Categories of Differentiable Polynomial Circuits for Machine Learning

Paul Wilson*, Fabio Zanasi

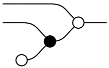
July 17, 2022

Motivation

- Narrow view: What's in the paper?
- Wide view: Why is what's in the paper in the paper?

This Paper: Narrow View

- A machine learning model class PolyCirc_S
- A graphical account of *reverse derivatives*
- A recipe to construct and extend *reverse derivative categories*
- An extension of PolyCirc to gain *functional completeness*



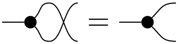
$$x_1 \;\; x_2 \mapsto x_1 \cdot (x_2 + 1)$$

Presentations by Generators and Equations

Generators (example):

Build terms with composition and tensor:

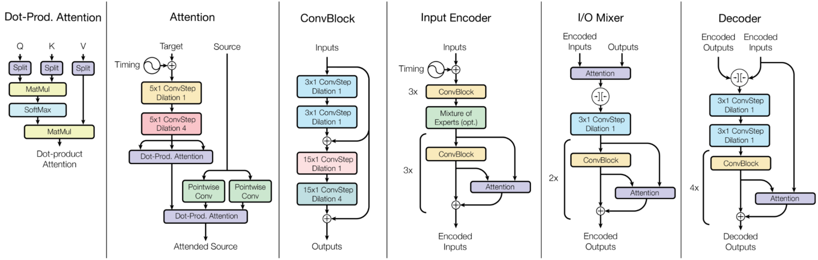
Equations (example):



This Paper: Wide View

Slogan: Machine Learning with String Diagrams

- ML papers often use diagrammatic exposition (below from [KGS+17])
- We want to make this completely formal
- Use string diagrams: gain access to lots of free theoretical tools!



Why is graphical structure important?

Morphisms of PolyCirc will represent machine learning models. We want to...

- ... *represent* terms easily on a computer [WZ21a]
- ... *manipulate* terms (rewriting/optimization) [BGK+20]
- ... *evaluate* and *compile* (to unusual targets!)
- ... *visualise* execution + model internals

Aside from this, we also have an immediate application in mind...

Application: Gradient based learning without $\mathbb{R}$ I

In 5 bullet points:

- Want to learn a function $f : \mathbb{R}^a \to \mathbb{R}^b$
- Define a model $m : \mathbb{R}^p \times \mathbb{R}^a \to \mathbb{R}^b$
- Learning: repeatedly nudge your parameters in the 'direction of best improvement'.
- Final result: parameters $\theta \in \mathbb{R}^p$
- ... giving a function $m(\theta, -) : \mathbb{R}^a \to \mathbb{R}^b$

This paper: what about for arbitrary semirings instead of $\mathbb{R}$?

Application: 'Gradient' Based Learning without $\mathbb{R}$ II

Problems with $\mathbb{R}$:

- We can't *really* represent values of $\mathbb{R}$ on a computer anyway
- Instead, we need to deal with *finite representations*
- Floating-point is relatively expensive: sometimes not available!

Another option:

- An extreme choice: use $\mathbb{Z}_2$ instead of $\mathbb{R}$ [WZ21b]
- 'Nudging an input' = flipping a bit
- We can express any function $\mathbb{B}^a \to \mathbb{B}^b$ in terms of polynomials over $\mathbb{Z}_2$ (functional completeness!)

What about other semirings S? That's where $\mathsf{PolyCirc}_S$ comes in

Summary

So we want categories which …

- … have RDC structure
- … are presented by generators and relations
- … represent a 'suitably expressive' class of models

So that we can …

- … do 'gradient' based learning
- … use computer representations to evaluate/compile them
- … define an appropriate model for a given problem

PolyCirc fits these criteria

Structure of this talk

- ~~Motivation~~
- Reverse Derivatives
- Polynomial Circuits
- Functional Completeness

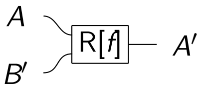
Presentation-Friendly Reverse Derivatives

- Original formulation
- What are reverse derivatives for?
- Alternative ‘presentation-friendly’ axioms
- ‘Extensibility theorem’

Reverse Derivative Categories (2019)
Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon Plotkin, Dorette Pronk

Defines categories with a reverse derivative combinator:

$$\frac{A \xrightarrow{f} B}{A \times B' \xrightarrow[\mathrm{R}[f]]{} A'}$$



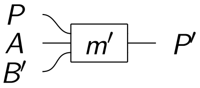
obeying some axioms **RD.1** - **RD.7**, along with some other ‘base’ structure.

Why do we need Reverse Derivatives?

Earlier we said...

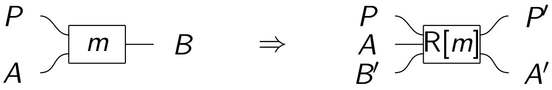
- Want to learn a map $f \colon A \to B$
- Define a model $m : P \times A \to B$
- Learning: repeatedly nudge your parameters in the 'direction of best improvement'.

We need something like this:



Reverse Derivatives

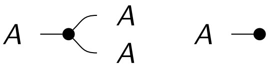
Taking the reverse derivative of our model gets us what we want:



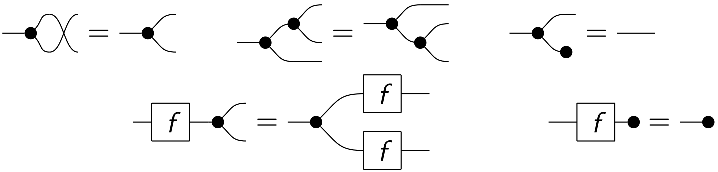
But RDCs have some required 'base' structure...

RDC Requirements I: Cartesian Structure

... means that each object A comes equipped with a *copy* and a *discard* map:



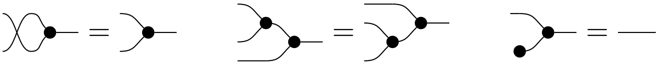
such that...



Cartesian Left Additive Structure I

A **Cartesian Left-Additive Category** ([CCG+19], [BCS09]) is a cartesian category in which each object A is equipped with a commutative monoid and zero map:

so that

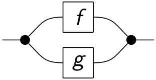


Cartesian Left Additive Structure II: Adding Morphisms

We used the ‘alternative’ definition of cartesian left-additive structure. The original has these axioms:

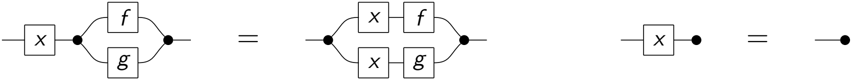
$$x \fatsemi (f + g) = (x \fatsemi f) + (x \fatsemi g) \qquad\qquad x \fatsemi 0 = 0$$

We can recover these by defining addition and zero:

$f + g :=$ 

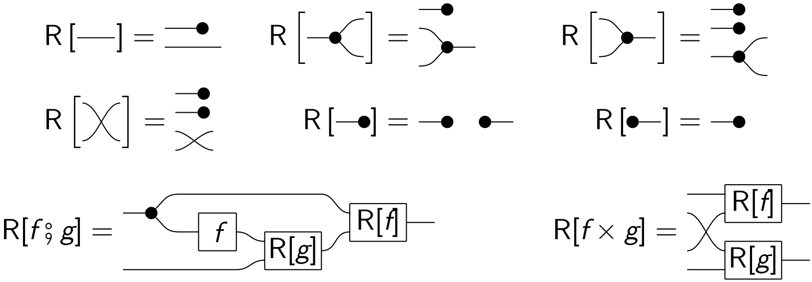
$0 :=$ 

Then the equations above can be written diagrammatically:



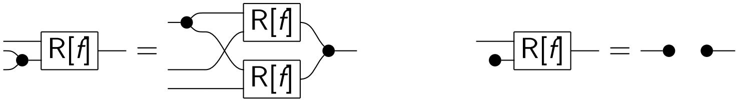
RDC Axioms I: Structural Axioms

[ARD.1] (Structural axioms, equivalent to RD.1, RD.3-5 in [CCG+19])



RDC Axioms II: Additivity of Change

[ARD.2] (Additivity of change, equivalent to RD.2 in [CCG+19])

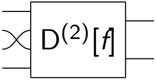


RDC Axioms III: Higher Derivatives

[ARD.3] (Linearity of change, equivalent to RD.6 in [CCG$^+$19])

$D_B[R[f]] =$ 

[ARD.4] (Symmetry of partials, equivalent to RD.7 in [CCG$^+$19])

$D^{(2)}[f] =$ 

Equivalence to Original Definition

- Original formulation had axioms **RD.1** - **RD.7**
- Our formulation has axioms **ARD.1** - **ARD.4**
- These are equivalent (Theorem 1)

Now let's use our formulation to show how to extend RDCs...

Extending RDC Presentations: A Theorem

How to extend an RDC $\mathscr{C}$ presented by generators Σ and equations E (Theorem 2):

- Add a new generator s and equations e.g. $l = r$
- Define $R[s]$
- Check R is well-defined $(\mathrm{R}[l] = \mathrm{R}[r])$
- Check R satisfies **ARD.2** - **ARD.4**

Formally:

Theorem

Let $\mathscr{C}$ be the cartesian left-additive category presented by generators (Obj, Σ) and equations E. If for each $s \in \Sigma$ there is some $\mathrm{R}[s]$ which is well-defined with respect to E, and which satisfies axioms ARD.1-4, then $\mathscr{C}$ is a reverse derivative category.

Summary

We've done this:

- Redefined the RDC axioms in a 'presentation friendly' way
- Showed how we can extend an RDC with new generators and equations

Now we can slowly build up PolyCirc from parts

Polynomial Circuits

- Definition
- Relationship to POLY_S
- Examples

Defining PolyCirc$_S$

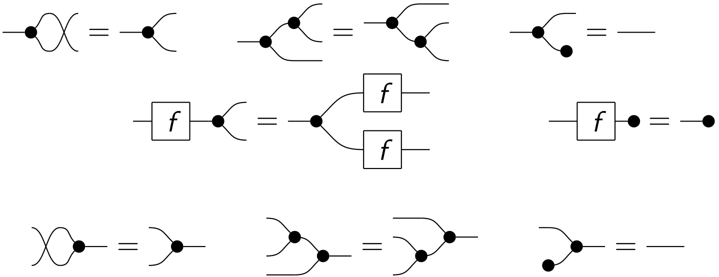
Piece-by-piece:

- Cartesian left-additive structure
- A multiplication operation
- Constants and equations

Cartesian Left Additive Structure

Generators:

Equations:

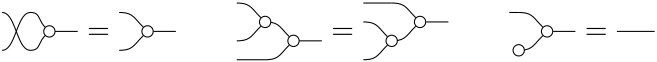


The reverse derivative is fixed by **ARD.1**

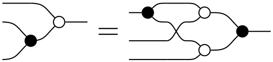
Cartesian Distributive Categories

Now add a multiplication  and 1 constant  to get a **Cartesian Distributive Category**:

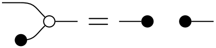
Satisfying cartesian left-additive and *multiplicativity* equations



and the *distributivity* and *annihilation* equations



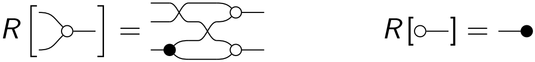
$$x_1 \cdot (x_2 + x_3) = x_1 \cdot x_2 + x_1 \cdot x_3$$



$$x_1 \cdot 0 = 0$$

Cartesian Distributive Categories II: Reverse Derivative

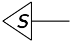
- Take an RDC
- Add the generators and equations of Cartesian Distributive categories
- Give it a reverse derivative:

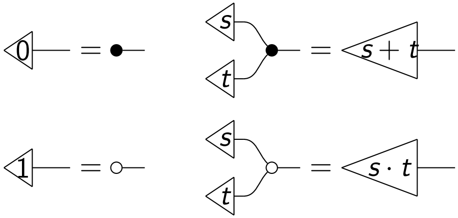


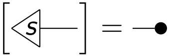
This is well-defined and satisfies ARD.1-4.

Polynomial Circuits

We define $\mathsf{PolyCirc}_S$ as the cartesian distributive category presented by:

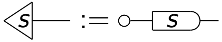
- one generating object 1 (so the objects are natural numbers)
- for each $s \in S$, a generating morphism  $: 0 \to 1$,
- the 'constant' equations (below)

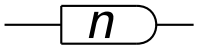


$\mathsf{PolyCirc}_S$ is an RDC with R  .

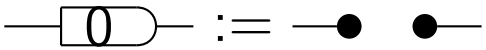
Polynomial Circuits Examples I: $\mathrm{PolyCirc}_{\mathbb{N}}$

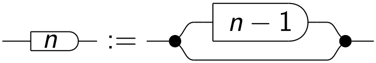
define each constant $s \in S$ as repeated addition:



where we define 

inductively as

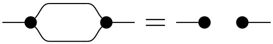




$\mathrm{PolyCirc}_{\mathbb{N}}$ is the free Cartesian Distributive Category on one generating object

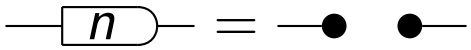
Polynomial Circuits Examples II: $\mathrm{PolyCirc}_{\mathbb{Z}_n}$

$\mathrm{PolyCirc}_{\mathbb{Z}_2}$ is the same, but we need one additional equation:



This says that $1 + 1 = 0$ (it's XOR)

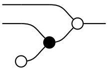
More generally for $\mathrm{PolyCirc}_{\mathbb{Z}_n}$:



$\mathrm{PolyCirc}_S$ and POLY_S

- Take a morphism $f : m \to n$ of $\mathrm{PolyCirc}_S$
- It's the same as an n-tuple of m-variable polynomials...
- i.e. an element of the free module over polynomial ring $S[x_1 \ldots x_m]^n$
- This makes $\mathrm{PolyCirc}_S \cong \mathrm{POLY}_S$ (POLY_S is from [CCG+19])

Recall our first example:



$$x_1 \;\; x_2 \mapsto x_1 \cdot x_2 + x_1$$

Except something is missing for functional completeness: we need to extend $\mathrm{PolyCirc}_S$ by adding a new operation (and we'll no longer have polynomials)

Functional Completeness

- Why do we want it?
- How do we define it?
- When do we have it?
- Extending $\mathsf{PolyCirc}_S$ to get it

Why do we want it?

- We want to use morphisms of PolyCirc_S as ML models
- Interpreting morphisms $m \to n$ gives us functions $S^m \to S^n$
- We would like to be able to express *any* function using our syntax
- If we can do this, we have functional completeness
- This is like a discrete analog of "Universal Approximator" theorems for NNs
- We will now be working only with **finite** semirings!

How do we define it?

More formally...

- We want to interpret morphisms $f : m \to n$ of $\mathrm{PolyCirc}_S$ as functions between sets.
- Define FinSet_S as the PROP whose morphisms are functions $S^m \to S^n$
- When any function in FinSet_S has a corresponding morphism $f \in \mathrm{PolyCirc}_S$, then $\mathrm{PolyCirc}_S$ is functionally complete.

Definition

We say a category $\mathscr{C}$ is **functionally complete** with respect to a finite set S when there a full identity-on-objects functor $\mathrm{F} : \mathscr{C} \to \mathrm{FinSet}_S$.

For some S, $\mathsf{PolyCirc}_S$ is *already* functionally complete

Example: $\mathsf{PolyCirc}_{\mathbb{Z}_2}$...

- Addition as XOR
- Multiplication as AND

...is functionally complete

... but not always!

Example: For $\mathbb{B}$ the boolean semiring with:

- Addition as OR
- Multiplication as AND

$\mathsf{PolyCirc}_{\mathbb{B}}$ is *not* functionally complete (you need a NOT gate!)

The missing piece

$$\text{compare}(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

Function Tables

The idea is that because S is finite, we can simply encode the function table of any function $f : S^m \to S$.

e.g., for $f(x_1, x_2) = x_1 \cdot (x_2 + 1)$ in $\mathbb{Z}_2$:

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	0
1	0	1
1	1	0

How do we encode this?

Function Tables II

We can encode function tables using only constants, addition, and multiplication:

$$x \mapsto \sum_{s \in S^m} \text{compare}(s, x) \cdot f(s)$$

- f is 'syntactic' above- we only use it to build the expression
- compare is 1 only when $s = x$ (i.e. exactly once!)

Example: $S = \mathbb{Z}_3,\ m = 1,\ f(x) = x + 2$

$$\begin{aligned} x \mapsto \quad & \text{compare}(0, x) \cdot 2 \\ & + \text{compare}(1, x) \cdot 0 \\ & + \text{compare}(2, x) \cdot 1 \end{aligned}$$

Theorem

Let S be a finite commutative semiring. A category $\mathscr{C}$ is functionally complete with respect to S iff. there is a monoidal functor $\mathsf{F} : \mathscr{C} \to \mathsf{FinSet}_S$ in whose image are the following functions:

- $\langle\rangle \mapsto s$ *for each* $s \in S$ *(constants)*
- $\langle x, y \rangle \mapsto x + y$ *(addition)*
- $\langle x, y \rangle \mapsto x \cdot y$ *(multiplication)*
- compare

Example: $\text{PolyCirc}_{\mathbb{Z}_p}$

$\text{PolyCirc}_{\mathbb{Z}_p}$ is functionally complete for prime p. Fermat's Little Theorem says:

$$a^{p-1} \equiv 1 (\text{mod } p)$$

for $a > 0$. In other words, this is the 'nonzero indicator' function. We can construct the 'zero indicator' function like this:

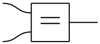
$$\delta(a) := (p-1) \cdot a^{p-1} + 1 = \begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}$$

Now we can construct compare (because S is finite):

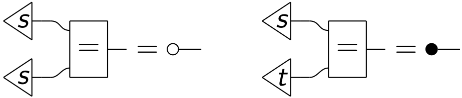
$$\text{compare}(x_1, x_2) = \sum_{s \in S} \delta(x_1 + s) \cdot \delta(x_2 + s)$$

$\mathsf{PolyCirc}_S^{=}$

To get functional completeness, we just add one more widget, which we will interpret as the compare function

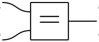


and equations



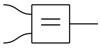
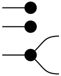
for $s, t \in S$ with $s \neq t$.

The Reverse Derivative of Comparison

To make $\mathrm{PolyCirc}_S^{=}$ an RDC, we need to define R[] in a way that:

- Is consistent with its equations
- Satisfies axioms **ARD.2** - **ARD.4**

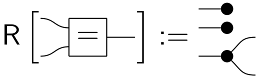
We choose this:

R [] := 

This satisfies the conditions, but is it reasonable?

The Straight-Through Estimator

- DNN architectures sometimes use functions with zero derivatives
- Example: ‘Thresholding’ function $\delta_{x \geq 0}$
- Instead of using the zero derivative, just ‘pass through’ the gradients to not lose information
- This is called the *straight through estimator*



Summary

- We defined PolyCirc in an 'extensible' way
- We showed how RDCs can be defined in a 'presentation friendly' way too
- We used our definition to extend PolyCirc with an additional operation (comparison)
- We ended up with a useful model class for machine learning

Thanks for listening!

Questions?

Richard F. Blute, J. R. B. Cockett, and R. A. G. Seely.
Cartesian differential categories.
Theory and Applications of Categories, 22, 2009.

Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi.
String diagram rewrite theory i: Rewriting with frobenius structure, 2020.

Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon Plotkin, and Dorette Pronk.
Reverse derivative categories, 2019.

Lukasz Kaiser, Aidan N. Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit.
One model to learn them all, 2017.

Paul Wilson and Fabio Zanasi.