Position: Categorical Deep Learning is an Algebraic Theory of All Architectures

Bruno Gavranović [*1,2] **Paul Lessard** [*1] **Andrew Dudzik** [*3]
Tamara von Glehn [3] **João G.M. Araújo** [3] **Petar Veličković** [3,4]

Abstract

We present our position on the elusive quest for a general-purpose framework for specifying and studying deep learning architectures. Our opinion is that the key attempts made so far lack a coherent bridge between specifying *constraints* which models must satisfy and specifying their *implementations*. Focusing on building a such a bridge, we propose to apply category theory—precisely, the universal *algebra of monads* valued in a 2-category of *parametric maps*—as a single theory elegantly subsuming both of these flavours of neural network design. To defend our position, we show how this theory recovers constraints induced by geometric deep learning, as well as implementations of many architectures drawn from the diverse landscape of neural networks, such as RNNs. We also illustrate how the theory naturally encodes many standard constructs in computer science and automata theory.

1. Introduction

One of the most coveted aims of deep learning theory is to provide a guiding *framework* from which all neural network architectures can be principally and usefully derived. Many elegant attempts have recently been made, offering frameworks to categorise or describe large swathes of deep learning architectures: Cohen et al. (2019); Xu et al. (2019); Bronstein et al. (2021); Chami et al. (2022); Papillon et al. (2023); Jogl et al. (2023); Weiler et al. (2023) to name a few.

The title of the paper should be read as "Categorical Deep Learning is an Algebraic {Theory of All Architectures}", *not* "Categorical Deep Learning is an {Algebraic Theory} of All Architectures".

[*]Equal contribution [1]Symbolica AI [2]University of Edinburgh [3]Google DeepMind [4]University of Cambridge. Correspondence to: Bruno Gavranović <bruno@brunogavranovic.com>, Paul Lessard <paul@symbolica.ai>, Andrew Dudzik <adudzik@google.com>, Petar Veličković <petarv@google.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

We observe that there are, typically, two broad ways in which deep learning practitioners describe models. Firstly, neural networks can be specified in a *top-down* manner, wherein models are described by the *constraints* they should satisfy (e.g. in order to respect the structure of the data they process). Alternatively, a *bottom-up* approach describes models by their *implementation*, i.e. the sequence of tensor operations required to perform their forward/backward pass.

1.1. Our Opinion

It is our **opinion** that ample effort has already been given to both the top-down and bottom-up approaches *in isolation*, and that there hasn't been sufficiently expressive theory to address them both *simultaneously*. *If we want a **general** guiding framework for **all** of deep learning, this needs to change.* To substantiate our opinion, we survey a few ongoing efforts on both sides of the divide.

One of the most successful examples of the top-down framework is *geometric deep learning* (Bronstein et al., 2021, GDL), which uses a group- and representation-theoretic perspective to describe neural network layers via symmetry-preserving constraints. The actual realisations of such layers are derived by solving *equivariance constraints*.

GDL proved to be powerful: allowing, e.g., to cast *convolutional layers* as an exact solution to linear translation equivariance in grids (Fukushima et al., 1983; LeCun et al., 1998), and *message passing* and *self-attention* as instances of permutation equivariant learning over graphs (Gilmer et al., 2017; Vaswani et al., 2017). It also naturally extends to exotic domains such as *spheres* (Cohen et al., 2018), *meshes* (de Haan et al., 2020b) and *geometric graphs* (Fuchs et al., 2020). While this elegantly covers many architectures of practical interest, GDL also has inescapable constraints.

Firstly, usability of GDL principles to *implement* architectures directly correlates with how easy it is to resolve equivariance constraints. While `PyG` (Fey & Lenssen, 2019), `DGL` (Wang, 2019) and `Jraph` (Godwin et al., 2020) have had success for permutation-equivariant models, and `e3nn` (Geiger & Smidt, 2022) for $\mathrm{E}(3)$-equivariant models, it is hard to replicate such success for areas where it is not known how to resolve equivariance constraints.

Because of its focus on groups, GDL is only able to represent equivariance to symmetries, but not all operations we may wish neural networks to align to are invertible (Worrall & Welling, 2019) or fully compositional (de Haan et al., 2020a). This is not a small collection of operations either; if we'd like to align a model to an arbitrary *algorithm* (Xu et al., 2019), it is fairly common for the target algorithm to irreversibly transform data, for example when performing any kind of a path-finding contraction (Dudzik & Veličković, 2022). Generally, in order to reason about alignment to constructs in **computer science**, we must go beyond GDL.

On the other hand, bottom-up frameworks are most commonly embodied in *automatic differentiation* packages, such as `TensorFlow` (Abadi et al., 2016), `PyTorch` (Paszke et al., 2019) and `JAX` (Bradbury et al., 2018). These frameworks have become indispensable in the implementation of deep learning models at scale. Such packages often have grounding in *functional programming*: perhaps `JAX` is the most direct example, as it is marketed as *"composable function transformations"*, but such features permeate other deep learning frameworks as well. Treating neural networks as "pure functions" allows for rigorous analysis on their computational graph, allowing a degree of type- and shape-checking, as well as automatic tensor shape inference and fully automated backpropagation passes.

The issues, again, happen closer to the boundary between the two directions—specifying and controlling for *constraint satisfaction* is not simple with tensor programming. Inferring general properties (*semantics*) of a program from its implementation (*syntax*) alone is a substantial challenge for all but the simplest programs, pointing to a need to model more abstract properties of **computer science** than existing frameworks can offer directly. The similarity of the requirement on both sides leads us to our present position.

1.2. Our Position

It is our **position** that constructing a guiding framework for all of deep learning, requires robustly *bridging* the top-down and bottom-up approaches to neural network specification with a *unifying mathematical theory*, and that the concepts for this bridging should be coming from **computer science**. *Moreover, such a framework must generalise both* ***group theory*** *and* ***functional programming****—and a natural candidate for achieving this is* ***category theory****.*

It is worth noting that ours is not the first approach to either (a) observe neural networks through the lens of computer science constructs (Baydin et al., 2018), (b) explore the connection between syntax and semantics in neural networks (Sonoda et al., 2023a;b; 2024) (b) apply Category Theory to machine learning (Gavranović, 2020).

However, we are unaware of any prior work that tackles the connection of neural network architectures and the algebras of parametric maps, as we will do in this paper. Further, prior art in syntax-semantics connections either assumes that the operations are taking place in some topological space or that neural network architectures have a very specific form—our framework assumes neither. Lastly, prior papers exploring Category Theory and Machine Learning are fragmented, scarce, and not cohesive—our paper seeks to establish a common, unifying framework for how category theory can be applied to AI.

To defend our position, we will demonstrate a unified categorical framework that is expressive enough to rederive standard GDL concepts (invariance and equivariance), specify implementations of complex neural network building blocks (recurrent neural networks), as well as model other intricate deep learning concepts such as weight tying.

1.3. The Power of Category Theory

To understand where we are going, we must first put the field of category theory in context. Minimally, it may be conceived of as a battle-tested system of *interfaces* that are learned once, and then reliably applied across scientific fields. Originating in abstract mathematics, specifically algebraic topology, category theory has since proliferated, and been used to express ideas from numerous fields in an uniform manner, helping reveal their previously unknown shared aspects. Other than modern pure mathematics, which it thoroughly permeates, these fields include *systems theory* (Capucci et al., 2022; Niu & Spivak, 2023), *bayesian learning* (Braithwaite et al., 2023; Cho & Jacobs, 2019), and *information theory and probability* (Leinster, 2021; Bradley, 2021; Sturtz, 2015; Heunen et al., 2017; Perrone, 2022).

This growth has resulted in a reliable set of mature theories and tools; from algebra, geometry, topology, combinatorics to recursion and dependent types, etc. all of them with a mutually compatible interface. Recently category theory has started to be applied to machine learning, in *automatic differentiation* (Vákár & Smeding, 2022; Alvarez-Picallo et al., 2021; Gavranović, 2022; Elliott, 2018), *topological data analysis* (Guss & Salakhutdinov, 2018), *natural language processing* (Lewis, 2019), *causal inference* (Jacobs et al., 2019; Cohen, 2022), even producing an entire categorical picture of gradient-based learning – from architectures to backprop – in Cruttwell et al. (2022); Gavranović (2024), with a more implementation-centric view in Nguyen & Wu (2022), and important earlier work (Fong et al., 2021).

1.3.1. Essential Concepts

Before we begin, we recall three essential concepts in category theory, that will be necessary for following our exposition. First, we define a *category*, an elegant axiomatisation of a compositional structure.

Definition 1.1 (Category)**.** A category, $\mathcal{C}$, consists of a collection[1] of *objects*, and a collection of *morphisms* between pairs of objects, such that:

- For each object $A \in \mathcal{C}$, there is a unique *identity* morphism $\mathrm{id}_A : A \to A$.
- For any two morphisms $f : A \to B$ and $g : B \to C$, there must exist a unique morphism which is their *composition* $g \circ f : A \to C$.

subject to the following conditions:

- For any morphism $f : A \to B$, it holds that $\mathrm{id}_B \circ f = f \circ \mathrm{id}_A = f$.
- For any three composable morphisms $f : A \to B$, $g : B \to C$, $h : C \to D$, composition is *associative*, i.e., $h \circ (g \circ f) = (h \circ g) \circ f$.

We denote by $\mathcal{C}(A, B)$ the collection of all morphisms from $A \in \mathcal{C}$ to $B \in \mathcal{C}$.

We provide a typical first example:

Example 1.2 (The Set Category)**.** Set is a category whose objects are *sets*, and morphisms are *functions* between them.

And another example, important for geometric DL:

Example 1.3 (Groups and monoids as categories)**.** A group, G, can be represented as a category, $\mathcal{B}G$, with a single object (G), and morphisms $g : G \to G$ corresponding to elements $g \in G$, where composition is given by the group's binary operation. Note that G is a group if and only if these morphisms are *isomorphisms*, that is, for each $g : G \to G$ there exists $h : G \to G$ such that $h \circ g = g \circ h = \mathrm{id}_G$. More generally, we can identify one-object categories, whose morphisms are not necessarily invertible, with monoids.

The power of category theory starts to emerge when we allow different categories to *interact*. Just as there are functions of sets and homomorphisms of groups, there is a more generic concept of *structure preserving maps* between categories, called *functors*.

Definition 1.4 (Functor)**.** Let $\mathcal{C}$ and $\mathcal{D}$ be two categories. Then, $F : \mathcal{C} \to \mathcal{D}$ is a functor between them, if it maps each object and morphism of $\mathcal{C}$ to a corresponding one in $\mathcal{D}$, and the following two conditions hold:

- For any object $A \in \mathcal{C}$, $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$.
- For any composable morphisms f, g in $\mathcal{C}$, $F(g \circ f) = F(g) \circ F(f)$.

[1]The term "collection", rather than set, avoids Russell's paradox, as objects may themselves be sets. Categories that can be described with sets are known as *small categories*.

An *endofunctor* on $\mathcal{C}$ is a functor $F : \mathcal{C} \to \mathcal{C}$.

Just as a functor is an interaction between categories, a *natural transformation* specifies an interaction between functors; this is the third and final concept we cover here.

Definition 1.5 (Natural transformation)**.** Let $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$ be two functors between categories $\mathcal{C}$ and $\mathcal{D}$. A natural transformation $\alpha : F \Rightarrow G$ consists of a choice, for every object $X \in \mathcal{C}$, of a morphism $\alpha_X : F(X) \to G(X)$ in $\mathcal{D}$ such that, for every morphism $f : X \to Y$ in $\mathcal{C}$, it holds that $\alpha_Y \circ F(f) = G(f) \circ \alpha_X$.

The morphism α_X is called the component of the natural transformation α at the object X.

The components of a natural transformation assemble into "naturality squares", commutative diagrams:

$$\begin{tikzcd} F(X) \arrow[r, "F(f)"] \arrow[d, "\alpha_X"'] & F(Y) \arrow[d, "\alpha_Y"] \\ G(X) \arrow[r, "G(f)"'] & G(Y) \end{tikzcd}$$

where a diagram *commutes* if, for any two objects, any two paths connecting them correspond to the same morphism.

2. From Monad Algebras to Equivariance

Having set up the essential concepts, we proceed on our quest to define a categorical framework which subsumes and generalises geometric deep learning (Bronstein et al., 2021). First, we will define a powerful notion (**monad algebra homomorphism**) and demonstrate that the special case of monads induced by *group actions* is sufficient to describe *geometric deep learning*. Generalising from monads and their algebras to arbitrary endofunctors and their algebras, we will find that our theory can express functions that process structured data from computer science (e.g. *lists* and *trees*)[2] and behave in stateful ways like *automata*.

2.1. Monads and their Algebras

Definition 2.1 (Monad)**.** Let $\mathcal{C}$ be a category. A monad on $\mathcal{C}$ is a triple (M, η, μ) where $M : \mathcal{C} \to \mathcal{C}$ is an endofunctor, and $\eta : \mathrm{id}_{\mathcal{C}} \Rightarrow M$ and $\mu : M \circ M \Rightarrow M$ are natural transformations (where here $_ \circ _$ is functor composition), making diagrams in Definition B.1 commute.

Example 2.2 (Group action monad)**.** Let G be a group. Then the triple $(G \times -, \eta, \mu)$ is a monad on Set, where

- $G \times - : \mathsf{Set} \to \mathsf{Set}$ is an endofunctor mapping a set X to the set $G \times X$;

[2]To the best of our knowledge, these ideas were first conjectured in Olah (2015) in the language of functional programming.

- $\eta : \mathrm{id}_{\mathsf{Set}} \Rightarrow G \times - : \mathsf{Set} \to \mathsf{Set}$ whose component at a set X is the function $x \mapsto (e, x)$ where e is the identity element of the group G; and
- $\mu : G \times G \times - \Rightarrow G \times - : \mathsf{Set} \to \mathsf{Set}$ whose component at a set X is the function $(g, h, x) \mapsto (gh, x)$ with the implicit multiplication that of the group G.

Group action monads are formal theories of group actions, but they do not allow us to actually *execute* them on data. This is what *algebras* do.

Definition 2.3 (Algebra for a monad)**.** An algebra for a monad (M, η, μ) on a category $\mathcal{C}$ is a pair (A, a), where $A \in \mathcal{C}$ is a *carrier object* and $a : M(A) \to A$ is a morphism of $\mathcal{C}$ (*structure map*) making the following diagram commute:

$$\begin{tikzcd} A \arrow[r, "\eta_A"] \arrow[dr, equal] & M(A) \arrow[d, "a"] \\ & A \end{tikzcd} \qquad \begin{tikzcd} M(M(A)) \arrow[r, "M(a)"] \arrow[d, "\mu_A"'] & M(A) \arrow[d, "a"'] \\ M(A) \arrow[r, "a"'] & A \end{tikzcd}$$

Example 2.4 (Group actions)**.** Group actions for a group G arise as algebras of the aforementioned group action monad $G \times -$. Consider the carrier $\mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h}$, thought of as data on a $w \times h$ grid, and any of the usual group actions on $\mathbb{Z}_w \times \mathbb{Z}_h$: translation, rotation, permutation, scaling, or reflections.

Each of these group actions induce an algebra on the carrier set $\mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h}$. For instance, the translation group $(\mathbb{Z}_w \times \mathbb{Z}_h, +, 0)$ induces the algebra

$$\blacktriangleright : \mathbb{Z}_w \times \mathbb{Z}_h \times \mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h} \to \mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h}$$

defined as $((i', j') \blacktriangleright x)(i, j) = x(i - i', j - j')$. Here x represents the grid data, i, j specific pixel locations, and i', j' the translation vector. We also specifically mention the trivial action of any group $\pi_X : G \times X \to X$ by projection.

A monad algebra can capture a particular input or output for group equivariant neural networks (as its carrier). That being said, geometric deep learning concerns itself with *linear equivariant layers* between these inputs and outputs. In order to be able to describe those, we need to establish the concept of a *morphism of algebras* for a monad.

Definition 2.5 (M-algebra homomorphism)**.** Let (M, μ, η) be a monad on $\mathcal{C}$, and (A, a) and (B, b) be M-algebras. An M**-algebra homomorphism** $(A, a) \to (B, b)$ is a morphism $f : A \to B$ of $\mathcal{C}$ s.t. the following commutes:

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

We recover equivariant maps as morphisms of algebras.

Example 2.6 (Equivariant maps)**.** Equivariant maps are group action monad algebra homomorphisms. Consider any action from Example 2.4. An endomorphism of such an action—that is, a G-algebra on $\mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h}$—is an endormorphism of $\mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h}$ which induces a commutative diagram

$$\begin{tikzcd} G \times \mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h} \arrow[r, "G \times f"] \arrow[d, "\blacktriangleright"'] & G \times \mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h} \arrow[d, "\blacktriangleright"] \\ \mathbb{R}^{Z_w \times \mathbb{Z}_h} \arrow[r, "f"'] & \mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h} \end{tikzcd}$$

which, elementwise, unpacks to the equation

$$f(g \blacktriangleright x) = g \blacktriangleright f(x)$$

The translation example, for instance, recovers the equation $f(((i', j') \blacktriangleright x)(i, j)) = (i', j') \blacktriangleright f(x)(i, j)$ which reduces to the usual constraint: $f(x(i - i', j - j')) = f(x)(i - i', j - j')$.

The concept of *invariance* — a special case of equivariance — is unpacked in the Appendix (Example H.3). It's worth reflecting on the fact that we have just successfully derived the key aim of **geometric deep learning**: finding neural network layers that are *monad algebra homomorphisms* of monads associated with *group actions*!

Indeed, the template illustrated in Example 2.6 is sufficient to explain *any* architectures which are explained by Geometric DL; should the reader wish to see concrete examples—deriving graph neural networks (Veličković, 2023), Spherical CNNs (Cohen et al., 2018) and G-CNNs (Cohen & Welling, 2016)—they may be found in Appendix C.

To concretely derive such layers from these constraints, we need to make concrete the category in which $f : A \to B$ lives. A standard choice is to use Vect, a category where objects are finite-dimensional *vector spaces* and morphisms are *linear maps* between these spaces. In such a setting, morphisms can be specified as *matrices*, and the equivariance condition places constraints on the matrix's entries, resulting in effects such as *weight sharing* or *weight tying*. We provide a detailed derivation for two examples on a two-pixel grid in Appendix H.1.

Remark 2.7. When our monad is of the form $M \times -$, with M a monoid, algebras are equivalent to M-actions, i.e. functors $\mathcal{B}M \to \mathsf{Set}$, where $\mathcal{B}M$ is the one-object category given in Example 1.3, and algebra morphisms are equivalent to natural transformations. So in this case, our definition of equivariance coincides with the functorial version given in de Haan et al. (2020a). But the connection here is much deeper—for any monad, we can think of its algebras, which we can think of as the semantics of the monad, as functors on certain categories encoding the monad's syntax, such as *Lawvere theories*. For example, Dudzik & Veličković (2022) use the fact that functors on the category of finite polynomial

diagrams[3] encode the algebraic structure of commutative semirings. We give further details on this connection in Appendix D.

2.2. Endofunctors and their (Co)algebras

Geometric deep learning, while elegant, is fundamentally constrained by the axioms of group theory. Monads and their algebras, however, are naturally generalised beyond group actions. Here we show how, by studying (co)algebras of arbitrary *endofunctors*, we can rediscover standard computer science constructs like lists, trees and automata. This rediscovery is not merely a passing observation; in fact, the endofunctor view of lists and trees turns out to naturally map to *implementations* of neural architectures such as recurrent and recursive neural networks; see Appendix I.

Then, in the next section we'll show how these more minimal structures, endofunctors and their algebras, may be augmented into the more structured notions of monads and their algebras.

Definition 2.8 (Algebra for an endofunctor)**.** Let $\mathcal{C}$ be a category and $F : \mathcal{C} \to \mathcal{C}$ an endofunctor on $\mathcal{C}$. An algebra for F is a pair (A, a) where A is an object of $\mathcal{C}$ and $a : F(A) \to A$ is a morphism of $\mathcal{C}$.

Note that, compared to Definition 2.3, there are no equations this time; F is not equipped with any extra structure with which the structure map of an algebra could be compatible. Examples of endofunctor algebras abound (Jacobs, 2016), many of which are familiar to computer scientists.

Example 2.9 (Lists)**.** Let A be a set, and consider the endofunctor $1 + A \times - : \mathsf{Set} \to \mathsf{Set}$. The set $\mathsf{List}(A)$ of lists of elements of type A together with the map $[\mathsf{Nil}, \mathsf{Cons}] : 1 + A \times \mathsf{List}(A) \to \mathsf{List}(A)$ forms an algebra of this endofunctor.[4]: Here Nil and Cons are two constructors for lists, allowing us to represent lists as the following datatype:

```
data List a = Nil
            | Cons a (List a)
```

It describes $\mathsf{List}(A)$ *inductively*, as being formed either out of the empty list, or an element of type A and another list. In Figure 1 we will see how this relates to *folding* RNNs.

Example 2.10 (Binary trees)**.** Let A be a set. Consider the endofunctor $A + (-)^2 : \mathsf{Set} \to \mathsf{Set}$. The set $\mathsf{Tree}(A)$ of binary trees with A-labelled leaves, together with the map $[\mathsf{Leaf}, \mathsf{Node}] : A + \mathsf{Tree}(A)^2 \to \mathsf{Tree}(A)$ forms an algebra of this endofunctor. Here Leaf and Node are constructors for binary trees, enabling the following datatype representation:

```
data Tree a = Leaf a
            | Node (Tree a) (Tree a)
```

It describes $\mathsf{Tree}(A)$ inductively, as being formed either out of a single A-labelled leaf or two subtrees.[5] In Figure 1 we will relate this to recursive neural networks.

Dually, we also study *coalgebras* for an endofunctor (where the structure morphism $a : A \to F(A)$ points the other way (Definition B.2). Intuitively, while algebras offer us a way to model computation guaranteed to terminate, coalgebras offer us a way to model potentially infinite computation. They capture the semantics of programs whose guarantee is not termination, bur rather *productivity* (Atkey & McBride, 2013), and as such are excellent for describing servers, operating systems, and automata (Rutten, 2000; Jacobs, 2016). We will use endofunctor coalgebras to describe one such automaton—the Mealy machine (Mealy, 1955).

Example 2.11 (Mealy machines)**.** Let O and I be sets of possible *outputs* and *inputs*, respectively. Consider the endofunctor $(I \to O \times -) : \mathsf{Set} \to \mathsf{Set}$. Then the set $\mathsf{Mealy}_{O,I}$ of Mealy machines with outputs in O and inputs in I, together with the map $\mathsf{next} : \mathsf{Mealy}_{O,I} \to (I \to O \times \mathsf{Mealy}_{O,I})$ is a coalgebra of this endofunctor.

```
data Mealy o i = MkMealy {
  next :: i -> (o, Mealy o i)
}
```

This describes Mealy machines *coinductively*, as *systems* which, given an input, produce an output and another Mealy machine. In Figure 1 we will relate this to full recurrent neural networks, and, in Examples H.4 and H.7 we coalgebraically express two other fundamental classes of automata: streams and Moore machines.

We have expressed data structures and automata using (co)algebras for an endofunctor. Just as in the case of GDL, in order to describe (linear) *layers of neural networks* between them, we need to establish the concept of a *homomorphism of endofunctor (co)algebras*. The definition of a homomorphism of algebras for an endofunctor mirrors[6] Definition 2.5, while the definition of a homomorphism of coalgebras has the structure maps pointing the other way.

Example 2.12 (Folds over lists as algebra homomorphisms)**.** Consider the endofunctor $(1 + A \times -)$ from Example 2.9, and an algebra homomorphism from $(\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}])$

[3] Equivalently, the category of finitary dependent polynomial functors.

[4] $[f, g] : A + B \to C$ is notation for maps out of a coproduct; where $f : A \to C$ and $g : B \to C$

[5] This framework can also model any variations, e.g., n-ary trees with A-labelled leaves as algebras of $A + \mathsf{List}(-)$, or binary trees with A-labelled nodes as algebras of $1 + A \times (-)^2$.

[6] Because it does not rely on the extra structure monads have.

to any other $(1 + A \times -)$-algebra $(X, [r_0, r_1])$:

$$\begin{tikzcd} 1 + A \times \mathsf{List}(A) \arrow[r, "1+A\times f_r"] \arrow[d, "{[\mathsf{Nil},\mathsf{Cons}]}"'] & 1 + A \times X \arrow[d, "{[r_0,r_1]}"] \\ \mathsf{List}(A) \arrow[r, "f_r"'] & X \end{tikzcd}$$

Then the map $f_r : \mathsf{List}(A) \to X$ is, necessarily, a *fold* over a list, a concept from functional programming which describes how a single value is obtained by operating over a list of values. It is implemented by recursion on the input:

```
f_r :: List a -> x
f_r Nil = r_0 ()
f_r (Cons h t) = r_1 h (f_r t)
```

This recursion is *structural* in nature, meaning it satisfies the following two equations which arise by unpacking the algebra homomorphism equations elementwise:

$$f_r(\mathsf{Nil}) = r_0(\bullet) \tag{1}$$
$$f_r(\mathsf{Cons}(h, t)) = r_1(h, f_r(t)) \tag{2}$$

Equation (1) tells us that we get the same result if we apply f_r to Nil or apply r_0 to the unique element of the singleton set[7]. Equation (2) tells us that, starting with the head and tail of a list, we get the same result if we concatenate the head to the tail, and then process the entire list with f_r, or if we process the tail first with f_r, and then combine the result with the head using r_1.

It is important to remark that these equations ***generalise*** *equivariance constraints* over a list structure. Both group equivariance and Equations (1) and (2) intuitively specify a function that is predictably affected by certain operations—but for the case of lists, these operations (concatenating) are *not* group actions, as attaching an element to the front of the list does *not* leave the list unchanged.

Remark 2.13. Interestingly, given an algebra $(X, [r_0, r_1])$, there can only ever be *one* algebra homomorphism from lists to it! This is because $(\mathsf{List}(A), [\mathsf{Nil}, \mathsf{Cons}])$ is an *initial object* (Definition A.2) in the category of $(1+A\times-)$-algebras. The fact that these are initial arises from a deeper fact related to the fact that, in many cases, for a given endofunctor there is a monad whose category of monad algebras is equivalent to the original category of endofunctor algebras. We note this because the construction which takes us from one to the other, the so-called *algebraically free monad* on an endofunctor, will be seen in Part 3 to derive RNNs and other similar architectures from first principles.

Example 2.14 (Tree folds as algebra homomorphisms)**.** Consider the endofunctor $A+(-)^2$ from Example 2.10, and an algebra homomorphism from $(\mathsf{Tree}(A), [\mathsf{Leaf}, \mathsf{Node}])$ to any other $(A + (-)^2)$-algebra $(X, [r_0, r_1])$:

$$\begin{tikzcd} A + \mathsf{Tree}(A)^2 \arrow[r, "A+f_r^2"] \arrow[d, "{[\mathsf{Leaf},\mathsf{Node}]}"'] & A + X^2 \arrow[d, "{[r_0,r_1]}"] \\ \mathsf{Tree}(A)^2 \arrow[r, "f_r"'] & X \end{tikzcd}$$

Then the map f_r is necessarily a *fold* over a tree. As with lists, it is implemented by recursion on the input, which is *structural* in nature:

```
f_r :: Tree a -> x
f_r (Leaf a) = r_0 a
f_r (Node l r) = r_1 (f_r l) (f_r r)
```

This means that it satisfies the following two equations which arise by unpacking the algebra homomorphism equations elementwise:

$$f_r(\mathsf{Leaf}(a)) = r_0(a) \tag{3}$$
$$f_r(\mathsf{Node}(l, r)) = r_1(f_r(l), f_r(r)) \tag{4}$$

These can also be thought as describing generalised equivariance over binary trees, analogously to lists.

Dual to algebra homomorphisms and folds over inductive data structures, *coalgebra homomorphisms* are categorical semantics of *unfolds* over coinductive data structures.

Example 2.15 (Unfolds as coalgebra homomorphisms)**.** Consider the endofunctor $(I \to O \times -)$ from Example 2.11, and a coalgebra homomorphism from $(\mathsf{Mealy}_{O,I}, \mathsf{next})$ to any other $(I \to O \times -)$-coalgebra (X, n):

$$\begin{tikzcd} X \arrow[r, "f_n"] \arrow[d, "n"'] & \mathsf{Mealy}_{O,I} \arrow[d, "\mathsf{next}"] \\ (I \to O \times X) \arrow[r, "{(I\to O\times f_n)}"'] & (I \to O \times \mathsf{Mealy}_{O,I}) \end{tikzcd}$$

The map f_n here can be thought of as a generalised *unfold*, a concept from functional programming describing how a potentially infinite data structure is obtained from a single value. It is implemented by a *corecursive* function:

```
f_n :: x -> Mealy o i
f_n x = MkMealy
   \i -> let (o', x') = n x i
         in (o', f_n x')
```

which is again structural in nature. This means that it satisfies the following two equations which arise by unpacking the coalgebra homomorphism equations elementwise:

$$n(x)(i)_1 = \mathsf{next}(f_n(x))(i)_1 \tag{5}$$
$$f_n(n(x)(i)_2) = \mathsf{next}(f_n(x))(i)_2 \tag{6}$$

[7]Where `()` was used to denote it in Haskell notation.

Equation (5) tells us that the output of the Mealy machine produced by f_n at state x and input i is given by the output of n at state x and input i, and Equation (6) tells us that the next Mealy machine produced at x and i is the one produced by f_n at $n(x)_2(i)$.

This, too, ***generalises equivariance*** constraints, now describing an interactive automaton which is by no means invertible. Instead, it is *dynamic* in nature, producing outputs which are dependent on the current state of the machine and previously unknown inputs. Lastly, in Examples H.5 and H.8, we show that two kinds of automata—streams and Moore machines—are also examples of coalgebra homomorphisms. Just as before, we can embed all of our objects and morphisms into Vect to study the weight sharing constraints induced by such a condition—see Example H.5.

2.3. Where to Next?

Let's take a step back and understand what we've done. We have shown that an existing categorical framework *uniformly* captures a number of different data structures and automata, as particular (co)algebras of an endofunctor. By choosing a well-understood data structure, we induce a structural constraint on the control flow of the corresponding neural network, by utilising homomorphisms of these endofunctor (co)algebras. These follow the same recipe as monad algebra homomorphisms, and hence can be thought of as *generalising equivariance*—describing functions *beyond* what geometric deep learning can offer.

This is concrete evidence for our position—that ***categorical algebra homomorphisms*** *are suitable for capturing various constraints one can place on deep learning architectures*. Our evidence so far rested on *endofunctor algebras*, which are a particularly fruitful variant.

However, this construct leaves much to be desired. One major issue is that, to prescribe any notion of *weight sharing*, for all of these examples we have implicitly assumed homomorphisms to be linear transformations by placing them into the category Vect. But most neural networks aren't simply linear maps, meaning that these analyses are limited to analysing their individual layers. In the standard example of recurrent neural networks, an RNN cell is an *arbitrary* differentiable function, usually composed out of a sequence of linear and non-linear maps. Further, in both practice and theory, neural networks are treated as *parametric* functions. Specifically, in frameworks like `JAX`, the parameters often need to be passed *explicitly* to any forward or backward pass of a neural network.

How can we explicitly model parameters and non-linear maps, without abandoning the presented categorical framework? Furthermore, in practice—with recurrent or recursive (Socher et al., 2013) neural networks, for instance—there are established techniques for weight sharing. Can we establish formal criteria for when these techniques are *correct*? Just as how we generalised GDL to the setting of category theory, we can go further: to the setting of *2-categories*—the setting we use to study *parametric morphisms*.

3. 2-Categories and Parametric Morphisms

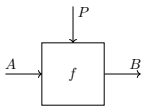
While category theory is a powerful framework, it leaves much to be desired in terms of *higher-order relationships* between morphisms. It only deals with *sets* of morphisms, with no possible way to *compare* elements of these sets. This is where the theory of *2-categories* comes in, which deals with an entire *category* of morphisms. While in a (1-)category, one has objects and morphisms between objects, in a 2-category one has objects (known as *0-morphisms*), morphisms between objects (*1-morphisms*), and morphisms between *morphisms* (*2-morphisms*). We have, in fact, already secretly seen an instance of a 2-category, Cat, when defining the essential concepts of category theory. Specifically, in Cat, objects are *categories*, morphisms are *functors* between them, and 2-morphisms are *natural transformations* between functors.

3.1. The 2-category Para

In this section we define an established 2-category Para (Cruttwell et al., 2022; Capucci et al., 2022), and proceed to unpack the manner which we posit weight sharing can be modelled formally in it.

While it shares objects with the category Set, its 1-morphisms are not functions, but *parametric* functions. That is, a 1-morphism $A \to B$ here consists of a pair (P, f), where $P \in \mathsf{Set}$ and $f : P \times A \to B$.

Para morphisms admit an elegant graphical formalism. Parameters (P) are drawn *vertically*, signifying that they are part of the *morphism*, and not objects.



The 2-category Para models the algebra of composition of neural networks; the sequential composition of parametric morphisms composes the parameter spaces in parallel (Figure 4).

The 2-morphisms in Para capture *reparameterisations* between parametric functions. Importantly, this allows for the **explicit treatment of weight tying**, where a parametric morphism $(P \times P, f)$ can have its weights tied by precomposing with the copy map $\Delta_P : P \to P \times P$.

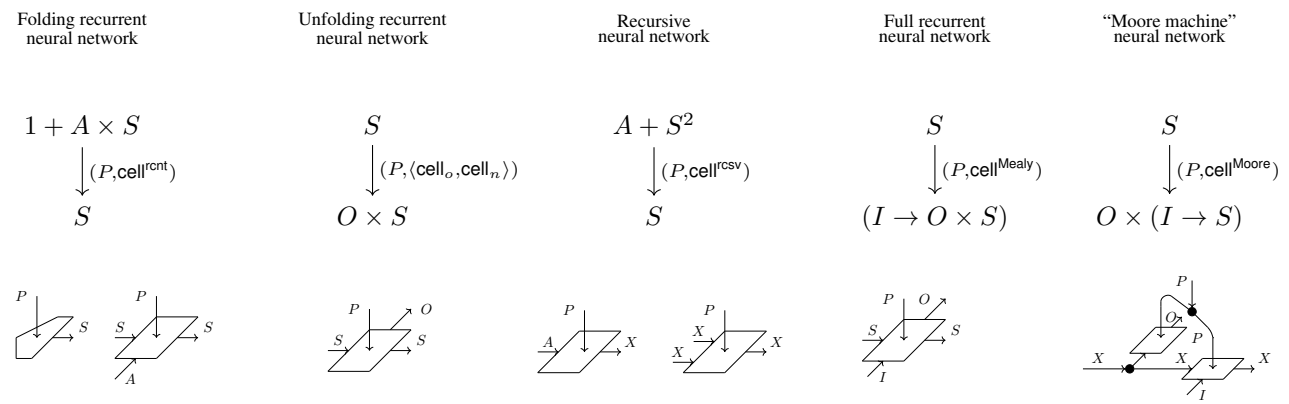
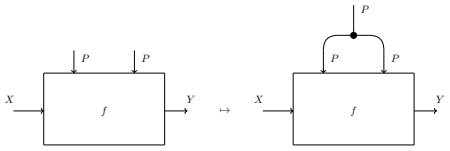


Figure 1. Parametric (co)algebras provide a high-level framework for describing structured computation in neural networks.



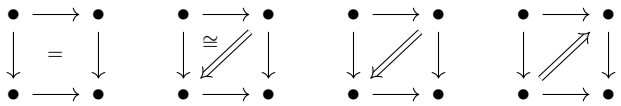
This 2-category[8] is one of the key components in the categorical picture of gradient-based learning (Cruttwell et al., 2022). But we hypothesise that more is true (Appendix I):

It is our position that the 2-category Para and 2-categorical algebra valued in it provide a ***formal theory of neural network architectures**, establish* **formal criteria** *for weight tying correctness and* **inform design** *of new architectures.*

3.2. 2-dimensional Categorical Algebra

2-category theory is markedly richer than 1-category theory.

While diagrams in a 1-category either commute or do not commute, in a 2-category, they serve as a *1-skeleton* to which 2-morphisms *attach*. In *any* 2-category a square may: commute, pseudo-commute, lax-commute, or oplax[9] commute, meaning, respectively, that relevant paths paths are equal, isomorphic, or there is a 2-morphism from one to the other in one direction or the other. The diagrams below present these four options, with 2-morphisms denoted by double arrows.



In the long run, we expect that *all* of these notions will apply to, either explaining or specifying, aspects of neural architecture past, present and future. Focusing on just one of them, the *lax algebras* are sufficient to derive *recursive*, *recurrent*, and similar neural networks from first principles. Notably, morphisms of lax algebras are also expressive enough to capture *1-cocycles*, used to formalise asynchronous neural networks in (Dudzik et al., 2024)—see Appendix H.1.

Interestingly, this story of how an individual recurrent, recursive, etc. neural network cell generates a full recursive, recurrent etc. neural network is a particular 2-categorical analogue to the story of *algebraically free monads* on an endofunctor we briefly mentioned in Remark 2.13.

For all the examples of endofunctors in Section 2.2, there is a monad whose category of algebras $\mathsf{Free}_{\mathsf{Mnd}}(F)$ is equivalent to the category of algebras for the original endofunctor F. We obtain $\mathsf{Free}_{\mathsf{Mnd}}(F)$ by iterating F until it *stabilises*, meaning further application of the endofunctor does not change the composition. Functional programmers may recognise this from the implementation of free monads in Haskell, while formally this is defined using *colimits* (see Appendix B.2). Using this concept, we can define a functor mapping an F-algebra (A, a) to the $\mathsf{Free}_{\mathsf{Mnd}}(F)$-algebra $(A, \varinjlim(a \circ Fa \circ F^2a \circ \cdots \circ F^n a))$, connecting appropriate endofunctor algebras to monad algebras.

But in the 2-dimensional case, we study the relationship between $\mathsf{Lax\text{-}Alg}_{\mathsf{Endo}}(F)$ and $\mathsf{Lax\text{-}Alg}_{\mathsf{Mnd}}(\mathsf{Free}_{\mathsf{Mnd}}(F))$ and need contend not only with generating the 1-dimensional structure map, but also the 2-cells of the lax algebra for a monad.

To reconcile this with concrete applications, we note that we do not need to study general 2-endofunctors and 2-monads on Para. Rather, examples which concern us arise from specific 1-categorical algebras (group action monads, inductive types, etc.), which are *augmented* into 2-monads on Para. As we prove in Theorem G.10, the lax cells of such algebras are actually *comonoids*. The fact that we can duplicate or delete entries in vectors—the essence of tying weights—is

[8] More precisely, the construction $\mathsf{Para}(_)$. See Appendix G.
[9] Often also called *colax*.

the informal face of this comonoid structure.

We can now describe, even if space constraints prevent us from adequate level of detail, the universal properties of recurrent, recursive, and similar models: they are *lax algebras* for *free parametric monads* generated by *parametric endofunctors*! Having lifted the concept of algebra introduced in Part 2 into 2-categories, we can now describe several influential neural networks *fully* (not just their individual layers!) from first principles of functional programming.

4. New Horizons

Our framework gives *the correct definition* of numerous variants of structured networks as universal parametric counterparts of known notions in computer science. This immediately opens up innumerable avenues for research.

Firstly, any results of categorical deep learning as presented here rely on *choosing* the right category to operate in; much like results in geometric deep learning relied on the choice of symmetry group. However, we have seen that monad algebras—which *generalise* equivariance constraints—can be parametric, and *lax*. As a consequence, the kinds of equivariance constraints we can *learn* become more general: we hypothesise neural networks that can learn not merely conservation laws (as in Alet et al. (2021)), but verifiably correct logical argument, or code. This has ramifications for *code synthesis*: we can, for example, specify neural networks that learn only *well-typed functions* by choosing appropriate algebras as their domain and codomain.

This is made possible by our framework's generality: for example, by choosing *polynomial functors* as endofunctors we get access to *containers* (Abbott et al., 2003; Altenkirch et al., 2010), a uniform way to program with and reason about datatypes and polymorphic functions. By combining these insights with recent advances enabling *purely functional* differentiation through inductive and coinductive types (Nunes & Vákár, 2023), we open new vistas for *type-safe* design and implementation of neural networks in functional languages.

One major limitation of geometric deep learning was that it was typically only able to deal with individual neural network layers, owing to its focus on *linear* equivariant functions (see e.g. Maron et al. (2018) for the case of graphs). All nonlinear behaviours can usually be obtained through composition of such layers with nonlinearities, but GDL typically makes no attempt to explain the significance of the choice of nonlinearity—which is known to often be a significant decision (Shazeer, 2020). Within our framework, we can reason about architectural blocks spanning multiple layers—as evidenced by our weight tying examples—and hence we believe CDL should enable us to have a theory of architectures which properly treats nonlinearities.

Our framework also offers a *proactive* path towards *equitable* AI systems. GDL already enables the architectural imposition of protected classes invariance (see Choraria et al. (2021) for example). This deals, at least partially, both with issues of inequity in training data and inequity in algorithms since such an invariant model is, by construction, exclusively capable of inference on the dimensions of latent representation which are orthogonal to protected class.

With CDL, we hope to enable even finer grained control. By way of categorical logic, we hope that CDL will lead us to a new and deeper understanding of the relationship between architecture and logic, in particular clarifying the logics of inductive bias. We hope that our framework will eventually allow us to specify the kinds of arguments the neural networks can use to come to their conclusions. This is a level of expressivity permitting reliable use for assessing bias, fairness in the reasoning done by AI models deployed at scale. We thus believe that this is the right path to AI compliance and safety, and not merely explainable, but verifiable AI.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgments

The authors wish to thank Razvan Pascanu and Yee Whye Teh for reviewing the paper prior to submission.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Abbott, M., Altenkirch, T., and Ghani, N. Categories of containers. In Gordon, A. D. (ed.), *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pp. 23–38. Springer, 2003. ISBN 978-3-540-36576-1. doi: 10.1007/3-540-36576-1_2.

Alet, F., Doblar, D., Zhou, A., Tenenbaum, J., Kawaguchi, K., and Finn, C. Noether networks: meta-learning useful conserved quantities. *Advances in Neural Information Processing Systems*, 34:16384–16397, 2021.

Altenkirch, T., Levy, P., and Staton, S. Higher-order containers. In Ferreira, F., Löwe, B., Mayordomo, E., and Mendes Gomes, L. (eds.), *Programs, Proofs,*

Processes, volume 6158, pp. 11–20. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13961-1 978-3-642-13962-8. doi: 10.1007/978-3-642-13962-8_2. URL http://link.springer.com/10.1007/978-3-642-13962-8_2. Series Title: Lecture Notes in Computer Science.

Alvarez-Picallo, M., Ghica, D., Sprunger, D., and Zanasi, F. Functorial string diagrams for reverse-mode automatic differentiation, 2021.

Atkey, R. and McBride, C. Productive coprogramming with guarded recursion. In *Proceedings of the 18th ACM SIGPLAN international conference on Functional programming*, ICFP '13, pp. 197–208. Association for Computing Machinery, 2013. ISBN 978-1-4503-2326-0. doi: 10.1145/2500365.2500597. URL https://doi.org/10.1145/2500365.2500597.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018. URL http://jmlr.org/papers/v18/17-468.html.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., et al. Jax: composable transformations of python+ numpy programs. 2018.

Bradley, T.-D. Entropy as a topological operad derivation. 23(9):1195, 2021. ISSN 1099-4300. doi: 10.3390/e23091195. URL http://arxiv.org/abs/2107.09581.

Braithwaite, D., Hedges, J., and St Clere Smithe, T. The compositional structure of bayesian inference. In *DROPS-IDN/v2/document/10.4230/LIPIcs.MFCS.2023.24*. Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2023. doi: 10.4230/LIPIcs.MFCS.2023.24. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.MFCS.2023.24.

Brandenburg, M. Large limit sketches and topological space objects. *arXiv preprint arXiv:2106.11115*, 2021.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Capucci, M. and Gavranović, B. Actegories for the working amthematician, 2023. URL http://arxiv.org/abs/2203.16351.

Capucci, M., Gavranović, B., Hedges, J., and Rischel, E. F. Towards foundations of categorical cybernetics. 372:235–248, 2022. ISSN 2075-2180. doi: 10.4204/EPTCS.372.17. URL http://arxiv.org/abs/2105.06332.

Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., and Murphy, K. Machine learning on graphs: A model and comprehensive taxonomy. *The Journal of Machine Learning Research*, 23(1):3840–3903, 2022.

Cho, K. and Jacobs, B. Disintegration and bayesian inversion via string diagrams. 29(7):938–971, 2019. ISSN 0960-1295, 1469-8072. doi: 10.1017/S0960129518000488. URL http://arxiv.org/abs/1709.00322.

Choraria, M., Ferwana, I., Mani, A., and Varshney, L. R. Balancing fairness and robustness via partial invariance. NeurIPS 2021 Workshop on Algorithmic Fairness through the Lens of Causality and Robustness, 2021. URL https://arxiv.org/abs/2112.09346.

Cohen, T. Towards a grounded theory of causation for embodied ai. *arXiv preprint arXiv:2206.13973*, 2022.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In *International conference on machine learning*, pp. 2990–2999. PMLR, 2016.

Cohen, T. S., Geiger, M., Köhler, J., and Welling, M. Spherical cnns. *arXiv preprint arXiv:1801.10130*, 2018.

Cohen, T. S., Geiger, M., and Weiler, M. A general theory of equivariant cnns on homogeneous spaces. *Advances in neural information processing systems*, 32, 2019.

Cruttwell, G. S. H., Gavranović, B., Ghani, N., Wilson, P., and Zanasi, F. Categorical foundations of gradient-based learning. In Sergey, I. (ed.), *Programming Languages and Systems*, Lecture Notes in Computer Science, pp. 1–28. Springer International Publishing, 2022. ISBN 978-3-030-99336-8. doi: 10.1007/978-3-030-99336-8_1.

de Haan, P., Cohen, T. S., and Welling, M. Natural graph networks. *Advances in neural information processing systems*, 33:3636–3646, 2020a.

de Haan, P., Weiler, M., Cohen, T., and Welling, M. Gauge equivariant mesh cnns: Anisotropic convolutions on geometric graphs. *arXiv preprint arXiv:2003.05425*, 2020b.

Dudzik, A. J. and Veličković, P. Graph neural networks are dynamic programmers. *Advances in Neural Information Processing Systems*, 35:20635–20647, 2022.

Dudzik, A. J., von Glehn, T., Pascanu, R., and Veličković, P. Asynchronous algorithmic alignment with cocycles. In Villar, S. and Chamberlain, B. (eds.), *Proceedings of the Second Learning on Graphs Conference*, volume 231 of *Proceedings of Machine Learning Research*, pp. 3:1–3:17. PMLR, 27–30 Nov 2024. URL https://proceedings.mlr.press/v231/dudzik24a.html.

Elliott, C. The simple essence of automatic differentiation. *Proceedings of the ACM on Programming Languages*, 2(ICFP):70:1–70:29, July 2018. doi: 10.1145/3236765. URL https://dl.acm.org/doi/10.1145/3236765.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Fong, B., Spivak, D., and Tuyéras, R. Backprop as functor: a compositional perspective on supervised learning. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '19, pp. 1–13, Vancouver, Canada, June 2021. IEEE Press.

Fuchs, F., Worrall, D., Fischer, V., and Welling, M. Se (3)-transformers: 3d roto-translation equivariant attention networks. *Advances in neural information processing systems*, 33:1970–1981, 2020.

Fukushima, K., Miyake, S., and Ito, T. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (5):826–834, 1983.

Gambino, N. and Kock, J. Polynomial functors and polynomial monads. In *Mathematical proceedings of the cambridge philosophical society*, volume 154, pp. 153–192. Cambridge University Press, 2013.

Gavranović, B. Fundamental Components of Deep Learning: A category-theoretic approach. *arXiv e-prints*, art. arXiv:2403.13001, March 2024. doi: 10.48550/arXiv.2403.13001.

Gavranović, B. Category theory and machine learning, 2020. URL https://github.com/bgavran/Category_Theory_Machine_Learning.

Gavranović, B. Space-time tradeoffs of lenses and optics via higher category theory, 2022. URL http://arxiv.org/abs/2209.09351.

Geiger, M. and Smidt, T. e3nn: Euclidean neural networks, 2022. URL https://arxiv.org/abs/2207.09453.

Ghani, N., Lüth, C., de Marchi, F., and Power, J. Algebras, coalgebras, monads and comonads. 44(1):128–145. ISSN 1571-0661. doi: 10.1016/S1571-0661(04)80905-8. URL https://www.sciencedirect.com/science/article/pii/S1571066104809058.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Godwin, J., Keck, T., Battaglia, P., Bapst, V., Kipf, T., Li, Y., Stachenfeld, K., Veličković, P., and Sanchez-Gonzalez, A. Jraph: A library for graph neural networks in jax., 2020. URL http://github.com/deepmind/jraph.

Guss, W. H. and Salakhutdinov, R. On characterizing the capacity of neural networks using algebraic topology, 2018. URL http://arxiv.org/abs/1802.04443.

Heunen, C., Kammar, O., Staton, S., and Yang, H. A convenient category for higher-order probability theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–12, 2017. doi: 10.1109/LICS.2017.8005137. URL http://arxiv.org/abs/1701.02547.

Jacobs, B. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. ISBN 978-1-316-82318-7. doi: 10.1017/CBO9781316823187. URL https://doi.org/10.1017/CBO9781316823187.

Jacobs, B., Kissinger, A., and Zanasi, F. Causal inference by string diagram surgery. In Bojańczyk, M. and Simpson, A. (eds.), *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pp. 313–329. Springer International Publishing, 2019. ISBN 978-3-030-17127-8. doi: 10.1007/978-3-030-17127-8_18.

Jogl, F., Thiessen, M., and Gärtner, T. Expressivity-preserving gnn simulation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Johnson, N., Yau, D., Johnson, N., and Yau, D. *2-Dimensional Categories*. Oxford University Press. ISBN 978-0-19-887137-8.

Kelly, G. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22:1 – 83, 1980. URL https://api.semanticscholar.org/CorpusID:117985498.

Kelly, G. The basic concepts of enriched category theory. *Reprints in Theory and Applications of Categories [electronic only]*, 2005, 01 2005.

Lack, S. *A 2-categories companion*, pp. 105–191. IMA volumes in mathematics and its applications. Springer, Springer Nature, United States, 2010. ISBN 9781441915238. doi: 10.1007/978-1-4419-1524-5_4.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Leinster, T. *Entropy and Diversity: The Axiomatic Approach*. Cambridge University Press, 2021. doi: 10.1017/9781108963558. URL https://www.cambridge.org/core/books/entropy-and-diversity/496CF94AEA7B33F15904BD4FC8CC2369.

Lewis, M. Compositionality for recursive neural networks. 6(4):709–724, 2019. URL https://collegepublications.co.uk/ifcolog/?00033.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

Mealy, G. H. A method for synthesizing sequential circuits. 34(5):1045–1079, 1955. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1955.tb03788.x. URL https://ieeexplore.ieee.org/document/6771467. Conference Name: The Bell System Technical Journal.

Nguyen, M. and Wu, N. *Folding over Neural Networks*, pp. 129–150. Springer International Publishing, 2022. ISBN 9783031169120. doi: 10.1007/978-3-031-16912-0_5. URL http://dx.doi.org/10.1007/978-3-031-16912-0_5.

Niu, N. and Spivak, D. I. *Polynomial Functors: A Mathematical Theory of Interaction*. arXiv, 2023. URL http://arxiv.org/abs/2312.00990.

Nunes, F. L. and Vákár, M. CHAD for expressive total languages. 33(4):311–426, 2023. ISSN 0960-1295, 1469-8072. doi: 10.1017/S096012952300018X. URL http://arxiv.org/abs/2110.00446.

Olah, C. Neural Networks, Types, and Functional Programming – colah's blog, September 2015. URL https://colah.github.io/posts/2015-09-NN-Types-FP/.

Papillon, M., Sanborn, S., Hajij, M., and Miolane, N. Architectures of topological deep learning: A survey on topological neural networks. *arXiv preprint arXiv:2304.10031*, 2023.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Perrone, P. Markov categories and entropy, 2022. URL http://arxiv.org/abs/2212.11719.

Rutten, J. J. M. M. Universal coalgebra: a theory of systems. 249(1):3–80, 2000. ISSN 0304-3975. doi: 10.1016/S0304-3975(00)00056-6. URL https://www.sciencedirect.com/science/article/pii/S0304397500000566.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642. Association for Computational Linguistics, 2013. URL https://aclanthology.org/D13-1170.

Sonoda, S., Hashimoto, Y., Ishikawa, I., and Ikeda, M. Deep ridgelet transform: Voice with koopman operator proves universality of formal deep networks. *arXiv preprint arXiv:2310.03529*, 2023a.

Sonoda, S., Ishi, H., Ishikawa, I., and Ikeda, M. Joint group invariant functions on data-parameter domain induce universal neural networks. *arXiv preprint arXiv:2310.03530*, 2023b.

Sonoda, S., Ishikawa, I., and Ikeda, M. A unified fourier slice method to derive ridgelet transform for a variety of depth-2 neural networks. *arXiv preprint arXiv:2402.15984*, 2024.

Sturtz, K. Categorical probability theory, 2015. URL http://arxiv.org/abs/1406.6030.

Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Veličković, P. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, 2023.

Vákár, M. and Smeding, T. CHAD: Combinatory homomorphic automatic differentiation. 44(3):20:1–20:49, 2022. ISSN 0164-0925. doi: 10.1145/3527634. URL https://doi.org/10.1145/3527634.

Wang, M. Y. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*, 2019.

Weiler, M., Forré, P., Verlinde, E., and Welling, M. *Equivariant and Coordinate Independent Convolutional Networks*. 2023. URL https://maurice-weiler.gitlab.io/cnn_book/EquivariantAndCoordinateIndependentCNNs.pdf.

Worrall, D. and Welling, M. Deep scale-spaces: Equivariance over scale. *Advances in Neural Information Processing Systems*, 32, 2019.

Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K.-i., and Jegelka, S. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

A. Category Theory Basics

The natural notion of 'sameness' for categories is *equivalence*:

Definition A.1 (Equivalence)**.** An **equivalence** between two categories $\mathcal{C}$ and $\mathcal{D}$, written $\mathcal{C} \xrightarrow{\sim} \mathcal{D}$, consists of a pair of functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$ together with natural isomorphisms (natural transformations where every component has an inverse) $F \circ G \cong 1_{\mathcal{D}}$ and $G \circ F \cong 1_{\mathcal{C}}$.

Definition A.2 (Initial object)**.** An object I in a category $\mathcal{C}$ is called **initial** if for every $X \in \mathcal{C}$ it naturally holds that $\mathcal{C}(I, X) \cong 1$, meaning that there is only one map of type $I \to X$.

Definition A.3 (Terminal object)**.** An object T in a category $\mathcal{C}$ is called **terminal** if for every $X \in \mathcal{C}$ it naturally holds that $\mathcal{C}(X, T) \cong 1$, meaning that there is only one map of type $X \to T$.

Definition A.4 (Limit)**.** For categories $\mathcal{J}$ and $\mathcal{C}$, a **diagram of shape** $\mathcal{J}$ in $\mathcal{C}$ is a functor $D : \mathcal{J} \to \mathcal{C}$. A **cone** to a diagram D consists of an object $C \in \mathcal{C}$ and a natural transformation from a functor constant at C to the functor D, i.e. a family of morphisms $c_j : C \to D(j)$ for each object $j \in \mathcal{J}$, such that for any $f : i \to j$ in $\mathcal{C}$ the following diagram commutes:

$$\begin{tikzcd} C \arrow[r, "c_i"] \arrow[dr, "c_j"'] & D(i) \arrow[d, "D(f)"] \\ & D(j) \end{tikzcd}$$

A morphism of cones $\theta : (C, c_j) \to (C', c'_j)$ is a morphism θ in $\mathcal{C}$ making each diagram

$$\begin{tikzcd} C \arrow[r, "\theta"] \arrow[dr, "c_j"'] & C' \arrow[d, "c'_j"] \\ & D(j) \end{tikzcd}$$

commute. The **limit** of a diagram D, written $\varprojlim D$, is the terminal object in the category $\mathsf{Cone}(D)$ of cones to D and morphisms between them.

Definition A.5 (Colimit)**.** The **colimit** $\varinjlim D$ of a diagram $D : \mathcal{J}^{op} \to \mathcal{C}$ is the initial object in the category $\mathsf{Cocone}(D)$ of cocones to D, where a cocone $(C, c_j : D(j) \to C)$ has the dual property of a cone (above) with the morphisms reversed. **Small** (respectively κ-directed, connected, ...) colimits are colimits for which the indexing category $\mathcal{J}$ is small (respectively κ-directed, connected, ...).

Example A.6. A terminal object in $\mathcal{C}$ is a limit of the unique diagram from the empty category to $\mathcal{C}$. Similarly an initial object is an example of a colimit.

B. 1-Categorical Algebra

Definition B.1 (Monad coherence diagrams)**.** A triple (M, η, μ) of:

- an endofunctor $M : \mathcal{C} \to \mathcal{C}$;
- a natural transformations $\eta : \mathrm{id}_{\mathcal{C}} \Rightarrow M$; and
- a natural transformation $\mu : M \circ M \Rightarrow M$

constitute a **monad** if the following diagrams commute:

$$\begin{tikzcd} M \arrow[r, "\eta"] \arrow[dr, equal] & M \circ M \arrow[d, "\mu"] \\ & M \end{tikzcd} \qquad \begin{tikzcd} M \circ M \circ M \arrow[r, "M \circ \mu"] \arrow[d, "\mu"'] & M \circ M \arrow[d, "\mu"] \\ M \circ M \arrow[r, "\mu"'] & M \end{tikzcd}$$

Definition B.2 (Coalgebra for an endofunctor)**.** Let $\mathcal{C}$ be a category and F an endofunctor on $\mathcal{C}$. A coalgebra for F is a pair (A, a) where A is an object of $\mathcal{C}$ and $a : A \to F(A)$ is a morphism of $\mathcal{C}$.

B.1. Well-pointed Endofunctors and Algebraically Free Monads

In this subsection we'll relate endofunctors and their algebras to monads and their algebras by way of *well-pointed endofunctors* and the transfinite construction for an *algebraically free monad* on such an endofunctor.

Definition B.3. A **pointed endofunctor** (F, σ) comprises an endofunctor $F : \mathcal{C} \to \mathcal{C}$, and a natural transformation $\sigma : \mathrm{id}_{\mathcal{C}} \Rightarrow F$. A pointed endofunctor is said to be **well-pointed** if the whiskering of F and σ either from the left or the right gives the same result, or in other words, if the natural transformations $\sigma \circ F$ and $F \circ \sigma$ (which are of type $F \Rightarrow F \circ F$) are equal.

An **algebra for a pointed endofunctor** (F, σ) is an algebra (A, a) for the endofunctor F for which the following diagram commutes:

$$\begin{tikzcd} A \arrow[r, "\sigma_A"] \arrow[dr, "\mathrm{id}_A"'] & FA \arrow[d, "a"] \\ & A \end{tikzcd}$$

Remark B.4. As well-pointedness is a property, and not structure, algebras for well-pointed endofunctors are just algebras for the pointed endofunctors. Morphisms of algebras for pointed endofunctors are morphisms of algebras for the underlying endofunctor.

Example B.5 (Monads and pointed endofunctors, idempotent monads and well-pointed endofunctors)**.** The unit and underlying endofunctor of a monad constitute a pointed endofunctor. If moreover that monad is idempotent, then that pointed endofunctor is well-pointed.

Definition B.6. For a given endofunctor F, its algebras and algebra homomorphisms form a category we denote by $\mathsf{Alg}_{\mathsf{Endo}}(F)$. For (F, σ) a (well)-pointed endofunctor we denote by $\mathsf{Alg}_{\mathsf{Pendo}}(F)$ the category of algebras for F and homomorphisms thereof. For (F, μ, η) a monad its algebras and homomorphisms thereof form a category we denote by $\mathsf{Alg}_{\mathsf{Mnd}}(F)$.

Lemma B.7. *Suppose F is an endofunctor on a category $\mathcal{C}$ with coproducts. Then there is an equivalence of categories* $\mathsf{Alg}_{\mathsf{Endo}}(F) \xrightarrow{\sim} \mathsf{Alg}_{\mathsf{Pendo}}(F + id_{\mathcal{C}})$.

Definition B.8. Given an endofunctor $F : \mathcal{C} \to \mathcal{C}$, an **algebraically free monad** on F is a monad $\mathsf{Free}_{\mathsf{Mnd}}(F)$ together with an equivalence of categories $\mathsf{Alg}_{\mathsf{Endo}}(F) \xrightarrow{\sim} \mathsf{Alg}_{\mathsf{Mnd}}(\mathsf{Free}_{\mathsf{Mnd}}(F))$ which preserves the respective functors to $\mathcal{C}$ that forget the algebraic structure.

B.2. Kelly's Unified Transfinite Construction

The existence theorem for algebraically free monads is Kelly's unified transfinite construction (Kelly, 1980).

Definition B.9 (Reflective subcategory)**.** A full subcategory $\mathcal{D}$ of a category $\mathcal{C}$ is **reflective** if the inclusion functor $F : \mathcal{D} \to \mathcal{C}$ admits a left adjoint $G : \mathcal{C} \to \mathcal{D}$. A reflective subcategory of a presheaf category is called a **locally presentable category**.

Example B.10 (Categories are reflective in graphs)**.** The category of small categories is a reflective subcategory of the category of graphs, which is itself a presheaf category.

Example B.11 (Ubiquity of local presentability)**.** Nearly every category often encountered in practice is a locally presentable category. The categories of monoids, groups, rings, vector spaces, and modules are locally presentable. As are the categories of topological spaces, manifolds, metric spaces, and uniform spaces. While its beyond the scope of this document to expound too much upon it, local presentability is a particularly powerful notion of what it means for objects and morphisms to be of things defined by equalities of set-sized expressions.

Definition B.12 (Accessible category and accessible functor)**.** For an ordinal κ, a κ-**accessible category** $\mathcal{C}$ is a category such that:

- $\mathcal{C}$ has κ-directed colimits; and
- there is a set of κ-compact objects which generates $\mathcal{C}$ under κ-directed colimits.

A κ-**accessible functor** $F : \mathcal{C} \to \mathcal{D}$ is an functor between κ-accessible functor which preserves κ-filtered colimits.

Remark B.13. The accessibility of a functor can be thought of as an upper-bound on the arity of the operations which it abstracts. For example finite sums of finite sums are again finite sums.

Definition B.14. Let $\mathcal{C}$ be a κ-accessible locally presentable category and $F : \mathcal{C} \to \mathcal{C}$ a pointed κ-accessible endofunctor. Let F^κ be the κ-directed colimit of the diagram

$$F^0 \to F^1 \to F^2 \to \cdots \to F^\kappa$$

where $F^0 = \mathrm{id}_{\mathcal{C}}$ and $F^{\alpha+1} = F \circ F^\alpha$ for $\alpha < \kappa$ and $F^\alpha = \varinjlim_{\beta<\alpha} F^\beta$ for α a limit ordinal.

Lemma B.15. *For $\mathcal{C}$ and F as above, F^κ is a monad. The unit is the canonical inclusion of $\mathrm{id}_{\mathcal{C}}$ into the colimit F^κ and the multiplication comes from the preservation of κ-filtered colimits by F.*

Theorem B.16. *Assume the hypotheses of Lemma B.15 with κ the ordinal in those hypotheses. Then $\mathsf{Alg}_{\mathsf{Pendo}}(F)$ is equivalent to $\mathsf{Alg}_{\mathsf{Mnd}}(F^\kappa)$ - i.e. F^κ is an algebraically free monad for F.*

Remark B.17. For the endofunctors F we study here, $\mathsf{Free}(F) \xrightarrow{\sim} (F+\mathrm{id})^\omega$ is the underlying endofunctor of an algebraically free monad for F, $\mathsf{Free}(F)$.

The above formula can be related to the explicit formula for computing free monads, which has a dual formula in the case of cofree comonads (Ghani et al.).

Proposition B.18 ((Co)free (co)monads, explicitly)**.** *Let $F : \mathcal{C} \to \mathcal{C}$ be an endofunctor. Then $\mathsf{Free}_{\mathsf{Mnd}}(F)$, the free monad on F is given by $\mathsf{Free}_{\mathsf{Mnd}}(F)(Z) = \mathsf{Fix}(X \mapsto F(X) + Z)$. Dually, we compute $\mathsf{Cofree}_{\mathsf{Cmnd}}(F)$, the cofree comonad on F as $\mathsf{Cofree}_{\mathsf{Cmnd}}(F)(Z) = \mathsf{Fix}(X \mapsto F(X) \times Z)$.*

Example B.19 (Free monad on $1 + A \times -$)**.** Free monad on the endofunctor $1 + A \times -$ is $\mathsf{List}_{-+1}(A) : \mathsf{Set} \to \mathsf{Set}$, the endofunctor mapping an object Z to the set $\mathsf{List}_{Z+1}(A)$ of lists of elements of type A whose last element is not $[]$, i.e. an element of type 1, but instead an element of type $Z + 1$. That is, these are lists which end potentially with an element of Z.

Example B.20 (Free monad on $A+(-)^2$)**.** The free monad on the endofunctor $A+(-)^2$ is given by $\mathsf{Tree}(A+-) : \mathsf{Set} \to \mathsf{Set}$, mapping a set Z to the set of trees with $A + Z$ labelled leaves.

Example B.21 (Cofree comonad on $O \times -$)**.** The cofree comonad on the endofunctor $O \times -$ is $\mathsf{Stream}(O \times -)$, mapping an object Z to the set of streams whose outputs are of type $O \times Z$.

Example B.22 (Cofree comonad on $(I \to O \times -)$)**.** The cofree comonad of the endofunctor $(I \to O \times -)$ is $\mathsf{Fix}(X \mapsto (I \to O \times X) \times -)$, mapping a set Z to a set of hybrids of Moore and Mealy machines, outputting an additional element of Z at each step which does not depend on I.

C. Additional Geometric Deep Learning Examples

To further illustrate the power of categorical deep learning as a framework that subsumes geometric deep learning (Bronstein et al., 2021), as well as make the reader more comfortable in manipulating monad algebras and their homomorphisms, we provide three additional examples deriving equivariance constraints of established geometric deep learning architectures, leveraging the framework of CDL.

All of these examples should be familiar to Geometric DL practitioners, and are covered in detail by prior papers (Maron et al., 2018; Cohen et al., 2018; Thomas et al., 2018; Cohen & Welling, 2016), hence we believe that relegating their exact derivations to appendices is appropriate in our work.

Before we begin, we recall the core template of our work: that we represent neural networks $f : A \to B$ as monad algebra homomorphisms between two algebras (A, a) and (B, b), for a monad (M, η, μ):

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

and that geometric deep learning can be recovered by making our monad be the *group action monad*; $M(X) = G \times X$.

C.1. *Permutation*-equivariant Learning on *Graphs*

leading to graph neural networks (Veličković, 2023).

$$\begin{tikzcd}
\Sigma_n \times \mathbb{R}^n \times \mathbb{R}^{n\times n} \arrow[r, "\Sigma_n \times f"] \arrow[d, "P_{X,A}"'] & \Sigma_n \times \mathbb{R}^n \arrow[d, "P_X"] \\
\mathbb{R}^n \times \mathbb{R}^{n\times n} \arrow[r, "f"'] & \mathbb{R}^n
\end{tikzcd}$$

In this case:

- The group $G = \Sigma_n$ is the permutation group of n elements,
- The carrier object for the algebras includes (scalar) node features $\mathbb{R}^n$ and, potentially, adjacency matrices $\mathbb{R}^{n\times n}$,
- The structure map for the first algebra, $P_{X,A} : \Sigma_n \times \mathbb{R}^n \times \mathbb{R}^{n\times n} \to \mathbb{R}^n \times \mathbb{R}^{n\times n}$, executes the permutation: $P_{X,A}(\sigma, \mathbf{X}, \mathbf{A}) = (\mathbf{P}(\sigma)\mathbf{X}, \mathbf{P}(\sigma)\mathbf{A}\mathbf{P}(\sigma)^\top)$, where $\mathbf{P}(\sigma)$ is the permutation matrix specified by σ.
- The structure map for the second algebra, P_X, still executes the permutation, but only over the node features. This reduction is not strictly necessary, but is often the standard when designing graph neural networks—as they are often assumed to not modify their underlying computational graph. We deliberately assume this reduction here to illustrate how our framework can handle neural networks transitioning across different algebras.

C.2. *Rotation*-equivariant Learning on *Spheres*

leading to the first layer of spherical CNNs (Cohen et al., 2018).

$$\begin{tikzcd}
\mathrm{SO}(3) \times (S^2 \to \mathbb{R}) \arrow[r, "\mathrm{SO}(3)\times f"] \arrow[d, "\rho_{S^2}"'] & \mathrm{SO}(3) \times (\mathrm{SO}(3) \to \mathbb{R}) \arrow[d, "\rho_{\mathrm{SO}(3)}"] \\
S^2 \to \mathbb{R} \arrow[r, "f"'] & \mathrm{SO}(3) \to \mathbb{R}
\end{tikzcd}$$

In this case:

- The group $G = \mathrm{SO}(3)$ is the special orthogonal group of 3D rotations.
- The carrier object for the first algebra is $S^2 \to \mathbb{R}$; (scalar) data defined over the sphere. Note that in practice, this will usually be discretised, so we will be able to represent it using matrices.
- The structure map of the first algebra, $\rho_{S^2} : \mathrm{SO}(3) \times (S^2 \to \mathbb{R}) \to (S^2 \to \mathbb{R})$, executes a 3D rotation on the spherical data, as follows: $\rho_{S^2}((\alpha, \beta, \gamma), \psi) = \phi$, such that $\phi(\mathbf{x}) = \psi(\mathbf{R}(\alpha, \beta, \gamma)^{-1}\mathbf{x})$, where $\mathbf{R}(\alpha, \beta, \gamma)$ is the rotation matrix specified by the ZYZ-Euler angles $(\alpha, \beta, \gamma) \in \mathrm{SO}(3)$. This (inverse) rotation is applied to points $\mathbf{x} \in S^2$ on the sphere.
- The carrier object for the second algebra is $\mathrm{SO}(3) \to \mathbb{R}$; (scalar) data defined over rotation matrices. Once again, this will usually be discretised in practice.
- The structure map of the second algebra, $\rho_{\mathrm{SO}(3)} : \mathrm{SO}(3) \times (\mathrm{SO}(3) \to \mathbb{R}) \to (\mathrm{SO}(3) \to \mathbb{R})$ now executes a 3D rotation over the rotation-matrix data, as follows: $\rho_{\mathrm{SO}(3)}((\alpha, \beta, \gamma), \psi) = \phi$ such that $\phi(\mathbf{R}(\alpha', \beta', \gamma')) = \psi(\mathbf{R}(\alpha, \beta, \gamma)^{-1}\mathbf{R}(\alpha', \beta', \gamma'))$.

C.3. G-equivariant Learning on G

leading to G-CNNs (Cohen & Welling, 2016), as well as the subsequent layers of spherical CNNs (Cohen et al., 2018).

$$\begin{tikzcd} G \times (G \to \mathbb{R}) \arrow[r, "G \times f"] \arrow[d, "A_G"'] & G \times (G \to \mathbb{R}) \arrow[d, "A_G"] \\ G \to \mathbb{R} \arrow[r, "f"'] & G \to \mathbb{R} \end{tikzcd}$$

In this case:

- The group G is also the domain of the carrier objects ($G \to \mathbb{R}$).
- Both algebras' structure map follows the execution of the regular representation of G, $A_G : G \times (G \to \mathbb{R}) \to (G \to \mathbb{R})$, by composition, as follows: $A_G(g, \psi)(h) = \psi(g^{-1}h)$.

D. Lawvere Theories and Syntax

de Haan et al. (2020a) expanded the theory of equivariant layers in neural networks using the abstraction of natural transformations of functors. In this section, we will explain how to understand morphisms of monad algebras in the same terms.

Indeed, this comparison is crucial to understanding the *syntax* of monads, in addition to the *semantics* given by their category of algebras.

Definition D.1. If $\mathcal{C}$ is a category, a **presheaf on** $\mathcal{C}$ is a functor $\mathcal{C}^{op} \to \mathsf{Set}$. The category whose objects are presheaves and morphisms are natural transformations is denoted $\mathsf{Psh}(\mathcal{C})$.

Fix a monad T on Set and let $\mathcal{C}_T$ denote its category of algebras and algebra homomorphisms.

Given an algebra $A \in \mathcal{C}_T$, the easiest way to interpret A as a functor is via the *Yoneda embedding* $\mathcal{C}_T \to \mathsf{Psh}(\mathcal{C}_T)$, which identifies A with the presheaf $[-, A]$. It is a standard result that the Yoneda functor is fully faithful, which means that we can identify morphisms as algebras with morphisms as presheaves.

Furthermore, these presheaves have a special property. If $\varinjlim j$ is a small colimit in $\mathcal{C}_T$, then $[\varinjlim j, A] = \varprojlim [j, A]$, essentially by the definition of limits and colimits.

Definition D.2. If $\mathcal{C}$ is a category and J is a class of small colimits in $\mathcal{C}$, then a J**-continuous presheaf on** $\mathcal{C}$ is a presheaf $F \in \mathsf{Psh}(\mathcal{C})$ satisfying $F(\varinjlim j) = \varprojlim F(j)$ for all $j \in J$. The corresponding full subcategory of $\mathsf{Psh}(\mathcal{C})$ is denoted $\mathsf{CPsh}_J(\mathcal{C})$, or simply $\mathsf{CPsh}(\mathcal{C})$ in the case that J is the class of all small colimits. We will also use $\coprod$ to refer to the class of all coproducts, $+$ to refer to the class of binary coproducts, and $\emptyset$ to refer to the empty class.

It turns out that $\mathcal{C}_T$ has a nice property: it is a *strongly compact* category, meaning that every continuous presheaf is representable as above by an object of $\mathcal{C}_T$. In other words, we have the identification $\mathcal{C}_T = \mathsf{CPsh}(\mathcal{C}_T)$.

However, it is usually impractical to work with the entire category $\mathcal{C}_T$. When possible, we want to reason in terms of a more tractable subcategory. These will provide us with workable *syntax* for our monad. Following (Brandenburg, 2021), we make use of Ehresmann's concept of a "colimit sketch": a category equipped with a restricted class of colimits. Rather than giving the general definition, we will describe a few special cases of prime interest.

Definition D.3. The **Kleisli category** $\mathcal{K}_T$ is the full subcategory of $\mathcal{C}_T$ on the free algebras. The **finitary Kleisli category** $\mathcal{K}_T^{\mathbb{N}}$ is the full subcategory on the free algebras of the form TS for S a finite set. And the **unary Kleisli category** $\mathcal{K}_T^1$ is the full one-object subcategory on the free algebra $T1$.

We have the following sequence of nested full subcategories:

$$\mathcal{C}_T \supset \mathcal{K}_T \supset \mathcal{K}_T^{\mathbb{N}} \supset \mathcal{K}_T^1$$

By composing with these inclusions, we get a sequence of functors between presheaf categories:

$$\mathsf{Psh}(\mathcal{C}_T) \to \mathsf{Psh}(\mathcal{K}_T) \to \mathsf{Psh}(\mathcal{K}_T^{\mathbb{N}}) \to \mathsf{Psh}(\mathcal{K}_T^{1})$$

By restricting our class of colimits, we can restrict this to the corresponding continuous presheaf categories:

$$\mathcal{C}_T \cong \mathsf{CPsh}(\mathcal{C}_T) \to \mathsf{CPsh}_{\amalg}(\mathcal{K}_T) \to \mathsf{CPsh}_{+}(\mathcal{K}_T^{\mathbb{N}}) \to \mathsf{CPsh}_{\emptyset}(\mathcal{K}_T^{1})$$

It turns out that in many cases, these arrows are equivalences of categories.

Definition D.4. In the case that $\mathcal{C}_T \to \mathsf{CPsh}_{\amalg}(\mathcal{K}_T)$ is an equivalence, we say that $\mathcal{K}_T^{op}$ is the **infinitary Lawvere theory** for T.

Remark D.5. In fact, all monads have an infinitary Lawvere theory. For a proof, see (Brandenburg, 2021).

Definition D.6. In the case that $\mathcal{C}_T \to \mathsf{CPsh}_{+}(\mathcal{K}_T^{\mathbb{N}})$ is an equivalence, we say that $(\mathcal{K}_T^{\mathbb{N}})^{op}$ is the **Lawvere theory** for T, and T is a **finitary monad**.

Example D.7. If T is the monad sending a set S to the free commutative semiring on S, then $\mathcal{C}_T$ is the category of commutative semirings. Since the axioms for commutative semirings consist of equations with a finite number of variables, T has a Lawvere theory $(\mathcal{K}_T^{\mathbb{N}})^{op}$.

It is well known that this is the category of "finite polynomials", see e.g. (Gambino & Kock, 2013). This connection was observed in (Dudzik & Veličković, 2022) to relate message passing in Graph Neural Networks to polynomial functors.

In fact, we could further restrict this theory to just the four objects $\{T0, T1, T2, T3\}$, because we can fully axiomatise commutative semirings with only three variables, e.g. $a(b + c) = ab + ac$.

Example D.8 (Monoids)**.** If $\mathcal{C}_T \to \mathsf{CPsh}_{\emptyset}(\mathcal{K}_T^{1})$ is an equivalence, then in fact $\mathcal{C}_T \cong \mathsf{Psh}(\mathcal{K}_T^{1})$, which is the category of M-sets for the monoid $M = (\mathcal{K}_T^{1})^{op}$. So we can see that "unary Lawvere theories" exactly correspond to monads of the form $M \times -$, where M is a monoid.

Example D.9 (Suplattices)**.** Not all monads are finitary; that is, not all monads have an associated Lawvere theory.

For an example of a non-finitary monad, let $\mathcal{P} : \mathsf{Set} \to \mathsf{Set}$ be the covariant powerset functor. That is, $\mathcal{P}(S)$ is the set of all subsets of S, and if $f : S \to T$ is a function, then $\mathcal{P}(f) : \mathcal{P}(S) \to \mathcal{P}(T)$ is defined by $\mathcal{P}(f)(A) := \{f(a) \mid a \in A\}$.

We equip $\mathcal{P}$ with the monad structure given by the unit $1 \to \mathcal{P}$ sending $s \in S$ to $\{s\} \in \mathcal{P}(S)$ and the composition $\mathcal{P}^2 \to \mathcal{P}$ sending $\mathcal{A} \subset \mathcal{P}(S)$ to $\bigcup_{A \in \mathcal{A}} A$.

The category of $\mathcal{P}$-algebras $\mathcal{C}_{\mathcal{P}}$ is the category of **suplattices**, which is the category of posets with all least upper bounds, and morphisms preserving them. Equivalently, a suplattice is given by a set L together with a join map $\bigvee \mathcal{P}(L) \to L$ satisfying unit and composition axioms.

Implementing $\bigvee$ requires implementing operations $L^{\kappa} \to L$ for arbitrarily large cardinal numbers κ, so we can see intuitively that suplattices are not described by a finitary theory.

Note that the Kleisli category $\mathcal{K}_{\mathcal{P}}$ is equivalent to the category of sets and relations.

Example D.10 (Semilattices)**.** While $\mathcal{P}$ above isn't finitary, it has a finitary counterpart $\mathcal{P}_{fin}$, the functor that takes each set to its *finite* subsets.

Algebras for $\mathcal{P}_{fin}$ are sometimes called **join-semilattices**, partial orders where every finite subset has a least upper bound. It is a nice exercise to show that a join-semilattice is equivalently a commutative idempotent monoid.

E. Monoidal Categories and Actegories

Monoidal categories and actegories are the 'categorified' version of monoids and actions.

Definition E.1 (Strict monoidal category, (Johnson et al., Def. 1.2.1))**.** Let $\mathcal{M}$ be a category. We call $\mathcal{M}$ a **strict monoidal category** if it is equipped with the following data a functor $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ called *the monoidal product*; an object $I \in \mathcal{M}$ called **the monoidal unit**, such that

- $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ for all $A, B, C \in \mathcal{M}$;
- $A \otimes I = A = I \otimes A$ for all $A \in \mathcal{M}$;
- $f \otimes (g \otimes h) = (f \otimes g) \otimes h$ for all $f, g, h \in \mathcal{M}$;
- $\text{id} \otimes f = f = f \otimes \text{id}$ for all $f \in \mathcal{M}$.

Definition E.2 (Actegories, see (Capucci & Gavranović, 2023))**.** Let $(\mathcal{M}, \otimes, I, \alpha, \lambda, \rho)$ be a monoidal category. An $\mathcal{M}$-**actegory** $\mathcal{C}$ is a category $\mathcal{C}$ together with a functor $\blacktriangleright : \mathcal{M} \times \mathcal{C} \to \mathcal{C}$ together with natural isomorphisms $\eta_X : I \blacktriangleright X \cong X$ and $\mu_{M,N} : (M \otimes N) \blacktriangleright X \cong M \blacktriangleright (N \blacktriangleright X)$, such that:

- **Pentagonator.** For all $M, N, P \in \mathcal{M}$ and $C \in \mathcal{C}$ the following diagram commutes.

$$\begin{tikzcd}
(M \otimes N) \otimes P \blacktriangleright C \arrow[r, "\mu_{M\otimes N,P,C}"] \arrow[dd, "\alpha_{M,N,P} \blacktriangleright C"'] & M \otimes N \blacktriangleright P \blacktriangleright C \arrow[dr, "\mu_{M,N,P\blacktriangleright C}"] & \\
& & M \blacktriangleright N \blacktriangleright P \blacktriangleright C \\
M \otimes (N \otimes P) \blacktriangleright C \arrow[r, "\mu_{C,M\otimes N,P}"'] & M \blacktriangleright N \otimes P \blacktriangleright C \arrow[ur, "M \blacktriangleright \mu_{N,P,C}"'] &
\end{tikzcd} \tag{7}$$

- **Left and right unitors.** For all $C \in \mathcal{C}$ and $M \in \mathcal{M}$ The diagrams below commute.

$$\begin{tikzcd}
I \otimes M \blacktriangleright C \arrow[d, "\mu_{I,M,C}"'] \arrow[dr, "\lambda_M \blacktriangleright C"] & \\
I \blacktriangleright M \blacktriangleright C \arrow[r, "\eta_{M\blacktriangleright C}"'] & M \blacktriangleright C
\end{tikzcd}
\qquad
\begin{tikzcd}
(M \otimes I) \blacktriangleright C \arrow[d, "\mu_{M,I,C}"] \arrow[dl, "\rho_M \blacktriangleright C"'] & \\
M \blacktriangleright C & M \blacktriangleright I \blacktriangleright C \arrow[l, "M \blacktriangleright \eta_C"]
\end{tikzcd} \tag{8}$$

Remark E.3. Just as one may assume a monoidal category to be strict, via MacLane's coherence theorem, we may assume actegories are strict as well (Capucci & Gavranović, 2023, Remark 3.4). That is to say, we may assume for an $\mathcal{M}$ actegory $(\mathcal{C}, \blacktriangleright)$ that:

- The unitor η_X is an equality, i.e. that $I \blacktriangleright (_) = \text{id}_{\mathcal{C}}$ are equal as functors of type $\mathcal{C} \to \mathcal{C}$; and
- The multiplicator $\mu_{M,N}$ is an equality, i.e. that $((_) \otimes (_) \blacktriangleright (_) = (_) \blacktriangleright (_) \blacktriangleright (_)$ are equal as functors of type $\mathcal{M} \times \mathcal{M} \times \mathcal{C} \to \mathcal{C}$

We will call such a structure a **strict actegory**. We will use these later on to simplify exposition and some theorems.

Example E.4 (Monoidal action)**.** Any monoidal category gives rise to a self-action.

Example E.5 (Families actegories)**.** Any category $\mathcal{C}$ with coproducts has an action $\blacktriangleright : \mathsf{Set} \times \mathcal{C} \to \mathcal{C}$ which maps (X, A) to the coproduct of $|X|$ copies of A.

E.1. Morphisms of Actegories

Definition E.6 (Actegorical strong monad)**.** Let $(\mathcal{C}, \blacktriangleright)$ be a $\mathcal{M}$-actegory. A monad (T, μ, η) on $\mathcal{C}$ is called **strong**[10] if it is equipped with a natural transformation $\sigma_{P,A} : P \blacktriangleright T(A) \to T(P \blacktriangleright A)$, called **strength** such that diagrams in Definition E.9 commute.

Example E.7. All monads of the form $A \times - : \mathsf{Set} \to \mathsf{Set}$ are strong for the actegory $(\mathsf{Set}, \times)$, for any monoid A. This includes $G \times - : \mathsf{Set} \to \mathsf{Set}$ from Example 2.2.[11]

[10] Another name for this is a $\mathcal{M}$-linear morphism, used in (Capucci & Gavranović, 2023)

[11] The strength is in fact a natural isomorphism.

Definition E.8 (Actegorical strong endofunctor)**.** Let $(\mathcal{C}, \blacktriangleright)$ be a $\mathcal{M}$-actegory. An endofunctor F on $\mathcal{C}$ is called **strong** if it is equipped with a natural transformation $\sigma_{P,A} : P \blacktriangleright F(A) \to F(P \blacktriangleright A)$, called a **strength** such that diagrams **AS1** and **AS2** in Definition E.9 commute.

Definition E.9 (Actegorical strong monad coherence diagrams)**.** A monad (T, μ, η) on the category $\mathcal{C}$ of a $\mathcal{M}$-actegory $(\mathcal{C}, \blacktriangleright)$ is called **strong** if it is equipped with a natural transformation $\sigma_{P,A} : P \blacktriangleright T(A) \to T(P \blacktriangleright A)$, called **strength** making the diagrams below commute:

AS1: Compatibility of strength and the monoidal unit.

$$\begin{tikzcd} I \blacktriangleright T(A) \arrow[r, "\sigma_{I,A}"] \arrow[dr, "\lambda_{T(A)}"'] & T(I \blacktriangleright A) \arrow[d, "T(\lambda_A)"] \\ & T(A) \end{tikzcd}$$

AS2: Compatibility of strength and actegory multiplicator.

$$\begin{tikzcd} (P \otimes Q) \blacktriangleright T(X) \arrow[r, "\alpha_{P,Q,T(X)}"] \arrow[dd, "\sigma_{P\otimes Q,X}"'] & P \blacktriangleright (Q \blacktriangleright T(X)) \arrow[d, "P \blacktriangleright \sigma_{Q,X}"] \\ & P \blacktriangleright T(Q \blacktriangleright X) \arrow[d, "\sigma_{P,Q\blacktriangleright X}"] \\ T((P \otimes Q) \blacktriangleright X) \arrow[r, "T(\alpha_{P,Q,X})"'] & T(P \blacktriangleright (Q \blacktriangleright X)) \end{tikzcd}$$

AS3: Compat. of strength and monad multiplication.

$$\begin{tikzcd} A \blacktriangleright T(T(X)) \arrow[r, "A \blacktriangleright \mu_X"] \arrow[d, "\sigma_{A,T(X)}"'] & A \blacktriangleright T(X) \arrow[dd, "\sigma_{A,X}"] \\ T(A \blacktriangleright T(X)) \arrow[d, "T(\sigma_{A,T(X)})"'] & \\ T(T(A \blacktriangleright X)) \arrow[r, "\mu_{A\blacktriangleright X}"'] & T(A \blacktriangleright X) \end{tikzcd}$$

AS4: Compatibility of strength and monad unit.

$$\begin{tikzcd} & M \blacktriangleright X \arrow[dl, "M \blacktriangleright \eta_X"'] \arrow[dr, "\eta_{M\blacktriangleright X}"] & \\ M \blacktriangleright T(X) \arrow[rr, "\sigma_{M,X}"'] & & T(M \blacktriangleright X) \end{tikzcd}$$

Example E.10. The examples of endofunctors from Examples 2.9 to 2.11, H.4 and H.7 are all strong endofunctors, and their appropriate free monads are strong too. See Appendix (Example E.11)

Example E.11. Below we present a list the data of *strengths* $\sigma_{P,X}$ of some relevant endofunctors and (on the left) and their corresponding free monads (on the right).

Given the endofunctor $1 + A \times - : \mathsf{Set} \to \mathsf{Set}$, its strength $\sigma_{P,X} : P \times (1 + A \times X) \to 1 + A \times P \times X$ is given by:

$$\sigma_{P,X}(p, \text{inl}(\bullet)) = \text{inl}(\bullet)$$
$$\sigma_{P,X}(p, \text{inr}(a, x)) = \text{inr}(a, p, x)$$

Given the free monad $\mathsf{List}_{-+1}(A) : \mathsf{Set} \to \mathsf{Set}$, its strength $\sigma_{P,Z} : P \times \mathsf{List}_{Z+1}(A) \to \mathsf{List}_{P\times Z+1}(A)$ is given by:

$$\sigma_{P,Z}(p, \mathsf{Nil}) = \mathsf{Nil}$$
$$\sigma_{P,Z}(p, z) = (p, z)$$
$$\sigma_{P,Z}(p, \mathsf{Cons}(a, as)) = \mathsf{Cons}(a, \sigma_{P,Z}(p, as))$$

Given the endofunctor $A + (-)^2$, its strength $\sigma_{P,X} : P \times (A + X^2) \to A + (P \times X)^2$ is given by:

$$\sigma_{P,X}(p, \text{inl}(a)) = \text{inl}(a)$$
$$\sigma_{P,X}(p, \text{inr}(x, x')) = \text{inr}((p, x), (p, x'))$$

Given the free monad $\mathsf{Tree}(A + -) : \mathsf{Set} \to \mathsf{Set}$, its strength $\sigma_{P,Z} : P \times \mathsf{Tree}(A + Z) \to \mathsf{Tree}(A + P \times Z)$ is given by:

$$\sigma_{P,Z}(p, \mathsf{Leaf}(\text{inl}(a))) = \mathsf{Leaf}(a)$$
$$\sigma_{P,Z}(p, \mathsf{Leaf}(\text{inr}(z))) = \mathsf{Leaf}(\text{inr}(p, z))$$
$$\sigma_{P,Z}(p, \mathsf{Node}(l, r)) = \mathsf{Node}(\sigma_{P,Z}(p, l), \sigma_{P,Z}(p, r))$$

F. 2-Categorical Algebra

Good references for 2-categorical algebra are (Lack, 2010) or (Kelly, 2005). The latter deals with the more general notion of enriched categories of which 2-categories are a particular example.

F.1. 2-monads and their Lax Algebras

A 2-monad can conscisely be defined as a Cat-enriched monad (see Johnson et al., Sec. 6.5) An unpacking of its definition follows.

Definition F.1 (2-monad)**.** A **2-monad** on a 2-category $\mathcal{C}$ comprises:

- A 2-endofunctor T on $\mathcal{C}$;
- A 2-natural transformation $\mu : T^2 \Rightarrow T$; and
- A 2-natural transformation $\eta : \mathrm{id}_{\mathcal{C}} \Rightarrow T$;

such that the (Cat-enriched variant of) the axioms in Definition B.1 hold.

Definition F.2 (Lax T-algebra for a 2-monad)**.** Let (T, η, μ) be a 2-monad on $\mathcal{C}$. A lax T-algebra is a pair $(A, r, \epsilon_A, \delta_A)$ where A is an object of $\mathcal{C}$, $a : T(A) \to A$ is a morphism in $\mathcal{C}$, and ϵ_A and δ_A are the following 2-morphisms in $\mathcal{C}$:

$$\begin{tikzcd} T(A) \arrow[d, "a"'] & \\ A \arrow[r, equal] \arrow[ur, phantom, "\Nearrow \epsilon_A"] & A \arrow[ul, "\eta_A"'] \end{tikzcd} \qquad \begin{tikzcd} T(T(A)) \arrow[r, "\mu_A"] \arrow[d, "T(a)"'] \arrow[dr, phantom, "\Nearrow \delta_A"] & T(A) \arrow[d, "a"] \\ T(A) \arrow[r, "a"'] & A \end{tikzcd}$$

such that the lax unity and lax associativity conditions (Johnson et al., Eq. 6.5.6 and 6.5.7) are satisfied.

Definition F.3 (Lax algebra homomorphism)**.** Let $\mathcal{E}$ be a 2-category, T a 2-monad on $\mathcal{E}$, and $(A, a, \epsilon_A, \delta_A)$ and $(B, b, \epsilon_B, \delta_B)$ be lax T-algebras. Then a lax algebra morphism from $(A, a, \epsilon_A, \delta_A) \to (B, b, \epsilon_B, \delta_B)$ is pair (f, κ) where $f : A \to B$ is a morphism in $\mathcal{E}$ and κ is a 2-morphism spanned by the following diagram

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

such that the diagrams in (Johnson et al., Def. 6.5.9) commute.

Lax algebras and lax algebra homomorphisms for the 2-monad T form the category $\mathsf{Lax\text{-}Alg}_{\mathsf{Mnd}}(T)$.

Example F.4 (2-monad for M-modules)**.** If M is a monoid, it is useful to study "M-modules", which we define to be monoids equipped with a compatible action of M.[12] For example, the situation of a group acting on a vector space is fundamental in representation theory. We can interpret such modules in terms of the 2-monad $\mathtt{disc}(M) \times -$ on the 2-category Cat.

Here $\mathtt{disc}(M)$ is the category whose objects are elements of M, and whose morphisms are just the identity arrows. Since M is a monoid, $\mathtt{disc}(M)$ is a monoidal category, so $\mathtt{disc}(M) \times -$ is a 2-monad.

As the set of monoid endomorphisms on a monoid A is isomorphic to the set of endofunctors on the category given by the delooping of the said monoid, an M-module can be seen to be a one-object algebra for this 2-monad. Indeed, this is just a special case of an actegory.

Example F.5 (Cocycles as lax morphisms)**.** The monoid cocycles, or "crossed homomorphisms", used in (Dudzik et al., 2024) to study asynchrony in algorithms and networks, can be described as lax morphisms for 2-monad algebras. As above, if A is an M-module, then $\mathcal{B}A$ is an algebra for $\mathtt{disc}(M) \times -$. A lax morphism $1 \to \mathcal{B}A$ is a natural transformation of the unique functor $M \to \mathcal{B}A$, which is equivalently just a set map $D : M \to A$. The axiom for compositionality is exactly the (right) 1-cocycle condition for D, so 1-cocycles are the same as lax morphisms $1 \to \mathcal{B}A$.

[12] Contrary to a mere M-action A, where A is a set, in a M-module A, A comes equipped with a monoid structure too.

G. The 2-category $\mathsf{Para}(\mathcal{C})$

Definition G.1 (Para, compare (Cruttwell et al., 2022; Capucci et al., 2022))**.** Given a monoidal category $(\mathcal{M}, \otimes, I)$ and an $\mathcal{M}$-actegory $\mathcal{C}$, let $\mathsf{Para}_{\blacktriangleright}(\mathcal{C})$ be the 2-category whose:

- Objects are the objects of $\mathcal{C}$;
- Morphisms $X \to Y$ are pairs (P, f), where P is an object of $\mathcal{M}$ and $f : P \blacktriangleright X \to Y$ is a morphism of $\mathcal{C}$;

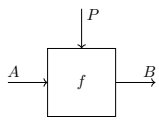


Figure 2. String diagram representation of a parametric morphism. We often draw the parameter wire on the *vertical* axis to signify that the parameter object is part of the data of the morphism.

- 2-morphisms $(P, f) \Rightarrow (P', f') : X \to Y$ are 1-morphisms $r : P' \to P$ of $\mathcal{M}$ such that the triangle

$$\begin{tikzcd} P \triangleright X \arrow[dr, "f"] & \\ & Y \\ P' \triangleright X \arrow[uu, "r \triangleright X"] \arrow[ur, "f'"'] & \end{tikzcd}$$

  commutes (equivalently an equality of parametric string diagrams as in Figure 3).

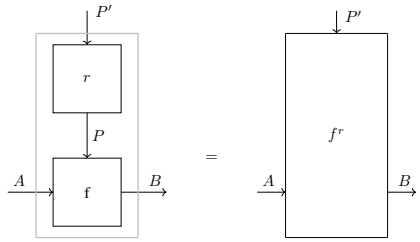


Figure 3. String diagram of reparameterisation. The reparameterisation map r is drawn vertically.

- Identity morphism on X is the parametric map $(I, {\eta_X}^{-1})$, where η is the unitor of the underlying actegory.

where composition of morphisms $(P, f) : X \to Y$ and $(Q, g) : Y \to Z$ is $(Q \otimes P, h)$ where h is the composite

$$(Q \otimes P) \blacktriangleright X \xrightarrow{\mu_{Q,P,X}} Q \blacktriangleright (P \blacktriangleright X) \xrightarrow{Q \blacktriangleright f} Q \blacktriangleright Y \xrightarrow{g} Z$$

and composition of 2-morphisms is given by the composition of $\mathcal{M}$.

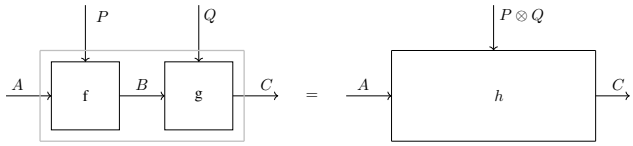


Figure 4. String diagram representation of the composition of parametric morphisms. By treating parameters on a separate axis we obtain an elegant graphical depiction of their composition.

Remark G.2. When the actegory is strict the identities of the actegory reduce to being the parametric morphisms of the form (I, id_X)

Example G.3 (Real Vector Spaces and Smooth Maps)**.** Consider the cartesian category Smooth whose objects are real vector spaces, and morphisms are smooth functions. As this category is cartesian, we can form $\mathsf{Para}(\mathsf{Smooth})$ modelling parametric smooth functions.

Lemma G.4 (Embedding of $\mathcal{C}$ into $\mathsf{Para}(\mathcal{C})$)**.** *There is a 2-functor $\gamma : \mathcal{C} \to \mathsf{Para}(\mathcal{C})$ which is identity-on-objects and treats every morphism $f : A \to B$ as trivially parametric, i.e. as $(I, f \circ \eta_A^{-1})$.*[13]

Remark G.5. When the actegory $(\mathcal{C}, \blacktriangleright)$ is strict the trivially parametric functions do not require precomposition with the unitor of actegory. That is to say a morphism $f : X \to Y$ of $\mathcal{C}$ is sent to the parametric morphism $(I, f) : X \to Y$.

Theorem G.6 (γ preserves connected colimits)**.** *Suppose $(\mathcal{M}, \eta, \mu)$ is a strict monoidal category, that $(\mathcal{C}, \blacktriangleright)$ is an $\mathcal{M}$-actegory, and that $X \blacktriangleright (_) : \mathcal{C} \to \mathcal{C}$ preserves connected colimits. Then, the 2-functor $\gamma : \mathcal{C} \to \mathsf{Para}(\mathcal{C})$ preserves connected colimits.*

Remark G.7. While in $\mathcal{C}$ they are the only kind of colimits, the colimits we mean in $\mathsf{Para}(\mathcal{C})$ are strict colimits. The relationship to other flavours of colimit is a topic of ongoing research.

Proof. Note that for a cone over a trivially parameterised connected diagram to commute, the parameters of the 1-cells of the cone must all be the same. Then, since $\blacktriangleright$ preserves connected colimits by hypothesis, we may assemble the following chain of isomorphisms.

$$
\begin{aligned}
\mathsf{Tr}_1\left(\mathsf{Para}\left(\mathcal{C}\right)\right)\left(\varinjlim_{d\in\mathcal{D}} X_d, Y\right) &\xrightarrow{\sim} \coprod_{P\in\mathcal{M}} \mathcal{C}\left(P \triangleright \varinjlim_{d\in\mathcal{D}} X_d, Y\right) \\
&\xrightarrow{\sim} \coprod_{P\in\mathcal{M}} \mathcal{C}\left(\varinjlim_{d\in\mathcal{D}} P \triangleright X_d, Y\right) \\
&\xrightarrow{\sim} \coprod_{P\in\mathcal{M}} \varprojlim_{d\in\mathcal{D}} \mathcal{C}\left(P \triangleright X_d, Y\right) \\
&\xrightarrow{\sim} \mathsf{Ob}\left(\mathcal{M}\right) \times \varprojlim_{\mathsf{d}\in\mathcal{D}} \mathcal{C}\left(\mathsf{P} \triangleright \mathsf{X_d}, \mathsf{Y}\right) \\
&\xrightarrow{\sim} \varprojlim_{d\in\mathcal{D}} \left(\mathsf{Ob}\left(\mathcal{M}\right) \times \mathcal{C}\left(P \triangleright X_d, Y\right)\right) \\
&\xrightarrow{\sim} \varprojlim_{d\in\mathcal{D}} \left(\coprod_{P\in\mathcal{M}} \mathcal{C}\left(P \triangleright X_d, Y\right)\right) \\
&\xrightarrow{\sim} \varprojlim_{d\in\mathcal{D}} \left(\mathsf{Tr}_1\left(\mathsf{Para}\left(\mathcal{C}\right)\right)\left(X_d, Y\right)\right)
\end{aligned}
$$

□

Example G.8. Fix $\mathcal{M}$-actegory $(\mathcal{C}, \blacktriangleright)$, and a monad (T, μ, η) on $\mathcal{C}$ with actegorical strength $\sigma : M \blacktriangleright T(X) \to T(M \blacktriangleright X)$. Then this gives us a 2-monad on $\mathsf{Para}_{\blacktriangleright}(T)$ on $\mathsf{Para}_{\blacktriangleright}(\mathcal{C})$ whose underlying 2-endofunctor:

- Acts as T does on objects[14]; and

[13] Here we are treating $\mathcal{C}$ as 2-category with only identity 2-morphisms.

[14] This is well defined since $\mathsf{Ob}(\mathsf{Para}(\mathcal{C})) = \mathsf{Ob}(\mathcal{C})$

- Sends $(P, f) : X \to Y$ to (P, f'), where f' is the composite

$$P \blacktriangleright T(X) \xrightarrow{\sigma_{P,X}} T(P \blacktriangleright X) \xrightarrow{T(f)} T(Y)$$

- Sends $r : P \to P'$ to itself.

and the unit and multiplication 2-natural transformations are defined as follows:

- Unit $\mathsf{Para}_{\blacktriangleright}(\eta) : \mathrm{id}_{\mathsf{Para}_{\blacktriangleright}(\mathcal{C})} \Rightarrow \mathsf{Para}_{\blacktriangleright}(T)$ whose component at $X : \mathcal{C}$ is the element of $\mathsf{Para}_{\blacktriangleright}(\mathcal{C})(X, T(X))$ given by the parametric morphism (I, u_X), where u_X is the composite

$$I \blacktriangleright X \xrightarrow{\eta_X^{\blacktriangleright -1}} X \xrightarrow{\eta_X^T} T(X)$$

  For every parametric map $(P, f) \in \mathsf{Para}(\mathcal{C})(X, Y)$ we have the strictly natural square in Equation (9). The 2-cell is given by the reparameterisation $\beta_{P,I} : P \otimes I \to I \otimes P$ which is identity since we are assuming $\mathcal{M}$ is strict monoidal. It can be checked that this indeed satisfies the conditions of a 2-morphism in $\mathsf{Para}_{\blacktriangleright}(\mathcal{C})$. This makes $\mathsf{Para}_{\blacktriangleright}(\eta)$ a 2-natural transformation.

$$\begin{tikzcd} X \arrow[r, "{(P,f)}"] \arrow[d, "{(I,u)}"'] \arrow[dr, phantom, "\Nearrow \beta_{P,I}"] & Y \arrow[d, "{(I,u_Y)}"] \\ T(X) \arrow[r, "{T(f)\circ\sigma_{P,X}}"'] & T(Y) \end{tikzcd} \tag{9}$$

- Multiplication $\mathsf{Para}_{\blacktriangleright}(\mu) : \mathsf{Para}_{\blacktriangleright}(T)^2 \Rightarrow \mathsf{Para}_{\blacktriangleright}(T)$ whose component at $X \in \mathsf{Para}_{\blacktriangleright}(\mathcal{C})$ is the parametric morphism (I, m_X) where m_X is the composite

$$I \blacktriangleright T^2(X) \xrightarrow{\eta_X^{\blacktriangleright -1}} T^2(X) \xrightarrow{\mu_X^T} T(X)$$

  For every parametric morphism $(P, f) \in \mathsf{Para}_{\blacktriangleright}(\mathcal{C})(X, Y)$ we have the strictly natural square in Equation (10). It is again given by $\beta_{P,I}$, and is strict because $\mathcal{M}$ is strict monoidal. It can be checked that this too satisfies the conditions of a 2-morphism in $\mathsf{Para}_{\blacktriangleright}(\mathcal{C})$ This makes $\mathsf{Para}_{\blacktriangleright}(\mu)$ a 2-natural transformation.

$$\begin{tikzcd} T^2(X) \arrow[r, "{(P,(T(f)\circ\sigma_{P,X})\circ\sigma_{P,T(X)})}"] \arrow[d, "{(I,m_x)}"'] \arrow[dr, phantom, "\Nearrow \beta_{P,I}"] & T^2(Y) \arrow[d, "{(I,m_Y)}"] \\ T(X) \arrow[r, "{T(f)\circ\sigma_{P,X}}"'] & T(Y) \end{tikzcd} \tag{10}$$

Lastly, we need to check that this indeed satisfies the 2-monad coherence conditions. Since the parameters of the components of $\mathsf{Para}_{\blacktriangleright}(\eta)$ and $\mathsf{Para}_{\blacktriangleright}(\mu)$ are all trivial, this becomes straightforward to check.

Remark G.9. For strictly monoidal $\mathcal{M}$ and a strict actegory $(\mathcal{C}, \blacktriangleright)$ the unit and multiplication of $\mathsf{Para}(T)$ are exactly the unit and multiplication of T.

The following theorem shows us how out of the abstract 2-categorical framework the notion of weight tying (usage of the same weight in two different places obtained by copying the value) arises automatically as the structure of a comonoid induced by a lax algebra of a strong actegorical monad on $\mathcal{C}$.

Theorem G.10 (Lax (co)algebras for $\mathsf{Para}_{\blacktriangleright}(T)$ induce comonoids)**.** *Let $(\mathcal{C}, \blacktriangleright)$ be a $\mathcal{M}$-actegory and $T : \mathcal{C} \to \mathcal{C}$ a strong actegorical monad on $\mathcal{C}$. Consider a lax algebra $(A, (P, a), \epsilon_A, \delta_A)$ for the induced 2-monad $\mathsf{Para}(T)$. Then P is a comonoid in $\mathcal{M}$ where ϵ_A is the data of its counit, and δ_A the data of its comultiplication, and the comonoid laws follow from lax algebra coherence conditions. Dually, the same statement holds for a lax coalgebra of a $\mathsf{Para}(T)$ comonad.*

Proof. We start by unpacking the data of the lax algebra 2-cells.

$$\begin{tikzcd} & T(X) \arrow[dl, "{(P,a)}"'] & \\ X \arrow[rr, equal, "{(I,\lambda_X)}"'] & \arrow[u, phantom, "\Uparrow \epsilon_P"] & X \arrow[ul, "{(I,\eta^T_X \circ \eta^{\blacktriangleright\,-1}_X)}"'] \end{tikzcd} \qquad \begin{tikzcd} T(T(X)) \arrow[r, "{(I,m_x)}"] \arrow[d, "{(P,T(a)\circ\sigma_{P,X})}"'] & T(X) \arrow[d, "{(P,a)}"] \\ T(X) \arrow[r, "{(P,a)}"'] \arrow[ur, phantom, "\Nearrow \delta_P"] & X \end{tikzcd}$$

We can see that they are given by two reparameterisations: $\epsilon_P : P \otimes I \to I$ and $\delta_P : I \otimes P \to P \otimes P$. These uniquely determine the morphisms which we call $!_P : P \to I$ and $\Delta_P : P \to P \otimes P$ respectively. To show that this is indeed a comonoid, we need to unpack the lax algebra laws. We note that all the conditions unpack only to conditions on morphisms in $\mathcal{M}$. It is relatively tedious, but straightforward to check that these conditions ensure that $!_P$ and Δ_P satisfy the comonoid laws. □

Corollary G.11. *The functor* $\mathsf{Lax\text{-}Alg}_{\mathsf{Mnd}}(\mathsf{Para}(T)) \to \mathsf{CoMon}(\mathcal{C}) \times \mathsf{Lax}(\to, \mathcal{C})$ *which takes lax algebras for* $\mathsf{Para}(T)$ *to the pair of the underlying comonoid parameter and the* $\mathcal{C}$*-morphism which interprets the parametric structure map is full-and-faithful.*

Proof. The proof is formal and left to the reader. □

Conjecture G.12. Given a cartesian monoidal category $(\mathcal{M}, \times, 1)$, an $\mathcal{M}$-actegory $(\mathcal{C}, \blacktriangleright)$, and a strong endofunctor F on $\mathcal{C}$ such that F and $\mathcal{C}$ satsify the hypotheses of Theorem B.16, then the assignment of a lax algebra $(X, (P, f))$ to

$$(X, ((P, !_P : P \to 1, \Delta_P : P \to P \times P), \underrightarrow{\mathsf{laxlim}}((P, f) \circ \mathsf{Para}(F)(P, f) \circ \cdots \circ \mathsf{Para}(F)^{\alpha}(P.f)) : F^{\kappa}X \to X))$$

defines an equivalence of categories

$$\mathsf{Lax\text{-}Alg}_{\mathsf{Endo}}(\mathsf{Para}(F)) \to \mathsf{Lax\text{-}Alg}_{\mathsf{Mnd}}(F^{\kappa})$$

H. Weight Tying Examples

H.1. Examples from Geometric Deep Learning

Example H.1 (Linear equivariant layers for a pair of pixels)**.** Consider the category Vect of finite-dimensional vector spaces and linear maps. For simplicity, we will assume that the carrier set for our data is $R^{\mathbb{Z}_2}$, which is a pair of pixels. Consider a linear endofunction on such data, $f_{\mathbf{W}} : \mathbb{R}^{\mathbb{Z}_2} \to \mathbb{R}^{\mathbb{Z}_2}$. It is well known that this function can be represented as a multiplication with a 2×2 matrix $\mathbf{W} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix}$.

Now consider the group of 1D translations $(\mathbb{Z}_2, +, 0)$—which in this case amounts to *pixel swaps*—and the induced action $\blacktriangleright$ on $\mathbb{R}^{\mathbb{Z}_2}$. If $f : \mathbb{R}^{\mathbb{Z}_2} \to \mathbb{R}^{\mathbb{Z}_2}$ is equivariant with respect to $\blacktriangleright$, then via Example 2.6, for any input $[x_1, x_2] \in \mathbb{R}^{\mathbb{Z}_2}$ it must hold that

$$\begin{bmatrix} w_1 x_2 + w_2 x_1 \\ w_3 x_2 + w_4 x_1 \end{bmatrix} = \begin{bmatrix} w_3 x_1 + w_4 x_2 \\ w_1 x_1 + w_2 x_2 \end{bmatrix} \tag{11}$$

This implies that $w_3 = w_2$ and $w_4 = w_1$, meaning every $\mathbf{W}$ is of the form

$$\begin{bmatrix} w_1 & w_2 \\ w_2 & w_1 \end{bmatrix}$$

where the weight tying makes the matrix a symmetric one. Any neural network $f_{\mathbf{W}}$ satisfying this constraint will be a *linear translation equivariant layer* over $\mathbb{Z}_2$, and can be used as a building block to construct geometric deep learning architectures.

Remark H.2 (Circulants and CNNs). Note that this concept generalises to larger input domains (e.g. $\mathbb{R}^{\mathbb{Z}_k}$ for $k > 2$). Generally, it is a well-known fact in signal processing that, for $f_{\mathbf{W}}$ to satisfy a linear translation equivariance constraint, $\mathbf{W}$ must be a *circulant* matrix. Circulant matrices are known in neural networks as *convolutional layers*, the essence of modern CNNs (Fukushima et al., 1983; LeCun et al., 1998).

Example H.3 (Invariant maps)**.** Invariant maps are also $(G \times -)$-algebra homomorphisms where the codomain is a trivial group action. Specifically, setting the domain to any of the group actions from Example 2.6, the corresponding induced commutative diagram is below:

$$\begin{tikzcd} G \times \mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h} \arrow[r, "G \times f"] \arrow[d, "\blacktriangleright"'] & G \times \mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h} \arrow[d, "\pi_2"] \\ \mathbb{R}^{Z_w \times \mathbb{Z}_h} \arrow[r, "f"'] & \mathbb{R}^{\mathbb{Z}_w \times \mathbb{Z}_h} \end{tikzcd}$$

Elementwise, the commutativity of that diagram unpacks to the equation

$$f(g \blacktriangleright x) = f(x)$$

The translation example, for instance, recovers the equation $f(((i', j') \blacktriangleright x)(i, j)) = (i', j') \blacktriangleright f(x)(i, j)$ which reduces to $f(x(i - i', j - j')) = f(x)(i, j)$. Intuitively, it states that, for any displacement (i', j'), the result of translating the input by (i', j') and then applying the function f is the same as just applying f.

We can run the analogous calculation with weight sharing here. We consider the same linear endofunction $f_{\mathbf{W}} \in \mathbb{R}^{\mathbb{Z}_2} \to \mathbb{R}^{\mathbb{Z}_2}$ on a pair of pixels (represented as a matrix $W = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix}$), and the same group of translations $(\mathbb{Z}_2, +, 0)$ on the domain, but this time we consider the identity endomorphism on the codomain. If $f_{\mathbf{W}}$ is invariant to $\blacktriangleright$, then Example H.3 for any input $[x_1, x_2] \in \mathbb{R}^{\mathbb{Z}_2}$ it has to hold that

$$\begin{bmatrix} w_1 x_2 + w_2 x_1 \\ w_3 x_2 + w_4 x_1 \end{bmatrix} = \begin{bmatrix} w_1 x_1 + w_2 x_2 \\ w_3 x_1 + w_4 x_2 \end{bmatrix} \tag{12}$$

implying $w_1 = w_2$ and $w_3 = w_4$, meaning the equivariance induced a particular weight tying scheme

$$\begin{bmatrix} w_1 & w_3 \\ w_1 & w_3 \end{bmatrix}$$

in which the matrix has shared rows. Note that this implies that each of the pixel values in the output of $f_{\mathbf{W}}$ will be the same, and hence this layer could also be represented as just a single dot product with $\begin{bmatrix} w_1 & w_3 \end{bmatrix}$, *eliminating* the grid structure in the original set. Such a layer would not be an endofunction on $\mathbb{R}^{\mathbb{Z}_2}$, however, which was our initial space of exploration.

H.2. Examples from Automata Theory

H.2.1. Streams

Example H.4 (Streams)**.** Let O be a set, thought of as the set of outputs. Consider the endofunctor from $O \times - : \mathsf{Set} \to \mathsf{Set}$. Then the set $\mathsf{Stream}(O)$ of streams[15] with outputs O, together with the map $\langle \mathsf{output}, \mathsf{next} \rangle : \mathsf{Stream}(O) \to O \times \mathsf{Stream}(O)$ forms a coalgebra of this endofunctor. They have a representation as the following datatype

```
data Stream o = MkStream {
  output :: o
  next :: Stream o
}
```

This datatype describes streams *coinductively*, as something from which we can always extract an output, and another stream. In Example I.3 we will see how this will be related to unfolding recurrent neural networks.

Example H.5 (Unfolds to streams as coalgebra homomorphisms)**.** Consider the endofunctor $(O \times -)$ from Example H.4, and a coalgebra homomorphism from any other $(O \times -)$-coalgebra $(X, \langle o, n \rangle)$ into $(\mathsf{Stream}(O), \langle \mathsf{output}, \mathsf{next} \rangle)$:

[15] An infinite sequence of outputs, isomorphic to the set $(\mathbb{N} \to O)$ of functions from the natural numbers into O.

$$\begin{tikzcd} X \arrow[r, "f_{o,n}"] \arrow[d, "\langle o,n\rangle"'] & \mathsf{Stream}(O) \arrow[d, "\langle \mathsf{output},\mathsf{next}\rangle"] \\ O \times X \arrow[r, "O\times f_{o,n}"'] & O \times \mathsf{Stream}(O) \end{tikzcd}$$

Then the map $f_{o,n} : X \to \mathsf{Stream}(O)$ is necessarily a *unfold* to a stream, a concept from functional programming describing how a stream of vaues can be obtained from a single value. It is implemented by *corecursion* on the input:

```
f_{o,n} :: x -> Stream o
f_{o,n} x = MkStream (o x) (f_{o,n} (n x))
```

This corecursion is structural in nature, meaning it satisfies the following two equations which arise by unpacking the algebra homomorphism equations elementwise:

$$o(x) = \mathsf{output}(f_{o,n}(x)) \tag{13}$$

$$f_{o,n}(n(x)) = \mathsf{next}(f_{o,n}(x)) \tag{14}$$

Here Equation (13) tells us that the output of the stream produced by $f_{o,n}$ at x is $o(x)$, and Equation (14) tells us that the rest of the stream is the stream produced by $f_{o,n}$ at x is $f_{o,n}(n(x))$.

Remark H.6. Analogous to Remark 2.13, there can only ever be *one* unfold of this type. This is because streams are a terminal object in the category of $(O \times -)$-coalgebras.

We can study the weight sharing induced by streams. Consider the category Vect and the coreader comonad $\mathbb{R} \times -$ given by the output set $\mathbb{R}$. Fix a coalgebra $(\mathbb{R}, \langle o, n\rangle)$, and represent the coalgebra map with scalars w_o and w_n, denoting the output and the next state, respectively. Then the universal coalgebra homomorphism is a linear map $f_{o,n} : \mathbb{R} \to \mathsf{Stream}(\mathbb{R})$ satisfying Equations (13) and (14). If we represent this as an infinite-dimensional matrix

$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 & \dots \end{bmatrix}$$

then the induced weight sharing scheme removes any degrees of freedom: the matrix is completely determined by w_o and w_n, and is of the form:

$$\begin{bmatrix} w_o & w_n w_o & w_n^2 w_o & w_n^3 w_o & \dots \end{bmatrix}$$

H.2.2. Moore Machines.

Example H.7 (Moore machines)**.** Let O and I be sets, thought of as sets of outputs and inputs, respectively. Consider the endofunctor $O \times (I \to -) : \mathsf{Set} \to \mathsf{Set}$. Then the set $\mathsf{Moore}_{O,I}$ of Moore Machines with O-labelled outputs and I-labelled inputs together with the map $\langle \mathsf{output}, \mathsf{next}\rangle : \mathsf{Moore}_{O,I} \to O \times (I \to \mathsf{Moore}_{O,I})$ forms a coalgebra for this endofunctor. They can be represented as the following datatype.

```
data Moore o i = MkMoore {
  output :: o,
  nextStep :: (i -> Moore o i)
}
```

Like with Mealy machines, this description is coinductive. From a Moore machine we can always extract an output and a function which given an input produces another Moore machine. In Example I.5 we will see how this will be related to general recurrent neural networks of a particular form.

Example H.8 (Unfolds to Moore machines are terminal $(O \times -)$-coalgebras)**.** Consider the endofunctor $O \times (I \to -)$ from Example H.7, and any other $(O \times -)$-coalgebra $(X, \langle o, n\rangle)$ to $(\mathsf{Moore}_{O,I}, \langle \mathsf{output}, \mathsf{next}\rangle$:

$$\begin{tikzcd} X \arrow[r, "f_{o,n}"] \arrow[d, "\langle o,n \rangle"'] & \mathsf{Moore}_{O,I} \arrow[d, "\langle \mathsf{output}, \mathsf{next} \rangle"] \\ O \times (I \to X) \arrow[r, "O \times (I \to f_{o,n})"'] & O \times (I \to \mathsf{Moore}_{O,I}) \end{tikzcd}$$

Then the map $f_{o,n}$ is a necessarily an *unfold* to a Moore machine, a function describing how a Moore machine is obtained from a single value. It is a corecursive function as below:

```
fm :: x -> Moore o i
fm x = MkMoore (m1 x) (λi ↦ fm (m2 x i))
```

which is structural in nature, meaning it satisfies the following two equations which arise by unpacking the coalgebra homomorphism equations elementwise:

$$o(x) = \mathsf{output}(f_{o,n}(x)) \tag{15}$$

$$f_{o,n}(n(x)(i)) = \mathsf{next}(f_{o,n}(x))(i) \tag{16}$$

Here Equation (15) tells us that the output of the Moore machine produced by $f_{o,n}$ at state x is given by the output of n at state x, and Equation (16) tells us that the next Moore machine produced at x and i is the one produced by $f_{o,n}$ at $n(x)(i)$.

I. Parametric 2-endofunctors and their Algebras

Just like before, often at our disposal we will have a more minimal structure. Instead of requiring a strong 2-monad, we can instead require merely a strong 2-endofunctor. The data of an algebra for a 2-endofunctor will be the same as the algebra for an endofunctor Definition 2.8.

Example I.1 (Folding RNN cell)**.** Consider the endofunctor $1 + A \times -$ from Example 2.9. Via strength from Example E.10 we can form the 2-endofunctor $\mathsf{Para}(1 + A \times -) : \mathsf{Para}(\mathsf{Set}) \to \mathsf{Para}(\mathsf{Set})$. Then a folding recurrent neural network arises as its algebra.

More concretely, an algebra here consists of the carrier set S (name suggestively chosen to denote *hidden state*) and a parametric map $(P, \mathsf{cell}^{\mathsf{rcnt}}) \in \mathsf{Para}(\mathsf{Set})(1 + A \times S, S)$. Via the isomorphism $P \times (1 + A \times S) \cong P + P \times A \times S$ we can break $\mathsf{cell}^{\mathsf{rcnt}}$ into two pieces: the choice of the initial hidden state $\mathsf{cell}^{\mathsf{rcnt}}_0 : P \to X$ and the folding recurrent neural network cell $\mathsf{cell}^{\mathsf{rcnt}}_1 : P \times A \times X \to X$, as shown in the figure below. We will see how iterating this construction will

Figure 5. An algebra for $\mathsf{Para}(1 + A \times -)$ consists of a recurrent neural network cell and an initial hidden state.

produce a folding recurrent neural network which consumes a sequence of inputs and iteratively updates its hidden state.[16]

Example I.2 (Recursive NN cell)**.** Consider the endofunctor $A + (-)^2$ from Example 2.10. Via strength from Example E.10 we can form the 2-endofunctor $\mathsf{Para}(A + (-)^2)$ on $\mathsf{Para}(\mathsf{Set})$. Then a recursive neural network arises as its algebra.

More concretely, an algebra here consists of the carrier set S and a parametric map $(P, \mathsf{cell}^{\mathsf{rcsv}}) \in \mathsf{Para}(\mathsf{Set})(A + S^2, S)$. Via the isomorphism $P \times (A + S^2) \cong P + P \times A \times S^2$ we can break $\mathsf{cell}^{\mathsf{rcsv}}$ into two pieces: the choice of the initial hidden state $\mathsf{cell}^{\mathsf{rcsv}}_0 : P \to S$ and the recursive neural network cell $\mathsf{cell}^{\mathsf{rcsv}}_1 : P \times A \times S^2 \to S$, as shown in the figure below.

[16] We note here that the hidden state is allowed to depend on the parameter, something which is not possible in the usual definition of a recurrent neural network cell.

Figure 6. An algebra for $\mathsf{Para}(A + (-)^2)$ consists of a recursive neural network cell and an initial hidden state.

Example I.3 (Unfolding RNN cell)**.** Consider the endofunctor $O \times -$ from Example H.4. Via strength $\sigma_{P,X} : P \times (O \times X) \xrightarrow{\cong} O \times P \times X$ we can form the 2-endofunctor $\mathsf{Para}(O \times -) : \mathsf{Para}(\mathsf{Set}) \to \mathsf{Para}(\mathsf{Set})$. Then an unfolding recurrent neural network arises as its coalgebra.

More concretely, a coalgebra here consists of the carrier set S, and a parametric map $(P, \langle \mathsf{cell}_o, \mathsf{cell}_n \rangle) \in \mathsf{Para}(\mathsf{Set})(S, O \times S)$. Here $\langle \mathsf{cell}_o, \mathsf{cell}_n \rangle$ consists of maps $\mathsf{cell}_o : P \times S \to O$ which computes the output and $\mathsf{cell}_n : P \times S \to S$ which computes the next state.[17] We will see how iterating this construction will produce an unfolding recurrent neural network

Figure 7. An algebra for the $\mathsf{Para}(O \times -)$ 2-endofunctor consists of an unfolding recurrent neural network cell.

which given a starting state produces a stream of outputs O.

Example I.4 (Mealy machine cell / Full RNN cell)**.** Consider the endofunctor $I \to O \times -$ from Example 2.11. Via strength from Example E.10 we can form the 2-endofunctor $\mathsf{Para}(I \to O \times -) : \mathsf{Para}(\mathsf{Set}) \to \mathsf{Para}(\mathsf{Set})$. Then a Mealy machine cell arises as its coalgebra.

More concretely, a coalgebra here consists of the carrier set S and a parametric map $(P, \mathsf{cell}^{\mathsf{Mealy}}) \in \mathsf{Para}(\mathsf{Set})(S, I \to O \times S)$. Here $\mathsf{cell}^{\mathsf{Mealy}}$ can be thought of as a map $P \times S \to I \to O \times S$, as shown in the figure below. We interpret it as a

Figure 8. An algebra for $\mathsf{Para}((I \to O \times -)$ 2-endofunctor consists of an object S and a map $f : P \times S \times I \to O \times S$ which we have taken the liberty to uncurry.

full recurrent neural network consuming a hidden state S, input I and producing an output O and an updated hidden state S.

This suggests that recurrent neural networks can be thought of as learnable Mealy machines, a perspective seldom advocated for in the literature.

Example I.5 (Moore machine cell)**.** Consider the endofunctor $O \times (I \to -)$ from Example H.7. Via strength $\sigma_{P,X} : P \times O \times (I \to O) \to O \times (I \to P \times X)$ defined as $(p, o, f) \mapsto (o, \lambda i \mapsto (p, f(i)))$ we can form the 2-endofunctor $\mathsf{Para}(O \times (I \to -)) : \mathsf{Para}(\mathsf{Set}) \to \mathsf{Para}(\mathsf{Set})$. Then a Moore machine cell arises as its coalgebra.

More concretely, a coalgebra here consists of the carrier set S and a parametric map $(P, \mathsf{cell}^{\mathsf{Moore}}) \in \mathsf{Para}(\mathsf{Set})(S, O \times (I \to S))$. Here $\mathsf{cell}^{\mathsf{Moore}}$ can be thought of as a map $P \times S \to O \times (I \to S)$, as shown in the figure below. We can break it down into two pieces $\mathsf{cell}_o^{\mathsf{Moore}} : P \times S \to O$ and $\mathsf{cell}_n^{\mathsf{Moore}} : P \times S \times I \to S$.

[17] By universal property of the product we can treat them as one cell of type $P \times S \to O \times S$ or as two cells.

$$\begin{tikzcd} X \arrow[d, "{(P,\mathsf{cell}^{\mathsf{Moore}})}"] \\ O \times (I \to X) \end{tikzcd}$$

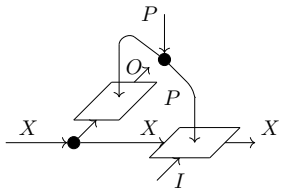


Figure 9. An algebra for $\mathsf{Para}(O \times (I \to -))$ as a parametric Moore cell. Here the output O does not depend on the current input I.

J. Unrolling Neural Networks via Transfinite Construction

We will now reap the benefits of the transfinite construction of free monads on endofunctors, and unpack the corresponding unrolling of these networks. Specifically, with Theorem G.10 hinting towards the path of having weight tying arise abstractly out of the categorical framework, we unpack the unrolling of these networks. For the purposes of space and clarity, the unpaking is done as lax homomorphism of algebras for 2-endofunctors instead of for 2-monads.

Example J.1 (Iterated Folding RNN)**.** Consider an algebra $(P, \mathsf{cell}^{\mathsf{rcnt}})$ of $\mathsf{Para}(1 + A \times -)$, i.e. a folding RNN cell (Example I.1). Its unrolling is a $\mathsf{Para}(1 + A \times -)$-algebra homomorphism below, where $\Delta_P : P \to P \times P$ is the copy map.

$$\begin{tikzcd} 1 + A \times \mathsf{List}(A) \arrow[r, "{\mathsf{Para}(1+A\times-)((P,f_{\mathsf{rcnt}}))}"] \arrow[d, "{\gamma([\mathsf{Nil},\mathsf{Cons}])}"'] & 1 + A \times X \arrow[d, "{(P,\mathsf{cell}^{\mathsf{rcnt}})}"] \arrow[dl, Rightarrow, "\Delta_P"'] \\ \mathsf{List}(A) \arrow[r, "{(P,f_{\mathsf{rcnt}})}"'] & X \end{tikzcd} \tag{17}$$

Here the map f_{rcnt} is the parametric analogue of a *fold* (Example 2.12):

```
f_rcnt :: (p, List a) -> x
f_rcnt p Nil = cell^rcnt p (inl ())
f_rcnt p (Cons a as) = cell^rcnt p (inr a (f_rcnt p as))
```

We can see that f_{rcnt} is structurally recursive: it processes the head of the list by applying $\mathsf{cell}^{\mathsf{rcnt}}$ to the parameter p and the output of the f_{rcnt} with the same parameter, and the tail of the same list.

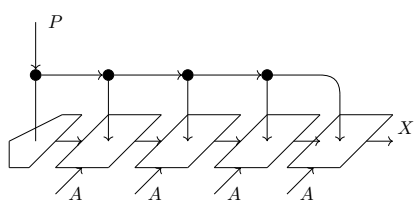


Figure 10. String diagram representation of the unrolled "folding" recurrent neural network.

We proceed to show that Δ_P is indeed a valid reparameterisation for these parametric morphisms. Unpacking the composition on top right side, and the bottom left side of Equation (17), respectively, yields parametric maps

$$P \times P \times (1 + A \times \mathsf{List}(A)) \to X \qquad\qquad P \times (1 + A \times \mathsf{List}(A)) \to X$$

whose implementations are, respectively:

$$(p, q, \mathrm{inl}(\bullet)) \mapsto \mathsf{cell}^{\mathsf{rcnt}}(p, \mathrm{inl}(\bullet)) \qquad\qquad (p, \bullet) \mapsto \mathsf{cell}^{\mathsf{rcnt}}(p, \mathrm{inl}(\bullet))$$
$$(p, q, \mathrm{inr}(a, as)) \mapsto \mathsf{cell}^{\mathsf{rcnt}}(q, a, f(p, as)) \qquad\qquad (p, (a, as)) \mapsto \mathsf{cell}^{\mathsf{rcnt}}(p, a, f(p, as))$$

It is easy to see that Δ_P is a valid reparameterisation between them, "tying" the parameters p and q together.

Example J.2 (Iterated unfolding RNN)**.** Consider a coalgebra $(P, \langle \mathsf{cell}_o, \mathsf{cell}_n \rangle)$ of $\mathsf{Para}(O \times -)$, i.e. an unfolding RNN cell (Example I.3). Its unrolling is a $\mathsf{Para}(O \times -)$-coalgebra homomorphism below:

$$\begin{tikzcd} X \arrow[r, "{(P, f_{o,n})}"] \arrow[d, "{(P, \langle \mathsf{cell}_o, \mathsf{cell}_n \rangle)}"'] \arrow[dr, phantom, "\Delta_P"] & \mathsf{Stream}(O) \arrow[d, "\gamma(\langle \mathsf{output}, \mathsf{next} \rangle)"] \\ O \times X \arrow[r, "{\mathsf{Para}(O \times -)((P, f_{o,n}))}"'] & O \times \mathsf{Stream}(O) \end{tikzcd} \tag{18}$$

Here $f_{o,n} : P \times X \to \mathsf{Stream}(O)$ is the parametric analogue of an *unfold* (Example H.5):

```
f_{o,n} :: (p, x) -> Stream o
f_{o,n} p x = MKStream (cell_o p x) (f_{o,n} p (cell_n x))
```

We can see that $f_{o,n}$ is structurally corecursive: it produces a stream whose head is $\mathsf{cell}_o(p, x)$, and whose tail is the stream produced by $f_{o,n}$ at $\mathsf{cell}_n(p, x)$.

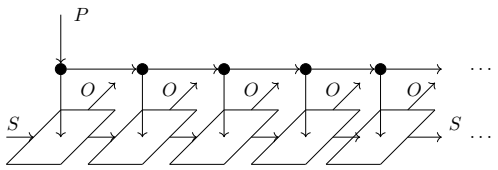


Figure 11. String diagram representation of the unrolled "unfolding" recurrent neural network.

Example J.3 (Iterated Recursive NN)**.** Consider an algebra $(P, \mathsf{cell}^{\mathsf{rcsv}})$ of $\mathsf{Para}(A + (-)^2)$, i.e. a recursive neural network cell (Example I.2). Its unrolling is a $\mathsf{Para}(A + (-)^2)$-algebra homomorphism below:

$$\begin{tikzcd} A + \mathsf{Tree}(A)^2 \arrow[r, "{\mathsf{Para}(A+(-)^2)((P, f_{\mathsf{rcsv}}))}"] \arrow[d, "\cong"'] \arrow[dr, phantom, "\Delta_P"] & A + X^2 \arrow[d, "{(P, \mathsf{cell}^{\mathsf{rcsv}})}"] \\ \mathsf{Tree}(A) \arrow[r, "{(P, f_{\mathsf{rcsv}})}"'] & X \end{tikzcd} \tag{19}$$

Here f_{rcsv} is a function with the following implementation

```
f_rcsv :: (p, Tree a) -> x
f_rcsv p (Leaf a) = cell^rcsv p (inl a)
f_rcsv p (Node l r) = cell^rcsv p inr (f (f_rcsv p l) (f_rcsv p r)
```

This function too performs weight sharing, as the recursive call to subtrees is done with the same parameter p.

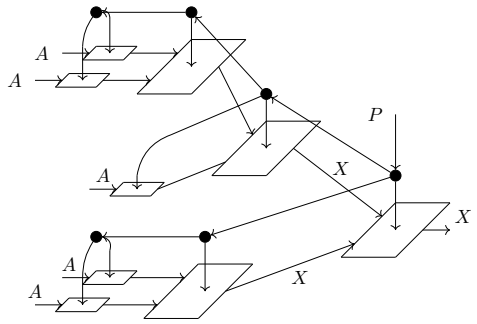


Figure 12. String diagram representation of the unrolled recursive neural network.

Example J.4 (Iterated Mealy machine cell)**.** Consider an algebra $(P, \mathsf{cell}^{\mathsf{Mealy}})$ of $\mathsf{Para}(I \to O \times -)$, i.e. a Mealy machine cell (Example 2.11). Its unrolling is a $\mathsf{Para}(I \to O \times -)$-algebra homomorphism below:

$$\begin{tikzcd} X \arrow[rr, "{(P,f_n)}"] \arrow[d, "{(P,n)}"'] \arrow[drr, phantom, "\Delta_P"] & & \mathsf{Mealy}_{O,I} \arrow[d, "\gamma(\mathsf{next})"] \\ (I \to O \times X) \arrow[rr, "{\mathsf{Para}(I \to O \times -)((P,f_n))}"'] & & (I \to O \times \mathsf{Mealy}_{O,I}) \end{tikzcd} \tag{20}$$

Here f_n is a corecursive function with the following implementation

```
f_n :: (p, x) -> Mealy o i
f_n (p, x) = MkMealy $ \i -> let (o', x') = n p x i
                              in (o', f_n p x')
```

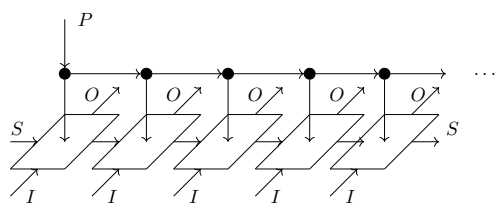


Figure 13. String diagram representation of a parametric Mealy machine — a recurrent neural network.

Example J.5 (Iterated Moore machine cell)**.** Consider a coalgebra $(P, \mathsf{cell}^{\mathsf{Moore}})$ of $\mathsf{Para}(O \times (I \to -))$, i.e. a Moore machine cell (Example H.7). Its unrolling is a $\mathsf{Para}(O \times (I \to -))$-coalgebra homomorphism below:

$$\begin{tikzcd} X \arrow[rr, "{(P,f_n)}"] \arrow[d, "{(P,\langle o,n \rangle)}"'] \arrow[drr, phantom, "\Delta_P"] & & \mathsf{Moore}_{O,I} \arrow[d, "\gamma(\mathsf{next})"] \\ O \times (I \to X) \arrow[rr, "{\mathsf{Para}(O \times (I \to -))((P,f_n))}"'] & & (I \to O \times \mathsf{Moore}_{O,I}) \end{tikzcd} \tag{21}$$

In code, f_n is a corecursive function

```
f_n :: (p, x) -> Moore o i
f_n (p, x) = MkMoore $ \i -> (o p x, f_n p (n p x i))
```

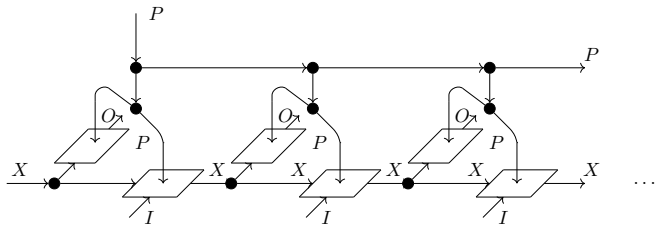


Figure 14. String diagram representation of an unrolled Moore machine.