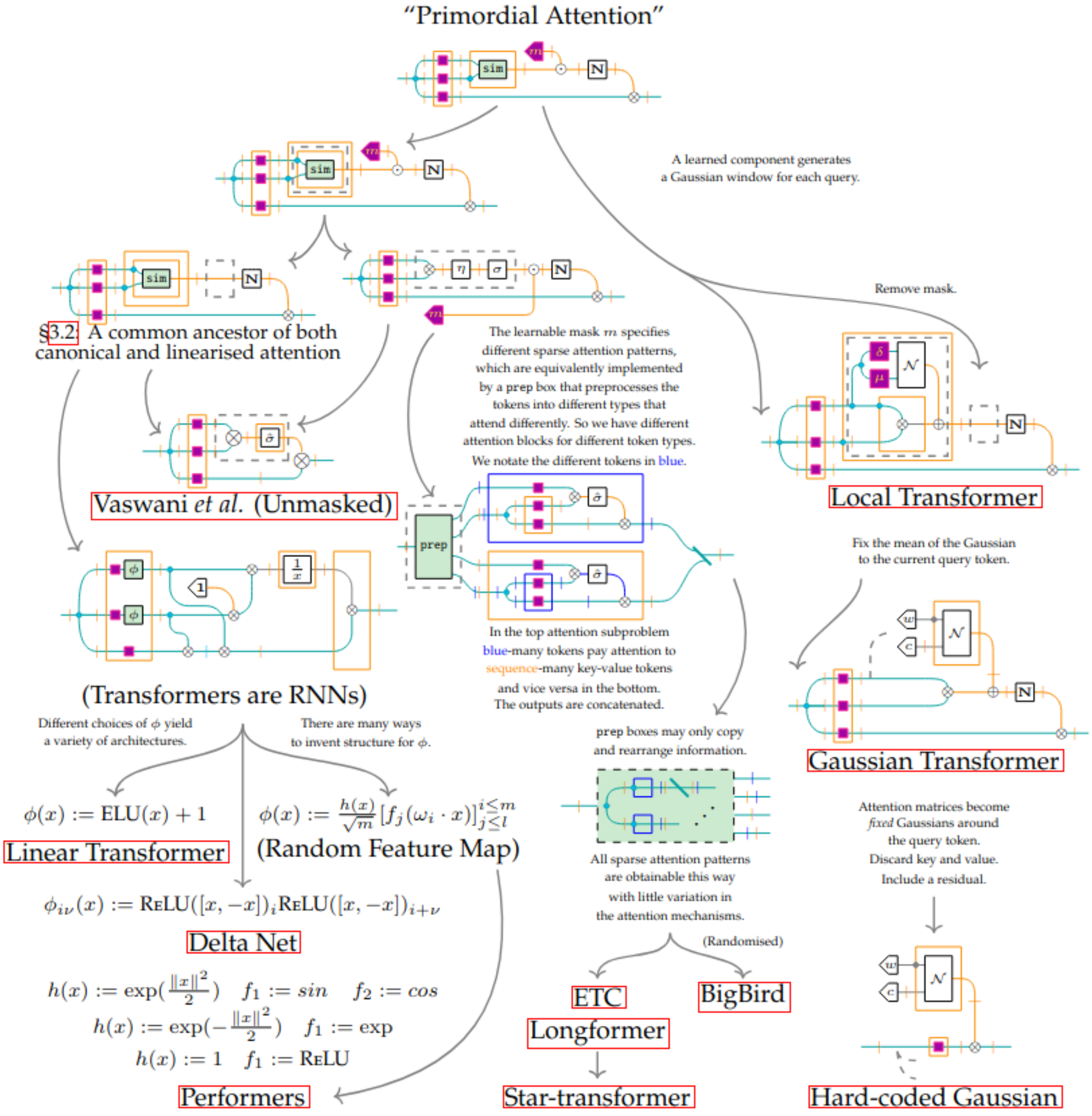


Figure 4.9: Abstract blueprint for the self-attention mechanism for the transformer architecture of [VSP$^+$17]. (Image taken from [KLLWM24].)


"Primordial Attention"
sim
m
N
η
σ
$\hat{\sigma}$
§3.2: A common ancestor of both canonical and linearised attention
Vaswani *et al.* (Unmasked)
ϕ
$\frac{1}{x}$
(Transformers are RNNs)
prep
δ
μ
$\mathcal{N}$
w
c
Local Transformer
Gaussian Transformer
Linear Transformer
(Random Feature Map)
Delta Net
Performers
ETC
BigBird
Longformer
Star-transformer
Hard-coded Gaussian
(Randomised)

A learned component generates a Gaussian window for each query.

Remove mask.

The learnable mask m specifies different sparse attention patterns, which are equivalently implemented by a **prep** box that preprocesses the tokens into different types that attend differently. So we have different attention blocks for different token types. We notate the different tokens in blue.

In the top attention subproblem blue-many tokens pay attention to sequence-many key-value tokens and vice versa in the bottom. The outputs are concatenated.

Fix the mean of the Gaussian to the current query token.

prep boxes may only copy and rearrange information.

All sparse attention patterns are obtainable this way with little variation in the attention mechanisms.

Attention matrices become *fixed* Gaussians around the query token. Discard key and value. Include a residual.

Different choices of ϕ yield a variety of architectures.

There are many ways to invent structure for ϕ.

$\phi(x) := \mathrm{ELU}(x) + 1$

$\phi(x) := \frac{h(x)}{\sqrt{m}}[f_j(\omega_i \cdot x)]_{j \leq l}^{i \leq m}$

$\phi_{i\nu}(x) := \mathrm{ReLU}([x, -x])_i \mathrm{ReLU}([x, -x])_{i+\nu}$

$h(x) := \exp(\frac{\|x\|^2}{2}) \quad f_1 := sin \quad f_2 := cos$

$h(x) := \exp(-\frac{\|x\|^2}{2}) \quad f_1 := \exp$

$h(x) := 1 \quad f_1 := \mathrm{ReLU}$

Figure 1: The taxonomy generated by starting with a 'primordial attention' mechanism and applying specialisations and expressivity rewrites. Details of the notation and rewrites are given in Section 4.

Figure 4.10: Evolutionary tree of attention mechanisms according to [KLLWM24]. Child nodes are obtained by applying rewrites to parent nodes. (Image taken from [KLLWM24].)

Conclusions

Bibliography

[AACFM22] Enrique Amigó, Alejandro Ariza-Casabona, Victor Fresno, and M Antònia Martí. Information theory–based compositional distributional semantics. *Computational Linguistics*, 48(4):907–948, 2022.

[Abb23] Vincent Abbott. *Robust Diagrams for Deep Learning Architectures: Applications and Theory.* PhD thesis, Honours Thesis, The Australian National University, Canberra, 2023.

[Abb24] Vincent Abbott. Neural circuit diagrams: Robust diagrams for the communication, implementation, and analysis of deep learning architectures. *arXiv preprint arXiv:2402.05424*, 2024.

[ASM19] Mahtab Ahmed, Muhammad Rifayat Samee, and Robert E Mercer. Improving tree-lstm with tree attention. In *2019 IEEE 13th international conference on semantic computing (ICSC)*, pages 247–254. IEEE, 2019.

[AZ24] Vincent Abbott and Gioele Zardini. Functor string diagrams: A novel approach to flexible diagrams for applied category theory. *arXiv preprint arXiv:2404.00249*, 2024.

[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.

[BDGC$^+$22] Cristian Bodnar, Francesco Di Giovanni, Benjamin Chamberlain, Pietro Lio, and Michael Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. *Advances in Neural Information Processing Systems*, 35:18527–18541, 2022.

[BQL21] Joshua Bassey, Lijun Qian, and Xianfang Li. A survey of complex-valued neural networks. *arXiv preprint arXiv:2101.12249*, 2021.

[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.

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms.* MIT press, 2022.

[CP07] Stephen Clark and Stephen Pulman. Combining symbolic and distributional models of meaning. 2007.

[CSC10] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning. *arXiv preprint arXiv:1003.4394*, 2010.

[DV22] Andrew J Dudzik and Petar Veličković. Graph neural networks are dynamic programmers. *Advances in neural information processing systems*, 35:20635–20647, 2022.

[DvGPV24] Andrew Joseph Dudzik, Tamara von Glehn, Razvan Pascanu, and Petar Veličković. Asynchronous algorithmic alignment with cocycles. In *Learning on Graphs Conference*, pages 3–1. PMLR, 2024.

[Ell18] Conal Elliott. The simple essence of automatic differentiation. *Proceedings of the ACM on Programming Languages*, 2(ICFP):1–29, 2018.

[EPWJ80] Michael G Eastwood, Roger Penrose, and Raymond O Wells Jr. Cohomology and massless fields. 1980.

[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$^+$24] 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.

[GSR$^+$17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[Han18] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? *Advances in neural information processing systems*, 31, 2018.

[HG20] Jakob Hansen and Thomas Gebhart. Sheaf neural networks. *arXiv preprint arXiv:2012.06333*, 2020.

[IKP+22] Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, et al. A generalist neural algorithmic learner. In *Learning on graphs conference*, pages 2–1. PMLR, 2022.

[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.

[JSV96] André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. In *Mathematical proceedings of the cambridge philosophical society*, volume 119, pages 447–468. Cambridge University Press, 1996.

[KLLWM24] Nikhil Khatri, Tuomas Laakkonen, Jonathon Liu, and Vincent Wang-Maścianica. On the anatomy of attention. *arXiv preprint arXiv:2407.02423*, 2024.

[Lam99] Joachim Lambek. Type grammar revisited. In *Logical Aspects of Computational Linguistics: Second International Conference, LACL'97 Nancy, France, September 22-24, 1997 Selected Papers 2*, pages 1–27. Springer, 1999.

[LCRS22] Vishnu Suresh Lokhande, Rudrasis Chakraborty, Sathya N Ravi, and Vikas Singh. Equivariance allows handling multiple nuisance variables when analyzing pooled neuroimaging datasets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10432–10441, 2022.

[Lew19] Martha Lewis. Compositionality for recursive neural networks. *arXiv preprint arXiv:1901.10723*, 2019.

[Mar14] Daniel Marsden. Category theory using string diagrams. *arXiv preprint arXiv:1401.7220*, 2014.

[Mar19] Yoshihiro Maruyama. Compositionality and contextuality: the symbolic and statistical theories of meaning. In *Modeling and Using Context: 11th International and Interdisciplinary Conference, CONTEXT 2019, Trento, Italy, November 20–22, 2019, Proceedings 11*, pages 161–174. Springer, 2019.

[Nak23] Kenji Nakahira. Diagrammatic category theory. *arXiv preprint arXiv:2307.08891*, 2023.

[OCV+20] Chris Olah, Nick Cammarata, Chelsea Voss, Ludwig Schubert, and Gabriel Goh. Naturally occurring equivariance in neural networks. *Distill*, 5(12):e00024–004, 2020.

[PAS24] Samuel Pfrommer, Brendon G Anderson, and Somayeh Sojoudi. Transport of algebraic structure to latent embeddings. *arXiv preprint arXiv:2405.16763*, 2024.

[PVM+21] Wei Peng, Tuomas Varanka, Abdelrahman Mostafa, Henglin Shi, and Guoying Zhao. Hyperbolic deep neural networks: A survey. *IEEE Transactions on pattern analysis and machine intelligence*, 44(12):10023–10044, 2021.

[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.

[SK19] David Sprunger and Shin-ya Katsumata. Differentiable causal computations via delayed trace. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2019.

[SL06] Paul Smolensky and Géraldine Legendre. *The harmonic mind: From neural computation to optimality-theoretic grammar.* MIT, 2006.

[Spi12] David I Spivak. Functorial data migration. *Information and Computation*, 217:31–51, 2012.

[SXM23] Eli Sennesh, Tom Xu, and Yoshihiro Maruyama. Computing with categories in machine learning. In *International Conference on Artificial General Intelligence*, pages 244–254. Springer, 2023.

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

[VB21] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7), 2021.

[VBB$^+$22] Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pages 22084–22102. PMLR, 2022.

[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.

[Wil10] Simon Willerton. Integral transforms and the pull-push perspective, 2010.

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

[WJL$^+$22] Isaac Ronald Ward, Jack Joyner, Casey Lickfold, Yulan Guo, and Mohammed Bennamoun. A practical tutorial on graph neural networks. *ACM Computing Surveys (CSUR)*, 54(10s):1–35, 2022.

[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.

[XLZ$^+$19] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

[Zag24] Olga Zaghen. Nonlinear sheaf diffusion in graph neural networks. *arXiv preprint arXiv:2403.00337*, 2024.

[ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

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 me throughout this journey, and without whom all of this would not have been possible. To all of you, my most sincere gratitude.