Learners' Languages

David I. Spivak

Topos Institute
Berkeley USA

david@topos.institute

In "Backprop as functor", the authors show that the fundamental elements of deep learning—gradient descent and backpropagation—can be conceptualized as a strong monoidal functor **Para**(**Euc**) → **Learn** from the category of parameterized Euclidean spaces to that of learners, a category developed explicitly to capture parameter update and backpropagation. It was soon realized that there is an isomorphism **Learn** $\cong$ **Para**(**SLens**), where **SLens** is the symmetric monoidal category of simple lenses as used in functional programming.

In this note, we observe that **SLens** is a full subcategory of **Poly**, the category of polynomial functors in one variable, via the functor $A \mapsto Ay^A$. Using the fact that (**Poly**, $\otimes$) is monoidal closed, we show that a map $A \to B$ in **Para**(**SLens**) has a natural interpretation in terms of dynamical systems (more precisely, generalized Moore machines) whose interface is the internal-hom type $[Ay^A, By^B]$.

Finally, we review the fact that the category p-**Coalg** of dynamical systems on any $p \in$ **Poly** forms a topos, and consider the logical propositions that can be stated in its internal language. We give gradient descent as an example, and we conclude by discussing some directions for future work.

1 Introduction

In the paper "Backprop as functor" [9], the authors show that gradient descent and backpropagation, as used in deep learning, can be conceptualized as a strong monoidal functor $L\colon$ **Para**(**Euc**) → **Learn** from the category of parameterized euclidean spaces to that of learners, a category developed explicitly to capture parameter update and backpropagation. Here, **Para** is a monad on the category of symmetric monoidal categories. It sends $(\mathcal{C}, I, \otimes)$ to a category with the same objects Ob **Para**$(\mathcal{C}) \coloneqq$ Ob $\mathcal{C}$, but with hom-sets that involve a parameterizing object

$$\textbf{Para}(\mathcal{C})(c_1, c_2) \coloneqq \{(p, f) \mid p \in \mathcal{C}, f\colon c_1 \otimes p \to c_2\}/\sim$$

where the parameterizing object p is considered up to an equivalence relation $\sim$.[1] The composite of $c_1 \otimes p_1 \to c_2$ and $c_2 \otimes p_2 \to c_3$ in **Para**$(\mathcal{C})$ has parameterizing object $p_1 \otimes p_2$ and is given by the ordinary composite

$$c_1 \otimes (p_1 \otimes p_2) \to c_2 \otimes p_2 \to c_3.$$

The domain of the backpropagation functor, **Para**(**Euc**), is thus the Para construction applied to the Cartesian monoidal category of Euclidean spaces $\mathbb{R}^n$ and smooth maps.

---

[1] The equivalence relation $\sim$ is generated by regarding $(p, f) \sim (p', f')$ if there exists an epimorphism $g\colon p \twoheadrightarrow p'$ with $f = g \,\mathbin{\fatsemi}\, f'$. As discussed in Gavranović's thesis [10], it is often preferable to dispense with the equivalence relation and instead conceive of **Para**$(\mathcal{C})$ as a bicategory. This is what we shall do as well, though we use different 2-morphisms; see Warning 2.12.

But it was soon realized that **Learn** is in fact also given by a Para construction; namely there is an isomorphism **Learn** $\cong$ **Para**(**SLens**), where **SLens** is the symmetric monoidal category of simple lenses as used in functional programming. The objects of **SLens** are sets $\mathrm{Ob}(\mathbf{SLens}) = \mathrm{Ob}(\mathbf{Set})$, but a morphism consists of a pair of functions

$$\mathbf{SLens}(A,B) := \{(f_1, f^\sharp) \mid f_1 \colon A \to B, \quad f^\sharp \colon A \times B \to A\}. \tag{1}$$

Thus a map $A \to B$ in **Para**(**SLens**) consists of a set P and functions $f_1 \colon A \times P \to B$ and $f^\sharp \colon A \times B \times P \to A \times P$. The authors of [9] developed this structure in order to conceptualize the compositional nature of deep learning as comprising a parameterizing set P (often called "the space of weights and biases") and three functions:

$$
\begin{aligned}
I &\colon A \times P \to B & &\text{implement} \\
U &\colon A \times B \times P \to P & &\text{update} \\
R &\colon A \times B \times P \to A & &\text{request}
\end{aligned}
\tag{2}
$$

The implement function is a P-parameterized function $A \to B$, and the update and request functions take a pair (a,b) of "training data" and respectively update the parameter—e.g. by gradient descent—and return an element of the input space A, which is used to train another such function in the network. This last step—the request—is not just found in deep learning as practiced, but is in fact crucial for defining composition.[2]

But by this point, the notation (P,I,U,R) of **Learn** has become heavy and the structure seems to be getting lost. Even knowing **Learn** $\cong$ **Para**(**SLens**) seems ad-hoc since the morphisms (1) of **SLens** are—to this point—mathematically unmotivated. This is where **Poly** comes in.

In this note, we observe that **SLens** is a full subcategory of **Poly**, the category of polynomial functors in one variable, via the functor $A \mapsto Ay^A$. Using the fact that $(\mathbf{Poly}, \otimes)$ is monoidal closed, we will reconceptualize **Para**(**SLens**) in terms of polynomial coalgebras, which can be understood as dynamical systems: machines with states that can be observed as "output" and updated based on "input" [7]. In particular, a morphism $A \to B$ in **Learn** will be recast as a coalgebra on the internal hom polynomial $[Ay^A, By^B]$, and we will explain this in terms of dynamics.

This viewpoint allows us to substantially generalize the construction in **Learn**, a construction which also appears prominently in the theory of open games [6]. Perhaps more interestingly, it allows us to use the fact that the category p-**Coalg** of dynamical systems on any interface $p \in \mathbf{Poly}$ forms a *topos*. A topos is a setting in which one can do dependent type theory and higher-order logic. In fact, the topos of p-coalgebras is in some ways as simple as possible: it is not only a copresheaf topos p-**Coalg** $\cong$ **Set**$^{\mathcal{C}}$ for a certain category $\mathcal{C}$, but in fact the site $\mathcal{C}$ is the free category on a directed graph that we will call Tree_p. This makes the logic of p-coalgebras—and hence of dynamical systems, learners, and game-players— quite simple. However, the particular graph Tree_p associated to p is highly-structured, and we should find that this structure is inherited by the internal language of p-**Coalg**.

---

[2]The reader can check that given only $(P,I,U) \colon A \to B$ and $(Q,J,V) \colon B \to C$, one *can* construct a composed parameter set $P \times Q$, and one *can* construct a composed implement function $A \times P \times Q \to C$, but one *cannot* construct an associative update operation $A \times C \times P \times Q \to P \times Q$. In order to get it, one needs the request function $B \times C \times Q \to B$. By endowing morphisms with the request function, as in **Learn** (see (2)), composition and a monoidal structure is easily defined.

The point is to consider logical propositions that can be stated in the internal language of p-**Coalg** and to use these propositions in order to constrain the behavior of learners and game-players (categorified as discussed above), and of interaction patterns between dynamical systems more generally. For example, "gradient descent and backpropagation" is a property we can express in the internal language. Note that the term *language* in the title refers to the internal language of the topos p-**Coalg**, which can be thought of as a language for specifying or constraining learning algorithms or dynamic organizational patterns more generally.

Plan for the paper

In Section 2 we will discuss various relevant constructions in the category **Poly** of polynomial functors in one variable. In particular, we will review its symmetric monoidal closed structure (**Poly**, $y, \otimes, [-, -]$), its composition monoidal structure $(y, \triangleleft)$, and the notion of coalgebras. We explain how morphisms in **Learn** can be phrased in terms of coalgebras on internal hom objects, and we reconstruct **Para**(**SLens**) in these terms.

In Section 3, we first show that the category p-**Coalg** of p-coalgebras for the endofunctor $p\colon$ **Set** $\to$ **Set** is a presheaf topos. We then discuss the internal logic of p-**Coalg**, and we conclude by giving several directions for future work.

Notation

We denote the category of sets by **Set**; we generally denote sets with upper-case letters A, B, etc. Given a natural number $N \in \mathbb{N}$, we write $\mathsf{N} := \{1, \ldots, N\}$, so $0 = \varnothing$, $1 = \{1\}$, $2 = \{1, 2\}$, etc. Given sets A, B, we often write $AB := A \times B$ to denote their Cartesian product. We will denote polynomials with lower-case letters, p, q, etc.

Acknowledgments

2 Constructions in Poly

In this section we review the category **Poly**, for which [4] is an excellent reference; we also discuss its symmetric monoidal closed structure. Then we discuss polynomial coalgebras and reconceptualize the category **Learn** and Gavranović's bicategorical variant, in the language of coalgebras.

2.1 Background on Poly as a monoidal closed category

For any set A, let $y^A\colon$ **Set** $\to$ **Set** be the functor *represented* by A; that is, y^A applied to a set S is **Set**$(A, S) \cong S^A$. In particular, $y := y^1$ is (isomorphic to) the identity functor $S \mapsto S$ and $1 := y^0$ is the constant functor $S \mapsto 1$. Note that $y^A(1) \cong 1^A \cong 1$ for any A.

The coproduct of functors F and G, denoted $F + G$, is taken pointwise; this means there is a natural isomorphism

$$(F+G)(S) \cong F(S) + G(S)$$

where the coproduct $F(S) + G(S)$ is taken in **Set**. Similarly, for any set I and functors F_i, one for each $i \in I$, their coproduct is computed pointwise

$$\left(\sum_{i \in I} F_i\right)(S) \cong \sum_{i \in I} F_i(S).$$

Definition 2.1. A *polynomial functor p* is any coproduct

$$p := \sum_{i \in I} y^{p[i]}$$

of representable functors, where $I \in$ **Set** and each $p[i] \in$ **Set** are sets. We denote the category of polynomial functors and natural transformations between them by **Poly**.

We note that if $p = \sum_{i \in I} y^{p[i]}$ then $p(1) \cong I$; hence we can write any $p \in$ **Poly** in canonical form

$$p \cong \sum_{i \in p(1)} y^{p[i]}. \tag{3}$$

We refer to each $i \in p(1)$ as a *position* in p and to each $d \in p[i]$ as a *direction* at i.

Example 2.2. We can consider any set S as a *constant* polynomial $\sum_{s \in S} y^0$.

We can consider a polynomial $p \in$ **Poly** as a set (or discrete category) $p(1)$ equipped with a functor $p[-]: p(1) \to$ **Set**. Then a map of polynomials $\varphi: p \to q$ can be identified with a diagram as follows

$$
\begin{array}{ccc}
p(1) & \xrightarrow{\quad \varphi_1 \quad} & q(1) \\
& \Longleftarrow & \\
{}^{p[-]}\searrow & {}^{\varphi^\sharp} & \swarrow {}^{q[-]} \\
& \mathbf{Set} &
\end{array}
\tag{4}
$$

That is, φ can be decomposed into a function $\varphi_1: p(1) \to q(1)$ on positions, and for every $i \in p(1)$ with $j := \varphi_1(i)$, a component function $\varphi_i^\sharp: q[j] \to p[i]$ on directions. This follows from the Yoneda lemma and the universal property of coproducts. We will sometimes use this $(\varphi_1, \varphi^\sharp): p \to q$ notation below.

Example 2.3. A morphism $A_1 y^{A_2} \to B_1 y^{B_2}$ can be identified with a function $\varphi_1: A_1 \to B_1$ and a function $\varphi^\sharp: A_1 \times B_2 \to A_2$. That is,

$$\mathbf{Poly}(A_1 y^{A_2}, B_1 y^{B_2}) \cong B_1^{A_1} A_2^{A_1 B_2}. \tag{5}$$

Proposition 2.4. *The composite of polynomial functors $p, q \in$ **Poly**, which we denote $p \triangleleft q$, is again polynomial with formula*

$$p \triangleleft q \cong \sum_{i \in p(1)} \sum_{j: \, p[i] \to q(1)} y^{\sum_{d \in p[i]} q[jd]} \tag{6}$$

*The composition operation $\triangleleft$ is a (nonsymmetric) monoidal structure on **Poly**, with unit y.*

Proof. See page 25. $\qquad\qquad\square$

Example 2.5. If $p = y^2$ and $q = y + 1$, then $p \triangleleft q \cong y^2 + 2y + 1$ whereas $q \triangleleft p \cong y^2 + 1$.

Example 2.6. Applying a polynomial p to a set S is given by composition: $p(S) \cong p \triangleleft S$.

Proposition 2.7 (The symmetric monoidal category $(\mathbf{Poly}, y, \otimes)$). *The category* $\mathbf{Poly}$ *has a symmetric monoidal structure with unit* y *and monoidal product* $\otimes$ *on objects given by the following formula*

$$p \otimes q := \sum_{i \in p(1)} \sum_{j \in q(1)} y^{p[i] \times q[j]}$$

Proof. See page 25.   □

Proposition 2.8 (Internal hom $[-,-]$). *The* $\otimes$ *monoidal structure on* $\mathbf{Poly}$ *is closed; that is, for every* $p, q \in \mathbf{Poly}$ *there is a polynomial*

$$[p,q] := \sum_{\varphi \colon p \to q} y^{\sum_{i \in p(1)} q[\varphi_1(i)]} \tag{7}$$

for which we have a natural isomorphism

$$\mathbf{Poly}(r \otimes p, q) \cong \mathbf{Poly}(r, [p,q]). \tag{8}$$

Proof. See page 25.   □

Example 2.9. Given sets A and B, we use Eqs. (5) and (7) to compute that the internal hom between Ay^A and By^B is

$$[Ay^A, By^B] \cong B^A A^{AB} y^{AB}.$$

The counit of the adjunction (8) is a natural map eval: $p \otimes [p,q] \to q$ called *evaluation*. In very much the same way, it induces two sorts of morphisms we will use later:

$$[p_1, q_1] \otimes [p_2, q_2] \to [p_1 \otimes p_2, q_1 \otimes q_2] \qquad \text{and} \qquad [p,q] \otimes [q,r] \to [p,r]. \tag{9}$$

2.2   Coalgebras, generalized Moore machines, and learners

Coalgebras for endofunctors $F \colon \mathbf{Set} \to \mathbf{Set}$ form a major topic of study [1, 3, 7]. In this section we recall the definition and explain the relevance to dynamical systems (generalized Moore machines) and learners.

Definition 2.10 (Coalgebra). Given a polynomial p, a *p-coalgebra* is a pair (S, β) where $S \in \mathbf{Set}$ and $\beta \colon S \to p \triangleleft S$. A *$p$-coalgebra morphism* from (S, β) to (S', β') consists of a function $f \colon S \to S'$ such that the following diagram commutes:

$$\begin{array}{ccc} S & \xrightarrow{\ \beta\ } & p \triangleleft S \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle p \triangleleft f} \\ S' & \xrightarrow[\ \beta'\ ]{} & p \triangleleft S' \end{array} \tag{10}$$

We denote the category of p-coalgebras and their morphisms by p-$\mathbf{Coalg}$.

Proposition 2.11. *A p-coalgebra (S, β) can be identified with a map of polynomials*

$$Sy^S \to p. \tag{11}$$

Proof. One finds an isomorphism $\mathbf{Poly}(S, p \triangleleft S) \cong \mathbf{Poly}(Sy^S, p)$ by direct calculation. □

Warning 2.12. Looking at Proposition 2.11, one might be tempted to think that a map of p-coalgebras as in (10) can be identified with a commuting triangle

$$Sy^S \xrightarrow{?} S'y^{S'} \longrightarrow p$$

but this is not the case; for one thing, the marked arrow does not arise from a function $f \colon S \to S'$. The point is, (11) *can be misleading* when it comes to maps, and hence we will depart from the so-called **Para** construction for 2-cells. For us, the correct sort of map between p-coalgebras is the usual one, as shown in (10).

Proposition 2.13. *For any $p, q \in \mathbf{Poly}$ there is a functor*

$$p\text{-}\mathbf{Coalg} \times q\text{-}\mathbf{Coalg} \to (p \otimes q)\text{-}\mathbf{Coalg}$$

making $\bullet$-$\mathbf{Coalg}$ a lax monoidal functor $\mathbf{Poly} \to \mathbf{Cat}$.

Proof. See page 26. □

The relevance of coalgebras to dynamics was of interest in the earliest of references we know of, namely [1], where they are referred to as codynamics. We will proceed with our own terminology.

Definition 2.14 (Moore machine). For sets A, B, an (A, B)-*Moore machine* consists of

- a set S, elements of which are called *states*,

- a function $r \colon S \to B$, called *readout*, and

- a function $u \colon S \times A \to S$, called *update*.

It is further called *initialized* if it is equipped with an element $s_0 \in S$.

With an initialized (A, B)-Moore machine (S, r, u, s_0), we can take any A-stream $a \colon \mathbb{N} \to A$ and produce a B-stream $b \colon \mathbb{N} \to B$ inductively using the formula

$$s_{n+1} := u(s_n, a_n) \qquad \text{and} \qquad b_n := r(s_n).$$

Proposition 2.15. *An (A, B)-Moore machine with states S can be identified with a map of polynomials $Sy^S \to By^A$, and hence with a By^A-coalgebra $S \to By^A \triangleleft S$ by Proposition 2.11.*

Proof. The identification uses $\varphi_1 := r$ and $\varphi^\sharp := u$. □

Replacing By^A with an arbitrary polynomial $p \in \mathbf{Poly}$, we think of p-coalgebras as generalized Moore machines. We will refer to them as *p-dynamical systems* and call p the *interface*. Mathematically, given $\beta \colon S \to p \triangleleft S$, we also get the two-fold composite

$$S \xrightarrow{\beta} p \triangleleft S \xrightarrow{p \triangleleft \beta} p \triangleleft p \triangleleft S$$

and indeed the *n-fold composite* $S \to p^{\triangleleft n} \triangleleft S$ for any $n \in \mathbb{N}$. The idea is that for every state $s \in S$, we get a position $r(s) \in p(1)$, and for every direction $d \in p[r(s)]$ there, we get a new state $u(s, d)$. We thus think of p as an interface for the dynamical system: $p(1)$ says what the world can see about the current state—i.e. its outward position $i := r(s)$—and $p[i]$ says what sort of forces or inputs the state can be subjected to.

A map of polynomials $\varphi \colon p \to p'$ is a change of interface. We can transform a p-dynamical system into a p'-dynamical system that has the same set of states. Indeed, simply compose any $S \to p \triangleleft S$ with $\varphi \triangleleft S \colon p \triangleleft S \to p' \triangleleft S$.

More generally, a map $\varphi \colon p_1 \otimes \cdots \otimes p_k \to p'$ allows us to take k-many dynamical systems $S_1 \to p_1 \triangleleft S_1$ through $S_k \to p_k \triangleleft S_k$ and use Proposition 2.13 to combine them into a single dynamical system

$$S \to (p_1 \otimes \cdots \otimes p_k) \triangleleft S \xrightarrow{\varphi} p' \triangleleft S$$

with interface p' and states $S := S_1 \times \cdots \times S_k$.

Example 2.16. Wiring diagrams are one way of combining dynamical systems as above.

$$\tag{12}$$
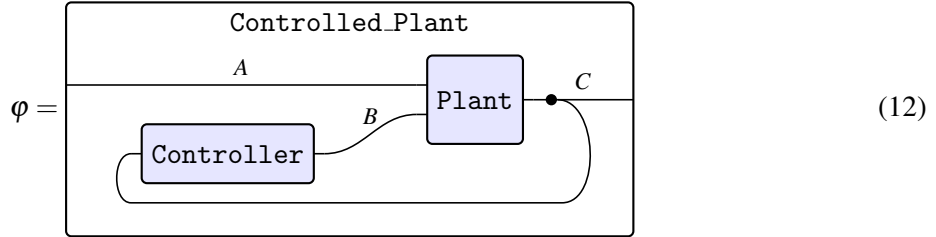
In the wiring diagram (12) three boxes are shown: the controller, the plant, and the system; we can consider each as having a monomial interface:

$$\mathtt{Plant} = Cy^{AB} \qquad \mathtt{Controller} = By^C \qquad \mathtt{Controlled\_Plant} = Cy^A. \tag{13}$$

The wiring diagram itself represents a morphism

$$\varphi \colon Cy^{AB} \otimes By^C \to Cy^A$$

in $\mathbf{Poly}$. Defining φ requires a function $\varphi_1 \colon C \times B \to C$ and a function $\varphi^{\sharp} \colon C \times B \times A \to A \times B \times C$; the first is the projection and the second is an isomorphism. Together these simply say how the wiring diagram shuttles information within the controlled plant. Indeed, the wiring diagram lets us put together dynamics of the controller and the plant to give dynamics for the controlled plant. That is, given Moore machines $Sy^S \to \mathtt{Plant}$ and $Ty^T \to \mathtt{Controller}$, we get a Moore machine $STy^{ST} \to \mathtt{Controlled\_Plant}$.

More generally, we can think of transistors in a computer as dynamical systems, and the logic gates, adder circuits, memory circuits, a connected keyboard or monitor, etc. each as a wiring diagram comprising these simpler systems.

Example 2.17. In Example 2.16 the wiring pattern is fixed, but as we show in [11], **Poly** also supports wiring diagrams for dynamical systems that can change their interaction pattern based on their internal states.

We now come to learners. As mentioned in the introduction, the category **Learn** from [9] is better understood as a bicategory which we will denote as $\mathbb{L}$**earn**. Its objects are sets, $\mathrm{Ob}(\mathbb{L}\mathbf{earn}) = \mathrm{Ob}(\mathbf{Set})$, a 1-morphism (learner) from A to B consists of a set P and maps $I\colon A \times P \to B$ and $(R,U)\colon A \times B \times P \to A \times P$, and a 2-morphism—a morphism between learners—is a function $f\colon P \to P'$ making the following squares commute:

$$
\begin{array}{ccc}
A \times P & \xrightarrow{\ I\ } & B \\
{\scriptstyle A \times f}\downarrow & & \| \\
A \times P' & \xrightarrow[I']{} & B
\end{array}
\qquad
\begin{array}{ccc}
A \times B \times P & \xrightarrow{\ (R,U)\ } & A \times P \\
{\scriptstyle A \times B \times f}\downarrow & & \downarrow{\scriptstyle A \times f} \\
A \times B \times P' & \xrightarrow[(R',U')]{} & A \times P'
\end{array}
\tag{14}
$$

We denote the category of learners from A to B as $\mathbb{L}$**earn**$(A,B) \in \mathbf{Cat}$.

Proposition 2.18. *For sets A,B, there is an equivalence of categories*

$$\mathbb{L}\mathbf{earn}(A,B) \cong [Ay^A, By^B]\text{-}\mathbf{Coalg}.$$

Proof. See page 26. □

We will now give a definition that generalizes the bicategory $\mathbb{L}$**earn**, give examples, and discuss intuition. In particular, we define a category-enriched operad $\mathbb{S}$**ys**, which includes $\mathbb{L}$**earn** as a full subcategory.

Definition 2.19 (The operad $\mathbb{S}$**ys**). We define $\mathbb{S}$**ys** to be the category-enriched operad defined as follows. The objects of $\mathbb{S}$**ys** are polynomials: $\mathrm{Ob}(\mathbb{S}\mathbf{ys}) := \mathrm{Ob}(\mathbf{Poly})$. For objects $p_1, \ldots, p_k, p'$, the category of maps between them is defined by

$$\mathbb{S}\mathbf{ys}(p_1, \ldots, p_k; p') := [p_1 \otimes \cdots \otimes p_k, p']\text{-}\mathbf{Coalg}.$$

For any object p, the identity on p is given by the $[p,p]$-coalgebra $1 \to [p,p](1) \cong \mathbf{Poly}(p,p)$ that sends $1 \mapsto \mathrm{id}_p$.

Given objects $p_{1,1}, \ldots, p_{1,j_1}, \ldots, p_{k,1}, \ldots, p_{k,j_k}$, the composition functor

$$[p_{1,1} \otimes \cdots \otimes p_{1,j_1}, p_1]\text{-}\mathbf{Coalg} \times \cdots \times [p_{k,1} \otimes \cdots \otimes p_{k,j_k}, p_k]\text{-}\mathbf{Coalg}$$
$$\times [p_1 \otimes \cdots \otimes p_k, p']\text{-}\mathbf{Coalg} \to [p_{1,1} \otimes \cdots \otimes p_{k,j_k}, p']\text{-}\mathbf{Coalg}$$

is given by repeated application of the maps in (9) and Proposition 2.13.

How do we think of a morphism $(S, \beta)\colon (p_1, \ldots, p_k) \to p'$ in $\mathbb{S}$**ys**? It is a dynamical system which has a set S of states. For every state $s \in S$, we can read out an associated element $\beta_1(s)\colon p_1 \otimes \cdots \otimes p_k \to p'$; we can think of this as a wiring diagram as in Example 2.16 or a generalization thereof. That is, the current state s dictates an organization pattern $\beta_1(s)$: how outputs of the internal systems are aggregated and output from the outer interface, and how feedback from outside is distributed internally.

But so far, this is only the readout of β. What's an input? An input to this system consists of a tuple of outputs $i \coloneqq (i_1, \ldots, i_k) \in p_1(1) \times \cdots \times p_k(1)$, one output for each of the internal systems, together with an input $d \in p'[\beta_1(i)]$ to the outer system.

Imagine you're the officer in charge of an organization: you're in charge of the system by which your employees and other resources are arranged, how they send information to each other and the outside world, and how the feedback from the outside world is disbursed to the employees and resources. You see what they do, you see how the world responds, and you update your internal state and hence the system itself, however you see fit. In this image, you as the officer are playing the role of (S, β), i.e. a morphism in $\mathbb{S}\mathbf{ys}$. But even a simple logic gate or adder circuit in a computer—something that doesn't have a changing internal state or update how resources are connected—counts as a morphism in $\mathbb{S}\mathbf{ys}$. Again, the only difference in that case is that the state set $S \cong 1$, the way the internal resources are connected—is unchanged by inputs.

Example 2.20. For any operad, there is an algebra of 0-ary morphisms. In the case of $\mathbb{S}\mathbf{ys}$, this algebra sends $p \mapsto p\text{-}\mathbf{Coalg}$, the category of dynamical systems on p, since the unit of $\otimes$ is y and $[y, p] \cong p$.

Next we will give a mathematical language for describing dynamical systems as in Example 2.20 as well as the generalized learners (or officers) described above.

3   Toposes of learners

We ended the previous section by defining the (category-enriched) operad $\mathbb{S}\mathbf{ys}$ and explaining how it generalizes the bicategory $\mathbb{L}\mathbf{earn}$. In this section we mainly discuss the internal language for each learner. That is, given $p, p' \in \mathrm{Ob}(\mathbf{Poly}) = \mathrm{Ob}(\mathbb{S}\mathbf{ys})$, where perhaps $p = p_1 \otimes \cdots \otimes p_k$, we discuss the category $\mathbb{S}\mathbf{ys}(p; p')$ of such learners.

Our first job is to show that every such category is a topos; this will give us access to the Mitchell-Benabou language and Kripke-Joyal semantics—the so-called *internal language* of the topos and its interpretation [2]. We then explain the sorts of things—propositions—that one can express in this language, e.g. the proposition "I will follow the gradient descent algorithm" is a particular case.

3.1   The topos of p-coalgebras

In this section, we show that for any polynomial p, there is a category $\mathcal{C}_p$, called the *cofree category on p*, for which we can find an equivalence

$$p\text{-}\mathbf{Coalg} \cong \mathcal{C}_p\text{-}\mathbf{Set}$$

between p-coalgebras and functors $\mathcal{C}_p \to \mathbf{Set}$. In fact, the category $\mathcal{C}_p$ is free on a graph, making it quite easy to understand in certain respects.[3]

Following [12], we define a *rooted tree* to be a graph T whose free category has an initial object, called the *root*; the idea is that for any node n, there is exactly one path from the root to n. We denote the nodes of T by T_0, the root by $\mathrm{root}_T \in T_0$, and for any node $n \in T_0$ we denote the set of arrows emanating

---

[3]The name "cofree category" comes from the fact that—up to isomorphism—comonoids in **Poly** are categories; see [5]. So we're really taking the cofree comonoid on p.
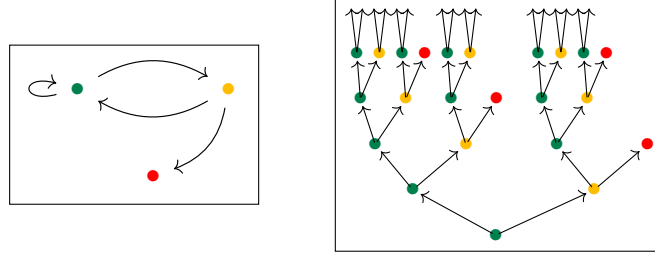
Figure 1: Left: a dynamical system, i.e. coalgebra, for the polynomial $p := \{\bullet, \bullet\}y^2 + \bullet \cong 2y^2 + 1$. Right: the p-tree corresponding to the node $\bullet$.

from n by $T[n]$. Note that at the target n' of any arrow $a \in T[n]$, there sits another rooted tree with root n'; we denote this tree by $\mathrm{cod}_T(a)$.

Definition 3.1 (The graph Tree_p of p-trees). For a polynomial $p \in \mathbf{Poly}$, define a p-*tree* to be a tuple $(T, \phi_1, \phi^\sharp)$, where T is a rooted tree, $\phi_1 : T_0 \to p(1)$ is a function called the *position* function, and $\phi_n^\sharp$ is a bijection
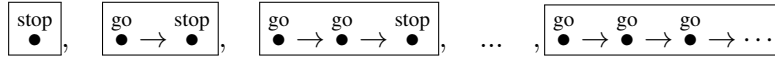
$$\phi_n^\sharp : p[\phi_1(n)] \xrightarrow{\cong} T[n]$$

for each node $n \in T_0$, identifying the set of branches in the tree T at node n with the set of directions in the polynomial p at the position $\phi_1(n)$.

We denote by Tree_p the graph whose vertex set is the set of p-trees, and for which an arrow $a : T \to T'$ is a branch $a \in T[\mathrm{root}_T]$ with $T' = \mathrm{cod}_T(a)$.

Example 3.2. If $p = y^A$ for a nonempty set A then there is only one p-tree: each node has the unique label $p(1) \cong 1$ and A-many branches.

If $p = \{\mathrm{go}\}y^1 + \{\mathrm{stop}\}y^0 \cong y + 1$ then counting the number of nodes gives a bijection between set of p-trees and the set $\mathbb{N} \cup \{\infty\}$

$$\boxed{\begin{matrix} \mathrm{stop} \\ \bullet \end{matrix}} , \quad \boxed{\begin{matrix} \mathrm{go} & & \mathrm{stop} \\ \bullet & \to & \bullet \end{matrix}} , \quad \boxed{\begin{matrix} \mathrm{go} & & \mathrm{go} & & \mathrm{stop} \\ \bullet & \to & \bullet & \to & \bullet \end{matrix}} , \quad \dots \quad , \boxed{\begin{matrix} \mathrm{go} & & \mathrm{go} & & \mathrm{go} \\ \bullet & \to & \bullet & \to & \bullet & \to & \cdots \end{matrix}}$$

Theorem 3.3. *For any polynomial p there is an equivalence of categories*

$$p\text{-}\mathbf{Coalg} \cong \mathbf{Tree}_p\text{-}\mathbf{Set}$$

where $\mathbf{Tree}_p$ *is the free category on the graph* Tree_p *of p-trees.*

Proof. See page 27. □

3.2 The internal language of p-**Coalg**

For any category $\mathcal{C}$, the category $\mathcal{C}$-**Set** of functors $\mathcal{C} \to \mathbf{Set}$ forms a topos. In particular, this means that mathematicians have already developed a language and logic that faithfully represents the structures of $\mathcal{C}$-**Set**, and we can import it wholesale; see [8, Chapter 7] or [2, Chapter VI]. Now that we know from Theorem 3.3 that p-**Coalg** is a topos for any $p \in \mathbf{Poly}$, we are interested in corresponding language

for the topos $\mathbb{L}\mathbf{earn}(A,B) = [Ay^A, By^B]\text{-}\mathbf{Coalg}$ of learners, for any sets A, B; hence the title "learners' languages." However since most of the relevant abstractions work more generally for $p\text{-}\mathbf{Coalg}$, we will mainly work there.

Not assuming the reader knows topos theory, we will proceed as though we are defining the few relevant concepts from scratch, when in actuality we are merely "reading them off" from the established literature. For example Definition 3.4 simply unpacks the topos-theoretic definition of a logical proposition as a subobject of the terminal object in the topos $p\text{-}\mathbf{Coalg}$.

Definition 3.4. A *logical proposition (about p-coalgebras)* is defined to be a set $P \subseteq \mathsf{Tree}_p$ of p-trees satisfying the condition that if $T \in P$ is a tree in P, then for any direction $d \in T[\mathrm{root}_T]$, the tree $\mathrm{cod}_T(d) \in P$ is also a tree in P.

Proposition 3.5 gives us an easy way to construct logical propositions about p-coalgebras, and hence learners. Namely, it says if we put a condition on the p-positions that can show up as labels, and if we put a condition on the codomain map (how directions in the tree lead to new positions), we get a logical proposition. Of course, these aren't the only ones, but they form a nice special case.

Recall from Definition 3.1 that a p-tree is a rooted tree T equipped with a position function $\phi_1 \colon T_0 \to p(1)$; we elide the bijections (earlier denoted $\phi_n^\sharp \colon T[n] \cong p[\phi_1(n)]$) in what follows.

Proposition 3.5. *Given $p \in \mathbf{Poly}$, suppose given subsets*

$$Q \subseteq p(1) \qquad and \qquad R \subseteq \prod_{i \in Q} \prod_{d \in p[i]} Q.$$

Then the following set of trees is a logical proposition:

$$P_Q^R := \{T \in \mathsf{Tree}_p \mid \forall(i : T_0).\phi_1(i) \in Q \land \forall(d : T[i]).\mathrm{cod}_T(d) \in (Rid)\}.$$

Proof. The result is immediate from Definition 3.4. $\square$

Example 3.6 (Gradient descent). The gradient descent, backpropagation algorithm used by each "neuron" in a deep learning architecture can be phrased as a logical proposition about learners. The whole learning architecture is then put together as in [9], or as we've explained things above, using the operad $\mathbb{S}\mathbf{ys}$ from Definition 2.19.

So suppose a neuron is tasked with learning a function $\mathbb{R}^m \to \mathbb{R}^n$, and it has a parameter space $\mathbb{R}^k$, i.e. we are given a smooth function $f \colon \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}^n$. We will define a corresponding logical proposition using Proposition 3.5. Define $p \in \mathbf{Poly}$ by

$$p := [\mathbb{R}^m y^{\mathbb{R}^m}, \mathbb{R}^n y^{\mathbb{R}^n}] \cong \sum_{g \colon \mathbb{R}^m y^{\mathbb{R}^m} \to \mathbb{R}^n y^{\mathbb{R}^n}} y^{\mathbb{R}^m \times \mathbb{R}^n}.$$

Define $Q \subseteq \{(g_1, g^\sharp) \mid g_1 \colon \mathbb{R}^m \to \mathbb{R}^n, g^\sharp \colon \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m\}$ by saying that $g_1(x)$ must be of the form $f(a, x)$ for some $a \in \mathbb{R}^k$ in the parameter set and that $g_x^\sharp$ is given by "pulling back gradient vectors" using the map on cotangent spaces defined by composing with the derivative of g_1 at x, in the usual way.

Now given $(g_1, g^\sharp) \in Q$, we continue with the setup of Proposition 3.5. That is, we define $R(g_1, g^\sharp) \colon \mathbb{R}^m \times \mathbb{R}^n \to Q$ to say how the learner updates its current parameter value $a \in \mathbb{R}^k$ given an input-output pair; this again is specified by the deep learning algorithm. Typically, it uses a loss function to calculate a

cotangent vector at $f(a,x)$ which is passed back to a cotangent vector at a, and a dual vector of some "learning rate" ε is traversed.

The details are important for implementation, but not for understanding the idea. The idea is that as long as we say what sorts of maps are allowed (smooth maps with reverse derivatives) and how they update, we have defined a logical proposition.

The logical propositions that come from Proposition 3.5 are very special. More generally, one could have a logical proposition like "whenever I receive two red tokens within three seconds, I will wait five seconds and then send either three blue tokens or two blues and six reds." As long as this behavior has the "whenever" flavor—more precisely as long as it satisfies the condition in Definition 3.4—it will be a logical proposition in the topos.

3.3 Future work

There are many avenues for future work. One is to give a richer language—beyond the logical symbols `true`, `false`, $\wedge, \vee, \Rightarrow, \neg, \forall, \exists$ that exist in any topos—for building logical propositions in the p-**Coalg** toposes specifically. Another is to understand various modalities in these toposes.

The sort of morphisms between toposes that seem to arise most naturally in this context are not the usual kind—adjoint functors $\mathcal{E} \leftrightarrows \mathcal{E}'$ for which the left adjoint preserves all finite limits, called *geometric morphisms*—but instead adjoint functors $\mathcal{E} \leftrightarrows \mathcal{E}'$ for which the left adjoint preserves all *connected limits*. Thus another avenue for future research is to consider how logical and type-theoretic statements move between toposes that are connected in this way.

A   Proofs

Proof of Proposition 2.4. It is well-known that composition of functors is a monoidal operation, so it suffices to see that the polynomial (6) is the composite of functors p, q. To show this, we use the fact that for any set A we have a bijection $y^A \cong \prod_{a \in A} y$ to calculate the composite

$$
\begin{aligned}
p \triangleleft q &\cong \sum_{i \in p(1)} \prod_{d \in p[i]} y \triangleleft \sum_{j \in q(1)} \prod_{e \in q[j]} y \\
&\cong \sum_{i \in p(1)} \prod_{d \in p[i]} \sum_{j \in q(1)} \prod_{e \in q[j]} y \\
&\cong \sum_{i \in p(1)} \sum_{j \colon p[i] \to q(1)} \prod_{d \in p[i]} \prod_{e \in q[j(d)]} y \cong \sum_{i \in p(1)} \sum_{j \colon p[i] \to q(1)} y^{\sum_{d \in p[i]} q[jd]}
\end{aligned}
$$

where the first isomorphism is (3), the second is substitution, the third is the distributive law, and the fourth is properties of exponents. $\qquad \square$

Proof of Proposition 2.7. With the formula given, it is clear that the $\otimes$-operation is associative (up to isomorphism), and that y, which has $y(1) \cong 1$ and $y[1] \cong y$, is a unit. One can also check that the formula is functorial in p and q, completing the proof. $\qquad \square$

Proof of Proposition 2.8. The natural isomorphism is given by rearranging terms:

$$\mathbf{Poly}(r\otimes p,q)\cong \prod_{k\in r(1)}\prod_{i\in p(1)}\sum_{j\in q(1)}\prod_{e\in q[j]} r[k]\times p[i]$$

$$\cong \prod_{k\in r(1)}\sum_{\varphi_1\colon p(1)\to q(1)}\prod_{i\in p(1)}\prod_{e\in q[\varphi_1(i)]} r[k]\times p[i]$$

$$\cong \prod_{k\in r(1)}\sum_{\varphi_1\colon p(1)\to q(1)}\left(\prod_{i\in p(1)}\prod_{e\in q[\varphi_1(i)]} p[i]\right)\times\left(\prod_{i\in p(1)}\prod_{e\in q[\varphi_1(i)]} r[k]\right)$$

$$\cong \prod_{k\in r(1)}\sum_{\varphi\colon p\to q}\prod_{i\in p(1)}\prod_{e\in q[\varphi_1(i)]} r[k]$$

$$\cong \mathbf{Poly}\left(r,\sum_{\varphi\colon p\to q} y^{\sum_{i\in p(1)}q[\varphi_1(i)]}\right)\cong \mathbf{Poly}(r,[p,q]).$$

In order, these isomorphisms are given by: unfolding the definition of morphisms in **Poly**, distributivity, products commuting with products, definition of morphisms in **Poly**, rules of exponents, and the definition of $[p,q]$ from Eq. (7), respectively. $\square$

Proof of Proposition 2.13. We need to give not only the functor $\lambda\colon p\text{-}\mathbf{Coalg}\times q\text{-}\mathbf{Coalg}\to (p\otimes q)\text{-}\mathbf{Coalg}$ for any $p,q\in\mathbf{Poly}$, but also a functor $\{1\}\to y\text{-}\mathbf{Coalg}$, which we can identify with a y-coalgebra; we take the latter to be the unique function $1\to y\triangleleft 1$. For the former, one could proceed abstractly using the fact that there is a duoidal structure on **Poly**

$$(p\triangleleft s)\otimes(q\triangleleft t)\to (p\otimes q)\triangleleft(s\otimes t).$$

Indeed, since for sets S,T we have $S\otimes T\cong S\times T$, the result will follow from the properties of duoidal structures (applied in the case where $s\coloneqq S$ and $t\coloneqq T$ are constant polynomials). However, for the reader's convenience, we will give the map $\lambda\colon p\text{-}\mathbf{Coalg}\times q\text{-}\mathbf{Coalg}\to(p\otimes q)\text{-}\mathbf{Coalg}$ more explicitly.

Given $\beta\colon S\to p\triangleleft S$ and $\gamma\colon T\to q\triangleleft T$, we define a function

$$ST\to (p\otimes q)\triangleleft(ST)\cong \sum_{i\in p(1)}\sum_{j\in q(1)}(ST)^{p[i]\times q[j]}$$

$$(s,t)\mapsto \big(i\coloneqq\beta_1(s),j\coloneqq\gamma_1(t),(d,e)\mapsto(\beta_i^\sharp(d),\gamma_j^\sharp(e))\big).$$

This is natural in S,T, which makes λ a functor for any p,q. One can check that all the axioms of a lax monoidal functor are satisfied, in the sense that the required diagrams commute up to natural isomorphism. $\square$

Proof of Proposition 2.18. On one hand, an object in $\mathbb{L}\mathbf{earn}(A,B)$ as described in (2) consists of a set P and functions $A\times P\to B$ and $A\times B\times P\to A$ and $A\times B\times P\to P$. On the other hand, we have $[Ay^A,By^B]\cong B^A A^{AB}y^{AB}$ by Example 2.9, so a coalgebra $P\to[Ay^A,By^B]\triangleleft P$ consists of a function $P\to B^A$, a function $P\to A^{AB}$, and a function $P\to P^{AB}$. The two descriptions can be identified by currying. The $[Ay^A,By^B]$-**Coalg** morphisms

$$
\begin{array}{ccc}
P & \longrightarrow & B^A A^{AB} P^{AB}\\
\downarrow & & \downarrow\\
P' & \longrightarrow & B^A A^{AB}(P')^{AB}
\end{array}
$$

are easily seen to coincide with those shown in (14). □

Proof of Theorem 3.3. It is well-known that $\mathcal{C}$-**Set** is equivalent to the category of discrete opfibrations over $\mathcal{C}$ via the category-of-elements construction. When $\mathcal{C}$ is free on a graph G, the category of elements for any functor $h\colon \mathcal{C} \to$ **Set** is also free on a graph, say H. In this case the opfibration can be identified with a graph homomorphism $\pi\colon H \to G$ with the property ("opfib") that for any vertex $h \in H$, the function on arrows $H[h] \xrightarrow{\cong} G[\pi(h)]$ induced by π is a bijection. Under this correspondence, a morphism $h \to h'$ of copresheaves is identified with a graph homomorphism $f\colon H \to H'$ for which $\pi = \pi' \circ f$.

Thus it suffices to show that there is an equivalence between p-**Coalg** and the category of those graph homomorphisms $\pi\colon H \to \mathsf{Tree}_p$ that have the opfib property. Suppose given a p-coalgebra $\beta\colon S \to p \triangleleft S$; it includes a function $\beta_1\colon S \to p(1)$ and for each $s \in S$ a function $\beta_s^\sharp\colon p[\beta_1(s)] \to S$. We define the corresponding graph $G_{S,\beta}$ to have vertex set S, and each $s \in S$ to have $p[\beta_1(s)]$-many outgoing arrows; the target of each outgoing arrow $d \in p[\beta_1(s)]$ is defined to be $\beta_s^\sharp(d)$. The graph homomorphism $\pi\colon G_{S,\beta} \to \mathsf{Tree}_p$ is defined inductively: for any $s \in S$, the p-tree $\pi(s)$ has root labeled $\beta_1(s)$, and for each outgoing branch $d \in p[\beta_1(s)]$ the target vertex is assigned the label $\beta_1(s')$, where $s' := \beta_s^\sharp(d)$, and for each outgoing branch $d' \in p[\beta_1(s')]$ the target vertex is assigned the label $\beta_1(s'')$ where $s'' := \beta_{s'}^\sharp(d')$, and so on. It is clear that π satisfies the opfib property, since it assigns to each vertex s in the graph $G_{S,\beta}$ a vertex in Tree_p (the p-tree $\pi(s)$) with the same set $p[\beta_1(s)]$ of outgoing arrows.

Conversely, given a graph homomorphism $\pi\colon G \to \mathsf{Tree}_p$ with the opfib property, let S_G be the set of vertices in G. The required coalgebra map $\beta\colon S_G \to p \triangleleft S_G$ consists of a function $\beta_1\colon S_G \to p(1)$ and a function $\beta_s^\sharp\colon p[\beta_1(s)] \to S_G$ for every $s \in S_G$. We take the function β_1 to send vertex s to the root label $\phi(\mathsf{root}_{\pi(s)})$ for tree $\pi(s)$. Since we have a bijection $p[\beta_1(s)] \cong G[s]$, we can take $\beta_s^\sharp$ to simply be the target function $G[s] \to S_G$ for the graph G.

It is a straightforward calculation to check that these two constructions are mutually inverse, and to check that graph homomorphisms over Tree_p correspond bijectively to morphisms of p-coalgebras. □

References

[1]   Michael A Arbib and Ernest G Manes. "Parametrized data types do not need highly constrained parameters". In: *Information and Control* 52.2 (1982), pp. 139–158. DOI: 10.1016/S0019-9958(82)80026-0.

[2]   Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Springer, 1992.

[3]   Jırı Adámek. "Introduction to coalgebra". In: *Theory and Applications of Categories* 14.8 (2005), pp. 157–199.

[4]   Nicola Gambino and Joachim Kock. "Polynomial functors and polynomial monads". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 154.1 (Sept. 2012), pp. 153–192. DOI: 10.1007/BFb0066201.

[5]   Danel Ahman and Tarmo Uustalu. "Directed Containers as Categories". In: *EPTCS 207, 2016, pp. 89-98* (2016). DOI: 10.4204/EPTCS.207.5. eprint: arXiv:1604.01187.

[6]   Neil Ghani et al. "Compositional game theory". In: *Proceedings of Logic in Computer Science (LiCS) 2018* (2016).

[7]   Bart Jacobs. *Introduction to Coalgebra*. Vol. 59. Cambridge University Press, 2017. DOI: `10.1017/CBO9781316823187`.

[8]   Brendan Fong and David I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, 2019. DOI: `10.1017/9781108668804`.

[9]   Brendan Fong, David I. Spivak, and Rémy Tuyéras. "Backprop as Functor: A compositional perspective on supervised learning". In: *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM. 2019. DOI: `10.1109/LICS.2019.8785665`. eprint: `arXiv:1711.10455`.

[10]   Bruno Gavranović. *Compositional Deep Learning*. 2019. eprint: `arXiv:1907.08292`.

[11]   David I. Spivak. *Poly: An abundant categorical setting for mode-dependent dynamics*. 2020. eprint: `arXiv:2005.01894`.

[12]   nLab authors. *tree*. `http://ncatlab.org/nlab/show/tree`. Revision 26. Feb. 2021.