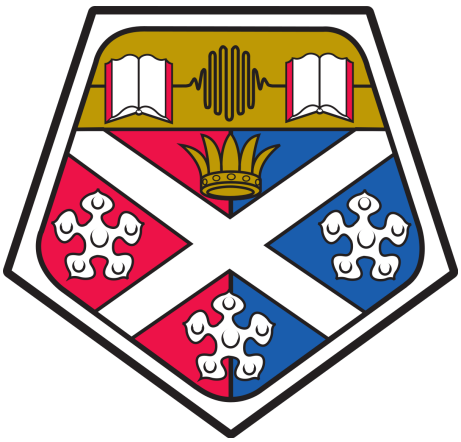


Fundamental Components of Deep Learning

A category-theoretic approach

PhD Thesis

Bruno Gavranović

Mathematically Structured Programming Group

Computer and Information Sciences

University of Strathclyde, Glasgow

February 25, 2024

Abstract

Deep learning, despite its remarkable achievements, is still a young field. Like the early stages of many scientific disciplines, it is marked by the discovery of new phenomena, ad-hoc design decisions, and the lack of a uniform and compositional mathematical foundation. From the intricacies of the implementation of backpropagation, through a growing zoo of neural network architectures, to the new and poorly understood phenomena such as double descent, scaling laws or in-context learning, there are few unifying principles in deep learning.

This thesis develops a novel mathematical foundation for deep learning based on the language of category theory. We develop a new framework that is a) end-to-end, b) unform, and c) not merely descriptive, but prescriptive, meaning it is amenable to direct implementation in programming languages with sufficient features. We also systematise many existing approaches, placing many existing constructions and concepts from the literature under the same umbrella.

In Part I, the theory, we identify and model two main properties of deep learning systems: they are *parametric* and *bidirectional*. We expand on the previously defined construction of actegories and **Para** to study the former, and define *weighted optics* to study the latter. Combining them yields *parametric weighted optics*, a categorical model of artificial neural networks, and more: constructions in Part I have close ties to many other kinds of bidirectional processes such as bayesian updating, value iteraton, and game theory.

Part II justifies the abstractions from Part I, applying them to model backpropagation, architectures, and supervised learning. We provide a lens-theoretic axiomatisation of differentiation, covering not just smooth spaces, but discrete settings of boolean circuits as well. We survey existing, and develop new categorical models of neural network architectures. We formalise the notion of optimisers and lastly, combine all the existing concepts together, providing a uniform and compositional framework for supervised learning.

Contents

Acknowledgements

I'm incredibly grateful to people around me for the last four years. Despite not being trained as a mathematician, I had the privilege of engaging with so many people who think in such crystal clear ways, ones which were not even on my radar of being possible. And these people — the Mathematically Structured Programming group (MSP) where I've done my PhD, and the Applied Category Theory commmunity in general — are also such a welcoming and inclusive community, without whose help and encouragement I wouldn't have been able to be where I am today.

Specifically, I want to thank my advisor Neil Ghani whose guidance, lectures on category theory (and lectures on writing!), as well as the provided space to explore my own ideas were central in allowing me to learn so much. Thanks to Jules Hedges who in many ways was not only my second, informal supervisor, but also a mentor, and more importantly, a friend whom I've had the pleasure to explore abstract nonsense with. Thanks to Jérémy Ledent for patience in our conversations in which he often helped me understand what was it that I actually *meant* to say mathematically. I thank Conor McBride for imparting priceless wisdom on how to *think with types*. The conversations stayed with me in a way that is hard to summarise. It's hard to overstate how positive my exprience was at MSP, and I can't thank enough all the people with whom I've had the pleasure to share a chat and a pint with — Clemens, Bob, Fredrik, Dylan, Riu, Joe, Georgi, André, Alasdair, Guillaume, Malin, Zanzi and Ezra, to name just some. Thanks especially to Joe fun whiteboard chats! There's countless other collaborators, friends and everything in between: Igor Baković, Fabio Zanasi, Geoffrey Cruttwell, Dan Shiebler, David Jaz Myers, Fabrizio Genovese, Brandon Shapiro, Owen Lynch, Toby Smithe, Tali Beynon, Nima Motamed, Mattia Villani. I thoroughly appreciate all of the things I learned from you, and the fun times we've had. Thanks especially to Brendan Fong for originally introducing me to Neil, and getting me started with category theory. Thanks to people who gave me feedback on my thesis: Neil, Jérémy, Paul Lessard, and my thesis examiners

Radu Mardare and Sam Staton.

I thank my family for support, and sacrifices they've made so I could get where I am. *Hvala vam.* And lastly, Ieva. I've grown so much with you over the last four years. Thank you for your love, support, and immeasurable kindness. You've been my partner, my friend, and I'm proud to have you by my side. You hold a special place for me, and I hope you know it.

Chapter 1

Introduction

> The task of the academic is not to scale great intellectual mountains but to flatten them.
>
> Conor McBride

Throughout the last few decades, many behaviours thought to have been unique to humans have systematically been replicated by computer programs.

Computer programs have been exhibiting increasingly better performance on variety of benchmarks aimed at measuring high-level reasoning, perception, and planning. These include generation and classification of images ([RDN+22, RBL+22, DBK+21]), text ([Ope22]), audio ([CRBD18, DXX18]), few-shot learning ([BMR+20, FAL17, HAMS22]), game-playing — notably Jeopardy ([Fer12]), Atari games ([MKS+15]), and Starcraft II ([VBC+19]), but also traditional games. Today, the best chess and Go players in the world are not human ([SHS+18]). The chess engine Stockfish is estimated to hold an ELO rating of 3542 ([Tea23]), while the best human player, Magnus Carlsen, has an ELO of 2882.[1] In Go, the system AlphaGo beat Lee Sedol — the one of the best players in the world — in 2016, overturning centuries of best Go practices, and making players around the world rethink how they approach the game ([SKvOG23]).

Many of these achievements were once thought to be impossible, or even distinguishing factors between humans and machines. In mid 20th century, it was not uncommon to believe that making

[1] ELO is a rating system for calculating the relative skill of players in zero-sum games such as chess.

computers play chess well would be a significant milestone towards understanding intelligence. Likewise, until very recently it was not uncommon to believe that producing and responding to queries in natural language is something that's uniquely correlated with intelligent behaviour.

But these are not widespread beliefs anymore. With the defeat of the reigning world chess champion Gary Kasparov by a computer system called Deep Blue in 1997, it became clear that the kind of analytical thinking needed for chess is only one aspect of what we deem to be intelligence. And with the advent of large language models (LLMs) ([Ope22]) it became clear that natural language understanding is also only one aspect of what we might call intelligence, as LLMs often produce sensible sounding, but logically incorrect natural language phrases.

These and many other developments gave rise to the field of Artificial Intelligence (AI), which has since experienced an explosive rise, having permeated almost every realm of technology. It is receving continued investment from both industry, venture capital and governments, increasing the rate at which these systems are being implemented.

Despite this, the consensus on what constitutes "Artificial" and "Intelligence" has not been well-established. The goalposts for these definitions have been consistently shifted throughout history. They have been shifted so many times that John McCarthy, one of the pioneers of AI, has given this phenomenon a name: the "AI effect" ([HK19]), describing the tendency to redefine the concept of Artificial Intelligence to exclude whatever has been recently accomplished, under the justification that this recent accomplisment never required intelligence to begin with.

The AI effect highlights the challenges with predicting such developments, and understanding their implications. Indeed - where will this continued pace of developments lead us? How far will we move the goalposts 30, or 50 years into the future? Even when it comes to fundamental questions such as "what distinguishes humans from machines", history has shown us that our common sense has repeatedly been wrong about these matters. "Obvious" answers such as *only humans can play chess*, or *only humans can communicate in natural language* seemed plausible before, but they are not anymore (Fig. 1.1). Even the revered Turing test ([TUR50]) is not helpful here. It leaves out the matter of defining artificial intelligence to the discretion of a panel of human judges without any guarantees that this panel is working with a good definition to begin with.

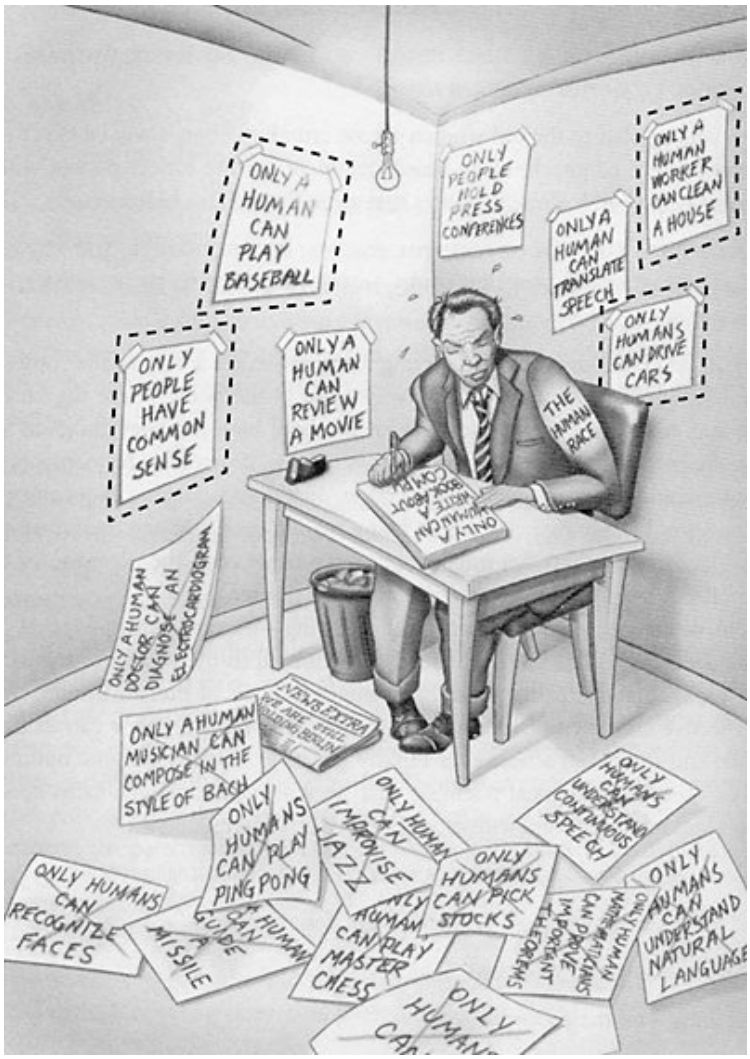


Figure 1.1: The number of behaviours that can still be exhibited *only* by humans has steadily been shrinking. Graphic taken from [Kur99].

In this thesis, we take a step back, and search for a conceptual framework that will stand the test of time. We study the machinery behind these systems — artificial neural networks, and the underlying field of deep learning — from a very specific vantage point. We examine them through the lens of a new, and radically different way of understanding mathematics, and science as a whole: Category Theory. We use this language as a means of flattening the mountain of almost insourmountable deep learning research ([OC17]) created in recent years, and aim to lay the mathematical foundation for deep learning informed by 21st century mathematics.

1.1 From alchemy to chemistry

To understand where we are going, it is good to take a step back.

As a scientific field, where is deep learning currently? It is not hard to see that it has achieved exceptional successes in recent years. But it is still a young field. Compared to millenia-old disciplines like mathematics or physics, deep learning is roughly 50-year-old. It is notoriously ad-hoc, and only beginning to find its footing. However, this initial *ad-hoc* phase is not unique to deep learning — this is how scientific fields start.

For example, people have been classifying animals and plants for thousands of years, but our conceptual understanding of the field of taxonomy changed drastically with the discovery of the concept of evolution. In programming, researchers started tinkering with vacuum tubes, but most of what we call programming today — pointers, recursion, map-reduce or monads — is heavily abstracted from that. People have been stacking rocks on top of each other since the dawn of time, but only in the last hundred years or so we have discovered materials science, finite-element analysis and general math and physics that enabled us to build skyscrapers once considered impossible.

And many parallels can be drawn between deep learning and chemistry. Throughout the centuries, there have been many practical advances in the field of chemistry. These include the development of metallurgy, glass making processes, lab equipment, as well as many systematic studies of chemical substances and their transformations. However, most of these advances were made by early researchers with no knowledge of the periodic table of elements, protons, neutrons, nor electrons: the basic building blocks of what they were studying. Researchers were documenting many new and unexpected phenomena, and most of their theories interpreting the experiments did not survive the test of time. Many researchers of that era are today called *alchemists*. Among them were ones who believed they could transmute base metals into gold, find the elixir of life, or create panacea — a universal remedy able to cure all diseases.

But science progressed, and our understanding deepened. We discovered the atom, and are now aware of the many phenomena of quantum physics. Modern chemistry provided us with theories that allowed us to predict new elements and advance medicine, enabling us to build and scale today's sprawling healthcare systems. Today, we have the ability to cure many diseases alchemists tried and failed to cure. However, we are doing it based on completely different principles.

As argued by many, deep learning today is still in the alchemy stage ([Pre18, Hut18]). Just like alchemists, deep learning researchers are running experiments and making practical advancements.

They are creating recipes for countless neural network architectures and optimisation procedures, while also documenting a number of new and poorly understood phenomena. These include vanishing and exploding gradients ([PMB13]), double descent and grokking ([NKB+21, HY21, LKN+22]), lottery ticket hypothesis ([FC19, FDRC20]), scaling laws ([KMH+20, BDK+21]), and in-context learning ([VONR+23]), to name a few.[2]

However, all of these phenomena, architectures and new practical advancements are made without any consensus on the overarching theory of deep learning. Architectures and advancements are guided by heuristics of what works well in practice, often involve using ad-hoc design decisions and conflicting ways of thinking about the field as a whole. Existing theories are usually highly specialised to a particular subfield of deep learning, while implementations are often brittle. There are examples of the performance machine learning models dropping drastically after internals of frameworks they used changed the default way of rounding numbers ([Pre18]), or with different initial seeds ([Irp18]), a problem arising particularly in reinforcement learning.

Unlike with chemistry, we are not at a stage where we can confidently predict new phenomena, or easily map out a space of neural network architectures.

There is no periodic table of neural network architectures.

Finally, just like alchemists were yearning for a remedy that could cure any disease, it is not uncommon for deep learning researchers to be motivated by the discovery of a general neural network architecture that can solve *any* task.

From chemistry to nuclear physics

Despite our lack of understanding of deep learning systems, we are deploying them at scale. Deep learning systems drive cars, recommend products, amplify thoughts on social media, screen job applicants, and have generally permeated our daily lives. The rate at which they are being implemented is increasing. This is because of continued investment by major tech companies, venture capital firms and governments, with the biggest experiments in deep learning crossing the price point of $100 million ([Kni23]).

[2] Unlike alchemists, we understand *exactly* what happens at the low levels of abstraction. We understand fundamentally everything about bits are moving through logic gates, or compositions of linear layers with activation functions. What we don't have is a good understanding of the modular building blocks of neural networks *at the intermediate scale*. This is because *more is different* ([And72]), and knowledge of the low-level abstractions does not allow us to easily predict emergent behaviour.

While undeniably producing immense value and benefit to our society, deep learning systems also exhibit numerous issues: lack of explainability ([XUD+19]), amplification of biases ([Llo18]), and adversarial attacks ([VOFA24]), to name a few. With the increase of integration of these systems into our society there is a growing need for sensible regulatory paradigms and safety assurances. Unlike the regulation of nuclear weapons, where the dangers are clear and mitigation is generally a matter of execution, in the case of AI both the dangers and their mitigation are not clear.

As deep learning systems are generally *black boxes* it is hard to prevent unwanted behaviour. When it comes to regulation, the prevalent practice is mostly *retroactive*, meaning the focus is on explaining decisions a deep learning model has already made. Instead, if we wanted to express the kind of reasoning we would *proactively* uphold a model to, we'd need a formal language that is both general enough to cover a wide-array of possible use cases, and also precise enough to give rise to enforceable legislation. But there is no consensus about what might such a language look like.

The deep learning community is aware of these issues. Researchers have called for a deep learning "Langlands programme" ([Rie20]) and a deep learning "Erlangen programme" ([BBCV21]), both originally inspired by far-reaching programs aimed at unifying disparate parts of mathematics. Recently, the first PhD thesis on the safety of artificial general intelligence has been published ([Eve19]).

There is agreement in the deep learning community about the fact that we have a pressing need for a principled, rigorous and verifiable theory of deep learning. Nonetheless, there does not seem to be widespread consensus about how exactly to reach that goal. However, science will advance. It is not unreasonable to expect that we will have solved many of these problems in the future. The only question is, how will our understanding have changed? Looking at deep learning in 30, or 50 years, how many theories will have stood the test of time?

1.2 Category Theory

While the mountain of deep learning research has steadily been growing, another field has gradually begun the arduous task of dismantling the "alchemical" theories of deep learning, and assembling a more rigorous, transparent and systematic foundation. The field of *Category Theory* has quietly been flourishing, unbeknownst to most deep learning researchers and the general scientific population.

Category theory (CT) is a language, a way of thinking and of structuring knowledge, markedly

different from others. Originating in abstract mathematics CT has since proliferated, and been used to express ideas from numerous different fields (Box 1.2.1) in a uniform way. This consequently revealed commonalities between these fields which were previously unknown, resulting in a battle-tested system of interfaces which can reliably be applied across scientific fields. It also resulted in a wide array of well-developed tools and theories (geometry, algebra, programming, combinatorics, and topology, to name a few), as well as a wide talent pool of researchers. The number of CT researchers is steadily growing, and it is relatively easy to onboard them to other fields, as they share the same conceptual knowledge base.

Category theory has evolved jointly with the consolidation of a community of researchers interested in *Applied Category Theory* (ACT), a field aiming to implement ideas from category theory in other fields. Unlike conferences in other disciplines, where researchers coalesce to discuss different techniques for thinking about the same problem, ACT conferences can be thought of as assembling researchers from different fields interested in applying the same techniques to different problems.[3]

So, what is category theory, precisely? We delay a more detailed introduction to Section 2.2, and instead here present the following quote from [Lei14], accompanied by a graphic from [Rie20]:

"Category theory takes a bird's eye view of mathematics. From high in the sky, details become invisible, but we can spot patterns that were impossible to detect from ground level."

[3] I have first seen CT described as a "battle-tested system of interfaces" with a deep talent pool of researchers in [Lyn23], while the way of presenting ACT conferences is something I first heard from Jamie Vicary.

Personally, I found it astounding the kinds of things that are possible to express in this language. It changed the way I approach and think about problems, in a way that is hard to express.[4] Hopefully a part of the spark that guides me towards category theory is transmitted throughout this thesis.

Which fields has Category Theory permeated? **Box 1.2.1**

Other than modern mathematics which it thoroughly permeates, category theory has been used to describe:[a]

- Quantum physics ([HVHV19, CK17])
- Probability ([Stu15, HKSY17, Per22])
- Chemistry ([BCC+22, BP17, GS20])
- Cryptography ([BK22, Pav14, BPT23])
- Systems theory ([CGHR22, Mye22a])
- Automata theory ([BLLL23, AM75])
- Functional programming ([Mil19])
- Trading protocols ([GLP21])
- Recursion schemes ([HWG13, KV06])
- Ergodic theory ([MP23, Del19])
- Database theory ([Spi23a, MSW22])
- Thermostatics [BLM23]
- Game theory ([GHWZ18, CGLF22]
- Reinforcement learning ([HS23])
- Information theory ([Lei21, Bra21])
- Electrical circuits ([BF18, BS22])
- Control theory ([HSH+23, BE15])
- Version control ([MDG13])

[a] We give no pretense of thoroughness, and note that many of these fields overlap. We list only one or two references but in most cases there are dozens, sometimes hundreds more.

Lastly, to understand how category theory compares to other mathematical theories, we present the following remark.

Remark 1.1 (Differences between category, set, and graph theory.)**.** One might rightfully point out that set and graph theory, for instance, are all a part of the above mentioned fields. This raises the question: why is category theory singled out? Essentially, category theory tracks more information. While we can study everything with only sets and functions, we effectively start doing category theory as soon as we start tracking properties of these sets and functions in a consistent manner.

[4] I often say that category theory takes time to learn not because it's complex, but because it's so unintuitively simple.

If we're only interested in sets with some structure and functions that preserve that structure — closed under composition — we get a category. When it comes to graph theory, the story is similar. Graphs and graph homomorphisms, when organised into collections, indeed form a category as well — not just a graph. This is symptomatic of many constructions in various aspects of mathematics: they all assemble into categories. The extra data that category theory *coherently* tracks ends up being an indespensible tool for describing numerous scientific and mathematical phenomena in a systematic manner that would not be possible without it.

1.2.1 Category Theory ∩ Machine Learning

In the recent years, a number of authors have began to study machine learning concepts through the lens of category theory.

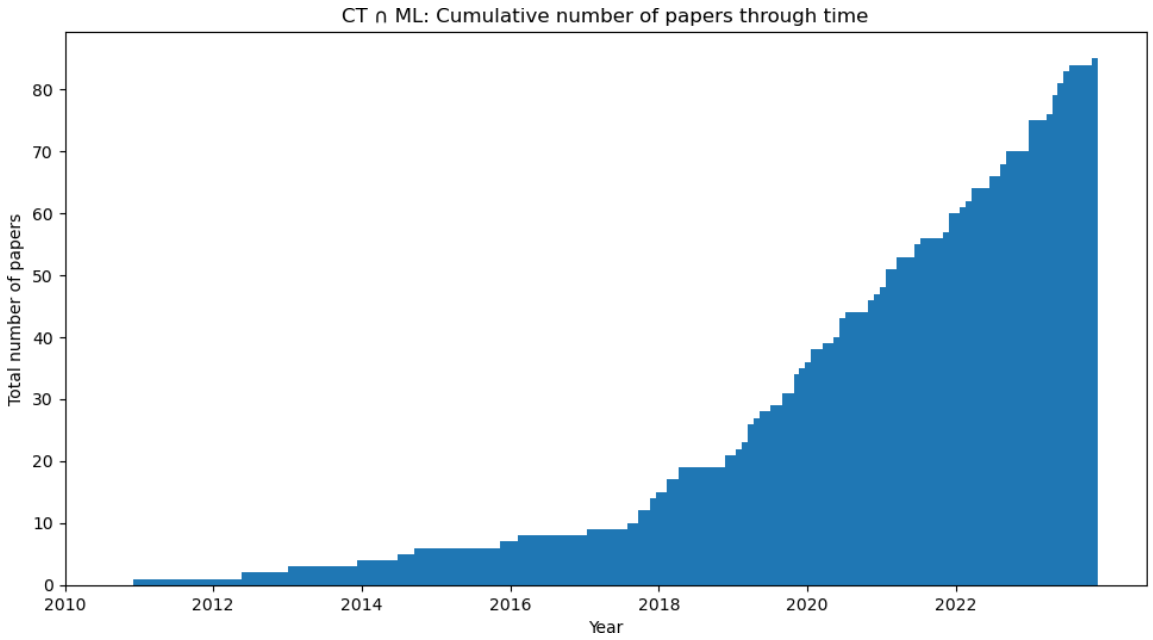


Figure 1.2: Category Theory ∩ Machine Learning: cumulative number of papers through time. Data and figure taken from [Gav20a].

Figure 1.2 shows the total number of such papers so far, including papers in deep learning, bayesian and causal inference, general probability, differentiable programming, topological data analysis, and more. While the total quantity of research is still relatively low, there is a steady rise. The current stage of development of the machine learning in the context of category theory is best

understood through the vantage point emphasized by the following prescient quote from [BF18][5]:

> "While diagrams of networks have been independently introduced in many disciplines, we do not expect formalizing these diagrams to immediately help the practitioners of these disciplines. At first the flow of information will mainly go in the other direction: by translating ideas from these disciplines into the language of modern mathematics, we can provide mathematicians with food for thought and interesting new problems to solve. We hope that in the long run mathematicians can return the favor by bringing new insights to the table."

This quote indirectly suggests partitioning applied category theory fields into two groups. We call these **Type 1** and **Type 2** fields of ACT. In a **Type 1** field, the flow of information still only goes from the application to the theory, and all compositionality that the category theory describes already exists in the application domain. A **Type 2** field genuinely carves out new compositionality in the domain, bringing *new insights* to the table that were not known before.

While merely a rough guideline, it helps give us a sense where Categorical Deep Learning is: currently a borderline Type 2 field. It can describe *a lot* of existing compositionality, but hasn't quite justified itself with many new insights. In this thesis, we hope to provide a stable foundation that can enable these insights to happen.

Goals and Challenges

The goal of this thesis is the provision of a mathematical foundation for artificial neural networks that is

- **End-to-end.** We aim to deal with the entirety of neural network theory and practice — from implementation details of backpropagation, over the algebra of parametric composition of neural networks, to specific architectures, optimisers, and a complete description of supervised learning.
- **Uniform.** We take special care ensuring all the described pieces fit in a consistent way, through a uniform and consistent language.

[5] Unfortunately, this quote can only be found in version 1 of the paper on ArXiv.

- **Prescriptive, not merely descriptive.** The aim is to provide a prescriptive framework for *implementing* all of the above, one which can eventually become a part of future deep learning frameworks in programming languages with sufficiently expressive features.

To achieve this, we utilise the aforementioned language of category theory to its full extent. We bring many constructs and insights from this abstract field of mathematics into deep learning, developing an entire foundation for deep learning in this language.

The thesis structure is as follows.

Part I: Theory Part I covers two main components of deep learning: parameterisation and bidirectionality. Its main contribution is a formal mathematical framework for studying the essential structure behind these concepts. It is written with more than just deep learning in mind: the level of abstraction covers many other kinds of bidirectional processes found in machine learning such as bayesian updating and value iteration, and also having close ties to game theory. As such, despite my best attempts, the level of abstraction here is intended to a seasoned category theorist.

In Chapter 3, I build upon the **Para** construction in its full bicategorical nature, unpacking all the underlying structure in detail. I define the conditions under which it can be strictified, equipped with a monoidal structure, and trivialisation conditions, as well as relate it to existing definitions in the literature. In Chapter 4, I define the category of *weighted optics*, a new foundation for bidirectional systems that allows modelling backpropagation not just denotationally correct, but also one which operationally captures distinctions relevant in practical implementations. I define cartesian actegories and show how they are useful in studying particular kinds of weighted optics: lenses. Chapter 5 combines the previous two chapters, defining a bicategory in which morphisms are *parametric weighted optics*, the central construction of this thesis. This is a versatile construction that models neural networks, loss functions, and whose 2-cells also model optimisers. It also a category in which scalars (morphisms from the monoidal unit to the monoidal unit) model a parameter update step of an entire supervised learning system.

Part II: Applications Part II justifies the abstractions from Part I, instantiating and using them to describe settings relevant for deep learning: backpropagation, neural network architectures, and supervised learning. In Chapter 6 I provide a lens-theoretic axiomatisation of differentiation. It gives an abstract view of differentiation covering more than just the standard smooth spaces, but also discrete settings of boolean circuits, while using the same general framework for bidirection-

ality. Specifically, this includes the definition of a particular kind of an additively closed category, definition of $\mathbf{Lens}_A$ as an endofunctor and the definition of a generalised cartesian reverse derivative category as its coalgebra. I also identify differences between checkpointed and non-checkpointed reverse-mode automatic differentiation solely in the language of category theory. Chapter 7 describes how the framework of parametric weighted optics can be used to provide a solid foundation for defining a general theory of neural network architectures. I develop the a categorical formalism for weight tying, generative adversarial networks and graph convolutional neural networks, to name a few, as well as survey existing categorical work on recurrent and recursive neural networks. Lastly, Chapter 8 extends the previous frameworks of supervised learning in [CGG+22], and puts all the pieces together. I describe how learning rates and optimisers can be modelled and composed within this framework, eventually in conjunction with a model and a loss function forming a morphism that compositionally captures the notion of supervised learning of parameters. This is instantiated in a variety of settings, even those that are not usually considered as supervised learning, such as generative adversarial networks, shedding light on their semantics as supervised learners. I also describe how this framework captures supervised learning of *inputs*, a method usually called *deep dreaming* in the literature.

Future work. While great care has been taken to separate out as many conceptual moving parts as possible, some things are still stuck together. The last and the shortest chapter of this thesis lists a number of open research questions, branching out in multiple directions. From establishing bridges between differential geometry, tangent categories and practical concerns of automatic differentiation (such as gradient checkpointing), over the design of new architectures and connection of Geometric Deep Learning ([BBCV21]) to structural recursion, to ultimatively, *taking dependent types seriously*.

1.2.2 Publications and preprints

The "**Contributions**" environment at the end of the introduction of each chapter in Part I and II summarises the novel research done in that chapter, alongside with the relevant publications and preprints I (co)authored. The complete list of these preprints and publications is below.

[CGG+22] **Categorical Foundations of Gradient-Based Learning**

[CG23] **Towards Foundations of Categorical Cybernetics**

[SGW21] **Category theory in machine learning**

[CG23] **Actegories for the working amthematician**

[BCG+21] **Fibre optics**

[Gav22b] **Space-time tradeoffs of lenses and optics via higher category theory**

[GV22] **Graph Convolutional Neural Networks as Parametric CoKleisli morphisms**

[GHWZ18] **Compositional game theory, compositionally**

Part I

Theoretical Foundations

Chapter 2

Setting the Stage

> To deal with hyper-planes in a 14-dimensional space, visualize a 3-D space and say 'fourteen' to yourself very loudly. Everyone does it.
>
> Geoffrey Hinton

What is Deep Learning, more closely? What about Category Theory? In this chapter we provide the necessary background material.

2.1 Deep Learning: a bird's eye view

Deep Learning is a subfield of the field of machine learning that studies the theory and practice of *artificial neural networks*. Loosely inspired by neural networks in the brains of humans and animals, the goal of deep learning is the construction of networks of interconnected artificial neurons that have the ability to learn from data by seeing lots of examples.

Consider one of the most common deep learning scenarios: supervised learning with a neural network. This technique trains the model towards a certain task, i.e. the recognition of visual patterns in an image data set (Fig. 2.1). There are several diferent ways of implementing this scenario. Typically, at their core, there is a gradient update algorithm (often called the "optimiser"),

depending on a given loss function, which updates in steps the parameters of the network, based on some learning rate controlling the scaling of the update. All of these components can vary independently in a supervised learning algorithm and a number of choices is available for loss maps (quadratic error, Softmax cross entropy, dot product, etc.) and optimisers (Adagrad [DHS11], Momentum [Pol64], and Adam [KB15], etc.). Furthemore, the setting in which we perform the learning can vary as well: in addition to euclidean spaces, we can differentiate functions between hyperbolic spaces ([PVM+22]), between complex numbers ([BQL21]), and even between discrete settings of boolean circuits [HCS+16, QGL+20, WZ21].

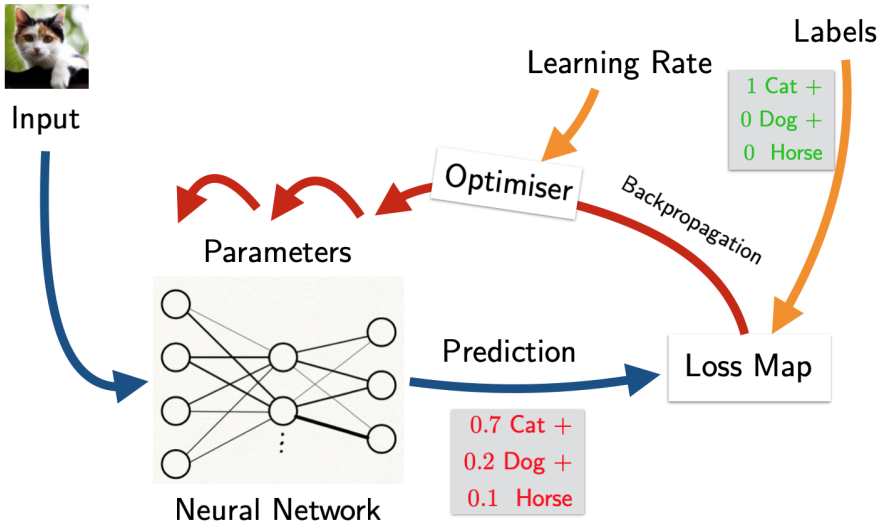


Figure 2.1: An informal illustration of gradient-based learning. This neural network is trained to distinguish different kinds of animals in the input image. Given an input X, the network predicts an output Y, which is compared by a *loss function* with what would be the correct answer (*label*). The loss function returns a real value expressing the error of the prediction; this information, together with the *learning rate* (a weight controlling how much the model should be changed in response to error) is used by an *optimiser*, which computes by gradient-descent the update of the parameters of the network, with the aim of improving its accuracy. The neural network, the loss map, the optimiser and the learning rate are all components of a supervised learning system, and can vary independently of one another.

Thinking of neural networks as processes, we can delineate their two main properties.

1. They are **parametric.** Each network comes equipped with a choice of "weights" (or parameters) on which the output depends on. In the simplest case, we are given a function $f : X \times P \to Y$ and are tasked with finding a parameter $p : P$ such that $f(-, p) : X \to Y$ is

the best function according to some criteria. As we compose neural networks, the manner by which the weight spaces are composed too is non-trivial, and the algebra of such composition is often overlooked. This is the topic of Chapter 3.

2. They are **bidirectional.** These processes have a forward and a backward direction. In the forward direction the computation turns inputs via a sequence of layers into predicted outputs, and then a loss value. In the reverse direction it backpropagates changes through the layers, and then turns them into parameter updates. This too has a non-trivial algebra of composition, with a) different modes of composition giving rise to different performance profiles of these bidirectional processes, and b) details and properties of both the forward and backward part being difficult to formally specify in a uniform and consistent manner. This is the topic of Chapter 4.

This gives us the high-level story of deep learning. While deep learning is our main motivation, we keep in mind adjacent machine learning fields such as bayesian learning and reinforcement learning. Throughout Part I of the thesis a lot of the abstract machinery for bidirectionality will inadvertently end up capturing structure that is present in Bayes' law, or value iteration. As such, Part I is developed at a level of generality not specifically needed for neural networks. But we will see that this level of generality gives us a fresh and useful perspective on neural networks themselves.

The main concern of deep learning is however, in many ways, much more specific. Most of deep learning research studies the manner by which we structure the backward pass from the forward one (i.e. backpropagation), the manner by which we structure the parametric morphisms (i.e. the *architecture* of the forward passes), and the manner by which all of these components fit into a coherent story of *supervised learning*. These three components define the contents of the three chapters in Part II of the thesis.

1. **Differentiation.** The forward and the backward pass of a bidirectional process are related: the backward pass is automatically constructed from the forward one by the process of differentiation. We are interested in providing formal models of each that capture all the relevant examples, but can also be integrated in a coherent whole. This means that we should systematically be able to change the kind of space we're differentiating, or the kind of parameters we're considering, in and still be able to perform supervised learning. Likewise, here we too

want to be operationally aware, and distinguish between different performacne profiles of different modes of composition.

2. **Architectures.** In recent years the number of architectures of neural networks has proliferated. There exist countless architectures, combinations theoref, and entire theories describing how each class of architectures operates, and their expressive power. As such, there is a large need for a principled and formal specification language for these architectures, one which would obviate the need for informal diagrams that are used by most of today's deep learning literature.

3. **Supervised learning.** The most pervasive mode of deep learning is that by *supervision*, where a dataset of input-output examples is used to specify an optimisation procedure which produces a function mapping each input to the corresponding output as closely as possible, while ensuring it generalises well. This supervised learning system involves the interaction of a neural network, an optimiser, a loss function, and their corresponding backward passes, and forms a good litmus test for any end-to-end framework: how well can it explicitly describe the interaction of all of these components?

Having given a high-level overview of deep learning, we now turn to category theory.

2.2 Categories and Systems

You shall know a word by the company it keeps.

This prescient quote from the linguist John Rupert Firth could be seen as a backbone of category theory.[1] It tells us that the string of characters a word is composed out of — its syntactic content — is in many ways meaningless. Instead, what gives a word much of its meaning is the way we use it in relation to all the other words in the language.

Describing things *exclusively* by how they can be related and interacted with from the outside, without looking at their insides, is one way to frame what category theory is about. From its humble beginnings in 1940s, the field of category theory has experienced immense growth, now becoming what is called "the mathematics of mathematics". It grew into an entire language in

[1] Similar phrases have been uttered by various people. "The meaning of words lies in their use." is one by Wittgenstein.

which people have described all sorts of scientific phenomena in, coming with its own markedly distinct way of reasoning and thinking.

One of the foundational principles that sets category theory apart is its rigorous approach to *encapsulation*. Just as in software we aim for a modular design of components, so in category theory we aim for a modular design of *concepts*. Ideas and concepts are abstracted away and separated from each other in a principled way, whose interaction is mediated by precise and mutually compatible interfaces.

Take the example of *enriched* category theory. In it we can turn a dial that chooses a category and, poetically, selects different branches of mathematics. Setting it to $\{\text{false}, \text{true}\}$ selects order theory. Turning it to $[0, \infty]$ selects metric geometry. Turning it to **Set** selects category theory itself. Turning it to **Cat** selects 2-category theory. And so on.[2] In the study of bidirectional systems — there is a dial that selects the kind of bidirectional systems we study. One position selects deterministic updates, another one probabilistic, another on updates with computational effects, and so on. Enriched category — and category theory as a whole — is extremely general, and most importantly — reliable.

If something requires a category with some structure X to work, we are assured that *every* category with structure X works; it will never be the case that the construction breaks because of an unforeseen consequence. This is because we are working through a well-specified interface, shielded from the specifics of a particular domain. And we have to learn a particular interface only once, as opposed to repeatedly relearning ad-hoc interfaces for each new domain. The mutual compatibility of these interfaces in turn allows us to scale up our systems more easily, and manage their difficulty of dealing with them as we add moving parts (Fig. 2.2).

On the other hand, using category theory requires a large buy-in. If you're coming from computer science, physics, or other adjacent fields it takes much more time than you're probably accustomed to pick up and apply these concepts. The learning curve is steep, and from my humble experience category theory was just *very surprising*; my meta-model of how my thinking would change by using it was completely off. I did not expect *this* level of change. Category theory cannot easily be picked up in a month, and the sheer number of concepts and ideas in it takes a lifetime to understand.

For the reasons above I will not provide a comprehensive introduction to category theory in this thesis. If you already know category theory, you will not need it, and if you don't, you will not

[2] Paraphrased from [Lei11].

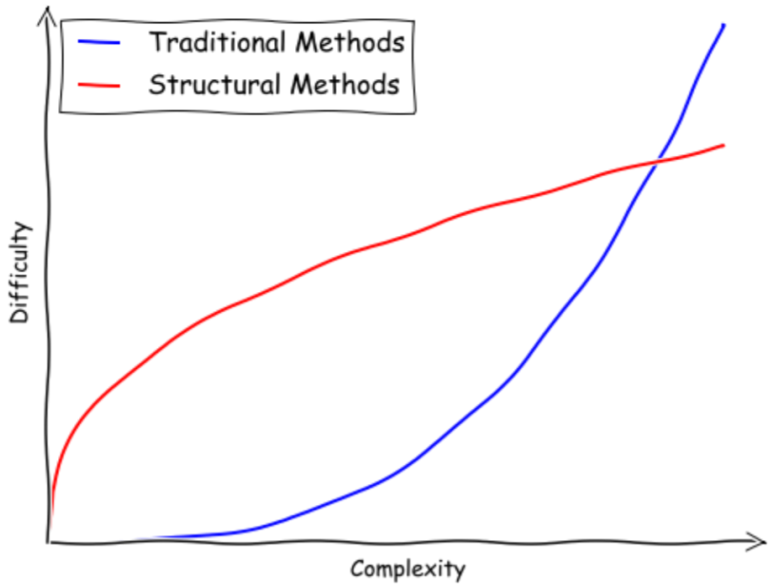


Figure 2.2: In traditional methods it is easy to start building a system. But as the system grows, it becomes difficult to understand how all the moving pieces inside interact. This slows down development, and introduces bugs and side effects. In structural methods, the situation is the opposite. It takes longer to start, but as special care is taken in accounting for all the moving pieces, it becomes easier to manage their complexity, and scale these systems up. (Figure taken from [Bre18])

learn it from here. I will take *some* time to give a sense of some of the main concepts appearing throughout the thesis with no pretense of thoroughness. The thesis does quickly ramps up in complexity, and someone new to category theory might struggle with the terminology and the way of thinking in this thesis. I refer the reader to the list of pedagogical resources for learning category theory in [Gav23a].

Here are some of the main categories used throughout this thesis.

Definition 2.1 (**Set**)**.** One of the central categories in category theory is **Set**, the category whose objects are sets and morphisms are functions.

Definition 2.2 (**Smooth**, [CCG+20, Example 2.3])**.** We write **Smooth** for the category whose objects are real-valued vector spaces of the form $\mathbb{R}^n$ (i.e. objects are effectively a choice of a natural number $n : \mathbb{N}$) and morphisms are smooth functions $\mathbb{R}^n \to \mathbb{R}^m$.

Definition 2.3 (**Poly**$_R$, [CCG+20, Example 2.2])**.** Let R be a commutative rig.[3] We use **Poly**$_R$

[3] Also known as a commutative semiring.

to denote the category of polynomials with coefficients in R. Its objects are natural numbers $n : \mathbb{N}$, and a morphism $p : n \to m$ is an m-tuple of polynomials $\langle p_1(x), p_2(x), \ldots, p_m(x) \rangle$ where $p_i(x)$ is an element of $R[x_1, \ldots, x_n]$, the polynomial ring over R in n variables.[4]

The above definition is dauntingly formal, but is best understood with an example: a morphism $2 \to 1$ in $\mathbf{Poly}_{\mathbb{R}}$ is the polynomial $p(x_1, x_2) = 5x_1^2 + 3x_1x_2 + 2x_2^2$. This is a morphism with codomain 1; if it was instead n, we'd have n polynomials at our disposal. Examples of commutative rigs relevant for this thesis are $\mathbb{R}$ and $\mathbb{Z}_2$.

The categories $\mathbf{Smooth}$ and $\mathbf{Poly}_{\mathbb{R}}$ will be the main examples of categories in which we model supervised learning (Chapter 8). The category $\mathbf{Poly}_{\mathbb{Z}_2}$ will be used to model supervised learning on boolean circuits (Example 8.20).

Definition 2.4 ($\mathbf{FVect}_F$)**.** Let F be a field. The category of finite-dimensional vector spaces $\mathbf{FVect}_F$ has as objects vector spaces over F and as morphisms linear maps.

In this thesis we will focus most on the base field $\mathbb{R}$, i.e. $\mathbf{FVect}_{\mathbb{R}}$.

Definition 2.5 ($\mathbf{Mark}$ (compare [Fri20, Ex. 2.6] and [HS23, Ex. 3]))**.** Let $\mathbf{Mark}$ be a category whose objects are finite sets, and where a morphism $X \to Y$ is a **Markov kernel**, i.e. a $Y \times X$ matrix with non-negative real entries, whose columns sum to 1. Composition is given by matrix multiplication, and identities are given by the identity matrices.

Morphisms in $\mathbf{Mark}$ are best thought of as probabilistic functions: to each element of the domain, they do not assign *one* element of the codomain, but instead a probability distribution over all elements. Markov kernels will be used to model bidirectional process whose forward passes are probabilistic (Examples 4.22 and 4.53)

Definition 2.6 (Natural numbers)**.** The set of natural numbers $\mathbb{N}$ can be thought of as a category where a morphism $n \to m$ exists iff $n \geq m$.[5]

All the categories above have infinitely many objects and morphisms. Not all categories do — some some have just a few.

Definition 2.7 (Edge cases)**.** There is a category with no objects or morphisms, this is often called **the empty category**, and denoted by $\mathbf{0}$. There is a category with only one object, and only the

[4] Some readers of this thesis might be familiar with the category of polynomial functors (briefly mentioned in the literature review in Section 4.5) which is often denoted by **Poly** too. This is a different construction!

[5] This is an example of a *thin* category, i.e. one where there is at most one morphism between any two objects.

identity morphism on it, often called **the terminal category**, and denoted by **1**. Every set A can canonically be turned into a **discrete category**, one whose objects are elements of A, and morphisms are only identities.

Definition 2.8 (Product of categories)**.** Let $\mathcal{C}$ and $\mathcal{D}$ be two categories. Then we can form the **product category** $\mathcal{C} \times \mathcal{D}$ whose objects are pairs of objects (C, D) where $C : \mathcal{C}$ and $D : \mathcal{D}$, and a morphism $(C, D) \to (C', D')$ is a pair (f, g) where $f : C \to C'$ in $\mathcal{C}$ and $g : D \to D'$ in $\mathcal{D}$.

There are numerous more, and countless combinations thereof. We mention one last example — the category of categories.

Definition 2.9 (**Cat**)**.** In **Cat** the objects are categories themselves, and functors are morphisms between categories.[6]

These are only some examples of concepts appearing in category theory. We do not mention monads, algebras, limits, fibrations, adjunctions nor bicategories here, instead referring the reader to the aforementioned list of pedagogical resources for learning those ([Gav23a]).

In the appendix we mention some propositions related to coends (Prop. E.9), profunctors Lemma E.14, and the (co)Yoneda lemma (Props. E.11 and E.12) which will be used in this thesis. We do not introduce these concepts here, instead referring the reader to the invaluable resource for these concepts, the book "Coend calculus" ([Lor21]) whose wisdom I've consulted time and time again.[7]

2.2.1 Monoidal categories

Monoidal categories are a powerful tool in the applied category theory toolbox. Analogous to categories which capture the algebra of sequential composition of processes, monoidal categories additionally capture the algebra of their parallel composition. They have been used in quantum theory ([CK17]), digital and electric circuit theory ([GJL17, BS22]), game theory ([GHWZ18]), bayesian learning ([BHSCS23]), control theory ([BE15, BSZ17]), linear algebra ([PRS22]), and many more fields.

Their widespread use arose because they come equipped with a systematic 2-dimensional language of string diagrams ([Str12, Sel11]). Unlike in many other fields where pictures are merely

[6] We're ignoring size issues in this thesis — the usual techniques dealing with them apply here.

[7] This reference does assume a fair bit of category theory. For a more pedagogical introduction to profunctors see [FS19, Def. 4.8].

informal supplements to proofs, here instead pictures *are* formal proofs. This is made possible by strict rules about what counts as a picture, and how they can be manipulated, giving us a fully formal graphical method of reasoning.

A monoidal structure[8] on a category $\mathcal{C}$ consists of a functor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ (which we can think of as multiplication), and an object $I : \mathcal{C}$ (which we can think of as the neutral element of the said multiplication), as well as other structure morphisms: the associator α, and the left and the right unitor λ and ρ, together with properties they have to satisfy fully described in appendix (Definition A.1. Functoriality of $\otimes$ gives us one of the above mentioned strict rules: the *interchange law*.

Proposition 2.10 (Interchange law)**.** *Let $(\mathcal{C}, \otimes, I)$ be a monoidal category. Let*

$$(f, h) : (A, D) \to (B, E) \quad \textit{and} \quad (g, i) : (B, E) \to (C, F)$$

be morphisms in $\mathcal{C} \times \mathcal{C}$. The interchange law tells us that the following equation holds

$$(f \otimes h) \fatsemi (g \otimes i) = (f \fatsemi g) \otimes (h \fatsemi i) \tag{2.11}$$

describing that we get the same result if we first compose the morphisms in parallel, and then in sequence, or in sequence and then in parallel.

When dealing with string diagrams, we do not have to explicitly think about equations such as Eq. (2.11). These symbolic equations are built-in to the geometry of the plane, alleviating us from dealing with the bureaucracy of equational reasoning. This can best be seen in the example below.

$$(X \otimes (Y \otimes I)) \otimes Z \xrightarrow{(X \otimes \rho) \otimes Z} (X \otimes Y) \otimes Z \xrightarrow{f \otimes Z} (X' \otimes Y') \otimes Z \xrightarrow{\alpha_{X',Y',Z}} X' \otimes (Y' \otimes Z) \xrightarrow{X' \otimes g} X' \otimes T \xrightarrow{h} W \tag{2.12}$$

This is a composite morphism which has many components, many of which are mere bookkeeping. These are mostly rebracketing of terms, and introduction/elimination of the monoidal unit I. The string diagram representation of the composite morphism above is depicted in Fig. 2.3, where this bookkeeping is completely invisible: the unit I, associator α, and the unitor ρ are absorbed into the geometry of the plane. This graphical representation aids intuition, and helps us see the high-level structure.

[8] For the full definition, see Appendix A. For a pedagogical introduction, see [FS19, Sec. 4.4.3], or [WMLC23].

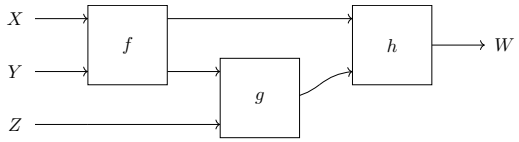


Figure 2.3: String diagram representation of the morphism Eq. (2.12). Objects of $\mathcal{C}$ are denoted as wires, and morphisms as boxes. Notably, unit I, the associator α and the unitor ρ are completely invisible in the graphical representation.

All of the categories mentioned above (**Set**, **Smooth**, **Poly**$_R$, **FVect**, **Mark**, **Cat**, ...) are monoidal categories, often in more than one way.

Example 2.13. The category **Set** is monoidal with its cartesian product $\times$ and the singleton set 1, but also with the disjoint union $\sqcup$ and the empty set $\emptyset$. This means that a morphism $X \otimes Y \to Z$ in $(\mathbf{Set}, \times, 1)$ describes a process which produces an element of Z by consuming X *and* Y, while a morphism of the same type in $(\mathbf{Set}, \sqcup, \emptyset)$ describes a process which produces an element of Z by consuming X *or* Y.

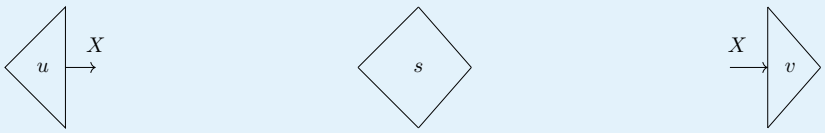
Example 2.14. The categories **Smooth** and **Poly**$_R$ are monoidal with their cartesian product inherited from **Set**, while **Mark** is monoidal with the cartesian product of sets on objects, and tensor product of matrices on morphisms (see [Fri20, Eq. 2.9]). The monoidal products of **FVect** are studied in detail in Section 2.2.3.

When reasoning about morphisms in monoidal categories, we often talk about *open* and *closed* systems. The morphism above is an example of an open system: roughly, it is one which exposes external ports other morphisms can be plugged in to. A closed system does not. Many systems can be partially closed in different ways, and we unpack the details of this in Box 2.2.1. Such boxes will be recurrent characters in this thesis, and we will see how open and partially closed systems look like in as we vary the monoidal category.

States, costates, and scalars in a monoidal category **Box 2.2.1**

The three kinds of (partially) closed systems in any monoidal category are called states, scalars, and costates. They are systems closed from respectively, the right, both sides, or left side. Graphically we represent them as below, drawing the partially closed side with a triangle, emphasising there are no exposed ports from that side. Systems that are closed from both the left and the right are often called "scalars", following the intuition from monoidal categories like $(\mathbf{FVect}_{\mathbb{R}}, \otimes, \mathbb{R})$ where maps of type $\mathbb{R} \multimap \mathbb{R}$ are in 1-1 correspondence with elements of $\mathbb{R}$.

States Scalars Costates



$\mathcal{C}(I, X)$ $\mathcal{C}(I, I)$ $\mathcal{C}(X, I)$

Monoidal categories can come equipped with additional structure called *braiding*. A braided monoidal category is one where wires can be crossed (Fig. 2.4).

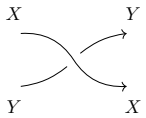


Figure 2.4: String diagram representation of a morphism $\beta_{X,Y} : X \otimes Y \to Y \otimes X$ in a braided monoidal category. Note how one wire is explicitly drawn on top of another. This is to emphasize that β may not be equal to β^{-1}.

Formally this means that we additionally have a natural family of morphisms $\beta_{X,Y} : X \otimes Y \to Y \otimes X$ i.e. a natural isomorphism $\mathcal{M} \times \mathcal{M} \overset{\otimes}{\underset{\otimes^{\mathsf{rev}}}{\Downarrow \beta}} \mathcal{M}$ subject to axioms defined in the Definition A.8) in the appendix, where $\otimes^{\mathsf{rev}}$ is the reversed monoidal product on $\mathcal{M}$ (Definition A.7). Braiding often satisfies a particular property: one telling us braiding twice is the same as not doing it at all (Fig. 2.5). In this case, we call this category a *symmetric* monoidal category. For a full formal definition Definition A.13 in the appendix.

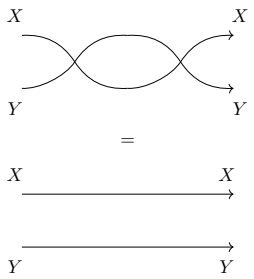


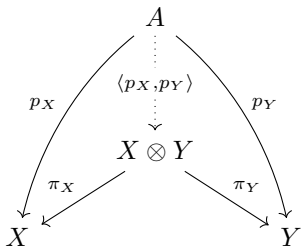
Figure 2.5: In a symmetric monoidal category braiding twice is *equal* to the identity morphism. This means that $\beta = \beta^{-1}$, and in this case we do not distinguish between under- and over-crossings.

Lastly, we mention that functors between monoidal categories can be monoidal in a few ways (lax, strong and strict) (Definition A.6), and if the monoidal categories are braided (resp. symmetric), then the monoidal functor structure can additionally be braided (resp. symmetric) (Definition A.11).

2.2.2 Cartesian monoidal categories

Cartesian monoidal categories are monoidal categories in which we think of processes as being *deterministic*. They are often considered to live in the "classical" world (as opposed to quantum, or a probabilistic), as they allow unlimited copying and deleting of information. Cartesian monoidal categories are often referred to as "cartesian categories" where monoidality is implied.

Formally, this means that the monoidal product has a particular universal property: it is *a categorical product*. This means is that for any two objects X, Y in $\mathcal{C}$ there exist two maps out of $X \otimes Y$: $\pi_X : X \otimes Y \to X$ and $\pi_Y : X \otimes Y \to Y$ (called *projections*), such that the triple $(X \otimes Y, \pi_X, \pi_Y)$ is the "best cartesian product". Being "best" means that any other candidate we might want to call a "cartesian product" (other such triples of objects $A : \mathcal{C}$ that come equipped with candidate projections $p_X : A \to X$ and $p_Y : A \to Y$) can be factored through $(X \otimes Y, \pi_X, \pi_Y)$. Formally, this means there exists a map $\langle p_X, p_Y \rangle : A \to X \otimes Y$ making the diagram (Eq. (2.15)) commute. Moreover, this map has to be *unique*, meaning there's no other maps satisfying this condition.



(2.15)

In a cartesian category we often use 1 to denote the monoidal unit (inspired by the monoidal unit of **Set** which is a one-element set), and $\times$ to denote their monoidal product (inspired by the cartesian product of sets).

There are two equivalent approaches of showing a category is cartesian: a) by exhibiting the following (natural) isomorphism

$$\mathcal{C}(A, X \times Y) \cong \mathcal{C}(A, X) \times \mathcal{C}(A, Y) \tag{2.16}$$

or b) by equipping every object $A : \mathcal{C}$ with a unique comonoid structure $(A, \delta_A, \epsilon_A)$ and showing that all morphisms preserve comonoids ([Fox76]).

The latter approach provides us with a formal way of augmenting the graphical language of the underlying monoidal structure of this cartesian category. That is, the comonoid maps δ_X and ϵ_X have special interpretations as the *copy* and *delete* maps, drawn in a suggestive manner. The preservation of comonoid structure then unpacks to the following equations:

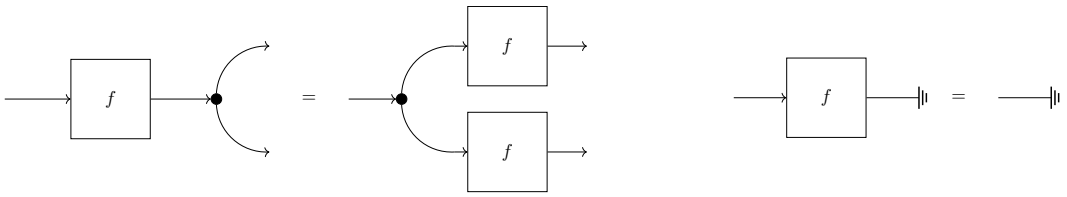
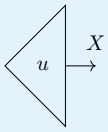


Figure 2.6: A morphism f preserves comonoids if the above graphical equations are satisfied.

describing (on the left) that applying the map f and copying the result is the same as copying the result and applying f individually to each copy. This is not the case in probabilistic settings: rolling a dice and copying the output is not the same as rolling two die. Likewise, the right equation describes that applying a map to an input, and then discarding the output is the same as just discarding the input. In other words, there should be only one way to delete something.

States, costates, and scalars in a cartesian monoidal category **Box 2.2.2**

States, costates and scalars for cartesian monoidal categories follow those for monoidal categories. Additionally, maps into 1 now trivialise.

States	Scalars	Costates
		
$\mathcal{C}(1, X)$	$\mathcal{C}(1, 1)$	$\mathcal{C}(X, 1)$
	$\cong 1$	$\cong 1$

States do not reduce in general. Though, if $\mathcal{C}$ is **Set**, then $\mathbf{Set}(1, X) \cong X$. Scalars are drawn as an empty diagram, because there is only one possible map of type $1 \to 1$: the identity map. Costates also trivialise, because all morphisms have to be comonoid homomorphisms. And the counit preservation law of such a homomorphism implies that here can only be one morphism of type $X \to 1$: usually interpeted as the "delete" map described above.

Most of the above mentioned monoidal categories are cartesian. The notable exception is $(\mathbf{Mark}, \otimes, 1)$, which is not a cartesian category as its morphisms are probabilistic maps.

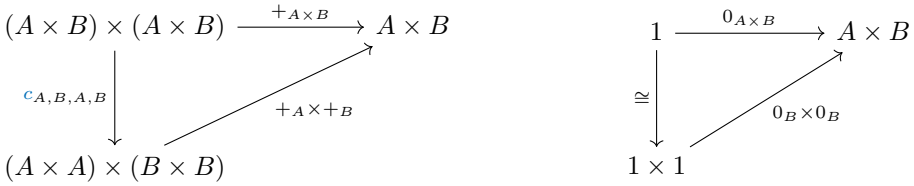
Given any cartesian category $\mathcal{C}$ and any object X therein we will always be able to form a particular category called $\mathbf{coKl}(- \times X)$ (see Section 3.1.2) that will allow us to describe properties of morphism that hold *in one variable* (see Definition 2.22).

2.2.3 Monoids on every object = "left-additive"

In addition to a category having the cartesian structure, we will often find that it comes equipped with a particular "additive" structure on each object. For instance, in **Smooth** given any object $\mathbb{R}^n$ we can always define the operation $+ : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ pointwise via the addition of $\mathbb{R}$, and its netural element as a map $0 : \mathbb{R}^0 \to \mathbb{R}^n$ picking out zeroes. Categorically this is captured by every object $A : \mathcal{C}$ being equipped with a monoid structure in a manner that interacts coherently with the existing cartesian structure. Such categories are often called *cartesian left-additive* categories.

Definition 2.17 ([BCS09, Proposition 1.2.2.])**.** A cartesian category $\mathcal{C}$ is said to be cartesian **left-**

additive (CLA) if every object $A : \mathcal{C}$ is equipped with a commutative monoid $(+_A, 0_A)$ compatible[9] with the cartesian structure, i.e. such that for all objects $A, B : \mathcal{C}$ the following diagrams commute:



where here $c_{A,B,A,B}$ is the interchanger (Definition A.10).

The categories **Smooth** and $\mathbf{Poly}_R$ are examples of cartesian left-additive categories. Specifically for $\mathbb{R}$, the category $\mathbf{Poly}_{\mathbb{R}}$ is a cartesian left-additive *subcategory* of **Smooth**. The categories **Set**, **Cat** and $\mathbb{N}$ are not examples of cartesian left-additive categories.

Remark 2.18 (Why "left-additive"?)**.** A cartesian left-additive category is called so because for every $A, B : \mathcal{C}$ we can always define a monoid struture on the hom-set $\mathcal{C}(A, B)$.[10] That is, we can define a function $+ : \mathcal{C}(A, B) \times \mathcal{C}(A, B) \to \mathcal{C}(A, B)$ mapping (f, g) to $f + g$ where

$$f + g \coloneqq \boxed{A \xrightarrow{\Delta_A} A \times A \xrightarrow{f\times g} B \times B \xrightarrow{+_B} B}$$

and a function $0 : 1 \to \mathcal{C}(A, B)$ mapping $\bullet : 1$ to $\boxed{A \xrightarrow{!_A} 1 \xrightarrow{0_B} B}$. The adjective *left*-additive is there because this monoid is not "fully" natural. Only composition from the left (pre-composition) fully preserves the additive structure (see [CCG+20, Def. 1]). In what follows we will see that additionally preserving the hom-set monoid by composition from the right yields an *additive category*.

Categories that have monoids on every object come equipped with a graphical way of interpreting these monoids, analogously to the graphical language of categories with commutative *comonoids* on every object (such as cartesian categories defined above). That is, the monoid maps $+_X$ and 0_X have special interpretations as the *sum* and *zero* maps, drawn in a suggestive manner with string diagrams.

An arbitrary morphism in a cartesian left-additive category $\mathcal{C}$ is not required to preserve its monoid structure. If it does, we call it an *additive* morphism.

[9] In particular, this compatibility means that the monoid maps $+_A : A \times A \to A$ and $0_A : 1 \to A$ are comonoid homomorphisms, giving a bimonoid structure to each object.

[10] Sometimes this monoid structure is also called *convolution* ([GLP21, Sec. 4.1]).

Definition 2.19 (Additive morphism, [BCS09, Definition 1.1.1])**.** A morphism $f : A \to B$ in a cartesian left-additive category is **additive** if it preserves the monoid structure, i.e. the following diagrams commute:

In equations, these diagrams unpack to $f(a +_A a') = f(a) +_B f(a')$ and $0_B = f(0_A)$. Additive morphisms form a subcategory of $\mathcal{C}$.

Definition 2.20. Let $\mathcal{C}$ be a cartesian left-additive category. We use $\mathbf{CMon}(\mathcal{C})$ to denote its subcategory of additive morphisms i.e. the category of commutative monoids and commutative monoid homomorphisms in $\mathcal{C}$.

Analogously to how a category where all maps preserve the underlying comonoid structure is cartesian (i.e. has products), one where they preserve the underlying monoid strucure is *cocartesian* (i.e. has coproducts). In this case, products and coproducts coincide, making this is biproduct category.

Proposition 2.21 ($\mathbf{CMon}(\mathcal{C})$ is a biproduct category)**.** *See [CCG+20, Example 2.1].*

The categories $\mathbf{Poly}_{\mathbb{R}}$ and $\mathbf{Smooth}$ share the subcategory of additive maps, i.e. $\mathbf{CMon}(\mathbf{Poly}_{\mathbb{R}}) = \mathbf{CMon}(\mathbf{Smooth})$, and this subcategory is equivalent to the category $\mathbf{FVect}_{\mathbb{R}}$ of linear maps between finite-dimensional vector spaces ([IL23, Ex. 2.7]).

So far we have talked about additivity of a morphism with respect to the entirety of its domain. If we want to reason about additivity (or any other property) holding in *one variable* only, we can can make use of the previously mentioned category $\mathbf{coKl}(- \times X)$.

Definition 2.22 (Additive in second component, (compare [BCS09, Lemma 1.2.3]))**.** A morphism $f : X \times A \to B$ is additive in the variable A if it is an additive morphism of type $A \to B$ in the cartesian left-additive category $\mathbf{coKl}(X \times -)$.[11]

It can routinely be shown by unpacking the condition of an additive morphism that the condition of additivity here ends up pertaining only to the component A.

[11] See Section 3.1.2 for the definition of $\mathbf{coKl}(X \times -)$.

Peculiarities of the category $\mathbf{FVect}_{\mathbb{R}}$

The category $\mathbf{FVect}_{\mathbb{R}}$ is full of intricate structure. The one most relevant to us are its multple monoidal structures. Products in $\mathbf{FVect}_{\mathbb{R}}$ (the ones defined as in $\mathbf{Smooth}$) here also become coproducts, making $\mathbf{FVect}_{\mathbb{R}}$ a biproduct category. Often this biproduct is denoted with $(\oplus, \mathbb{R}^0)$ since it can computed by taking the coproduct of the basis sets. Taking the *product* of the basis sets gets us the other important monoidal structure on $\mathbf{FVect}_{\mathbb{R}}$: the tensor product $(\otimes, \mathbb{R})$. This is a monoidal structure that is neither cartesian nor cocartesian, but has a separate universal property.

This universal property states that $U \otimes V$ — an object of $\mathbf{FVect}_{\mathbb{R}}$ — comes equipped with a bilinear map $U \times V \to U \otimes V$ (this map is often also denoted by $\otimes$ since it takes two vectors and computes their tensor product) such that for any other object X and any other bilinear map $U \times V \to X$ there is a unique *linear* map $U \otimes V \to X$ making the diagram commute.[12]

$$\begin{array}{ccc} U \times V & \xrightarrow{\text{bilinear}} & U \otimes V \\ & \searrow^{\text{bilinear}} & \downarrow{\scriptstyle \text{linear}} \\ & & X \end{array} \tag{2.23}$$

The category $\mathbf{FVect}_{\mathbb{R}}$ is closed with respect to this monoidal structure, where $\underline{\mathbf{FVect}_{\mathbb{R}}}(\mathbb{R}^n, \mathbb{R}^m) = \mathbb{R}^{n \times m}$. There is an identity-on-objects faithful functor $\iota : \mathbf{FVect}_{\mathbb{R}} \to \mathbf{Smooth}$, making $\mathbf{FVect}_{\mathbb{R}}$ a subcategory of $\mathbf{Smooth}$. As $\mathbf{Smooth}$ is cartesian, and $\mathbf{FVect}_{\mathbb{R}}$ has two monoidal structures, it becomes natural to ask whether ι is a monoidal functor. This is indeed the case: this functor becomes a monoidal functor in two different ways depending on the monoidal structure of the domain. With $(\oplus, \mathbb{R}^0)$ this monoidal functor becomes a product-preserving functor, meaning it is strong monoidal. Interestingly, with the tensor product on the domain ι is only lax monoidal.

Lemma 2.24. *The functor $\iota : (\mathbf{FVect}_{\mathbb{R}}, \otimes, \mathbb{R}) \to (\mathbf{Smooth}, \times, \mathbb{R}^0)$ is lax monoidal, and its data is defined as follows.*

- *The laxator $\phi_{\mathbb{R}^n, \mathbb{R}^m}$, as ι is identity-on-objects, unpacks to a map of type $\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n \otimes \mathbb{R}^m$. It is given by the bilinear structure map in Eq. (2.23) defining the tensor product of vector spaces. That is, the laxator $\phi_{\mathbb{R}^n, \mathbb{R}^m}(u, v) = uv^\top$ computes the outer product of u and v.*
- *The unitor unpacks to a map of type $\mathbb{R}^0 \to \mathbb{R}$, and is given by the constant map at the multiplicative identity $1 : \mathbb{R}$.*

[12] This diagram, perhaps surprisingly, does not live in $\mathbf{FVect}_{\mathbb{R}}$, but rather in $\mathbf{Set}$, or more precisely in the multicategory of finite dimensional vector spaces and multilinear maps, which we only mention for completeness.

2.2.4 Notation

We use uppercase letters $A, B, \dots$ to denote objects of categories and lowercase letters $f, g, \dots$ to denote their morphisms. The identity morphism on an object A is denoted by id_A or simply just A, trusting that is clear from the context that A refers to a morphism, and not the object. When defining composite morphisms, we often use the blueprint

$$f \coloneqq \boxed{A \xrightarrow{f_1} B \xrightarrow{f_2} C \xrightarrow{f_3} D}$$

to define $f : A \to D$ as the composite of f_1, f_2 and f_3. Otherwise, we use the diagrammatic notation, i.e. $f = f_1 \fatsemi f_2 \fatsemi f_3$.

We use the boldface font for named categories ($\mathbf{Set}, \mathbf{FVect}_F, \mathbf{Cat}, \dots$), otherwise we use calligraphic letters ($\mathcal{C}, \mathcal{D}, \dots$). In a 2-category, we use use $\to$ to denote 1-morphisms, and $\Rightarrow$ to denote 2-morphisms. In $\mathbf{FVect}_F$ we often use $\multimap$ instead of $\to$ as arrows to emphasize the property of morphisms being linear.

In categories with products we label projections using π, subscripted with either indices, or names of objects. For instance, the projection $A \times B \times C \to C$ might be written as π_3 or π_C. For a composite $g \fatsemi \pi_C$ we sometimes write f_3 to save space. For categories with coproducts we label injections using i analogously, i.e. the injection $B \to A + B + C$ might be labelled either as i_2 or i_B. We sometimes write pairs as $\binom{A}{A'}$ to emphasize their usage in a bidirectional setting: the top object A is the forward object, while the bottom object A' is the backward one. We always additionally mark the object of the backward pass with a superscript $'$. For composition of parametric morphisms we often use the superscript $^{\mathsf{p}}$, while for coparametric ones use the subscript $_{\mathsf{p}}$. This applies to both sequential ($\fatsemi^{\mathsf{p}}$, $\fatsemi_{\mathsf{p}}$) and parallel ($\boxtimes^{\mathsf{p}}$, $\boxtimes_{\mathsf{p}}$) composition.[13]

We call the structure morphisms of both lax monoidal functors and lax functors "unitors" (when it comes to identities) and "laxators" (when it comes to composition). We use $\mathcal{C}/A$ to denote the slice category over $A : \mathcal{C}$. When referring to internal hom objects of some category $\mathcal{C}$, we write $\underline{\mathcal{C}}(A, B)$. We write 1 for the set $\{\bullet\}$ with one element, $\mathbf{1}$ for the category with one object and one morphism, and, in a category with a terminal object, $!_A$ for the unique morphism from some object A to the terminal one. Given an element of a set $a : A$, we use $\lceil a \rceil : 1 \to A$ for the "name" of a: the function picking out that element in A.

[13] A good way to remember this is via their graphical languages: reparameterisations in **Para** are composed from the top, while reparameterisations in **coPara** from the bottom.

Chapter 3

Parameterisation

> The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.
>
> Edsger W. Dijkstra

Thought of as processes, the forward passes of neural networks have a few distinguishing characteristics.

Firstly, the space of neural network weights i.e. their parameter space is an additional type of inputs to this process, playing a special role. For instance, no matter whether we compose neural networks in sequence, or in parallel, the parameter spaces are always composed in parallel. Likewise, it is also possible to reparameterise neural networks, for instance by tying weights together, or by reindexing them to a more convenient space. All of these compositions and reparameterisations satisfy particular properties. For instance, reparameterising two morphisms individually and then composing the results is equal to composing the original morphisms and applying the product of original reparameterisations. Or, reparameterising a morphism twice with different morphisms is equal to reparameterising it once with their composite. But are there others? Is there a compact way to describe this algebraic structure, and all the underlying invariants it possesses?

Second, the way neural networks are specified in the literature is often needlesly specific, or unnecessarily vague. For example, the description of forward passes often relies on euclidean

spaces, despite neural networks being well-defined in complex-valued spaces ([BQL21]), graphs ([WPC+21]), simplicial and cellular complexes ([PSHM23]), and even boolean circuits ([WZ21]). What minimal structure do we need to talk about proceses with parameters, and their algebra of composition?

And lastly, in many different fields there are analogous concepts of "hidden" inputs:

- They manifest as weights in neural networks ([FST21, CGG+22]);
- They represent strategies of players in game theory ([CGHR22, Cap23]);
- They form closures of programs ([New17]);
- They serve as sources of randomness in probabilistic programming ([HKSY17]);

In all of them we can reason about notions of reparameterisation, where for instance weight tying constraints two players to play the same strategy, or in probabilistic programming we can bind two random variables together.

Is there a uniform formalism that captures all of these? Is there a formalism that is stable across changes of architectures, and one that will later permit dealing with the flow of gradients going backwards? Likewise, analogous to the graphical language of string diagrams for processes in monoidal categories, is there an analogous graphical language for parametric processes?

In this thesis we give a positive answer to all of these, using the construction of *actegories* (Section 3.1) and the **Para** construction (Section 3.1.1).

Remark 3.1. I've made the pedagogical decision not to throw all the actegory and **Para** constructions and the definitions at the reader in this chapter. Instead, they appear throughout the next chapter as well, thoroughly motivated.

Contributions. *Large fragments of this chapter contain novel research. While the definition of actegories and the definition of* **Para** *are not novel, a lot of the surrounding research around it is. This includes notions of reparameterisation of actegories (Notation 3.11), strictification and quotients of* **Para** *(Boxes 3.1.1, 3.1.2 and 3.2.2), trivialisation condition (Theorem 3.28), correspondence between local and global contexts (Section 3.1.2), monoidal structure of actegories and* **Para** *(Sections 3.2 and 3.2.1). Parts of this chapter previously appeared in my publications [CG23, CGHR22].*

Epistemic status. *The fact that the formulation of* **Para** *yields a relatively elegant construction makes me confident that the research here is on the right track, especially when it comes to the emphasis on its 2-dimensional stucture as something that should be explicitly tracked, instead of quotiented out. On the other hand, I expect its formulation via actegories to be cleaned up. In Section 3.3 I describe two independent generalisations of actegories: locally graded categories and dependent actegories, both of which I believe remove many artifical identifications that are made in the actegorical framework. Likewise, while the 2-dimensional graphical language of* **Para** *is freely used, no formal rules for this particular kind of a graphical language have been established in the literature, nor has its connection to the graphical language of double categories been studied. There is therefore some uncertainty about the actual boundaries of this language, despite which its use in the literature has only increased.*

3.1 Actegories and the Para construction

The goal of this section is to establish the categorical structure necessary to reason about parametric processes.

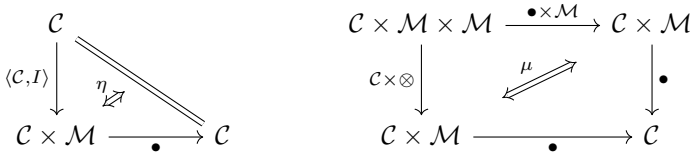
In essence, we aim to construct a category where a morphism $A \to B$ encapsulates both a parameter space P and the "implementation" map of type $f : A \otimes P \to B$. Since the domain is a product, this necessitates the provision of a underlying monoidal category C. For instance, this would allow us to reason about parametric maps $f : A \otimes (P \otimes Q \otimes R) \to B$ where A and B are classical inputs and outputs, respectively, while $P \otimes Q \otimes R$ is the parameter space. However, this does not capture the full extent of what we can do with parameters: it falls short in scenarios where the types of parameters and the types of standard inputs differ.

This calls for an additional layer of refinement — one provided by "actegories", a term we use to describe actions of monoidal categories ([CG23, JK01]). Actegories are not a new construction. They have previously appeared in numerous incarnations in the literature (see Section 3.3). Actegories will serve as our formal framework to distinguish the types of parameters from the types of usual inputs. We will see how properties of actegories translate to properties of various parametric morphisms, and many of the theorems here will also shed light on our subsequent discussions on bidirectionality (Chapter 4). We give their description below, after which we describe how they can be used to talk about parametric morphisms.

Definition 3.2 (Actegory ([CG23, JK01]))**.** Let $(\mathcal{M}, \otimes, I)$ be a monoidal category. A $(\mathcal{M}, \otimes, I)$-actegory $\mathcal{C}$ consists of a category $\mathcal{C}$ equipped with a functor

$$\bullet : \mathcal{C} \times \mathcal{M} \to \mathcal{C}$$

and two natural isomorphisms called **the unitor** and **the multiplicator**



(3.3)

whose components at each $C : \mathcal{C}$ and $M, N : \mathcal{M}$ are explicitly the maps

$$\eta_c : C \xrightarrow{\cong} C \bullet I \qquad \text{and} \qquad \mu_{C,M,N} : C \bullet (M \otimes N) \xrightarrow{\cong} (C \bullet M) \bullet N \tag{3.4}$$

satisfying the coherence laws defined in Definition A.22.

Notation 3.5. We often refer to the above actegory as an "$\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$", leaving $\otimes$, I, η and μ implicit. When multiple actegories and their natural isomorphisms are in scope, we might refer to the natural isomorphisms as $\eta^\bullet$ and $\mu^\bullet$, superscripting the relevant action functor.

This is often called a *right* $\mathcal{M}$-actegory, because we think of $\mathcal{M}$ as acting on $\mathcal{C}$ from the right side. That is, acting with $M \otimes N$ on C from the right side multiplies C first by M, and then by N (giving us $(C \bullet M) \bullet N$). One can analogously define a *left* $\mathcal{M}$-actegory $\mathcal{C}$, where the action happens from the left side, i.e. where acting with $M \otimes N$ on C would multiply C first by N, and then by M (giving us $M \bullet (N \bullet C)$). Categorically, the side from which "an action happens" is defined by the orientation of the monoidal product on $\mathcal{M}$: in addition to $(\mathcal{M}, \otimes, I)$ we can always form *the reversed monoidal product* (Definition A.7). Then a right actegory defined on the monoidal category $(\mathcal{M}, \otimes^{\mathsf{rev}}, I)$ is in the literature defined as a left $(\mathcal{M}, \otimes, I)$-actegory. ([CG23, Remark 3.1.3])

Even though we don't need any extra structure to *define* right actegories in terms of left ones, in order to *turn* a specific right actegory into a left one we will additionally need $\mathcal{M}$ to be braided monoidal.

Lemma 3.6. *If $(\mathcal{M}, \otimes, I)$ is a braided monoidal category, then any right $(\mathcal{M}, \otimes, I)$-actegory can be turned into a left $(\mathcal{M}, \otimes^{\mathsf{rev}}, I)$-actegory, and vice-versa.*

Proof. We leave the full proof to the appendix (Appendix A), here only mentioning that the braiding is required only when defining the multiplicator of this left-actegory, as it is the only component of the actegory interacting with the monoidal structure of $\mathcal{M}$. □

In braided settings we will often omit any notational distinction between the two actions.

Remark 3.7. Even though actegories can be defined without braiding on the base, we will see that in order to state most theorems about them braiding will be necessary, and — as it turns out — existent in all our applications. Despite the equivalence of left and right actions under braiding, we will often make a notational distinction between them in the context of **Para** and **coPara** constructions (Section 3.1.1 and Section 4.2). This is inspired by Section 3.3 where we will briefly descibe a generalisation of actegories called *lax actegories* where such equivalence is nonexistent, and also by the general principle of being cognisant of the arrow of time: things which appear first in time will appear first in notation, reading left to right.

Example 3.8. If $\mathcal{C}$ is a monoidal category, then we can consider it as acting on itself from either left or right. From the right side, the action is defined by $\otimes$, the unitor ρ^{-1} and the multiplicator α^{-1}. From the left side, the action is defined by $\otimes^{\mathsf{rev}}$, the unitor λ^{-1} annd the multiplicator α.

Examples abound. For us the most relevant will be **Set**, **FVect**, and **Smooth**, both of which have at least one monoidal structure. Actegories allow us to additionally restrict the type of parameters involved in the action, something that is not possible within the framework of just monoidal categories.

Example 3.9. Let $\mathcal{C}$ be a monoidal category, and $\mathcal{B}$ a monoidal category which comes equipped with a strong monoidal functor $E : \mathcal{B} \to \mathcal{C}$. Then we can form a $\mathcal{B}$-actegory $(\mathcal{C}, \bullet)$ with $\bullet$ defined as the composite $\mathcal{C} \times \mathcal{B} \xrightarrow{\mathcal{C}\times E} \mathcal{C} \times \mathcal{C} \xrightarrow{\otimes} \mathcal{C}$, where $\otimes$ is the monoidal product of $\mathcal{C}$.

The above construction is a good example of a *reparameterisation of actegories* which works for general action, and not just a self-action.

Definition 3.10 (Reparameterisation of actegories, compare [CG23, Prop. 3.6.1])**.** Let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$-actegory, and $E : \mathcal{N} \to \mathcal{M}$ a strong monoidal functor. Then we can define a $\mathcal{N}$-actegory $(\mathcal{C}, \bullet^E)$ where

$$\bullet^E := \boxed{\mathcal{C} \times \mathcal{N} \xrightarrow{\mathcal{C}\times E} \mathcal{C} \times \mathcal{M} \xrightarrow{\bullet} \mathcal{C}}$$

If the functor E was merely lax monoidal, the actegory would only be lax (Section 3.3) too.

Notation 3.11. We highlight an important piece of notation: reparameterisations will be written with a superscript, i.e. as $\bullet^E = \mathcal{M}(-, E(-))$. This notation will make continued appearance in other kinds of reparameterisations that we will encounter, especially as 2-morphisms in forthcoming bicategories like **Para** (Definition 3.14).

Example 3.12 (Markov kernels and expectations)**.** Let **Mark** be the monoidal category of Markov kernels (Definition 2.5 and Example 2.14), and **Conv** be the monoidal category of convex sets[1], where $\Delta : \mathbf{Mark} \to \mathbf{Conv}$ is a strong monoidal functor with respect to these monoidal structures ([HS23, Ex. 4]). Here Δ is an example of a reparameterisation of actegories, turning the self-action of **Conv** (labelled with $\otimes'$) into an action of **Mark** defined as ${\otimes'}^{\Delta} : \mathbf{Conv} \times \mathbf{Mark} \to \mathbf{Conv}$.

Lastly, we mention the trivial action that can be defined for any category, and note that many more examples can be found in [CG23, Section 3.2].

Example 3.13 (Trivial action, [CG23, Ex. 3.2.1])**.** Every category $\mathcal{C}$ has a trivial action of the terminal category **1** on it, and this action $\bullet : \mathcal{C} \times \mathbf{1} \to \mathcal{C}$ is an isomorphism.

In the next chapter we will see that monoidal, cartesian and closed structures can be all generalised and stated in the actegorical setting.

3.1.1 The Para construction

Having described actegories, we will now see how to use them to construct a category whose morphisms are parametric in the way that the actegory describes. This will be done with the **Para** construction which takes in an actegory and produces a *bicategory*, where the 2-morphisms will model reparameteristaions, and whose bicategory laws completely capture all the necessary invariants we might ask of these parametric maps. Para is also not a new construction, and has appeared in numerous incarnations in the literature (see Section 3.3).

Definition 3.14 (Parametric maps, compare [CGHR22, Def. 2])**.** Let $(\mathcal{M}, \otimes, I)$ be a monoidal category, and let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$-actegory. We define the bicategory $\mathbf{Para}_{\bullet}(\mathcal{C})$ with the following data:

- **Objects** are those of $\mathcal{C}$;

[1] A convex set is a set X with an abstract expectation operator, and a morphism of convex sets is given by an algebra morphism of the finite support probability monad. For more details see [HS23, Ex. 4]

- **Morphisms** are often referred to as *parametric morphisms*. A parametric morphism of type $A \to B$ is a pair (P, f) where we think of $P : \mathcal{M}$ as its *parameter space* and $f : A \bullet P \to B$ in $\mathcal{C}$ as its *implementation*. Every parametric morphism has a horizontal, but also a vertical component, emphasised by its string diagram representation (Fig. 3.1).

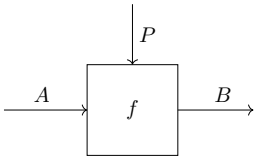
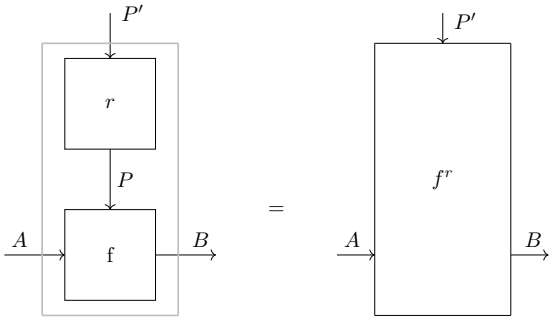


Figure 3.1: String diagram representation of a parametric morphism. We often draw the parameter wire on the *vertical* axis to signify that the parameter object is part of the data of the morphism.

- **2-morphisms** are called *reparameterisations*. A reparameterisation from $(P, f) \Rightarrow (P', f')$ is a morphism $r : P' \to P$ in $\mathcal{M}$ such that the diagram in Eq. (3.15) commutes[2] in $\mathcal{C}$. Following Notation 3.11 we will often write f^r for the reparameterisation of f with r.



$$\begin{array}{ccc} A \bullet P' & \xrightarrow{f'} & B \\ {\scriptstyle A \bullet r}\downarrow & \nearrow_{f} & \\ A \bullet P & & \end{array} \tag{3.15}$$

Figure 3.2: String diagram of reparameterisation. The reparameterisation map r is drawn vertically.

- **Identity morphism.** For every object A there is an identity parametric map $(I, {\eta_A}^{-1})$ where $I : \mathcal{M}$ and $\eta_A : A \bullet I \to A$ is the unitor of the underlying actegory.

- **Morphism composition.** The composition of parametric morphisms

$$A \xrightarrow{(P,f)} B \xrightarrow{(Q,g)} C$$

[2] Observe that the r goes in the opposite direction of the 2-cell. See the variance of $\mathcal{M}$ in Prop. 3.23.

i.e. of

$$P : \mathcal{M} \qquad A \bullet P \xrightarrow{f} B \quad \text{in} \quad \mathcal{C} \qquad \text{and} \qquad Q : \mathcal{M} \qquad B \bullet Q \xrightarrow{g} C \quad \text{in} \quad \mathcal{C}$$

is the pair $(P \otimes Q, f \fatsemi^{\mathbf{p}} g)$ where

$$P \otimes Q : \mathcal{M}$$

$$f \fatsemi^{\mathbf{p}} g \coloneqq \boxed{A \bullet (P \otimes Q) \xrightarrow{\mu_{A,P,Q}} (A \bullet P) \bullet Q \xrightarrow{f \bullet Q} B \bullet Q \xrightarrow{g} C} \qquad \text{in} \quad \mathcal{C}$$

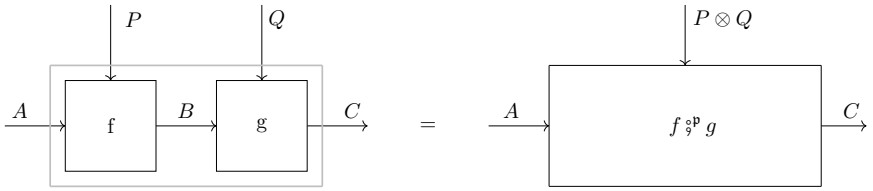


Figure 3.3: String diagram representation of the composition of parametric morphisms. By treating parameters on a separate axis we obtain an elegant graphical depiction of their composition.

- **Horizontal and vertical composition of 2-morphisms** is given by parallel and sequential product of $\mathcal{M}$, respectively.

It is routine to verify that the unitor and associator data — defined in the only possible way (and remarked upon in Box 3.1.1) — satisfy the unity and the pentagon axioms ([JYJY21, Def. 2.1.3]). This concludes the definition of the bicategory $\mathbf{Para}_{\bullet}(\mathcal{C})$.

Notation 3.16. For a self-action of a monoidal category $\mathcal{C}$ we'll sometimes omit the subscript of the action, writing just $\mathbf{Para}(\mathcal{C})$ and trusting the monoidal structure is clear from the context.

Example 3.17 ($\mathbf{Para}(\mathcal{C})$ for monoidal $\mathcal{C}$)**.** Any monoidal category $\mathcal{C}$ gives rise to $\mathbf{Para}(\mathcal{C})$. In this case a parametric morphism $A \to B$ is a pair (P, f) where $f : P \otimes A \to B$. If $\mathcal{C}$ is cartesian, then $f : P \times A \to B$ consumes both the input and the parameter. If $\mathcal{C}$ is cocartesian, then $f : P + A \to B$ consumes *either* the input or the parameter.

Monoidal categories abound, such as **Set**, **Smooth**, **FVect**. The construction **Para** can be instantiated in any of those with any of their monoidal products. In Chapter 7 (specifically Defini-

tions 7.7 and 7.8 and Section 7.1.1), we will show how to model standard neural network layers as morphisms in **Para**(**Smooth**).

Remark 3.18. The graphical language we used for depicting parametric morphims is slightly different than the graphical language of classical string diagrams. Consider a parametric map $f : A \otimes P \to B$. With classical string diagrams we might draw them as on the left. However this notation does not emphasise the special role played by the parameter P: it is part of the data of *the morphism*. By separating them on two different axes (as on the right) we obtain the graphical language which more closely mirrors their semantics.

An important class of parametric morphisms arises out of subcategory actions (Example 3.9).

Example 3.19. Let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$-actegory and let $E : \mathcal{N} \to \mathcal{M}$ be a strong monoidal functor. Then we can form $\mathbf{Para}_{\bullet^E}(\mathcal{C})$. Here a parametric morphism $A \to B$ is a pair (P, f) where P lives in $\mathcal{N}$ (not $\mathcal{M}$!) and the implementation f is of type $E(P) \otimes A \to B$. Reparameterisations, like parameters, live strictly in $\mathcal{N}$.

From bicategories to 2-categories **Box 3.1.1**

As defined, $\mathbf{Para}_\bullet(\mathcal{C})$ is a bicategory, i.e. a weak 2-category. This is easy to see: take any parametric morphism $(P, f) : \mathbf{Para}_\bullet(\mathcal{C})(A, B)$ and postcompose it with the identity on B, i.e. $(I, \eta_B^{-1}) : \mathbf{Para}_\bullet(\mathcal{C})(B, B)$. The result of this is a map with the parameter object $P \otimes I$. This is different from P, as the unitor law of a 2-category would require.[a] This is because an arbitrary monoidal category isn't necessarily *strict*, i.e. objects $P \otimes I$ and P are not necessarily equal, but only isomorphic. This leads us to ponder — is strictness of $\mathcal{M}$ all we need to turn $\mathbf{Para}_\bullet(\mathcal{C})$ into a 2-category? Or for example, do the unitor η and the multiplicator μ of the actegory also need to be strict? As it turns out, strictness of $\mathcal{M}$ is all we need.

Definition 3.20 (Strict actegory (compare [CG23, Subsection 3.4.]))**.** A $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$ is called **strict** if $\mathcal{M}$ is strict monoidal.

Example 3.21 (Strict self-action)**.** Any strict monoidal category give rise to a strict self-action from both the left and the right side.

Strict actegories allow us to state the following proposition (proved in Appendix C) reducing a bicategory to a 2-category.

Proposition 3.22. *If the $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$ is strict, then $\mathbf{Para}_\bullet(\mathcal{C})$ is a 2-category.*

[a] Similar story arises with associativity; composing three parametric morphisms yields either $(P \otimes Q) \otimes R$ or $P \otimes (Q \otimes R)$ as the parameter object, depending on the order of composition.

On a similar note, if $\mathcal{M}$ is a discrete monoidal category (i.e. a monoid), then $\mathbf{Para}_\bullet(\mathcal{C})$ is a category. This follows because 2-morphisms in **Para** only exist when morphisms in $\mathcal{M}$ do, meaning that, if $\mathcal{M}$ has only identity morphisms, then $\mathbf{Para}_\bullet(\mathcal{C})$ has only identity 2-morphisms.

A good way to understand this brings us to another important property of $\mathbf{Para}_\bullet(\mathcal{C})$: its hom-categories are categories of elements. This proposition will be instrumental in Chapter 4.

Proposition 3.23. *For given objects A and B, the hom-category $\mathbf{Para}_\bullet(\mathcal{C})(A, B)$ can be computed as a category of elements (Definition D.1) of the functor $\mathcal{C}(A \bullet -, B) : \mathcal{M}^{\mathrm{op}} \to \mathbf{Set}$. That is,*

$$\mathbf{Para}_\bullet(\mathcal{C})(A, B) = \mathbf{El}(\mathcal{C}(A \bullet -, B)) = \sum_{P : \mathcal{M}^{\mathrm{op}}} \mathcal{C}(A \bullet P, B)$$

We can see that, if $\mathcal{M}$ is discrete, then any category of elements formed over it must be discrete too. Though, not all categories that go under the name **Para** in the literature arise from a discrete action.

From 2-categories to categories **Box 3.1.2**

Throughout its brief history, $\mathbf{Para}_\bullet(\mathcal{C})$ wasn't always thought of as a 2-category.[a] Starting humbly as a category, it was defined a number of times, each with its own distinct flavour of morphisms. We list all of them below, and show how they arise as a shadow of the 2-categorical perspective, i.e. as an image of the enriched base change $F_* : \mathbf{2Cat} \to \mathbf{Cat}$ for a given lax monoidal functor $F : (\mathbf{Cat}, \times, 1) \to (\mathbf{Set}, \times, 1)$. Each choice of such a functor describes how we turn the hom-category of parametric morphisms $A \to B$ into a hom-*set*.

Example 3.24 (Discretisation, $\mathsf{Ob} : \mathbf{Cat} \to \mathbf{Set}$)**.** Simply ignores the 2-cells. This results in a set of parametric morphisms that are only related by strict equality in $\mathbf{Para}_\bullet(\mathcal{C})$.

This prohibits us from modelling the parameter update step of neural networks with gradient descent, or other gradient updates (Section 8.1.3).

Example 3.25 (Connected components, $\pi_0 : \mathbf{Cat} \to \mathbf{Set}$)**.** Identifies any two morphisms connected by a (potentially non-invertible) reparameterisation.

This is a strong condition. Via Prop. E.6 this is equivalent to computing the local colimit of the functor $\mathcal{C}(A \bullet -, B)$, instead of its category of elements (Prop. 3.23). It's used in [DL19] where the colimit is expressed as the coend mute in one variable. In Theorem 3.28 we describe conditions under which this quotient trivialises the parametric category.

Example 3.26 (Isomorphisms, $\mathbf{Cat} \xrightarrow{\mathsf{Core}} \mathbf{Cat} \xrightarrow{\pi_0} \mathbf{Set}$)**.** Identifies two morphisms if they are connected by an invertible reparameterisation.

Used in [FST21, HL21, Gav19]. This is one of the better behaved quotients.

Example 3.27 (Epimorphisms, $\mathbf{Cat} \xrightarrow{\mathsf{Epi}} \mathbf{Cat} \xrightarrow{\pi_0} \mathbf{Set}$)**.** Identifies two morphisms if they are connected by a reparameterisation which is an epimorphism.

Used in [Spi22b] without justification.

[a] Here we assume a strict actegory (Box 3.1.1).

The examples above suggest that attempts to simplify the 2-categorical structure of **Para** into a category generally ends up losing important information relevant to modelling deep learning. $\mathbf{Para}_\bullet(\mathcal{C})$ is truly a higher-dimensional beast!

Information is tangibly lost under the connected components quotient if the monoidal unit of $\mathcal{M}$ is initial, which happens for self-actions of cocartesian categories. Quotienting $\mathbf{Para}_\bullet(\mathcal{C})$ here trivialises the entire structure.

Theorem 3.28. *Let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$-actegory, where the monoidal unit of $\mathcal{M}$ is initial. Then*

$$\pi_{0*}(\mathbf{Para}_\bullet(\mathcal{C})) \cong \mathcal{C}$$

Proof. As they both share objects, all we have to prove is that there is a one-to-one correspondence between their hom-sets that respects composition and identities. The hom-set isomorphism follows from the fact that $\mathcal{M}$ has an initial object (which is terminal in $\mathcal{M}^{\mathsf{op}}$ allowing us to reduce the colimit) and the fact that this initial object is the monoidal unit (allowing us to further simplify the term using the actegory unitor).[3]

$$\begin{aligned}
& & \pi_{0*}(\mathbf{Para}_\bullet(\mathcal{C})(A, B)) \\
&\text{(Def.)} & = \pi_0(\mathbf{El}(\mathcal{C}(A \bullet -, B))) \\
&\text{(Prop. E.6)} & \cong \mathsf{colim}(\mathcal{C}(A \bullet -, B)) \\
&\text{(Prop. E.5)} & \cong \mathcal{C}(A \bullet 0, B) \\
&\text{(Eq. (3.4), left)} & \cong \mathcal{C}(A, B)
\end{aligned}$$

It is routine to show that this respects composition and identities. □

Intuitively, this means that any parametric morphism $f : A \bullet M \to B$ in $\mathbf{Para}_\bullet(\mathcal{C})$ can be precomposed with the initial object $0_M : 0 \to M$ yielding $f^{0_M} : A \bullet 0 \to B$. If 0 is the unit of the monoidal product, then f^{0_M} and f are equivalent in $\pi_{0*}(\mathbf{Para}_\bullet(\mathcal{C}))$. This theorem has an important corollary in terms of self-actions of a cocartesian category.

Corollary 3.29. *Let $\mathcal{C}$ be a cocartesian category. Then $\pi_{0*}(\mathbf{Para}(\mathcal{C})) \cong \mathcal{C}$.*

Note that this doesn't necessarily hold for a cartesian category. Consider the category $\mathbb{N}$ (Definition 2.6). This category is cartesian where $m \times n = \max(m, n)$ and where the terminal

[3] The converse doesn't necessarily hold, as the converse of Prop. E.5 doesn't necessarily hold.

object is 0. A parametric morphism $a \to b$ in **Para**$(\mathbb{N})$ consists of a number $p : \mathbb{N}$ and a morphism $f : \max(a, p) \to b$, i.e. an inequality $\max(a, p) \geq b$. Therefore, even though $\mathbb{N}(0, 1) = \emptyset$, $\pi_0(\textbf{Para}(\mathbb{N})(0, 1)) \neq \emptyset$ as it is inhabited, for example by $42 : \mathbb{N}$ and an inequality $\max(0, 42) \geq 1$.

3.1.2 Local vs. global contexts

An instructive way to understand **Para**$_\bullet(\mathcal{C})$ is as a model of processes which have access to a *local* context. Effectively, this means that every morphism can choose its own parameter space, and that composition of morphisms tensors the parameters together. This is in contrast to categories which model a *global* context, which we now describe, and compare to **Para**.

A family of such categories can be formed by considering the coKleisli categories over comonads formed on a base cartesian category $\mathcal{C}$. For each $X : \mathcal{C}$ there is a one such comonad in the literature called the coreader comonad.[4] It is given by the functor $- \times X : \mathcal{C} \to \mathcal{C}$ and corresponding counit and comultiplication natural transformations defined in [GV22, Def. 4]. Its coKleisli category has the following form.

Definition 3.30 (Category **coKl**$(- \times X)$)**.** The category **coKl**$(- \times X)$ has the following data.

- **Objects** are those of $\mathcal{C}$;
- **Morphisms** are X-parametric morphisms; a map $A \to B$ in **coKl**$(- \times X)$ consists of a map $f : A \times X \to B$;
- **Morphism composition.** The composition of morphisms

$$A \xrightarrow{f} B \xrightarrow{g} C$$

i.e. of

$$A \times X \xrightarrow{f} B \quad \text{in} \quad \mathcal{C} \qquad \text{and} \qquad B \times X \xrightarrow{g} C \quad \text{in} \quad \mathcal{C}$$

is map $(f \fatsemi^{\mathsf{p}} g)^{\Delta_X}$ defined using the notion of reparameterisation (Fig. 3.2), i.e.

$$(f \fatsemi^{\mathsf{p}} g)^{\Delta_X} = \boxed{A \times X \xrightarrow{A \times \Delta_X} A \times (X \times X) \xrightarrow{\cong} (A \times X) \times X \xrightarrow{f \times X} B \times X \xrightarrow{g} C} \quad \text{in} \quad \mathcal{C}$$

[4] The coreader comonad is also sometimes referred to as the "the writer comonad" (though this is often confused with the writer *monad* which additionally requires X to have a monoid structure), "the reader comonad" (because of the adjunction $- \times X \dashv \underline{\mathcal{C}}(X, -)$ when $\mathcal{C}$ is cartesian closed), "product comonad" or "environment comonad".

- **Identity.** Identity on A is the projection $\pi_A : A \times X \to A$.

We see that every morphism in $\mathbf{coKl}(- \times X)$ has access to an additional *global* parameter X, and *only* this parameter. A composition of multiple X-parameterised morphisms does not take the product of individual X objects, instead it is still parameterised by only one X. What this composition then does is relay the input X to its constituents by copying (Fig. 3.4), which is where the requirement that the base category $\mathcal{C}$ is cartesian arises. Compare this to **Para** where each morphism can choose its type of parameter. And when the morphisms are composed, the total parameter space gets bigger. For more details on this comparison, we invite the reader to [GV22, Sec. 3.1.1]. The coKleisli construction will be relevant in studying weight tying (Definition 7.11) and graph convolutional neural networks (Section 7.4).

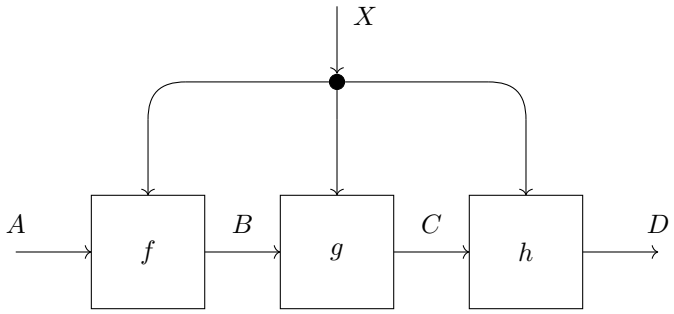


Figure 3.4: Unlike **Para** composition which allows each morphism to be parameterised by an arbitrary object, the $\mathbf{coKl}(- \times X)$ construction fixes a specific object X as the parameter for all of them.

Since $\mathbf{coKl}(- \times X)$ models parametric processes, we would expect that this category is related to $\mathbf{Para}(\mathcal{C})$ in some way. That is indeed the case: $\mathbf{coKl}(- \times X)$ can be embedded in $\mathbf{Para}(\mathcal{C})$, but as there is this extra reparameterisation step we need to perform in composition and identities, this embedding is only *oplax*.

Lemma 3.31. *Let $\mathcal{C}$ be a cartesian category. Then for every object $X : \mathcal{C}$ there is an identity-on-objects oplax functor*

$$\mathbf{coKl}(- \times X) \to \mathbf{Para}(\mathcal{C})$$

mapping every morphism $f : \mathbf{coKl}(- \times X)(A, B)$ to $(X, f) : \mathbf{Para}(\mathcal{C})(A, B)$. The opunitor and oplaxator are respectively the counit $\epsilon_X : X \to 1$ and the comultiplication $\Delta_X : X \to X \times X$ of the unique comonoid on $\mathcal{C}$ arising from the cartesian structure.

This shows that **Para**$(\mathcal{C})$ can also model processes with a global context, and that special care has to be taken when composing them.

For completeness, we mention the variant with *no* context: this is simply an ordinary category $\mathcal{C}$. Unlike the previous case which gives us an oplax functor into **Para**$(\mathcal{C})$, this one gives us a pseudofunctor. We state it in a form not specific to just self-actions, but arbitrary ones.

Lemma 3.32 ($\mathcal{C}$ embeds into **Para**$_\bullet(\mathcal{C})$)**.** *Let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$ actegory. Then there is an identity-on-objects pseudofunctor*

$$\mathcal{C} \to \textbf{Para}_\bullet(\mathcal{C})$$

mapping a morphism $f : A \to B$ to the composite $A \bullet I \xrightarrow{\rho_A} A \xrightarrow{f} B$.[5] *Since all the parameter object is always I, it is easy to check that this is indeed a pseudofunctor.*

Furthermore, if $\mathcal{M}$ is strict, then this reduces to a 2-functor.

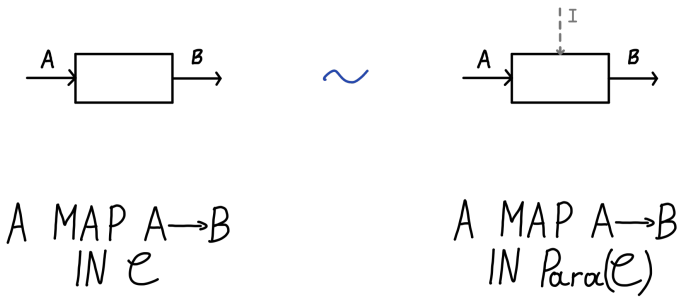


Figure 3.5: Every morphism in $\mathcal{C}$ is an I-parametric morphism in **Para**$_\bullet(\mathcal{C})$.

3.2 Monoidal actegories and monoidal Para

Categories describe processes that can be composed sequentially, and parametric categories describe parametric morphisms that can be composed sequentially. Just like making categories monoidal permits us to compose processes in parallel, we can ponder: are there *monoidal actegories* that allow us to compose parametric systems in parallel? In this section, we will see that the answer is yes. We begin by defining monoidal actegories and note that the acting monoidal category $\mathcal{M}$ is

[5] If the actegory is strict, then this becomes a mere 1-functor.

now required to be braided monoidal.[6]

Definition 3.33 (Monoidal actegory, compare [CG23, Def. 5.1.1.])**.** Let $(\mathcal{M}, \otimes, I)$ be a braided monoidal category. A $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$ is called a **monoidal $\mathcal{M}$-actegory** if the underlying category $\mathcal{C}$ has a monoidal structure $(\boxtimes, J)$ and the action respects that structure. This means that

- The underlying functor $\bullet : \mathcal{C} \times \mathcal{M} \to \mathcal{C}$ is strong monoidal;
- The underlying natural transformations η and μ are monoidal.

If $\mathcal{M}$ is a strict monoidal category, then we call $(\mathcal{C}, \bullet)$ a **strict** monoidal $\mathcal{M}$-actegory. If the functor is only (op)lax monoidal, then we call the actegory an **(op)lax** monoidal actegory.

Explicitly, the oplaxator of $\bullet$ is a map

$$c_{A,A',M,M'} : (A \boxtimes A') \bullet (M \otimes M') \cong (A \bullet M) \boxtimes (A' \bullet M') \tag{3.34}$$

which we call *the mixed interchanger*.[7]

Remark 3.35 (Why does $\mathcal{M}$ need to be braided?)**.** To state that μ is monoidal natural transformation, both the functors it is spanned between ($(\bullet \times \mathcal{C}) \fatsemi \bullet$ and $(\mathcal{C} \times \otimes) \fatsemi \bullet$) need to be monoidal. The latter is monoidal only if $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ itself is a monoidal functor, which happens only when $\mathcal{M}$ is braided. See Appendix A for more details.

Example 3.36 (Self-action of a braided monoidal category)**.** Any braided monoidal category $\mathcal{C}$ is also a monoidal $\mathcal{C}$-actegory with the interchanger defined using braiding. In this case the mixed interchanger reduces to the braided monoidal category interchanger (Definition A.10), effectively swapping the order of middle two components.

Examples of this are **Set**, **Smooth** and $\mathbf{FVect}_{\mathbb{R}}$, when instantiated with their cartesian product as the self-action. A large class of examples of monoidal actegories arises by reparameterisations induced by a braided monoidal functor.

Definition 3.37 (Reparameterisation of monoidal actegories)**.** In Definition 3.10 we have described how to reparameterise a $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$ with a strong monoidal functor $E : \mathcal{N} \to \mathcal{M}$. If the

[6] Despite symmetry of monoidal categories being widespread in our applications, we do not assume it for two reasons: the theory does not require it, and the notable exception of a monoidal category of used to form *affine traversals* does not possess it (Remark A.14).

[7] Given an actegory, it is the only additional piece of data that needs to be naturally defined in order to make it into a monoidal actegory. See [CG23, Remark 5.1.2.]

starting actegory is monoidal — implying $\mathcal{M}$ is braided — then the resulting actegory will be too if E is additionally braided (Definition A.11).

Example 3.38 (Markov kernels and expectation)**.** Example 3.12 yields a monoidal actegory since **Mark** is monoidal (Example 2.14), and the reparameterisation Δ : **Conv** $\to$ **Mark** is a strong braided monoidal functor.

3.2.1 Monoidal Para

Having discussed the monoidal structure of actegories, we are now ready to discuss the monoidal structure of the bicategory $\mathbf{Para}_\bullet(\mathcal{C})$.

Proposition 3.39 (Monoidal structure of **Para** (compare [CGHR22, Section 2.1.]))**.** *Let* $(\mathcal{C}, \bullet)$ *be monoidal* $\mathcal{M}$*-actegory with the underlying product* $(\boxtimes, J)$*. Then* $\mathbf{Para}_\bullet(\mathcal{C})$ *becomes a monoidal bicategory with the following data.*

- *The monoidal product of two objects* X, Y *is given by the monoidal product* $X \boxtimes Y$ *of* $(\mathcal{C}, \boxtimes, J)$*;*
- *The monoidal unit is the monoidal unit* J *of* $(\mathcal{C}, \boxtimes, J)$*;*
- *The monoidal product of two parametric morphisms*

$$(P, f : A \bullet P \to B) \quad \textit{and} \quad (Q, g : C \bullet Q \to D)$$

is given by $(P \otimes Q, f \boxtimes^{\mathbf{p}} g)$ *where*

$$f \boxtimes^{\mathbf{p}} g := \boxed{(A \boxtimes C) \bullet (P \otimes Q) \xrightarrow{{}^{C}_{A,C,P,Q}} (A \bullet P) \boxtimes (B \bullet Q) \xrightarrow{f \boxtimes g} B \boxtimes D}$$

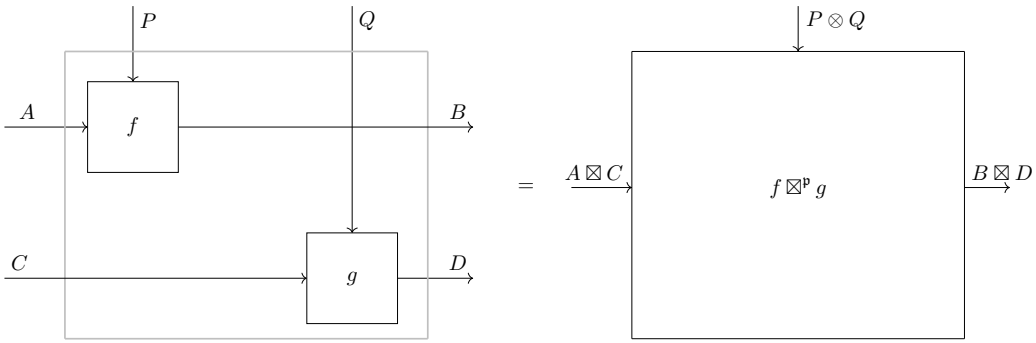


Figure 3.6: String diagram of the monoidal product of two parametric morphisms

- *The associators and unitors are those of* $(\mathcal{C}, \boxtimes, J)$.

Examples of the monoidal bicategory **Para** arise out of any action that is monoidal, such as cartesian and cocartesian self-actions, but also their subactions (Example 3.9), and any reparameterisations along braided monoidal functors.

When reasoning about this monoidal structure, we rely on a series of simplifications.

From monoidal bicategories to monoidal 2-categories **Box 3.2.1**

In Box 3.1.1 we have seen that the bicategory $\mathbf{Para}_\bullet(\mathcal{C})$ becomes a 2-category under an additional condition of the underlying actegory: strictness (of the acting category $\mathcal{M}$). But if $\mathbf{Para}_\bullet(\mathcal{C})$ is a *monoidal bicategory*, strictness condition is not sufficient for turning it into a *monoidal 2-category*. We additionally need commutativity.

Proposition 3.40. *Let* $(\mathcal{C}, \bullet)$ *be a strict monoidal* $\mathcal{M}$*-actegory. Then* $\mathbf{Para}_\bullet(\mathcal{C})$ *is a monoidal 2-category if and only if* $\mathcal{M}$ *is commutative monoidal.*

Proof. Appendix C. □

In the previous section, assuming $\mathcal{M}$ is strict monoidal was a mild assumption. Many such categories are found in nature, and moreover, every monoidal category is monoidally equivalent to a strict one. On the other hand, assuming a category is *commutative* monoidal is extremely prohibitive. Commutative monoidal categories are seldom found in nature, and they are not monoidally equivalent to strict monoidal ones. Assuming only strictness of $\mathcal{M}$, can we somehow simplify

$\mathbf{Para}_\bullet(\mathcal{C})$ and its monoidal structure? In the following box we describe how to apply a quotient (as we did in Box 3.1.2), and obtain a monoidal *category*.

From monoidal bicategories to monoidal categories **Box 3.2.2**

In Box 3.1.2 we described how change of base functors $F : \mathbf{Cat} \to \mathbf{Set}$ allow us to obtain a category $F_*(\mathbf{Para}_\bullet(\mathcal{C}))$ from a 2-category $\mathbf{Para}_\bullet(\mathcal{C})$. If $\mathbf{Para}_\bullet(\mathcal{C})$ additionally has monoidal structure as a bicategory, which of these functors transfer this monoidal structure to $F_*(\mathbf{Para}_\bullet(\mathcal{C}))$?

We provide an answer for functors in Box 3.1.2. Its proof can be found in Appendix C.

Theorem 3.41. *Let $(\mathcal{C}, \bullet)$ be a monoidal $\mathcal{M}$-actegory. Then the following three categories are monoidal.*

$$\pi_{0*}(\mathbf{Para}_\bullet(\mathcal{C})) \quad , \quad \mathsf{Iso}_*(\mathbf{Para}_\bullet(\mathcal{C})) \quad \textit{and} \quad \mathsf{Epi}_*(\mathbf{Para}_\bullet(\mathcal{C}))$$

On the other hand, $\mathsf{Ob}_*(\mathbf{Para}_\bullet(\mathcal{C}))$ is not a monoidal category.

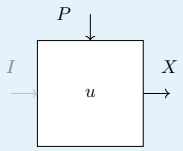
If in addition to $\mathcal{M}$ the category $\mathcal{C}$ is braided (resp. symmetric), then the three monoidal categories become braided monoidal (resp. symmetric). For example, the symmetric monoidal category **Para** as defined in [FST21, Def. III.1] is the symmetric monoidal category $\mathsf{Iso}_*(\mathbf{Para}_\times(\mathbf{Smooth}))$ in this thesis.

Lastly, as we have equipped $\mathbf{Para}(\mathcal{C})$ with a monoidal structure we can now reason about states, costates and scalars in this setting.

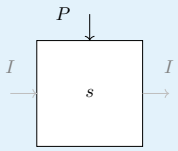
States, costates, and scalars in Para($\mathcal{C}$) **Box 3.2.3**

For a self-action of a monoidal category $\mathcal{C}$ the states, scalars, and costates of **Para**($\mathcal{C}$) have an interesting form.

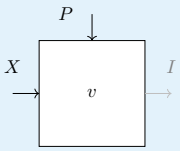
States



Scalars



Costates



$$\textbf{Para}(\mathcal{C})(I,X) \cong (\mathcal{C}/X)^{\mathsf{op}} \qquad \textbf{Para}(\mathcal{C})(I,I) \cong \sum_{P:\mathcal{C}} \mathcal{C}(P,I) \qquad \textbf{Para}(\mathcal{C})(X,I) \cong \sum_{P:\mathcal{C}} \mathcal{C}(X \otimes P, I)$$

Here we see that states of **Para**($\mathcal{C}$) correspond to morphisms in $\mathcal{C}$, and scalars correspond to costates of $\mathcal{C}$. This is going to be especially relevant when $\mathcal{C}$ is a more complex category, such as the category of optics (Chapter 5). Just like for monoidal categories, if the category is cartesian, then scalars and costates trivialise.

3.2.2 Braided, symmetric monoidal actegories?

Monoidal categories can additionally have braided structure which might or might not satisfy the property of symmetry (Section 2.2.1). The same story is true for monoidal *actegories*. We do not explicitly restate their definitions here, and instead point them out in the literature: braided monoidal actegories are defined in [CG23, Def. 5.4.1], and symmetric monoidal actegories in [CG23, Remark 5.4.3]. In both cases they require the acting category $\mathcal{M}$ to be symmetric.

These actegories are important because they provide an avenue for equipping **Para**$_\bullet(\mathcal{C})$ with braided and symmetric monoidal structure. As not just proving, but merely defining braided and symmetric monoidal categories is an extremely laborious task (see [JYJY21, Sec. 12.1]), in this subsection we only conjecture the following.

Conjecture 3.42 (Braiding and symmetry of **Para**$_\bullet(\mathcal{C})$)**.** *If $(\mathcal{C},\bullet)$ is a braided (resp. symmetric) monoidal actegory, then* **Para**$_\bullet(\mathcal{C})$ *is a braided (resp. symmetric) monoidal bicategory.*[8]

[8] Following the periodic table of elements higher categories ([BS10, Sec. 2.1]), we conjecture that a braided

3.3 Actegories and Para in the literature

Actegories are not a new construction. They've appeared throughout the literature in many forms (see [Fuj19, Def. 5.1] and [JK01] for two examples), and we refer the reader to their only existing survey [CG23] for more information. The **Para** construction is also not new. Its earliest appearance known to us is in [HT12, Sec. 2.2] where it appeared under the name of a *monoidal category with indeterminates*, as a 1-category.[9] Under the name **Para** it was originally introduced in [FST21], albeit in a slightly different form. We leave a nuanced analysis to Section 8.3).

We review the work related to actegories and **Para** through the concept of graded monads, as that gives an enlightening view of **Para**.

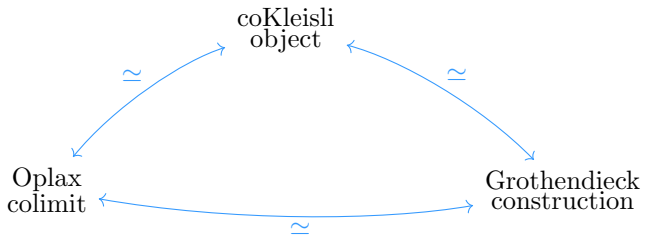
Definition 3.43 (Graded monad, compare [Fuj19, Def. 2.1])**.** Let $(\mathcal{M}, \otimes, I)$ be a strict monoidal category. Let $\mathcal{C}$ be a category. A $\mathcal{M}$-graded monad on $\mathcal{C}$ is a lax monoidal functor

$$(T, \eta, \mu) : (\mathcal{M}, \otimes, I) \to (\underline{\mathbf{Cat}}(\mathcal{C}, \mathcal{C}), \circ, 1_{\mathcal{C}})$$

Unpacking the definition, it can easily be seen that a graded monad (resp. comonad) is similar to a strict actegory, except the natural isomorphisms η and μ are weakened into mere natural transformations. Such actegories are in [Fuj19, Def. 5.2] called *lax* (resp. *oplax*) actegories, and in fact, they're equivalent to graded (co)monads (see [Fuj19, Sec. 5.1]). This correspondence allows us to then restate various kinds of results for graded (co)monads to those of (op)lax actegories. For instance, it allows us to see ordinary (**1**-graded) (co)monads as (op)lax **1**-actegories. It also allows us to study the analogue of the (co)Kleisli construction for (co)monads. As it turns out — the coKleisli construction of a graded comonad is precisely **Para**! Following the intuitions about equivalences between coKleisli objects, oplax colimits and the Grothendieck construction, this allows us to restate **Para** as a kind of a bicategorical Grothendieck construction ([Bak09]).

monoidal actegory doesn't make **Para** merely braided, but additionally *sylleptic* monoidal.

[9] Section 2.1 of the aforementioned paper also remarks about the difference between local and global contexts (Section 3.1.2) in the language of terms and indeterminates.



(3.44)

Proposition 3.45. *Let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$-actegory. Then the bicategorical Grothendieck construction of the composite pseudofunctor*

$$\mathbf{B}\mathcal{M} \xrightarrow{\mathbf{B}\bullet} \mathbf{B}\underline{\mathbf{Cat}}(\mathcal{C},\mathcal{C}) \hookrightarrow \mathbf{Cat}$$

is equivalent to the bicategory $\mathbf{Para}_\bullet(\mathcal{C})$, where $\mathbf{B}$ denotes the delooping of monoidal categories and monoidal functors.

This is especially relevant in light of ([OWEI20]) which serves as a good reference for these constructions, outlining a number of further appearances in theory and practice.

Another generalisation of actegories are *dependent actegories* [Mye22b], a topic of ongoing research. It stems from the observation that the domain of the actegory functor $\bullet : \mathcal{C} \times \mathcal{M} \to \mathcal{C}$ is a product of categories, i.e. a special version of the Grothendieck construction for a constant functor $\mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$ at $\mathcal{M}$. Dependent actegories allow an arbitrary functor of this type here, and model the notion that the type of a parameter can depend on the input type.

We do not study this in detail, but merely give some more intuition in the case of $\mathbf{Set}$. In the non-dependent case the domain of a parametric morphism $A \to B$ for the self-action of the cartesian product on $\mathbf{Set}$ is a product i.e. the set $A \times P$. This is the constant version of the *dependent pair type*. If we want to make this dependent, we can replace the constant self-action of $\mathbf{Set}$ by the (non-constant) representable functor $\underline{\mathbf{Cat}}(-, \mathbf{Set}) : \mathbf{Set}^{\mathsf{op}} \to \mathbf{Cat}$ mapping A to the functor category $\underline{\mathbf{Cat}}(A, \mathbf{Set})$.[10] Appropriately defining dependent $\mathbf{Para}$ here (see [Mye22b] for details) we obtain a bicategory where a morphism $A \to B$ consists of a indexed parameter function $P : A \to \mathbf{Set}$ and the implementation function $f : \sum_{a:A} P(a) \to B$. The former describes for each $a : A$ the set of possible parameters, and the latter consumes an $a : A$, a $p : P(a)$ and produces an element of B.

Yet another way to generalise $\mathbf{Para}$ stems from the observation that bicategories are a special kind of double categories ([JYJY21, Sec. 12.3]) with only *loose* 1-morphisms. That is, the only

[10] We are ignoring universe levels, and omitting the inclusion $\mathbf{Set} \to \mathbf{Cat}$.

kinds of 1-cells here are the parametric ones. As it turns out, $\mathbf{Para}_\bullet(\mathcal{C})$ can be turned into a double category into a coherent way, where the non-parametric morphisms play the role of *tight* 1-cells. The constructions in [Mye22b] defining dependent **Para** automatically takes care of this too, producing a double category as just described.

Lastly, during the writing of this thesis I learned[11] that there is a promising generalisation of actegories to *locally graded categories* (Appendix B): categories enriched in $\underline{\mathbf{Cat}}(\mathcal{M}^{\mathrm{op}}, \mathbf{Set})$. Locally graded categories unify parameterisation that is *internal* (the one defined in this chapter), allowing us to see actegories as locally graded categories with copowers by representables (see Prop. B.9), but also parameterisation that is *external* — something we will explore in Section 4.4.4. Locally graded categories also have a coherent interpretation with respect to global parameterisation (Section 3.1.2), explored in (Appendix B). See Appendix B for some preliminary work exploring the relation of locally graded categores to parameterisation.

[11] From Dylan Braithwaite.

Chapter 4

Bidirectionality

> You're stealing all the
> terminology from physics.
>
> Ieva Čepaitė

What is there in common between neural networks, bayesian learners, reinforcement learners, and game-theoretic agents?

At their core, all of these systems are bidirectional. They involve two separate processes, happening sequentially one after the other, connected by a shared internal state. Neural networks propagate values forward, and valuations on gradients backward. Bayesian learners propagate evidence forward, and belief updates backward. In both value iteration and game theory we propagate values forward, while we compute policy improvements and payoffs, respectively, going backward.

Each of these has its own distinct character, involving functions that are differentiable or probabilistic, for example. Each of these cases is an entire mathematical field in itself with its own notation, conventions, theorems and ideas. However, these fields are mostly pursued and developed independently, and there is no formal and unifying mathematical framework that would allow us to systematically translate ideas, techniques, and theorems from any one of these fields to the other. In this chapter we set out to provide such a framework.

Contributions. *Large fragments of this chapter are a novel contribution. Most notably, the definition of weighted optics (Definition 4.14), the isomorphism of their hom-objects to weighted colimits*

(Prop. 4.10), the definition of a cartesian actegory (Example 4.32), the definition of a cartesian optic (Definition 4.35), factorisation theorem for **coPara** *(Theorem 4.43). Lastly, novel contribution is the provision of formal desiderata for dependent optics, survey of existing work done on this topic (Section 4.5.1).*

Epistemic status. *I have found that the prevalent perspective on categorical models of bidirectionality in the literature is that of dependently-typed* denotational *models, such as those given by dependent lenses or tangent categories. This is why in this thesis — motivated by the discovery of the operational interpretation of optic composition — I instead focus on the categorical semantics which can model the* operational *aspects of bidirectionality, which I felt was an unjustifiably underexplored area. This direction took me to 2-category theory, lax functors, emergent effects, and many more interesting concepts, but also, at the moment, unfortunately a relatively complex theory. As such I believe many constructions in this chapter will eventually be subsumed by advances in locally graded categories and dependent optics, leading to a cleaner formulation. Nonetheless, many constructions — such as weighted* **coPara***, and new variants of weighted mixed optics — are those I find to be useful abstractions nonetheless, deserving of further exploration.*

Remark 4.1 (A note on terminology.)**.** Originating in functional programming, bidirectional processes of a particular kind emerged under the name of *lenses*. Later on these were generalised to other settings where they were called *prisms*, *grates*, and *glasses*, for instance. A unifying framework was thereafter proposed unifying them called under the name of *optics* and later generalised to *mixed* optics). Despite the fact that this terminology that has no semblance whatsoever to types of cut glass, we continue using it.

4.1 High-level intuition

Following the intuition of describing processes using monoidal categories from Section 2.2, let's first describe informally what we mean by a process being *bidirectional*. Fundamentally it means that we have structured information about the underlying category it is built upon. If the domain of a process (i.e. an object X of some category $\mathcal{C}$) specifies the type of inputs consumed by said process, then in the bidirectional setting each object specifies a type of both an input and an output consumed by a bidirectional process — an input going forward, and an output going backward. We will write these objects as $\binom{X}{X'}$ where the top symbol denotes the type of inputs consumed

by the forward part of the bidirectional process, and the bottom symbol X' denotes the type of outputs produced by its backward part. Sometimes we will have to refer to these types outside of the stacked representation $\binom{X}{X'}$ in which case we will always rely on the fact that forward parts do not come with a superscript $'$, but backward parts do. We will draw such processes as in Fig. 4.1.[1]

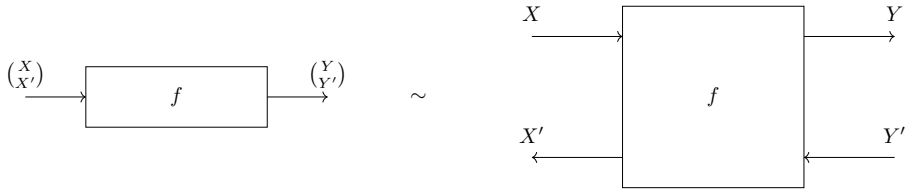


Figure 4.1: A bidirectional process drawn as a string diagram.

This describes the objects. What about the morphisms? Here the situation is more complex. Looking at the above box of type $\binom{X}{X'} \to \binom{Y}{Y'}$, in many ways we want to abstract away what happens *inside* of it, and instead provide a general formulation of these processes by only describing what they do from the outside. One thing we might want to do is say that given any map of type $Y \to Y'$ (which we think of as the environment producing a Y and responding with Y'), the bidirectional process $\binom{X}{X'} \to \binom{Y}{Y'}$ needs to turn that into a map of type $X \to X'$. This will prove useful later in the Section 4.3.

In other ways, we might want to be more concrete. What are the types of these forward and backward processes? A naive approach might attempt to mirror the structure of objects, and take bidirectional processes to be morphisms in $\mathcal{C} \times \mathcal{C}^{\mathsf{op}}$, giving their forward and backward maps types $X \to Y$ and $Y' \to X'$, respectively. But the problem with this is that the backward part needs to be able to depend on the forward one. For instance, when we perform backpropagation, the type of the gradient we send backwards depends on the original input we received.

We could simply add this as an argument. That is, a bidirectional process could be considered to have types $X \to Y$ on the forward part and $X \times Y' \to X'$ on the backward one. This is a construction that is often called *a lens*, and we will thoroughly unpack it in Section 4.4.1. But this has problems as well. Specifying the backward type as $X \times Y' \to X'$ prevents us from a) modelling cases where, say, depending on the input X we conditionally might choose not to interact with the environment at all (see "prisms" in Section 4.4.2), and b) reusing (in the backward pass) useful

[1] We will see later that this is formal graphical notation for bidirectional processes called *optics* ([Boi20]).

values of a type different than X computed in the forward pass (see Section 6.3.1).

But we can alleviate these problems as well. We can allow the bidirectional process to choose the type of information it communicates from the forward to the backward pass. This gives it something like an existential quantifier in the type, and at the same time allows us to abstract away from the categorical product $\times$ into a more general monoidal one:

$$\exists M.(X \to M \otimes Y) \times (M \otimes Y' \to X')$$

Roughly, this is a construction that is called *an optic* (Fig. 4.2), and its particular generalisation will be the topic of this chapter.

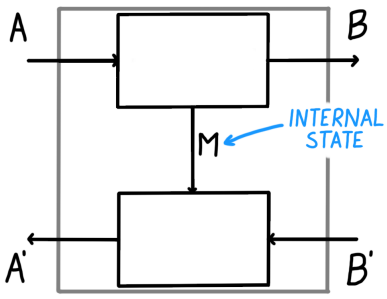


Figure 4.2: Schematic of a bidirectional process with an internal state.

But this story has problems as well. What if the type of the backward pass is different than the one of the forward one? In backpropagation, for instance, while the forward pass can in principle be any differentiable map, its reverse derivative at some point $x : X$ will always be a *linear map*. The above framework does not allow us to accurately capture these types, as the forward and the backward maps live in the same category.

But this can be solved as well. Using actegories defined in the previous chapter (Definition 3.2), we can separate out the type of the forward category $\mathcal{C}$, the backward category $\mathcal{D}$ and the monoidal category $\mathcal{M}$ that serves as the type of intermediate data. Its type (now more precisely specified) is

$$\exists (M : \mathcal{M}).\mathcal{C}(X, M \bullet Y) \times \mathcal{D}(M \blacksquare Y', X')$$

These are called *mixed optics* (see Box 4.3.1), and represent the penultimate step of our abstraction. There is one last problem. This particular construction has the same monoidal category $\mathcal{M}$ acting on both the forward and the backward part. In many of the cases of interest, this is not the case. In backpropagation, for instance, while the action of the forward part uses the categorical product of a suitable category of differentiable maps (such as $(\mathbf{Smooth}, \times, 1)$), this categorical product is not something we want for the backwards category of linear maps. In $\mathbf{FVect}_{\mathbb{R}}$ it would necessitate that the reverse derivative is linear with respect to the product of the type of the point X we're differentiating at, and the type of the gradient Y'. But this is not true, because the derivative is linear only with respect to the latter. The monoidal product we want here is the *tensor product*, which is not a categorical product.

But this kind of a complex gadget has not appeared in the literature so far. It models an intricate interaction between the forward and backward categories, as well as the forward and backward *monoidal categories* of intermediate information. As a matter of fact, we will see that the data needed to form these kind of optics arises as an *action* of, roughly, the product of monoidal categories of intermediate information *on* the product of forward and backward categories. We will also see that the category of these optics — that we will name *weighted* optics — arises as the dual of the aforementioned **Para** construction on this actegory. We proceed in making this intuition precise and mathematically formal — leading us to the definition of the construction **coPara**.

4.2 The coPara construction

The **coPara** construction arises as a systematic dualisation of the bicategory **Para**. Because of its central role in describing bidirectional processes, we take special care in unpacking its details.

Definition 4.2 (Coparametric maps (compare [CGHR22, Rem. 4]))**.** Let $(\mathcal{M}, \otimes, I)$ be a monoidal category, and $(\mathcal{C}, \bullet)$ a left $\mathcal{M}$-actegory. We define the bicategory $\mathbf{coPara}_{\bullet}(\mathcal{C})$ with the following data:

- **Objects** are those of $\mathcal{C}$;
- **Morphisms** are often referred to as *coparametric morphisms* (Fig. 4.3). A coparametric morphism $A \to B$ is a pair (P, f) where $P : \mathcal{M}$ and $f : A \to P \bullet B$ is a morphism in $\mathcal{C}$. As parametric morphisms, coparametric ones also have a horizontal and a vertical component (Fig. 4.3).

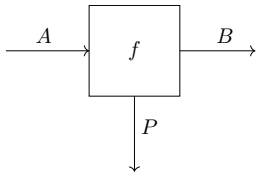
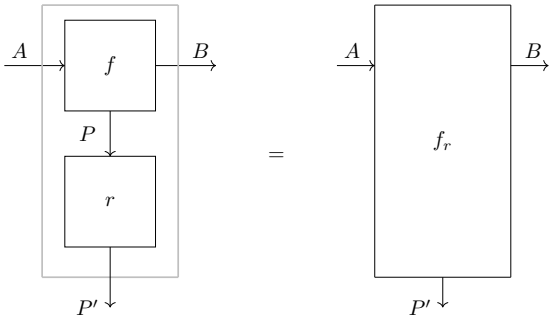


Figure 4.3: String diagram representation of a coparametric morphism. As with parametric maps (Fig. 3.1), here we also draw the coparameter wire vertically, but *below*.

- **2-morphisms** are called *reparameterisations*. A 2-morphism from $(P, f) \Rightarrow (P', f')$ is a morphism $r : P \to P'$ in $\mathcal{M}$ such that the following diagram commutes in $\mathcal{C}$. As with parametric morphisms, we will often write f_r for the reparameterisation of f with r, where r now appears as a subscript instead of a superscript.



$$\begin{array}{ccc} A & \xrightarrow{f} & P \bullet B \\ & \searrow^{f'} & \downarrow{\scriptstyle r \bullet B} \\ & & P' \bullet B \end{array} \tag{4.3}$$

Figure 4.4: String diagram of reparameterisation. As with Fig. 3.2, we exploit the second dimension. Note that here 2-cells point in a different direction than with **Para**!

- **Identity morphism.** For every object A there is an identity coparametric morphism (I, η_A) where $I : \mathcal{M}$ and $\eta_A : I \bullet A \to A$ is the unitor of the underlying actegory;

- **Morphism composition.** The composition of coparametric morphisms

$$A \xrightarrow{(P,f)} B \xrightarrow{(Q,g)} C$$

i.e. of

$$\begin{array}{ll} P : \mathcal{M} & \\ A \xrightarrow{f} P \bullet B & \end{array} \quad \text{and} \quad \begin{array}{ll} Q : \mathcal{M} & \\ B \xrightarrow{g} Q \bullet C & \end{array}$$

is given by $(P \otimes Q, f \fatsemi_{\mathfrak{p}} g)$ where

$$P \otimes Q : \mathcal{M}$$

$$f \fatsemi_{\mathfrak{p}} g \coloneqq \boxed{A \xrightarrow{f} P \bullet B \xrightarrow{P \bullet g} P \bullet (Q \bullet C) \xrightarrow{\mu^{-1}_{P,Q,C}} (P \otimes Q) \bullet C} \qquad \text{in} \quad \mathcal{C}$$

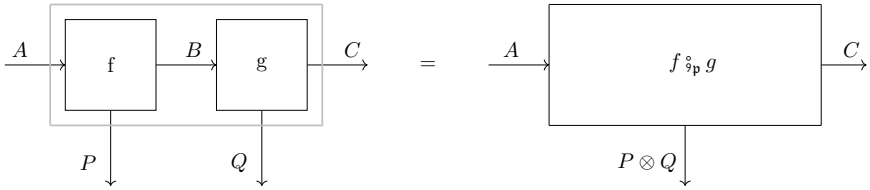


Figure 4.5: String diagram representation of the composition of coparametric morphisms, following the same organisational principles as Fig. 3.3.

This concludes the definition of the bicategory $\mathbf{coPara}_{\bullet}(\mathcal{C})$.

Unlike with **Para**, here we require $(\mathcal{C}, \bullet)$ to be a *left* actegory, inspired by the idea that, despite the equivalence of left and right actegories when $\mathcal{M}$ is braided (Lemma 3.6) (which is the assumption peremeating this thesis) it is good *mental hygiene* to distinguish between the two (Remark 3.7). In the same manner that left and right actegories become equivalent under braiding, so do **Para** and **coPara** constructions thereof.

Proposition 4.4 (Relation between **Para** and **coPara**)**.** *Let $(\mathcal{C}, \bullet)$ be a left $\mathcal{M}$-actegory where $\mathcal{M}$ is braided. Then*

$$\mathbf{Para}_{\bullet}(\mathcal{C}^{\mathsf{op}}) \cong \mathbf{coPara}_{\bullet}(\mathcal{C})^{\mathsf{coop}} \quad \textit{and} \quad \mathbf{coPara}_{\bullet}(\mathcal{C}^{\mathsf{op}}) \cong \mathbf{Para}_{\bullet}(\mathcal{C})^{\mathsf{coop}}$$

where we're overloading $\bullet$ to also include the action from the right given by (Lemma 3.6).

Proof. We prove both claims side by side.

$$\begin{aligned}
&\mathbf{Para}_\bullet(\mathcal{C}^{\mathsf{op}})(A,B) \\
&= \sum_{P:\mathcal{M}} \mathcal{C}^{\mathsf{op}}(A \bullet P, B) \\
&= \sum_{P:\mathcal{M}} \mathcal{C}(B, A \bullet P) \\
&= \mathbf{coPara}_\bullet(\mathcal{C})^{\mathsf{co}}(B,A) \\
&= \mathbf{coPara}_\bullet(\mathcal{C})^{\mathsf{coop}}(A,B)
\end{aligned}
\qquad
\begin{aligned}
&\mathbf{coPara}_\bullet(\mathcal{C}^{\mathsf{op}})(A,B) \\
&= \sum_{M:\mathcal{M}} \mathcal{C}^{\mathsf{op}}(A, M \bullet B) \\
&= \sum_{P:\mathcal{M}} \mathcal{C}(M \bullet B, A) \\
&= \mathbf{Para}_\bullet(\mathcal{C})^{\mathsf{co}}(B,A) \\
&= \mathbf{Para}_\bullet(\mathcal{C})^{\mathsf{coop}}(A,B)
\end{aligned}$$

The reason $^{\mathsf{co}}$ appears is because 2-cells in **Para** and **coPara** point in different directions. Where braiding comes in play is in morphism composition – when we need to change the order in which the composite $M \otimes N$ acts on an object of $\mathcal{C}$. □

The above proposition allows us to translate all the theorems and propositions about **Para** to theorems about **coPara**. This includes (Prop. 3.23) which, translated to **coPara** in Prop. 4.5, tells us that locally the hom-category of **coPara** can also be computed as a category of elements, albeit now of a functor whose domain is $\mathcal{M}$ and not $\mathcal{M}^{\mathsf{op}}$.

Proposition 4.5. *Let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$-actegory. Let $A, B : \mathcal{C}$. Then*

$$\mathbf{coPara}_\bullet(\mathcal{C})(A,B) = \mathbf{El}(\mathcal{C}(A, B \bullet -)) = \sum_{P:\mathcal{M}} \mathcal{C}(A, B \bullet P)$$

This means that the trivialisation theorem (Theorem 3.28) now holds for the case of $\mathcal{M}$ with a monoidal unit which is *terminal*, instead of initial.[2] This yields the following theorem, and corollary.

Theorem 4.6. *Let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$-actegory where the monoidal unit of $\mathcal{M}$ is terminal. Then*

$$\pi_{0*}(\mathbf{coPara}_\bullet(\mathcal{C})) \cong \mathcal{C}$$

Corollary 4.7. *Let $\mathcal{C}$ be a cartesian category. Then $\pi_{0*}(\mathbf{coPara}(\mathcal{C})) \cong \mathcal{C}$.*

As we will see in Remark 4.33, $\mathbf{coPara}(\mathcal{C})$ will not trivialise for a generalisation of a cartesian category to the setting of actegories.

[2] Intuitively, notice that any coparametric map $f : A \to M \bullet B$ can be reparameterised with $!_M : M \to 1$ yielding $f \fatsemi (!_M \bullet B)$ which under connected components becomes equivalent to f.

4.2.1 Forget, then act: a piece of notation

We are almost ready to define the category of bidirectional processes. We just need to introduce one important piece of notation. This notation will capture the idea that we are mostly interested in the **coPara** construction on actegories of a particular form.[3]

Consider a $\mathcal{E}$-actegory $(\mathcal{C}, \bullet)$ for some monoidal category $\mathcal{E}$. Think about the role of an object E in a coparametric morphism $(E : \mathcal{E}, f : X \to E \bullet Y)$. Even though it appears in the codomain of f, it really is a *part of the data of the coparametric morphism*. And it is a part of the data which determines *the type* of the implementation map f. This determination can happen in many different ways, and the type of implementation maps may end up being the same for different coparameters E and E'. For instance, this is the case when the actions of E and E' on B are equal (i.e. $E \bullet Y = E' \bullet Y$), in which case we have forgotten something about E and E' in the action. This suggests that often, our action should really be of the form $\pi(E) \bullet Y$, where $\pi : \mathcal{E} \to \mathcal{B}$ is some kind of a forgetful functor, and $\bullet$ is of type $\mathcal{B} \times \mathcal{C} \to \mathcal{C}$. When is this the case?

This happens under two conditions. The first one is that the category $\mathcal{E}$ arises as the category of elements $\mathbf{El}(W)$ (Definition D.1) of some functor $W : \mathcal{B}^{\mathsf{op}} \to \mathbf{Set}$, where $\mathbf{El}(W)$ is how we will refer to $\mathcal{E}$ from now on. This substantiates the idea that an object $E : \mathbf{El}(W)$ consists of two parts — a part $B : \mathcal{B}$ which acts on an object of $\mathcal{C}$, and a part $B_W : W(B)$ which only serves as an inert piece of additional data. In this case we have a canonical functor $\mathbf{El}(W) \xrightarrow{\pi_W} \mathcal{B}^{\mathsf{op}}$ which forgets these additional pieces of data[4], allowing us to form the functor $\bullet^{\pi_W^{\mathsf{op}}}$ below (where we're using the notation for reparameterisation of actegories from Definition 3.10).

$$\bullet^{\pi_W^{\mathsf{op}}} = \boxed{\mathbf{El}(W)^{\mathsf{op}} \times \mathcal{C} \xrightarrow{\pi_W^{\mathsf{op}} \times \mathcal{C}} \mathcal{B} \times \mathcal{C} \xrightarrow{\bullet} \mathcal{C}} \tag{4.8}$$

This brings us to the second condition. For this to be a valid reparameterisation of actegories (Definition 3.10), π_W^{op} needs to be a strong monoidal functor. And that indeed happens if W is lax monoidal (Prop. D.4).

With these two conditions we get a a $\mathbf{El}(W)^{\mathsf{op}}$-actegory $\mathcal{C}$ which explicitly acknowledges these two components in the acting category. More specifically, by instantiating $\mathbf{coPara}_{\bullet^{\pi_W^{\mathsf{op}}}}(\mathcal{C})$ we can see that a coparametric morphism $X \to Y$ now consists of a parameter object (B, B_W) (where $B : \mathcal{B}$ and $B_W : W(B)$), and an implementation map $f : X \to B \bullet X$ which is independent of the

[3] The entire story holds for **Para** as well, but is less relevant for us.

[4] This functor is a *fibration*, a concept we mention for completeness but do not touch upon in this thesis.

choice of B_W. Additionally, we will often be interested in the coparametric *category*, and one that arises out of the connected components quotient. This is why we introduce the following piece of notation.

$$\mathbf{coPara}^W_\bullet(\mathcal{C}) \coloneqq \pi_{0*}(\mathbf{coPara}_{\bullet^{\pi_W^{\mathsf{op}}}}(\mathcal{C})) \tag{4.9}$$

This piece of notation will be central in our definition of optics. It describes a *weighted* coparametric category of an action $\bullet$ and weight W, where in the presence of the superscript W on **coPara** the subscript action $\bullet$ is meant to be reparameterised by π_W. The reason we use the adjective "weighted" is because, analogously to Prop. 3.23 which tells us that the connected components quotient locally computes a colimit of the functor $\mathcal{C}(X, - \bullet Y)$, here we will see that the same quotient ends up computing a colimit of $\mathcal{C}(X, - \bullet Y)$ *weighted* by W (Prop. 4.10).

Proposition 4.10. *Let* $X, Y : \mathbf{coPara}^W_\bullet(\mathcal{C})$. *Then*

$$\mathbf{coPara}^W_\bullet(\mathcal{C})(X,Y) \cong \mathsf{colim}^W \mathcal{C}(X, - \bullet Y)$$

Proof. The proof below relies crucially on identification the connected components of the category of the elements with the colimit, and identification of **Set**-colimits of a particular form with weighted colimits (Prop. E.7).

$$\begin{aligned}
&\mathbf{coPara}^W_\bullet(\mathcal{C})(X,Y) \\
(\text{Def.}) \quad &= \pi_0(\mathbf{coPara}_{\bullet^{\pi_W}}(\mathcal{C})(X,Y)) \\
(\text{Prop. 4.5}) &\cong \pi_{0*}(\mathbf{El}(\mathbf{El}(W)^{\mathsf{op}} \xrightarrow{\pi_W^{\mathsf{op}}} \mathcal{B} \xrightarrow{\mathcal{C}(X,-\bullet Y)} \mathbf{Set}) \\
(\text{Prop. E.6}) &\cong \mathsf{colim}(\mathbf{El}(W)^{\mathsf{op}} \xrightarrow{\pi_W^{\mathsf{op}}} \mathcal{B} \xrightarrow{\mathcal{C}(X,-\bullet Y)} \mathbf{Set}) \\
(\text{Prop. E.7}) &\cong \mathsf{colim}^W \mathcal{C}(X, - \bullet Y)
\end{aligned}$$

□

Remark 4.11. We've defined $\mathbf{coPara}^W_\bullet$ using the projection from the category of elements of W. This was only possible because the weight is a functor into **Set**, i.e. because we are dealing with **Set**-enriched categories. The fact that in **Set** this construction ends up being isomorphic to a colimit weighted by W (Prop. 4.10) suggests an avenue for generalisation to a setting enriched in an arbitrary monoidal category $\mathcal{V}$: by taking the aforementioned proposition as a definition.

4.3 Weighted optics

We are now in position to understand one of the central definitions of this thesis: weighted optics. The first step is to fix a "forward" left $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$ and a "backward" left $\mathcal{N}$-actegory $(\mathcal{D}, \blacksquare)$, and form their product.

Proposition 4.12 (Product of actegories, [CG23, Prop. 4.2.1.])**.** *Fix the following.*

$$\text{A } \mathcal{M}\text{-actegory } (\mathcal{C}, \bullet)$$
$$\text{A } \mathcal{M}'\text{-actegory } (\mathcal{D}, \blacksquare)$$

Their cartesian product is a $\mathcal{M} \times \mathcal{M}'$*-actegory* $(\mathcal{C} \times \mathcal{D}, \left(\begin{smallmatrix}\bullet\\ \blacksquare\end{smallmatrix}\right))$ *where the action is defined as*

$$\left(\begin{smallmatrix}\bullet\\ \blacksquare\end{smallmatrix}\right) := \boxed{\mathcal{M} \times \mathcal{M}' \times \mathcal{C} \times \mathcal{D} \xrightarrow{\mathcal{M} \times \mathsf{swap} \times \mathcal{D}} \mathcal{M} \times \mathcal{C} \times \mathcal{M}' \times \mathcal{D} \xrightarrow{\bullet \times \blacksquare} \mathcal{C} \times \mathcal{D}} \tag{4.13}$$

A sensible attempt at defining these bidirectional processes is to compute the **coPara** construction of this product actegory.[5] That is, we can dualise the second actegory first, and then take the product — forming a $\mathcal{M} \times \mathcal{M}'^{\mathsf{op}}$-actegory $(\mathcal{C} \times \mathcal{D}^{\mathsf{op}}, \left(\begin{smallmatrix}\bullet\\ \blacksquare^{\mathsf{op}}\end{smallmatrix}\right))$. However, simply taking **coPara** of this actegory will not suffice. Let's see why that is the case.

A coparametric morphism in $\mathbf{coPara}_{\left(\begin{smallmatrix}\bullet\\ \blacksquare^{\mathsf{op}}\end{smallmatrix}\right)}(\mathcal{C} \times \mathcal{D}^{\mathsf{op}})$ consists of two choices of coparameters $M : \mathcal{M}$ and $M' : \mathcal{M}'$ and two maps $f : X \to M \bullet Y$ and $f' : Y' \blacksquare M' \to X'$, depicted in Fig. 4.6.

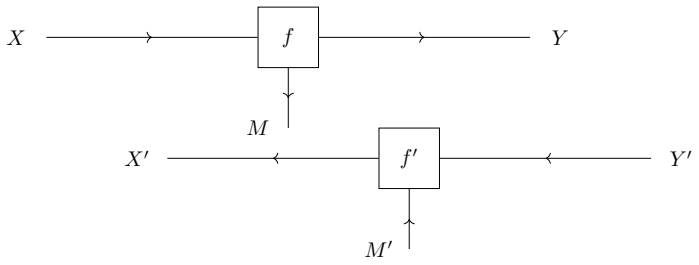


Figure 4.6: Graphic of this coparametric category showing its missing internal state.

But this is problematic — as there is nothing connecting M and M'. That is, given any way of turning a Y into a Y', using f and f' we can't turn X into an X' because we are missing this piece of connecting tissue. This extra piece of data, we can also see, would not impact what the action $\left(\begin{smallmatrix}\bullet\\ \blacksquare\end{smallmatrix}\right)$ does — it would simply be a part of the data of the morphism that would enable us to make

[5] Recall that a $\mathcal{M}$-actegory $\bullet : \mathcal{M} \times \mathcal{D} \to \mathcal{D}$ gives rise to a $\mathcal{M}^{\mathsf{op}}$ actegory $\mathcal{D}^{\mathsf{op}}$ given by $\mathcal{M}^{\mathsf{op}} \times \mathcal{D}^{\mathsf{op}} \to \mathcal{D}^{\mathsf{op}}$.

sense of bidirectionality this way. How do we formally describe this? As hinted by Section 4.2.1: we can do that with weighted Copara.

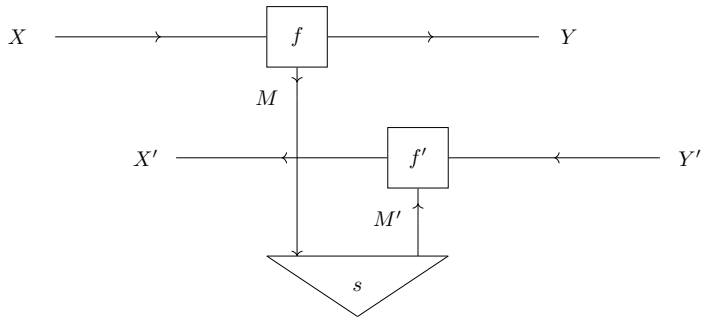


Figure 4.7: Graphic of the putative correct coparametric category (optic, not a category) augmented with the data of a "weight".

Since we need this for any possible residual, this can be included as an additional piece of data in the definition of a bidirectional process: a functor $W : \mathcal{M} \times \mathcal{M}' \to \mathbf{Set}$ which, given any two objects (M, M') gives us a set of "connecting tissues" between M and M', where s in Fig. 4.7 above is an element of this set. This results in a description of this bidirectional process which is "closed off" in the vertical dimension by W which we think of as the costate. As we will see below, in order for everything to be well-defined we will need W to be a lax monoidal functor with respect to the induced monoidal structure on the domain, and the product monoidal structure of **Set** on the codomain.

This brings us to the central definition of the paper.

Definition 4.14 (Category of weighted optics)**.** Fix the following data.

$$\text{A } \mathcal{M}\text{-actegory } (\mathcal{C}, \bullet)$$
$$\text{A } \mathcal{M}'\text{-actegory } (\mathcal{D}, \blacksquare)$$
$$\text{A lax monoidal functor } (W, \phi, \epsilon) : \mathcal{M}^{\mathsf{op}} \times \mathcal{M}' \to \mathbf{Set}$$

We define the **category of weighted optics** $\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}}$ as a weighted coparametric category

$$\mathbf{coPara}^{W}_{\binom{\bullet}{\blacksquare^{\mathsf{op}}}}(\mathcal{C} \times \mathcal{D}^{\mathsf{op}})$$

This construction packs a lot of punch, and we will now take the time to unpack its contents. An object in $\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}}$ is a pair $\binom{A}{A'}$, where $A : \mathcal{C}$ and $A' : \mathcal{D}^{\mathsf{op}}$. Following Prop. 4.10, the set

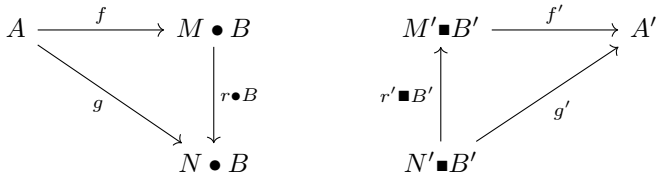
of morphisms $\binom{A}{A'} \to \binom{B}{B'}$ is given by the weighted colimit, where we'll often use the following explicit coend representation.

$$\begin{aligned}\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}}\left(\begin{matrix}A & B\\ A' & B'\end{matrix}\right) =& \mathsf{colim}^{W}_{\binom{M}{M'}}\mathcal{C}(A, M \bullet B) \times \mathcal{D}(M' \blacksquare B', A') \\ \cong& (\text{Weighted colimit as coend (Prop. E.9)}) \\ & \int^{M:\mathcal{M}, M':\mathcal{M}'} \mathcal{C}(A, M \bullet B) \times W(M, M') \times \mathcal{D}(M' \blacksquare B', A')\end{aligned} \tag{4.15}$$

Weighted optics are elements of this set, and they consist of the following three pieces of data:

- A coparameter $\binom{M}{M'}$ consisting of a forward and a backward residual, where $M : \mathcal{M}$ and $M' : \mathcal{M}'$;
- A map $s : W(M, M')$ which we think of as the connecting tissue between the forward and the backward residuals;
- A coparametric map $\binom{f}{f'} : (\mathcal{C} \times \mathcal{D}^{\mathsf{op}})\left(\begin{matrix}A & M \bullet B\\ A' & M' \blacksquare B'\end{matrix}\right)$, i.e. a pair of maps $f : A \to M \bullet B$ in $\mathcal{C}$ and $f' : M' \blacksquare B' \to A'$ in $\mathcal{D}$ (Fig. 4.7).

This triple is quotiented by an equivalence relation which identifies it with any other $(\binom{N}{N'}, t, \binom{g}{g'})$ if there exists a pair of maps $r : M \to N$ in $\mathcal{M}$ and $r' : N' \to M'$ in $\mathcal{M}'$ such that $W(r, r')(t) = s$ and the following diagrams commute:



(4.16)

Identity and composition of weighted optics follow by those of **coPara**.

Identity. The identity optic $\binom{A}{A'} \to \binom{A}{A'}$ is the one whose coparameters are $\binom{I}{I'}$, the map connecting them is one given by the opunitor $\epsilon(\bullet) : W(I, I')$ of W on the unique element $\bullet : 1$, and implementation $\binom{\eta^{\bullet}_{A}}{\eta^{\blacksquare^{-1}}_{A'}}$ is given by the unitors of the two actegories (Fig. 4.8).

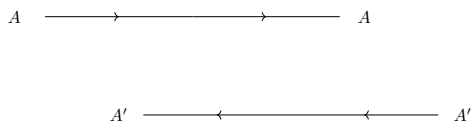


Figure 4.8: The identity weighted optic.

Composition. Composition of

$$\begin{pmatrix} A \\ A' \end{pmatrix} \xrightarrow{(\binom{M}{M'},s,\binom{f}{f'})} \begin{pmatrix} B \\ B' \end{pmatrix} \xrightarrow{(\binom{N}{N'},t,\binom{g}{g'})} \begin{pmatrix} C \\ C' \end{pmatrix}$$

i.e. of

$$\begin{pmatrix} M \\ M' \end{pmatrix} : \begin{pmatrix} \mathcal{M} \\ \mathcal{M}' \end{pmatrix} \qquad \begin{pmatrix} N \\ N' \end{pmatrix} : \begin{pmatrix} \mathcal{M} \\ \mathcal{M}' \end{pmatrix}$$

$$s : W(M, M') \qquad \text{and} \qquad t : W(N, N')$$

$$\begin{pmatrix} A \\ A' \end{pmatrix} \xrightarrow{\binom{f}{f'}} \begin{pmatrix} M \bullet B \\ M' \blacksquare B' \end{pmatrix} \qquad \begin{pmatrix} B \\ B' \end{pmatrix} \xrightarrow{\binom{g}{g'}} \begin{pmatrix} N \bullet C \\ N' \blacksquare C' \end{pmatrix}$$

is a triple $(\binom{M \otimes N}{M' \otimes N'}, \phi^W((s,t)), \binom{f \fatsemi_{\mathfrak{p}} g}{g' \fatsemi^{\mathfrak{p}} f'})$ where

$$\begin{pmatrix} M \otimes N \\ M' \otimes N' \end{pmatrix} : \begin{pmatrix} \mathcal{M} \\ \mathcal{M}' \end{pmatrix}$$

$$\phi^W((s,t)) : W(M \otimes N, M' \otimes' N')$$

and $f \fatsemi_{\mathfrak{p}} g : A \to (M \otimes N) \bullet C$ and $g' \fatsemi^{\mathfrak{p}} f' : (M' \otimes' N') \blacksquare C \to A$ are compositions of coparametric and parametric maps, respectively (Fig. 4.9).

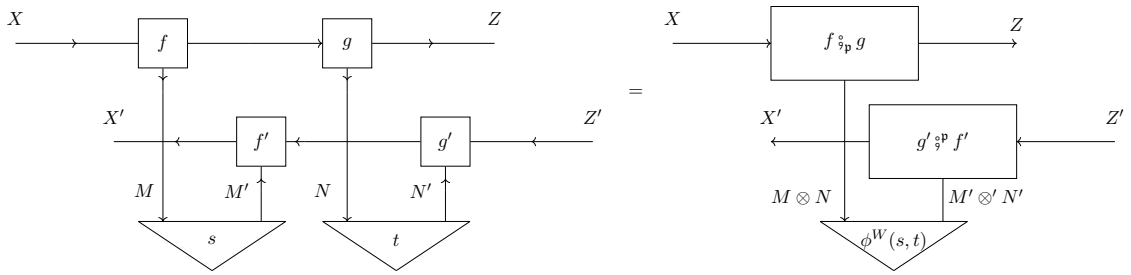


Figure 4.9: Composition of weighted optics.

Notation 4.17. Like we did with **Para** and **coPara**, we will simplify the notation for weighted op-

tics in special cases. We will omit the superscript W if the weight is a hom-functor (see Box 4.3.1).If optics are given by a self-action of some monoidal category $\mathcal{C}$ we will write **Optic**$(\mathcal{C})$ (see Examples 4.20 and 4.21), trusting that it is clear from the context what the underlying monoidal structure is. These reductions agree with the notation in [CEG+22, Ril18].

While this weighted colimit definition might seem daunting, there are a number of benefits of thinking about bidirectional processes this way. First of all, it's conceptual clarity. Weighted optics allow us to separate the formal structure of the acting categories (describing how residuals compose) from the formal structure of the forward and backward categories (describing the kinds of processes on the forward and the backward pass), from the intermediate process happening in between the forward and the backward pass. This conceptual clarity becomes useful in implementation of these systems on computers (Chapter 6), where the posession of mathematical tools to talk about their practical aspects such as the memory footprint of their composition improves our ability to implement them correctly and efficiently. Second, the reduction of the definition of weighted optics to that of the **coPara** construction provides us with general means of proving many theorems about optics (Sections 4.3.2, 4.4.1 and 4.4.4). Third, this definition suggests a generalisation to *dependent optics*, an important unification of dependent lenses and optics (Section 4.5.1).

We now proceed to reduce this abstract definition into many of its concrete forms. Many special cases of optics were only originally defined as these reductions, each of them with their own composition rule.

Proposition 4.18. *Consider an optic* $\mathbf{Optic}^{W}_{(\bullet,\blacksquare)}$ *with a weight represented by a strong monoidal functor* $i : \mathcal{M} \to \mathcal{M}'$ *(resp. corepresented by a strong monoidal functor* $j : \mathcal{M}' \to \mathcal{M}$*). Then there is the following isomorphism of categories:*

$$\mathbf{Optic}^{W}_{(\bullet,\blacksquare)} \cong \mathbf{Optic}_{(\bullet,\blacksquare^i)} \quad \textit{(resp.)} \quad \mathbf{Optic}^{W}_{(\bullet,\blacksquare)} \cong \mathbf{Optic}_{(\bullet^j,\blacksquare)}$$

Proof. Note that i (resp. j) needs to be strong in order for $\blacksquare^i$ (resp. $\bullet^j$) to be a well-defined reparameterisation of actegories. We prove only the left isomorphism by exhibiting an isomorphism of hom-sets and noting that naturality with respect to identities and composition follows by routine.

$$\mathbf{Optic}^{W}_{\left(\begin{smallmatrix}\bullet\\ \blacksquare\end{smallmatrix}\right)}\begin{pmatrix}A & B\\ A' & B'\end{pmatrix}$$

$$\text{(Def.)} \quad = \int^{M,M'} \mathcal{C}(A, M \bullet B) \times W(M, M') \times \mathcal{D}(M' \blacksquare B', A')$$

$$\text{(W is representable)} \quad \cong \int^{M,M'} \mathcal{C}(A, M \bullet B) \times \mathcal{M}'(i(M), M') \times \mathcal{D}(M' \blacksquare B', A')$$

$$\text{(coYoneda)} \quad \cong \int^{M} \mathcal{C}(A, M \bullet B) \times \mathcal{D}(i(M) \blacksquare B', A')$$

$$\text{(Def.)} \quad = \mathbf{Optic}_{\left(\begin{smallmatrix}\bullet\\ \blacksquare_i\end{smallmatrix}\right)}\begin{pmatrix}A & B\\ A' & B'\end{pmatrix}$$

□

This tells us the following.

Mixed optics are weighted optics **Box 4.3.1**

A variant of optics called *mixed optics* ([CEG+22]) is a special case of this definition. It arises when $\mathcal{M} = \mathcal{M}'$, and when the weight is given by the Hom-functor of $\mathcal{M}$.[a]

$$\mathbf{Optic}^{\mathsf{Hom}}_{\left(\begin{smallmatrix}\bullet\\ \blacksquare\end{smallmatrix}\right)}\begin{pmatrix}A & B\\ A' & B'\end{pmatrix} = \mathsf{colim}^{\mathsf{Hom}}_{\left(\begin{smallmatrix}M\\ M'\end{smallmatrix}\right)} \mathcal{C}(A, M \bullet B) \times \mathcal{D}(M' \blacksquare B', A')$$

$$\cong \text{(Colimit weighted by hom is coend)}$$

$$\int^{M} \mathcal{C}(A, M \bullet B) \times \mathcal{D}(M \blacksquare B', A')$$

This yields the string diagrams of optics as found in [CGHR22, Boi20].

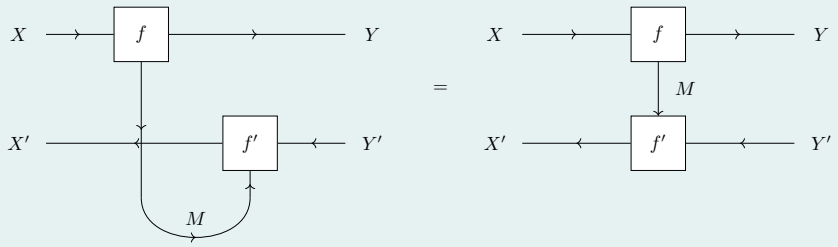


Figure 4.10: Optics weighted by the hom-functor can absorb the weight into into either the forward or the backward map.

[a] This functor needs to be lax monoidal in order to be a well-defined weight, but that is indeed always the case if $\mathcal{M}$ itself is monoidal. Intrestingly it is only lax monoidal, and not strong monoidal.

This gives us a straightforward way to see that there are classes of weighted optics not captured by mixed ones: they're the ones where the representing functor is merely lax monoidal instead of strong. We now proceed to study examples of weighted optics, and then give general classes of theorems allowing us to state when optics are monoidal, cartesian or closed.

4.3.1 Examples of weighted optics

We now aim to give the flavour of the kind of generality this definition possesses. They capture a disparate array of phenomena, each of a radically different nature: deterministic, conditional, and probabilistic, for example. The simplest kind of bidirectional systems are those whose forward and backward processes do not communicate.

Example 4.19 (Adapter, [CEG+22, Def. 3.38])**.** Consider the trivial action $\bullet$ on some category $\mathcal{C}$ (Example 3.13). Then a morphism $\binom{A}{A'} \to \binom{B}{B'}$ in the category $\mathbf{Optic}_{(\bullet)}$ is often called **an adapter** and it consists of two separate maps in $\mathcal{C}$ of type $A \to B$ and $B' \to A'$.

The category in the above example is isomorphic to the product category $\mathcal{C} \times \mathcal{C}^{\mathrm{op}}$, something which clearly is not the case for all optics. The two main and non-trivial examples of these are optics for a products (often called *lenses*) and optics for a coproducts (often called *prisms*).

Example 4.20 (Optics for products)**.** Consider any cartesian category $\mathcal{C}$. Then morphisms in the category $\mathbf{Optic}(\mathcal{C})$ for self-action of $\mathcal{C}$ (following the reduction from Box 4.3.1) of type $\binom{A}{A'} \to \binom{B}{B'}$ involve a choice of a residual $M : \mathcal{C}$, a forward map $A \to B \times M$ and a backward map $M \times B' \to A'$. We interpret this optic as a system which irrespective of the input A, always performs some internal computation (via M) *and* also interacts with the environment (via the query which produces a B and consumes a B').[6] Often called **lenses**; for full disambiguaton see Section 4.4.1.

Examples of this are the usual suspects of **Set**, **Smooth**, and so on. But we can also describe "conditional" bidirectional systems.

Example 4.21 (Optics for coproducts)**.** Consider any cocartesian category $\mathcal{C}$. A morphism in $\mathbf{Optic}(\mathcal{C})$ of type $\binom{A}{A'} \to \binom{B}{B'}$ involves a choice of a residual $M : \mathcal{C}$, a forward map $A \to B + M$ and a backward map $M + B' \to A'$. We interpret this optic as a system that consumes an input A and provides the response A' by *either* a) querying the environment (i.e. producing a B while

[6] For more detail on this operational perspective see [Gav22a].

saving no intermediate computation, and turning the B' received by the environment into an A') or b) not querying the environment, instead turning that A directly into some A' locally. Often called **prisms**; for full disambiguation see Section 4.4.2.

So far, the forward and the backward categories equal, and acting categories either trivial, or equal to forward/backward categories. We now study examples where that's not the case.

Example 4.22 (Markov kernels and expectations)**.** Recall the setting of Markov kernels and expectations from Example 3.12 where we defined an action

$$\otimes'^{\Delta} : \mathbf{Mark} \times \mathbf{Conv} \to \mathbf{Conv}$$

By using this as the backward action, and the self-action of **Mark** as the forward action, we can form the category of mixed optics $\mathbf{Optic}_{\binom{\otimes}{\otimes'^{\Delta}}}$ of Markov kernels and expectations ([HS23]). Unlike previous examples, the forward part of an optic $\binom{A}{A'} \to \binom{B}{B'}$ here consists of a forward map $f : \mathbf{Mark}(A, M \times B)$ which in probabilistic manner interacts with its internal state, not producing a residual $m : M$, but instead a joint distribution over the space of residuals and outputs.

While we don't directly study probability theory in this thesis, the above example — useful in reinforcement learning [SB18], where value iteration can be expressed in terms of optic composition [HS23] — illustrates the breadth of examples covered by the definition of weighted optics.

This example too does not require the full breadth of weighted optics — this is merely a mixed optic. But we note that weighted optics here provide an equivalent definition which has a more principled compartmentalisation of its components. Via Prop. 4.18 we can show that the above category is isomorphic to

$$\mathbf{Optic}_{\binom{\otimes}{\otimes'^{\Delta}}} \cong \mathbf{Optic}^{\mathbf{Conv}(\Delta(-),-)}_{\binom{\otimes}{\otimes'}}$$

where on the right side we have pulled out the weight from the action, leaving only the monoidal product $\otimes'$ in the backward pass. This gives us an interpretation of mixed optics used in value iteration as corepresentably weighted optics with self-actions of two monoidal categories.

The reason why we were able to do that is because Δ is a *strong* monoidal functor. While the previous example is captured in [HS23] with merely mixed optics, the following example is a construction that requires the full generality of weighted optics because it contains an analogue of Δ which is not strong, but only lax.

Example 4.23 (Smooth maps and linear maps)**.** Recall the setting of the cartesian category **Smooth** (Definition 2.2), the monoidal category $\mathbf{FVect}_{\mathbb{R}}$ (Definition 2.4), and their relationship via the lax monoidal functor ι (Lemma 2.24). Then by setting **Smooth** as forward self-action (denoted by $\times$), and $\mathbf{FVect}_{\mathbb{R}}$ as the backward self-action (denoted by $\otimes'$) we can define the category

$$\mathbf{Optic}_{\binom{\times}{\otimes}}^{\mathbf{Smooth}(-,\iota(-))}$$

whose forward passes are smooth functions, and backward maps are bilinear maps. Explicitly, a morphism $\binom{A}{A'} \to \binom{B}{B'}$ here consists of the following three pieces of data, quotiented out by the equivalence relation in Eq. (4.16):

- A coparameter $\binom{M}{M'}$, where $M : \mathbf{Smooth}$ and $M' : \mathbf{FVect}_{\mathbb{R}}$;
- A map $s : \mathbf{Smooth}(M, \iota(M'))$;
- A coparametric map $\binom{f}{f'} : \mathbf{Smooth} \times \mathbf{FVect}_{\mathbb{R}}{}^{\mathsf{op}} \begin{pmatrix} A & M \times B \\ A' & M' \otimes B' \end{pmatrix}$, i.e. a pair of maps $f : \mathbf{Smooth}(A, M \times B)$ and $f' : \mathbf{FVect}_{\mathbb{R}}(M' \otimes B', A')$.

This will be instrumental in thinking about neural networks. Intuitively, the map f is the forward map of our neural network. In addition to computing the output B it also explicitly models the intermediate result M that will be needed for the backward pass. The object M is most often A. If we set $M' = [B', A']$ as the space of linear maps $B' \multimap A'$, then the s could be thought of as the Jacobian of $A \xrightarrow{f} M \times B \xrightarrow{\pi_B} B$, and f' as the evaluation map $\mathsf{eval}_{B',A'}$. We describe all of this in detail in Chapter 6.

Because ι here is merely lax, and not strong monoidal, we cannot reduce this to mixed optics.[7]

4.3.2 When are optics monoidal?

As the category of optics is defined as a **coPara** construction, we might wonder whether existing theorems for monoidality of **Para** (Prop. 3.39 and Theorem 3.41) can be appropriated for this coparametric setting. That is indeed the case. To turn **coPara** monoidal the requirements on the base actegory are equivalent requirements to those for turning **Para** monoidal.[8]

[7] If we tried reducing it, we'd find that the forward actegory isn't strong, but lax. On the other hand, theere is a natural place for a lax component in the definition of weighted optics: in the weight.

[8] A difference between these arises when we study weakenings of actegories. Requirements for **coPara** can be weakened to a lax monoidal actegory, and requirements for **Para** to an *oplax* monoidal actegory.

Proposition 4.24 (Monoidal structure of **coPara**, [CGHR22, p. 238])**.** *Let $(\mathcal{C}, \bullet)$ be a monoidal $\mathcal{M}$-actegory with the underlying product $(\boxtimes, J)$. Then $\mathbf{coPara}_{\bullet}(\mathcal{C})$ becomes a monoidal bicategory, defined analogous to Prop. 3.39, where the monoidal product of two coparametric morphisms*

$$(M, f : A \to M \bullet B) \quad \textit{and} \quad (N, g : C \to N \bullet D)$$

is the pair $(M \otimes N, f \boxtimes_{\mathfrak{p}} g)$ where

$$f \boxtimes_{\mathfrak{p}} g \coloneqq \boxed{A \boxtimes C \xrightarrow{f \boxtimes g} (M \bullet B) \boxtimes (N \bullet D) \xrightarrow{c^{-1}_{M,B,N,D}} (M \otimes N) \bullet (B \boxtimes D)}$$

Let's unpack what this means for optics. To form optics as a coparametric category, we need a product of two actegories, and reparameterisation thereof. It is straightforward to see that the product of two actegories is monoidal if each individually is.

Proposition 4.25 (Product of monoidal actegories is a monoidal actegory)**.** *Fix a monoidal $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$ and a monoidal $\mathcal{N}$-actegory $(\mathcal{D}, \blacksquare)$. Then their product (as defined in Prop. 4.12) is a monoidal $\mathcal{M} \times \mathcal{N}$-actegory.*

When it comes to the reparameterisation, we have to ensure that the projection from the category of elements of the weight (Eq. (4.8)) is a braided monoidal functor (Definition 3.37), which is the case when the underlying weight is braided monoidal (Prop. D.5). If the weight is the hom-functor of a braided monoidal category, then this always holds (Corollary D.6).

Theorem 4.26 (Monoidal optics (compare [Ril18, Thm. 2.0.12.]))**.** *Consider the category of weighted optics $\mathbf{Optic}^{W}_{(\bullet, \blacksquare)}$. This category is monoidal if the underlying actegories are, and the weight W is a braided monoidal functor (Definition A.11).*

Proof. Direct consequence of Prop. 4.24, and Prop. 4.25. □

Let's see what this means more concretely. Consider the category of weighted optics $\mathbf{Optic}^{W}_{(\bullet, \blacksquare)}$ where the monoidal product of the forward actegory is $(\mathcal{C}, \boxtimes, J)$ and that of the backward actegory is $(\mathcal{D}, \boxtimes', J')$. Then the monoidal product of two weighted optics

$$\begin{pmatrix} A \\ A' \end{pmatrix} \xrightarrow{\left(\binom{M}{M'}, s, \binom{f}{f'}\right)} \begin{pmatrix} B \\ B' \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} C \\ C' \end{pmatrix} \xrightarrow{\left(\binom{N}{N'}, s, \binom{g}{g'}\right)} \begin{pmatrix} D \\ D' \end{pmatrix}$$

is a weighted optic of type $\binom{A\boxtimes C}{A'\boxtimes' C'} \to \binom{B\boxtimes D}{B'\boxtimes' D'}$ whose residual is the pair $\binom{M\otimes N}{M'\otimes' N'}$, the map connecting them is $\phi^W_{M,N}(s,t)$, and the implementation maps are $\binom{f\boxtimes_{\mathrm{p}} g}{g'\boxtimes^{\mathrm{p}} f'}$.

Example 4.27 (Self actions)**.** Given two braided monoidal categories $(\mathcal{M},\otimes,I)$ and $(\mathcal{M}',\otimes',I')$ and a braided monoidal functor $W : \mathcal{M}^{\mathrm{op}} \times \mathcal{M}' \to \mathbf{Set}$ the category $\mathbf{Optic}^W_{\binom{\otimes}{\otimes'}}$ is monoidal. If $\mathcal{M} = \mathcal{M}'$ then this reduces to the monoidal category of non-mixed optics $\mathbf{Optic}(\mathcal{M})$.

We can go further, and say when the category of optics is braided, or symmetric monoidal. Following arguments in Section 3.2.2, both of these require the acting category $\mathcal{M}$ to be symmetric.

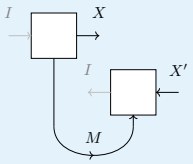
Proposition 4.28. *If the monoidal $\mathcal{M}$-actegories $(\mathcal{C},\bullet)$ and $(\mathcal{D},\blacksquare)$ are braided (resp. symmetric), then* $\mathbf{Optic}_{(\bullet\,,\blacksquare)}$ *is braided (resp. symmetric) monoidal.*

This is only a sufficient condition, because optics involve a notion of a weight coupling the actegories together, allowing them to interact in non-trivial ways. We will explore this in Chapter 6.

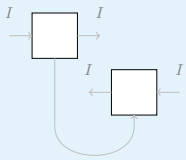
States, costates, and scalars in Optic($\mathcal{C}$) **Box 4.3.2**

We unpacked states, scalars, and costates in a general monoidal category $\mathcal{C}$ (Box 2.2.1) and in $\mathbf{Para}(\mathcal{C})$ (Box 3.2.3). Now we unpack them for $\mathbf{Optic}(\mathcal{C})$.

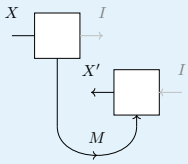
States



Scalars



Costates



$$\mathbf{Optic}(\mathcal{C})\begin{pmatrix} I & X \\ I & X' \end{pmatrix} \cong \int^{M} \mathcal{C}(I, M\otimes X)\times\mathcal{C}(M\otimes X', I)$$

$$\mathbf{Optic}(\mathcal{C})\begin{pmatrix} I & I \\ I & I \end{pmatrix} \cong \mathcal{C}(I,I)$$

$$\mathbf{Optic}(\mathcal{C})\begin{pmatrix} X & I \\ X' & I \end{pmatrix} \cong \mathcal{C}(X,X')$$

Scalars and costates (but not states) have a particularly interesting form. Costates become morphisms in the base category and scalars become endomorphisms of the monoidal unit. Additionally, if $\mathcal{C}$ is cartesian scalars become trivial, as we will see in Box 4.4.1.

4.4 Lenses, prisms, and closed optics

We now proceed to study weighted optics defined on actegories with particular structure.

4.4.1 Lenses; from cartesian actegories

We turn our focus to an important class of optics. These are optics with a forward actegory that's of a particular kind to be defined in this section: a cartesian actegory, and their particular reduced form — *lenses* — is originally what sparked a lot of research on bidirectionality. As we will see in Section 6.3.1, lenses give a denotationally equivalent, but an operationally distinct perspective on such processes. They are immensely useful in practice and, in addition to modelling deep learning, they have been used to model bayesian learning ([BHSCS23, Smi22]), database theory ([Spi23a]), dynamical systems ([Mye22a]), game theory [GHWZ18, Cap23], data accessors ([PGW17]), trading protocols ([GLP21]), server operations ([VC22]) and more ([Gav23b]). We start by giving their formal definition in the most abstract form, based on the concept of a *cartesian actegory*.[9]

Definition 4.29 (Cartesian actegory)**.** A monoidal $\mathcal{M}$-actegory $\mathcal{C}$ is a **cartesian actegory** if the monoidal product of the base $\mathcal{C}$ is given by a cartesian one. Dually, when the monoidal product is cocartesian, the resulting actegory is too.

Remark 4.30. Note that there is no requirement of $\mathcal{M}$ being cartesian, nor any requirement about the interaction of the cartesian structure of $\mathcal{C}$ and the monoidal one of $(\mathcal{M}, \otimes, I)$ and $\bullet$, other than the one already imposed by the definition of a monoidal actegory.

If the actegory is cartesian, we can formulate the actegorical analogue of the classical isomorphism in a category with products. It is a property reminiscent of the *universal product* in a cartesian category, but one which only admits the isomorphism representation (Eq. (2.16)), and not one defined by the universal morphism (Eq. (2.15)).

Proposition 4.31. *Let $(\mathcal{C}, \bullet)$ be a cartesian $\mathcal{M}$-actegory. Then we have the natural isomorphism:*[10]

$$\mathcal{C}(X, M \bullet B) \cong \mathcal{C}(X, M \bullet 1) \times \mathcal{C}(X, I \bullet B)$$

[9] Idea for this concept arose in a conversation with Matteo Capucci.

[10] While the codomain of the right element could be further reduced to B, the same can't be said for the left one — there is no way to reduce $\mathcal{C}(X, M \bullet 1)$ to $\mathcal{C}(X, M)$ in general since M isn't an object of $\mathcal{C}$.

Proof. Exhibited below.

$$
\begin{aligned}
&\mathcal{C}(X, M \bullet B) \\
\cong &\mathcal{C}(X, (M \otimes I) \bullet (1 \times B)) && \text{(Interchanger)} \\
\cong &\mathcal{C}(X, (M \bullet 1) \times (I \bullet B)) && \text{(Universal property of the product)} \\
\cong &\mathcal{C}(X, M \bullet 1) \times \mathcal{C}(X, I \bullet B)
\end{aligned}
$$

□

Example 4.32. Every cartesian category $\mathcal{C}$ canonically becomes a cartesian $\mathcal{C}$-actegory by virtue of being a monoidal $\mathcal{C}$-actegory (Example 3.36).

Remark 4.33. Given a self-action of a cartesian category we have established that connected components of the **coPara** construction on it trivialise (Corollary 4.7). Cartesian actegories allow us to circumvent this problem, as the necessary condition for trivialisation — the monoidal unit of $\mathcal{M}$ being terminal — is not something that holds for a general cartesian actegory.

The example of the above follows. In it, despite the lack of cartesian structure of $\mathcal{B}$, the braided monoidal functor produces a cartesian actegory via reparameteristaion.

Example 4.34. Let $(\mathcal{C}, \times, 1)$ be a cartesian category, $\mathcal{B}$ a braided monoidal category, and let $E : \mathcal{B} \to \mathcal{C}$ be a strong braided monoidal functor (Definition A.11). Then $\times^E : \mathcal{C} \times \mathcal{B} \to \mathcal{C}$ is a cartesian $\mathcal{B}$-actegory.

We are now ready to study cartesian optics.

Definition 4.35 (Cartesian optic)**.** Consider optics $\mathbf{Optic}^{W}_{(\bullet\,,\blacksquare)}$ where the forward actegory is a cartesian actegory. We call morphisms in this category **cartesian optics**[11].

As mentioned, these have been in the literature defined under the name of "lens", though the actual definition often varies. There seems to be a general "folkloric" definition of what a lens is, but many different mathematical formulations, and no clear hierarchy between them. In this thesis, we claim that a lens $\binom{A}{A'} \to \binom{B}{B'}$ is a kind of a bidirectional process that allows us to "extract" a morphism of type $A \to B$, and deterministically separate it from the backward one. To the best of our knowledge the above definition of a cartesian optic captures this phenomenon in the most general setting. Formally, this means the following.

[11] Not to be confused with the property of optics being a cartesian category, which holds in an even more specialised case.

Proposition 4.36 (Compare [Ril18, Proposition 2.0.4.])**.** *In any cartesian weighted optic we can factor out the forward pass:*

$$\mathbf{Optic}^{W}_{(\bullet\blacksquare)}\begin{pmatrix} A & B \\ A' & B' \end{pmatrix} \cong \mathcal{C}(A,B) \times \mathbf{Optic}^{W}_{(\bullet\blacksquare)}\begin{pmatrix} A & 1 \\ A' & B' \end{pmatrix}$$

Proof.

$$\begin{aligned}
&\mathbf{Optic}^{W}_{(\bullet\blacksquare)}\begin{pmatrix} A & B \\ A' & B' \end{pmatrix} \\
\text{(Def.)} \quad &= \int^{M,M'} \mathcal{C}(A, M \bullet B) \times W(M,M') \times \mathcal{D}(M' \blacksquare B', A') \\
\text{(Prop. 4.31)} &\cong \int^{M} \mathcal{C}(A, M \bullet 1) \times \mathcal{C}(A, I \bullet B) \times W(M,M') \times \mathcal{D}(M' \blacksquare B', A') \\
&\cong \mathcal{C}(A,B) \times \int^{M,M'} \mathcal{C}(A, M \bullet 1) \times W(M,M') \times \mathcal{D}(M' \blacksquare B', A') \\
\text{(Def.)} \quad &= \mathcal{C}(A,B) \times \mathbf{Optic}^{W}_{(\bullet\blacksquare)}\begin{pmatrix} A & 1 \\ A' & B' \end{pmatrix}
\end{aligned}$$

□

This is how lenses historically originated, by being described as gadgets with a separate forward and backward passes. This kind of formalism that separates out the backward and forward passes necessitates a different composition rule, which as we'll see is denotationally equivalent, but operationally quite different than the one given directly by optics (Definition 4.14).

All of the definitions of lenses in the literature are special cases of *mixed optics* (Box 4.3.1), as weighted optics are only introduced in this thesis. In [CEG+22] lenses are defined as mixed optics whose forward actegory is given by the self-action of a cartesian category $\mathcal{C}$ (thus necessitating that the acting category of both the forward and the backward pass is $\mathcal{C}$). In such a setting Prop. 4.36 further reduces to

$$\mathbf{Optic}_{(\times\blacksquare)}\begin{pmatrix} A & B \\ A' & B' \end{pmatrix} \cong \mathcal{C}(A,B) \times \mathcal{D}(A \blacksquare B', A')$$

In this setting, we take the liberty of providing a definition of composition and identity.

Definition 4.37 (Lens composition and identity)**.** Given

$$\begin{pmatrix} A \\ A' \end{pmatrix} \xrightarrow{\binom{f}{f'}} \begin{pmatrix} B \\ B' \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} B \\ B' \end{pmatrix} \xrightarrow{\binom{g}{g'}} \begin{pmatrix} C \\ C' \end{pmatrix}$$

where $f : \mathcal{C}(A, B)$ and $f' : \mathcal{D}(A \blacksquare B', A')$ (analogously for $\binom{g}{g'}$) their composite is defined as a lens whose forward part is $f \fatsemi g$ and the backward part is $A \blacksquare C' \xrightarrow{(g' \fatsemi^{\mathsf{p}} f')^{\mathsf{graph}(f)}} A'$ (where $\mathsf{graph}(f)$ is defined in Definition E.15). To specify the backward pass here we have used parametric compositon and reparameterisation thereof (Definition 3.14). Identity $\binom{A}{A'} \to \binom{A}{A'}$ is defined as $\mathrm{id}_A : A \to A$ on the forward pass and reparameterisation of $\eta^{\blacksquare}_A : 1 \blacksquare A' \to A'$ with $!_A : A \to 1$ on the backward one. It is routine to check that composition and identity are strictly preserved going between the abstract and the concrete representation.

Many others ([Ril18]) specialise lenses further. They define them in a non-mixed setting, where the forward and the backward category are the same, and are given by the cartesian product of $\mathcal{C}$. In this setting we can further reduce the above to

$$\mathbf{Optic}(\mathcal{C})\begin{pmatrix} A & B \\ A' & B' \end{pmatrix} \cong \mathcal{C}(A, B) \times \mathcal{C}(A \times B', A') \tag{4.38}$$

which yields the more familiar representation of a lens in the literature. We will often denote this category as in Eq. (4.39). Its string diagram representation is in Fig. 4.11.

$$\mathbf{Lens}(\mathcal{C}) \coloneqq \mathbf{Optic}(\mathcal{C}) \tag{4.39}$$

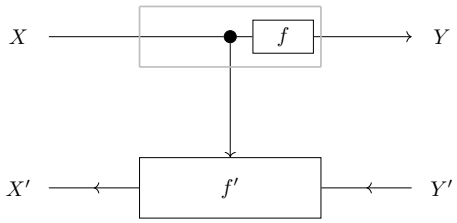


Figure 4.11: String diagram representation of a lens. The area in the gray box $(A, \mathsf{graph}(f)$ is the forward part: a morphism in $\mathbf{Para}(\mathcal{C})(X, Y)$. Notably, the input X is explicitly copied.

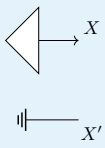
These sometimes go under the name of *bimorphic* ([Hed19]) lenses, as they allow different choices

of objects on the forward and backward pass. Some authors specialise this construction even more and require that objects are those given by the diagonal $\binom{A}{A}$ in which case they are referred to as *monomorphic*, or *simple* lenses.

States, costates, and scalars in Lens($\mathcal{C}$) **Box 4.4.1**

The descriptions of states, costates and scalars from Boxes 2.2.2 and 4.3.2 can be specialised to lenses.

States

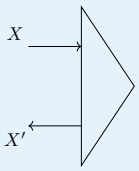


$\mathbf{Lens}(\mathcal{C})\binom{1 \quad X}{1 \quad X'} \cong \mathcal{C}(1, X)$

Scalars

$\mathbf{Lens}(\mathcal{C})\binom{1 \quad 1}{1 \quad 1} \cong 1$

Costates



$\mathbf{Lens}(\mathcal{C})\binom{X \quad 1}{X' \quad 1} \cong \mathcal{C}(X, X')$

4.4.2 Prisms; from cocartesian actegories

Dual to the setting of cartesian actegory on the forward pass is the setting of a cocartesian actegory on the backward pass. Analously to the setting above, this is in the literature known under the name of *prism* ([CEG+22, Def. 3.16]). These yield a generalisation of the definition of optics for coproducts (Example 4.21) where we allow the forward category to be potentially different than the backward one.

Definition 4.40 (Cocartesian optics)**.** Consider the category of optics $\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}}$ for a backwards cocartesian category. We call this category the category of the category of **cocartesian optics**, and often refer to them as prisms.

In such a case, we can exhibit a dual reduction which extracts out the backward pass:

Proposition 4.41 (Compare [Ril18, Sec. 4.2])**.** *In any cocartesian weighted optic we can factor*

out the backward pass:

$$\mathbf{Optic}^{W}_{(\bullet,\blacksquare)}\begin{pmatrix}A & B\\ A' & B'\end{pmatrix} \cong \mathbf{Optic}^{W}_{(\bullet,\blacksquare)}\begin{pmatrix}A & B\\ A' & 0\end{pmatrix} \times \mathcal{D}(B', A')$$

The proof follows analogously to the one of Prop. 4.36.

While we do not directly study prisms in this thesis, we mention them for completeness, and to bring to attention an important disambiguation thereof.

Remark 4.42 (Two kinds of prisms)**.** In the literature there are two non-isomorphic constructions referred to as prisms: one arising out of coproducts or generally cocartesian actegories ([Ril18, CEG+22]) and another one arising as lenses in the opposite of a cartesian category ([Spi22a, Ex. 2.4]). Only the former has 1) a clear operational interpretation and 2) a form that is compositional i.e. allowing us to compose prisms with lenses, obtaining affine traversals ([CEG+22, Sec. 3.2.1]). For more detail see [Gav23c].

4.4.3 A lens and a prism at the same time

Cartesian actegories allow us to make a particularly interesting reduction for **coPara**.

Theorem 4.43. *Let $(\mathcal{C}, \bullet)$ be a cartesian $\mathcal{M}$-actegory. Then we have the following factorisation:*

$$\mathbf{coPara}_{\bullet}(\mathcal{C})(X, Y) \cong \mathcal{C}(X, Y) \times \mathbf{coPara}_{\bullet}(\mathcal{C})(X, 1)$$

Proof.

$$\begin{aligned}
&&&\mathbf{coPara}_{\bullet}(\mathcal{C})(X, Y)\\
&\text{(Def.)}&&= \sum_{M:\mathcal{M}} \mathcal{C}(X, M \bullet Y)\\
&\text{(Prop. 4.31)}&&\cong \sum_{M:\mathcal{M}} \mathcal{C}(X, M \bullet 1) \times \mathcal{C}(X, I \bullet Y)\\
&&&\cong \mathcal{C}(X, Y) \times \sum_{M:\mathcal{M}} \mathcal{C}(X, M \bullet 1)\\
&\text{(Def.)}&&= \mathcal{C}(X, Y) \times \mathbf{coPara}_{\bullet}(\mathcal{C})(X, 1)
\end{aligned}$$

□

Intuitivelly, this tells us that any coparametric map $f : X \to M \bullet Y$ of a cartesian actegory can

be decomposed into the horizontal part with a trivial vertical component (a map $X \to Y$) and a vertical part with a trivial horizontal component (a map $X \to M \bullet 1$). In a way, we have seen this happening already in Prop. 4.36. We can ponder what this means for optics. Recall that they are defined as an action on the product of $\mathcal{C}$ and $\mathcal{D}^{\mathsf{op}}$

Proposition 4.44 (Product of cartesian actegories is a cartesian actegory)**.** *Consider a cartesian $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$ and a cartesian $\mathcal{N}$-actegory $(\mathcal{D}, \blacksquare)$. Then their product (as defined in Prop. Prop. 4.12) is a cartesian $\mathcal{M} \times \mathcal{N}$-actegory.*

As our backward actegory is always dualised — this means that for optics we need a *cocartesian* $\mathcal{M}'$-actegory $(\mathcal{D}, \blacksquare)$. In such a case, following Theorem 4.43 we get the following isomorphism:

$$\mathbf{Optic}_{(\bullet,\blacksquare)}\begin{pmatrix} A & B \\ A' & B' \end{pmatrix} \cong \mathcal{C}(A, B) \times \mathcal{D}(B', A') \times \mathbf{Optic}_{(\bullet,\blacksquare)}\begin{pmatrix} A & 1 \\ A' & 0 \end{pmatrix}$$

This is a reduction that is both lens-like, and prism-like at the same time! It can be seen as a generalisation of optics constructed on a biproduct category.

4.4.4 Closed optics; from closed actegories

So far we've described actegories which parameterise systems *internally*. This means that any parametric map $f : P \bullet A \to B$ is a morphism of the base category. For instance, if our base category is **Smooth**, then f is a smooth map. However, sometimes we're interested in a map f which is smooth only in one variable, and, say, linear in the other. In this case we're interested in *external* parameterisation. This is the main idea behind *closed* actegories, which generalise the setting of a monoidal closed category. Recall that monoidal closure (Definition E.19) means that for every $D : \mathcal{D}$ the functor $- \otimes D$ has a right adjoint denoted by $\underline{\mathcal{D}}(D, -)$ which forms the internal hom functor $\underline{\mathcal{D}}(-, -) : \mathcal{D}^{\mathsf{op}} \times \mathcal{D} \to \mathcal{D}$ such that there is an isomorphism

$$\mathcal{D}(X \otimes Y, Z) \cong \mathcal{D}(X, \underline{\mathcal{D}}(Y, Z))$$

natural in all bound variables. A closed actegory generalises this setting into that with two categories, while preserving the essence of the idea.

Definition 4.45 (Closed actegory (compare [CG23, Example 3.2.6.]))**.** Let $(\mathcal{D}, \blacksquare)$ be a $\mathcal{M}$-actegory.

If for every $D : \mathcal{D}$ the functor $-\blacksquare D : \mathcal{M} \to \mathcal{D}$ has a right adjoint, we call $(\mathcal{D}, \blacksquare)$ a **closed** actegory.[12]

We often denote the right adjoint as $\underline{\mathcal{D}}(D, -)$. In this setting we can generalise the standard tensor-hom isomorphism to

$$\mathcal{D}(M \blacksquare X, Y) \cong \mathcal{M}(M, \underline{\mathcal{D}}(X, Y)) \tag{4.46}$$

leading us to the following proposition.

Proposition 4.47 (External parameterisation (compare [Smi22, Def. 3.2.8.]))**.** *When the actegory is closed, we have the following isomorphism* $\mathbf{Para}_{\blacksquare}(\mathcal{D})(X, Y) \cong \mathcal{M}/\underline{\mathcal{D}}(X, Y)$.[13]

Example 4.48 (Monoidal closed category $\mathcal{C}$)**.** Any monoidal closed category $\mathcal{C}$ is a closed actegory whose action is given by the self-action of the monoidal product.

An especially relevant example is the category $\mathbf{FVect}_{\mathbb{R}}$ with monoidal structure $(\otimes, \mathbb{R})$, which can be used to model linearity of the backward pass in backpropagation. We can state what this means abstractly for weighted optics whose backward actegory is closed, and weight is corepresentable with a lax monoidal functor.

Definition 4.49 (Closed optics)**.** Consider the category $\mathbf{Optic}^{W}_{(\bullet \blacksquare)}$ whose backward actegory is closed and weight is corepresented by a lax monoidal functor $j : \mathcal{M}' \to \mathcal{M}$. We call this the category of **closed**[14] **optics**.

Remark 4.50. Unlike in Prop. 4.18 we do not require j to be strong monoidal because it does not need to be a part of a reparameterisation of actegories.

Closed optics are important is because they allow us to make the following reduction to *closed lenses*, also often called *linear lenses* in the literature ([Ril18, Sec. 4.8], [CEG+22, Def. 3.14]). Despite the string "lens" in their name, they do not require the forward actegory to be cartesian, though in (Corollary 4.56) we will study what happens in that case.

Proposition 4.51. *Consider the category of closed optics* $\mathbf{Optic}^{W}_{(\bullet \blacksquare)}$. *Then we can exhibit the*

[12] If $\mathcal{M}$ is also monoidal closed, then this is equivalent to a $\mathcal{M}$-enriched category $\mathcal{D}$ with copowers. ([CG23, Example 3.2.6])

[13] Note that one can in principle define external parameterisation when $\mathcal{D}$ is merely enriched in $\mathcal{M}$, without any actions. Giving a general formulation of parameterisation that encompasses both of these settings is something we hope to explore in future work based on locally graded categories (see end of Section 3.3 and Appendix B).

[14] Sometimes also called "linear" optics.

following isomorphism

$$\mathbf{Optic}^{W}_{(\bullet)}\begin{pmatrix} A & B \\ A' & B' \end{pmatrix} \cong \mathcal{C}(A, j(\mathcal{D}(B', A')) \bullet B)$$

Proof.

$$\begin{aligned}
& \mathbf{Optic}^{W}_{(\bullet)}\begin{pmatrix} A & B \\ A' & B' \end{pmatrix} \\
\text{(Def.)} \quad & = \int^{M,M'} \mathcal{C}(A, M \bullet B) \times W(M, M') \times \mathcal{D}(M' \bullet B', A') \\
\text{(Tensor-hom (Eq. (4.46)))} \quad & \cong \int^{M,M'} \mathcal{C}(A, M \bullet B) \times W(M, M') \times \mathcal{M}(M', \mathcal{D}(B', A')) \\
\text{(coYoneda)} \quad & \cong \int^{M} \mathcal{C}(A, M \bullet B) \times W(M, \mathcal{D}(B', A')) \\
\text{(Prop. 4.18)} \quad & \cong \int^{M} \mathcal{C}(A, M \bullet B) \times \mathcal{M}(M, j(\mathcal{D}(B', A'))) \\
\text{(coYoneda)} \quad & \cong \mathcal{C}(A, j(\mathcal{D}(B', A')) \bullet B)
\end{aligned}$$

□

Analogously to cartesian optics, we can deal with concrete closed lenses directly, and provide a composition rule and identity in terms of their explicit representation.

Definition 4.52 (Closed lens composition and identity)**.** Given

$$\begin{pmatrix} A \\ A' \end{pmatrix} \xrightarrow{f} \begin{pmatrix} B \\ B' \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} B \\ B' \end{pmatrix} \xrightarrow{g} \begin{pmatrix} C \\ C' \end{pmatrix}$$

where

$$f : A \to j(\mathcal{D}(B', A')) \bullet B \quad \text{and} \quad g : B \to j(\mathcal{D}(C', B')) \bullet C$$

the composite closed lens is a map of type $A \to j(\mathcal{D}(C', A')) \bullet C$ that can compactly be written as $(f \fatsemi_{\mathsf{p}} g)^{\mu^j \fatsemi j(\fatsemi)}$. Intuitively, here we first composed f and g as coparametric maps, yielding

$$f \fatsemi_{\mathsf{p}} g : A \to (j(\mathcal{D}(B', A')) \otimes j(\mathcal{D}(C', B'))) \bullet C$$

and then reparameterised the result along

$$j(\mathcal{D}(B', A')) \otimes j(\mathcal{D}(C', B')) \xrightarrow{\mu^j} j(\mathcal{D}(B', A') \otimes' \mathcal{D}(C', B')) \xrightarrow{j(\fatsemi)} j(\mathcal{D}(C', A'))$$

where μ^j is the laxator of j and $j(\fatsemi)$ composition inside the category $\mathcal{D}$ under j.

The identity closed lens $\binom{A}{A'} \to \binom{A}{A'}$ is given by the unitor $\eta^{\bullet}_A : A \to I \bullet A$ reparameterised by $I \xrightarrow{\epsilon^j} j(I') \xrightarrow{j(\mathrm{id}_{A'})} j(\mathcal{D}(A', A'))$. It is routine to show that composition and identity is strictly preserved going between the abstract and the concrete closed optic representation.

Example 4.53 (Markov kernels and expectation)**.** The category of mixed optics of Markov kernels and expectations (Example 4.22) is a category of closed optics, as **Conv** is monoidal closed ([HS23, Ex. 4]).

Example 4.54 (Smooth maps and linear maps)**.** The category of weighted optics of smooth and linear maps (Example 4.23) is a category of closed optics, as $\mathbf{FVect}_{\mathbb{R}}$ is monoidal closed.

Specifically, the composition of linear lenses $f : A \to B \times (B' \multimap A')$ and $g : B \to C \times (C' \multimap B')$ in **Smooth** is the following composite in $\mathcal{C}$:

$$A \xrightarrow{f} B \times (B' \multimap A') \xrightarrow{g \times (B' \multimap A')} C \times (C' \multimap B') \times (B' \multimap A') \xrightarrow{C \times \multimap_{\fatsemi}} C \times (C' \multimap A') \tag{4.55}$$

And lastly, if the forward actegory is additionally cartesian then, like before, we can separate out the forward pass from the backward one.

Corollary 4.56 (Optics with cartesian on the forward and closed actegories on the backward pass)**.** *Let the category* $\mathbf{Optic}^W_{(\bullet,\blacksquare)}$ *be the category of optics where the actegory* $\bullet$ *on the forward pass is cartesian, and the actegory* $\blacksquare$ *on the backward pass is a closed. Then we have*

$$\mathbf{Optic}^W_{(\bullet,\blacksquare)}\left(\begin{matrix} A & B \\ A' & B' \end{matrix}\right) \cong \mathcal{C}(A, B) \times \mathcal{C}(A, j(\mathcal{D}(B', A')) \bullet 1)$$

If the forward actegory is given by the self-action of a cartesian actegory, then we can reduce the above further to

$$\mathbf{Optic}^W_{(\times,\blacksquare)}\left(\begin{matrix} A & B \\ A' & B' \end{matrix}\right) \cong \mathcal{C}(A, B) \times \mathcal{C}(A, j(\mathcal{D}(B', A')))$$

Out of the two examples above, only smooth and linear maps are cartesian, as **Mark** does not have products.

4.5 Bidirectionality in the literature

A lot of research on optics took place because they arose as a generalisation of the category of lenses in a cartesian category, and lenses are a widely studied construction (Section 4.5). But optics aren't the *only* generalisation of lenses. The category of *dependent lenses* ([BCG+21]) is another generalisation that was rediscovered numerous times under the names such as "the category of polynomial functors" ([GK13, Spi22a, Spi20]) (often denoted[15] by **Poly**), or "the category of containers" ([AAG03, AAGM03, ALS10]) (often denoted by **Cont**).[16]

What is the category of dependent lenses? It is one where the type of the backward part — both on object and morphism level — depends on the value of the forward one. Instead of having independent choices of A and A' in $\mathcal{C}$ forming $\binom{A}{A'}$, now the possible choices for the backward object depend on the forward one. In **Set** a possible notation for these kinds of objects would be $\binom{a:A}{A'(a)}$ where A' is now a function of type $A \to \mathbf{Set}$. Similar idea applies to morphisms. The forward part $f : A \to B$ stays the same, and the backward part f' is now of type $\sum_{a:A} B'(f(a)) \to A'(a)$. It differs from the non-dependent backward type $A \times B' \to A'$ in two ways: 1) the pair type $A \times B'$ is replaced with the *dependent pair* type $\sum_{a:A} B'(f(a))$, and 2) the function type $A \times B' \to A'$ is replaced with the *dependent function type*. The definition of dependent lenses works in any base category $\mathcal{C}$ with pullbacks, where it is denoted by $\mathbf{DLens}(\mathcal{C})$ (analogously, $\mathbf{Poly}(\mathcal{C})$ or $\mathbf{Cont}(\mathcal{C})$) and defined as the Grothendieck construction of the pointwise opposite of the contravariant slice functor $\mathcal{C}/- : \mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$ (Example D.16).

When the base category is **Set**, this category is often referred to as simply **DLens**, **Poly**, and **Cont**, respectively. It is an important construction that has found uses in type theory ([VG15, AAG03, AAGM03]), mathematical theory of interaction ([NS23]), dynamical systems ([Mye22a]), and more. It is rich in structure, admitting all small limits and colimits, exponentials, initial algebras, final coalgebras, and more than a dozen interesting monoidal products ([Spi23b, NS23]).

[15] Not to be confused with the category $\mathbf{Poly}_R$ defined in [CCG+20, BCS09] and this thesis.

[16] Despite being the same category, the first category is named after its morphisms, and the second one after its objects.

Why not Poly? **Box 4.5.1**

As **Poly** is a versatile construction rich in structure, one might wonder why we are not using it as a foundation of bidirectionality in this thesis. There are a few reasons.

- **Baked-in determinism.** As formation of dependent lenses over $\mathcal{C}$ requires $\mathcal{C}$ to have products, this necessitates that the interaction between the forward and the backward pass is *deterministic*. That is, to make composition in $\mathbf{DLens}(\mathcal{C})$ is associative we need to assume that copying the input $a : A$ and applying $f : A \to B$ to each instance is the same as applying f and copying A (see Fig. 2.6 or [Gav22b]). This is not an assumption that holds in bayesian settings ([BHSCS23, BHZ23, Smi22]) or general effectful settings.

- **Blindness to performance concerns.** We believe studying how systems behave operationally can give us new insights about their denotational models. This is because observing them only from the outside squashes out a lot of information which is useful in building models of their internals. And the category **Poly** is a denotational, or extensional model of interaction, describing interaction as observed from the *outside* of these systems. It is not suited for operational, or intensional software oriented approaches where we are not merely observing these systems from the outside, but building them with their internal setups in mind. This is especially important in deep learning, where different implementations of dependent lenses yield radically different performance profiles. See "Memoisation" in Chapter 6, Section 6.3.1, and [Gav22b].

- **Inadequate enforcement of typing discipline.** The forward pass, the residual, and the backward pass of bidirectional systems often have different semantics, and live in different categories. Such typing discipline is not possible to enforce in **Poly**: both the forward and the backward pass are drawn from the same base category $\mathcal{C}$, while the notion of residual is not reified at all.

- **Additivity fundamentally changes the category.** When it comes to deep learning specifically, the assumption of additivity of the second component of the backward pass yields the category $\mathbf{Poly}_A$, a fundamentally different category than **Poly**. For instance, the monoidal product on **Poly** given by products both on the forward and the backward types[a] becomes a categorical product in on $\mathbf{Poly}_A$. And the embedding $\mathbf{Poly}_A \to \mathbf{Poly}$ with these products is not a monoidal functor in any sense: it's not strict, strong, lax nor oplax. This in turn impacts the study of closure with respect to this product, for instance, which give us higher-order functions, an important feature of modern automatic differentiation frameworks (Chapter 6).

[a] In literature called the *Dirichlet*, or the *Hancock* tensor product.

All of the above questions are something we deal with in the next subsection on *dependent optics*, where we tackle the question: how to generalise the optics to a dependently typed setting so we get the best of both worlds — of optics, but also of **Poly**?

Before this, we mention that lenses and optics are not a new construction: bidirectional tranformations in general have a long history. They include many cousins of optics in lenses that we do not tackle in this thesis: Delta lenses [Cla21, Cla20, DKL19, Dis20] and Dialectica categories ([DP89]), for example. For the history of lenses we point to [Hed18], and for a general literature list related to this topic to [Gav23b].

Most closely related to weighted optics are the definitions of non-mixed optics ([Ril18]), and those of mixed optics ([CEG+22]). We unpack the definitions of their hom-sets below.

Definition 4.57. Let $\mathcal{C}$ be a monoidal category. The set of **(non-mixed) optics** $\binom{A}{A'} \to \binom{B}{B'}$ is defined as the following coend

$$\mathbf{Optic}(\mathcal{C})\binom{A \quad B}{A' \quad B'} \coloneqq \int^{M:\mathcal{C}} \mathcal{C}(A, M \otimes B) \times \mathcal{C}(M \otimes B', A')$$

Its elements are equivalence classes of triples (M, f, f'), where $M : \mathcal{C}$, $f : A \to M \otimes B$ and $f' : M \otimes B' \to A'$. They're quotiented out by the equivalence relation where $(M, f, f') \sim (N, g, g')$ if there is a residual morphism $r : M \to N$ in $\mathcal{C}$ such that the following diagrams commute:

$$\begin{array}{ccc} A & \xrightarrow{f} & M \otimes B \\ & \searrow^{g} & \downarrow{\scriptstyle r \otimes B} \\ & & N \otimes B \end{array} \qquad \begin{array}{ccc} M \otimes B' & \xrightarrow{f'} & A' \\ \downarrow{\scriptstyle r \otimes B'} & \nearrow_{g'} & \\ N \otimes B' & & \end{array} \tag{4.58}$$

Note the difference between Eq. (4.58) and Eq. (4.16). It is a good exercise to check that these are indeed different ways of stating the same thing.[17] Mixed optics, on the other hand, conceptually separate the forward category, the acting category, and the backward category.

Definition 4.59. Let $(\mathcal{C}, \bullet)$ and $(\mathcal{D}, \blacksquare)$ be $\mathcal{M}$-actegories. The set of **mixed optics** $\binom{A}{A'} \to \binom{B}{B'}$

[17] Hint: there is a canonical way to factorise any morphism in $\mathbf{tw}(\mathcal{C})$.

is defined as the following coend, quotiented out in the same way as Eq. (4.58).

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

4.5.1 Dependent optics

As dependent lenses are a distinct and important generalisation of lenses on a category with products to the one with *pullbacks*[18] an important question surfaced:

Can optics be made dependent in the same way?

This is an infamous question that has attracted a lot of attention and spawned a myriad of proposed solutions ([BCG+21, Ver23, Win23, Mil21a, Cap22]) in the span of approximately two years (which were in the middle of my PhD). Additionally, during those two years different generalisations of optics were defined in parallel ([Mil22, Mil21b]) which lead to (at the time of the writing of this thesis) a situation with many competing generalisations of optics, and their unclear hierarchy.[19] The main reason for this diverse jungle of solutions is that there are many different things the phrase above "*in the same way*" could mean, and that there is a lack of agreement by the applied category theory community of what the above phrase *should* mean.

There are at least four different characterisations of dependent optics. We first provide a short description, and then elaborate each of them.

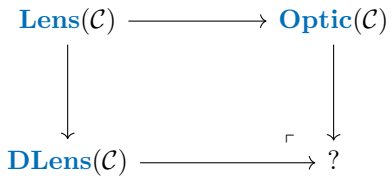
1. **Actions.** The category **Lens**($\mathcal{C}$) is an example of an optic for the self-action of a cartesian category (Eq. (4.38)). But the category **DLens**($\mathcal{C}$) is not, because optics do not allow the type of the backward pass to be dependent on the values of the forward one. Is there a generalisation of optics such that instantiating it with appropriate actions yields the category of dependent lenses?

2. **Pushout.** There exist canonical embeddings of non-dependent lenses into both optics and dependent lenses. This suggests a question: is there a generalisation of optics arising as the

[18] A category with all pullbacks doesn't necessarily have all products: it only does if it also has a terminal object. This fact seems to be the main source of tension of what's to come.

[19] The names of these generalisations appear to be obtainable by prefixing the noun "optic" any of the adjectives such as "indexed", "fibre", "dependent", or "presheaf".

pushout of these embeddings (Eq. (4.60))?



(4.60)

3. **Non-determinism.** The requirement that morphisms in our base are deterministic (Fig. 2.6) is a crucial component in ensuring lenses form a category. It arises in defining lens composition, where we need comonoids, and in proving this composition is associative, where we need maps to preserve these comonoids. Operationally, this means that we need to be able to copy the input to the lens, and slide forward maps through it without any emergent effects. As elaborated in Chapter 6 ("Memoisation" bullet point), Section 6.3.1, and [Gav22b], this is one way of composing bidirectional systems, via *checkpointing*.

   The framework of optics explicitly shows us something well-known in the deep learning community: this is not the only way of composing bidirectional systems. We can instead store intermediate results of the forward pass in memory, relinquishing us from the need to recompute them again. This in turn removes the requirement that maps preserve comonoids, i.e. that they are deterministic, allowing us to model bidirectional systems where the forward map is probabilistic (Example 4.22). Can the same kind of reasoning be applied to the setting of dependent lenses, and what does it mean for something to be dependent on the output of a non-deterministic map?

4. **Coproducts.** Even if the base category $\mathcal{C}$ has coproducts[20], **Lens**$(\mathcal{C})$ does not have all coproducts. It has *some* coproducts, for instance those of the form $\binom{X}{Z} + \binom{X}{Z}$ which share the backward type. On the other hand, the category **DLens**$(\mathcal{C})$ does have all coproducts. If **DLens**$(\mathcal{C})$ arises as the coproduct completion[21] of **Lens**$(\mathcal{C})$, can we define dependent optics as the coproduct completion of **Optic**$(\mathcal{C})$?

Let's unpack all of them. The characterisation with actions is the most common one — appearing as a proposed solution in [BCG+21, Ver23, Mil21a, Cap22]. For instance, the work of [Ver23], given a bicategory $\mathcal{B}$ and two $\mathcal{B}$-indexed bicategories L, R defines dependent optics as a particular kind

[20] In a way that interacts well with existing pullbacks, making it an *extensive* category.
[21] Not to be confused with *the free coproduct completion* which does not preserve any existing coproducts.

of a coend construction. They show that the category of dependent lenses and the category of optics both arise as particular (but different!) instantiations of $\mathcal{B}, L, R$. Therefore, this satisfies the first characterisation of dependent optics as ones arising from actions. But it doesn't satisfy the pushout characterisation. This is simply because this work does not provide a definition of a *single* category which both dependent lenses and optics include into, but rather a mechanism for producing them separately. Likewise, the resulting categories of optics and dependent lenses have shared components that do not align with their semantics: the forward pass of dependent lenses is constructed in a different manner than the forward pass of optics.

This brings us to the pushout characterisation, first posed in [BCG+21]. Here, $\mathcal{C}$ is a category with all finite limits[22], **Optic**($\mathcal{C}$) is the category of (non-mixed) optics defined by the cartesian monoidal structure of the products in $\mathcal{C}$, and **DLens**($\mathcal{C}$) is the category of dependent lenses over $\mathcal{C}$. The embedding of lenses into optics is given by Eq. (4.39), while the embedding of lenses into dependent lenses is the embedding defined in [NS23, Sec. 2.3.3]. This characterisation ensures that dependent optics have a shared subcategory of non-dependent lenses, consistent with embeddings arising either through dependent lenses or optics.

While this a well-defined characterisation, the simplicity of its solution suggests that a more refined approach is needed. Namely, the embedding from lenses into optics is an isomorphism, and pushouts along isomorphisms are isomorphisms. Via this characterisation, then, dependent optics are simply dependent lenses! This is unsatisfactory for two reasons. Firstly, it does not tell us how to generalise dependent optics. In the non-dependent setting by moving from lenses to optics we lost some of our chains: we discovered that we can forget about the cartesian structure, because we need only the monoidal one to form optics.[23] But here it is not clear that there's any structure we can remove, and it's not clear how to generalise this solution.

The second problem is that this definition does not inform implementation. Like any other definition by universal properties, this definition is a specification only up to isomorphism. And the particular isomorphism here is blind to the performance profile of the resulting implementation. But performance in the context of deep learning is of utmost importance. Ignoring this issue we'd be forced to go outside of our compositional framework to achieve a performant implementation, when in fact it should be the other way around: performance should come because of compositionality, not at its expense.

[22] i.e. a category with all equalisers and all binary products i.e. a category with all pullbacks and a terminal object.
[23] Of course, we've seen that we can generalise far beyond monoidal too!

This leads us to search for a pushout in a more refined ambient setting than just **Cat**. As noted in [Gav22b], one can study the weakening of **Optic**($\mathcal{C}$) to a bicategory **2Optic**($\mathcal{C}$), in which case the embedding **Lens**($\mathcal{C}$) $\to$ **Optic**($\mathcal{C}$) becomes a *lax* functor,[24] which now detects the different composition rule of lenses and optics. 2-categories and lax functors form a category, and this category is the simplest setting that has the needed level of fidelity to distinguish operational issues outlined above, and prevent the pushout from being an isomorphism. Alternatively, the pushout might still need to be weakened in order to exist, though it's not clear to what extent. This is because this lax functor is not a part of an isomorphism anymore, but rather an adjunction. The simplest setting where this adjunction can live is the 2-category of 2-categories, lax functors and *icons* ([Lac10, Gav22b]). Another option is **LxDbl**, the 2-category of double categories, lax double functors, and lax transformations ([Gra19, p. 148]). In such cases we might want to ask for a 2-pushout, a lax pushout or a quasi-pushout. It's not clear which one, and in the context of optics, none of these options have been acknowledged in literature.

A different path to dependent optics was explored through coproduct completions. Even if the base category $\mathcal{C}$ has coproducts[25], **Lens**($\mathcal{C}$) does not have all coproducts. It has *some* coproducts, for instance those of the form $\binom{X}{Z}+\binom{X}{Z}$ which share the backward type. On the other hand **DLens**($\mathcal{C}$) has all coproducts, suggesting that a coproduct completion[26] of **Optic**($\mathcal{C}$) could yield dependent optics. This was the idea behind solutions in [BCG+21] and [Ver23] which define categories called *indexed* and *dependent* optics, respectively, and show that they have all coproducts. It is not clear to me what the relationship between these definitions is, nor their relation to the pushout definition (Eq. (4.60)). As cherry on top, the situation is made more complicated by the fact that during the writing of this thesis it was discovered that while **DLens**(**Set**) has all coproducts, the embedding from **Lens**(**Set**) does not preserve the ones that already exist![27] For example, the coproduct of $\binom{1}{0}$ and $\binom{2}{0}$ exists in **Lens**(**Set**), but its image under the embedding into **DLens**(**Set**) is not a coproduct of the embeddings of $\binom{1}{0}$ and $\binom{2}{0}$.

Lastly, it is not abundantly clear whether these four characterisations are even seeing the whole picture. Optics for coproducts (Example 4.21) form prisms, and comparatively little research was devoted to the question of existence of dependent prisms, actions they might be formed by, the pushout of the analogous diagram, or their operational interpretation. In [Ver23] dependent prisms

[24] With κ still being a strict 1-functor.
[25] In a way that interacts well with existing pullbacks, making it an *extensive* category.
[26] Not to be confused with *the free coproduct completion* which does not preserve any existing coproducts.
[27] I learned this from Jules Hedges and Eigil Rischel.

are defined, albeit for *one* of the two different definitions of prisms in the literature (see Section 4.4.2 for disambiguation): the one that does not have the needed compositional properties. Do dependent prisms for the other definition exist, do they satisfy the pushout diagram analogous to Eq. (4.60), and what can be said about their relation to dependent lenses? It is not clear.

This concludes the problem statement of dependent optics — it's rather lengthy! Hopefully this will help us understand the direction we need to move in order to solve it better, and we believe the definition of optics using weighted **coPara** provides a good avenue for doing so.[28]

[28] Our work-in-progress Idris 2 code repository of dependent optics is at `https://github.com/bgavran/DependentOpticsIdris2`.

Chapter 5

Putting the pieces together

> Computer science is no more about computers than astronomy is about telescopes.
>
> Edsger W. Dijkstra

Having described parameterisation and bidirectionality separately, in this short chapter we will take advantage of the compositional nature of category theory and put these two concepts together. This will yield *parametric weighted optics*.

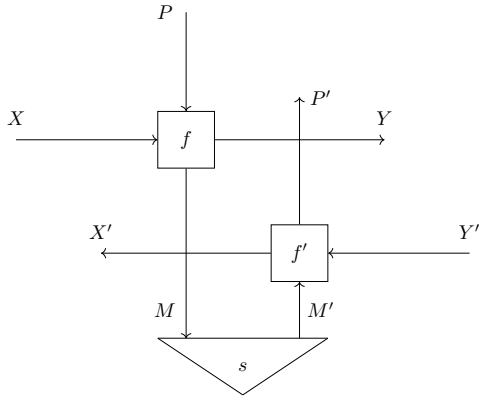


Figure 5.1: A parametric weighted optic, with its "three-legged" shape. This is a versatile construction can be used to model both neural networks in deep learning and players in game theory.

The diagram to have in mind is the one above (Fig. 5.1). Here we see the three-legged shape of parametric weighted optics[1], where each leg has both inputs flowing in and outputs flowing out. Horizontal direction (one containing $\binom{X}{X'}$ and $\binom{Y}{Y'}$) is the one where information flows that is in many ways *publicly available* — other layers of neural networks or players in game theory can plug in this interface. On the other hand, vertical direction (one containing $\binom{X}{X'}$ and $\binom{Y}{Y'}$) where information flows that is *private*. These are the parameters of a neural networks or strategies of players in game theory (on the top), or, intermediate results used for backprop or intermediate results used to calculate payoffs in game theory (on the bottom).

We think of X as the input to the neural network, for instance an encoding of an image to be classified (i.e. $\mathbb{R}^{w\times h\times 3}$ where w is the width of the image, h is its height, and 3 denotes that the image is encoded in RGB). The vertical input P here denotes the space of possible weights for the network, and Y denotes the output of this neural network, which could be yet another hidden layer to which other networks will connect, or something that is ready to be consumed by the loss function (for instance, an element of $\mathbb{R}$ denoting the classification of this image as a cat vs. dog). The part of this diagram that has primes $'$ in it denotes everything that is related to gradients and derivatives. We think of Y' as feedback the network received as a response to its produced output. Say, if the network produced an output $y : Y$, then elements of Y' denote the gradient of the loss with respect to that output y. In conjunction with the information from the forward pass, this is turned into a gradient with respect to a) the parameter $p : P$ and b) the original input $x : X$.

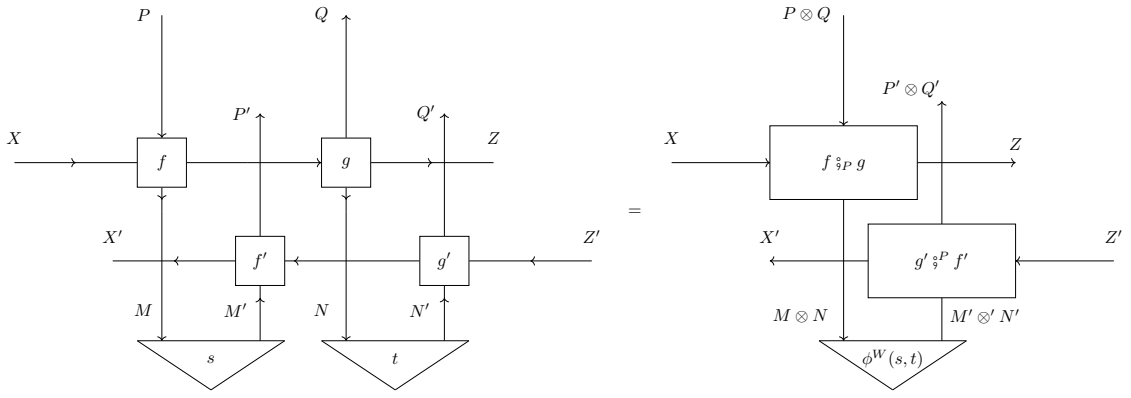


Figure 5.2: Composition of parametric optics

[1] We'll often refer to them as just "parametric optics".

Composition is again natural in this formulation. Given another box with input-output wires $\binom{Y}{Y'}$ on the right and another one with input-output wires $\binom{Y}{Y'}$ on the left, we can plug them together as in Fig. 5.2. The algebra of composition of **Para** and **Optic** automatically takes care of backpropagating the relevant information to correct ports, both on parameter level (vertical) and on the input level (horizontal).

Reparameterisation in this setting also takes on a useful form. Depicted in Fig. 5.3, it allows us to model neural network optimisers, described thoroughly in Section 8.1.3. They modify the parameters before passing it on to actual layers of the forward pass of a neural network (a central feature of Nesterov momentum (8.11)), but also perform the gradient update step on the backward one.

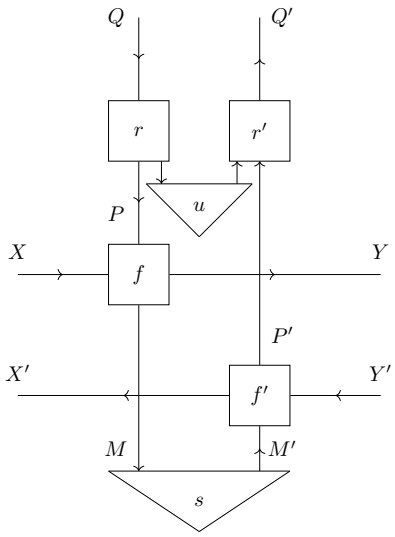


Figure 5.3: Reparameterisation of parametric optics models optimisers of neural networks.

Lastly, we also see that states, costates, and scalars of parametric optics all provide useful semantics for supervised learning (Box 5.1.1). We continue to put all of these informal stories on a concrete mathematical footing.

Contributions. *The entire chapter is novel contribution. This includes formulation of parametric weighted optics, morphisms of actegories, and the base change of* **Para**. *Fragments of this work appeared in ([CGG+22, CGHR22, CG23]).*

Epistemic status. *The epistemic status of this chapter mirrors that of the two previous ones, as it mainly involves putting these chapters together. I believe the general simplification of two previous ones compounds here, yielding a definition of parametric optics that might have an even more pragmatic compartmentalisation of all the moving pieces.*

5.1 Parametric optics and how to construct them

Recall the data we need to form weighted optics (Definition 4.14). It includes two monoidal categories $(\mathcal{M}, \otimes, I)$ and $(\mathcal{M}', \otimes', I')$, the weight $W : \mathcal{M}^{\mathsf{op}} \times \mathcal{M}' \to \mathbf{Set}$, and actions of $\mathcal{M}$ and $\mathcal{M}'$. To form parametric weighted optics, we first need to define an action of some monoidal category on $\mathbf{Optic}^{W}_{(\bullet,\blacksquare)}$. Which monoidal category is it?

This is the category of weighted optics $\mathbf{Optic}^{W}_{(\otimes,\otimes')}$ arising from self actions of $(\mathcal{M}, \otimes, I)$ and $(\mathcal{M}', \otimes', I')$. If $\mathcal{M}$ and $\mathcal{M}'$ are additionally braided, then the category $\mathbf{Optic}^{W}_{(\otimes,\otimes')}$ is monoidal, and braided (following Example 4.27 and Prop. 4.28), meaning we can act with it on $\mathbf{Optic}^{W}_{(\bullet,\blacksquare)}$.

Proposition 5.1 ([CGHR22, Prop. 10])**.** *Fix the data required to form weighted optics:*

$$\begin{gathered}
\text{A } \mathcal{M}\text{-actegory } (\mathcal{C}, \bullet) \\
\text{A } \mathcal{M}'\text{-actegory } (\mathcal{D}, \blacksquare) \\
\text{A lax monoidal functor } (W, \phi, \epsilon) : \mathcal{M}^{\mathsf{op}} \times \mathcal{M}' \to \mathbf{Set}
\end{gathered}$$

where here the monoidal categories $(\mathcal{M}, \otimes, I)$ and $(\mathcal{M}', \otimes', I')$ are additionally braided. Then we can form an action of $\mathbf{Optic}^{W}_{(\otimes,\otimes')}$ on $\mathbf{Optic}^{W}_{(\bullet,\blacksquare)}$ defined on objects as:

$$\begin{pmatrix} M \\ M' \end{pmatrix} \circledast \begin{pmatrix} X \\ X' \end{pmatrix} \coloneqq \begin{pmatrix} M \bullet X \\ M' \blacksquare X' \end{pmatrix}$$

This allows us to form *parametric weighted optics* by taking its **Para** construction.

Definition 5.2 (Parametric weighted optics (compare [CGHR22, Sec. 4]))**.** Fix the data required to form weighted optics:

$$\begin{gathered}
\text{A } \mathcal{M}\text{-actegory } (\mathcal{C}, \bullet) \\
\text{A } \mathcal{M}'\text{-actegory } (\mathcal{D}, \blacksquare) \\
\text{A lax monoidal functor } (W, \phi, \epsilon) : \mathcal{M}^{\mathsf{op}} \times \mathcal{M}' \to \mathbf{Set}
\end{gathered}$$

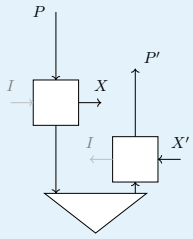
where here the monoidal categories $(\mathcal{M}, \otimes, I)$ and $(\mathcal{M}', \otimes', I')$ are additionally braided. Then we call $\mathbf{Para}_{\circledast}(\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}})$ the bicategory of parametric weighted optics.

The definition of parametric optics packs a lot of punch, since it is composed out of already complex definitions of **Para** (Definition 3.14) and $\mathbf{Optic}^{W}_{\binom{\bullet}{\blacksquare}}$ (Definition 4.14). In its full generality a parametric optic $\binom{A}{A'} \to \binom{B}{B'}$ (depicted in Fig. 5.1) consists of a choice of parameters $\binom{P}{P'}$ where $P : \mathcal{M}$ and $P' : \mathcal{M}'$; and a weighted optic $\binom{A \bullet P}{A' \blacksquare P'} \to \binom{B}{B'}$ which itself consists of a choice of residuals $\binom{M}{M'}$, a map $s : W(M, M')$, and a pair of maps $\binom{f}{f'} : \binom{A \bullet P}{A' \blacksquare P'} \to \binom{M \bullet B}{M' \blacksquare B'}$. Despite this, in many cases, especially those that pertain to neural networks, parametric optics will take on a much simpler form. We proceed to describe their states, costates and scalars, and then a general recipe for constructing parametric optics from only their forward passes.

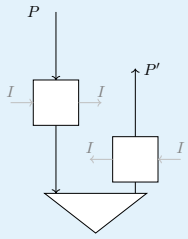
States, costates, and scalars in Para(Optic($\mathcal{C}$)) **Box 5.1.1**

Here we focus on parametric optics for the self-action of a monoidal category $\mathcal{C}$.

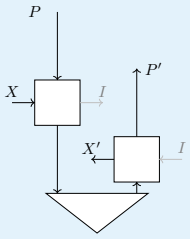
States



Scalars



Costates



$$\mathbf{Para}(\mathbf{Optic})\begin{pmatrix} I & X \\ I & X' \end{pmatrix} \cong \sum_{\binom{P}{P'}:\mathbf{Optic}} \mathbf{Optic}\begin{pmatrix} P & X \\ P' & X' \end{pmatrix}$$

$$\mathbf{Para}(\mathbf{Optic})\begin{pmatrix} I & I \\ I & I \end{pmatrix} \cong \sum_{\binom{P}{P'}:\mathbf{Optic}} \mathbf{Optic}\begin{pmatrix} P & I \\ P' & I \end{pmatrix} \cong \sum_{\binom{P}{P'}:\mathbf{Optic}} \mathcal{C}(P, P')$$

$$\mathbf{Para}(\mathbf{Optic})\begin{pmatrix} X & I \\ X' & I \end{pmatrix} \cong \sum_{\binom{P}{P'}:\mathbf{Optic}} \mathcal{C}(P \otimes X, P' \otimes X')$$

We see that states correspond simply to optics $\binom{P}{P'} \to \binom{X}{X'}$ for some $\binom{P}{P'}$. Scalars correspond to maps $P \to P'$ in $\mathcal{C}$, and costates to maps $P \otimes X \to P' \otimes X'$ in $\mathcal{C}$. Scalars will especially be important in Section 8.2 where they will model a step of supervised learning.

5.1.1 Constructing parametric optics

In most cases we are intersted in constructing parametric weighted optics from the data of only its forward pass — which is parametric as well. This is often done by first constructing *weighted optics* from their forward pass, and then showing the construction coherently translates to the parametric setting.

5.1.2 Para is functorial in the base

In Chapter 3 we have shown how to construct the bicategory **Para**$_\bullet(\mathcal{C})$ given a base $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$. We now study what happens if we have a morphism of actegories, and show that **Para** then induces a morphism of the corresponding bicategories. We start by describing a special kind of a morphism of actegories: one which does not change the acting category.

Definition 5.3 ($\mathcal{M}$-linear morphism, compare [CG23, Def. 3.3.2])**.** Let $(\mathcal{M}, \otimes, I)$ be a monoidal category, and $(\mathcal{C}, \bullet)$ and $(\mathcal{D}, \boxtimes)$ be $\mathcal{M}$-actegories. Then a **$\mathcal{M}$-linear functor** $(\mathcal{C}, \bullet) \to (\mathcal{D}, \boxtimes)$ is a functor $R^\sharp : \mathcal{C} \to \mathcal{D}$ together with a natural isomorphism

$$\begin{array}{ccc} \mathcal{C} \times \mathcal{M} & \xrightarrow{R^\sharp \times \mathcal{M}} & \mathcal{D} \times \mathcal{M} \\ {\scriptstyle \bullet}\downarrow & \swarrow{\scriptstyle \ell} & \downarrow{\scriptstyle \boxtimes} \\ \mathcal{C} & \xrightarrow[R^\sharp]{} & \mathcal{D} \end{array} \tag{5.4}$$

whose component at each $C : \mathcal{C}$ and $M : \mathcal{M}$ is explicitly the map $\ell_{C,M} : R^\sharp(C) \boxtimes M \to R^\sharp(C \bullet M)$ making the following diagrams commute for all $C : \mathcal{C}$ and $M, N : \mathcal{M}$:

$$\begin{array}{ccc} R^\sharp(C) \boxtimes I & & \\ {\scriptstyle \ell_{C,I}}\downarrow & \searrow^{\eta^{-1}_{R^\sharp(C)}} & \\ R^\sharp(C \bullet I) & \xrightarrow[R^\sharp(\eta_C^{-1})]{} & R^\sharp(C) \end{array} \qquad \begin{array}{ccccc} (R^\sharp(C) \boxtimes M) \boxtimes N & \xrightarrow{\ell_{C,M} \boxtimes N} & R^\sharp(C \bullet N) \boxtimes N & \xrightarrow{\ell_{C \bullet M, N}} & R^\sharp((C \bullet M) \bullet N) \\ {\scriptstyle \mu^{\boxtimes -1}_{R^\sharp(C), M, N}}\downarrow & & & & \downarrow{\scriptstyle R^\sharp(\mu^{\bullet -1}_{C,M,N})} \\ R^\sharp(C) \boxtimes (M \otimes N) & & \xrightarrow[\ell_{C, M \otimes N}]{} & & R^\sharp(C \bullet (M \otimes N)) \end{array}$$

Without any other qualifications, we also refer to $(R^\sharp, \ell)$ as a **lax** $\mathcal{M}$-linear functor. If ℓ is invertible, we call it a **strong**, and when it is identity a **strict** $\mathcal{M}$-linear functor.

This definition is a generalisation of the notion of a tensorial strengh (Definition E.18).

Example 5.5 (compare [CG23, Ex. 3.3.5])**.** Every strong endofunctor (Definition E.18) is a morphism of canonical right self-actions.

As we will see, a general morphism of actegories will also capture strong monoidal functors.

Definition 5.6 (Morphism of actegories, compare [CG23, Prop. 3.6.5])**.** Let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$-actegory, and $(\mathcal{D}, \boxtimes)$ be a $\mathcal{N}$-actegory. A morphism of actegories $(\mathcal{C}, \mathcal{M}, \bullet) \to (\mathcal{D}, \mathcal{N}, \boxtimes)$ is a pair of a strong monoidal functor $R : \mathcal{M} \to \mathcal{N}$ and a $\mathcal{M}$-linear functor $(R^\sharp, \ell) : (\mathcal{C}, \bullet, \mathcal{M}) \to (\mathcal{D}, \mathcal{M}, (\mathcal{D} \times R) \fatsemi \boxtimes)$. It can be depicted by the data of the following 2-cell:

$$\begin{array}{ccc} \mathcal{C} \times \mathcal{M} & \xrightarrow{R^\sharp \times R} & \mathcal{D} \times \mathcal{N} \\ \downarrow{\scriptstyle \bullet} & \swarrow{\scriptstyle \ell} & \downarrow{\scriptstyle \boxtimes} \\ \mathcal{C} & \xrightarrow[R^\sharp]{} & \mathcal{D} \end{array} \tag{5.7}$$

which on components defines a map $\ell_{C,M} : R^\sharp(C) \boxtimes R(M) \to R^\sharp(C \bullet M)$ satisfying conditions of a lax $\mathcal{M}$-linear functor (Definition 5.3). The naming conventions of laxness, strength and strictness from Definition 5.3 apply here too.

Example 5.8. Every strong monoidal functor (Definition A.6) is a morphism of canonical right self-actions.

In the next chapter we will see an example of such a functor – the functor $\mathcal{C} \to \mathbf{Lens}_A(\mathcal{C})$ that performs differentiation on a cartesian left-additive category. Interestingly, a lax monoidal functor is *not* an example of a morphism of actegories.[2] More examples of morphisms of actegories can be found in [CG23, Sec. 3.3].

With the definition of morphism of actegories in hand, we are now ready to see how **Para** acts on this, yielding a pseudofunctor between the corresponding **Para** bicategories.

Proposition 5.9 (A morphism of actegories induces a morphism of parametric bicategories)**.** *Let $(\mathcal{C}, \mathcal{M}, \bullet)$ and $(\mathcal{D}, \mathcal{N}, \boxtimes)$ be two actegories. Let $(R, R^\sharp, \ell)$ be a morphism of actegories $(\mathcal{C}, \mathcal{M}, \bullet) \to (\mathcal{D}, \mathcal{N}, \boxtimes)$.*

Then **Para** *respects that structure, inducing a pseudofunctor*

$$\mathbf{Para}((R, R^\sharp, \ell)) : \mathbf{Para}_\bullet(\mathcal{C}) \to \mathbf{Para}_\boxtimes(\mathcal{D})$$

[2] Though, it is an example of a morphism of *lax* actegories whose detailed unpacking we leave for future work.

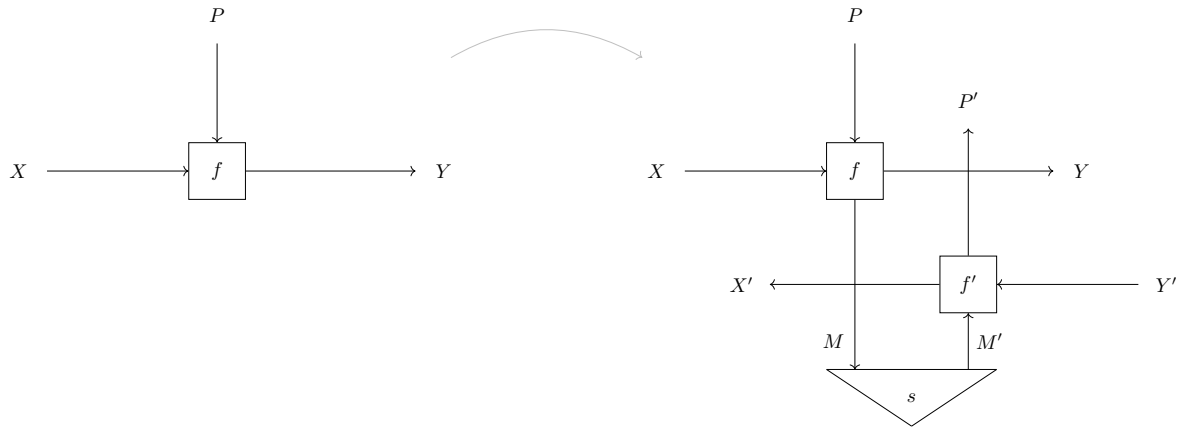


Figure 5.4: From a parametric map to a parametric weighted optic

with the following data.

- *On* ***objects*** *it borrows the action of* $R^\sharp$*;*
- *On* ***morphisms*** *at acts as below, and graphically depicted in Fig. 5.4:*

$$\mathbf{Para}_\bullet(\mathcal{C}) \longrightarrow \mathbf{Para}_\boxtimes(\mathcal{D})$$

$$\begin{array}{ccc} A & \longmapsto & R^\sharp(A) \\ {\scriptstyle (P,f)}\downarrow & & \downarrow{\scriptstyle (R(P),R^\sharp_\ell(f))} \\ B & \longmapsto & R^\sharp(B) \end{array}$$

where $R^\sharp_\ell(f)$ *is shorthand for the composite*

$$R^\sharp(A) \boxtimes R(P) \xrightarrow{\ell_{A,P}} R^\sharp(A \bullet P) \xrightarrow{R^\sharp(f)} R^\sharp(B)$$

- *On* ***2-morphisms*** *it borrows the action of* R*.*

Proof. Appendix C. □

Proposition 5.10 (When is **Para**$((R, R^\sharp, \ell))$ a 2-functor?)**.** *If the morphism of actegories* $(R, R^\sharp, \ell)$

is a strict *morphism, then so is the induced pseudofunctor* **Para**$((R, R^\sharp, \ell))$*, i.e. it becomes a 2-functor.*

This induced functor will be a central component of neural networks – as elaborated in [CGG+22, Sec. 3.1], and the second part of this thesis. Its strict 2-functoriality will tell us that starting with two composable neural networks, we get the same result if we a) augment each neural network separately with its backward pass and then compose the result, or b) compose the networks and then augment the result with its backwards pass (Fig. 5.5).

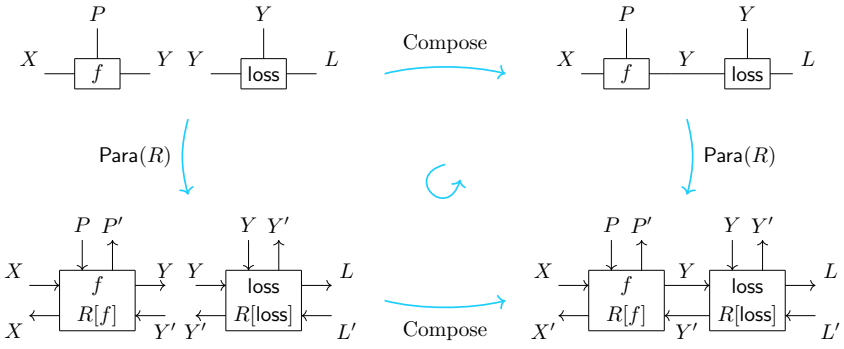


Figure 5.5: "Parametric chain rule": starting with a neural network and a loss function, we can either differentiate them individually and compose the results, or first compose them and differentiate the result. These yield the same result, enabling us to break down very large composites into manageable pieces.

Furthemore, the generalisation of this pseudofunctor to *a lax functor* is explored in [Cap23] where applications to settings such as that of game theory are studied.[3] Building on top of this, we could spend a lot of time unpacking further categorical structure — morphisms of monoidal actegories, and consequently morphisms of monoidal paras; morphisms of cartesian actegories; morphisms of cartesian paras, and so on. We don't do that in this thesis, and instead move on to applications: backpropagation, neural network architectures, and supervised learning.

We will see how optics can be specialised to a particular category $\mathbf{Lens}_A(\mathcal{C})$ of lenses additive in the backward pass, and how **Para** can be applied to a symmetric monoidal functor $\mathcal{C} \to \mathbf{Lens}_A(\mathcal{C})$ giving us a way to augment parametric maps with their derivative.

[3] Here laxness is crucial, providing an argument that a good abstraction for these settings is that of a *lax* actegory.

Part II

Applications

Chapter 6

Backpropagation

> Automatic differentiation is
> symbolic differentiation
> performed by a compiler.
>
> Conal Elliott [Ell18b]

When we think about derivatives, we often think about differentiable functions between euclidean spaces. This is a setting in which derivatives are easy to formalise in, and one which been thoroughly studied. Unfortunately, this setting does not capture numerous things done with derivatives in practice, and in theory.

Firstly, not all differentiable functions are functions between *euclidean* spaces. Deep learning has been studied on complex numbers ([BQL21]), hyperbolic spaces ([PVM+22]), finite-state automata ([HPKH20]), graphs ([WPC+21]), hypergraphs, simplicial complexes, cellular complexes ([PSHM23]), and even boolean circuits ([WZ21]) which — surprisingly — also admit a coherent notion of a derivative. Is there a precise notion of derivative that coherently captures all of these?

Secondly, euclidean spaces are not cartesian closed, don't have all coproducts, nor do they host dependent types. This means modelling derivatives in this category prevents us from reasoning about derivatives of higher-order functions[1], differentiation with branching, and differentiation on manifolds. This is especially important with the advent of numerous frameworks heavily used in practice which often provide branching or higher-order differentiation such as PyTorch ([PGM+19])

[1] Not to be confused with higher-order derivatives of functions.

or JAX ([BFH+18]), but whose categorical semantics are unknown. Indeed: what are their categorical semantics? Can we perform branching in hyperbolic spaces, or boolean circuits? What about derivatives of higher-order functions?

Lastly, we are interested in not just studying neural networks, but *implementing* them. The process of computing derivatives in a programming language is often referred to as *automatic differentiation* (AD) ([BPRS17]) where — in addition to correctness — this is done with an additional constraint of *efficiency*. There are two main efficiency concerns.

- **Order of multiplication of Jacobian matrices.** Given many Jacobian matrices to be multiplied together, different orders of their composition induce a different computational cost. All left-associated composition is often referred to as "forward-mode AD" while all-right one is called "reverse-mode AD". If n and m are respectively the dimensionality of the domain and the codomain of the composite function we are differentiating, forward-mode AD tends to be more efficient $n \ll m$, and reverse-mode AD tends to be more efficient when $m \ll n$.

- **Memoisation.** An important efficiency aspect is whether to save intermediate results of derivative computation or to recompute them from scratch. The former is often the standard in automatic differentiation, while the latter is called *gradient checkpointing* ([CXZG16, GW00, DH06]). Gradient checkpointing, i.e. always recomputing the derivatives from scratch is useful when dealing with large models where saving intermediate results would quickly consume the available GPU memory. On the other hand, if we do have enough memory, saving intermediate results is faster.

When it comes to the multiplication of Jacobians, implementing forward-mode AD in a language with complex features is well understood, admitting an elegant proof of correctness ([HSV20]). But this is not the case for reverse-mode AD. Until very recently, reverse-mode AD was only well-understood only as a compile-time source-code transformation on programming languages with limited features, such as only first order functions ([Ell18a]). Implementations of reverse-mode AD in more expressive languages often rely on their interpreted features, where a *computational graph* is built during runtime in a manner where higher-order functions and branching are evaluated, leaving only a simple first-order program. This approach suffers the main drawback that it presents a separate language (happening at runtime) outside of the reach of the compiler and any existing compiler optimisations. As such, proofs of correctness have often introduced mutable state ([PS08]),

increasing the difficulty of understanding and verifying their correctness. Recent work tackled these problems ([VS22, Vá21, APGSZ21]). The first two provide structured translation for specific languages that host higher-order types. They also provide explicit categorical semantics in terms of structure preserving functors, though explicit relationship to existing categorical frameworks such as tangent categories are not established ([CC14, CL23, Gar18]).[2] The latter work provides a purely functional and formal interpretation of ([PS08]), for the first time proving its soundness.

On the other hand, when it comes to memoisation, the distinction between gradient-checkpointed AD and non-checkpointed AD is mostly ignored. Indeed, none of the above mentioned literature acknowledge it. The only exception is [Gav22b] which provides a 2-categorical framework distinguishing between reverse mode AD that performs gradient checkpointing versus one that does not.

The points above motivate a need for a characterisation of backpropagation that has a number of properties. It needs to a) cover exotic spaces; b) host sufficient structure for higher-order functions and branching, and c) be operationally aware, providing a well defined procedure of computing these derivatives efficiently in terms of order of multiplication of Jacobians and memoisation. Is there general notion of derivative encompassing all these? Can we have a framework that is both descriptive — denotationally describing all the relevant aspects — but also *prescriptive* — equipped with a recipe for implementing it that is cognizant of the various efficiency concerns? Moreover, motivated by our constructions in Chapter 4 we ask: does this framework arise as a special case of a general construction that covers a variety of other kinds of bidirectional processes, such as bayesian learning and value iteration?

In this chapter, we provide a resounding "yes" to the last question, and a partial answer to the previous ones. We define *additively closed cartesian left-additive categories* (Definition 6.4), serving as our foundation for differentiation. In Eq. (6.10) we describe how weighted optics defined on this base model backpropagation, and how they can be reduced to their more concrete formulation $\mathbf{Lens}_A(\mathcal{C})$ of lenses whose backward passes are additive in the second component. In Section 6.3 we formally study $\mathbf{Lens}_A$, defining it as a functor, and consequently introducing *generalised reverse derivative categories* as coalgebras of this functor. This gives us a framework for categorically modelling differentiation that we use in the rest of this thesis, capturing semantics of differentiation solely in in the language of lenses. This framework covers euclidean spaces and boolean circuits (among others), but not hyperbolic spaces, or simplicial complexes, for instance. It is closely related

[2] Especially with respect to the axioms of the *vertical lift* or the *canonical flip*.

to *reverse derivative categories* ([CCG+20]) for whose axioms it serves as a justification. Likewise, while we don't directly tackle the question of higher-order differentiation[3], we believe the general categorical constructions in this thesis can provide a way forward in understanding those.

Contributions. *In this chapter, the novel contribution is the lens- and optic-theoretic axiomatisation of differentiation. Specifically, this includes the definition of an additively closed cartesian left-additive category (Definition 6.4), definition of* $\mathbf{Lens}_A$ *as an endofunctor (Theorem 6.13) and a generalised cartesian reverse differential category as its coalgebra (Theorem 6.21). What is also novel is the identification of checkpointed and non-checkpointed automatic differentiation as lens and optic composition, respectively (Section 6.3.1), originally appearing in my preprint [Gav22b]. Seeds of the idea of axiomatisation of differential structure as lenses appeared in [CGG+22], though this work still used the framework of reverse derivative categories.*

Epistemic status. *Follows that of the chapter on bidirectionality. Operational aspects were chosen in favour of dependent types, with the hope that in the long run the operational and intensional view will shed more light on these systems than a mere extensional and denotational one could. The state of the research leaves much to be desired: I want to establish formal connections to (reverse) tangent categories, provide a deeper characterisation of operational aspects, and simplify the definition of ACCLA by making it clearer what is the interaction between the (non-cartesian) monoidal closed structure of the backward pass and the cartesian structure of the forward pass. Likewise, I do not expect left-additive structure to be to be necessary in the long-run either, as we can study the category of additive maps in any category that's cartesian, and even merely just monoidal.*

6.1 What is differentiation?

We often interpret the derivative of a function as its slope at a particular point, measuring how fast the function changes at that point. For instance, starting with a function $f(x) = x^2$ on real numbers, the derivative of f is usually written as $\frac{df}{dx} = 2x$ and evaluated to a real number representing the coefficient of the slope. At the point $x = 3$ the derivative of f is 6, telling us that for each step we move to the right in the graph of the function, we move 6 steps up (Fig. 6.1).

Of course, we see that the further out we go, the less accurate the derivative is. This is because the derivative is only a linear approximation, local at a point. And the fact that we are able to

[3] Meaning derivatives of higher-order functions as opposed to higher-order derivatives of functions.

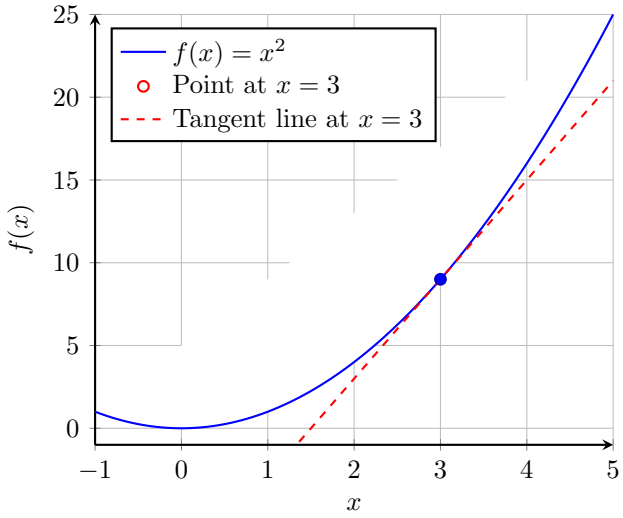


Figure 6.1: Function $f(x) = x^2$ and its tangent at the point $x = 3$.

represent it as a number is merely a consequence of the isomorphism between the vector space of real numbers $\mathbb{R}$ and the space of linear maps $\mathbb{R} \multimap \mathbb{R}$ (Box 2.2.1). In a multivariate case we will have to represent it as a matrix instead.

Take the example of a smooth function $f : \mathbb{R}^3 \to \mathbb{R}^2$ defined as

$$f(x_1, x_2, x_3) = (x_1^2 + x_1 x_2 x_3, x_3^3 + 3x_1)$$

Its derivative is a linear function of the same type $\mathbb{R}^3 \multimap \mathbb{R}^2$ that best approximates f at a particular point. It can be represented as a 2×3 matrix whose elements contain partial derivatives of f with respect to each component of the input. That is, at the point $(x_1, x_2, x_3) : \mathbb{R}^3$ the derivative of f is

$$\begin{bmatrix} 2x_1 + x_2 x_3 & x_1 x_3 & x_1 x_2 \\ 3 & 0 & 3x_3^2 \end{bmatrix}$$

and it describes how each of the two outputs (rows) change as we change any one of the inputs (columns). This leads us to the well-known definition of *the Jacobian matrix* in the category **Smooth** (Definition 2.2) which we recall has a monoidal closed subcategory $\mathbf{FVect}_{\mathbb{R}}$ (Definition 2.4) of linear maps.

Proposition 6.1 (compare [CCG+20], Example 5.3, Example 14.2)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a map*

in **Smooth**. *Then the* ***Jacobian*** *of f is a map of type*

$$J[f] : \mathbb{R}^n \to (\mathbb{R}^n \multimap \mathbb{R}^m)$$

defined as a matrix of partial derivatives at every point $x : \mathbb{R}^n$:

$$J[f](x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

The Jacobian tells us how at every point $x : \mathbb{R}^n$ a change in the input causes a change in the output. However, often we're interested in percolating this information backwards — computing how a change in the output affects the input. This can be done by transposing the Jacobian at x, obtaining $J[f](x)^\dagger : (\mathbb{R}^m \multimap \mathbb{R}^n)$. Not to be confused with computing the *inverse* of the Jacobian at x, which is something related to Netwon's method, and computationally more expensive.[4] The transposed Jacobian is often called "the reverse derivative", while the non-transposed version is called "the forward derivative", matching closely the intuition behind reverse-mode and forward-mode AD.

Notation 6.2. Every Jacobian $J[f] : X \to (X \multimap Y)$ and its transposed version — which we sometimes denote by $J[f]^\dagger : X \to (Y \multimap X)$ — can be additionally be uncurried, and treated as maps linear in their second argument. We introduce the following notation for this.[5]

$$D[f] : X \times X \to Y \qquad R[f] : X \times Y \to X$$
$$D[f](x, v) = J[f](x)(v) \qquad R[f](x, v) = J[f]^\dagger(x)(v)$$

6.2 Jacobians and weighted optics

Jacobians are an important concept in differential calculus, and will be the central component of this chapter. In fact, the direction where we are heading might have already become apparent once the type of $J[f]^\dagger$ and $R[f]$ were specified. These types precisely match the type of the backward

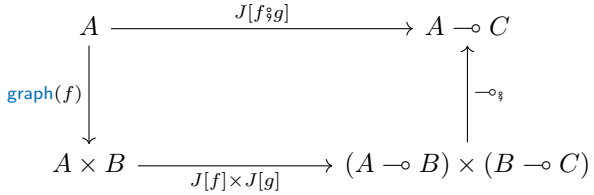
[4] This too forms a lens, and was discovered together with Mario Alvarez-Picallo.

[5] We also note that $R[\dots]$ notation is also used in Definition 2.3 for polynomial rings, but it will always be clear from the context and variable names which one is being used.

part of lenses, with f matching their forward one!

It might seem like a spurious coincidence, but it turns out that the similarities run deep. For instance, let's take a look at an important property of Jacobians that relates them to sequential composition of maps in **Smooth**: the chain rule.

Proposition 6.3 (Chain Rule)**.** *Let $f : A \to B$ and $g : B \to C$ be two maps in* **Smooth***. Then* ***the chain rule*** *is a property of Jacobians: it describes how Jacobians of f and g are related to the Jacobian of their sequential composite. It can be expressed as the commutativity of the diagram below:*



Equationally, it tells us that $J[f \fatsemi g](x) = J[f](x) \multimap_{\fatsemi} J[g](f(x))$.[6]

The chain rule bears a striking similarity to the composition rule for closed lenses Definition 4.52. As a matter of the fact, they are exactly the same!

Another similarity arises when we look at the way we assign Jacobians to smooth maps. Note that the Jacobian is an assignment that is made to *every* morphism in **Smooth**. This suggests that it might be a functor. Which functor is it, and what is its codomain? The problem is that the type of the Jacobian does not have a straightforward interpretation as a morphism in a category. But this is only a problem because we assumed the original f has to be forgotten. Instead, if we keep the forward map f and bundle it together with the Jacobian as the pair $(f, J[f]^\dagger)$, we can see that $f : A \to B$ and $J[f]^\dagger : A \to (B \multimap A)$ resemble the types of forward and backward maps in a closed lens. This is precisely what is done in automatic differentiation: the forward map f is never forgotten — instead it is paired with the Jacobian since it is needed to compute the input of $J[g]^\dagger : B \to (C \multimap B)$ when expressing the chain rule.

And lastly, we observe that in most implementations of automatic differentiation[7] the above formulation of chain rule *is not* the one explicitly used. This is because the definition of $J[f \fatsemi g]$ explicitly recomputes something that we can memoise: the output of f. Such memoisation will

[6] If we consider functions of type $\mathbb{R} \to \mathbb{R}$, then the composition of linear maps reduces to multiplication of their coefficients. This yields the familiar representation of chain rule $(f \fatsemi g)'(x) = g'(f(x)) \cdot f'(x)$, where the superscript $'$ here denotes the Jacobian, i.e. $f' = J[f]$.

[7] At least, the ones aiming to minimise redundant computation.

require not just bundling f and $J[f]$ together into a pair of maps, but treating them as a singular unit[8]: one which given an input $a : A$ it produces two things: the output $f(b) : B$ and also the linear map $J[f]^\dagger(a) : B \multimap A$. A careful reader might have noticed that this resembles the definition of a closed lens (Definition 4.49). And as we will see, a more efficient formulation of the chain rule is precisely the one that arises from the definition of closed lens composition (Eq. (4.55)).[9]

The above points motivate the usage of weighted optics as a framework for studying differentiation, as they give us not just a way of modelling differentiation denotationally, but also operationally. More precisely, a coherent way of assigning Jacobians to every morphism will be modelled a part of the data of the functor

$$\mathcal{C} \to \mathbf{Optic}_{\binom{\times}{\otimes}}^{\mathcal{C}(-,\iota(-))}$$

for a particular kind of a category $\mathcal{C}$: an *additively closed cartesian left-additive category*, a novel construction we define in this thesis.

Definition 6.4 (Additively closed cartesian left-additive category)**.** Let $\mathcal{C}$ be a cartesian left-additive category. We call it an **additively closed cartesian left-additive category (ACCLA)** if

- Its subcategory of additive maps $\mathbf{CMon}(\mathcal{C})$ has a monoidal product $(\mathbf{CMon}(\mathcal{C}), \otimes, I)$ with respect to which it is closed;
- The embedding $\iota : \mathbf{CMon}(\mathcal{C}) \to \mathcal{C}$ is lax monoidal with respect to the monoidal structure $(\otimes, I)$ of $\mathbf{CMon}(\mathcal{C})$ and $(\times, 1)$ of $\mathcal{C}$.

This is a construction that takes advantage of the fact that in all the settings of interest to us ($\mathbf{Smooth}$ and $\mathbf{Poly}_R$) additive and linear maps coincide.[10] This is why it can also reasonably be called a **linearly closed cartesian left-linear category**.[11] It enables us to compartmentalise many moving pieces of derivatives, as well as provide us with an operational perspective on their composition.

Remark 6.5. The above definition implies a few things. The category $\mathbf{CMon}(\mathcal{C})$ is always a cartesian left-additive subcategory of $\mathcal{C}$. The existence of lax monoidal structure of ι and self-enrichment of $\mathbf{CMon}(\mathcal{C})$ via the enriched base change (Definition A.21) implies enrichment of

[8] This is something we can do because their domains agree.
[9] I've first seen this observed in [Ell18a, Sec. 3.1], though the terminology of lenses was not used.
[10] In general, linear maps are special cases of additive maps. For an example where they do not coincide, see [BCS09, p. 541]. It is not clear to me what the implications of this counterexample are.
[11] Despite the unfortunate fact that LCCLA is a less punchy acronym.

$\mathbf{CMon}(\mathcal{C})$ in $\mathcal{C}$. Additionally, the functor ι is — in addition to being lax monoidal with respect to $\otimes$ — always strong monoidal with respect to the product structure of $\mathbf{CMon}(\mathcal{C})$.

In this kind of a category, we can exhibit a generalised kind of a tensor-hom adjunction which captures the idea of a property of a morphism being valid in only one variable (see Section 4.4.4 for more details and intuition).

Proposition 6.6. *Let $\mathcal{C}$ be an ACCLA. Then for every $X, Y', X' : \mathcal{C}$ the following isomorphism holds:*

$$\mathcal{C}(X, \iota(\underline{\mathbf{CMon}(\mathcal{C})}(Y', X'))) \cong \mathbf{CMon}(\mathbf{coKl}(X \times -))(Y', X')$$

This is a proof that follows by routine, but formally tedious calculation.

Example 6.7. The category $\mathbf{Smooth}$ is an additively closed cartesian left-additive category: its subcategory of additive maps $\mathbf{CMon}(\mathbf{Smooth})$ coincides with $\mathbf{FVect}_{\mathbb{R}}$ which is monoidal closed, and the embedding is a lax monoidal functor (Lemma 2.24).

Example 6.8. The category $\mathbf{Poly}_R$ is additively closed, as its subcategory of additive maps also coincides with $\mathbf{FVect}_R$ which is monoidal closed.

Now let's unpack the contents of the category of weighted optics $\mathbf{Optic}_{\binom{\times}{\otimes}}^{\mathcal{C}(-,\iota(-))}$. Its forward action is the self-action of the cartesian structure of $\mathcal{C}$, backward action is the self-action of the monoidal closed structure of $\mathbf{CMon}(\mathcal{C})$, and the weight is the composite

$$\mathcal{C}^{\mathsf{op}} \times \mathbf{CMon}(\mathcal{C}) \xrightarrow{\mathcal{C}^{\mathsf{op}} \times \iota} \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \xrightarrow{\mathsf{Hom}_{\mathcal{C}}} \mathbf{Set}$$

Concretely, a weighted optic $\binom{X}{X'} \to \binom{Y}{Y'}$ here consists of

- A coparameter $\binom{M}{M}$ where $M : \mathcal{C}$ and $M' : \mathbf{CMon}(\mathcal{C})$;
- A map $s : \mathcal{C}(M, \iota(M'))$;
- A forward map $f : X \to M \times Y$ in $\mathcal{C}$;
- A backward map $f' : M' \otimes Y' \to X'$ in $\mathbf{CMon}(\mathcal{C})$

quotiented out by diagrams in Eq. (4.16). Despite its modest presentation, this construction captures many of the things we care about in differentiation, including its operational characteristics.

The reductions described in Section 4.4.1 can be applied here, and we can unpack them explicitly by chooosing appropriate types of residuals. Set $M = X$ and $M' = \underline{\mathbf{CMon}(\mathcal{C})}(Y', X')$. This reduces the type of the underlying maps to $\mathcal{C}(X, X \times Y)$, $\mathcal{C}(X, \iota(\underline{\mathbf{CMon}(\mathcal{C})}(Y', X')))$ and $\mathbf{CMon}(\mathcal{C})(\underline{\mathbf{CMon}(\mathcal{C})}(Y', X') \otimes Y', X')$. These are exactly the types of $\mathsf{graph}(f)$, $J[f]$ and eval (Fig. 6.2)!

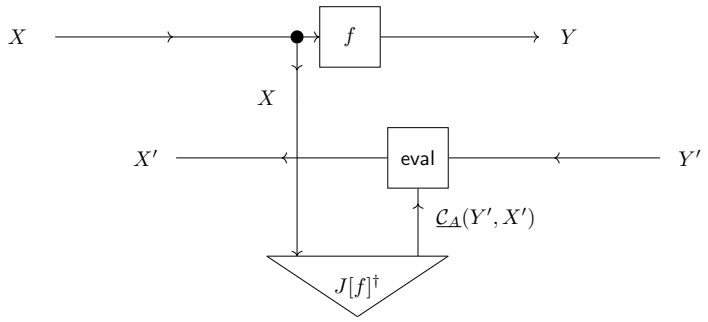


Figure 6.2: An optic implementing reverse mode AD.

What do these maps do? The forward pass computes the output of f while at the same time saving the input X to be used on the backward pass. Via the Jacobian this X is transformed into a linear map $Y' \multimap X'$ which is in conjuction with a Y' evaluated into an X'. This looks like an implementation of reverse mode AD, and we will prove in Section 6.3 that this is exactly the case.

Alternatively, we can abstract away the eval and look at a different concrete representative given by Corollary 4.56:

$$\mathbf{Optic}^{\mathcal{C}(-,\iota(-))}_{\binom{\times}{\otimes}}\left(\begin{matrix} X & Y \\ X' & Y' \end{matrix}\right) \cong \mathcal{C}(X, Y) \times \mathcal{C}(X, \iota(\underline{\mathbf{CMon}(\mathcal{C})}(Y', X'))) \tag{6.9}$$

yielding the types of f and $J[f]^\dagger$. By applying Prop. 6.6 we can provide another representation

$$\mathbf{Optic}^{\mathcal{C}(-,\iota(-))}_{\binom{\times}{\otimes}}\left(\begin{matrix} X & Y \\ X' & Y' \end{matrix}\right) \cong \mathcal{C}(X, Y) \times \mathbf{CMon}(\mathbf{coKl}(X \times -))(Y', X') \tag{6.10}$$

which has concrete morphisms of the form $f : \mathcal{C}(X, Y)$ and $f' : \mathcal{C}(X \times Y', X')$ with the additional property that f' is additive in the second component (Definition 2.22). This is precisely the uncurrying of $J[f]^\dagger$ described in Notation 6.2. Similarities do not stop here. While the lens repre-

sentation defines the implementation of backpropagation that performs gradient checkpointing[12], it can be shown that their composition in the optic representation computes it *without* gradient checkpointing. This suggest that optics do not merely describe reverse mode AD, but give us a prescriptive recipe for implementing it.[13]

In general, additivity as we have defined it is crucial, as it enables optics to have products.

Proposition 6.11. *The category* $\mathbf{Optic}_{\binom{\times}{\otimes}}^{\mathcal{C}(-,\iota(-))}$ *has products defined as*

$$\begin{pmatrix} A \\ A' \end{pmatrix} \times \begin{pmatrix} B \\ B' \end{pmatrix} = \begin{pmatrix} A \times B \\ A' \oplus B' \end{pmatrix}$$

where $\oplus$ *is here suggestive notation for the biproduct structure of* $\mathbf{CMon}(\mathcal{C})$ *(Prop. 2.21).*

Proof. We prove the universal property of the product (Eq. (2.16)).

$$\begin{aligned}
& \mathbf{Optic}_{\binom{\times}{\otimes}}^{\mathcal{C}(-,\iota(-))}\begin{pmatrix} X & A \\ X' & A' \end{pmatrix} \times \mathbf{Optic}_{\binom{\times}{\otimes}}^{\mathcal{C}(-,\iota(-))}\begin{pmatrix} X & B \\ X' & B' \end{pmatrix} \\
\text{(Def.)} \quad & = \mathcal{C}(X,A) \times \mathcal{C}(X, \iota(\underline{\mathbf{CMon}(\mathcal{C})}(A',X'))) \times \mathcal{C}(X,B) \times \mathcal{C}(X, \iota(\underline{\mathbf{CMon}(\mathcal{C})}(B',X'))) \\
\text{(Univ. prop. of product)} \quad & \cong \mathcal{C}(X, A \times B) \times \mathcal{C}(X, \iota(\underline{\mathbf{CMon}(\mathcal{C})}(A',X')) \times \iota(\underline{\mathbf{CMon}(\mathcal{C})}(B',X'))) \\
(\iota \text{ is strong monoidal}) \quad & \cong \mathcal{C}(X, A \times B) \times \mathcal{C}(X, \iota(\underline{\mathbf{CMon}(\mathcal{C})}(A',X') \oplus \underline{\mathbf{CMon}(\mathcal{C})}(B',X'))) \\
\text{(Univ. prop. of coproduct)} \quad & \cong \mathcal{C}(X, A \times B) \times \mathcal{C}(X, \iota(\underline{\mathbf{CMon}(\mathcal{C})}(A' \oplus B',X'))) \\
\text{(Def.)} \quad & = \mathbf{Optic}_{\binom{\times}{\otimes}}^{\mathcal{C}(-,\iota(-))}\begin{pmatrix} X & A \times B \\ X' & A' \oplus B' \end{pmatrix}
\end{aligned}$$

□

These similarities between derivatives and optics lead us to stating the following conjecture:

Weighted optics provide good denotational and operational semantics for differentiation.

We say "conjecture" because this is not something we fully prove in this thesis. This is because we deal exlusively with a specific representation of optics that prohibits modelling non-checkpointed automatic differentiation. In other words, to *formally prove*, given the time constraints in writing this thesis, a correspondence between differentiation and optics, we use the concrete representation

[12] Gradient checkpointing being a method of computing derivatives mentioned in the introduction of this chapter.
[13] See [Gav22b] for more information.

of optics in Eq. (6.10) as lenses with backward pass *additive in the second component*. This means that, instead of using optic composition and identities, we restricted to use the defined lens composition and identities in Definitions 4.37 and 4.52. The reason we do this is because this allows us to reduce the underlying structure to just a cartesian left-additive category (note that the right-hand side of Eq. (6.10) does not use any closed structure of $\mathbf{CMon}(\mathcal{C})$), and thus greatly simplify the proofs in the rest of this chapter.[14]

Despite its reduced expresiveness, this is still a fully formal statement which can be made more general in a systematic way. We introduce the shorthand notation for this construction:

$$\mathbf{Lens}_A(\mathcal{C}) \coloneqq \mathbf{Optic}_{\binom{\times}{\otimes}}^{\mathcal{C}(-,\iota(-))} \tag{6.12}$$

In light of this, the formal statement of the main theorems of this chapter is:

The construction $\mathbf{Lens}_A$ is a functor (Theorem 6.13), and generalised reverse differential structure is its coalgebra (Definition 6.19).

As we will see in Theorem 6.21, this compact definition of a coalgebra will specify concepts such as the chain rule, additivity of the backward pass, behaviour with respect to products and summing, and so on.

6.3 Backpropagation as a $\mathbf{Lens}_A$-coalgebra

We first aim to prove the following.

Theorem 6.13. *Lenses with backward passes additive in the second component form a functor*

$$\mathbf{Lens}_A : \mathbf{CLACat} \to \mathbf{CLACat}$$

In order to do that, we need to precisely specify the category of cartesian left-additive categores.

Definition 6.14. We call $\mathbf{CLACat}$ the category whose objects are cartesian left-additive categories and whose morphisms are cartesian left-additive functors (functors which preserve products and commutative monoid structure on objects ([BCS09, Definition 1.3.1])).

[14] This is a peculiar thing, because it implies that non-checkpointed reverse mode AD necessitates the equipment of $\mathcal{C}$ with an additively closed subcategory. This is something we hope to explore in future work.

Now we can prove $\mathbf{Lens}_A$ is a functor. Recall that it takes in a cartesian left-additive category $\mathcal{C}$ and produces the category $\mathbf{Lens}_A(\mathcal{C})$. We have shown it is cartesian, and we now proceed to show it is left-additive as well.

Proposition 6.15. *The category $\mathbf{Lens}_A(\mathcal{C})$ is cartesian left-additive.*

Proof. We need to equip $\mathbf{Lens}_A(\mathcal{C})$ with a commutative monoid on every object in a way that's compatible with the cartesian structure.[15] That is, for every object $\binom{A}{A'}$ we need to provide two morphisms:

- **Unit** $0_{\binom{A}{A'}} : \binom{1}{1} \to \binom{A}{A'}$. This is a lens whose forward map we set as the zero map 0_A and the backward map as the delete $!_{1 \times A'}$.
- **Multiplication** $+_{\binom{A}{A'}} : \binom{A \times A}{A' \times A'} \to \binom{A}{A'}$. This is a lens whose forward map we set as sum $+_A$ and the backward map as copy, i.e. $\pi_2 \fatsemi \Delta_{A'}$.

Additionally, these morphisms need to obey the monoid laws. This can be verified by routine. □

This defines the action of $\mathbf{Lens}_A$ on objects of $\mathbf{CLACat}$. Action on morphisms is defined below.

Proposition 6.16. *Let $F : \mathcal{C} \to \mathcal{D}$ be a cartesian left-additive functor. This induces a cartesian left-additive functor $\mathbf{Lens}_A(F)$ between the corresponding categories of lenses:*

$$\mathbf{Lens}_A(\mathcal{C}) \xrightarrow{\mathbf{Lens}_A(F)} \mathbf{Lens}_A(\mathcal{D})$$

$$\begin{array}{ccc} \binom{A}{A'} & \longmapsto & \binom{F(A)}{F(A')} \\ \downarrow{\scriptstyle \binom{f}{f'}} & & \downarrow{\scriptstyle \binom{F(f)}{\overline{f'}}} \\ \binom{B}{B'} & \longmapsto & \binom{F(B)}{F(B')} \end{array} \tag{6.17}$$

where $\overline{f'} \coloneqq F(A') \times F(B') \cong F(A' \times B') \xrightarrow{F(f')} F(A')$.

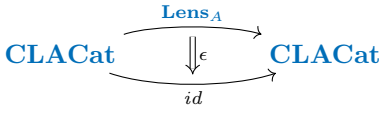
Proof. The above proposition defines the action on objects and morphisms. What remains to prove is that identities and composition is preserved, as well as that this functor is cartesian left-additive. We defer this to the Appendix (Appendix D). □

[15] We don't need to show that this monoid is unique, just that it exists and can be canonically defined.

What remains to show is that $\mathbf{Lens}_A$ preserves identities and composition, which follows routinely, concluding the proof of Theorem 6.13.

This functor has additional structure — it is copointed.[16]

Proposition 6.18 (Copointed structure of $\mathbf{Lens}_A$)**.** *There is a natural transformation*

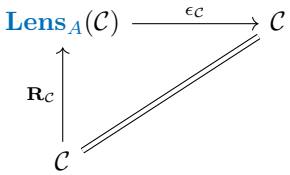


assigning to cartesian left-additive category $\mathcal{C}$ a cartesian left-additive functor $\epsilon_{\mathcal{C}} : \mathbf{Lens}_A(\mathcal{C}) \to \mathcal{C}$ which forgets the backward part of the lens.

This brings us to the main definition. Its name is inspired by the existing categorical framework for differentiation called *reverse derivative categories* (RDCs) ([CCG+20]). We remark more on it in Theorem 6.21, Remark 6.22 and the literature review (Section 6.4).

Definition 6.19. A **generalised cartesian reverse derivative category** is a coalgebra of the copointed $\mathbf{Lens}_A$ endofunctor.

This definition packs a lot of punch. Explicitly, it consists of a choice of a cartesian left-additive category $\mathcal{C}$ and a cartesian left-additive functor $\mathbf{R}_{\mathcal{C}} : \mathcal{C} \to \mathbf{Lens}_A(\mathcal{C})$ such that the following diagram commutes



(6.20)

A good way to understand it is through the aforementioned framework of reverse derivative categories, for whose axioms it serves as a justification.

Theorem 6.21. *Every cartesian reverse derivative category (Definition D.21) is a* generalised *cartesian reverse derivative category.*

Proof. As we're already starting with a cartesian left-additive category $\mathcal{C}$, what remains to prove is that there is reverse derivative combinator ([CCG+20, Definition 13]) for which the first five

[16] Despite this the functor $\mathbf{Lens}_A$ does not have the comonad structure, for similar reasons that tangent categories do not.

axioms of RDC's are satisfied. The reverse derivative combinator is precisely the data of the functor $\mathbf{R}_{\mathcal{C}} : \mathcal{C} \to \mathbf{Lens}_A(\mathcal{C})$, where Eq. (6.20) necessitates that the only choice involved in this functor is one of the backward pass. When it comes to the axioms, we show how they arise one by one:

1. **Additivity of reverse differentiation.** This is recovered by $\mathbf{R}_{\mathcal{C}}$ preserving left-additive structure.

2. **Additivity of reverse derivative in the second component.** This is recovered by definition of $\mathbf{Lens}_A$ — the backward maps are additive in the 2^{nd} component.

3. **Coherence with identities and projections.** Coherence with identities is recovered by preservation of identities of the functor $\mathbf{R}_{\mathcal{C}}$, where for every $X : \mathcal{C}$, $\mathbf{R}_{\mathcal{C}}(\mathrm{id}_X) = \mathrm{id}_{\mathbf{R}_{\mathcal{C}}(X)} = (\mathrm{id}_X, \pi_2 : X \times X \to X)$. Coherence with projections is recovered by $\mathbf{R}_{\mathcal{C}}$ preserving cartesian structure.

4. **Coherence with pairings.** Recovered by $\mathbf{R}_{\mathcal{C}}$ preserving cartesian structure.

5. **Reverse chain rule.** This is recovered by functoriality of $\mathbf{R}_{\mathcal{C}}$.

□

Remark 6.22. A cartesian reverse derivative category has two additional axioms — the 6th and 7th axiom — which are independent from the others, and not captured with our coalgebraic construction [CC14, Sec. 2.3]. As we will see in the rest of this thesis, we will not need these two axioms to compositionally model supervised learning.[17]

Example 6.23 (**Smooth**, compare [CCG+20, Ex. 14.2])**.** The cartesian left-additive category **Smooth** is an example of a generalised cartesian reverse derivative category, with $\mathbf{R}_{\mathcal{C}}(f) \coloneqq \begin{pmatrix} f \\ R[f] \end{pmatrix}$.

Example 6.24 (**Poly**$_R$, compare [CCG+20, Ex. 14.1])**.** The cartesian left-additive category **Poly**$_R$ is an example of a generalised cartesian reverse derivative category, with $\mathbf{R}_{\mathcal{C}}(f) \coloneqq \begin{pmatrix} f \\ R[f] \end{pmatrix}$.

[17] Though they might become important in future work where we hope to prove properties of these learning systems.

What does it mean to backpropagate bits? **Box 6.3.1**

In categories like **Smooth** or $\mathbf{Poly}_{\mathbb{R}}$, given a map $f : X \to Y$ the input $y' : Y$ to its reverse derivative $R[f](x) : Y' \to X'$ at a point $x : X$ has a straightforward interpretation: we think of it as a vector which denotes the direction of steepest descent at $f(x)$.[a] The map $R[f](x)$ then backpropagates this information to X', computing the direction of steepest descent at x. But how does this work in $\mathbf{Poly}_{\mathbb{Z}_2}$? That is, given some polynomial $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$, what is the "steepest-descent" interpretation of the input of $R[f][x] : \mathbb{Z}_Z^m \to \mathbb{Z}_2^n$ for some input x? The answer is as follows.

Here the gradient vector $y' : \mathbb{Z}_2^m$ is *bit-valued*, and the information it communicates is whether flipping each one of bits $f(x)_i$ would have yielded a better outcome. The map $R[f](x)$ backpropagates this information to $\mathbb{Z}_2^n$, computing whether each of the bits x_i should be flipped.

[a] Even more preciely, it consumes a *covector*. See Section 6.4.

Example 6.25 (coKleisli category (compare [GV22, Theorem 2]))**.** If $\mathcal{C}$ is a generalised cartesian reverse derivative category, then $\mathbf{coKl}(A \times -)$ is too, for every $A : \mathcal{C}$.

This concludes the main aspect of this section — showing us how to categorically model differentiation using only the framework of lenses. We will see later how this functor will prove important for differentiating parametric morphisms in a compositional way.

6.3.1 Operational concerns

In this short subsection we mention the distinction between denotational and operational semantics of optics. Denotational ones we have already described — they are the ones arising our the 1-categorical framework. In this setting cartesian optics are isomorphic to lenses (Eq. (4.38)). This allows us to move between their representatives as we wish, as they all compute the same output. However, implementing these in code necessitates a choice of representation. And different choices induce programs with different performance profiles.

This is one of the issues of implementing the chain rule naively: thinking of it as a dynamic programming task, it is one with *overlapping subproblems*. In other words, with sufficient memory, computing the derivative of a composite of functions can take advantage of already computed solutions. There is a systematic way to interpret optics as a bicategory where instead of computing

a quotient and obtaining hom-sets we instead think of the hom-objects in their natural form as categories. In such a setting lenses embody a *different* composition rule than optics, one with a different *space-time tradeoff*. This tradeoff precisely captures distinction between code that is gradient checkpointed versus one that is not. We hypothesise that such a 2-categorical framework can provide formal grounds for informing various rewrite rules and compiler optimisations. See [Gav22b] for more details.

6.4 Backpropagation via category theory in the literature

Differentiation comes in many shapes and forms, and so do its categorical frameworks.

One of the biggest clusters is centered around differential categories (CDCs) ([BCS09]), a construction admitting numerous generalisations and extensions ([CCG+20, CS11, CGLP22, CC14]). We note that the second reference from this list formulates CDCs as a coalgebra of a particular functor, much like we do in this thesis. Unlike us, they capture all 7 axioms of CDC! Contrary to us, they do not establish any connections reverse derivatives, nor the optical literature, something we deem is crucial if one wants to obtain a refined operational specification of these constructions, and generalise them to a wide array of bidirectional processes, from bayesian learning to value iteration. Reverse derivative categories have also been generalised to the purely monoidal setting in [CGLP22]. We conjecture this construction fits into the general story with monoidal weighted optics, as optics do not assume any cartesianness. We also believe the optical framework in general serves as a powerful justification the axioms of reverse derivative categories. It removes the need for a monolithic list of axioms, instead compartmentalising them into more conceptual pieces (see the proof of Theorem 6.21).

Differential categories are examples of tangent categories ([CC14, Gar18]), a framework that additionally captures manifolds and various other dependently-typed constructs, outside of reach of differential categories which are non-dependent. Tangent categories spawned a lot of interesting research, especially related to exponentials and curve objects ([CCL21]), concepts useful for studying differential equations in tangent categories. We conjecture an analogous characterisation of tangent categories can be done in the language of dependent lenses, where $\mathbf{Lens}_A$ would be replaced by $\mathbf{DLens}_A$, the category of dependent lenses additive in the second component of their backward pass. Understanding operational aspects of these dependent lenses through their optical characterisation is part of ongoing research on dependent optics (Section 4.5.1).

On the other hand, there is a lot of work in many ways disjoint from cartesian differential and tangent categories that studies more practical aspects of differentiation. As briefly mentioned in the introduction, the work of ([APGSZ21]) proves the soundness of the original description of AD ([PS08]) using the semantic category of reverse derivative categories. It's worthy to mention that they also use a graphical language, but not the one arising from the graphical language of lenses and optics themselves. Unifying these two graphical languages is something we hope to explore in the future. The work [Ell18a] presents an elegant, purely functional formulation of both forward and reverse mode automatic differentiation that is homomorphic with respect to a collection of standard categorical abstractions — like those described in Section 6.3, but applicable to a language with only limited features, such as first-order functions. Combinatory Homomorphic Automatic Differentiation (CHAD) ([VS22], and also its predecessor [Vá21]) takes this work further and introduced a compositional, type-respecting method that performs source code translation and generates purely functional code, extending it to languages with higher-order functions. We believe this bears a close resemblance to the framework presented here, especially with respect to usage of the linear subcategories of smooth maps. We note that like us, they also do not distinguish between linear and additive maps. While they provide *a* categorical framework, they do not axiomatise it, or show how it is related to existing axiomatic frameworks such as tangent categories.

A more precise way to characterise backpropagation of gradient vectors is by framing it as the pullback of gradient *covectors*. Covectors of some vector space V are vectors in the dual vector space of linear functionals, i.e. *valuations* on vectors, represented as linear maps into the base field. This appears amenable for capturing with the putative folkoric idea of *reverse categories*, yet to be defined.[18] Fragments of this idea can already be seen in [Ell18a] where it is noted that backpropagation can be studied as the image of the representable functor $\mathbf{FVect}_{\mathbb{R}}(-,\mathbb{R}) : \mathbf{FVect}_{\mathbb{R}}{}^{\mathsf{op}} \to \mathbf{FVect}_{\mathbb{R}}$ which flips the direction from forward-mode AD to backward-mode AD. Despite this, the underlying covectors as still treated as mere vectors as a result of the isomorphism $V \cong \underline{\mathbf{FVect}_{\mathbb{R}}}(V,\mathbb{R})$ in the finite-dimensional case. Covectors are more explicitly acknowledged in [DL19]. There the author formulates reverse-mode AD as a functor from trivialisable diffeological spaces ([Lau06]) into optics whose objects are of the form $\left(\underline{\mathbf{FVect}_{\mathbb{R}}}^{X}(X',\mathbb{R})\right)$. This is a functor that is symmetric monoidal, and the author conjectures it can be made lax, additionally capturing counterfactual reasoning in game theory. This is subsequently built upon by [Cap23] where it is shown that gradient-based learning can be thought of as "infinitesimal counterfactual reasoning"

[18] They have been defined after this thesis was submitted for examination; see [CL23].

in a way that almost completely overlaps with feedback propagation in open games [GHWZ18].[19] This suggests that a unified framework of a general notion of differentiation can capture a variety of settings, including even those of bayesian reasoners.

Interesting work has been done related to formalising differentiation in a setting of incremental computation under the name of *change actions* [AP20]. For completeness we also mention that automatic differentiation has been characterised also as a fold [NW22]. And lastly, the $\mathbf{Lens}_A$-coalgebra framework can be seen as an example of a dynamical systems doctrine ([Mye22a]), establishing connections to general systems theories.

[19] On the other hand, this assumption completely breaks down compositionality of players! See [Cap23, Remark 2.1]. Understanding the implications of this fact is something we hope to explore in the future.

Chapter 7

Backprop through Structure

> When the limestone of imperative programming has worn away, the granite of functional programming will be revealed underneath.
>
> Simon Peyton Jones

Throughout the last few decades the number of neural network architectures — structured ways of constructing their forward passes — has proliferated. Today neural networks can be "feedforward", "recurrent" ([Sch19]), "convolutional" ([BL95]), "residual" ([HZRS16]), "graph" ([WPC+21]), "topological" ([PSHM23]), "generative-adversarial" ([GPAM+14]), "autoregressive" ([GDM+14]), "autoencoding" ([BKG21]), and even hybrid, i.e. composed out of more than one type of architecture at the same time ([RBL+22, RDN+22, Ope22]). The number of architectures is increasing, and many of them are becoming respective subfields of deep learning with sizeable communities of researchers.

Despite this explosive growth, this zoo of architectures is tied together by very few unifying principles. Most of these architectures come with their own theoretical underpinning, best practices, evaluation measures and often different ways of thinking about the field as a whole. They require different types of underlying datasets, possess different expressive power, and exhibit a wide range of inductive biases. Although many of these architectures are special cases of each other,

the current fast-paced nature of deep learning research made it hard to translate results and theorems across them. This is problematic for both beginners trying to learn the sheer volume of the underlying concepts, but also for experts trying to get a sense of the research landscape. Many ideas repeat across subfields, and it is becoming easier than ever to reinvent concepts. Even if some individual components of neural network architectures are well-understood, their combinations and composition are not.

Lastly, all neural networks architectures are eventually implemented as programs. But despite this, the research on construction and design of architectures has been largely disjoint from the research on construction and design of programs. The advances from type theory and functional programming such as dependent ([McK06]) and quantitative ([Atk18]) types, constructive reasoning and type-driven development ([Bra16]) have generally not percolated to mainstream deep learning. While we think about and denote programs using a rich and expressive typed language, when it comes to neural networks — despite the advent of architectures which deal with structure — the main mode of reasoning still involves thinking about arrays of numbers. One consequence of this is that there is no type theory for neural network architectures. Architectures are instead specified using a mixture of mathematical notation and natural language, often using ambiguous notation ([CRB21]), and no clear specification of underlying types. Unlike with typed programs — which can be verified to be well-typed before running them — there are no such guarantees when it comes to architectures. There is no formal way to verify an implementation of architecture is correct — because there are no formal specifications of most architectures.

The deep learning community is aware of many of these issues. Some researchers have begun to organise workshops focused on compositionality of deep learning components ([THJ+19, MM22]). Others have called for a deep learning "Langlands programme" ([Rie20]) or a deep learning "Erlangen programme" ([BBCV21]), both inspired by far-reaching mathematical programs aimed at unifying seemingly disparate parts of mathematics, A completely different approach was suggested by [Ola15], emphasising the relationship between many kinds of neural network architectures and concepts in functional programming such as folds, unfolds, and recursion schemes.

This resulted in some progress, most notably *Geometric Deep Learning* ([BBCV21]), an approach focusing on geometric structures of neural networks and symmetries within, using graph neural networks as a foundation. Another one is *Topological Deep Learning* ([PSHM23]), an approach focused on generalising the pairwise relational structures of graphs to those of higher arity. Geometric and topological deep learning made tremendous progress in unifying neural networks

based on graphs, and represent the state of the art in this regard. But they answer only some of the concerns above. Their thorough reliance on the algebraic structure of *groups* comes built in with an assumption of invertibility which is an assumption that does not hold in general for programs.

When it comes to unification between the way we think about neural networks and the way we think about programs — in terms of types, data structures, and often recursion — the search for the underlying granite foundation of deep learning architectures is still in its infancy.

Acknowledged in [BBCV21] and [Ola15], category theory is a natural continuation of these efforts. As seen throughout this thesis, it is a *language of structure* that has already described various phenomena throughout the sciences. As we will see in the rest of this chapter, in recent years category theory has started to be used to describe neural network architectures.[1]

Category theory is also foundation for many structural components of programming languages we use to *implement* the neural networks we are trying to understand. Functional features such as *reduce* in Python is an example of a concept of a *fold*, i.e. of a catamorphism in category theory, while others, such as *map* can often be seen as examples of actions of functors on morphisms. This aligns with our goal to be *operationally aware*, and gives credence to hypotheses in [Ola15], suggesting implementation of many architectures in practice can potentially be conceptually simplified.

Nonetheless, in this chapter we provide a comprehensive survey of existing developments, and provide a useful vantage point: we recast all of them through the unified framework of parametric lenses.

document the existing developments, define a number of architectures, and provide a vantage point that we believe will be useful in the long-term. In such a fast paced field, aim this to be a foundation that will not be outdated in a few years, but will continue to provide a stable base upon which we can constrwe hope to write content that will not be outdated

As opposed to doing a literature review at the end of this chapter, we opt for discussing avenues for explorations at the end of each section.

Contributions. *The novel research in this chapter consists of the formalisation of a) weight tying (Section 7.1.3), b) graph convolutional neural networks (Definition 7.14), c) encoding and decoding neural networks (Fig. 7.14), and d) generative adversarial networks (Definition 7.15), e) and e) Transformers (Section 7.6). Loss functions (Section 7.7) are also formalised, though the nov-*

[1] Despite the increase in research, when compared for the size of the field of deep learning the total volume is negligible.

elty here boils down to an assignment of types to their inputs and outputs. Other than concrete definitions, the novel contribution of this chapter is a survey and a unified framing of numerous category-theoretic papers on neural network architectures scattered around the literature.

Epistemic status. *This chapter is the one I am the most confident in when it comes to utility of category theory and its constructions. This is because there currently exists a schism between the theory of architectures of deep learning and the classical theory of computer science, and almost no work combining algebraic constructs in traditional computer science theory such as automata, (co)recursion or dynamic programming to the fuzzy, differentiable landscape of deep learning. Many of the contents of this chapter (and especially Section 7.3) will be useful for defining a generalised theory of equivariance that will allow us to provide formal guarantees of the behaviour of neural networks with respect to arbitrary algebraic operations, and in general many algorithms found throughout computer science. I envision this in turn having a downstream effect on regulation (see Chapter 9), and therefore issues of AI safety and bias.*

7.1 Layers and how to compose them

Before describing some of the well-known neural network layers, we start humbly with some of the simplest ones: those without any parameters.

7.1.1 Layers with no parameters

The layers without parameters described in this section are pervasively used throughout practice and applications. At the same time, they mostly go unnoticed and unacknowledged in any formal descriptions. These are various gadgets that copy, sum, delete or create information. We will see throughout the rest of this chapter how they are central components of weight tying (Definition 7.11), or generative adversarial networks (Definition 7.15), for instance. We start by acknowledging that most of these have one additional property: they have trivial residuals, i.e. they are adapters (7.1).

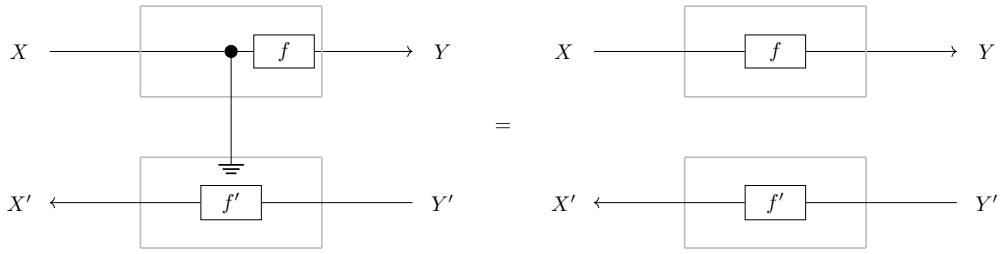


Figure 7.1: The backward part (the gray box at the bottom) of many lenses can be factored through the delete map. This means that each of these has an equivalent optic representation whose residual is the terminal object 1. In turn, this means that these lenses are adapters (Example 4.19).

Example 7.1 (Identity)**.** There is a trivial neural network that doesn't change the input: it is the identity map $\mathrm{id}_X : X \to X$. Its reverse derivative is the projection $R[\mathrm{id}_X](x, \alpha) = \alpha$.

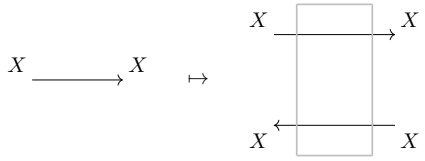


Figure 7.2: Identity and its derivative.

Example 7.2 (Copy)**.** We can copy information via the comonoid $\Delta_X : X \to X \times X$. Its reverse derivative is $R[\Delta_X](x, (\alpha, \beta)) = \alpha + \beta$.

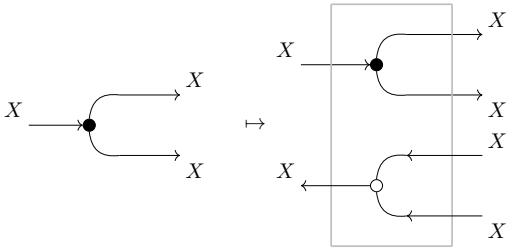


Figure 7.3: Copy and its derivative.

Example 7.3 (Sum)**.** We can sum information using the monoid structure $+_X : X \times X \to X$. Its reverse derivative is copy $R[+_X](x, \alpha) = (\alpha, \alpha)$.

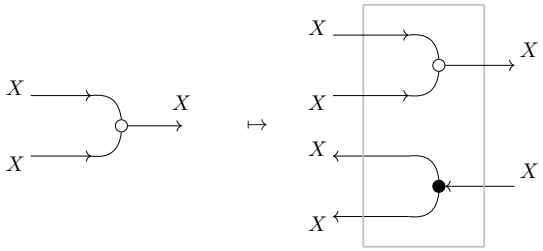


Figure 7.4: Sum and its derivative.

Example 7.4 (Delete)**.** The morphism $!_X : X \to 1$ deletes all information. Its reverse derivative is the zero gradient map, i.e. $R[!_X](x, \alpha) = 0$.

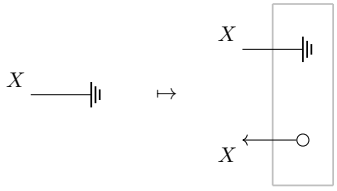


Figure 7.5: Delete and its derivative.

Example 7.5 (State)**.** Given any object X, all morphisms $s : 1 \to X$ have the same reverse derivative map $R[s] = !_{1 \times X}$.

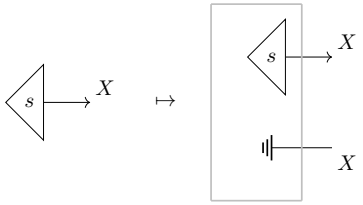


Figure 7.6: State and its derivative.

Lastly, we mention one such lens which is not an adapter: a lens which performs multiplication on the forward, and a derivative thereof on the backward pass.

Example 7.6 (Multiplication)**.** The reverse derivative of the multiplication map[a] $(x, y) \mapsto xy$ is the map: $R[\cdot]((x, y), \alpha) = (\alpha y, \alpha x)$.

[a] Formalisable in any cartesian distributive category ([WZ22, Def. 4]).

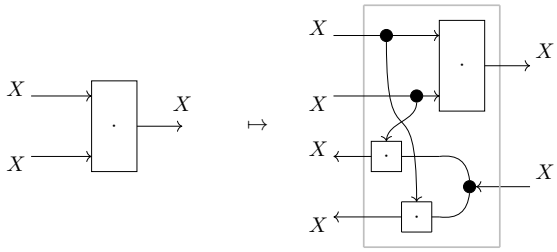


Figure 7.7: Multiplication and its derivative.

7.1.2 Feedforward neural networks

Feedforward neural networks are one of the central building blocks of deep learning. Such networks often consist of a sequence of alternating linear and non-linear layers. Below we focus on **Smooth**, but many of these layers can be defined in more general settings.

Definition 7.7 (Linear layer)**.** Fix input and output types $x, y : \mathbb{N}$. Then a linear layer $x \to y$ is

a morphism in **Para**(**Smooth**)($\mathbb{R}^x, \mathbb{R}^y$) whose parameter space is $\mathbb{R}^{x\times y}$ and the implementation is

$$\mathsf{eval} : \mathbb{R}^x \times \mathbb{R}^{x\times y} \to \mathbb{R}^y$$
$$\mathsf{eval}(X, W) \mapsto W^\top X$$

It is called eval is because it evaluates a linear map (represented as a matrix W) at a point X (represented as a vector).[2] Its reverse derivative is $R[\mathsf{eval}](X, W, Y') = (Y'W, Y'^\top X)$

In practice, the term linear layer is often used interchangeably with an *affine* layer, which is a composition of a linear layer and a bias layer defined below.

Definition 7.8 (Bias layer)**.** Given a number of neurons $y : \mathbb{N}$, the bias layer is the endomorphism in **Para**(**Smooth**)($\mathbb{R}^y, \mathbb{R}^y$) whose parameter space is $\mathbb{R}^y$, and implementation is pointwise addition:

$$+ : \mathbb{R}^y \times \mathbb{R}^y \to \mathbb{R}^y$$
$$+(Y, B) \mapsto Y + B$$

Its reverse derivative is given by the reverse derivative of $+$, i.e. copy: $R[+](Y, B, \alpha) = (\alpha, \alpha)$.[3]

Example 7.9 (Activation function)**.** An activation function is usually a function $a : \mathbb{R} \to \mathbb{R}$ in **Smooth** applied pointwise (i.e. a function $a^n : \mathbb{R}^n \to \mathbb{R}^n$) Notable examples are below, and graphed in Fig. 7.8.

$\mathrm{id}(x) = x$	Identity
$\sigma(x) = \dfrac{\exp(x)}{\exp(x)+1}$	Sigmoid (or logistic) function
$\tanh(x) = \dfrac{e^{\exp(2x)}-1}{e^{\exp(2x)}+1}$	Hyperbolic tangent
$\mathsf{ReLU}(x) = \max(0, x)$	Rectified linear unit ([Fuk75])
$\mathsf{LeakyReLU}(x) = \max(\alpha x, x)$	"Leaky" ReLU ([Maa13])[4]
$\mathsf{GELU}(x) = x\sigma(1.702x)$	Gaussian Error Linear Unit ([HG20])

[2] In category theory parlance, it is the counit of the tensor-hom adjunction in the subcategory $\mathbf{FVect}_{\mathbb{R}}$ of **Smooth**.
[3] Note the similarities with Fig. 7.4. In the bias layer, one of the summands lives on the vertical axis.

Though, not all examples are like this. The commonly used Softargmax function[5] is of type $\mathbb{R}^n \to \mathbb{R}^n$, and can't be factored as a product of n functions $\mathbb{R} \to \mathbb{R}$. For every output $i : N$ it's defined as a function of type $\mathbb{R}^n \to \mathbb{R}$ whose implementation is

$$\mathsf{Softargmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

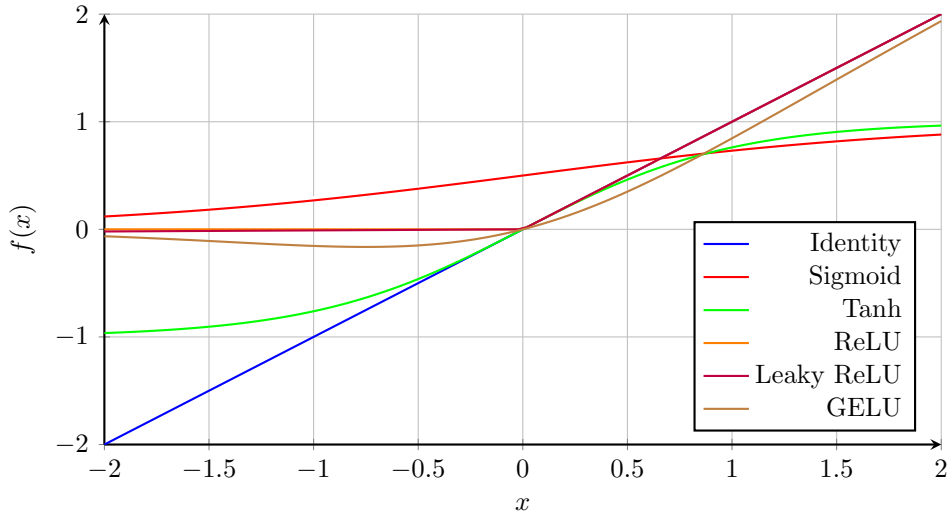


Figure 7.8: Graphs of activation functions: Identity, Sigmoid, Tanh, ReLU, Leaky ReLU, and GELU.

Remark 7.10. Softargmax is a particularly interesting function, having numerous uses in probability theory, statistical mechanics and reinforcement learning. In the case when $n = 2$, Softargmax recovers the logistic function. Since its output is always a vector whose entires sum up to 1 it can also be interpreted as a probability distribution. This suggests probability monads as a useful abstraction in providing more refined characterisation of this map.

Given all the components defined above, a standard "fully-connected" layer usually consists of a sequential composition of a linear layer, a bias layer and an activation function. (Fig. 7.9)

[5] Commonly known as "softmax" though this is a misleading name. See [GBC16, Sec. 6.2.2.3]

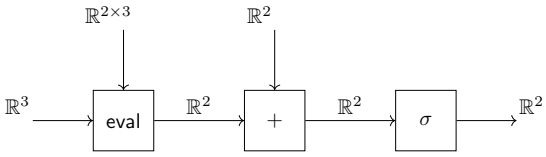


Figure 7.9: String diagram representation of a linear layer with some activation function σ.

As we will see in the next section, most of the interesting neural networks are composed out of something else than just a sequence of standard layers.

7.1.3 Weight tying

The concept of *encapsulation* is one of the most fundamental aspects of computer science. It allows us to abstract away the details of a system, and reuse functionality by simply calling the appropriate method. This reduces the amount of bugs and allows us to write more understandable and composable code. Encapsulation is in neural networks achieved by *weight tying*, the method by which we use multiple copies of a neuron in the same place. We thus only have to learn a functionality once, and can reuse it in different places. This consequently makes learning easier, as the search space is reduced, and can be seen as the core idea behind the recent success of deep learning.[6]

While most often considered in **Smooth**, the essential idea behind weight tying works in any cartesian category.

Definition 7.11 (Weight tying)**.** Fix a cartesian category $\mathcal{C}$.[7] Given any two objects X, Y and a parametric map $f : \mathbf{Para}(\mathcal{C})(X, Y)$ whose parameter space is $P \times P$ (for any $P : \mathcal{C}$) the **weight tying** of f is the parametric morphism

$$(P, f^{\Delta_P}) : \mathbf{Para}(\mathcal{C})(X, Y) \quad \text{where} \quad f^{\Delta_P} = \boxed{X \times P \xrightarrow{X\times\Delta_P} X \times (P \times P) \xrightarrow{f} X}$$

obtained by reparameterising f with the copy map $\Delta_P : P \to P \times P$. (the notation f^{Δ_P} follows the convention outlined in Fig. 3.2.)[8]

[6] In many ways, weight tying is a neural network equivalent of higher-order functions, and is something we hope to explore in future work.

[7] This statement can be generalised to any distributive algebroidal actegory ([CG23, Def. 5.2.4]).

[8] A galaxy-brain definition of weight tying is that it is precisely *the functor* described in Lemma 3.31.

This seemingly simple definition (whose string diagram representation is shown in Fig. 7.10) we believe is the cornerstone of modern deep learning. It takes a parametric morphism with two independent parameters and couples them together, removing a degree of freedom. This makes the "neuron" that the parameter is implementing be used in two different places, and reduces the search space as whatever is learned can now be reused.

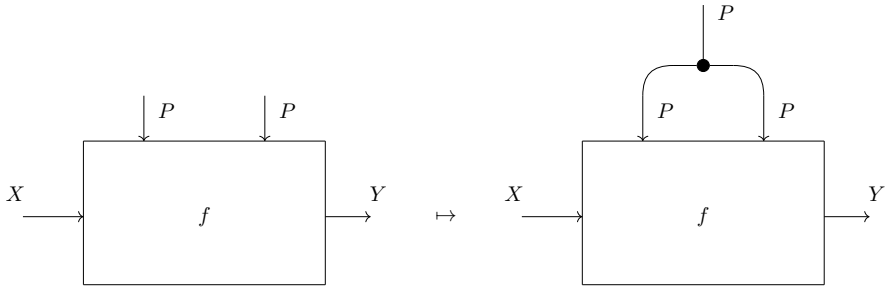


Figure 7.10: String diagram representation of weight tying.

Weight tying is most often applied in settings of sequential (Definition 3.14) and parallel (Prop. 3.39) composition of parametric morphisms. In sequential composition (Fig. 7.11 (left)), starting with two composable parametric morphisms that share the parameter, i.e.

$$(P, f) : \mathbf{Para}(\mathcal{C})(X, Y) \quad \text{and} \quad (P, g) : \mathbf{Para}(\mathcal{C})(Y, Z)$$

we can form the composite morphism $(f \fatsemi^{\mathbb{P}} g)^{\Delta_P}$ which now has coupled parameters. This is often used in recurrent neural networks (Section 7.2), where we want to ensure that the same action is implemented at every time step. For parallel composition (Fig. 7.11 (right)) the idea is the same. Starting with two parametric maps that have the same parameter type (here P):

$$(P, f) : \mathbf{Para}(\mathcal{C})(X, Y) \quad \text{and} \quad (P, g) : \mathbf{Para}(\mathcal{C})(Z, W)$$

the weight tying of f and g is the morphism $(f \times^{\mathbb{P}} g)^{\Delta_P}$ with coupled parameters, now used in parallel. This is used in generative adversarial networks (Definition 7.15), for instance, where the weight tying constrains the discriminator to use the same parameter while processing samples from the generator and those from the dataset.

Weight tying can also be used to model *batching*, a process by which from a neural network we

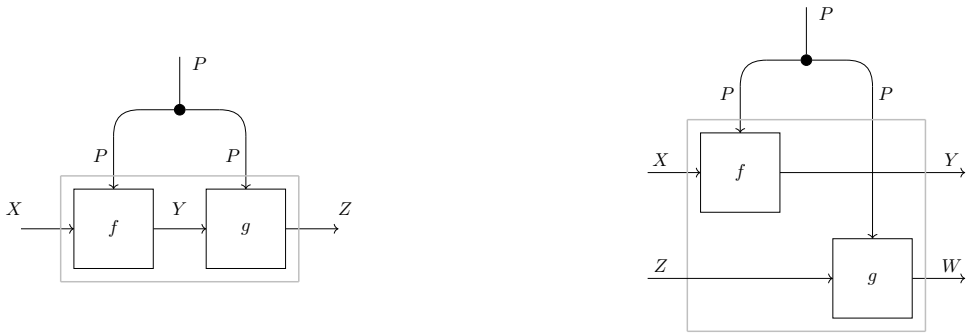


Figure 7.11: Weight tying a a) sequential and b) parallel composition of morphisms.

obtain a new one that can process multiple inputs at the same time.

Definition 7.12 (Batching as weight tying)**.** Let $(P, f) : \mathbf{Para}(\mathcal{C})(A, B)$ be a morphism. Then the n**-fold batching** of f is the morphism $(P, f^{\Delta_P^n}) : \mathbf{Para}(\mathcal{C})(A^n, B^n)$, where $\Delta_P^n : P \to P^n$ is the n-fold copy map.

In practice we can't merely copy neurons arbitrarily: we usually have to exploit some structure in our data, and perform weight tying in a principled way. In the next sections we will see a number of patterns of weight tying, describing recurrent, recursive, generative-adversarial, and many other neural network architectures.

Avenues for exploration. *Can we formulate the notion of a linear layer internal to any generalised cartesian reverse derivative category? What are the categorical semantics of activation functions? Can we think of them as (co)effects applied to linear maps, much like those that arise from (co)Kleisli categories of comonads? Is there a formal connection between weight-tying and higher order functions, as suggested in [Ola15]?*

7.2 Backpropagation through time: recurrent neural networks

So far we did not explicitly acknowledge the structure of inputs of neural networks. Often, the input to a neural network is an entire sequence of characters, frames, tokens or various other temporal inputs. Recurrent neural networks ([HS97, Sch19], [GBC16, Sec. 10.2]), or just RNNs, are those

that consume an input of this form, and backpropagation in this setting is often referred to as *backpropagation through time*. Thinking of a RNN as a morphism, it is a *stateful morphism*: it consumes a sequence of inputs X^n and produces another sequence Y^n, where here n is the length of the sequence, and each $j : n$ of the output depends on all $i : n$ of the input for $i \leq j$. Most often, this is done by repeating a single recurrent neural network *cell* across every time step.

Recurrent neural networks have thoroughly been characterised in [SK21] in terms of cartesian differential categories ([BCS09]) and the concept of the *delayed trace* introduced in the same paper. For more details we refer the interested reader to the aforementioned paper, and here we instead outline its key details. The paper is based on the definition of $\mathbf{St}(\mathcal{C})$: the category of stateful morphisms.

Definition 7.13 (Causal extension of a cartesian category (compare [SK21, Def. 11]))**.** Let $\mathcal{C}$ be a cartesian category. Its **causal extension** is a category denoted by $\mathbf{St}(\mathcal{C})$ whose objects are $\mathbb{N}$-indexed families of objects, and a morphisms are stateful morphism sequences [9] (see [SK21, Fig. 1]) Likewise, $\mathbf{St}_0(\mathcal{C})$ is a subcategory of $\mathbf{St}(\mathcal{C})$ whose family of objects are all the same, and morphisms given by iterating f. (See [SK21, Def. 15] for a precise specification).

There are two important considerations related to the framework of this thesis. Is there a parametric version of $\mathbf{St}(\mathcal{C})$, and can it be equipped with the coalgebra structure of $\mathbf{Lens}_A$?

As [SK21] does not mention **Para**, the first question is left unanswered. Nonetheless, as $\mathbf{St}(\mathcal{C})$ is a cartesian category, it is possible to form **Para** over it. On the other hand, $\mathbf{Para}(\mathbf{St}(\mathcal{C}))$ has too many degrees of freedom: in recurrent neural networks it is important that each recurrent cell inside the layer shares the same parameter (Fig. 7.12 (right)). We hypothesize there is an parametric category $\mathbf{Para}_\Delta(\mathbf{St}(\mathcal{C}))$ capturing this.

[9] That is, *equivalence classes* of stateful morphism sequences

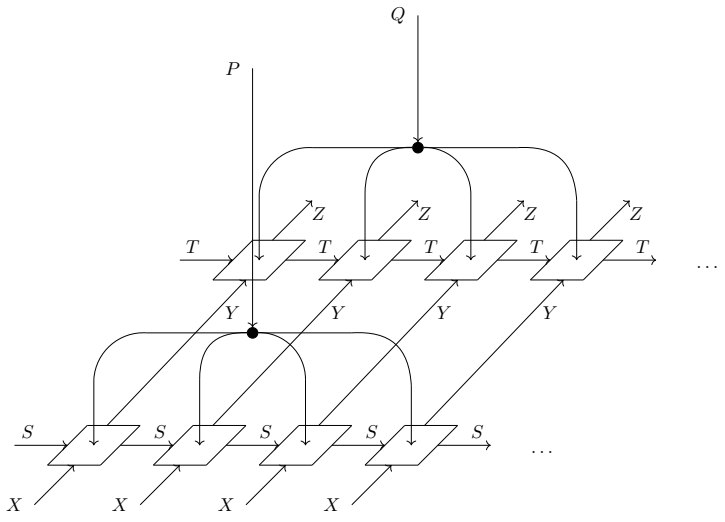


Figure 7.12: Two-layer recurrent neural network. Note that parameters are shared throughout time steps, but not throughout layers. This two-dimensional representation reinforces the idea that *parametric double categories* are a suitable framework for studying recurrent neural networks.

Pertaining to the second question, the paper shows that $\mathbf{St}(\mathcal{C})$ is a cartesian differential category when $\mathcal{C}$ is. It does not show $\mathbf{St}(\mathcal{C})$ is a *reverse* derivative category[10], though there do not seem to be any major obstacles to doing so.

Additionally, [SK21] also introduces *the delayed trace*, an operator generalising the categorical operation of trace with an implicit guardedness guarantee. This operator has potential in being applicable to more than just recurrent neural networks, as the authors hypothesize it could be useful for describing parameter update and meta-learning ([SK21, Sec. VI]).

The most popular kind of recurrent neural networks are Long Short-Term Memory (LSTM) networks ([HS97]) which introduce various gating mechanisms aimed at reducing the effect of vanishing and exploding gradients ([PMB13]). They appear in countless variants in practice ([YSHZ19]), none which appear to have been studied through the lens of category theory.

Avenues for exploration. *The work of [SK21] provides a comprehensive characterisation of recurrent neural networks through double categories. As mentioned in this section, avenues for exploration include establishing an action on* $\mathbf{St}(\mathcal{C})$ *which yields the appropriate bicategory* $\mathbf{Para}_{\Delta}(\mathbf{St}(\mathcal{C}))$*, and formulation of* $\mathbf{St}(\mathcal{C})$ *as a generalised reverse derivative category. Additionally, to the best of*

[10] Because [SK21] originally appeared before [CCG+20], the reverse derivative categories paper

our knowledge there is no work characterising recurrent neural networks to any of the recursion schemes or generalised kinds of (un)folds such as catamorphisms or anamorphisms. In other words, the structurally recursive component of $\mathbf{St}_0(\mathcal{C})$ *— being given as a repeat of just a single recurrent cell throughout the layer — is not captured by any of the usual categorical tools for doing so. We touch on this in the next chapter.*

7.3 Recursive neural networks, and more.

There is an interesting generalisation of recurrent neural networks called *recursive* neural networks ([SPW+13, GK96]).[11] Instead of consuming input data in the form of a list, recursive neural networks consume data in the form of a tree. This makes them amenable for use in a variety of settings, most notably that of parse trees in natural language processing. This was the original motivation for their introduction in [SPW+13] and seemingly the motivation for their only existing categorical model ([Lew19]). While this model establishes the relationship of recursive neural networks to the categorical compositional vector space semantics ([CSC10]), it does not establish any relationship of recursive neural networks to recurrent ones through the language of category theory as suggested by [Ola15]. Inspired by [Ola15] and unpublished research we have done on this topic, we provide hypotheses about category-theoretic formulation of recursive neural networks.

It is well-known in the category theory literature that lists and trees are examples of inductive data types whose categorical semantics are those of initial algebras of endofunctors.[12] More precisely, the set List_X of lists of elements of type X is the initial object of the category $(1 + X \times (-))$**-Alg**, and the set BTree_X of finite binary trees with X-labelled leaves is the initial object of the category $(X + (-)^2)$**-Alg**. As it happens, these endofunctors (and many more of interest) are all strong, enabling us to see them as actegory morphisms (Example 5.5). Consequently, via Prop. 5.9 this morphism induces an endomorphism $\mathbf{Para}(F)$ on $\mathbf{Para}(\mathbf{Set})$. And since every $\mathbf{Para}(\mathcal{C})$ holds a copy of $\mathcal{C}$ inside it (Lemma 3.32) we might expect that the category of $\mathbf{Para}(F)$-algebras holds a copy of F-algebras inside of it.

This is indeed the case, and part of our forthcoming work on this topic. We believe the claims of [Ola15] neural networks can be substantiated for encoding, decoding and recursive neural networks. Intuitively, if $\mathsf{BTree}_X : \mathbf{Set}$ arises as the initial algebra of the strong endofunctor $X + (-)^2 : \mathbf{Set} \to$

[11] From the latter reference comes the term "Backprop through structure".

[12] A more refined semantics of inductive data types and structural recursion in general is given by recursive coalgebras ([CUV06]).

Set, then it can be shown that this is an appropriate notion of a higher-dimensional inital algebra of $\mathbf{Para}(X + (-)^2) : \mathbf{Para}(\mathbf{Set}) \to \mathbf{Para}(\mathbf{Set})$.[13] In such a case, given any other algebra for (Y, f) for $\mathbf{Para}(X + (-)^2)$ there is a catamorphism unfolding the tree and performing weight sharing, roughly as in Fig. 7.13.

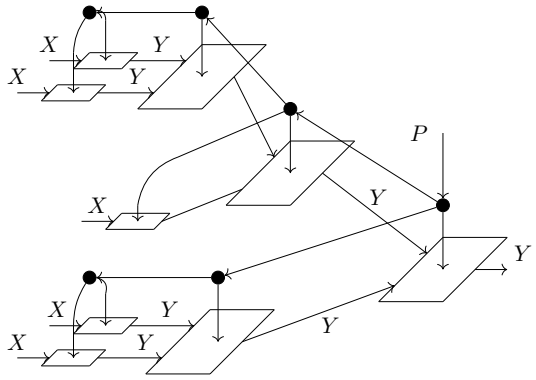


Figure 7.13: String diagram representation of a recursive neural network as a morphism in the category $\mathbf{Para}(X + (-)^2)$**-Alg**. The same parameter, and the same components of the underlying map $f : P \times (X + Y^2) \to Y$ are used throughout the network.

The generality of this initial algebra formulation suggests plenty of other examples. The initial algebra of $1 + X \times -$ is List_X, and its parametric form given some algebra (S, f) comes equipped with universal maps which describe *encoding neural networks*, as suggested by [Ola15]. We depict an example thereof in Fig. 7.14 (left) where we see a network consuming a list of inputs and folding over them with the map f in order to produce an accumulated value of type S. Dually, we can also consider *coalgebras* of $Y \times -$ where the terminal coalgebra is that of *streams* (Fig. 7.14 (right)) which generate an infinite sequence of outputs of type Y from a single starting state.

Avenues for exploration. *This section suggests a general categorical framework for studying structurally recursive neural networks: as that given by a category of algebras for a given endofunctor, and opens up a wide array of questions: what other (co)algebras can be studied in this way? It is known that Moore and Mealy machines can be seen as coalgebras for endofunctors [Rut00]. Are these suitable candidates for arhitectures? What other, potentially branching or non-deterministic architectures exist?*

[13] We believe that "quasi initial algebra" is the appropraite notion here.

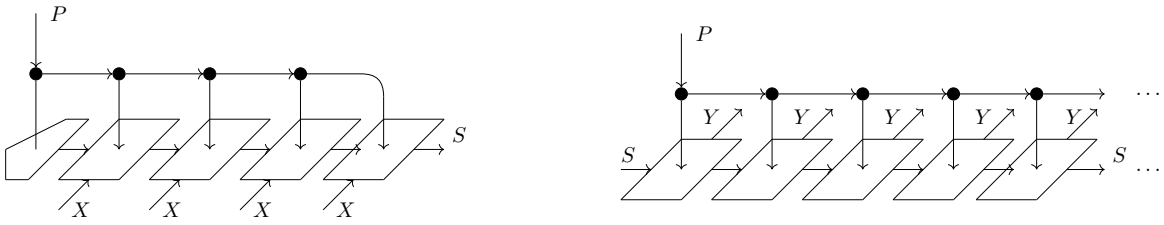


Figure 7.14: Encoding and decoding neural networks as morphisms in **Para**$(1 + X \times (-))$**-Alg** and **Para**$(Y \times (-))$**-Coalg**, respectively. As in Fig. 7.13, the same parameter value and the same type of implementation map is used througout.

7.4 Graph neural networks

In previous section we have described a categorical framework modelling neural networks that process a list, or a tree of inputs. Graphs are a more general data structure that can represent molecules, social neworks and transporation networks. In recent years graph neural networks became a sizeable subfield of deep learning, spawning a myriad of research and practical applications ([WPC+21, DV22, Vel22, MGY+21]).

Intuitively, consuming a graph as an input allows us to process each of its nodes by taking into account all of those around it. Most crudely, this can be described as a global underlying context (Section 3.1.2) the neural network layers have access to. These layers then compute updates for each node based on the connectivity pattern encoded by the graph. This was the category-theoretic treatment of *graph convolutional neural networks* (GCNNs) in [GV22] whose definition of a GCNN layer we present in Definition 7.14.

Definition 7.14 (GCNN Layer)**.** Fix the number of nodes $n : \mathbb{N}$ in a graph, input and output types $x, y : \mathbb{N}$, and an activation function $\sigma : \mathbb{R}^y \to \mathbb{R}^y$. A graph convolutional neural network layer $x \to y$ is a morphism in **Para**(**coKl**$(- \times \mathbb{R}^{n\times n}))(\mathbb{R}^{x\times n}, \mathbb{R}^{y\times n})$ whose parameter space is $\mathbb{R}^{x\times y+y}$ and the implementation is

$$f : \mathbb{R}^{x\times n} \times \mathbb{R}^{x\times y+y} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{y\times n}$$
$$(X, (W, B), A) \mapsto \sigma(W^\top X A + B)$$

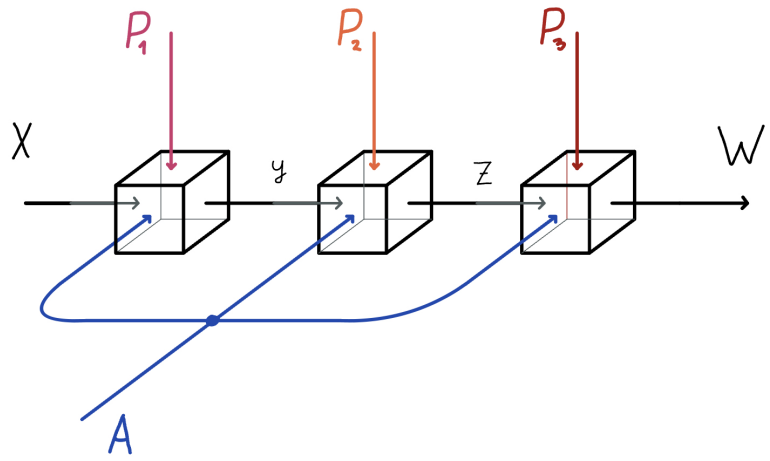


Figure 7.15: Composition of three layers in a graph convolutional neural network. On the vertical axis, we see three parameter spaces: P_1, P_2, and P_3 of each layer. On the horizontal axis, we see the adjacency matrix A as a "parameter" to each layer. By composing new layers on the right the number of parameters increases, but the adjacency matrix is just copied.

The global context of $\mathbf{coKl}(- \times \mathbb{R}^{n\times n})$ ensures that the same adjacency matrix used in one layer is used throughout the whole graph neural network. In [GV22] it has been shown that n-node convolutional graph neural networks are morphisms in $\mathbf{Para}(\mathbf{coKl}(- \times \mathbb{R}^{n\times n}))$, and that the base $\mathbf{coKl}(- \times \mathbb{R}^{n\times n})$ is a generalised reverse derivative category (Example 6.25). Despite this kind of a formulation having convolutional neural networks ([BL95]) as a special case, in many ways its categorical form is limited. Graphs are encoded as an adjacency matrix and their geometry is not explicitly accounted for in category-theoretic terms. Furthermore, the implemented multiplication by an adjacency matrix implements only a special form of *message passing* ([Vel22]). It is a special case because only nodes in this graph are assumed to have features attached to them, and edge-level and graph-level features are ignored.

General message passing equips graphs with node, edge and graph-level features, describing structured ways information is propagated ([Vel23, Eq. 8, 9, 10] and [DV22]). We do not study this further except to say that message passing (specifically, [Vel23, Eq. 10]) can be formulated as a lens whose forward part is tasked with broadcasting the information out of a node, and the backward pass is tasked with aggregating received information from other nodes.

We also mention the concept of equivariance. It can be thought of as a formal notion of consistency under transformation. For instance, convolutional neural networks the above example

models are translation equivariant, meaning they allow us to state how translations of features in the input correspond to a translation of the features of the output. In other words, in the task of image segmentation we get the same result if we a) translate an object in the original image, and then apply the convolutional neural network, or b) apply the convolutional neural network, and then translate the generated mask. But translations are not the only kind of transformation. Transformations can be equivariant with respect to a group of rotations, reflections, scaling, and many others. Equivariance with respect to an arbitrary group has been studied in [CW16], and in more general terms in [Har19, dHCW20, BBCV21].

Lastly, graph neural networks have recently related to the concept of dynamic programming through the language of category theory ([DV22]). As they use the formalism of polynomial functors whose morphisms are dependent lenses, we believe this is an exciting research direction that could place them on even more solid categorical foundations.

Avenues for exploration. *There is a well-understood way of generalising the category* $\mathbf{coKl}(- \times \mathbb{R}^{n\times n})$ *to a fibred setting, analogously how* $\mathbf{coKl}(- \times X)$ *can be embedded into the slice category* $\mathcal{C}/X$*. This allows the dimensionality of vector space of features to depend on the node it is fibred over. Likewise, the abstract machinery of the Grothendieck construction here yields a particular instantiation of message-passing.*[14] *Can this be used as a generalisation of the above formulation of graph convolutional neural networks?*

Lastly, [BBCV21, CW16] host numerous examples of group equivariant neural networks described as homomorphisms of group actions. All of these arise as morphisms of monad algebras for the group action monad. By generalising these homomorphisms to account for possibly different monads on the domain and codomain, we obtain group invariant neural networks. This suggests a host of other examples, and a generalised theory of (co)equivariance as that of (co)monad/endofunctor algebra homomorphisms. Such a formulation further suggests connections to structurally (co)recursive neural network formulations in Section 7.3 where they are also formulated as (co)algebra homomorphisms. Can this theory pave way for a theory of architectures of neural networks? We believe so.

[14] This came up in a conversation with Matteo Capucci.

7.5 Generative Adversarial Networks

Generative Adversarial Networks ([GPAM+14]), or GANs are an important architecture that lies in the centre of the intersection of deep learning and game theory. Unlike the architectures described above, GANs do not arise as an instantiation of the learning framework in a different kind of a category. Instead, a GAN is a system of two neural networks trained with "competing" optimisers. One neural network is called *the generator* whose optimiser is, as usual, tasked with moving in the direction of the negative gradient of the loss. However, the other network — called *the discriminator* — has an optimiser which is tasked with moving in the *positive*, i.e. ascending direction of the gradient of the total loss — maximising the loss. The actual networks are wired in such a way (Fig. 7.17) where the discriminator effectively serves as a loss function to the generator, i.e. being the generator's only source of information on how to update. Dually, taking the vantage point of the discriminator, the generator serves as an ever changing source of training data.

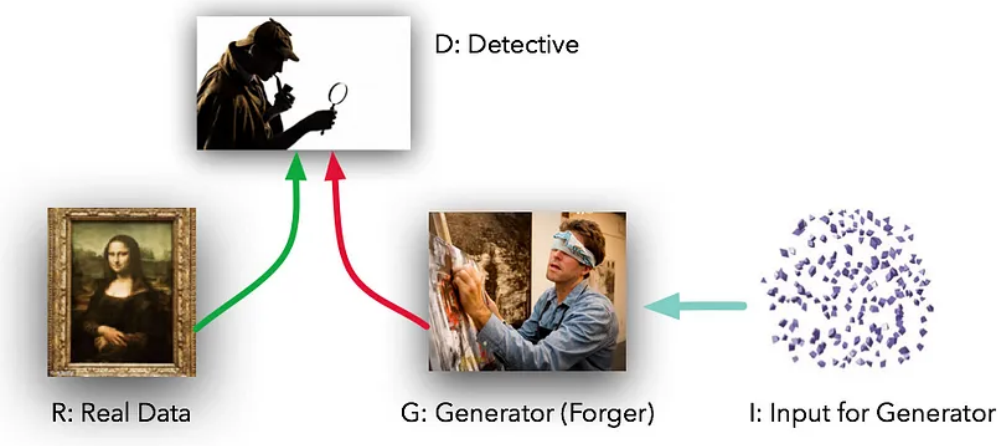


Figure 7.16: GANs can intuitively be understood as a cat-and-mouse game. Think of the generator as a forger producing counterfeit paintings from a particular artist, and the discriminator as the detective distinguishing between real and fake artwork. They both start without knowing how real artwork looks like, and slowly improve as they interact. For instance, the detective might first see a real painting. If they misclassify it as fake, they receive a training signal instructing them how to classifying paintings similar to it as real next time. An analogous story works if they see a forgery. But in this case, the forger also receives a training signal, and learns how to imrpove *so as to fool the detective* better next time. This causes a feedback loop where both of them continue improving, enabling the forger to create plausible looking artwork. Figure from [15].

[15] https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f

GANs have been a popular architecture that has received wide attention and has been applied in numerous domains [PYY+19]. Following our previous category-theoretic work on GANs in [CGHR22, Sec. 4.1], we present their novel definition below.[16]

Definition 7.15 (GAN)**.** Fix three objects Z, X and L in $\mathcal{C}$ (respectively called "the latent space", "the data space" and "the payoff space"). Then given two parametric morphisms

$$(P, g) : \mathbf{Para}(\mathcal{C})(Z, X) \quad \text{and} \quad (Q, d) : \mathbf{Para}(X, L)$$

a **generative adversarial network** is a morphism $(P \times Q, \mathsf{GAN}_{g,d}) : \mathbf{Para}(Z \times X, L \times L)$ where $\mathsf{GAN}_{g,d}$ is defined as the composite

$$\mathsf{GAN}_{g,d} \coloneqq \boxed{Z \times X \xrightarrow{g \times^{\mathrm{p}} \mathrm{id}_X} X \times X \xrightarrow{(d \times^{\mathrm{p}} d)^{\Delta_Q}} L \times L}$$

Its string diagram representation is shown in Fig. 7.17 where we see that a GAN consists of two parallel tracks. We will see in Example 8.19 how the first one will be used to process latent vectors, and the second one to process samples from a chosen dataset. Despite the fact that there are two boxes labeled d, they are weight tied (Definition 7.11), making them behave like a singular unit.

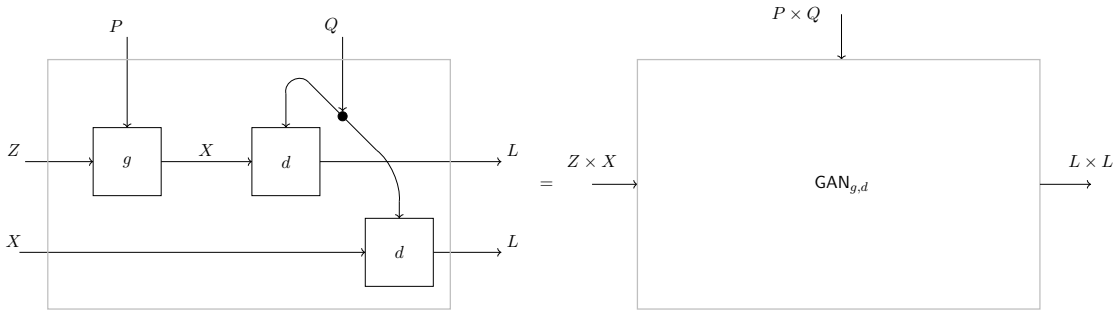


Figure 7.17: A generative adversarial network.

[16] Noting that their forward part can be stated in any cartesian category $\mathcal{C}$.

We can easily state what the reverse derivative of $\mathsf{GAN}_{g,d}$ is in terms of its components:

$$R[\mathsf{GAN}_{g,d}](z, x_r, p, q, \alpha_g, \alpha_r) = (z', x'_r, p', q'_g + q'_r) \quad \text{where} \quad \begin{aligned} (x'_g, q'_g) &= R[d](g(z,p), q, \alpha_g) \\ (x'_r, q'_r) &= R[d](x_r, q, \alpha_r) \\ (z', p') &= R[g](z, p, x'_g) \end{aligned} \tag{7.16}$$

The pair $(\mathsf{GAN}_{g,d}, R[\mathsf{GAN}_{g,d}])$ yields a parametric lens of type $\binom{Z\times X}{Z'\times X'} \to \binom{L\times L}{L'\times L'}$ (Fig. 7.18), which we interpret as follows.

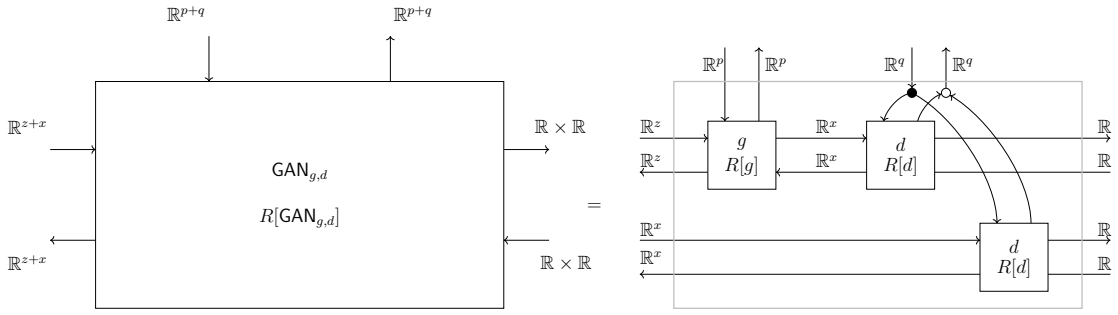


Figure 7.18: A generative adversarial network under the image of **Para**($\mathbf{R}_{\mathcal{C}}$).

It consumes two pieces of data, "a latent vector" $z : Z$, a "real" sample from the dataset $x_r : X$, in addition to the parameter $p : P$ for the generator and a parameter $q : Q$ for the discrminator. What happens then are two independent evaluations done by the discriminator. The first one uses generator's attempt of producing a sample from the dataset (the latent vector which was fed into it, producing $g(z, p) : X$) as input to the discriminator, producing a payoff $d((g, z, p), q) : L$ for this particular sample. The second one uses the actual sample from the dataset x_r, producing the payoff $d(x_r, q) : L$. We will see in Example 8.19 how the learning context this GAN is trained in produces dynamics which push the generator towards generating real-looking samples, and the discriminator towards discriminating between real and generated samples.

There are various other generalisations of GANs ([Hin23]) out of which we point out CycleGAN ([ZPIE17]). This is an architecture that combines autoencoders and generative adversarial networks in a non-trivial manner using the concept of "cycle-consistencies", a loss function enforcing high-level composition invariants of neural networks. In ([Gav20b]) it has been argued that this system can be seen as one arising out of a presentation of a category through generators and relations,

where each relation gives rise to a corresponding cycle-consistency loss.

Avenues for exploration. *An avenue for future work is GANs in the setting of a base category with coproducts. In such a setting we'd have the coproduct injection $\nabla_X : X + X \to X$ at our disposal. Intuitively, ∇_X consumes either a datapoint created by the generator, or a datapoint from the training data, and erases its label before passing it on to the discriminator (Fig. 7.19).*

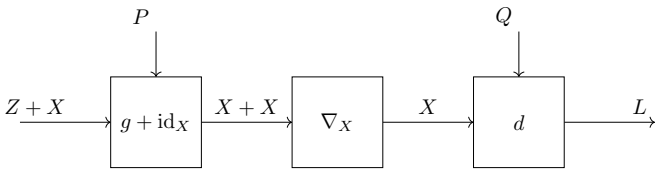


Figure 7.19: A generative adversarial network defined using coproducts.

This provides a more natural interpretation of GANs as it does not necessitate the processing of both Z and X at the same time (unlike Definition 7.15), and additionally simplifies the treatment of loss functions (see discussion after (Example 8.19)) Further work needs to be done providing categorical semantics of differentiation of this architecture in terms of lenses, as lenses do not have all coproducts even if their base does (Section 4.5.1). Another avenue for work is formalising GAN-specific optimisers such as competitive gradient descent *([SA19]) which we mention in more detail in Section 8.1.3.*

7.6 Transformers

Transformers are one of the most influential architectures in deep learning [VSP+17, TDBM22, PH22]. They're a recent model, originally proposed as a component of a sequence-to-sequence model for machine translation in the paper *Attention is All you Need* ([VSP+17]). They have since been used to achieve the state-of-the-art performance on various tasks in natural language processing ([BMR+20]), computer vision ([DBK+21]), and audio processing ([CRBD18, DXX18]), most notably being used as a central piece of the large language model ChatGPT ([Ope22]).

One of the most interesting properties of transformers is their ability to learn during the inference stage. This property — called *in-context learning* ([BMR+20, VONR+23]) — is a kind of learning that happens on the *forward pass*, without any parameter updates. Learning instructions (possibly with input-output examples) are encoded as part of the input prompt which the already trained

transformers then consumes. Afterwards, the transformer can be prompted with previously unseen examples to which it in in many cases successfully generalises by producing the desired output as a response.[17] In-context learning additionally blurred the lines between training and test stages, motivating research on the meta-learning ability of transformers ([MRCA18, Mel22b]).

So, what is a Transformer? Just like a recurrent neural network (Section 7.2), it is a model which a) consumes a sequence of inputs, b) can be repeated sequentially in layers, and c) consists of a modular component repeated across every time step. But importantly, unlike in a recurrent neural network, in transformers the modular component does not have a concept of *state* which is iteratively updated at every time step (Fig. 7.12). Likewise, unlike the "clean" design of a recurrent neural network, the transformer layer contains many other details that we remark on at the end of this subsection.

The aforementioned modular component is the *attention* mechanism: the centerpiece of the Transformer. This is a mechanism meant to mimic cognitive attention found in humans and animals: allocating more resources towards processing a particular part of the input, while ignoring the others. Variants of attention have been present in architectures such as Neural Turing Machines ([GWD14]) and Differentiable Neural Computer [GWR+16], though they were not explicitly defined, nor a centerpiece of the paper. We define the attention component below, and explain the manner by which it's embedded within the transformer architecture.

Definition 7.17 (Attention ([VSP+17]))**.** Fix $\mathsf{seq}, \mathsf{val}$ and $\mathsf{key} : \mathbb{N}$ denoting, respectively, the length of the sequence to be processed (whose elements are called *tokens*), the dimensionality of feature vectors associated with each token, and the dimensionality of *key*[18] vectors. Then attention is the following morphism in **Smooth**:

$$\begin{aligned}\mathsf{Attend} : \mathbb{R}^{\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}\times\mathsf{key}} \times \mathbb{R}^{\mathsf{seq}\times\mathsf{val}} &\to \mathbb{R}^{\mathsf{val}} \\ (Q, K, V) &\mapsto \mathsf{Softargmax}(\frac{QK^\top}{\sqrt{\mathsf{key}}})V\end{aligned}$$

We leave out a full description of its semantics to ([VSP+17, PH22]), instead here merely pointing out that the inputs Q, K and V are called *query*, *key*, and *value* vectors, and that Q represents the query of the token to the rest of the sequence which the Attend morphism evaluates.

Here Attend is a computation done *for each token* in the sequence. The manner by which this

[17] The manner by which they learn in-context was shown to be, surprisingly, *gradient descent* ([VONR+23])!
[18] One can think of this as an identifier of the feature, possibly even an encoding of its *type*.

is extended to the entire sequence involves an additional step of *reuse of the K and V matrices*. Formally, we can do this by treating Attend as a morphism in $\mathbf{coKl}(- \times \mathbb{R}^{\mathsf{seq} \times \mathsf{key}} \times \mathbb{R}^{\mathsf{seq} \times \mathsf{val}})(\mathbb{R}^{\mathsf{key}}, \mathbb{R}^{\mathsf{val}})$ and taking its parallel product seq times inside this category. This produces $\mathsf{Attend}^{\mathsf{seq}} : \mathbf{coKl}(- \times \mathbb{R}^{\mathsf{seq} \times \mathsf{key}} \times \mathbb{R}^{\mathsf{seq} \times \mathsf{val}})(\mathbb{R}^{\mathsf{seq} \times \mathsf{key}}, \mathbb{R}^{\mathsf{seq} \times \mathsf{val}})$ whose type in $\mathbf{Smooth}$ is $\mathsf{Attend}^{\mathsf{seq}} : \mathbb{R}^{\mathsf{seq} \times \mathsf{key}} \times \mathbb{R}^{\mathsf{seq} \times \mathsf{key}} \times \mathbb{R}^{\mathsf{seq} \times \mathsf{val}} \to \mathbb{R}^{\mathsf{seq} \times \mathsf{val}}$, and implementation is

$$(\{Q_i\}_i^{\mathsf{seq}}, K, V) \mapsto \{\mathsf{Attend}(Q_i, K, V)\}_i^{\mathsf{seq}}$$

Interestingly, this is not a morphism in $\mathbf{Para}(\mathbf{Smooth})$: there are no parameters here! If there are no parameters, how does the self-attention layer learn? The idea is simple: there is a fully-connected layer precomposed with attention that does the learning.[19] This fully-connected layer learns how to produce the correct query, key, and value information that then allows the entire transformer architecture to learn how to self-attend.

This is a very rough story. The actual transformer architecture contains multiple self-attention layers, each with multiple *attention heads* (meaning a token can produce multiple queries), residual connections, and various forms of normalisation (see Remark 7.18). Furthermore, its details often vary from implementation from implementation. For more detail, we point to [Bey23, TDBM22, PH22].

Remark 7.18 (Normalisation)**.** Normalisation is another active area of study in deep learning. It refers to the transformation of input data, activations or weights in a particular way to be on a similar scale, helping stabilies and speed up learning. They can be performed across the batch dimension ([IS15]), across the layer dimension ([BKH16]), or across weights ([SK16]). The Transformer architecture in particular uses layer normalisation.

Avenues for exploration. *Very little is known about the categorical semantics of transformers. Does any of the versions of transformers — potentially with some ad-hoc details removed — possess a universal property? What is the categorical semantics of in-context learning, and is it related in any way to the microcosm principle in category theory? Is the fact that vision transformers* learn to be equivariant *([GFGW23]) related in any way to parametric algebras from Section 7.3 whose homomorphisms are conjectured to model generalised notion of equivariance?*

[19] Of course, we first need to embed attention into $\mathbf{Para}(\mathbf{Smooth})$ via the functor $\mathcal{C} \to \mathbf{Para}(\mathcal{C})$ (Lemma 3.32), which treats it a trivially parametric morphism.

7.7 Loss functions

Loss functions are a central component of deep learning systems. The loss function consumes the output of our model and quantifies how well its prediction match the actual data. By differentiating the composite (Fig. 7.20) we can provide a signal for the learning process to follow.

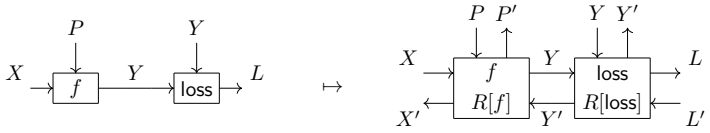


Figure 7.20: Loss function composed with a model.

In standard learning in **Smooth** the loss function is viewed as a map of type $Y \times Y \to \mathbb{R}$. Here Y is the output type of the model, and $\mathbb{R}$ is *a payoff* (or *loss*) object. In different kinds of learning settings there will be a different kind of a payoff object which quantifies how well the model is faring — for instance, in categories like $\mathbf{Poly}_{\mathbb{Z}_2}$ the payoff object will be $\mathbb{Z}$.[20] We thus abstract away this payoff object and denote it with L. Furthermore, in our setup it will be natural to view the loss map as a parametric map $Y \to L$ whose parameter space is Y. Thus the loss function on Y is here simply a morphism in $\mathbf{Para}(\mathcal{C})(Y, L)$.

We now give a series of examples. In all of them y_t will represent the ground **t**ruth, y_p model's **p**rediction, and loss the underlying implementation of this loss function.

Example 7.19 (Mean squared error)**.** One of the simplest and most common loss functions is mean squared error defined in **Smooth**. Given an object $\mathbb{R}^y$ mean squared error is a parametric morphism $(\mathbb{R}^y, \mathsf{loss}) : \mathbf{Para}(\mathbf{Smooth})(\mathbb{R}^y, \mathbb{R})$ where

$$\mathsf{loss}(y_p, y_t) = \frac{1}{2}\sum_{i=1}^{y}((y_p)_i - (y_t)_i)^2$$

Its reverse derivative is $R[e](y_p, y_t, \alpha) = (\alpha(y_p - y_t), \alpha(y_t - y_p))$.

Example 7.20 (Boolean error)**.** In $\mathbf{Poly}_{\mathbb{Z}_2}$ the loss function on $\mathbb{Z}^y$ that is implicitly used in [WZ21]

[20] At the present moment the choice of the payoff object is still done in an ad-hoc way, often as the base field of the underlying vector space. It is not what properties it needs to satisfy.

is given by the parametric morphism $(\mathbb{Z}^y, \mathsf{loss}) : \mathbf{Para}(\mathbf{Poly}_{\mathbb{Z}_2})(\mathbb{Z}^y, \mathbb{Z})$ where

$$\mathsf{loss}(y_p, y_t) = y_p + y_t$$

Note that $+$ in $\mathbb{Z}_2$ is given by XOR. Its reverse derivative, following Fig. 7.4, unpacks to $R[e](y_p, y_t, \alpha) = (\alpha, \alpha)$.

Example 7.21 (Softargmax cross-entropy)**.** One of the canonical functions used in classification is softargmax cross-entropy (often called softmax cross-entropy). Given an output object $\mathbb{R}^y$, it is a parametric map $(\mathbb{R}^y, \mathsf{loss}) : \mathbf{Para}(\mathbf{Smooth})(\mathbb{R}^y, \mathbb{R})$ where

$$\mathsf{loss}(y_p, y_t) = \frac{1}{2}\sum_{i=1}^{y} \mathsf{Softargmax}((y_t)_i)((y_p)_i - \log \mathsf{Softargmax}((y_p)_i))$$

and where $\mathsf{Softargmax}$ is defined in example Example 7.9.[21]

There are some stranger examples as well! Unbeknownst to most, the loss function that is at the heart of *Deep Dreaming* (Section 8.2.2) is the one given by the dot product.

Example 7.22 (Dot product)**.** In deep dreaming we often want to focus on a particular element $i : y_p$ of the network output $y_p : R^y$ and ignore all the others. This is done by supplying a one-hot vector y_t as the ground truth to this loss function. The dot product then performs masking of all elements of y_p, except the one at which one-hot vector y_t was active at. More precisely, given an output type $\mathbb{R}^y$, the dot product is a parametric map $(\mathbb{R}^y, \mathsf{loss}) : \mathbf{Para}(\mathbf{Smooth})(\mathbb{R}^y, \mathbb{R})$ where

$$\mathsf{loss}(y_p, y_t) = y_p \cdot y_t$$

Its reverse derivative is $R[e](y_p, y_t, \alpha) = (\alpha y_t, \alpha y_p)$.

Avenues for exploration. *Very little is known about the categorical semantics of loss functions. Do any of them — especially the ones involving cross-entropy — have a universal property? What is the minimal structure needed to express each one? Likewise, what is the minimal structure needed of the payoff object L for learning?*

[21] Observe that in [CGG+22, Ex. 8] the formula is slightly different because of the assumption that y_t is a probability distribution. Our approach is type-safe, making it appear distinct from those in the literature.

Chapter 8

Supervised Learning

> The question of whether a computer can think is no more interesting than the question of whether a submarine can swim.
>
> Edsger W. Dijkstra

We have done a lot of things so far. We explained how to understand parts of neural networks categorically: how to model their weights, backpropagation, how to model various architectures, all in ways that are somehow vastly general — for instance not tied to euclidean spaces, nor restricting ourselves to an implementation thereof — while still capturing the essence of backpropagation, architecture, loss functions and so on. What remains is to describe how the puzzle pieces fit together.

In Fig. 7.20 we have seen how to compose the model and the loss together. By applying the general machinery of Chapter 5, specifically Prop. 5.9, we can take the underlying $\mathbf{Lens}_A$-coalgebra, i.e. a functor $\mathbf{R}_{\mathcal{C}} : \mathcal{C} \to \mathbf{Lens}_A(\mathcal{C})$ and apply **Para** to it, yielding a pseudofunctor

$$\mathbf{Para}(\mathbf{R}_{\mathcal{C}}) : \mathbf{Para}(\mathcal{C}) \to \mathbf{Para}(\mathbf{Lens}_A(\mathcal{C}))$$

which takes a parametric map and augments it with its backward derivative. We note that, despite our approach being completely architecture agnostic, it is still a precise specification of the

derivative computation. For instance, if our base category $\mathcal{C} \coloneqq \mathbf{Smooth}$ with its usual derivative structure, we get standard backpropagation in the usual sense. For $\mathcal{C} \coloneqq \mathbf{coKl}(\mathbb{R}^{n\times n} \times -)$ and the induced differential structure from **Smooth** we model backpropagation of neural networks with a shared context, such as graph convolutional neural networks (Section 7.4). For $\mathcal{C} \coloneqq \mathbf{St}_0(\mathcal{C})$ and its induced differential structure we model backpropagation on recurrent neural networks (Section 7.2). Of course, the base category can be changed, and we could have studied $\mathbf{St}_o(\mathbf{Poly}_{\mathbb{Z}_2})$ for instance — backpropagation of recurrent neural networks in the setting of boolean circuits. And so on. After differentiating, in any of these cases we will end up with a parametric lens $\binom{X}{X'} \to \binom{L}{L'}$ like the one below.

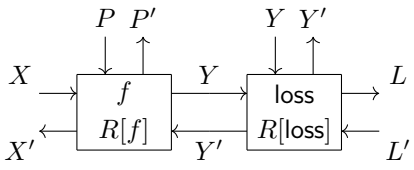


Figure 8.1: Model composed with a loss function, augmented with the reverse derivative.

This lens has a forward part $f \fatsemi^{\mathsf{p}} \mathsf{loss} : X \times P \times Y \to L$ which consumes an input datapoint, a parameter and a label and produces a measure of how well the neural network did on this example. Its backward pass (which we will denote by update in this chapter), is a morphism of type $X \times P \times Y \times L' \to X' \times P' \times Y'$. It consumes an input datapoint, a parameter, a label and a learning rate. What it then does is, via the composition rules of parametric maps (Definition 3.14) and lenses (Definition 4.37), is produce gradients of the loss with respect to: a) ground truth label (usually ignored in practice), parameter (used to update the network), and input (usually ignored in practice, unless one is doing deep dreaming (Section 8.2.2)). More precisely,

$$\mathsf{update}(x, p, y_t, \alpha) = (x', p', y_t') \quad \text{where} \quad \begin{aligned} y_p &= f(x, p) \\ (y_p', y_t') &= R[\mathsf{loss}](y_p, y_t, \alpha) \\ (x', p') &= R[f](x, p, y_p') \end{aligned}$$

In the next section we describe how to "close off" this update map from the left side, the right side, and even from the top.

Contributions. *Almost the entirety of this chapter is novel contribution, centering around the application of the base change of* **Para** *to the reverse derivative functor* $\mathcal{C} \to \mathbf{Lens}_A(\mathcal{C})$*. Novel contribution is the proof that* $\mathbf{Para}(\mathcal{C})$ *has quasi-initial objects (Box 8.1.1), optimiser composition (Section 8.1.3), and the framing of GANs as a supervised learning system (Example 8.19). Most of this was previously published in "Categorical Foundations of Gradient-Based Learning" ([CGG+22]). For a more nuanced discussion of originality, see Section 8.3.*

Epistemic status. *The fact that the resulting theory of supervised learning takes on a rather simple form makes me confident that the contents of this chapter are on the right track: there are no major issues as I see it. The categorical formalism between some smaller components has directions for improvement, though. For instance, the formulation of learning rates as parametric lenses is unsatisfactory because it is not in the image of a reverse derivative functor. It is plausible that better foundations of differentiation will help shed light on this issue, though a more conclusive answer might be obtained by providing a better theory of iteration of learning. Iteration of learning might be something that definitively informs whether the formulation of supervised learning in this chapter is correct, as meta-learning mentioned in Box 8.1.2 involves unrolling an iterated learner which processes not just datapoints, but entire datasets in one update step. Likewise, I envision that the refinement of optimisers by the way of dependent types is another necessary theoretical step, as currently many artificial identifications are made in their definition.*

8.1 Corners, learning rates, and optimisers

In order to perform learning we will need to supply datapoints, a learning rate, and importantly, an optimiser that will tell us how to update the parameter in light of gradient information.

8.1.1 Left side: corners

From the left side, we need *a state*: a parametric lens of type $\binom{1}{1} \to \binom{X}{X'}$. Interestingly, such a lens always exists!

Categorical aside: Para($\mathcal{C}$) has a quasi-initial object. **Box 8.1.1**

Proposition 8.1. *Let $(\mathcal{C}, \otimes, I)$ be a monoidal category. Then I is a quasi-initial object (Definition E.3) of* **Para**$(\mathcal{C})$*, where for every object X :* **Para**$(\mathcal{C})$ *the strict initial object of the hom-category* **Para**$(\mathcal{C})(I, X)$ *is (X, λ_X) where $\lambda_X : I \otimes X \to X$ is the left unitor of $\mathcal{C}$.*[a]

Proof. To prove that (X, λ_X) is initial in **Para**$(\mathcal{C})(I, X)$ we need to show that for any other (P, f) : **Para**$(\mathcal{C})(I, X)$ there is a unique reparameterisation $r : (X, \lambda_X) \Rightarrow (P, f)$ i.e. that there exists a unique map $r : P \to X$ such that $\lambda_X^r = f$. This map is

$$r := \boxed{P \xrightarrow{\lambda_P^{-1}} I \otimes P \xrightarrow{f} X}$$

and reparameterisation condition can be shown using naturality of λ. □

Note that since **Para**$(\mathcal{C})$ is a bicategory, this also means that given any (Q, g) : **Para**$(\mathcal{C})(X, Y)$ there is a unique 2-cell $(Y, \lambda_Y) \Rightarrow ((X, \lambda_X) \fatsemi^{\mathsf{p}} (Q, g))$ arising out of post-composition of (X, λ_X) with (Q, g). Its underlying reparameterisation is a map of type $X \otimes Q \to Y$ and as it turns out this map is precisely g. Dually, it can be shown that **coPara**$(\mathcal{C})$ has a quasi-*terminal* object. A variant of this was explored in [HS19], in which the "affine reflection" is an instance of **coPara**.

[a] This is something I learned from Dario Stein.

It is a parametric lens whose parameter space is that of the domain, and its implementation λ_X does nothing — it is isomorphic to the identity map on X. In string diagram notation it resembles a "corner", as it twists the input coming from the top to the right side. Like the ones before, this lens also arises as the image under the differentiation functor. In this case, it is the image of the identity map (Example 7.1), drawn in Fig. 8.2 in its parametric form.

Figure 8.2: Quasi-initial object in **Para**$(\mathcal{C})$ can be interpreted as a "corner".

Composing it with the rest of the system we obtain Fig. 8.3.

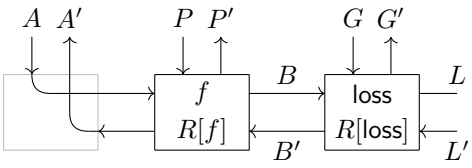


Figure 8.3: Model, loss function and a state, forming a parametric lens $\binom{1}{1} \to \binom{L}{L'}$.

8.1.2 Right side: learning rates

From the right side, we need *a costate*: a parametric lens of type $\binom{L}{L'} \to \binom{1}{1}$. This is a lens that consumes the produced the output L of the loss function, and "turns it backward" into a gradient L' to be backpropagated. We will see how this corresponds tightly to the concept of *the learning rate*: a tuning parameter that determines the step size at each iteration while moving towards the optimum of the loss function. In all the cases of interest it will be a non-parametric lens, hence the definition below.

Definition 8.2. A *learning rate* on a payoff object $L : \mathcal{C}$ is a morphism $\binom{L}{L'} \to \binom{1}{1}$ in **Lens**$(\mathcal{C})$.

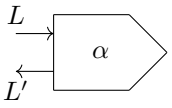


Figure 8.4: Learning rate as a lens with trivial parameters.

Note that, in addition to its lack of parameters, there is no requirement that the backward pass of this lens is additive in the second component. If it were additive, then there would exist only one learning rate: the one given in Example 7.4 whose backward pass is constant at zero. This is also the derivative of the forward pass, as it is the only plausible candidate. But as we will see — learning rate is hardly ever in the image of the differentiation functor R.

To see how this definition resembles a learning rate, it is useful to recall (Box 4.4.1) which tells us that a costate $\alpha : \binom{L}{L'} \to \binom{1}{1}$ in **Lens**$(\mathcal{C})$ is isomorphic to a morphism $\alpha : L \to L'$ in $\mathcal{C}$. This is precisely how we will think of this lens from now on: as a morphism $\alpha : L \to L'$ in $\mathcal{C}$.

Example 8.3. In standard supervised learning in **Smooth** with payoff object $\mathbb{R}$ we fix the learning rate $\mathbb{R} \to \mathbb{R}$ as constant one at some $\alpha \sim 10^{-2}$.

Example 8.4. In $\mathbf{Poly}_{\mathbb{Z}_2}$ the relevant learning rate will $\alpha : \mathbb{Z} \to \mathbb{Z}$ is the one given by the identity map.

Other learning rate morphisms are possible as well. One could fix some $\epsilon > 0$ and define the learning rate in **Smooth** by $\alpha(l) = \epsilon l$. Such a learning rate would take into account how far away the network is from the desired goal and adjust the output accordingly. Composing the learning rate with the model and loss in Fig. 8.1 we obtain Fig. 8.5.

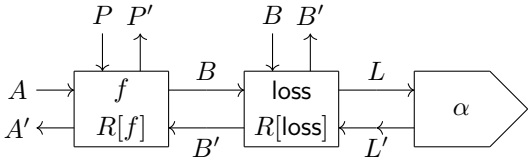


Figure 8.5: Model, loss function and a learning rate forming a parametric lens $\binom{X}{X'} \to \binom{1}{1}$.

8.1.3 Top side: optimisers

In addition to closing off this system from the left and the right side, we will also need to manipulate it from the top. Recall what happens at the ports $\binom{P}{P'}$ on the top. The parameter $p : P$ is supplied to the network, and the gradient $p' : P'$ is produced. Once this happens, we are interested in computing a new parameter as a function of these two, often by moving a small step size in the direction p'. This can succintly be captured as a lens $\binom{P}{P} \to \binom{P}{P'}$ which we think of being stacked on top of the model — it is precisely a reparameterisation! [1] This is what we call a gradient update, and is in neural network terminology called *an optimiser* ([Shi23]).

In this section we explore how to model optimisers using lenses and reparameterisations, modelling both non-stateful and stateful variants (Fig. 8.6). We begin by modelling gradient *ascent* — an update that can be captured in any cartesian left-additive category $\mathcal{C}$.

Definition 8.5 (Gradient ascent)**.** Let $\mathcal{C}$ be a cartesian left-additive category. Gradient ascent on $P : \mathcal{C}$ is a lens

$$\binom{\mathrm{id}_P}{+_P} : \binom{P}{P} \to \binom{P}{P'}$$

where $+_P : P \times P' \to P$ is the monoid structure of P.[a]

[a] Here we "tag" the second argument of $+_P$ with a superscript, denoting that it should be thought of as a "cotangent vector", though for us really $P' = P$. For a type-safe(r) formulation see [Cap23].

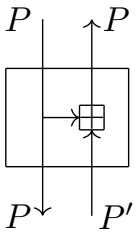


Figure 8.7: Gradient ascent

[1] Despite the fact that $P = P'$, here P' informally denotes *the tangent spoce* of P, to alleviate confusion.

Figure 8.6: Model reparameterised by basic gradient ascent (left) and a generic stateful optimiser (right).

Example 8.6 (Gradient ascent in **Smooth**)**.** In **Smooth** gradient ascent uses the monoid of P which takes the cotangent vector p' and adds it to the current point p.

Example 8.7 (Gradient update in $\mathbf{Poly}_{\mathbb{Z}_2}$)**.** In $\mathbf{Poly}_{\mathbb{Z}_2}$ gradient update will also performs "addition", but one implemented by XOR, following the implementation in [WZ21].

On the other hand, the commonly used gradient *descent* can only be captured in cartesian left-additive categories where the monoid structure on objects is additionally a group.[2]

Definition 8.8 (Gradient descent)**.** Gradient descent on P is a lens

$$\begin{pmatrix} \mathrm{id}_P \\ -_P \end{pmatrix} : \begin{pmatrix} P \\ P \end{pmatrix} \to \begin{pmatrix} P \\ P' \end{pmatrix}$$

where $-_P : (p, p') = p - p'$.

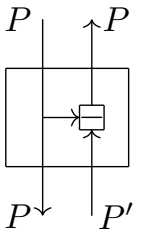


Figure 8.8: Gradient descent

In **Smooth** this instantiates to usual gradient descent. Gradient ascent in $\mathbf{Poly}_{\mathbb{Z}_2}$ is equal to gradient descent because XOR is its own inverse. Intuitively in $\mathbf{Poly}_{\mathbb{Z}_2}$ there is always only one direction we can move (other than staying still): its flipping the bit. Gradient descent and ascent are not usually seen as a lens — but they fit precisely into this picture that we are creating.

Other variants of gradient descent also fit naturally into this framework by allowing for additional input/output data with P. In particular, many of them keep track of the history of previous updates

[2]Since a homomorphism between groups needs to satisfy *less* equations than a monoid homomorphism, this means that such an assumption is a fairly easy one to add!

and use that to inform the next one. This is easy to model in our setup: instead of asking for a lens $\binom{P}{P} \to \binom{P}{P'}$, we ask instead for a lens $\binom{S\times P}{S\times P} \to \binom{P}{P'}$ where S is some "state" object. These turn out to be parametric lenses as well!

Definition 8.9 (Stateful gradient update)**.** A stateful gradient update of P consists of a choice of an object $S : \mathcal{C}$ (the state object) and a lens

$$\binom{S \times P}{S \times P} \to \binom{P}{P'}$$

Following the idea of drawing parameters in a different dimension, adding another higher-order "parameter" implies that stateful parameter update requires a three-dimensional graphical language in order to properly separate the "hyper-parameter"state object S from the parameter object S. This is a sensible direction to investigate (partially explored in [Gav21]), it is not the one we take in this thesis. Here we consider a stateful optimiser merely as just a lens whose domain is a product (Fig. 8.6, (right)).

Several well-known optimisers can be implemented this way. All of these reside in **Smooth**.

Example 8.10 (Momentum ([Pol64]))**.** In the momentum variant of gradient descent, we keep track of previous gradients used to update the parameter and use this information to inform how future updates should be computed. More precisely, set $S = P$ and fix $\gamma : \mathbb{R}$ (usually $\gamma > 0$). Then momentum is a lens

$$\binom{\pi_P}{U'} : \binom{P \times P}{P \times P} \to \binom{P}{P'}$$

with $U'(s, p, p') = (s', p+s')$ where $s' = -\gamma s+p'$. Note momentum recovers simple gradient update when $\gamma = 0$.

In both standard gradient ascent/descent and momentum our lens representation has a trivial forward part. This raises the question of whether lenses really capture the essence of optimisers. However, as soon as we move on to more complicated variants, having non-trivial forward part of the lens is important, and Nesterov momentum is a key example of this.

Example 8.11 (Nesterov momentum ([Nes93]))**.** In Nesterov momentum we make a small modification to the forward pass of the previous example. While in momentum the forward pass simply discards the accumulated state, Nesterov momentum computes a "lookahead" value by tewaking

the input parameter supplied to the network. More precisely, we again set $S = P$, fix $\gamma : \mathbb{R}$ (usually $\gamma > 0$) and define the **Nesterov momentum** lens

$$\begin{pmatrix} U \\ U' \end{pmatrix} : \begin{pmatrix} P \times P \\ P \times P \end{pmatrix} \to \begin{pmatrix} P \\ P' \end{pmatrix}$$

where $U(s, p) = p + \gamma s$ and U' as in the previous example.

Example 8.12 (Adagrad ([DHS11]))**.** Given any $\epsilon > 0$ and $\delta \sim 10^{-7}$, Adagrad is given by $S = P$ and a lens whose forward part is defined as $(g, p) \mapsto p$. The backward part is $(g, p, p') \mapsto (g', p + \frac{\epsilon}{\delta + \sqrt{g'}} \odot p')$ where $g' = g + p' \odot p'$ and $\odot$ is the elementwise (Hadamard) product. Unlike with other optimization algorithms where the learning rate is the same for all parameters, Adagrad divides the learning rate of each individual parameter with the square root of the past accumulated gradients.

Example 8.13 (ADAM ([KB15]))**.** Adaptive Moment Estimation is another method that computes adaptive learning rates for each parameter by storing exponentially decaying average of past gradients (m) and past squared gradients (v). For fixed $\beta_1, \beta_2 \in [0, 1)$, $\epsilon > 0$, and $\delta \sim 10^{-8}$, Adam is given by $S = P \times P$ with the lens whose forward part is $(m, v, p) \mapsto p$ and whose backward part is $(m, v, p, p') = (\widehat{m}', \widehat{v}', p + \frac{\epsilon}{\delta + \sqrt{\widehat{v}'}} \odot \widehat{m}')$, where

$$m' = \beta_1 m + (1 - \beta_1) p' \qquad \widehat{m}' = \frac{m'}{1 - \beta_1^t}$$

$$v' = \beta_2 v + (1 - \beta_2) p'^2 \qquad \widehat{v}' = \frac{v'}{1 - \beta_2^t}$$

We note that these are just a select few out of many more ([Kas22]) such as regularised gradient descent ([BLB17]) and Nesterov ADAM [Doz16]. The last example is notable because it suggests that "Nesterov" is a general principle for modifying the forward passes of optimisers.

Both optimisers and neural networks are parametric lenses! **Box 8.1.2**

Stateful optimisers such as Nesterov momentum (Example 8.11) can be seen as parametric lenses, whose parameters are states. Since we've seen that parameters of a parametric lens can be learned, is there any sense in which we can *learn an optimiser*? The work of [ADG+16] — appearing before any definitions of parametric lenses — confirmed this. Inspired by the ability to learn any parametrised map, the authors parameterise a stateful optimiser, and *learn* the gradient update function. Categorical foundations of such meta-learning is something we hope to thoroughly explore in future work.

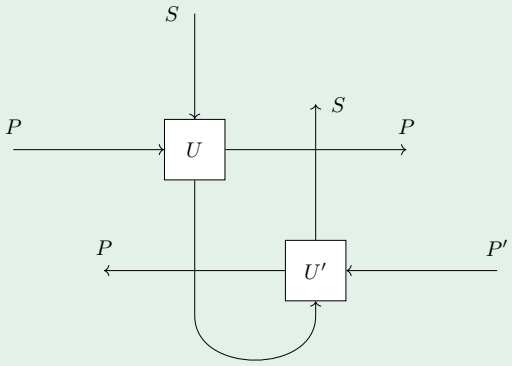


Figure 8.9: A stateful optimiser (Definition 8.9) can be seen as a parametric lens, hinting at the possibility that gradient updates can be learned. ([ADG+16])

Can we compose optimisers?

Even though not explicitly acknowledged in the literature, optimisers can be composed, and this composition plays an important role in settings where deep learning intersects with multivariable optimisation. In such settings we're interested in their *parallel* composition, therefore giving a positive answer to the above question.[3] Parallel composition of optimisers arises out of the fact that optimisers are lenses, and lenses have a monoidal product (Section 4.3.2). In such settings we might have an optimiser of two variables which descends in one direction, and ascends in the other one, for instance.

[3] One might wonder whether optimisers can be composed in sequence as well. The apparent sequential composability of optimisers is unfortunately an artefact of our limited view without dependent types.

Definition 8.14 (Gradient descent-ascent (GDA))**.** Given objects P and Q, gradient descent-ascent on $P \times Q$ is a lens

$$\begin{pmatrix} \mathrm{id}_{P\times Q} \\ \mathsf{gda} \end{pmatrix} : \begin{pmatrix} P \times Q \\ P \times Q \end{pmatrix} \to \begin{pmatrix} P \times Q \\ P \times Q \end{pmatrix}$$

where $\mathsf{gda}(p, q, p', q') = (p - p', q + q')$.

In **Smooth** this gives an optimiser which descends on P and ascends on Q. In $\mathbf{Poly}_{\mathbb{Z}_2}$ this map ends up computing the same function on both parameter spaces. This is something that ends up preventing us from modelling GANs in this setting (compare Example 8.19 where both positive and negative polarity of the optimiser map is needed).

When it comes to optimisers of two parameters, gradient descent-ascent is a particular type of an optimiser that is a product of two optimisers. But not all optimisers can be factored in such a way, much like a general monoidal product doesn't necessarily have to be cartesian. A good example of this is an optimiser on two parameters called *competitive gradient descent* ([SA19]). We don't explicitly define or use it in this thesis, instead inviting the reader to the aforementioned reference for more information.

Of course, these were examples of composition of non-stateful optimisers. For stateful ones the story becomes more complex, but appears to fully be captured by the already defined monoidal structure of **Para**. Fully defining and understanding the implications of these kinds of optimisers is an area of categorical machine learning ripe for research.

8.2 Supervised learning

We have now described all the main components of learning: neural networks, loss functions, learning rates and optimisers. We are ready to put the pieces together and describe supervised learning. Combining a model and a loss Fig. 8.1 with the state on the left side Fig. 8.2 and costate on the right we obtain a "scalar", i.e. a closed system: a parametric lens $\binom{1}{1} \to \binom{1}{1}$, drawn in Fig. 8.10.

Following Box 5.1.1 we can see that a scalar in $\mathbf{Para}(\mathbf{Lens}(\mathcal{C}))$ is simply a costate in $\mathbf{Lens}(\mathcal{C})$ which is via Box 4.4.1 isomorphic to a morphism in $\mathcal{C}$. What morphism is it? It is one of type $X \times P \times Y \to X' \times P' \times Y'$ whose implementation we still suggestively call update, as it is the backward map of this closed parametric lens. That is, given a model with a forward pass $f : X \times P \to Y$, a loss function $\mathsf{loss} : Y \times Y \to L$, a learning rate $\alpha : L \to L'$ the algebra of lens

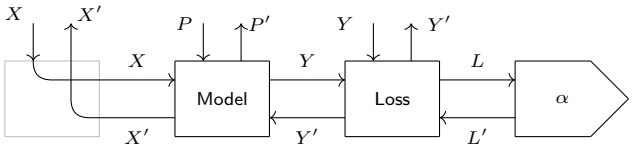


Figure 8.10: Closed parametric lens whose all inputs and outputs are now vertical wires.

composition defines update as

$$\begin{aligned}\mathsf{update}(x,p,y_t) = (x',p',y_t') \qquad \text{where} \qquad y_p &= f(x,p)\\ (y_p',y_t') &= R[\mathsf{loss}](y_p,y_t,\alpha(\mathsf{loss}(y_p,y_t)))\\ (x',p') &= R[f](x,p,y_p')\end{aligned}$$

In the next two subsections we will see how this closed system can be reparameterised with an optimiser on the parameter ports to describe supervised learning of parameters, but also how it can be reparameterised with an optimiser on *the input ports* describing *deep dreaming*.

8.2.1 Supervised learning of parameters given a loss

The most common type of learning performed on Fig. 8.10 is supervised learning of parameters. This is done by reparameterising it in the following manner. The parameter ports $\binom{P}{P'}$ are reparameterised by one of the (potentially stateful) optimisers described in Section 8.1.3, while the backward wires X' of inputs and Y' of labels are discarded. This finally gives us a complete picture of the system which performs one step of supervised learning of parameters (Fig. 8.11)

Fixing a particular optimiser $\binom{U}{U*} : \binom{S\times P}{S\times P} \to \binom{P}{P'}$ we again unpack the entire construction. By analogous arguments as before it now reduces to a map of type $X \times S \times P \times Y \to S \times P$ and unpacks to:

$$\begin{aligned}\mathsf{update}(x,s,p,y_t) = U^*(s,p,\overline{p}') \qquad \text{where} \qquad \overline{p} &= U(s,p)\\ y_p &= f(x,\overline{p})\\ (y_p',y_t') &= R[\mathsf{loss}](y_p,y_t,\alpha(\mathsf{loss}(y_p,y_t)))\\ (x',\overline{p}') &= R[f](x,\overline{p},y_p')\end{aligned} \tag{8.15}$$

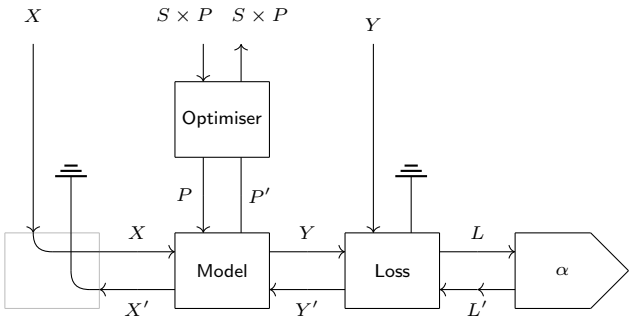


Figure 8.11: One step of supervised learning of parameters as a closed parametric lens. An animation of this supervised learning is available at this hyperlink.

While this formulation might seem daunting, we note that it just explicitly specifies the entire computation performed by this supervised learning system.[4] The variable $\overline{p}$ represents the parameter supplied to the network by the stateful gradient update rule (in many cases this is equal to p); y_p represents the prediction of the network (constrast this with y_t which represents the ground truth from the dataset). Variables with a tick $'$ represent changes: y'_p and y'_t are the changes on predictions and ground truth label respectively, while x' and p' are changes on the inputs and the parameters. Note that this entire update arises automatically out of the rule for lens composition: all we did was supply concrete lenses.

We justify and illustrate our approach on a series of case studies drawn from the machine learning literature, showing how in each case the parameters of our framework (in particular, loss functions and gradient updates) instantiate to familiar concepts. This presentation has the advantage of treating all these case studies unformly in terms of our basic constructs, highlighting their similarities and differences. We start in **Smooth** and fix some parametric map $(\mathbb{R}^p, f) : \mathbf{Para}(\mathbf{Smooth})(\mathbb{R}^x, \mathbb{R}^y)$ and the constant learning rate $\alpha : \mathbb{R}$ (Definition 8.2). We then vary the loss function and gradient update, seeing how the update map above reduces to many of the known cases.

Example 8.16 (Quadratic loss, basic gradient descent)**.** Fix the quadratic error (Example 7.19) as the loss map, and gradient descent (Definition 8.8) as the optimiser. Then the aforementioned update map simplifies. Since there is no state, its type reduces to $X \times P \times Y \to P$, and its

[4] It is notable that lenses forming the update in Eq. (8.15) were composed as optics. See Section 6.3.1.

implementation to

$$\mathsf{update}(x, p, y_t) = p - p' \qquad \text{where} \qquad (x', p') = R[f](x, p, \alpha \cdot (f(x, p) - y_t))$$

Note that α here is simply a constant, and due to the linearity of the reverse derivative we can slide the α from the costate into the gradient descent lens. Rewriting this update, and performing this sliding we obtain a closed form update step, resembling the way gradient descent with quadratic loss is usually formulated.

$$\mathsf{update}(x, p, y_t) = p - \alpha \cdot (R[f](x, p, f(x, p) - y_t)) \fatsemi \pi_1$$

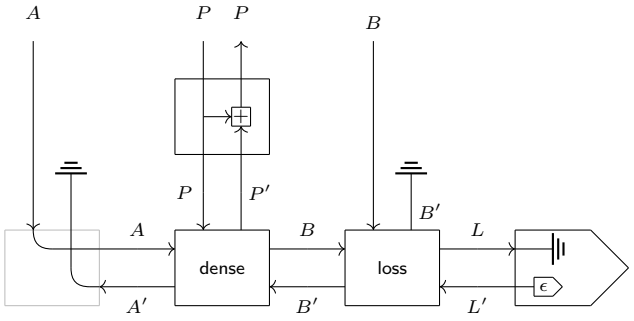


Figure 8.12: Closed parametric lens whose parameters are being learned.

For each parametric map $(\mathbb{R}^p, f) : \mathbf{Para}(\mathbf{Smooth})(\mathbb{R}^x, \mathbb{R}^y)$ this example gives us a variety of *regression* algorithms solved iteratively by gradient descent. If the map f is linear and $y = 1$, we recover simple linear regression with gradient descent. If the codomain y is multi-dimensional, i.e. we are predicting multiple scalars, then we recover multivariate linear regression. Of course, by changing the underlying parametric map we can model arbitrarily complex neural network architectures.

Example 8.17 (Softargmax cross-entropy, basic gradient descent)**.** Fix softargmax cross-entropy (Example 7.21) as the loss map and basic gradient descent (Definition 8.8) as the optimiser. Again the put map simplifies. The type reduces to $X \times P \times Y \to P$ and the implementation to

$$\mathsf{update}(x, p, y_t) = p - p' \qquad \text{where} \qquad (x', p') = R[f](x, p, \alpha \cdot (\mathsf{Softargmax}(f(x, p) - y_t))$$

The same rewriting performed on the previous example can be done here.

Example 8.18 (Mean squared-error, Nesterov momentum)**.** Fix the quadratic error (Example 7.19) as the loss map, and Nesterov momentum (Example 8.11) the optimiser. This time the update map does not have simplified type, as it is still $X \times S \times P \times Y \to S \times P$. Its implementation reduces to

$$\mathsf{update}(x, s, p, y_t) = (s', p + s') \qquad \text{where} \qquad \begin{aligned} \overline{p} &= p + \gamma s \\ (x', \overline{p}') &= R[f](x, \overline{p}, \alpha \dot{(} f(x, \overline{p}) - y_t)) \\ s' &= -\gamma s + \overline{p}' \end{aligned}$$

This example with Nesterov momentum differs in two key points from all the other ones: i) the optimiser is stateful, and ii) its forward map is not trivial. While many other optimisers are stateful, the non-triviality of the forward part here showcases the importance of lenses. They allow us to make precise the notion of computing a "lookahead" value for Nesterov momentum, something that is in practice usually handled in ad-hoc ways. Here the algebra of lens composition handles it naturally by using the forward map, a seemingly trivial, unused piece of data for previous optimisers.

Many kinds of systems that are traditionally considered unsupervised can be recast to their supervised form. One example is GANs (Section 7.5). By choosing $\mathsf{GAN}_{g,d}$ as the parametric map representing our supervised learning model, we can differentiate it as in Fig. 7.18, and, with the appropriate choice of a loss function, produce the learning system in the literature called *Wasserstein* GAN ([ACB17]).

Example 8.19 (GANs, Dot product, GDA)**.** Fix $\mathsf{GAN}_{g,d}$ as the parametric map (Definition 7.15), gradient descent-ascent (Definition 8.14) as the optimiser and dot product (Example 7.22) as the loss function. Then the update becomes a map of type $Z \times X \times P \times Q \times L \times L \to P \times Q$ and its implementation reduces to

$$\mathsf{update}(z, x, p, q, y_t) = (p - p', q + q') \qquad \text{where} \qquad (z', x', p', q') = R[\mathsf{GAN}_{g,d}](z, x, p, q, \alpha \cdot y_t)$$

We can further unpack the label $y_t = (y_{t_g}, y_{t_r})$[5] and via Eq. (7.16) express the update in terms of

[5] Whose interpretation here is more of a "mask", see Section 8.2.2.

reverse derivatives of g and d:

$$\mathsf{update}(z, x_r, p, q, y_{t_g}, y_{t_r}) = (p - p', q + q'_g + q'_r) \qquad \text{where} \qquad \begin{aligned}(x'_g, q'_g) &= R[d](g(z,p), q, \alpha \cdot y_{t_g}) \\ (z', p') &= R[g](z, p, x'_g) \\ (x'_r, q'_r) &= R[d](x_r, q, \alpha \cdot y_{t_r})\end{aligned}$$

This brings us to the last step, where by linearity of the backward pass we can extract α and components of y_t out:

$$\mathsf{update}(z, x_r, p, q, y_{t_g}, y_{t_r}) = (p - \alpha y_{t_g} p', q + \alpha(y_{t_g} q'_g + y_{t_r} q'_r)) \qquad \text{where} \quad \begin{aligned}(x'_g, q'_g) &= R[d](g(z,p), q, 1) \\ (z', p') &= R[g](z, p, x'_g) \\ (x'_r, q'_r) &= R[d](x_r, q, 1)\end{aligned}$$

The ultimate representation is a form in which it makes it possible to see how the update recovers that of Wasserstein GANs. The last missing piece is to note that the supervision labels y_t here are effectively "masks". Just like in standard supervised learning an input-output pair (x_i, y_i) consisted of an input value and a corresponding label which guided the direction in which the output $f(x_i, p)$ should've been improved, here the situation is the same. Given any latent vector z its corresponding "label" is the learning signal $y_{t_g} = 1$ which does not change anything in the update, effectively signaling to the generator's and discriminator's optimisers that they should descend (minimizing the assigned loss, making the image more realistic next time), and respectively ascend (maximizing the loss, becoming better at detecting when its input is a sample generated by the generator). On the other hand, given any real sample x_r its corresponding "label" is the learning signal $y_{t_r} = -1$ which signals to the discriminator's optimiser that it should do the opposite of what it usually does; it should descend, causing it to assign a lower loss value actual samples from the dataset. In other words, the input-output pairs are here always of the form $((z, x)_i, (1, -1)_i)$, making this in many ways GANs a *constantly* supervised model. Nonetheless, these different "forces" that pull the discriminator in different directions depending on the source of the input, coupled with the ever-changing generated inputs make GANs have intrinsically complex dynamics that are still being studied.

The fact that we were able to encode Wasserstein GAN in this form in our framework is a consequence of its simple formulation of its loss function, which is effectively given by subtraction

[ACB17, Theorem 3]. There are many things here left to future work. These include studying GANs outside of **Smooth**, studying GANs with different loss functions (such as the original paper ([GPAM+14]) which uses JS-divergence), and studying GANs with different optimisers (such as *competitive gradient descent* ([SA19])). A lot of these are predicated on a categorical framework that includes dependent types in its formalism (see the end of Section 7.5), and an integration of categorical frameworks of differentiation with those of probability, which would allow a refinement of the output type of GANs (and many other neural networks) into the type $D(n)$ if a probability distribution on $n : \mathbb{N}$ outputs.

We finish off these examples by moving to a different base category $\mathbf{Poly}_{\mathbb{Z}_2}$. This example shows that our framework describes learning in not just continuous, but discrete settings too. Again, we fix a parametric map $(\mathbb{Z}^p, f) : \mathbf{Poly}_{\mathbb{Z}_2}(\mathbb{Z}^x, \mathbb{Z}^y)$ but this time we fix the identity learning rate (Example 8.4) instead of a constant one.

Example 8.20 (Basic update in Boolean circuits)**.** Fix XOR as the loss map (Example 7.20), and gradient ascent (Example 8.7) as the optimiser. The type of the update map again simplifies to $X \times P \times Y \to P$. Its implementation reduces to

$$\mathsf{update}(x, p, y_t) = p + p' \qquad \text{where} \qquad (x', p') = R[f](x, p, f(x, p) + y_t)$$

8.2.2 Deep Dreaming: Supervised Learning of Inputs

We have seen that reparameterising the parameter port with gradient descent allows us to capture supervised learning of parameters. In this subsection we describe how reparameterising the *input port* provides us with a way to update the input datapoint — usually an image, used to elicit a particular intepretation. (Fig. 8.14) This is the idea behind the technique of deep dreaming, appearing in the literature in many forms ([DB16, MV15, NYC15, SVZ14]).

Deep dreaming is a technique which uses the parameters p of some trained classifier network to iteratively dream up, or amplify some features of a class y_i on a chosen input x. For example, if we start with an image of a landscape x_0, a label y_i of a "cat" and a parameter p of a sufficiently well-trained classifier, we can start performing "learning" as usual: computing the predicted class for the landscape x_0 for the network with parameters p, and then computing the distance between the prediction and our label of a cat y_i. When performing backpropagation, the respective changes computed for each layer tell us how the activations of that layer should have been changed to be

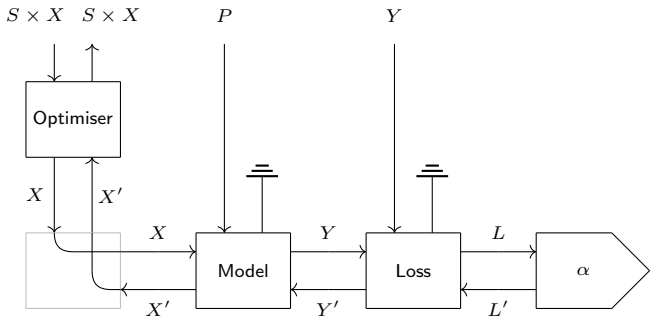


Figure 8.13: Deep dreaming: supervised learning of inputs.

more "cat" like. This includes the first (input) layer of the landscape x_0. Usually, we discard these changes and apply gradient update only to the parameters. In deep dreaming we *discard the parameter gradients* and apply gradient update to the input (Fig. 8.11). Gradient update here takes these changes and computes a new image x_1 which is the same image of the landscape, but changed slightly so to look more like whatever the network thinks a cat looks like. This is the essence of deep dreaming, where iteration of this process allows networks to dream up features and shapes on a particular chosen image [MOT15].

Just like in the previous subsection, we can write this deep dreaming system as a map in **Para**(**Lens**($\mathcal{C}$)) of type $\binom{1}{1} \to \binom{1}{1}$, and in an analogous way show it reduces to a map of type $S \times X \times P \times Y \to S \times X$ whose implementation is

$$\mathsf{update}(s, x, p, y_i) = U^*(s, x, \overline{x}') \qquad \text{where} \qquad \begin{aligned} \overline{x} &= U(s, x) \\ y_p &= f(\overline{x}, p) \\ (y_p', y_i') &= R[\mathsf{loss}](y_p, y_i, \alpha(\mathsf{loss}(y_p, y_i))) \\ (\overline{x}', p') &= R[f](\overline{x}, p, y_p') \end{aligned}$$

We note that deep dreaming is usually presented without any loss function as a maximisation of a particular activation in the last layer of the network output [SVZ14, Sec. 2]. As it is a maximisation, it is performed using gradient ascent, as opposed to gradient descent. However, this is just a special case of our framework where the loss function is the dot product (Example 7.22). The choice of the particular activation is encoded as a one-hot vector, and the loss function in that case essentially masks the network output, leaving active only the particular chosen activation. We

Figure 8.14: An image of a dog with features enhanced through the process of deep dreaming. Image taken from an official TensorFlow tutorial.

explicitly unpack this in the following example inside the base category **Smooth**, where again we fix a parametric map $(\mathbb{R}^p, f) : \mathbf{Para}(\mathbf{Smooth})(\mathbb{R}^x, \mathbb{R}^y)$.

Example 8.21 (Deep dreaming, dot product, basic gradient update)**.** Fix the dot product loss (Example 7.22), and gradient ascent (Example 8.6) as the optimiser. Then the above update map simplifies to $X \times P \times Y \to X$ as there is no state. Its implementation reduces to

$$\mathsf{update}(x, p, y_i) = x + x' \qquad \text{where} \qquad (x', p') = R[f](x, p, \alpha \cdot y_i)$$

Following Example 8.16, this update can be rewritten as

$$\mathsf{update}(x, p, y_i) = x + \alpha \cdot (R[f](x, p, y_i) \fatsemi \pi_1)$$

making a few things apparent. This update does not depend on the prediction $f(x, p)$: no matter what the network has predicted, the goal is always to maximize particular activations. Which activations? The ones chosen by y_i. When y_i is a one-hot vector, this picks out the activation of

just one class to maximize, which is often done in practice.

While we present only the most basic image, there is plenty of room left for exploration. The work of [SVZ14, Section 2.] adds an extra regularization term to the image. In general, the neural network f is sometimes changed to copy a number of internal activations which are then exposed on the output layer. Maximizing all these activations often produces more visually appealing results. In the literature we did not find an example which uses the softmax-cross entropy (Example 7.21) as a loss function in deep dreaming, which seems like the more natural choice in this setting. Furthermore, while deep dreaming commonly uses basic gradient descent, there is nothing preventing the use of any of the optimiser lenses discussed in the previous section, or even doing deep dreaming in the context of Boolean circuits. Lastly, learning iteration which was described in at the end of previous subsection can be modelled here in an analogous way.

8.3 Supervised learning in the literature

Categorical models of supervised learning in the literature are scarce. The seminal and by far the most important paper to mention is "Backprop as Functor" [FST21].

This was the first paper to explicitly tackle many of the neural network components using category theory at the same time: parameterisation, backpropagation, loss functions, and gradient descent. It was also the first paper to use string diagrams to depict the information flow in a composite neural network. Being seminal, it spawned a lot of subsequent research, and demonstrated viability of category theory as a modelling tool for deep learning in a manner that was not considered before.

To study parameterisation the authors defined the **Para** construction, which is to me the first known name of this construction. They also defined an analogue of **Para**(**Lens**(**Set**)) — a category they called **Learn**. Both of these were defined as categories and not bicategories.[6] Their **Para** definition is specialised to the self-cartesian action of **Smooth**, and arises out of the quotienting of the bicategory by isomorphism classes (Box 3.1.2). The category **Learn** (whose morphisms they call *learners*) could be seen as a special case of **Para**(**Lens**(**Set**)), also quotiented out by isomorphism classes. But there is an important distinction between Iso_*(**Para**(**Lens**(**Set**))) and **Learn**: the latter arises out of a quotient by a different kind of 2-cells than those of **Para**(**Lens**(**Set**)). The 2-cells in **Para**(**Lens**(**Set**)) are lenses, but those in **Learn** are *charts* (Example D.16).

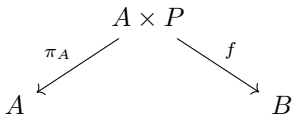
[6] Though, it was acknowledged that the bicategorical setting is more natural

They defined backpropagation as a functor $L : \mathbf{Para} \to \mathbf{Learn}$ ([FST21, Theorem III.2]), and used it to model one update step of supervised learning. However, the definition of L contains a number of problems. Firstly, it is not well-defined (see Remark E.21). Intuitively, since they do not deal with explicit 2-cells (but equivalence classes thereof) it was not noticed that this functor is not well-defined on these equivalence classes. This is precisely because the 2-cells in **Learn** are not those that arise out of 2-cells of **Para**. Compare this with the formulation in this thesis, where reverse differential structure is a functor $\mathbf{R}_{\mathcal{C}} : \mathcal{C} \to \mathbf{Lens}_A(\mathcal{C})$, and its parametric variant arises by application of **Para** to it (Prop. 5.9), ensuring well-definedness by construction.

This definition is also tied to the setting of Euclidean spaces, as they do not use any of the existing categorical frameworks for differentiation such as tangent or differential categories (Section 6.4). Since we do, we can thus go well beyond them in terms of examples — their example of smooth functions is just one of the examples we cover in this thesis.

Thirdly, their functor does not augment the forward map with just its reverse derivative, but also with the loss map and the optimiser as well. The functoriality condition of this functor then adds extra conditions on the partial derivatives of the loss function: it's required to be invertible in the 2$^{\text{nd}}$ variable. This constraint was not justified, nor it is a constraint that appears in deep learning practice. For instance, it precludes usage or the L1 loss or the Hinge loss as loss functions. When it comes to the optimiser, only simple gradient descent (Definition 8.8) is considered, and it is not clear whether abstracting away the optimiser would add additional conditions on its backward pass.

Related work to this is *Lenses and Learners* ([FJ19]) which shows that learners can be seen as asymmetric lenses. This arises because every parametric morphism $f : A \times P \to B$ can be seen as a span whose left leg is a projection (Eq. (8.22)).



(8.22)

We belive studying **Para** in its dependent variant (Section 3.3) might shed light on the connection of this work to this thesis. Lenses and Learners are related to the formalisation of supervised learning in terms of Delta lenses (Section 4.5) that appeared around the same time ([Dis20]) whose relationship to the work in this thesis is not clear.

The work that provides a prescient categorical characterisation of gradient based learning, solv-

ing many of the issues outlined above is *Dioptics* [DL19]. Despite containing only sketches and conjectures, in hindsight many of them seem to be the right ones. In addition to formulations of differentiation with covectors described in Section 6.4, this is the first place I have personally seen **Para** written using two dimensional notation and the three-legged shape of neural networks. It is also the first work to provide a full picture of supervised learning ([Dal19, Slide 28]) with concepts of neural network, optimiser, and the loss function conceptually separated on two axes, each of them bidirectional. Dioptics did not consider the loss function as a parametric map, but did break down the optimiser box into two components: the one given by the inner product on the tangent vector space, and the one given by integrating the produced tangent vectors along a geodesic and evaluating it at 1. Like this thesis, it also assumed a non-dependently typed setting (i.e. trivialisable bundles), but on the other hand, its main definition [DL19, Def. 3.0.1] is given in terms of a coend that trivialises in many cases of interest (see Box 3.1.2). The author does acknowledges that the coend definition may not be the right abstraction. As in Backprop as Functor, the analogous functor to **Para** $\to$ **Learn** is also defined by hand [DL19, Conjecture 4.2.1] and not factored through the **Para** construction as we do in this thesis (Prop. 5.9).

Part III

Conclusions

Chapter 9

To boldly go

> Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.
>
> Douglas Hoftstadter

At the end of spring of 2018, after I pointed out some of the issues of "Backprop as Functor" ([FST21]) to one of its coauthors David Spivak (articulated in a much less clear manner than in Section 8.3), he acknowledged them and enthusiastically told me "This might be a good summer project to work on.". In my infinite wisdom I thought "There is no way this takes that long". Four years later, here I am, writing the conclusion of my PhD thesis on this matter.

It has been a fascinating ride. In this thesis I've described what I believe to be the start of a good foundation for studying deep learning, and beyond. I've taken great care to distill as many conceptual moving parts as I could, but some things are still stuck together. Organised from the most immediate to the most ambitious, the directions for further research are outlined below.

- **Locally graded categories.** I expect locally graded categories to subsume actegories as a foundation of parameterisation. They unify both actegories and enriched categories, providing a general foundation that captures both internal and external parameterisation. This consequently informs the chapter on bidirectionality, as well as the chapter on differentiation[1].

[1] I note a resemblance of hom-sets of diffeological spaces (mentioned in Section 6.4 of the chapter on backpropa-

Some preliminary work on this topic is in Appendix B. Locally graded categories give rise to a significantly different way of thinking about some of the content in this thesis, and I'm excited to explore it.

- **Complete characterisation of differentiation.** This thesis contains three assumptions in modelling derivatives.
  - I assume that the type of the backward pass (i.e. the tangent space of a particular point) of some space is isomorphic to the space itself (i.e. $X \cong T_x(X)$, for all $x : X$). This is the case for euclidean spaces and rings of polynomials, but not true for manifolds and many other interesting structures.
  - I assume that the cotangent vector space (i.e. the space of linear functionals) at a point is isomorphic to the tangent space at that point (i.e. $T_x(X) \cong T_x^*(X)$ for all $x : X$). This is the case if our vector space comes equipped with a non-degenerate bilinear form, but is an assumption that prevents us form establishing a formal connection to settings where this isomorphism can't even be stated: for instance that of game theory.
  - I assume the base category has *products*. This models models only *deterministic* systems, and precludes us from dealing with settings where we do not have at our disposal the ability to copy and delete information such as probabilistic or quantum machine learning.

  Relaxing the first two assumptions leads to the work in [Cap23][2], and points to a generalised theory of "differentiation" which exhibits fundamental connections to game theory. Relaxing the last assumption leads to the work of [TYdF21] describing current categorical approaches to quantum machine learning. I hypothesise that all three generalisations in tandem can pave the way to a useful foundation of generalised theory of differentiation which covers a a wide-variety of learning schemes — from reinforcement learning, game theory, over bayesian learning to even the lesser known automata learning ([JS14, US20]).

- **Meta-learning**. I believe that findings from Section 8.1.3 about the resemblance of the shape of optimisers and neural networks themselves might inform future work on meta-learning [ADG+16, HAMS22, FAL17]. In [Gav21] a question is posed: how much of the theoretical foundation of parametric optics can be used to formalise *learning to learn by gradient descent*

gation) and hom-objects of locally graded categories.

[2] Though, in its current form at the expense of an operational view on differentiation.

by gradient descent ([ADG+16])? This necessitates more precise categorical semantics of optimisation, and we believe that following up on progress in [Shi22, Hin12] can make that happen.

- **Taking dependent types seriously.** The *types of outputs* of many processes we study are dependent on the *values of their inputs*. Dependent type theory is the formal theory that studies this fundamental, and incredibly expressive idea. It allows us to treat *proofs* about our systems in the same manner we treat the systems themselves: as data which can be plugged into functions as inputs, or produced by functions as outputs.[3] This paves the way for the construction of neural reasoning systems which learn to output not just arrays of numbers, but formal *proofs*. This enables more rigor and reliability, consequently unlocking the usage of deep learning in new markets and sectors where this reliability is crucial. Most of the concepts in this thesis can be made *dependent*, and I expect the complete theory of categorical deep learning to become far simpler by doing so.

- **New regulatory paradigms.** Current regulatory practices for AI are mostly *retroactive*: they are focused on explaining decisions a model has already made. This is because there is no formal language to express the kinds of reasoning we'd proactively want to uphold a model to. For instance, most unifying approaches to the theory of architectures of deep learning — such as Geometric Deep Learning ([BBCV21]) — give us a mathematical theory of equivariance and invariance of deep learning models. This theory enables regulatory agencies to construct falsifiable tests for invariance to protected classes such as race and gender in large-scale deployed systems. But geometric deep learning inherently deals only with explainability and formal guarantees of models with respect to simple kinds of transformations: those given given by the algebraic structure of a group, such as translation, rotation, and scaling. Because of the group constraints, all of these transformations are required to be invertible, meaning geometric deep learning cannot handle recursion, transition functions of automata/dynamical systems, aggregation steps of a dynamic programming algorithm, or general programs which write to or read from memory. Geometric deep learning thus deals with transformations which are inherently *geometric*, as opposed to *algebraic* in nature. As outlined in Section 7.3 and my forthcoming work, category theory provides us with a theory of generalised equivariance, and

[3] A particular reason why I find dependent types useful is because they incentivise us to be cognisant of the arrow of time, necessitating the tracking of the chain of causal dependencies. This sometimes — almost if by magic — prevents unwanted states or consequences *from even being expressible*.

the theory of initial algebras with formal specification of what it means to reason *inductively*. In the long run I envision this enabling the construction of falsifiable tests of deep learning models with respect to more than just invertible actions such as translations, but also with respect to *the kinds of reasoning* these models can use when coming to their conclusions. What if we could explicitly decide whether a deep learning model gets to use particular kinds of inductive arguments, or particular kinds of logics in its reasoning? Can we end the era of unstructured models by making structured as capable as the unstructured ones, but verifiably so?

The last direction is the one I am the most excited about. Category theory has given me and many other people a unique vantage point to understand the world through: a language so robust and compositional, that the distinction between different scientific fields almost becomes erased. Can we imbue our deep learning models with these insights, and enable them to model the world using categories? This, in many ways, is the north star of deep learning: construction of models that reason in structured ways. What better way to do that than to use the theory of structure: category theory?

Appendix A

Monoidal and enriched categories

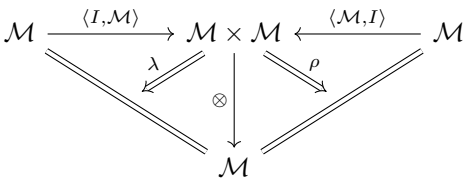
Monoidal categories

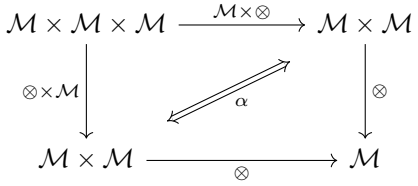
In this thesis, monoidal, braided monoidal, and symmetric monoidal categories play a central role.

Definition A.1 (Monoidal category (compare [JYJY21, Def. 12.1]))**.** Let $\mathcal{M}$ be a category. We call $\mathcal{M}$ a **monoidal category** if it comes equipped with

- A functor $\otimes : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ called *the monoidal product*;
- An object $I : \mathcal{M}$ called *the monoidal unit*;

and following three natural isomorphisms, called the *left unitor*, *right unitor* and *associator*, respectively:





(A.2)

whose components at each $C, D, E : \mathcal{M}$ are explicitly the maps

$$\lambda_C : I \otimes C \xrightarrow{\cong} C \quad , \quad \rho_C : C \otimes I \xrightarrow{\cong} C \quad \text{and} \quad \alpha_{C,D,E} : (C \otimes D) \otimes E \xrightarrow{\cong} C \otimes (D \otimes E)$$

satisfying the following coherence laws.

- **The pentagon identity.** This law has to hold for all $C, D, E, F : \mathcal{M}$. It tells us that, starting with all-left-association $((C \otimes D) \otimes E) \otimes F$, all the different ways of rebracketing to all-right association $C \otimes (D \otimes (E \otimes F))$ are the same. Equation (A.3) expresses this as a commutative diagram.

$$\begin{array}{ccc} & ((C \otimes D) \otimes E) \otimes F & \\ {\scriptstyle \alpha_{C,D,E\otimes F}} \swarrow & & \searrow {\scriptstyle \alpha_{C\otimes D,E,F}} \\ (C \otimes (D \otimes E)) \otimes F & & (C \otimes D) \otimes (E \otimes F) \\ \downarrow {\scriptstyle \alpha_{C,D\otimes E,F}} & & \downarrow {\scriptstyle \alpha_{C,D,E\otimes F}} \\ C \otimes ((D \otimes E) \otimes F) & \xrightarrow[C\otimes\alpha_{D,E,F}]{} & C \otimes (D \otimes (E \otimes F)) \end{array} \tag{A.3}$$

- **Triangle identity.** These have to hold for all $A, B : \mathcal{M}$. Analogously to above, Eq. (A.4) tells us that any two ways of getting rid of I in $(A \otimes I) \otimes B$ are the same.

$$\begin{array}{ccc} (A \otimes I) \otimes B & \xrightarrow{\alpha_{A,I,B}} & A \otimes (I \otimes B) \\ & {\scriptstyle \rho_A\otimes B} \searrow \quad \swarrow {\scriptstyle A\otimes\lambda_B} & \\ & A \otimes B & \end{array} \tag{A.4}$$

This definition, especially the coherence conditions might seem a bit contrived. We note that they arises out of the categorification[1] of the concept of a monoid in the monoidal category $(\mathbf{Set}, \times, 1)$ of sets and functions to a *pseudomonoid* ([DS97, Sec. 3]) in the monoidal 2-category $(\mathbf{Cat}, \times, \mathbf{1})$ of categories and functors. This pseudomonoid comes equipped with pasting diagrams that have a geometric interpretation, which give rise to the above pentagonator and triangle identities. This will be important when we study braided and symmetric monoidal categories, as their coherences can analogously be justified.

Notation A.5. We often refer to the monoidal category $(\mathcal{M}, \otimes, I, \alpha, \lambda, \rho)$ as simply $(\mathcal{M}, \otimes, I)$, or even just $\mathcal{M}$, when the monoidal structure is clear from context.

Morphisms of monoidal categories are called lax monoidal functors.

Definition A.6 (Lax monoidal functor ([JYJY21, Def 1.2.14]))**.** Let $(\mathcal{M}, \otimes, I)$ and $(\mathcal{N}, \otimes', I')$ be monoidal categories. A functor $F : \mathcal{M} \to \mathcal{N}$ is said to be **lax monoidal** if it comes equipped with:

[1] Taken to mean *vertical* categorification

- A natural transformation $\begin{array}{ccc} \mathcal{M}\times\mathcal{M} & \xrightarrow{F\times F} & \mathcal{N}\times\mathcal{N} \\ {\scriptstyle\otimes}\downarrow & \swarrow{\scriptstyle\mu} & \downarrow{\scriptstyle\otimes'} \\ \mathcal{M} & \xrightarrow[F]{} & \mathcal{N} \end{array}$ whose component at every $X, Y : \mathcal{M}$ is a morphism $\mu_{X,Y} : F(X) \otimes' F(Y) \to F(X \otimes Y)$ in $\mathcal{N}$;
- A natural transformation $\begin{array}{ccc} \mathbf{1} & & \\ {\scriptstyle I}\downarrow & \searrow^{I'} & \\ \mathcal{M} & \xrightarrow[F]{} & \mathcal{N} \end{array}$ which to the unique object $\bullet$ of $\mathbf{1}$ assigns a morphism $\epsilon_\bullet : I' \to F(I)$ in $\mathcal{N}$

such that the following coherence laws are satisfied.

- **Associativity.** This diagram has to commute for every $X, Y, Z : \mathcal{M}$.

$$\begin{array}{ccccc} (F(X)\otimes' F(Y))\otimes' F(Z) & \xrightarrow{\mu_{X,Y}\otimes' F(Z)} & F(X\otimes Y)\otimes' F(Z) & \xrightarrow{\mu_{X\otimes Y,Z}} & F((X\otimes Y)\otimes Z) \\ {\scriptstyle\alpha'_{F(X),F(Y),F(Z)}}\downarrow & & & & \downarrow{\scriptstyle F(\alpha_{X,Y,Z})} \\ F(X)\otimes'(F(Y)\otimes' F(Z)) & \xrightarrow[F(X)\otimes'\mu_{Y,Z}]{} & F(X)\otimes' F(Y\otimes Z) & \xrightarrow[\mu_{X,Y\otimes Z}]{} & F(X\otimes(Y\otimes Z)) \end{array}$$

- **Left and right unitality.** These diagrams have to commute for every $X : \mathcal{M}$.

$$\begin{array}{ccc} I'\otimes' F(X) & \xrightarrow{\lambda'_{F(X)}} & F(X) \\ {\scriptstyle\epsilon\otimes' F(X)}\downarrow & & \uparrow{\scriptstyle F(\lambda_X)} \\ F(I)\otimes' F(X) & \xrightarrow[\mu_{I,X}]{} & F(I\otimes X) \end{array} \qquad \begin{array}{ccc} F(X)\otimes' I' & \xrightarrow{\rho'_{F(X)}} & F(X) \\ {\scriptstyle F(X)\otimes'\epsilon}\downarrow & & \uparrow{\scriptstyle F(\rho_X)} \\ F(X)\otimes' F(I) & \xrightarrow[\mu_{X,I}]{} & F(X\otimes I) \end{array}$$

If μ and ϵ are natural isomorphisms, we call F a **strong monoidal functor**. If they are identities, we call it a **strict monoidal functor**.

Monoidal categories and strong monoidal functors form a category **MonCat**. Next, we define the *reverse* of a monoidal category, and show how it can be used to define the notion of a *braided monoidal category*.

Definition A.7 (Reversed monoidal category, [JYJY21, Example 1.2.9.])**.** Given a category $\mathcal{M}$ with a monoidal structure $(\otimes, I, \alpha, \lambda, \rho)$, we define the **reversed monoidal structure** on $\mathcal{M}$ as the tuple $(\otimes^{\mathsf{rev}}, I, \alpha^{-1}, \rho, \lambda)$ where $\otimes^{\mathsf{rev}} := \boxed{\mathcal{M}\times\mathcal{M} \xrightarrow{\mathsf{swap}_{\mathcal{M},\mathcal{M}}} \mathcal{M}\times\mathcal{M} \xrightarrow{\otimes} \mathcal{M}}$.[2] We often label the entirety of this structure just as $\mathcal{M}^{\mathsf{rev}}$.

[2] Here $\mathsf{swap}_{\mathcal{C},\mathcal{D}} : \mathcal{C}\times\mathcal{D} \to \mathcal{D}\times\mathcal{C}$ is a functor that swaps the order of the categories in a product.

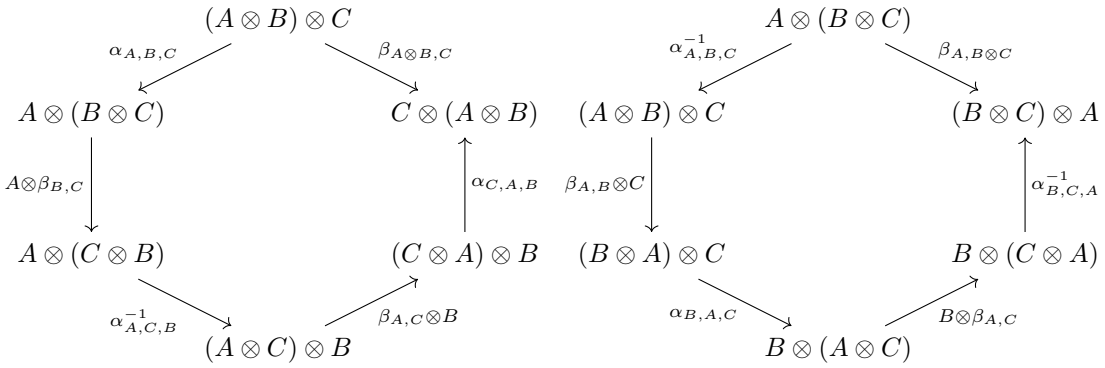
Definition A.8 (Braided monoidal category (compare [JYJY21, Def. 1.2.34]))**.** Let $(\mathcal{M}, \otimes, I, \alpha, \lambda, \rho)$ be a monoidal category. We call it a **braided monoidal category** if it comes equipped with a natural isomorphism

$$\mathcal{M} \times \mathcal{M} \;\overset{\otimes}{\underset{\otimes^{\mathsf{rev}}}{\Rightarrow}}\; \mathcal{M} \qquad \beta$$

whose component at every $X, Y : \mathcal{M}$ is explicitly the map

$$\beta_{X,Y} : X \otimes Y \to Y \otimes X$$

making the following two diagrams (called the *hexagon equations*) commute:



(A.9)

The hexagon equations tell us that braiding a monoidal product of objects is the same as braiding the objects individually one by one.

We have seen that monoidal categories are pseudomonoids in $(\mathbf{Cat}, \times, \mathbf{1})$. Interestingly, it turns out that pseudomonoids in $(\mathbf{MonCat}, \times, \mathbf{1})$ are precisely braided monoidal categories! For a full formal account of this deep fact about monoidal categories we refer the interested reader to [JS86], here just presenting the high-level intuition.

Since a monoidal category $(\mathcal{M}, \otimes, I)$ is an object of **MonCat**, a pseudomonoid structure on it necessitates the provision of strong monoidal functors $\mathcal{M} \times \mathcal{M} \to \mathcal{M}$ and $\mathbf{1} \to \mathcal{M}$ (and the corresponding unitors and associators, which are required to be monoidal natural transformations). Since $\mathcal{M}$ is monoidal, we have suitable candidates for both: functors $\otimes$ and $\ulcorner I \urcorner$. The latter can routinely be checked to already be strong monoidal, but there are some subtleties with the former.

Inducing strong monoidal structure on $\otimes$ necessitates the provision of the laxator and unitor natural isomorphisms. The unitor can also trivially be provided, but we need to take special care with the laxator of $\otimes$. This is a natural transformation that is in the aforementioned reference and [Kel82, p. 12] called the "middle-four interchange". It contains the data needed to define braiding ([JS86, Prop. 3]), and conversely, given a braided monoidal category, one can define the corresponding middle-four interchange [JS86, Prop. 2]. In the newer literature (such as [CGLP22, p. 12]), the "middle-four interchange" is often just referred to as the interchanger, which is the terminology we adopt in this thesis.

Definition A.10 (Interchanger (compare [CGLP22, p. 12] and [Kel82, p. 12]))**.** In any braided monoidal category $(\mathcal{M}, \otimes, I)$, for all $A, B, C, D : \mathcal{M}$ we can form **the interchanger**, a natural isomorphism $c_{A,B,C,D} : A \otimes B \otimes C \otimes D \to A \otimes C \otimes B \otimes D$ defined as the composite

$$c_{A,B,C,D} \coloneqq \boxed{A \otimes B \otimes C \otimes D \xrightarrow{A \otimes \beta_{B,C} \otimes D} A \otimes C \otimes B \otimes D}$$

Here we have omitted the associators, trusting it is clear how to formally account for them.[3] If all the underlying objects are given by the same object A, we will write c_A for this interchanger.

Here the interchanger is not to be confused with the interchange law, a property that holds in any monoidal category (not necessarily braided). When the monoidal categories are braided, it is possible for a morphism between them to additionally preserve this braided structure. These are called braided monoidal functors, and are relevant to us for studying reparameterisations of monoidal actegories, crucial in defining weighted optics (see Section 4.2.1 and Definition 4.14).

Definition A.11 (Braided monoidal functor (compare [JYJY21, Def. 1.2.30]))**.** Let $(\mathcal{M}, \otimes, I)$ and $(\mathcal{N}, \otimes', I')$ be braided monoidal categories. A lax monoidal functor $(F, \phi, \epsilon) : \mathcal{M} \to \mathcal{N}$ is **braided monoidal** if for all $X, Y : \mathcal{M}$ following diagram commutes:

$$\begin{array}{ccc} F(X) \otimes' F(Y) & \xrightarrow{\beta^{\mathcal{N}}_{F(X),F(Y)}} & F(Y) \otimes' F(X) \\ \downarrow{\scriptstyle \phi^{\mathcal{M}}_{X,Y}} & & \downarrow{\scriptstyle \mu^{\mathcal{N}}_{Y,X}} \\ F(X \otimes Y) & \xrightarrow[F(\beta^{\mathcal{M}}_{X,Y})]{} & F(Y \otimes X) \end{array} \tag{A.12}$$

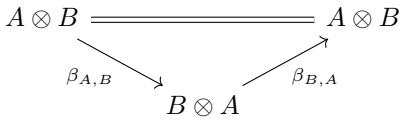
[3] Otherwise, see [JS86, Prop. 2]

We call the braided monoidal functor **strong** (resp. **strict**) if the lax monoidal functor (F, ϕ, ϵ) is strong (resp. strict).

Braided monoidal categories and strong braided monoidal functors also form a category. We can now continue the pattern and ask: if pseudomonoids in $(\mathbf{Cat}, \times, \mathbf{1})$ are monoidal categories, pseudomonoids in $(\mathbf{MonCat}, \times, \mathbf{1})$ are braided monoidal categories, then what are pseudomonoids in this newly defined category of braided monoidal categories and strong braided monoidal functors?

The answer is *symmetric monoidal categories*. We give an explicit definition.

Definition A.13 (Symmetric monoidal category (compare [JYJY21, Def. 1.2.24]))**.** Let $(\mathcal{M}, \otimes, I, \beta)$ be a braided monoidal category. We call it a **symmetric monoidal category** if braiding twice is the same as not doing anything at all; that is, if the following diagram commutes:



When $\mathcal{M}$ is symmetric, then either one of the hexagon identities is redundant ([JS86, p. 2]).

Remark A.14. While many braided monoidal categories are in practice symmetric, this is not always the case. A notable exception is the monoidal product used to define a class of optics called *affine traversals* (see [CG23, Prop. 5.2.7] and [CEG+22, Prop. 3.25]).

Symmetric monoidal functors, i.e. morphisms between symmetric monoidal categories are, interestingly, still just braided monoidal functors, and no extra condition is needed here.

And lastly, if we try iterating the previous process, and unpack what pseudomonoids in the 2-category of symmetric monoidal categories and symmetric monoidal functors is, we would get what we had already: a symmetric monoidal category. This curiousity is a part of the general periodic table of higher categories, and we refer the reader to [BS10, Sec. 2.1].

Enriched categories

Monoidal categories allow us to define the notion of an *enriched* category, a generalisation of a category where morphisms between objects need not form a set.

Definition A.15 (Enriched category (compare [Kel82, Sec. 1.2] or [JYJY21, Def. 1.3.1]))**.** Let $(\mathcal{V}, \otimes, I, \alpha, \lambda, \rho)$ be a monoidal category. Then a **$\mathcal{V}$-enriched category** $\mathcal{C}$ consists of the following data.

- A set $\mathsf{Ob}(\mathcal{C})$ of *objects*. We will often denote this set with just $\mathcal{C}$, trusting the meaning is clear from the context;
- For every $A, B : \mathcal{C}$ an object of $\mathcal{V}$ denoted as $\mathcal{C}(A, B)$. We think of it as the *hom-object*, or *object of morphisms* from A to B;
- For every $A : \mathcal{C}$ a morphism of type $I \to \mathcal{C}(A, A)$ in $\mathcal{V}$ denoted as $\lceil \mathrm{id}_A \rceil$ and called the *name*[4] of the identity morphism on A;
- For every $A, B, C : \mathcal{C}$, a morphism of type $\mathcal{C}(A, B) \otimes \mathcal{C}(B, C) \to \mathcal{C}(A, C)$ in $\mathcal{V}$ denoted as $\fatsemi_{A,B,C}$ and called *the composition*.

This data has to satisfy the following coherence conditions expressed as commutative diagrams in $\mathcal{V}$. The first one (Eq. (A.16)) which is defined for all $A, B, C, D : \mathcal{C}$ ensures the composition is associative. The second one (Eq. (A.17)), defined for all $A, B : \mathcal{C}$ ensures that it is unital from both left and right side.

$$\begin{array}{ccccc}
(\mathcal{C}(A,B) \otimes \mathcal{C}(B,C)) \otimes \mathcal{C}(C,D) & \xrightarrow{\fatsemi_{A,B,C} \otimes \mathcal{C}(C,D)} & \mathcal{C}(A,C) \otimes \mathcal{C}(C,D) & \xrightarrow{\fatsemi_{A,C,D}} & \mathcal{C}(A,D) \\
\Big\downarrow{\scriptstyle \alpha_{\mathcal{C}(A,B),\mathcal{C}(B,C),\mathcal{C}(C,D)}} & & & & \Big\| \\
\mathcal{C}(A,B) \otimes (\mathcal{C}(B,C) \otimes \mathcal{C}(C,D)) & \xrightarrow[\mathcal{C}(A,B) \otimes \fatsemi_{B,C,D}]{} & \mathcal{C}(A,B) \otimes \mathcal{C}(B,D) & \xrightarrow[\fatsemi_{A,B,D}]{} & \mathcal{C}(A,D)
\end{array} \tag{A.16}$$

$$\begin{array}{ccc}
I \otimes \mathcal{C}(A,B) & & \mathcal{C}(A,B) \otimes I \\
\Big\downarrow{\scriptstyle \lceil \mathrm{id}_A \rceil \otimes \mathcal{C}(A,B)} \quad \searrow^{\lambda_{\mathcal{C}(A,B)}} & & \swarrow_{\rho_{\mathcal{C}(A,B)}} \quad \Big\downarrow{\scriptstyle \mathcal{C}(A,B) \otimes \lceil \mathrm{id}_B \rceil} \\
\mathcal{C}(A,A) \otimes \mathcal{C}(A,B) \xrightarrow[\fatsemi_{A,A,B}]{} \mathcal{C}(A,B) & & \mathcal{C}(A,B) \xleftarrow[\fatsemi_{A,B,B}]{} \mathcal{C}(A,B) \otimes \mathcal{C}(B,B)
\end{array} \tag{A.17}$$

Notation A.18. In the nomenclature "$\mathcal{V}$-enriched category $\mathcal{C}$"(or simply a "$\mathcal{V}$-category $\mathcal{C}$") we trust that the monoidal structure of $\mathcal{V}$ is clear from the context. When multiple enriched categories

[4] It's called the *name* of identity because $\lceil \mathrm{id}_A \rceil$ is not the identity morphism, but instead it "picks it out", or "names it" in $\mathcal{C}(A, A)$.

are in scope, we might superscript the composition ($\fatsemi$) and identity maps (id) with the respective category to differentiate between them.

In place of functors, we have *enriched* functors.

Definition A.19 (Enriched functor (compare [Kel82, Sec. 1.2] or [JYJY21, Def. 1.3.5]))**.** Let $(\mathcal{V}, \otimes, I)$ be a monoidal category. Let $\mathcal{C}, \mathcal{D}$ be $\mathcal{V}$-enriched categories. A **$\mathcal{V}$-enriched functor** (or just a **$\mathcal{V}$-functor**) $F : \mathcal{C} \to \mathcal{D}$ consists of the following data.

- A function $F : \mathsf{Ob}(\mathcal{C}) \to \mathsf{Ob}(\mathcal{D})$ between the underlying sets of objects[5];
- For every $A, B : \mathcal{C}$ a morphism in $\mathcal{V}$ of type $\mathcal{C}(A, B) \to \mathcal{D}(F(A), F(B))$, often denoted as $F_{A,B}$ or, when it is clear from the context, just F;

This data has to, for every $A, B, C : \mathcal{C}$ satisfy the following coherence laws ensuring F is coherent with composition and identities, respectively.

$$\begin{array}{ccc} \mathcal{C}(A,B) \otimes \mathcal{C}(B,C) & \xrightarrow{\fatsemi^{\mathcal{C}}_{A,B,C}} & \mathcal{C}(A,C) \\ \downarrow{\scriptstyle F_{A,B} \otimes F_{B,C}} & & \downarrow{\scriptstyle F_{A,C}} \\ \mathcal{D}(F(A),F(B)) \otimes \mathcal{D}(F(B),F(C)) & \xrightarrow[\fatsemi^{\mathcal{D}}_{F(A),F(B),F(C)}]{} & \mathcal{D}(F(A),F(C)) \end{array} \qquad \begin{array}{ccc} I & & \\ \downarrow{\scriptstyle \lceil \mathrm{id}^{\mathcal{C}}_A \rceil} & \searrow^{\lceil \mathrm{id}^{\mathcal{D}}_{F(A)} \rceil} & \\ \mathcal{C}(A,A) & \xrightarrow[F_{A,A}]{} & \mathcal{D}(F(A),F(A)) \end{array}$$

Definition A.20 (Category of $\mathcal{V}$-enriched categories (compare [Kel82, Sec. 1.2]))**.** Given any monoidal category $(\mathcal{V}, \otimes, I)$, $\mathcal{V}$-enriched categories and $\mathcal{V}$-enriched functors form a category **$\mathcal{V}$-Cat**.

In this thesis there are two pertinent bases of enrichment. For the base of enrichment $(\mathbf{Set}, \times, 1)$ we obtain **Set-Cat**, the category of **Set**-enriched categories and **Set**-functors: these are precisely just categories and functors! That is, $\mathbf{Cat} \coloneqq \mathbf{Set}\text{-}\mathbf{Set}$[6]. Analogously, for the base of enrichment $(\mathbf{Cat}, \times, \mathbf{1})$ we obtain **Cat-Cat**, the category of **Cat**-enriched categories and **Cat**-functors: these are precisely 2-categories and 2-functors.

Definition A.21 (Enriched base change (compare [Kel82, Sec. 1]))**.** Let $(\mathcal{V}, \otimes, I)$ and $(\mathcal{W}, \oplus, J)$ be two monoidal categories. Then a lax monoidal functor

$$(F, \mu, \epsilon) : (\mathcal{V}, \otimes, I) \to (\mathcal{W}, \oplus, J)$$

[5] We are overloading the notation F here, trusting it is clear from the context what F is applied to.
[6] We remind the reader that we completely ignore universe levels here.

induces the **enriched base change**, an assignment which sends a $\mathcal{V}$-enriched category $\mathcal{C}$ to the $\mathcal{W}$-enriched category $F_*(\mathcal{C})$ whose data is defined as follows:

- Its objects are those of $\mathcal{C}$;
- Its hom-object is for every $A, B : F_*(\mathcal{C})$ defined as $F_*(\mathcal{C})(A,B) \coloneqq F(\mathcal{C}(A,B))$;
- The identity at $A : F_*(\mathcal{C})$ is the composite $J \xrightarrow{\epsilon} F(I) \xrightarrow{F(\mathrm{id}_A)} F(\mathcal{C}(A,A)) = F_*(\mathcal{C})(A,A)$;
- For every $A, B, C : F_*(\mathcal{D})$ the composition morphism $\fatsemi^{F_*(\mathcal{C})}_{A,B,C}$ is defined as the composite below, expressed as the commutativity of the diagram.

$$\begin{array}{ccccc} F_*(\mathcal{C})(A,B) \otimes F_*(\mathcal{C})(B,C) & & \xrightarrow{\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad} & & F_*(\mathcal{C})(A,C) \\ \| & & & & \| \\ F(\mathcal{C}(A,B)) \otimes F(\mathcal{C}(B,C)) & \xrightarrow[\mu_{\mathcal{C}(A,B),\mathcal{C}(B,C)}]{} & F(\mathcal{C}(A,B) \otimes \mathcal{C}(B,C)) & \xrightarrow[F(\fatsemi^{\mathcal{C}}_{A,B,C})]{} & F(\mathcal{C}(A,C)) \end{array}$$

The assignment F_* forms a functor

$$F_* : \mathcal{V}\textbf{-Cat} \to \mathcal{W}\textbf{-Cat}$$

whose details we do not unpack here.

Actegories

Definition A.22 (Coherence laws for a right actegory (compare [CG23, Def. 3.1.1]))**.** Coherence laws for a $\mathcal{M}$-actegory $(\mathcal{C}, \bullet, \mu, \eta)$ are analogous to the coherence laws of a monoidal category (Definition A.1). In other words, we have two laws:

- **Pentagonator.** This law has to hold for all $M, N, P : \mathcal{M}$ and $C : \mathcal{C}$. It tells us that, starting with $C \bullet (M \otimes (N \otimes P))$, any two ways of acting on C give us the same result. We can either apply actions of μ in the only way possible on this, or alpply the associator and *then* apply the actions of μ in the only way possible. This is what the diagram Eq. (A.23) describes: a

condition that has to hold.[7]

$$\begin{array}{ccccc} C \bullet (M \otimes (N \otimes P)) & \xrightarrow{\mu_{C,M,N\otimes P}} & (C \bullet M) \bullet (N \otimes P) & \xrightarrow{\mu_{C\bullet M,N,P}} & ((C \bullet M) \bullet N) \bullet P \\ \downarrow{\scriptstyle C\bullet\alpha^{-1}_{M,N,P}} & & & & \| \\ C \bullet ((M \otimes N) \otimes P)) & \xrightarrow[\mu_{C,M\otimes N,P}]{} & (C \bullet (M \otimes N)) \bullet P & \xrightarrow[\mu_{C,M,N}\bullet P]{} & ((C \bullet M) \bullet N) \bullet P \end{array} \tag{A.23}$$

- **Unitors.** These laws have to hold for all $C : \mathcal{C}$ and $M : \mathcal{M}$. The diagrams in Eq. (A.24) tell us that any two ways of getting rid of I in $\mathcal{C} \bullet (M \otimes I)$ (resp. $\mathcal{C} \bullet (I \otimes M)$) are equal. That is, we get the same result as if we a) apply the right unitor ρ_M (resp. left unitor λ_M) of the monoidal category, or b) apply the multiplicator of the actegory, and then the unitor of the actegory. See also [CG23, Remark 3.1.2].

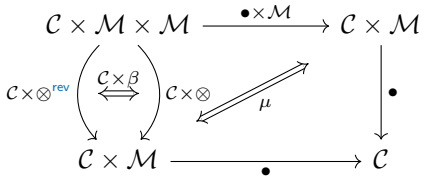
$$\begin{array}{ccc} C \bullet (M \otimes I) & & \\ \downarrow{\scriptstyle \mu_{C,M,I}} \quad \searrow{\scriptstyle C\bullet\rho_M} & & \\ (C \bullet M) \bullet I & \xrightarrow[\eta_{C\bullet M}]{} & C \bullet M \end{array} \qquad \begin{array}{ccc} & & C \bullet (I \otimes M) \\ & \swarrow{\scriptstyle C\bullet\lambda_M} \quad \downarrow{\scriptstyle \mu_{C,I,M}} & \\ C \bullet M & \xleftarrow[\eta_{C\bullet M}]{} & (C \bullet I) \bullet M \end{array} \tag{A.24}$$

Lemma 3.6. *If $(\mathcal{M}, \otimes, I)$ is a braided monoidal category, then any right $(\mathcal{M}, \otimes, I)$-actegory can be turned into a left $(\mathcal{M}, \otimes^{\mathsf{rev}}, I)$-actegory, and vice-versa.*

Proof. Let $\beta : \otimes \Rightarrow \otimes^{\mathsf{rev}}$ be the braiding of $(\mathcal{M}, \otimes, I)$. Then given a right $(\mathcal{M}, \otimes, I)$-actegory $(\mathcal{C}, \bullet, \eta, \mu)$ we can define a right $(\mathcal{M}, \otimes^{\mathsf{rev}}, I)$ actegory (i.e. a left $(\mathcal{M}, \otimes, I)$-actegory) with the same base $\mathcal{C}$, whose structure maps are $(\bullet, \eta, \mu^{\mathsf{rev}})$. That is, the action functor and the unitor are the same as the starting actegory, because they can be defined without a reference to the underlying monoidal product of the acting actegory. The only difference is in the multiplicator μ^{rev} which is

[7] Its name "pentagonator" arises out of the squashing of the righmost identity into a single object. While this seems justified in the monoidal case, the added distinction between the associator and multiplicator in an actegory in my opinion suggests a different conceptualisation of the diagram.

defined by the following pasting diagram.



(A.25)

Componentwise, for $C : \mathcal{C}$ and $M, N : \mathcal{M}$ it gives us a natural isomorphism

$$\mu^{\mathsf{rev}}_{C,M,N} \coloneqq \boxed{C \bullet (M \otimes^{\mathsf{rev}} N) \xrightarrow{C \times \beta_{M,N}} C \bullet (M \otimes N) \xrightarrow{\mu_{C,M,N}} (C \bullet M) \bullet N}$$

To show that $(\mathcal{C}, \bullet, \eta, \mu^{\mathsf{rev}})$ is indeed an actegory we have to show that the appropriate coherence conditions (Definition A.22) are satisfied. This follows routinely by unpacking the corresponding pasting diagrams and applying the naturality of β.

□

Appendix B

Locally graded and indexed categories

Locally graded categories

A common generaliation of enriched categories and actegories are *locally graded* categories, originally defined in [Woo76, Woo78].[1] They are categories enriched in $\underline{\mathbf{Cat}}(\mathcal{M}^{\mathrm{op}}, \mathbf{Set})$, for a given monoidal category $\mathcal{M}$. They capture both internal parameterisation (one given by actegories) and external parameterisation (one given by enriched categories).

Analogously to actegories, they have a corresponding **Para** construction which constructs a bicategory of parametric morphisms. We start by defining the monoidal structure of the category $\underline{\mathbf{Cat}}(\mathcal{M}^{\mathrm{op}}, \mathbf{Set})$ which is needed to talk about enrichment in it. It is given by *Day Convolution*.

Definition B.1 (Day convolution (compare [Lor21, Prop. 6.2.1], [Gar18, Def. 11]))**.** Let $(\mathcal{M}, \otimes, I)$ be a monoidal category. Then the functor category $\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$ can be equipped with a monoidal structure given by **Day convolution**. It is defined by the following data:

- The monoidal product $\otimes_D$ as

$$\begin{aligned} \underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}) \times \underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}) &\to \underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}) \\ (F, G) &\mapsto \int^{(M,N):\mathcal{M}\times\mathcal{M}} F(M) \times G(N) \times \mathcal{M}(M \otimes N, -) \end{aligned}$$

[1] See also [Gar18, Lemma 12], [Gow20, Def. 5.1.9], [Mel22a, Prop. 2], and [MU22, Def. 9]

- The monoidal unit as $\mathcal{M}(I, -) : \mathcal{M} \to \mathbf{Set}$

Associators and unitors are defined in [Lor21, Prop 6.2.1].

Remark B.2. To understand why this is called Day *convolution*, it is helpful to consider the case where $\mathcal{M}$ is a discrete monoidal category, i.e. a monoid. In such a case the coend reduces to a coproduct, and morphisms in $\mathcal{M}$ to equalites. In other words, we have

$$\begin{aligned}(F \otimes_D G)(X) &= \int^{M,N} F(M) \times G(N) \times \mathcal{M}(M \otimes N, X) \\ &= \sum_{\substack{M,N:\mathcal{M} \\ M \otimes N = X}} F(M) \times G(N)\end{aligned}$$

Furthermore specialising $\mathcal{M}$ to, for instance, the group $(\mathbb{Z}, +, 0)$ of integers with addition allows us to write the above as

$$(f \otimes_D g)(x) = \sum_{m:\mathbb{Z}} f(m) \times g(x - m)$$

making the connection with classical convolution formula more apparent (we have taken the liberty of turning variable names lowercase).

To better understand this monoidal structure, we study one if its interesting properties: monoids in this monoidal category are lax monoidal functors. And to understand that, we will first study the set of natural transformations from $\mathcal{M}(I, -)$ to any other functor F (Prop. B.3) and the set of natural transformations $F \otimes_D G \Rightarrow H$ (for any three functors F, G, H) (Prop. B.4), showing both of these admit a natural reduction.

Proposition B.3. *Let $(\mathcal{M}, \otimes, I)$ be a monoidal category, and $F : \mathcal{M} \to \mathbf{Set}$ a functor. Then*

$$\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})(\mathcal{M}(I, -), F) \cong F(I)$$

Proof. Follows by unpacking the set of natural transformations as an end, and applying Yoneda. □

Concretely, this means that every element $i : F(I)$ can be identified with the natural transformation $F(-)(i) : \underline{\mathbf{Set}}(\mathcal{M}(I, -), F)$, i.e. an assignment $F(-)(i) : \mathcal{M}(I, X) \to F(X)$ for every $X : \mathcal{M}$, and vice versa.

Proposition B.4. *Let $(\mathcal{M}, \otimes, I)$ be a monoidal category. Then*

$$\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})(F \otimes_D G, H) \cong \int_{(M,N):\mathcal{M}\times\mathcal{M}} \mathbf{Set}(F(M) \times G(N), H(M \otimes N))$$

Proof. Follows by the coend calculus below.

$$\begin{aligned}
&\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})(F \otimes_D G, H) \\
\text{(Natural transformation as end)} &\cong \int_{P:\mathcal{M}} \mathbf{Set}((F \otimes_D G)(P), H(P)) \\
\text{(Day convolution)} &\cong \int_{P:\mathcal{M}} \mathbf{Set}\Big(\int^{(M,N):\mathcal{M}\times\mathcal{M}} \mathcal{M}(M \otimes N, P) \times F(M) \times G(N), H(P)\Big) \\
\text{(Cocontinuity)} &\cong \int_{(P,M,N):\mathcal{M}\times\mathcal{M}\times\mathcal{M}} \mathbf{Set}(\mathcal{M}(M \otimes N, P) \times F(M) \times G(N), H(P)) \\
\text{(Tensor-hom adjunction)} &\cong \int_{(P,M,N):\mathcal{M}\times\mathcal{M}\times\mathcal{M}} \mathbf{Set}(\mathcal{M}(M \otimes N, P), \underline{\mathbf{Set}}(F(M) \times G(N), H(P))) \\
\text{(Yoneda)} &\cong \int_{(M,N):\mathcal{M}\times\mathcal{M}} \mathbf{Set}(F(M) \times G(N), H(M \otimes N))
\end{aligned}$$

□

Proposition B.5 (Lax monoidal functors as monoids)**.** *There is a 1-1 correspondence between monoids in $(\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}), \otimes_D, \mathcal{M}(I, -))$ and lax monoidal functors of type $(\mathcal{M}, \otimes, I) \to (\mathbf{Set}, \times, 1)$.*

Proof. A lax monoidal functor consists of a functor $F : \mathcal{M} \to \mathbf{Set}$ and two natural transformations $\mu : \int_{(M,N):\mathcal{M}\times\mathcal{M}} \mathbf{Set}(F(M)\times F(N), F(M\otimes N))$ and $\epsilon : \int_{\bullet:\mathbf{1}} \mathbf{Set}(1, F(I))$. A monoid in $\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$ consists of a functor F and two natural transformations $\mu : F \otimes_D F \Rightarrow F$ and $\epsilon : \mathcal{M}(I, -) \Rightarrow F$. Via Props. B.3 and B.4 it becomes straightforward to translate these representations one to another, and confirm that they are indeed well defined, and inverses. □

Since this is a monoidal category, we can use it as the base of enrichment.

Definition B.6 (Locally graded category ([MU22, Def. 9]))**.** Let $(\mathcal{M}, \otimes, I)$ be a monoidal category. Then we call a $(\underline{\mathbf{Cat}}(\mathcal{M}^{\mathrm{op}}, \mathbf{Set}), \otimes_D, \mathcal{M}^{\mathrm{op}}(I, -))$-enriched category a **locally $\mathcal{M}$-graded** category.

Let us unpack, following Definition A.15 what such an enriched category (call it $\mathcal{C}$) consists of.

- **Objects.** Like any enriched category, it consists of a set of objects.

- **Morphisms.** Given any two objects A, B of the hom-object $\mathcal{C}(A,B)$ is a *functor* of type $\mathcal{M}^{\mathsf{op}} \to \mathbf{Set}$. Since a functor does not have elements, this $\underline{\mathbf{Cat}}(\mathcal{M}^{\mathsf{op}}, \mathbf{Set})$-enriched category $\mathcal{C}$ does not have morphisms in the conventional sense. But there are "graded morphisms", i.e. for each object $M : \mathcal{M}^{\mathsf{op}}$ a set of morphisms $\mathcal{C}(A,B)(M)$;
- **Identity.** To talk about identities, we make use of Prop. B.3 which tells us that the set of possible identity natural transformations $\mathcal{M}^{\mathsf{op}}(I,-) \Rightarrow \mathcal{C}(A,A)$ is isomorphic to the set $\mathcal{C}(A,A)(I)$. Thus we refer to the identity morphism id_A as an element of $\mathcal{C}(A,A)(I)$;
- **Composition.** Analogous story follows here. Via Prop. B.4 the set of possible composition natural transformations $\mathcal{C}(A,B) \otimes_D \mathcal{C}(B,C) \Rightarrow \mathcal{C}(A,C)$ is isomorphic to the set $\int_{(M,N):\mathcal{M}^{\mathsf{op}}\times\mathcal{M}^{\mathsf{op}}} \mathbf{Set}(\mathcal{C}(A,B)(M) \times \mathcal{C}(B,C)(N), \mathcal{C}(A,C)(M \otimes N))$. Thus we refer to the composition morphism $\fatsemi_{A,B,C}$ as a natural assignment of a function

$$\mathcal{C}(A,B)(M) \times \mathcal{C}(B,C)(N) \to \mathcal{C}(A,C)(M \otimes N)$$

to every $A, B, C : \mathcal{C}$ and $M, N : \mathcal{M}^{\mathsf{op}}$.

We can now see how locally graded categories are a generalisation of actegories and enriched categories.

Example B.7 (Actegories are locally graded categories)**.** Let $(\mathcal{C}, \bullet)$ be a $\mathcal{M}$-actegory. Then we can construct a locally $\mathcal{M}$-graded category $\mathcal{C}'$ with the following data.

- Its objects are same as that of $\mathcal{C}$;
- For any $A, B : \mathcal{C}'$, the hom functor $\mathcal{C}'(A,B)$ is the composite $\mathcal{M}^{\mathsf{op}} \xrightarrow{A\bullet -} \mathcal{C}^{\mathsf{op}} \xrightarrow{\mathcal{C}(-,B)} \mathbf{Set}$;
- For every $A : \mathcal{C}'$ the set of potential identity morphisms $\mathcal{C}'(A,A)(I)$ reduces to the set $\mathcal{C}(A \bullet I, A)$ for which there is a clear candidate: the actegory unitor $\eta_A^{-1} : \mathcal{C}(A \bullet I, A) = \mathcal{C}'(A,A)(I)$;
- For every $A, B, C : \mathcal{C}'$ the set of potential composition natural transformations reduces to the set $\int_{(M,N):\mathcal{M}^{\mathsf{op}}\times\mathcal{M}^{\mathsf{op}}} \mathbf{Set}(\mathcal{C}(A \bullet M, B) \times \mathcal{C}(B \bullet N, C), \mathcal{C}(A \bullet (M \otimes N), C))$ for which there is a clear candidate: a function

$$\begin{aligned} \mathcal{C}(A \bullet M, B)) \times \mathcal{C}(B \bullet N, C)) &\to \mathcal{C}(A \bullet (M \otimes N), C)) \\ (f,g) &\mapsto \boxed{A \bullet (M \otimes N) \xrightarrow{\mu_{C,M,N}} (A \bullet M) \bullet N \xrightarrow{f \bullet N} B \bullet N \xrightarrow{g} C} \end{aligned}$$

defined naturally for each $(M, N) : \mathcal{M}^{\mathsf{op}} \times \mathcal{M}^{\mathsf{op}}$. Note that this is precisely the **Para** composition — describing the manner by which two parametric morphisms are composed.

Remark B.8. The unitor and the multiplicator of the actegory — which are isomorphisms — are used to define the identity and composition of a locally graded category. But note that only one direction of this isomorphism is used to define a locally graded category. This suggests that lax actegories too embed into locally graded categories (see [Gow20, Def. 5.1.9]).

Actegories, seen as locally graded categories come with an additional property: they have copowers by all representable functors. As it also turns out, every locally graded category with copowers by all representables can be turned into an actegory.

Proposition B.9 ([Gar18, Prop. 13])**.** *Let $\mathcal{M}$ be a symmetric monoidal category. Then there is a 1-1 correspondence between $\mathcal{M}$-actegories and locally $\mathcal{M}$-graded categories admitting powers by representables.*

Example B.10 (Enriched categories are locally graded categories)**.** Let $\mathcal{C}$ be a $\mathcal{M}$-category. Then we can construct a locally $\mathcal{M}$-graded category $\mathcal{C}'$ with the following data.

- Its objects are same as that of $\mathcal{C}$;
- For any $A, B : \mathcal{C}'$, the hom functor $\mathcal{C}'(A, B)$ is the composite $\mathcal{M}^{\mathsf{op}} \xrightarrow{\mathcal{M}(-,\mathcal{C}(A,B))} \mathbf{Set}$;
- For every $A : \mathcal{C}'$ the set of identity morphisms $\mathcal{C}'(A, A)(I)$ reduces to the set $\mathcal{M}(I, \mathcal{C}(A, A))$ for which there is a clear candidate: the name of the identity map $\lceil \mathrm{id}_A \rceil : \mathcal{M}(I, \mathcal{C}(A, A))$ of $\mathcal{C}$;
- For every $A, B, C : \mathcal{C}'$ set of potential composition natural transformations reduces to the set $\int_{(M,N):\mathcal{M}^{\mathsf{op}}\times\mathcal{M}^{\mathsf{op}}} \mathbf{Set}(\mathcal{M}(M, \mathcal{C}(A, B)) \times \mathcal{M}(N, \mathcal{C}(B, C)), \mathcal{M}(M \otimes N, \mathcal{C}(A, C))$ for which there is a clear candidate: a function

$$\begin{aligned} \mathcal{M}(M, \mathcal{C}(A, B)) \times \mathcal{M}(N, \mathcal{C}(B, C)) &\to \mathcal{M}(M \otimes N, \mathcal{C}(A, C)) \\ (f, g) &\mapsto \boxed{M \otimes N \xrightarrow{f\otimes g} \mathcal{C}(A, B) \otimes \mathcal{C}(B, C) \xrightarrow{\fatsemi^{\mathcal{C}}_{A,B,C}} \mathcal{C}(A, C)} \end{aligned}$$

defined naturally for each $(M, N) : \mathcal{M}^{\mathsf{op}} \times \mathcal{M}^{\mathsf{op}}$. This too is **Para** composition — but for externally parametric morphisms.

We do not fully prove it here, but both examples above assemble, respectively, into functors $\mathcal{M}$-**Act** $\to$ $\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$-**Cat**, and $\mathcal{M}$-**Cat** $\to$ $\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$-**Cat**.

These two embeddings show that locally graded categories can model both actegories and enriched categories. What remains now is to define, analogously to **Para**, a construction that consumes a locally graded category and produces a bicategory of parametric morphisms. We will see that to be the case, and, interestingly, we will see that this produces a 2-category, as opposed to a bicategory.

This will be done by taking the base of change along the category of elements functor. Recall that the category of elements is a functor $\mathbf{El} : \underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}) \to \mathbf{Cat}$ defined for any category $\mathcal{M}$. If the domain is equipped with the structure of Day convolution, and codomain with the cartesian product, we can show that this functor is lax monoidal.

Proposition B.11 ($\mathbf{El}$ is a lax monoidal functor)**.** *Let $(\mathcal{M}, \otimes, I)$ be a monoidal category. Then the functor $\mathbf{El} : \underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}) \to \mathbf{Cat}$ becomes a lax monoidal functor*

$$(\mathbf{El}, \mu, \epsilon) : (\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}), \otimes_D, \mathcal{M}(I, -)) \to (\mathbf{Cat}, \times, \mathbf{1})$$

where

- *The laxator $\mu_{F,G} : \mathbf{El}(F) \times \mathbf{El}(G) \to \mathbf{El}(F \otimes_D G)$ is given by*

$$((M, M_F), (N, N_G)) \mapsto (M \otimes N, ((M, N), (M_F, N_G, id_{M \otimes N})))$$

- *The unit $\epsilon : \mathbf{1} \to \mathbf{El}(\mathcal{M}(I, -))$ picks out $(I, id_I : \mathcal{M}(I, I))$.*

This means that we $(\mathbf{El}, \mu, \epsilon)$ is a suitable functor we can perform base change along. In doing so, we obtain a 2-functor $(\mathbf{El}, \mu, \epsilon)_* : \underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$-**Cat** $\to$ **2Cat**. In the following example, we study its action on objects, which results in a locally graded version of **Para**.

Example B.12. Let $\mathcal{C}$ be a $\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$-enriched category. Then the action of $(\mathbf{El}, \mu, \epsilon)_*$ on $\mathcal{C}$ results in a 2-category with the following data.

- **Objects** are the same as those of $\mathcal{C}$;
- **Morphisms.** A morphism $A \to B$ consists of an object $M : \mathcal{M}$ and an element $f : \mathcal{C}(A, B)(M)$.

- **2-morphisms.** A map $(M, f) \Rightarrow (N, g)$ is given by $r : M \to N$ such that $\mathcal{C}(A, B)(r)(g) = f$;
- **Identity morphism.** For every object A the identity morphism is given by the composite

$$\mathbf{1} \xrightarrow{\epsilon} \mathbf{El}(\mathcal{M}^{\mathsf{op}}(I, -)) \xrightarrow{\mathbf{El}(\mathrm{id}_A)} \mathbf{El}(\mathcal{C}(A, A))$$

  We note that precomposing $\mathbf{El}(\mathrm{id}_A)$ with ϵ amounts exactly to applying Yoneda lemma (Prop. E.11), and selecting the identity morphism id_A — already considered to be an element of $\mathcal{C}(A, A)(I)$. More precisely, the name of the identity $\epsilon \fatsemi \mathbf{El}(\mathrm{id}_A) : 1 \to \mathbf{El}(\mathcal{C}(A, A))$ picks out $(I : \mathcal{M}^{\mathsf{op}}, \mathrm{id}_A : \mathcal{C}(A, A)(I))$;
- **Morphism composition.** For every $A, B, C : \mathcal{C}$ the composition morphism is given by enriched base change which unpacks to a composite functor of type

$$\begin{array}{rl} \mathbf{El}(\mathcal{C}(A, B)) \times \mathbf{El}(\mathcal{C}(B, C)) \xrightarrow{\mu_{\mathcal{C}(A,B),\mathcal{C}(B,C)}} \mathbf{El}(\mathcal{C}(A, B) \otimes_D \mathcal{C}(B, C)) & \xrightarrow{\mathbf{El}(\fatsemi^{\mathcal{C}}_{A,B,C})} \mathbf{El}(\mathcal{C}(A, C)) \\ ((M, f), (N, g)) \mapsto (M \otimes N, ((M, N), (f, g, \mathrm{id}_{M\otimes N}))) & \mapsto (M \otimes N, f \fatsemi^{\mathcal{C}}_{A,B,C} g) \end{array}$$

It can be shown that in both cases of an actegory and an enriched category seen as a locally graded category, applying the base change along the category of elements produces the analogous 2-category to $\mathbf{Para}(\mathcal{C})$.

Locally indexed categories

In the previous section we equipped the category $\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$ with the Day convolution monoidal product. This required $\mathcal{M}$ itself to be a monoidal category. But there is another monoidal product we can equip $\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$ with that does not necessitate any monoidal structure on $\mathcal{M}$.

Definition B.13 (Pointwise product)**.** Let $\mathcal{M}$ be a category. Then the category $\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$ can be given cartesian monoidal structure where

- Its cartesian monoidal product $\times$ is defined as

$$\begin{array}{l} \underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}) \times \underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}) \to \underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}) \\ (F, G) \mapsto \boxed{\mathcal{M} \xrightarrow{\Delta_{\mathcal{M}}} \mathcal{M} \times \mathcal{M} \xrightarrow{F\times G} \mathbf{Set} \times \mathbf{Set} \xrightarrow{\times} \mathbf{Set}} \end{array}$$

- The unit is $\mathsf{const}(1) : \mathcal{M} \to \mathbf{Set}$

Associators and unitors can be defined routinely from the pointwise associators and unitors of $(\mathbf{Set}, \times, 1)$. It can be checked that this monoidal product is indeed cartesian, following similarly from the cartesian structure of $\mathbf{Set}$.

When the category $\mathcal{M}$ is cartesian, then we can also define the Day convolution product on $\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set})$, and these monoidal products turn out to be equivalent.

Definition B.14 (Locally indexed category, compare [Lev13, Def. 75])**.** Let $\mathcal{M}$ be a category. Then a we call a category enriched in $(\underline{\mathbf{Cat}}(\mathcal{M}, \mathbf{Set}), \times, \mathsf{const}(1))$ a **locally $\mathcal{M}$-indexed** category.

Let us unpack, following Definition A.15 what such an enriched category (call it $\mathcal{C}$) consists of. Like a locally graded category, it consists of a set of objects, and its hom-object $\mathcal{C}(A, B)$ is a functor of type $\mathcal{M}^{\mathsf{op}} \to \mathbf{Set}$. This means that a locally indexed category also does not have morphisms in the conventional sense, but instead has "graded" (or more appropriately, "indexed") morphisms, i.e. a set of morphisms for every object $M : \mathcal{M}^{\mathsf{op}}$. Where locally indexed categories differ from graded ones is in identities and composition.

- **Identity.** The identity on an object A is a natural transformation $\mathrm{id}_A : \mathsf{const}(1) \Rightarrow \mathcal{C}(A, A)$. To every object $M : \mathcal{M}^{\mathsf{op}}$ it assigns the map of type $\mathcal{C}(A, A)(M)$ which we suggestively call *projection* and label as π_A;
- **Composition.** The composition is a natural transformation $\fatsemi_{A,B,C} : \mathcal{C}(A, B)(-) \times \mathcal{C}(B, C)(-) \Rightarrow \mathcal{C}(A, C)(-)$. To every $M : \mathcal{M}$ it assigns a map

$$\fatsemi^{M}_{A,B,C} : \mathcal{C}(A, B)(M) \times \mathcal{C}(B, C)(M) \to \mathcal{C}(A, C)(M)$$

  which composes two morphisms of the same grade.

If locally graded categories are analogous to $\mathbf{Para}(\mathcal{C}$, then locally indexed categories are analogous to $\mathbf{coKl}(X \times -)$. Locally indexed categories define a composition of morphisms only if they are of the same grade, unlike locally graded categories. Likewise, note how locally indexed categories define the identity map for every object of $\mathcal{M}$, while locally graded categories only defined it for the monoidal unit I.

What follows is a locally indexed variant of the $\mathbf{coKl}(X \times -)$ construction from Definition 3.30.

Example B.15 (compare [Lev13, Def. 77])**.** Let $\mathcal{C}$ be a cartesian category. Then we can form a locally $\mathcal{C}$-indexed category $\mathcal{C}$ with the following data.

- Its objects are the same as those of $\mathcal{C}$;
- Given any $A, B : \mathcal{C}$ the hom functor is $\mathcal{C}(A, B) \coloneqq \boxed{\mathcal{C}^{\mathsf{op}} \xrightarrow{A\times -} \mathcal{C}^{\mathsf{op}} \xrightarrow{\mathcal{C}(-,B)} \mathbf{Set}}$;
- For every A the identity maps unpacks to a natural transformation $\mathrm{id}_A : \mathsf{const}(1) \Rightarrow \mathcal{C}(A \times -, B)$ whose component at every $M : \mathcal{M}^{\mathsf{op}}$ is a morphism $\lceil \pi_A \rceil : 1 \to \mathcal{C}(A \times M, A)$ picking out the projection morphism $\pi_A : A \times M \to A$;
- For every $A, B, C : \mathcal{C}$ and $M : \mathcal{M}^{\mathsf{op}}$ composition unpacks to a natural transformation $\mathcal{C}(A \times -, B) \times \mathcal{C}(B \times -, C) \Rightarrow \mathcal{C}(A \times -, C)$ which is for every $M : \mathcal{M}^{\mathsf{op}}$ defined as the function

$$\mathcal{C}(A \times -, B) \times \mathcal{C}(B \times M, C) \to \mathcal{C}(A \times M, C)$$
$$(f, g) \mapsto (f \fatsemi^{\natural} g)^{\Delta_M}$$

Appendix C

The Para construction

Proposition 3.22. *If the $\mathcal{M}$-actegory $(\mathcal{C}, \bullet)$ is strict, then* $\mathbf{Para}_\bullet(\mathcal{C})$ *is a 2-category.*

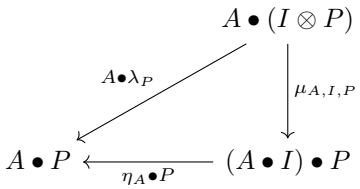
Proof. Showing that a bicategory is a 2-category reduces to showing that the unitors and associators are identity natural transformations. When it comes to the unitors, let's study what happens when we compose

$$(I, \eta_A) : \mathbf{Para}_\bullet(\mathcal{C})(A, A) \quad \text{and} \quad (P, f) : \mathbf{Para}_\bullet(\mathcal{C})(A, B)$$

where $\eta_A : A \bullet I \to A$ and $f : A \bullet P \to B$. The resulting composite (which we call g) is

$$A \bullet (I \otimes P) \xrightarrow{\mu_{A,I,P}} (A \bullet I) \bullet P \xrightarrow{\eta_A \bullet P} A \bullet P \xrightarrow{f} B$$

In order to show that the left unitor is the identity natural isomorphism, we need to show that the reparameterisation $(I \otimes P, g) \Rightarrow (P, f)$ is identity. This is witnessed by the identity morphism $\lambda_P^{-1} : P \to I \otimes P$ (as $\mathcal{M}$ is strict) for which the reparameterisation condition reduces to showing the diagram Eq. (C.1) commutes. And this diagram is precisely the left unitor coherence condition of an actegory (Eq. (A.24)). Right unitor follows analogously.



(C.1)

For the associator, given

$$(P, f) : \mathbf{Para}_\bullet(\mathcal{C})(A, B) \quad \text{and} \quad (Q, g) : \mathbf{Para}_\bullet(\mathcal{C})(B, C) \quad \text{and} \quad (R, h) : \mathbf{Para}_\bullet(\mathcal{C})(C, D)$$

we need to show that two different ways of composing these morphisms

$$((P \otimes Q) \otimes R, (f \fatsemi^{\mathbb{p}} g) \fatsemi^{\mathbb{p}} h) \quad \text{and} \quad (P \otimes (Q \otimes R), f \fatsemi^{\mathbb{p}} (g \fatsemi^{\mathbb{p}} h))$$

are equal on the nose. Since $\mathcal{M}$ is strict it is straightforward to provide the underlying identity map. All that remains is to check the reparameterisation condition, which reduces to the diagram Eq. (C.2).

$$\begin{array}{ccccc}
A \bullet ((P \otimes Q) \otimes R) & & \xrightarrow{\mu_{A,P\otimes Q,R}} & & (A \bullet (P \otimes Q)) \bullet R \\
\downarrow{\scriptstyle A\bullet\alpha_{P,Q,R}} & & & & \downarrow{\scriptstyle \mu_{A,P,Q}\bullet R} \\
A \bullet (P \otimes (Q \otimes R)) & \xrightarrow{\mu_{A,P,Q\otimes R}} & (A \bullet P) \bullet (Q \otimes R) & \xrightarrow{\mu_{A\bullet P,Q,R}} & ((A \bullet P) \bullet Q) \bullet R \\
 & & \downarrow{\scriptstyle f\bullet(Q\otimes R)} & & \downarrow{\scriptstyle (f\bullet Q)\bullet R} \\
 & & B \bullet (Q \otimes R) & \xrightarrow[\mu_{B,Q,R}]{} & (B \bullet Q) \bullet R
\end{array} \tag{C.2}$$

The top square commutes because of the actegory pentagonator Eq. (A.23), and the bottom square commutes because of naturality of μ. □

Proposition 3.39 (Monoidal structure of **Para** (compare [CGHR22, Section 2.1.]))**.** *Let* $(\mathcal{C}, \bullet)$ *be monoidal* $\mathcal{M}$*-actegory with the underlying product* $(\boxtimes, J)$*. Then* $\mathbf{Para}_\bullet(\mathcal{C})$ *becomes a monoidal bicategory with the following data.*

- *The monoidal product of two objects* X, Y *is given by the monoidal product* $X \boxtimes Y$ *of* $(\mathcal{C}, \boxtimes, J)$*;*
- *The monoidal unit is the monoidal unit* J *of* $(\mathcal{C}, \boxtimes, J)$*;*
- *The monoidal product of two parametric morphisms*

$$(P, f : A \bullet P \to B) \quad \textit{and} \quad (Q, g : C \bullet Q \to D)$$

is given by $(P \otimes Q, f \boxtimes^{\mathbf{p}} g)$ *where*

$$f \boxtimes^{\mathbf{p}} g \coloneqq \boxed{(A \boxtimes C) \bullet (P \otimes Q) \xrightarrow{c_{A,C,P,Q}} (A \bullet P) \boxtimes (B \bullet Q) \xrightarrow{f \boxtimes g} B \boxtimes D}$$

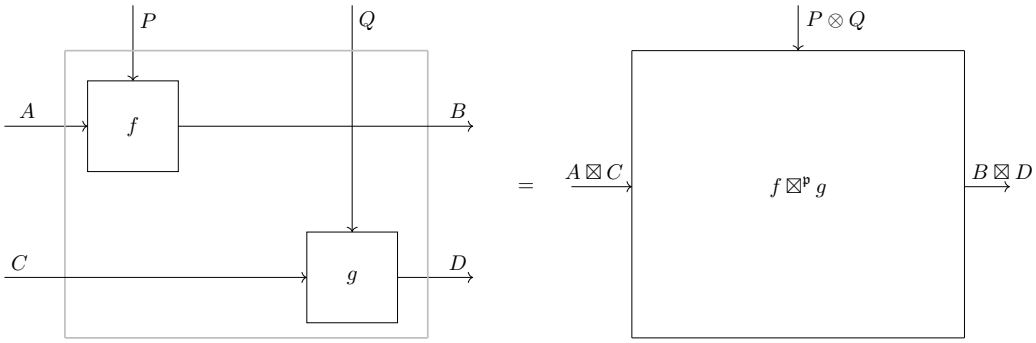


Figure C.1: String diagram of the monoidal product of two parametric morphisms

- *The associators and unitors are those of* $(\mathcal{C}, \boxtimes, J)$.

Proof. The monoidal bicategory structure ([JYJY21, Def. 12.1.2]) of $\mathbf{Para}_{\bullet}(\mathcal{C})$ consists of the following data.

- The monoidal product pseudofunctor $\boxtimes^{\mathbf{p}} : \mathbf{Para}_{\bullet}(\mathcal{C}) \times \mathbf{Para}_{\bullet}(\mathcal{C}) \to \mathbf{Para}_{\bullet}(\mathcal{C})$ defined by
  - An action on objects mapping X, Y to $X \boxtimes Y$;
  - For every $A, B, C, D : \mathbf{Para}_{\bullet}(\mathcal{C})$ a local functor

$$\mathbf{Para}_{\bullet}(\mathcal{C})(A, B) \times \mathbf{Para}_{\bullet}(\mathcal{C})(C, D) \to \mathbf{Para}_{\bullet}(\mathcal{C})(A \boxtimes C, B \boxtimes D)$$

    mapping objects i.e. parametric morphisms $(P, f : A \bullet P \to B)$ an $(Q, g : C \bullet Q \to D)$ to $(P \otimes Q, f \boxtimes^{\mathbf{p}} g)$ where

$$f \boxtimes^{\mathbf{p}} g \coloneqq \boxtimes (A \boxtimes C) \bullet (P \otimes Q) \xrightarrow{c_{A,C,P,Q}} (A \bullet P) \boxtimes (B \bullet Q) \xrightarrow{f \boxtimes g} B \boxtimes D$$

    and morphisms i.e. reparameterisations $r : (P, f) \Rightarrow (Q, g)$ and $s : (R, h) \Rightarrow (S, i)$ to $r \otimes s$.

– A natural isomorphism $\mu^{\boxtimes^{\mathbb{p}}}$

$$\begin{array}{ccc} \mathbf{P}(A,B)\times\mathbf{P}(X,Y)\times\mathbf{P}(B,C)\times\mathbf{P}(Y,Z) & \xrightarrow{\fatsemi} & \mathbf{P}(A,C)\times\mathbf{P}(X,Z) \\ \downarrow{\scriptstyle \boxtimes^{\mathbb{p}}{}_{A,B,X,Z}\times\boxtimes^{\mathbb{p}}{}_{B,C,Y,Z}} & \overset{\mu^{\boxtimes^{\mathbb{p}}}}{\Rightarrow} & \downarrow{\scriptstyle \boxtimes^{\mathbb{p}}{}_{A,C,X,Z}} \\ \mathbf{P}(A\boxtimes X,B\boxtimes Z)\times\mathbf{P}(B\boxtimes Y,C\boxtimes Z) & \xrightarrow[\fatsemi]{} & \mathbf{P}(A\boxtimes X,C\boxtimes Z) \end{array}$$

This natural isomorphism (where here $\mathbf{P}$ stands for $\mathbf{Para}_\bullet(\mathcal{C})$) is a weakened version of the interchange law (Eq. (2.11))). Componentwise it describes the relation between different orders of composition of parametric maps

$$(P,f):\mathbf{Para}_\bullet(A,B)\quad\text{and}\quad(R,h):\mathbf{Para}_\bullet(B,C)$$
$$(Q,g):\mathbf{Para}_\bullet(X,Y)\quad\text{and}\quad(S,i):\mathbf{Para}_\bullet(Y,Z)$$

We can either a) compose the maps in parallel, and then the results sequentially, or b) compose the maps sequentially, and then the results in parallel. These two options yield:

$$(f\boxtimes^{\mathbb{p}}g)\fatsemi^{\mathbb{p}}(h\boxtimes^{\mathbb{p}}i)\quad\text{and}\quad(f\fatsemi^{\mathbb{p}}h)\boxtimes^{\mathbb{p}}(g\fatsemi^{\mathbb{p}}i)$$

whose parameter spaces are $(P\otimes Q)\otimes(R\otimes S)$ and $(P\otimes R)\otimes(Q\otimes S)$, respectively. The invertible 2-cell $\mu^{\boxtimes^{\mathbb{p}}}$ ensures these are isomorphic, and is witnessed by the reparameterisation of type $(P\otimes R)\otimes(Q\otimes S)\to(P\otimes Q)\otimes(R\otimes S)$ given by the interchanger $c_{P,R,Q,S}$ (Definition A.10).

– A natural isomorphism $\epsilon^{\boxtimes^{\mathbb{p}}}$ which relates the product of two identity maps (I,η_A) on A and (I,η_B) on B with the identity $(I,\eta_{A\boxtimes B})$ on the product $A\boxtimes B$. The former unpacks to a parametric map

$$\eta_A\boxtimes^{\mathbb{p}}\eta_B\coloneqq\boxtimes(A\boxtimes B)\bullet(I\otimes I)\xrightarrow{c_{A,B,I,I}}(A\bullet I)\boxtimes(B\bullet I)\xrightarrow{f\boxtimes g}A\boxtimes B$$

It is routine to show that this is isomorphic to $(I,\eta_{A\boxtimes B})$ witnessed by the reparameterisation of either the left or the right unitor of I.

- Identity pseudofunctor $\mathbf{1}\xrightarrow{J^{\mathbb{p}}}\mathbf{Para}_\bullet(\mathcal{C})$ defined by:

  - A choice of $J : \mathbf{Para}(\mathcal{C})$, its identity object;
  - A morphism $(I, \eta_J) : \mathbf{Para}_\bullet(\mathcal{C})(J, J)$ where $\eta_J : J \bullet I \to I$;
  - An invertible 2-cell $\eta_J \fatsemi^{\mathfrak{p}} \eta_J \Rightarrow \eta_J$ in $\mathbf{Para}_\bullet(\mathcal{C})(J, J)$. Its domain can be unpacked to

$$J \bullet (I \otimes I) \xrightarrow{\mu_{J,I,I}} (J \bullet I) \bullet I \xrightarrow{\eta_J} J \bullet I \xrightarrow{\eta_J} J.$$

    and analogously to above it can be reparameterised by the isomorphism of type $I \otimes I \to I$ given by either the left or the right unitor of I, yielding η_J.
  - Invertible 2-cell $\mathrm{id}_J \Rightarrow (I, \eta_J)$. As these are equal on the nose, this 2-cell is identity.[1]

- The associator and the left and right unitor adjoint equivalences, all defined by the images of the associator, unitor and pentagonator isomorphisms of the monoidal category $\mathcal{C}$ under the embedding of $\mathcal{C}$ into $\mathbf{Para}_\bullet(\mathcal{C})$ (Lemma 3.32).
- The pentagonator and 2-unitor invertible 2-cells, all defined by iterated application of the isomorphism $\lambda_I : I \otimes I \to I$. They establish a coherence coherence of the associators and the unitors which in our case all have trivial parameters.

Lastly, it remains to prove this data satisfies the appropriate coherence conditions [JYJY21, 11.2.14, 11.2.16, and 11.2.17]. We note that all of the coherence conditions establish an equality between pasting diagrams involving only associators/unitors, pentagonators/2-unitors, and the unitor $\epsilon^{\boxtimes^{\mathfrak{p}}}$. As in our case all of them are identity reparameterisations, it is easy to see that all of the required diagrams are equal. This concludes the proof that $\mathbf{Para}_\bullet(\mathcal{C})$ is a monoidal bicategory. □

Proposition 3.40. *Let $(\mathcal{C}, \bullet)$ be a strict monoidal $\mathcal{M}$-actegory. Then $\mathbf{Para}_\bullet(\mathcal{C})$ is a monoidal 2-category if and only if $\mathcal{M}$ is commutative monoidal.*

Proof. To show $\mathbf{Para}_\bullet(\mathcal{C})$ is a monoidal 2-category we additionally need $\boxtimes^{\mathfrak{p}}$ and $J^{\mathfrak{p}}$ defining the monoidal bicategory structure to be strict 2-functors. It is easy to see that strictness of $\mathcal{M}$ implies strictness of $J^{\mathfrak{p}}$, as its natural isomorphisms are defined via the unitors of $\mathcal{M}$. When it comes to $\boxtimes^{\mathfrak{p}}$, only one of the natural isomorphisms becomes identity when $\mathcal{M}$ is strict: $\epsilon^{\boxtimes^{\mathfrak{p}}}$. The other natural isomorphism $\mu^{\boxtimes^{\mathfrak{p}}}$ is defined using the braiding of $\mathcal{M}$ which is not identity when $\mathcal{M}$ is strict

[1] This is why in the definition of a monoidal actegory the opunitor of $\bullet$ was redundant (Definition 3.33).

— only when $\mathcal{M}$ is *commutative monoidal*. It is routine to check that making $\mathcal{M}$ commutative monoidal makes $\mu^{\boxtimes^{\mathfrak{p}}}$ identity, $\boxtimes^{\mathfrak{p}}$ a strict 2-functor, and $\mathbf{Para}_\bullet(\mathcal{C})$ a monoidal 2-category.[2] It can also be seen that if $\mathbf{Para}_\bullet(\mathcal{C})$ is a monoidal 2-category, then $\boxtimes^{\mathfrak{p}}$ must be a strict 2-functor, $\mu^{\boxtimes^{\mathfrak{p}}}$ identity, meaning the braiding of $\mathcal{M}$ is strict, i.e. $\mathcal{M}$ is commutative monoidal. □

Theorem 3.41. *Let $(\mathcal{C}, \bullet)$ be a monoidal $\mathcal{M}$-actegory. Then the following three categories are monoidal.*

$$\pi_{0*}(\mathbf{Para}_\bullet(\mathcal{C})) \quad , \quad \mathsf{Iso}_*(\mathbf{Para}_\bullet(\mathcal{C})) \quad \textit{and} \quad \mathsf{Epi}_*(\mathbf{Para}_\bullet(\mathcal{C}))$$

On the other hand, $\mathsf{Ob}_*(\mathbf{Para}_\bullet(\mathcal{C}))$ is not a monoidal category.

Proof. Following the reasoning in the proof of Prop. 3.40 — we see that strictification of the monoidal structure boils down to ensuring $\boxtimes^{\mathfrak{p}}$ is a strict 2-functor, which consequently boils down to the requirement that the interchange law holds strictly, i.e. such that either of the ways of composing the parametric maps yield the same result. Since we are dealing with equivalence classes, "same" means up to a corresponding reparameterisation. This is something that holds for π_0, Core and Epi as they can all use the braiding $\beta_{Q,R}$ of $\mathcal{M}$ to define their quotient (which is an isomorphism, and thus an epimorphism). On the other hand, this does not hold for Ob: it is easy to see that the objects $(P \otimes Q) \otimes (R \otimes S)$ and $(P \otimes R) \otimes (Q \otimes S)$ fall into different equivalence classes. □

Proposition 5.9 (A morphism of actegories induces a morphism of parametric bicategories)**.** *Let $(\mathcal{C}, \mathcal{M}, \bullet)$ and $(\mathcal{D}, \mathcal{N}, \boxtimes)$ be two actegories. Let $(R, R^\sharp, \ell)$ be a morphism of actegories $(\mathcal{C}, \mathcal{M}, \bullet) \to (\mathcal{D}, \mathcal{N}, \boxtimes)$.*

Then **Para** *respects that structure, inducing a pseudofunctor*

$$\mathbf{Para}((R, R^\sharp, \ell)) : \mathbf{Para}_\bullet(\mathcal{C}) \to \mathbf{Para}_\boxtimes(\mathcal{D})$$

with the following data.

- *On* ***objects*** *it borrows the action of $R^\sharp$;*

[2] This was first pointed out to me by Dan Shiebler.

- *On* ***morphisms*** *at acts as below, and graphically depicted in Fig. 5.4:*

$$\mathbf{Para}_{\bullet}(\mathcal{C}) \longrightarrow \mathbf{Para}_{\boxtimes}(\mathcal{D})$$

$$\begin{array}{ccc} A & \longmapsto & R^\sharp(A) \\ {\scriptstyle (P,f)}\downarrow & \longmapsto & \downarrow{\scriptstyle (R(P),R^\sharp_\ell(f))} \\ B & \longmapsto & R^\sharp(B) \end{array}$$

where $R^\sharp_\ell(f)$ *is shorthand for the composite*

$$R^\sharp(A) \boxtimes R(P) \xrightarrow{\ell_{A,P}} R^\sharp(A \bullet P) \xrightarrow{R^\sharp(f)} R^\sharp(B)$$

- *On* ***2-morphisms*** *it borrows the action of* R*.*

Proof. As this functor is only *pseudo*, we have to define the respective composition and identities 2-cells, and show that they are invertible. For composition, starting with two composable 1-cells in $\mathbf{Para}_{\bullet}(\mathcal{C})$

$$A \xrightarrow{(P,f)} B \xrightarrow{(Q,g)} C$$

i.e. with

$$\begin{array}{ll} P : \mathcal{M} & \\ A \bullet P \xrightarrow{f} B & \end{array} \quad \text{and} \quad \begin{array}{ll} Q : \mathcal{M} & \\ B \bullet Q \xrightarrow{g} C & \end{array}$$

there are two ways to map them to $\mathbf{Para}_{\boxtimes}(\mathcal{D})$ using $\mathbf{Para}((R, R^\sharp, \ell))$. We can either map them individually and then compose them, or first compose them, and them map the result. In the first scenario, mapping them individually yields

$$R^\sharp(A) \boxtimes R(P) \xrightarrow{R^\sharp(f)} R^\sharp(B) \quad \text{and} \quad R^\sharp(B) \boxtimes R(Q) \xrightarrow{R^\sharp_\ell(g)} R^\sharp(C).$$

When composed, these two maps yield a morphism in $\mathbf{Para}_{\boxtimes}(\mathcal{D})$

$$R^\sharp(A) \boxtimes (R(P) \otimes R(Q)) \xrightarrow{\mu^{\boxtimes}_{R^\sharp(A),R(P),R(Q)}} (R^\sharp(A) \boxtimes R(P)) \boxtimes R(Q) \xrightarrow{R^\sharp_\ell(f) \boxtimes R(Q)} R^\sharp(B) \boxtimes R(Q) \xrightarrow{R^\sharp_\ell(g)} R^\sharp(C)$$

whose parameter space is $R(P) \otimes R(Q)$.[3] In the second scenario, first composing them yields

$$A \bullet (P \otimes Q) \xrightarrow{\mu^{\bullet}_{A,P,Q}} (A \bullet P) \bullet Q \xrightarrow{f \bullet Q} B \bullet Q \xrightarrow{g} C$$

and the image of this composite under the mapping yields the following morphism in $\mathbf{Para}_{\boxtimes}(\mathcal{D})$:

$$R^{\sharp}(A) \boxtimes (R(P \otimes Q)) \xrightarrow{R^{\sharp}_{\ell}(\mu^{\bullet}_{A,P,Q} \fatsemi (f \bullet Q) \fatsemi g)} R^{\sharp}(C)$$

As these two morphisms have different parameter objects they clearly aren't strictly equal. Though there is an invertible 2-cell connecting them — $R(P \otimes Q) \to R(P) \otimes R(Q)$ — the one given by the oplaxator $\mu^{-1}_{P,Q}$. Similar arguments can be given for the unitor of this lax functor which is given by the opunitor of the same monoidal functor R.[4] Showing composition and identities are preserved up to a coherent isomorphism is routine. □

Proposition C.3. *Let $\mathcal{C}$ be a cartesian category, treated as a locally discrete 2-category. Then we can define an identity-on-objects oplax functor (S, μ, ϵ)*

$$\mathcal{C} \xrightarrow{\quad S \quad} \mathbf{coPara}(\mathcal{C})$$

$$\begin{array}{ccc} A & \longmapsto & A \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{graph}(f)} \\ B & \longmapsto & B \end{array}$$

As it is only oplax, its identities and composition are preserved only up to a coherent 2-cell.

- ***Identity.*** *For every $A : \mathcal{C}$ we have the opunitor: a natural transformation $\epsilon_A : S(id^{\mathcal{C}}_A) \Rightarrow id^{\mathbf{coPara}(\mathcal{C})}_A$. It unpacks to a reparameterisaton of type $(A, \mathsf{graph}(f)) \to (1, \lambda_A)$, and is given by the terminal map $!_A : A \to 1$;*

- ***Composition.*** *For every $f : A \to B$ and $g : B \to C$ in $\mathcal{C}$ the oplaxator is a natural transformation $\mu_{f,g} : S(f \fatsemi g) \Rightarrow S(f) \fatsemi S(g)$ which unpacks to a reparameterisation of type $(A, \mathsf{graph}(f \fatsemi g)) \Rightarrow (A \times B, \mathsf{graph}(f) \fatsemi_{\mathsf{p}} \mathsf{graph}(g))$. This reparameteristaion is given by the map*

[3] We're overloading notation by using $\otimes$ for the monoidal structure of $\mathcal{M}$ and $\mathcal{N}$.
[4] Note that for a lax actegory the induced functor would be a lax functor instead of a pseudofunctor.

$\mathsf{graph}(f) : A \to A \times B$.[5]

It is routine to check that the oplaxator and opunitor satisfy the constraints of an oplax functor.

We note that there is also a *laxator* for S which is inverse to μ, but there's no unitor.

Proposition C.4. *Let $\mathcal{C}$ be a cartesian category, treated as a locally discrete 2-category. Then we can define an identity-on-objects strict functor $P : \mathbf{coPara}(\mathcal{C}) \to \mathcal{C}$ mapping every morphism $(M, f : A \to M \times B)$ to $f \fatsemi \pi_B$, and every 2-cell to the only possible choice: the identity 2-cells in $\mathcal{C}$. It's routine to check that this is indeed a strict 2-functor.*

[5] There is also a reparameteristaion $\pi_A : A \times B \to A$ going the other way, but it is not the inverse of $\mathsf{graph}(f)$. It is only a one-sided inverse, i.e. $\mathsf{graph}(f)$ is a section of π_A but not vice versa.

Appendix D

Category of elements and the Grothendieck construction

Category of elements

An essential construction in category theory and in the formulation of weighted $\mathbf{coPara}^W$ is the *category of elements*.

Definition D.1 (Category of elements)**.** Let $F : \mathcal{C} \to \mathbf{Set}$ be a functor. The **category of elements** of F — which we denote by $\mathbf{El}(F)$ — is a category with the following data:

- **Objects.** An object is a pair $\binom{C}{c'}$, where C is an object of $\mathcal{C}$ and c' is an element of $F(C)$;
- **Morphisms.** A morphism $\binom{C}{c'} \to \binom{D}{d'}$ consists of a morphism $f : C \to D$ in $\mathcal{C}$ such that $F(f)(c') = d'$. Alternatively,

$$\mathbf{El}(F)\binom{C \quad D}{c' \, , \, d'} \coloneqq \sum_{f : \mathcal{C}(C,D)} F(D)(F(f)(c'), d')$$

  Note that since $F(D)$ is a set, there is only one possible inhabitant of $F(D)(F(f)(c'), d')$. This allows us to present the set of all morphisms in $\mathbf{El}(F)$ in a potentially surprising form, as the set $\mathcal{C}(C, D) \times F(D)$ (see [MSV23, p. 4]);

- **Identity.** Identity morphisms are identities in $\mathcal{C}$;

- **Composition.** Composition is given by composition of morphisms in $\mathcal{C}$.

We often call $\mathcal{C}$ *the base* of the category of elements, and sets $F(c)$ its fibers. Each category of elements has an associated projection $\pi_F : \mathbf{El}(F) \to \mathcal{C}$ which forgets the data associated to F.

Example D.2 (Twisted arrow category)**.** The category of elements of the hom-functor of a category called the *twisted arrow category*, and given some category $\mathcal{C}$ it is denoted by $\mathbf{tw}(\mathcal{C})$.

$$\mathbf{tw}(\mathcal{C}) \coloneqq \mathbf{El}(\mathsf{Hom}_{\mathcal{C}})$$

An object in $\mathbf{tw}(\mathcal{C})$ is a map $f : A \to B$, and a morphism $f \to g$ in this category is a way of factorizing g through f. Following Definition D.1, this category comes equipped with the projection $\pi_{\mathcal{C}} : \mathbf{tw}(\mathcal{C}) \to \mathcal{C}^{\mathsf{op}} \times \mathcal{C}$.

Given a category $\mathcal{C}$ and a functor $F : \mathcal{C} \to \mathbf{Set}$, we have defined the category $\mathbf{El}(F)$. As F is merely an object of $\underline{\mathbf{Cat}}(\mathcal{C}, \mathbf{Set})$ we might wonder whether $\mathbf{El}$ can naturally be defined with respect to its morphisms. The answer yes.

Definition D.3. Let $\mathcal{C}$ be a category, $F, G : \mathcal{C} \to \mathbf{Set}$ two functors, and $\alpha : F \Rightarrow G$ a natural transformation between them. Then we can define a functor

$$\mathbf{El}(\alpha) : \mathbf{El}(F) \to \mathbf{El}(G)$$

as follows.

- **Objects.** It maps every $\binom{X}{x'} : \mathbf{El}(F)$ to $\binom{X}{\alpha_X(x')}$ i.e. it keeps base points the same;
- **Morphisms.** It maps $\binom{f}{f'} : \binom{X}{x'} \to \binom{Y}{y'}$ (f' is here explicit notation for the proof $f' : F(f)(x') = y'$) to $\binom{f}{\alpha_f \fatsemi \alpha_Y(f')} : \binom{X}{\alpha_X(x')} \to \binom{Y}{\alpha_y(y')}$, where the second component is the composite equality
$$G(f)(\alpha_X(x')) \overset{\alpha_f}{=} \alpha_Y(F(f)(x')) \overset{\alpha_Y(f')}{=} \alpha_Y(y')$$
and $\alpha_f : \alpha_X \fatsemi G(f) = F(f) \fatsemi \alpha_Y$ is the naturality condition of α.

Note that the action of this functor is identity on the base category, both on object and morphism level.

What is the intuitive explanation of this definition? Recall: to each component X in $\mathcal{C}$ it assigns a morphism $\alpha_X : F(X) \to G(X)$. We can think of the results of the action of the functor **El** on an object — a category of elements — as a pair consisting of a choice of a component of this natural transformation (say, X) and an input $x : F(X)$ to the morphism α_X. Then the action of **El** on morphisms simply keeps the component X the same, while applying α_X to the input x.

Monoidal structure of the category of elements

Categories of elements are the acting categories of actegories in the definition of weighted **coPara**. This makes us interested in conditions under which they are monoidal, and braided monoidal, as well as conditions under which the associated projection preserves the (braided) monoidal structure.

Proposition D.4 (Monoidal structure of the category of elements)**.** *Let $\mathcal{C}$ have monoidal structure $(\otimes, I)$ and let $F : \mathcal{C} \to$ **Set** have lax monoidal structure (ϕ, ϵ) where $\phi_{X,Y} : F(X) \times F(Y) \to F(X \otimes Y)$ and $\epsilon : 1 \to F(I)$. Then **El**(F) is a monoidal category. Its data is as follows:*

- *The monoidal product of $\binom{C}{c'}$ and $\binom{D}{d'}$ is $\binom{C \otimes D}{\phi_{c,d}(c',d')}$;*
- *The monoidal product of morphisms is given by that of $\mathcal{C}$;*
- *The monoidal unit is $\binom{I}{\epsilon(\bullet)}$, where $\bullet : 1$ is the unique element of monoidal unit 1 in in **Set**;*
- *Associators and unitors are those of $\mathcal{C}$*

*Furthermore, the functor $\pi_F :$ **El**$(F) \to \mathcal{C}$ is strict monoidal.*

If $\mathcal{C}$ is a braided monoidal category and (F, ϕ, ϵ) is a braided monoidal functor, then **El**(F) *becomes a braided monoidal category, and the projection $\pi_F :$ **El**$(F) \to \mathcal{C}$ becomes a strict braided monoidal functor.*

Previously we gave the example of the category of elements of the hom-functor, i.e. the twisted arrow category (Example D.2). In order for this category to be a monoidal category, the hom-functor of a category needs to be lax monoidal. And that indeed happens if the base category is monoidal.

Proposition D.5. *Let $(\mathcal{M}, \otimes, I)$ be a monoidal category. Then $\mathsf{Hom}_{\mathcal{M}} : (\binom{\mathcal{M}^{\mathsf{op}}}{\mathcal{M}}, \binom{\otimes}{\otimes}, \binom{I}{I}) \to$ (**Set**$, \times, 1)$ can be equipped with the lax monoidal structure (ϕ, ϵ) where*

- *The laxator* $\phi_{(M,M'),(N,N')} : \mathcal{M}(M,M') \times \mathcal{M}(N,N') \to \mathcal{M}(M \otimes N, M' \otimes N')$ *tensors the maps together, and is formally given by the action of* $\otimes$ *on morphisms.*
- *The unit* $\epsilon : 1 \to \mathcal{M}(I,I)$ *picks out the identity morphism on* I*.*

Additionally, if $\mathcal{M}$ *is braided, then* $(\mathsf{Hom}_{\mathcal{M}}, \phi, \epsilon)$ *is too.*

Corollary D.6. *If* $\mathcal{C}$ *is a monoidal category, then so is* $\mathbf{tw}(\mathcal{C})$*, and the functor* $\pi_{\mathsf{Hom}} : \mathbf{tw}(\mathcal{C}) \to \mathcal{C}^{\mathsf{op}} \times \mathcal{C}$ *is strict monoidal. Additionally, if* $\mathcal{C}$ *is braided monoidal, then so is* $\mathbf{tw}(\mathcal{C})$*, and the projection is too.*

Grothendieck construction

A category of elements is defined for a functor $F : \mathcal{C} \to \mathbf{Set}$. But often, the codomain of F is instead $\mathbf{Cat}$. Here are some examples.

Example D.7 ([Spi22a, Sec. 3.2])**.** Let $\mathcal{C}$ be a category with products. Then we can form the functor

$$\mathbf{coKl}(= \times -) : \mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$$

mapping an object X to the coKleisli category $\mathbf{coKl}(X \times -)$, and a morphism $f : X \to Y$ to the functor $\mathbf{coKl}(f \times -) : \mathbf{coKl}(Y \times -) \to \mathbf{coKl}(X \times -)$ given by reparameterisation. Furthermore, because $\mathbf{coKl}(X \times -)$ is always a cartesian monoidal category (Prop. D.19) we can form a restriction of this functor to commutative monoids

$$\mathbf{CMon}(\mathbf{coKl}(= \times -)) : \mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$$

mapping an object X to $\mathbf{CMon}(\mathbf{coKl}(X \times -))$, and analogously for reparameterisation.

This example is a subfunctor of the following example of the slice functor — which adds an extra layer of dependence on objects of the base category $\mathcal{C}$.

Example D.8 ([Spi22a, Ex. 3.5])**.** Let $\mathcal{C}$ be a category with pullbacks. Then we can form the contravariant slice functor

$$\mathcal{C}/- : \mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$$

mapping an object X to the slice category $\mathcal{C}/X$ and a morphism $f : X \to Y$ to the functor $f^* : \mathcal{C}/Y \to \mathcal{C}/X$ defined by pullback. Analogously to the example above, because $\mathcal{C}/X$ is always a cartesian monoidal category, we can form a restriction of this functor to commutative monoids

$$\mathbf{CMon}(\mathcal{C}/-) : \mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$$

which maps an object X to $\mathbf{CMon}(\mathcal{C}/X)$, and analogously for morphisms.

When the base $\mathcal{C}$ is $\mathbf{Set}$, there is an functor isomorphic to the one above which gives us the indexed perspective on dependence.

Example D.9 (Families)**.** For any category $\mathcal{C}$ we can form the following functor

$$\mathbf{Set}^{\mathsf{op}} \xrightarrow{\mathbf{Fam}(\mathcal{C})} \mathbf{Cat}$$

$$\begin{array}{ccc} X & \longmapsto & \underline{\mathbf{Cat}}(\mathsf{discr}(X), \mathcal{C}) \\ {\scriptstyle f}\downarrow & & \uparrow{\scriptstyle \underline{\mathbf{Cat}}(\mathsf{discr}(f), \mathcal{C})} \\ Y & \longmapsto & \underline{\mathbf{Cat}}(\mathsf{discr}(Y), \mathcal{C}) \end{array}$$

whose action on objects is given by a category whose objects are families of objects in $\mathcal{C}$, and action on morphisms is given by precomposition with $\mathsf{discr}(f)$. Just like with examples above, it's possible to restrict this functor to commutative monoids.

In such cases, we can talk about a higher-dimensional generalisation of the category of elements: the Grothendieck construction. It comes in two variants.

Definition D.10 ((Covariant) Grothendieck construction (compare [Spi22a, Def. 3.1] and [SS23, Def. A.4]))**.** Let $\mathcal{C}$ be a category. Let $(F, \mu, \epsilon) : \mathcal{C} \to \mathbf{Cat}$ be an oplax functor. Then the Grothendieck construction of F is a category we denote by $\overset{\circlearrowleft}{\mathbf{Gr}}(F)$ (note that arrow goes clockwise), defined by the following data.

- **Objects.** The set of objects is $\sum_{X:\mathcal{C}} F(X)$. We denote a particular object as $\binom{X}{X'}$ where $(X : \mathcal{C}$ and $X' : F(X))$;

- **Morphisms.** For every two objects $\binom{X}{X'}$ and $\binom{Y}{Y'}$ the hom-set is

$$\underset{\circlearrowleft}{\mathbf{Gr}}(F)\binom{X \quad Y}{X' \quad Y'} \coloneqq \sum_{f:\mathcal{C}(X,Y)} F(Y)(F(f)(X'), Y')$$

- **Identity.** For any $\binom{X}{X'}$ the identity morphism is the pair $\binom{\mathrm{id}_X}{\epsilon_{X\,X'}}$;
- **Composition.** The composition of morphisms

$$\binom{X}{X'} \xrightarrow{\binom{f}{f'}} \binom{Y}{Y'} \xrightarrow{\binom{g}{g'}} \binom{Y}{Y'}$$

i.e. of

$$f : X \to Y \qquad g : Y \to Z$$
$$\text{and}$$
$$f' : F(f)(X') \to Y' \qquad g' : F(g)(Y') \to Z'$$

is the pair $\binom{f \fatsemi g}{\mu_{f,g\,X'} \fatsemi F(g)(f') \fatsemi g'}$ where

$$X \xrightarrow{f} Y \xrightarrow{g} Z$$
$$F(f \fatsemi g)(X') \xrightarrow{\mu_{f,g\,X'}} (F(f) \fatsemi F(g))(X') = F(g)(F(f)(X')) \xrightarrow{F(g)(f')} F(g)(Y') \xrightarrow{g'} Z'$$

Definition D.11 ((Contravariant) Grothendieck construction (compare [Spi22a, Def. 3.1] and [JYJY21, Def. 10.1.2]))**.** Let $\mathcal{C}$ be a category. Let $(F, \mu, \epsilon) : \mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$ be a lax functor. Then the contravariant Grothendieck construction of F is a category we denote by $\underset{\circlearrowleft}{\mathbf{Gr}}(F)$ (note that arrow goes counter-clockwise), defined by the following data.

- **Objects and identities.** Same as in $\underset{\circlearrowleft}{\mathbf{Gr}}(\mathcal{C})$;
- **Morphisms.** For every two objects $\binom{X}{X'}$ and $\binom{Y}{Y'}$ the hom-set is

$$\underset{\circlearrowleft}{\mathbf{Gr}}(F)\binom{X \quad Y}{X' \quad Y'} \coloneqq \sum_{f:\mathcal{C}(X,Y)} F(X)(X', F(f)(Y'))$$

- **Composition.** The composition of morphisms

$$\begin{pmatrix} X \\ X' \end{pmatrix} \xrightarrow{\binom{f}{f'}} \begin{pmatrix} Y \\ Y' \end{pmatrix} \xrightarrow{\binom{g}{g'}} \begin{pmatrix} Y \\ Y' \end{pmatrix}$$

i.e. of

$$\begin{aligned} &f : X \to Y && && g : Y \to Z \\ &f' : X' \to F(f)(Y') && \text{and} && g : Y' \to F(g)(Z') \end{aligned}$$

is the pair $\begin{pmatrix} f \fatsemi g \\ f' \fatsemi F(f)(g') \fatsemi \mu_{f,g\,Z'} \end{pmatrix}$ where

$$\begin{aligned} &X \xrightarrow{f} Y \xrightarrow{g} Z \\ &X \xrightarrow{f'} F(f)(Y') \xrightarrow{F(f)(g')} F(f)(F(g)(Z')) = (F(f) \fatsemi F(g))(Z') \xrightarrow{\mu_{f,g\,Z'}} F(f \fatsemi g)(Z') \end{aligned}$$

It is tedious, but routine to check that this is indeed a category ([JYJY21, Lemma 10.1.8]).

To establish a relationship between these two definitions, we first need to formalise $(-)^{\mathsf{op}}$ as a functor.

Lemma D.12 (Op functor)**.** *The* op *construction taking a category to its opposite forms a 2-functor* $(-)^{\mathsf{op}} : \mathbf{Cat}^{\mathsf{co}} \to \mathbf{Cat}$*. Notably, this functor is* not *contravariant, i.e. it does not flip the direction of 1-cells. It is, however, contravariant on 2-cells, hence the superscript* $^{\mathsf{co}}$*.*

Proposition D.13 ((compare [Spi22a, Prop. 3.2]))**.** *In the following we fix* $\mathcal{C}$ *to be a category. Let* $F : \mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$ *be an oplax functor. Then there is an isomorphism of categories*

$$\underset{\circlearrowleft}{\mathbf{Gr}}(\mathcal{C}^{\mathsf{op}} \xrightarrow{F} \mathbf{Cat})^{\mathsf{op}} \quad \textit{and} \quad \underset{\circlearrowleft}{\mathbf{Gr}}(\mathcal{C}^{\mathsf{co}\mathsf{op}} \xrightarrow{F^{\mathsf{co}}} \mathbf{Cat}^{\mathsf{co}} \xrightarrow{(-)^{\mathsf{op}}} \mathbf{Cat})$$

Dually, let $G : \mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$ *be a lax functor. Then there is an isomorphism of categories*

$$\underset{\circlearrowleft}{\mathbf{Gr}}(\mathcal{C}^{\mathsf{co}\mathsf{op}} \xrightarrow{G^{\mathsf{co}}} \mathbf{Cat}^{\mathsf{co}} \xrightarrow{(-)^{\mathsf{op}}} \mathbf{Cat}) \quad \textit{and} \quad \underset{\circlearrowleft}{\mathbf{Gr}}(\mathcal{C}^{\mathsf{op}} \xrightarrow{G} \mathbf{Cat})$$

Technically, the type of F^{co} *and* G^{co} *is* $\mathcal{C}^{\mathsf{co}\mathsf{op}} \to \mathbf{Cat}^{\mathsf{co}}$*, so we can remove the* $^{\mathsf{co}}$ *superscript from* $\mathcal{C}$*. Compare with [Spi22a, Prop. 3.2] which assumes these are strict 1-functors, and thus a) removes the* $^{\mathsf{co}}$ *superscript entirely, and b) constructs only the top isomorphism.*

In the definition below we refer to the right-hand side of these isomorphisms. Equivalently we

could have referred to categories on the left-hand side.

Definition D.14 (F-charts ([Mye22a, Def. 3.5.0.6 and 3.5.0.8]) and F-lenses ([Spi22a, Def. 3.3]))**.** Let $\mathcal{C}$ be a category. Let $F : \mathcal{C}^{\mathsf{op}} \to \mathbf{Cat}$ be

a lax functor	(resp.)	an oplax functor

Then the category of

F**-charts**	(resp.)	F**-lenses**

is the contravariant Grothendieck construction of

$\mathcal{C}^{\mathsf{op}} \xrightarrow{F} \mathbf{Cat}$	(resp.)	$\mathcal{C}^{\mathsf{op}} \xrightarrow{F^{\mathsf{co}}} \mathbf{Cat}^{\mathsf{co}} \xrightarrow{(-)^{\mathsf{op}}} \mathbf{Cat}$

In all of the examples below, the distinction between lax and oplax dissapears: all the functors are strict (1-)functors.

Example D.15 (**coKl**$(= \times -)$-charts and **coKl**$(= \times -)$-lenses)**.** Denoted respectively as **Chart**$(\mathcal{C})$ and **Lens**$(\mathcal{C})$, these are categories whose objects are pairs $\binom{X}{X'}$ where $X, X' : \mathcal{C}$. Where they differ is in morphisms.

A **coKl**$(= \times -)$-chart of type $\binom{X}{X'} \to \binom{Y}{Y'}$ consists of a pair $\binom{f}{f'}$ where

- $f : \mathcal{C}(X, Y)$;
- $f' : \mathcal{C}(X \times X', Y')$.

A **coKl**$(= \times -)$-lens of type $\binom{X}{X'} \to \binom{Y}{Y'}$ consists of a pair $\binom{f}{f'}$

- $f : \mathcal{C}(X, Y)$;
- $f' : \mathcal{C}(X \times Y', X')$.

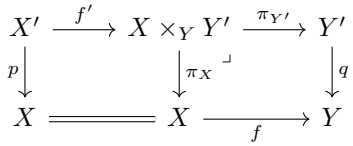
The category of **coKl**$(= \times -)$-charts is referred to in [Jac98, Def. 1.3.1] as the *simple fibration on* $\mathcal{C}$. If we restrict to **CMon**(**coKl**$(= \times -)$)-charts and **CMon**(**coKl**$(= \times -)$)-lenses, we obtain analogous categories to above (denoted as $\mathbf{Chart}_A(\mathcal{C})$ and $\mathbf{Lens}_A(\mathcal{C})$), except the backward objects X' and Y' are now commutative monoids, and backward maps f' are now additive in the second variable.

Example D.16 ($(\mathcal{C}/-)$-charts and $(\mathcal{C}/-)$-lenses)**.** Denoted as **DChart**$(\mathcal{C})$ and **DLens**$(\mathcal{C})$, these are categories whose objects are pairs $\binom{X}{p}$ where $X : \mathcal{C}$ and $p : X' \to X$. Where they differ is, like before, in morphisms.

A $(\mathcal{C}/-)$-chart of type $\binom{X}{p} \to \binom{Y}{q}$ consists of a pair $\binom{f}{f'}$ where

- $f : \mathcal{C}(X, Y)$;
- $f' : \mathcal{C}/X(X', X \times_Y Y')$

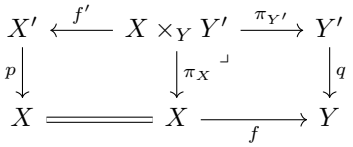
such that the diagram below commutes:



A $(\mathcal{C}/-)$-lens of type $\binom{X}{p} \to \binom{Y}{q}$ consists of a pair $\binom{f}{f'}$

- $f : \mathcal{C}(X, Y)$;
- $f' : \mathcal{C}/X(X \times_Y Y', X')$

such that the diagram below commutes:



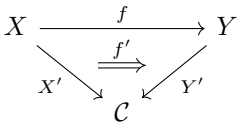
Analogously to the example above, we can restrict to **CMon**$(\mathcal{C}/-)$-charts and **CMon**$(\mathcal{C}/-)$-lenses (denoting them as $\mathbf{DChart}_A(\mathcal{C})$ and $\mathbf{DLens}_A(\mathcal{C})$) obtaining analogous categories except the backward object $p : X' \to X$ is fibrewise a commutative monoid, and the backward map f' is fibrewise additive.

When $\mathcal{C}$ is **Set** there is an isomorphic, indexed formulation of dependent lenses using the **Fam** construction, relying on the fact that $\mathbf{Fam}(\mathbf{Set})(X) \cong \mathbf{Set}/X$.[1]

Example D.17 (**Fam**$(\mathcal{C})$-charts and **Fam**$(\mathcal{C})$-lenses)**.** Often denoted also as $\mathbf{DChart}(\mathcal{C})$ and $\mathbf{DLens}(\mathcal{C})$, these are categories whose objects are pairs $\binom{X}{X'}$ where $X : \mathbf{Set}$ and $X' : \mathsf{discr}(X) \to \mathbf{Set}$. Where they differ is in morphisms.

[1] Dually, we can also say that slice functor is a generalisation of the indexed world to an arbitrary category; it necessitates that the contravariant slice functor can be coherently defined on the category $\mathcal{C}$ which we want to replace **Set** by, which happens if $\mathcal{C}$ has all pullbacks.

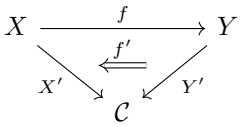
A **Fam**($\mathcal{C}$)-chart of type $\binom{X}{X'} \to \binom{Y}{Y'}$ is a lax triangle



consisting of:

- A function $f : X \to Y$;
- A (vacuously) natural transformation $f' : X' \Rightarrow F(f)(Y')$, i.e. for each $x : X$ a morphism $X'(x) \to F(f)(Y')(x)$ in $\mathcal{C}$.

A **Fam**($\mathcal{C}$)-lens of type $\binom{X}{X'} \to \binom{Y}{Y'}$ is a lax triangle



consisting of:

- A function $f : X \to Y$;
- A (vacuously) natural transformation $f' : F(f)(Y') \Rightarrow X'$, i.e. for each $x : X$ a morphism $F(f)(Y')(x) \to X'(x)$ in $\mathcal{C}$.

Restriction to commutative monoids (as in previous examples) gives us analogous categories, except that the backward object $X' : \mathsf{discr}(X) \to \mathbf{Set}$ is now an indexed commutative monoid, and the backward map f' is additive for each $x : X$.

Notation D.18. When the base category is **Set**, then **Fam**(**Set**)-lenses are isomorphic to $(\mathcal{C}/-)$-lenses, and notation sometimes used for this category is also **Poly** (following [NS23]).

These examples show us the power of Grothendieck construction as applied to a variety of functors. In Prop. 3.45 we remarked that Grothendieck construction can be defined for bicategories. We do not unpack the construction here, rather only mention that its details can be found in ([Bak09]).

Cartesian left-additive structure

Proposition D.19 (**coKl**$(X \times -)$ is cartesian left-additive)**.** *Let $\mathcal{C}$ be a cartesian left-additive category, and let $X : \mathcal{C}$. Then the category* **coKl**$(X \times -)$ *is cartesian left-additive.*

Proof. We need to prove 1) that **coKl**$(X \times -)$ is cartesian monoidal, and 2) that it is additionally left-additive.

1. The monoidal product on objects $A, B :$ **coKl**$(X \times -)$ is given by their underlying product in $\mathcal{C}$. The monoidal product of $f :$ **coKl**$(X \times -)(A, B)$ and $g :$ **coKl**$(X \times -)(C, D)$ is given

by the composite $(f \times^{\mathbb{p}} g)^{\Delta_X}$ which by notation in Definition 3.14 unpacks to the composite

$$X \times (A \times C) \xrightarrow{\Delta_X \times (A \times C)} (X \times X) \times (A \times C) \cong (X \times A) \times (X \times C) \xrightarrow{f \times g} B \times D$$

To show that this is a cartesian monoidal product, we need to equip every object with a commutative comonoid. This can be done by using the commutative comonoids in $\mathcal{C}$ and the canonical embedding $\mathcal{C} \to \mathbf{coKl}(X \times -)$. It is straightforward to show that such comonoids are natural.

2. We need to give a coherent choice of monoids to every object $A : \mathbf{coKl}(X \times -)$. This is simply given by the embedding $\mathcal{C} \to \mathbf{coKl}(X \times -)$. It is routine to show that this satisfies the two commutative diagrams in Definition 2.17.

□

Lemma D.20. *Let $F : \mathcal{C} \to \mathcal{D}$ be a morphism in* **CLACat**, *i.e. a cartesian left-additive functor. Let $f : A \to B$ be an additive morphism in $\mathcal{C}$. Then $F(f) : F(A) \to F(B)$ is also additive.*

Proof. To prove additivity of $F(f) : F(A) \to F(B)$, we need to prove it preserves the monoid structure. Since preservation of monoid structure is defined using two commutative diagrams (Definition 2.19), this follows from the fact that functors preserve commutative diagrams and the fact that F is a product-preserving functor (allowing us to replace $F(A \times A)$ with $F(A) \times F(A)$). □

Proposition 6.16. *Let $F : \mathcal{C} \to \mathcal{D}$ be a cartesian left-additive functor. This induces a cartesian left-additive functor $\mathbf{Lens}_A(F)$ between the corresponding categories of lenses:*

$$\begin{array}{ccc} \mathbf{Lens}_A(\mathcal{C}) & \xrightarrow{\mathbf{Lens}_A(F)} & \mathbf{Lens}_A(\mathcal{D}) \\ \\ \begin{pmatrix} A \\ A' \end{pmatrix} & \longmapsto & \begin{pmatrix} F(A) \\ F(A') \end{pmatrix} \\ \Big\downarrow {\scriptstyle \begin{pmatrix} f \\ f' \end{pmatrix}} & & \Big\downarrow {\scriptstyle \begin{pmatrix} F(f) \\ \overline{f'} \end{pmatrix}} \\ \begin{pmatrix} B \\ B' \end{pmatrix} & \longmapsto & \begin{pmatrix} F(B) \\ F(B') \end{pmatrix} \end{array} \tag{6.16}$$

where $\overline{f'} \coloneqq F(A') \times F(B') \cong F(A' \times B') \xrightarrow{F(f')} F(A')$.

Proof. We need to prove that $\mathbf{Lens}_A(F)$ is a cartesian left-additive functor. To prove it is a functor, we need to:

- Define its action on objects and morphisms. We've done this in Prop. 6.16 itself;
- Prove additivity of $\overline{f^\sharp}$. This follows from Lemma D.20;
- Prove identities are preserved. The identity $\binom{\mathrm{id}_A}{\pi_2} : \binom{A}{A'} \to \binom{A}{A'}$ in the domain gets mapped to $\binom{F(\mathrm{id}_A)}{F(\pi_2)}$. By preservation of identities and products of F this is equal to the identity map on $\binom{F(A)}{F(A')}$.
- Prove composition is preserved. This can be be verified by routine.

To prove that it is additionally cartesian, we need to show that the image of every comonoid $(\binom{A}{A'}, !_{\binom{A}{A'}}, \Delta_{\binom{A}{A'}})$ is also a comonoid, and that all maps preserve comonoids. We can understand the first part in terms of actions on the counit and comultiplication of the comonoid.

- **Counit.** The action on the counit unpacks to the pair $\binom{F(!_{\binom{A}{A'}})}{F(!_{A\times 1} \fatsemi 0_A)}$. By preservation of terminal and additive maps of F this morphism is equal to the counit of $\binom{F(A)}{F(A')}$.
- **Comultiplication.** The action on the comultiplication unpacks to $\binom{(F(\Delta_A)}{F(\pi_{2,3} \fatsemi +_A)}$. By F's preservation of products and additive morphisms this morphism is equal to the comultiplication of $\binom{F(A)}{F(A')}$.

It is routine to show it obey the corresponding laws and form a comonoid.

For the second part we need to show that the image of every lens $\binom{f}{f^\sharp} : \binom{A}{A'} \to \binom{B}{B'}$ preserves these comonoids. For the forward part this is true because F preserves products. For the backwards part this is true because F is left-additive.

Lastly, we need to prove that this functor is additionally left-additive. This means that it preserves the monoid $(\binom{A}{A'}, 0_{\binom{A}{A'}}, +_{\binom{A}{A'}})$ of every object. We unpack the action of $\mathbf{Lens}_A(F)$ on the unit $0_{\binom{A}{A'}}$ and multiplication $+_{\binom{A}{A'}}$ below.

- **Unit.** The action on the unit unpacks to the pair $\binom{F(0_A)}{F(!_{1\times A})}$. By preservation of additive and terminal maps of F this morphism is equal to the unit of $\binom{F(A)}{F(A)}$;
- **Multiplication.** The action on the multiplication unpacks to the pair $\binom{F(+_A)}{F(\pi_3 \fatsemi \Delta_A)}$. By preservation of coadditive maps and products of F this morphism is equal to the multiplication of $\binom{F(A)}{F(A')}$.

Seeing as these monoids in the codomain are of the same form as those in the domain, it is routine to show that they obey the monoid laws. This concludes the proof that $\mathbf{Lens}_A(F)$ is a cartesian left-additive functor. □

Definition D.21 (compare [CCG+20, Def. 13])**.** A cartesian left-additive category $\mathcal{C}$ is called a **cartesian reverse derivative category** if it comes equipped with a reverse differential combinator

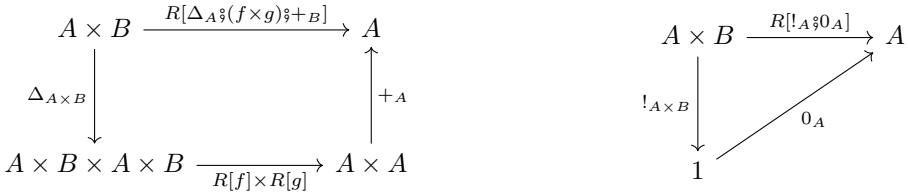
$$R : \mathcal{C}(A, B) \to \mathcal{C}(A \times B, A)$$

such that the following seven axioms hold expressed as commutative diagrams:

1. **Additivity of reverse differentiation.**
   For every $f, g : A \to B$ it is required that

   $$R[f + g] = R[f] + R[g] \quad \text{and} \quad R[0] = 0$$

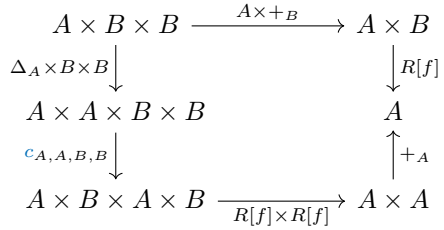
   It is useful unpack the $+$ notation for morphisms (Remark 2.18) in terms of commutative diagrams below

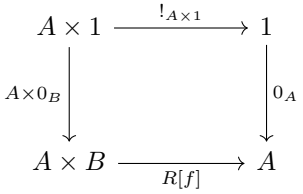
   

2. **Additivity of reverse derivative in the second component.**
   For every $f : A \to B$ it is required that

   $$\langle a, b + c\rangle R[f] = \langle a, b\rangle R[f] + \langle a, c\rangle R[f] \quad \text{and} \quad \langle a, 0\rangle R[f] = 0$$

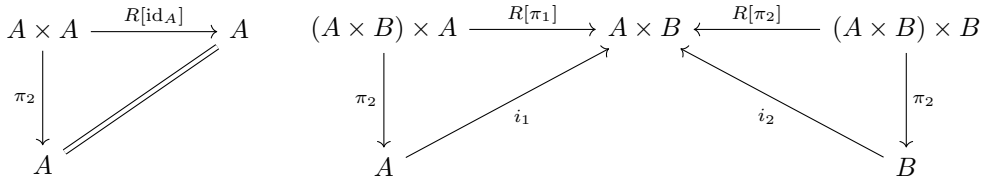
   Again, we find it instructive to unpack the underlying diagrams:





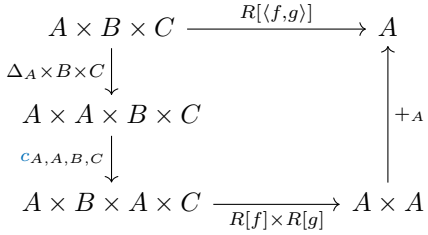
3. **Coherence with identities and projections.**

   For any two objects $A, B : \mathcal{C}$ it is required that the following diagrams commute:

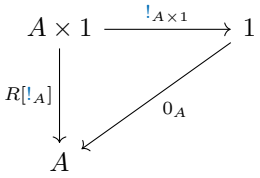


4. **Coherence with pairings.**

   For every $f : A \to B$ and $g : A \to C$ it is required that the following diagram commutes:

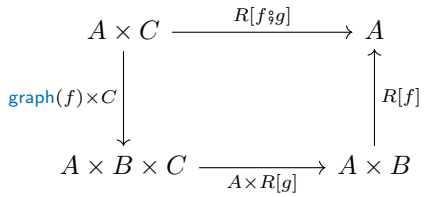


   and for every $A : \mathcal{C}$ it is required that the following diagram commmutes:



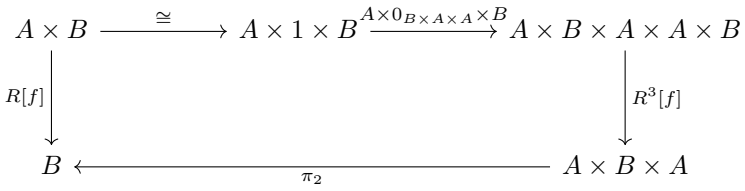
5. **Reverse chain rule.**

   For all composable pairs $f : A \to B$ and $g : B \to C$ the following diagram must commute:



6. **Linearity of the reverse derivative in its second component.**

   For every $f : A \to B$ the following diagram must commute:

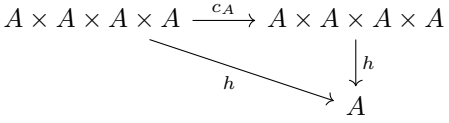


7. **Symmetry of mixed partial derivatives.**

   Given the following compact, but unilluminating definition of h

$$h \coloneqq (i_1 \times 1_{A \times A}) \fatsemi R[R[(i_1 \times A) \fatsemi R[R[f]] \fatsemi \pi_2]] \fatsemi \pi_2$$

   the symmetry of mixed partial derivatives can be simply stated as the requirement that the following diagram commutes:



   where c_A is the interchanger (Definition A.10).

Note that RD6 and RD7 are independent from the others (Remark 6.22).

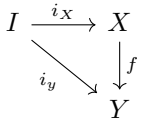
Appendix E

Miscellaneous

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

The naturality condition is superfluous, which can be seen in the following corollary.

Corollary E.2. *If a category $\mathcal{C}$ has an initial object I, then for every $f : X \to Y$ the triangle*



commutes.

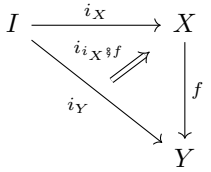
Proof. Since it holds that $\mathcal{C}(I, Y) \cong 1$, this means that there is only one morphism that can be chosen as the composite $i_X \fatsemi f$ — precisely i_Y. □

Definition E.3 (Quasi-initial object)**.** An object I in a bicategory $\mathcal{B}$ is **quasi-initial**[1] if for every object X there is a laxly natural adjunction $R : \mathcal{B}(I, X) \leftrightarrows \mathbf{1} : L$ where $\mathbf{1}$ is the terminal category in **Cat**.

Since we are in **Cat** the functor R is the terminal functor. And a choice of a left adjoint to it is equivalent to the choice of an initial object in the domain category $\mathcal{B}(I, X)$. We will suggestively label this object as i_X, and unique maps from it to any other $g : I \to X$ as i_g. The lax naturality condition is superfluous, which can be seen in the following corollary.

[1] This is mostly referred to as a quasi-initial object in [nLa23], though it does not appear that these concepts have been studied sufficiently for consistent modern terminology to be adopted.

Corollary E.4. *If a bicategory $\mathcal{B}$ has a quasi-initial object I, then for every $f : X \to Y$ the triangle*



laxly commutes up to the unique 2-cell $1_{i_X \fatsemi f}$.

Proof. Since i_Y is initial in $\mathcal{B}(I, Y)$, there is only one map of type $i_Y \Rightarrow i_X \fatsemi f$ which we have already labeled as $i_{i_X \fatsemi f}$ since we know that it exists and is uniquely defined. □

Proposition E.5. *Let $F : \mathcal{C} \to \mathcal{D}$ be a functor. If $\mathcal{C}$ has a terminal object 1, then $\mathsf{colim} F \cong F(1)$. Dually, if $\mathcal{C}$ has an initial object 0, then $\lim F \cong F(0)$*

Note that the converse of the above propositions doesn't necessarily hold.

Proposition E.6 (Colimit as the set of connected components of the category of elements)**.** *Let $F : \mathcal{C} \to \mathbf{Set}$ be a functor. Then*

$$\mathsf{colim} F \cong \pi_0(\mathbf{El}(F))$$

Proof. See [Kel82, (3.35)] or [FS19, Theorem 6.37.]. The latter reference unpacks the above isomorphism as the well-known formula for computing colimits in **Set**. We leave it to the reader to check that the equivalence relation described therein is the one arising as the image of π_0. □

Proposition E.7 (From **Set**-weighted colimits to colimits (compare [Kel82, (3.34)]), [Lor21, Prop. 4.1.13 WC3])**.** *Let $F : \mathcal{C} \to \mathcal{D}$ and $W : \mathcal{C}^{\mathsf{op}} \to \mathbf{Set}$ be two functors. Then*

$$\mathsf{colim}^W F \cong \mathsf{colim}(\mathbf{El}(W)^{\mathsf{op}} \xrightarrow{\pi_W^{\mathsf{op}}} \mathcal{C} \xrightarrow{F} \mathcal{D}) \tag{E.8}$$

Proof.

$$\begin{aligned}
&\mathsf{colim}^W F \\
(\text{Duality of limits and colimits}) &\cong \mathsf{lim}^W F^{\mathsf{op}} \\
([\text{Lor21, Prop. 4.1.11}]) \qquad &\cong \mathsf{lim}(\mathbf{El}(W) \xrightarrow{\pi_W} \mathcal{C}^{\mathsf{op}} \xrightarrow{F^{\mathsf{op}}} \mathcal{D}^{\mathsf{op}}) \\
(\text{Duality of limits and colimits}) &\cong \mathsf{colim}(\mathbf{El}(W)^{\mathsf{op}} \xrightarrow{\pi_W^{\mathsf{op}}} \mathcal{C} \xrightarrow{F} \mathcal{D})
\end{aligned}$$

□

Observe that if the domain of our functor is $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$, then there is always a choice of a canonical weight $\mathsf{Hom}_{\mathcal{C}} : \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathbf{Set}$. As a matter of fact, the colimit of some $F : \mathcal{C} \times \mathcal{C}^{\mathsf{op}} \to \mathcal{D}$ weighted

by $\mathsf{Hom}_{\mathcal{C}}$ is so ubiquitous that is has a special name: it's called the *coend* of F.

This gives us two further characterisations of weighted colimits of functors of type $\mathcal{C} \times \mathcal{C}^{\mathsf{op}} \to \mathcal{D}$.

Proposition E.9 (Weighted colimit as coend (compare [Lor21, Remark 4.1.13.1]))**.** *Let $F : \mathcal{C} \to \mathcal{D}$ and $W : \mathcal{C}^{\mathsf{op}} \to \mathbf{Set}$ be two functors. Then*

$$\mathsf{colim}^W F \cong \int^{C:\mathcal{C}} F(C) \bullet W(C)$$

holds assuming $\mathcal{D}$ admits the respective coend (which happens when $\mathcal{D}$ is cocomplete), where $\bullet : \mathcal{D} \times \mathbf{Set} \to \mathcal{D}$ is the respective $\mathbf{Set}$-copower of $\mathcal{D}$ arising from its coproduct structure (see [CG23, Ex. 3.2.6]).

In many of our use cases (such as Eq. (4.15)) the domain of F will be the product $\mathcal{M} \times \mathcal{M}^{\mathsf{op}}$, and codomain $\mathbf{Set}$, in which case the copower reduces to the cartesian product of $\mathbf{Set}$.

Lemma E.10 (Coend over a set)**.** *Let X be a set, and let $F : \mathsf{discr}(X)^{\mathsf{op}} \times \mathsf{discr}(X) \to \mathbf{Set}$ be a functor.*[2] *Then*

$$\int^{A:\mathsf{discr}X} F(A, A) \cong \sum_{A:\mathsf{discr}(X)} F(A, A)$$

We unpack the Yoneda lemma, which comes in a total of four variants.[3]

Proposition E.11 (Yoneda Lemma ([Lor21, Prop. 2.2.1]))**.** *Let $F : \mathcal{C} \to \mathbf{Set}$ be a functor (respectively $G : \mathcal{C}^{\mathsf{op}} \to \mathbf{Set}$). Then the following isomorphisms hold, natural in A:*

$$F(A) \cong \int_{X:\mathcal{C}} \underline{\mathbf{Set}}(\mathcal{C}(A, X), F(X)) \quad \textit{(and, respectively)} \quad G(A) \cong \int_{X:\mathcal{C}^{\mathsf{op}}} \underline{\mathbf{Set}}(\mathcal{C}(X, A), G(X))$$

Proposition E.12 (coYoneda ([Lor21, Prop. 2.2.1]))**.** *Let $F : \mathcal{C} \to \mathbf{Set}$ be a functor (respectively $G : \mathcal{C}^{\mathsf{op}} \to \mathbf{Set}$). Then the following isomorphisms hold, natural in A:*

$$\int^{X:\mathcal{C}} F(X) \times \mathcal{C}(X, A) \cong F(A) \quad \textit{(and, respectively)} \quad \int^{X:\mathcal{C}^{\mathsf{op}}} \mathcal{C}(A, X) \times G(X) \cong G(A)$$

Proof. We prove Prop. E.11 (left). The rest follows analogously.

[2] Here the $^{\mathsf{op}}$ is unnecessary (as $\mathsf{discr}(X)$ does not have any non-identity morphisms) but we have left it in nonetheless.

[3] To really understand Yoneda lemma, one needs to understand how it is related to all the other concepts in category theory.

- **Left to right.** Given an element $a : F(A)$ we need to produce a natural transformation $\mathcal{C}(A, -) \Rightarrow F$, i.e. for every $X : \mathcal{C}$ a function $\mathcal{C}(A, X) \to F(X)$ natural in X. This function consumes a $f : A \to X$ and is required to produce an element of $F(X)$. We do the *only thing we can*: apply F to f, obtaining $F(f) : F(A) \to F(X)$ and then use $a : F(A)$ as its input, obtaining $F(f)(a)$. It's straightforward to check that this naturally defined.

- **Right to left.** Going the other way, given a natural transformation $\alpha : \mathcal{C}(A, -) \Rightarrow F$, to obtain an element of $F(A)$ we simply again do the only thing we can: look at the component $\alpha_A : \mathcal{C}(A, A) \to F(A)$ and use $\mathrm{id}_A : A \to A$ as its input.

- **Isomorphism.** To confirm that this is indeed an isomorphism, we need to check that these are inverses. From one side it needs to hold that $(F(\mathrm{id}_A)(a))_A = a$, which is true since $F(\mathrm{id}_A) = \mathrm{id}_{F(A)}$. From the other side it needs to hold that $F(-)(\alpha_A(\mathrm{id}_A)) = \alpha$. At each component $X : \mathcal{C}$ and every map $f : A \to X$ it needs to hold that $F(f)(\alpha_A(\mathrm{id}_A))_X = \alpha_X(f)$. This follows from naturality of α (Eq. (E.13)).

$$\begin{array}{ccc} \mathcal{C}(A,A) & \xrightarrow{\alpha_A} & F(A) \\ {\scriptstyle \mathcal{C}(A,f)}\downarrow & & \downarrow{\scriptstyle F(f)} \\ \mathcal{C}(A,X) & \xrightarrow[\alpha_X]{} & F(X) \end{array} \qquad \begin{array}{ccc} \mathrm{id}_A & \longmapsto & \alpha_A(\mathrm{id}_A) \\ \downarrow & & \downarrow \\ f & \longmapsto & \alpha_X(f) = F(f)(\alpha_A(\mathrm{id}_A)) \end{array} \tag{E.13}$$

□

The following lemma tells us when a profunctor is *representable*, i.e. can be described as a composition of a hom-functor and a functor.

Lemma E.14 (Representable profunctor, [Lor21, Sec. 5.2])**.** *A profunctor $P : \mathcal{M}^{\mathsf{op}} \times \mathcal{M}' \to \mathbf{Set}$ is **representable** by $i : \mathcal{M} \to \mathcal{M}'$ (resp. **corepresentable** by $j : \mathcal{M}' \to \mathcal{M}$) if the following isomorphism holds*

$$P \cong \mathcal{M}'(i(-), -) \quad \textit{(resp.)} \quad P \cong \mathcal{M}(-, j(-))$$

in the functor category $\underline{\mathbf{Cat}}(\mathcal{M}^{\mathsf{op}} \times \mathcal{M}', \mathbf{Set})$.

Definition E.15. Let $\mathcal{C}$ be a cartesian category. Then the **graph** of any morphism $f : A \to B$ in $\mathcal{C}$ is the morphism

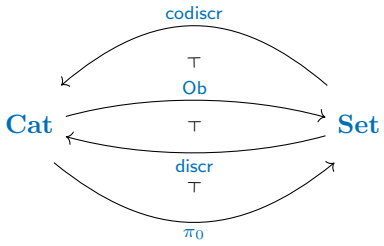
$$\mathsf{graph}(f) := \boxed{A \xrightarrow{\Delta_A} A \times A \xrightarrow{A \times f} A \times B}$$

It is called the "graph" of f because its image is a set of pairs $(a, f(a))$ which we can think of as points in a coordinate plane to be graphed.

Another important concept in this thesis is the notion of a coalgebra over a copointed endofunctor. It arises as a mid-point between the notions of a coalgebra over an endofunctor and coalgebra over a comonad.

Definition E.16 (Coalgebra over a copointed endofunctor)**.** Let $F : \mathcal{C} \to \mathcal{C}$ be endofunctor which is copointed, i.e. is equipped with a natural transformation $\epsilon : F \Rightarrow \mathrm{id}_{\mathcal{C}}$. Then **a coalgebra over** (F, ϵ) consists of a pair (A, f), where $A : \mathcal{C}$ and $f : A \to F(A)$ is a morphism in $\mathcal{C}$ such that the diagram $\begin{array}{ccc} F(A) & \xrightarrow{\epsilon_A} & A \\ {\scriptstyle f}\uparrow & \nearrow & \\ A & & \end{array}$ commutes.

Proposition E.17 (Adjoint quadruple)**.** *Between categories* **Cat** *and* **Set** *there exists a quadruple of functors, all adjoint one to the next:*



More specifically, they are:

- codiscr : **Set** → **Cat**, *the functor that sends a set X to a category whose objects are elements of X such that there exists a unique morphisms between every two objects*[4]*;*
- Ob : **Cat** → **Set**, *the left adjoint of* codiscr *that sends a category to the set of its underlying objects;*
- discr : **Set** → **Cat**, *the left adjoint of* Ob *that sends a set X to a category whose objects are elements of X and whose morphisms are only identity morphisms;*
- π_0 : **Cat** → **Set**, *the left adjoint of* discr *that sends a category to the set of its connected components.*

[4] Often also called a *chaotic*, or *indiscrete* category

Definition E.18 (Strong endofunctor)**.** Let $(\mathcal{C}, \otimes, I)$ be a monoidal category. Then an endofunctor $F : \mathcal{C} \to \mathcal{C}$ is called **(right) strong** if it comes equipped with a natural transformation

$$\begin{array}{ccc} \mathcal{C} \times \mathcal{C} & \xrightarrow{F \times \mathcal{C}} & \mathcal{C} \times \mathcal{C} \\ {\scriptstyle\otimes}\downarrow & \swarrow_{s} & \downarrow{\scriptstyle\otimes} \\ \mathcal{C} & \xrightarrow[F]{} & \mathcal{C} \end{array}$$

whose component at each $X, Y : \mathcal{C}$ is explicitly the map $s_{X,Y} : F(X) \otimes Y \to F(X \otimes Y)$ making the following diagrams commute for all $X, Y, Z : \mathcal{C}$:

$$\begin{array}{ccc} F(X) \otimes I & & \\ {\scriptstyle s_{X,I}}\downarrow & \searrow^{\rho_{F(X)}} & \\ F(X \otimes I) & \xrightarrow[F(\rho_X)]{} & F(X) \end{array} \qquad \begin{array}{ccccc} (F(X) \otimes Y) \otimes Z & \xrightarrow{s_{X,Y} \otimes Z} & F(X \otimes Y) \otimes Z & \xrightarrow{s_{X \otimes Y, Z}} & F((X \otimes Y) \otimes Z) \\ {\scriptstyle \alpha_{F(X),Y,Z}}\downarrow & & & & \downarrow{\scriptstyle F(\alpha_{X,Y,Z})} \\ F(X) \otimes (Y \otimes Z) & & \xrightarrow[s_{X,(Y \otimes Z)}]{} & & F(X \otimes (Y \otimes Z)) \end{array}$$

Definition E.19 (Monoidal closed category)**.** Let $(\mathcal{C}, \otimes, I)$ be a monoidal category. We call it a *monoidal closed category* if, for every $C : \mathcal{C}$ the functor $- \otimes C$ has a right adjoint denoted by $\underline{\mathcal{C}}(C, -)$ which forms the internal hom functor $\underline{\mathcal{C}}(-, -) : \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \to \mathcal{C}$ such that there is isomorphism

$$\mathcal{C}(X \otimes Y, Z) \cong \mathcal{C}(X, \underline{\mathcal{C}}(Y, Z))$$

natural in X, Y, Z.

Remark E.20. Notably, in any monoidal closed category for each $X, Y : \mathcal{C}$ there exists the map $\mathsf{eval}_{X,Y} : \underline{\mathcal{C}}(X, Y) \otimes X \to Y$ called the **evaluation map**. In **Set**, eval consumes a function $f : X \to Y$, an element $x : X$ and produces $f(x) : Y$.

Remark E.21 (The functor $L_{\epsilon,e}$ in [FST21, Theorem III.2] not well-defined)**.** The functor $L_{\epsilon,e} :$ **Para** $\to$ **Learn** is not well-defined with respect to the morphism equivalence classes.[5,6] That is, there are parametric maps where $(P, f) \sim (P', f')$ in **Para**, but $L_{\epsilon,e}((P, f)) \not\sim L_{\epsilon,e}((P', f'))$ in **Learn**. For a specific example, consider two $\mathbb{R}$-parametric morphisms of type $\mathbb{R} \to \mathbb{R}$:

- $f(a, p) = 3p + a$

[5] This counterexample was pointed out to me by Geoffrey Cruttwell.

[6] Here we are using **Para** as defined in [FST21, Def. III.1], which is isomorphic to $\mathsf{Iso}_*(\mathbf{Para}(\mathbf{Smooth}))$ using the definitions of this thesis.

- $f'(a, p) = p + a$;

They are equivalent in **Para**, as witnessed by the invertible reparameterisation $\alpha : \mathbb{R} \to \mathbb{R} \coloneqq x \mapsto 3x$. To show that they are equivalent in **Learn** we need a reparameterisation α as before, but now additionally by [FST21, p. 4, right column, top] it is required that the following equation

$$U_{f'}(\alpha(p), a, b) = \alpha(U_f(p, a, b)) \tag{E.22}$$

holds for all $p, a, b : \mathbb{R}$. By setting $\epsilon = 1$ and e to the mean squared error we can calculate

- $U_f(p, a, b) = -8p - 3a + 3b$;
- $U_{f'}(p, a, b) = -a + b$;

Substituting α and the two results above in Eq. (E.22) yields $-24p - 9a + 9b = -a + b$. This clearly doesn't hold for all $p, a, b \in \mathbb{R}$.

Bibliography

[AAG03] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Categories of Containers. In Andrew D. Gordon, editor, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 23–38, Berlin, Heidelberg, 2003. Springer. doi:10.1007/3-540-36576-1_2.

[AAGM03] Michael Abbott, Thorsten Altenkirch, Neil Ghani, and Conor McBride. Derivatives of Containers. In Gerhard Goos, Juris Hartmanis, Jan Van Leeuwen, and Martin Hofmann, editors, *Typed Lambda Calculi and Applications*, volume 2701, pages 16–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. Series Title: Lecture Notes in Computer Science. URL: http://link.springer.com/10.1007/3-540-44904-3_2, doi:10.1007/3-540-44904-3_2.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN, December 2017. arXiv:1701.07875 [cs, stat]. URL: http://arxiv.org/abs/1701.07875, doi:10.48550/arXiv.1701.07875.

[ADG+16] Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/hash/fb87582825f9d28a8d42c5e5e5e8b23d-Abstract.html.

[ALS10] Thorsten Altenkirch, Paul Levy, and Sam Staton. Higher-Order Containers. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen,

Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *Programs, Proofs, Processes*, volume 6158, pages 11–20. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. Series Title: Lecture Notes in Computer Science. URL: http://link.springer.com/10.1007/978-3-642-13962-8_2, doi:10.1007/978-3-642-13962-8_2.

[AM75] M. A. Arbib and E. G. Manes. A Categorist's view of automata and systems. In Ernest Gene Manes, editor, *Category Theory Applied to Computation and Control*, Lecture Notes in Computer Science, pages 51–64, Berlin, Heidelberg, 1975. Springer. doi:10.1007/3-540-07142-3_61.

[And72] P. W. Anderson. More Is Different. *Science*, 177(4047):393–396, August 1972. Publisher: American Association for the Advancement of Science. URL: https://www.science.org/doi/10.1126/science.177.4047.393, doi:10.1126/science.177.4047.393.

[AP20] Mario Alvarez-Picallo. Change actions: from incremental computation to discrete derivatives, June 2020. arXiv:2002.05256 [cs]. URL: http://arxiv.org/abs/2002.05256, doi:10.48550/arXiv.2002.05256.

[APGSZ21] Mario Alvarez-Picallo, D. Ghica, David Sprunger, and F. Zanasi. Functorial String Diagrams for Reverse-Mode Automatic Differentiation, 2021. URL: https://www.semanticscholar.org/paper/Functorial-String-Diagrams-for-Reverse-Mode-Alvarez-Picallo-Ghica/1b5ad3c47785ebf2793fb723334328db4d806955.

[Atk18] Robert Atkey. Syntax and Semantics of Quantitative Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 56–65, New York, NY, USA, July 2018. Association for Computing Machinery. doi:10.1145/3209108.3209189.

[Bak09] Igor Bakovic. Grothendieck construction for bicategories, 2009. URL: https://www2.irb.hr/korisnici/ibakovic/sgc.pdf.

[BBCV21] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. arXiv:2104.13478 [cs, stat]. URL: http://arxiv.org/abs/2104.13478, doi:10.48550/arXiv.2104.13478.

[BCC+22] John C. Baez, Simon Cho, Daniel Cicala, Nina Otter, and Valeria de Paiva. Applied Category Theory in chemistry, computing, and social networks, 2022. URL: https://math.ucr.edu/home/baez/mrc_2022.pdf.

[BCG+21] Dylan Braithwaite, Matteo Capucci, Bruno Gavranović, Jules Hedges, and Eigil Fjeldgren Rischel. Fibre optics, December 2021. arXiv:2112.11145 [math]. URL: http://arxiv.org/abs/2112.11145, doi:10.48550/arXiv.2112.11145.

[BCS09] R F Blute, J R B Cockett, and R A G Seely. Cartesian differential categories. *Theory and Applications of Categories*, 22:622–672, January 2009. URL: http://www.tac.mta.ca/tac/volumes/22/23/22-23abs.html.

[BDK+21] Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining Neural Scaling Laws, February 2021. arXiv:2102.06701 [cond-mat, stat]. URL: http://arxiv.org/abs/2102.06701, doi:10.48550/arXiv.2102.06701.

[BE15] John C. Baez and Jason Erbele. Categories in Control. *Theory and Applications of Categories*, 30(24):836–881, May 2015. arXiv:1405.6881 [quant-ph]. URL: http://www.tac.mta.ca/tac/volumes/30/24/30-24abs.html, doi:10.48550/arXiv.1405.6881.

[Bey23] Taliesin Beynon. Rainbow array algebra, November 2023. URL: http://example.org/rainbow-array-algebra/.

[BF18] John C. Baez and Brendan Fong. A Compositional Framework for Passive Linear Networks. *Theory and Applications of Categories*, 33(38):1158–1222, November 2018. arXiv:1504.05625 [math-ph]. URL: http://www.tac.mta.ca/tac/volumes/33/38/33-38abs.html, doi:10.48550/arXiv.1504.05625.

[BFH+18] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-

Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL: http://github.com/google/jax.

[BHSCS23] Dylan Braithwaite, Jules Hedges, and Toby St Clere Smithe. The Compositional Structure of Bayesian Inference. In *DROPS-IDN/v2/document/10.4230/LIPIcs.MFCS.2023.24*. Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2023. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.MFCS.2023.24, doi:10.4230/LIPIcs.MFCS.2023.24.

[BHZ23] Joe Bolt, Jules Hedges, and Philipp Zahn. Bayesian open games. *Compositionality*, 5:9, October 2023. arXiv:1910.03656 [cs, math]. URL: http://arxiv.org/abs/1910.03656, doi:10.32408/compositionality-5-9.

[BK22] Anne Broadbent and Martti Karvonen. Categorical composable cryptography. In Patricia Bouyer and Lutz Schröder, editors, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 161–183, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-030-99253-8_9.

[BKG21] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, April 2021. arXiv:2003.05991 [cs, stat]. URL: http://arxiv.org/abs/2003.05991, doi:10.48550/arXiv.2003.05991.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization, July 2016. arXiv:1607.06450 [cs, stat]. URL: http://arxiv.org/abs/1607.06450, doi:10.48550/arXiv.1607.06450.

[BL95] Y. Bengio and Yann Lecun. Convolutional Networks for Images, Speech, and Time-Series. *The Handbook of Brain Theory and Neural Networks*, 1995.

[BLB17] Aleksandar Botev, Guy Lever, and David Barber. Nesterov's accelerated gradient and momentum as approximations to regularised update descent. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1899–1903, May 2017. Conference Name: 2017 International Joint Conference on Neural Networks (IJCNN) ISBN: 9781509061822 Place: Anchorage, AK, USA Publisher: IEEE. URL: http://ieeexplore.ieee.org/document/7966082/, doi:10.1109/IJCNN.2017.7966082.

[BLLL23] Guido Boccali, Andrea Laretto, Fosco Loregian, and Stefano Luneia. Bicategories of automata, automata in bicategories, March 2023. arXiv:2303.03865 [cs, math]. URL: http://arxiv.org/abs/2303.03865, doi:10.48550/arXiv.2303.03865.

[BLM23] John C. Baez, Owen Lynch, and Joe Moeller. Compositional Thermostatics. *Journal of Mathematical Physics*, 64(2):023304, February 2023. arXiv:2111.10315 [math-ph]. URL: http://arxiv.org/abs/2111.10315, doi:10.1063/5.0089375.

[BMR+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[Boi20] G. Boisseau. String Diagrams for Optics. *International Conference on Formal Structures for Computation and Deduction*, 167:17:1–17:18, 2020. URL: https://www.semanticscholar.org/paper/String-Diagrams-for-Optics-Boisseau/ac617296e978bb7d753cbee3c38eb9c0a13bbe69.

[BP17] John C. Baez and Blake S. Pollard. A Compositional Framework for Reaction Networks. *Reviews in Mathematical Physics*, 29(09):1750028, October 2017. arXiv:1704.02051 [math-ph]. URL: http://arxiv.org/abs/1704.02051, doi:10.1142/S0129055X17500283.

[BPRS17] Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, January 2017.

[BPT23] Mugurel Barcau, Vicentiu Pasol, and George C. Turcas. Composing Bridges, May 2023. arXiv:2305.16435 [cs, math]. URL: http://arxiv.org/abs/2305.16435, doi:10.48550/arXiv.2305.16435.

[BQL21] Joshua Bassey, Lijun Qian, and Xianfang Li. A Survey of Complex-Valued Neural Networks, January 2021. arXiv:2101.12249 [cs, stat]. URL: http://arxiv.org/abs/2101.12249, doi:10.48550/arXiv.2101.12249.

[Bra16] Edwin Brady. *Type-driven Development With Idris*. Manning, 2016. URL: http://www.worldcat.org/isbn/9781617293023.

[Bra21] Tai-Danae Bradley. Entropy as a Topological Operad Derivation. *Entropy*, 23(9):1195, September 2021. arXiv:2107.09581 [cs, math]. URL: http://arxiv.org/abs/2107.09581, doi:10.3390/e23091195.

[Bre18] Spencer Breiner. Workshop Introduction, April 2018. URL: https://www.youtube.com/watch?v=OJTl_7pjvuU.

[BS10] John C. Baez and Michael Shulman. Lectures on N-Categories and Cohomology. In John C. Baez and J. Peter May, editors, *Towards Higher Categories*, The IMA Volumes in Mathematics and its Applications, pages 1–68. Springer, New York, NY, 2010. doi:10.1007/978-1-4419-1524-5_1.

[BS22] Guillaume Boisseau and Paweł Sobociński. String Diagrammatic Electrical Circuit Theory. *Electronic Proceedings in Theoretical Computer Science*, 372:178–191, November 2022. arXiv:2106.07763 [cs]. URL: http://arxiv.org/abs/2106.07763, doi:10.4204/EPTCS.372.13.

[BSZ17] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. The Calculus of Signal Flow Diagrams I: Linear relations on streams. *Information and Computation*, 252:2–29, February 2017. URL: https://www.sciencedirect.com/science/article/pii/S0890540116000390, doi:10.1016/j.ic.2016.03.002.

[Cap22] Matteo Capucci. Seeing double through dependent optics, April 2022. arXiv:2204.10708 [math]. URL: http://arxiv.org/abs/2204.10708, doi:10.48550/arXiv.2204.10708.

[Cap23] Matteo Capucci. Diegetic Representation of Feedback in Open Games. *Electronic Proceedings in Theoretical Computer Science*, 380:145–158, August 2023. arXiv:2206.12338 [cs, math]. URL: http://arxiv.org/abs/2206.12338, doi:10.4204/EPTCS.380.9.

[CC14] J. R. B. Cockett and G. S. H. Cruttwell. Differential Structure, Tangent Structure, and SDG. *Applied Categorical Structures*, 22(2):331–417, April 2014. doi:10.1007/s10485-013-9312-0.

[CCG+20] Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon Plotkin, and Dorette Pronk. Reverse Derivative Categories. In *Leibniz International Proceedings in Informatics (LIPIcs)*, page 16 pages. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany, 2020. Artwork Size: 16 pages Medium: application/pdf Version Number: 1.0. URL: https://drops.dagstuhl.de/opus/volltexte/2020/11661/, doi:10.4230/LIPICS.CSL.2020.18.

[CCL21] J. R. B. Cockett, G. S. H. Cruttwell, and J. S. P. Lemay. Differential Equations in a Tangent Category I: Complete Vector Fields, Flows, and Exponentials. *Applied Categorical Structures*, 29(5):773–825, October 2021. doi:10.1007/s10485-021-09629-x.

[CEG+22] Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregian, Bartosz Milewski, Emily Pillmore, and Mario Román. Profunctor Optics, a Categorical Update, March 2022. arXiv:2001.07488 [cs, math]. URL: http://arxiv.org/abs/2001.07488.

[CG23] Matteo Capucci and Bruno Gavranović. Actegories for the Working Amthematician, December 2023. arXiv:2203.16351 [math]. URL: http://arxiv.org/abs/2203.16351, doi:10.48550/arXiv.2203.16351.

[CGG+22] Geoffrey S. H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical Foundations of Gradient-Based Learning. In Ilya Sergey, editor, *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 1–28, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-030-99336-8_1.

[CGHR22] Matteo Capucci, Bruno Gavranović, Jules Hedges, and Eigil Fjeldgren Rischel. Towards Foundations of Categorical Cybernetics. *Electronic Proceedings in Theoretical Computer Science*, 372:235–248, November 2022. arXiv:2105.06332 [math]. URL: http://arxiv.org/abs/2105.06332, doi:10.4204/EPTCS.372.17.

[CGLF22] Matteo Capucci, Neil Ghani, Jérémy Ledent, and Fredrik Nordvall Forsberg. Translating Extensive Form Games to Open Games with Agency. *Electronic Proceedings in Theoretical Computer Science*, 372:221–234, November 2022. arXiv:2105.06763 [cs, math]. URL: http://arxiv.org/abs/2105.06763, doi:10.4204/EPTCS.372.16.

[CGLP22] Geoff Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, and Dorette Pronk. Monoidal reverse differential categories. *Mathematical Structures in Computer Science*, 32(10):1313–1363, 2022. Publisher: Cambridge University Press. URL: https://arxiv.org/abs/2203.12478, doi:10.1017/S096012952200038X.

[CK17] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, Cambridge, 2017. URL: https://www.cambridge.org/core/books/picturing-quantum-processes/1119568B3101F3A685BE832FEEC53E52, doi:10.1017/9781316219317.

[CL23] Geoffrey Cruttwell and Jean-Simon Pacaud Lemay. Reverse tangent categories, August 2023. arXiv:2308.01131 [math]. URL: http://arxiv.org/abs/2308.01131, doi:10.48550/arXiv.2308.01131.

[Cla20] Bryce Clarke. Internal lenses as functors and cofunctors. *Electronic Proceedings in Theoretical Computer Science*, 323:183–195, September 2020. arXiv:2009.06835 [math]. URL: http://arxiv.org/abs/2009.06835, doi:10.4204/EPTCS.323.13.

[Cla21] Bryce Clarke. Delta lenses as coalgebras for a comonad. In *STAF 2021 Workshop Proceedings*, volume 2999, pages 18–27. arXiv, 2021. arXiv:2108.00390 [math]. URL: http://arxiv.org/abs/2108.00390.

[CRB21] David Chiang, Alexander M. Rush, and B. Barak. Named Tensor Notation. *Trans. Mach. Learn. Res.*, February 2021. URL: https:

//www.semanticscholar.org/paper/Named-Tensor-Notation-Chiang-Rush/260d92c87296078ab4f822a5a95e309863d4b069.

[CRBD18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/hash/69386f6bb1dfed68692a24c8686939b9-Abstract.html.

[CS11] Robin Cockett and R.A.G. Seely. The Faà di Bruno construction. *Theory and Applications of Categories [electronic only]*, 25, January 2011.

[CSC10] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical Foundations for a Compositional Distributional Model of Meaning, March 2010. arXiv:1003.4394 [cs, math]. URL: http://arxiv.org/abs/1003.4394, doi:10.48550/arXiv.1003.4394.

[CUV06] Venanzio Capretta, Tarmo Uustalu, and Varmo Vene. Recursive coalgebras from comonads. *Information and Computation*, 204(4):437–468, April 2006. URL: https://www.sciencedirect.com/science/article/pii/S0890540105001963, doi:10.1016/j.ic.2005.08.005.

[CW16] Taco Cohen and Max Welling. Group Equivariant Convolutional Networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2990–2999. PMLR, June 2016. ISSN: 1938-7228. URL: https://proceedings.mlr.press/v48/cohenc16.html.

[CXZG16] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training Deep Nets with Sublinear Memory Cost, April 2016. arXiv:1604.06174 [cs]. URL: http://arxiv.org/abs/1604.06174, doi:10.48550/arXiv.1604.06174.

[Dal19] David Dalrymple. (Presentation) Dioptics: a common generalization of gradient-based learners and open games, May 2019. URL: https://www.cl.cam.ac.uk/events/syco/strings3-syco5/slides/dalrymple.pdf.

[DB16] Alexey Dosovitskiy and Thomas Brox. Inverting Visual Representations with Convolutional Networks. In *2016 IEEE Conference on Computer Vision and Pat-*

tern Recognition (CVPR), pages 4829–4837, June 2016. ISSN: 1063-6919. URL: https://ieeexplore.ieee.org/document/7780891, doi:10.1109/CVPR.2016.522.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[Del19] Jean-Charles Delvenne. Category Theory for Autonomous and Networked Dynamical Systems. *Entropy*, 21(3):302, March 2019. URL: https://www.mdpi.com/1099-4300/21/3/302, doi:10.3390/e21030302.

[DH06] Benjamin Dauvergne and Laurent Hascoët. The Data-Flow Equations of Checkpointing in Reverse Automatic Differentiation. In Vassil N. Alexandrov, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors, *Computational Science – ICCS 2006*, Lecture Notes in Computer Science, pages 566–573, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11758549_78.

[dHCW20] Pim de Haan, Taco S Cohen, and Max Welling. Natural Graph Networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 3636–3646. Curran Associates, Inc., 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/2517756c5a9be6ac007fe9bb7fb92611-Abstract.html.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*, 12(null):2121–2159, July 2011.

[Dis20] Zinovy Diskin. General Supervised Learning as Change Propagation with Delta Lenses. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 177–197, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-45231-5_10.

[DKL19] Zinovy Diskin, Harald König, and Mark Lawford. Multiple model synchronization with multiary delta lenses with amendment andK-Putput. *Formal Aspects of Computing*, 31(5):611–640, November 2019. doi:10.1007/s00165-019-00493-0.

[DL19] David Dalrymple and Eliana Lorch. Dioptics: a Common Generalization of Open Games and Gradient-Based Learners, 2019. URL: https://research.protocol.ai/publications/dioptics-a-common-generalization-of-open-games-and-gradient-based-learners/dalrymple2019.pdf.

[Doz16] Timothy Dozat. Incorporating Nesterov Momentum into ADAM. In *Workshop track*, 2016. URL: https://openreview.net/pdf/OM0jvwB8jIp57ZJjtNEZ.pdf.

[DP89] V. C. V. De Paiva. The Dialectica categories. In John W. Gray and Andre Scedrov, editors, *Contemporary Mathematics*, volume 92, pages 47–62. American Mathematical Society, Providence, Rhode Island, 1989. URL: http://www.ams.org/conm/092/, doi:10.1090/conm/092/1003194.

[DS97] Brian Day and Ross Street. Monoidal Bicategories and Hopf Algebroids. *Advances in Mathematics*, 129(1):99–157, July 1997. URL: https://www.sciencedirect.com/science/article/pii/S0001870897916492, doi:10.1006/aima.1997.1649.

[DV22] Andrew J Dudzik and Petar Veličković. Graph Neural Networks are Dynamic Programmers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20635–20647. Curran Associates, Inc., 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/8248b1ded388fcdbbd121bcdfea3068c-Paper-Conference.pdf.

[DXX18] Linhao Dong, Shuang Xu, and Bo Xu. Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5888, April 2018. ISSN: 2379-190X. doi:10.1109/ICASSP.2018.8462506.

[Ell18a] Conal Elliott. The simple essence of automatic differentiation. *Proceedings of the ACM on Programming Languages*, 2(ICFP):70:1–70:29, July 2018. URL: https://dl.acm.org/doi/10.1145/3236765, doi:10.1145/3236765.

[Ell18b] Conal Elliott. Video recording of "The Simple Essence of Automatic Differentiation", July 2018. URL: https://youtu.be/ne99laPUxN4?t=3066.

[Eve19] Tom Everitt. *Towards safe artificial general intelligence*. PhD Thesis, The Australian National University (Australia), 2019. URL: https://search.proquest.com/openview/dab94a88b22c1240a8c791ca0178099c/1?pq-origsite=gscholar&cbl=2026366&diss=y.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1126–1135. PMLR, July 2017. ISSN: 2640-3498. URL: https://proceedings.mlr.press/v70/finn17a.html.

[FC19] Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=rJl-b3RcF7.

[FDRC20] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear Mode Connectivity and the Lottery Ticket Hypothesis. In *Proceedings of the 37th International Conference on Machine Learning*, pages 3259–3269. PMLR, November 2020. ISSN: 2640-3498. URL: https://proceedings.mlr.press/v119/frankle20a.html.

[Fer12] D. A. Ferrucci. Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3.4):1:1–1:15, May 2012. Conference Name: IBM Journal of Research and Development. URL: https://ieeexplore.ieee.org/document/6177724, doi:10.1147/JRD.2012.2184356.

[FJ19] Brendan Fong and Michael Johnson. Lenses and Learners. In James Cheney and Hsiang-Shang Ko, editors, *Proceedings of the 8th International Workshop on Bidirectional Transformations co-located with the Philadelphia Logic Week, Bx@PLW 2019,*

Philadelphia, PA, USA, June 4, 2019, volume 2355 of *CEUR Workshop Proceedings*, pages 16–29. CEUR-WS.org, 2019. URL: https://ceur-ws.org/Vol-2355/paper2.pdf.

[Fox76] Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, January 1976. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00927877608822127. doi:10.1080/00927877608822127.

[Fri20] Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, August 2020. arXiv:1908.07021 [cs, math, stat]. URL: http://arxiv.org/abs/1908.07021, doi:10.1016/j.aim.2020.107239.

[FS19] Brendan Fong and David I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, Cambridge, 2019. URL: https://www.cambridge.org/core/books/an-invitation-to-applied-category-theory/D4C5E5C2B019B2F9B8CE9A4E9E84D6BC, doi:10.1017/9781108668804.

[FST21] Brendan Fong, David 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*, LICS '19, pages 1–13, Vancouver, Canada, June 2021. IEEE Press.

[Fuj19] Soichiro Fujii. A 2-Categorical Study of Graded and Indexed Monads, April 2019. arXiv:1904.08083 [cs, math]. URL: http://arxiv.org/abs/1904.08083.

[Fuk75] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3):121–136, September 1975. doi:10.1007/BF00342633.

[Gar18] Richard Garner. An embedding theorem for tangent categories. *Advances in Mathematics*, 323:668–687, January 2018. URL: https://www.sciencedirect.com/science/article/pii/S0001870817303122, doi:10.1016/j.aim.2017.10.039.

[Gav19] Bruno Gavranović. Compositional Deep Learning, July 2019. arXiv:1907.08292 [cs, math]. URL: http://arxiv.org/abs/1907.08292.

[Gav20a] Bruno Gavranović. Category Theory in Machine Learning, July 2020. original-date: 2020-07-09T14:16:58Z. URL: https://github.com/bgavran/Category_Theory_Machine_Learning.

[Gav20b] Bruno Gavranović. Learning Functors using Gradient Descent. *Electronic Proceedings in Theoretical Computer Science*, 323:230–245, September 2020. arXiv:2009.06837 [cs, math]. URL: http://arxiv.org/abs/2009.06837, doi:10.4204/EPTCS.323.15.

[Gav21] Bruno Gavranović. Meta-learning and Monads, October 2021. URL: https://www.brunogavranovic.com/posts/2021-10-13-meta-learning-and-monads.html.

[Gav22a] Bruno Gavranović. Lenses to the left of me, Prisms to the right, January 2022. URL: https://www.brunogavranovic.com/posts/2022-01-05-lenses-to-the-left-of-me.html.

[Gav22b] Bruno Gavranović. Space-time tradeoffs of lenses and optics via higher category theory, September 2022. arXiv:2209.09351 [cs, math]. URL: http://arxiv.org/abs/2209.09351, doi:10.48550/arXiv.2209.09351.

[Gav23a] Bruno Gavranović. Category Theory Resources, May 2023. original-date: 2021-11-19T10:25:28Z. URL: https://github.com/bgavran/Category_Theory_Resources.

[Gav23b] Bruno Gavranović. Theory and Applications of Lenses and Optics, May 2023. original-date: 2022-04-12T13:53:16Z. URL: https://github.com/bgavran/Lens_Resources.

[Gav23c] Bruno Gavranović. Two kinds of Prisms, February 2023. URL: https://www.brunogavranovic.com/posts/2023-02-12-two-kinds-of-prisms.html.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press, 2016. URL: http://www.deeplearningbook.org.

[GDM+14] Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep AutoRegressive Networks. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1242–1250. PMLR, June 2014. ISSN: 1938-7228. URL: https://proceedings.mlr.press/v32/gregor14.html.

[GFGW23] Nate Gruver, Marc Anton Finzi, Micah Goldblum, and Andrew Gordon Wilson. The Lie Derivative for Measuring Learned Equivariance. In *The Eleventh International*

Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023. URL: https://openreview.net/pdf?id=JL7Va5Vy15J.

[GHWZ18] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional Game Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 472–481, New York, NY, USA, July 2018. Association for Computing Machinery. doi:10.1145/3209108.3209165.

[GJL17] Dan R. Ghica, Achim Jung, and Aliaume Lopez. Diagrammatic Semantics for Digital Circuits. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CSL.2017.24.

[GK96] C. Goller and A. Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352 vol.1, June 1996. doi:10.1109/ICNN.1996.548916.

[GK13] Nicola Gambino and Joachim Kock. Polynomial functors and polynomial monads. *Mathematical Proceedings of the Cambridge Philosophical Society*, 154(1):153–192, January 2013. arXiv:0906.4931 [math]. URL: http://arxiv.org/abs/0906.4931, doi:10.1017/S0305004112000394.

[GLP21] Fabrizio Genovese, Fosco Loregian, and Daniele Palombi. Escrows are optics, May 2021. arXiv:2105.10028 [math]. URL: http://arxiv.org/abs/2105.10028, doi:10.48550/arXiv.2105.10028.

[Gow20] William John Gowers. *The Crossroads of Categorical Algebra and Game Semantics*. PhD thesis, University of Bath, 2020. URL: https://purehost.bath.ac.uk/ws/portalfiles/portal/205893351/complete_thesis.pdf.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 27. Curran Asso-

ciates, Inc., 2014. URL: https://papers.nips.cc/paper_files/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html.

[Gra19] Marco Grandis. *Higher Dimensional Categories: From Double To Multiple Categories*. World Scientific, September 2019. Google-Books-ID: fu60DwAAQBAJ.

[GS20] Fabrizio Genovese and David I. Spivak. A Categorical Semantics for Guarded Petri Nets. In Fabio Gadducci and Timo Kehrer, editors, *Graph Transformation*, Lecture Notes in Computer Science, pages 57–74, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-51372-6_4.

[GV22] Bruno Gavranović and Mattia Villani. Graph Convolutional Neural Networks as Parametric CoKleisli morphisms, December 2022. arXiv:2212.00542 [cs, math]. URL: http://arxiv.org/abs/2212.00542.

[GW00] Andreas Griewank and Andrea Walther. Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Transactions on Mathematical Software*, 26(1):19–45, March 2000. URL: https://dl.acm.org/doi/10.1145/347837.347846, doi:10.1145/347837.347846.

[GWD14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines, December 2014. arXiv:1410.5401 [cs]. URL: http://arxiv.org/abs/1410.5401, doi:10.48550/arXiv.1410.5401.

[GWR+16] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, October 2016. Number: 7626 Publisher: Nature Publishing Group. URL: https://www.nature.com/articles/nature20101, doi:10.1038/nature20101.

[HAMS22] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis & Machine Intelli-*

gence, 44(09):5149–5169, September 2022. Place: Los Alamitos, CA, USA Publisher: IEEE Computer Society. doi:10.1109/TPAMI.2021.3079209.

[Har19] Kenneth D. Harris. Characterizing the invariances of learning algorithms using category theory, May 2019. arXiv:1905.02072 [cs, math, stat]. URL: http://arxiv.org/abs/1905.02072, doi:10.48550/arXiv.1905.02072.

[HCS+16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/hash/d8330f857a17c53d217014ee776bfd50-Abstract.html.

[Hed18] Jules Hedges. Lenses for philosophers, August 2018. URL: https://julesh.com/2018/08/16/lenses-for-philosophers/.

[Hed19] Jules Hedges. Limits of bimorphic lenses, August 2019. arXiv:1808.05545 [cs, math]. URL: http://arxiv.org/abs/1808.05545, doi:10.48550/arXiv.1808.05545.

[HG20] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs), July 2020. arXiv:1606.08415 [cs]. URL: http://arxiv.org/abs/1606.08415, doi:10.48550/arXiv.1606.08415.

[Hin12] Ralf Hinze. Kan Extensions for Program Optimisation Or: Art and Dan Explain an Old Trick. In Jeremy Gibbons and Pablo Nogueira, editors, *Mathematics of Program Construction*, Lecture Notes in Computer Science, pages 324–362, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-31113-0_16.

[Hin23] Avinash Hindupur. The GAN Zoo, June 2023. original-date: 2017-04-14T16:45:24Z. URL: https://github.com/hindupuravinash/the-gan-zoo.

[HK19] Michael Haenlein and Andreas Kaplan. A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. *California Management Review*, 61(4):5–14, August 2019. Publisher: SAGE Publications Inc. doi:10.1177/0008125619864925.

[HKSY17] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A Convenient Category for Higher-Order Probability Theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, June 2017. arXiv:1701.02547 [cs, math]. URL: http://arxiv.org/abs/1701.02547, doi:10.1109/LICS.2017.8005137.

[HL21] Chris Heunen and Jean-Simon Pacaud Lemay. TENSOR-RESTRICTION CATEGORIES. *Theory and Applications of Categories*, 37:635–670, 2021. URL: http://www.tac.mta.ca/tac/volumes/37/21/37-21abs.html.

[HPKH20] Awni Hannun, Vineel Pratap, Jacob Kahn, and Wei-Ning Hsu. Differentiable Weighted Finite-State Transducers, October 2020. arXiv:2010.01003 [cs, stat]. URL: http://arxiv.org/abs/2010.01003, doi:10.48550/arXiv.2010.01003.

[HS97] Sepp Hochreiter and J{\"u}rgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–1780, 1997. URL: https://papers.baulab.info/Hochreiter-1997.pdf, doi:10.1162/neco.1997.9.8.1735.

[HS19] Mathieu Huot and Sam Staton. Universal Properties in Quantum Theory. *Electronic Proceedings in Theoretical Computer Science*, 287:213–223, January 2019. arXiv:1901.10117 [quant-ph]. URL: http://arxiv.org/abs/1901.10117, doi:10.4204/EPTCS.287.12.

[HS23] Jules Hedges and Riu Rodríguez Sakamoto. Value Iteration is Optic Composition. *Electronic Proceedings in Theoretical Computer Science*, 380:417–432, August 2023. arXiv:2206.04547 [math]. URL: http://arxiv.org/abs/2206.04547, doi:10.4204/EPTCS.380.24.

[HSH+23] Tyler Hanks, Baike She, Matthew Hale, Evan Patterson, Matthew Klawonn, and James Fairbanks. A Compositional Framework for Convex Model Predictive Control, May 2023. arXiv:2305.03820 [math]. URL: http://arxiv.org/abs/2305.03820, doi:10.48550/arXiv.2305.03820.

[HSV20] Mathieu Huot, Sam Staton, and Matthijs Vákár. Correctness of Automatic Differentiation via Diffeologies and Categorical Gluing. In Jean Goubault-Larrecq and Barbara

König, editors, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 319–338, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-45231-5_17.

[HT12] C. Hermida and R. D. Tennent. Monoidal indeterminates and categories of possible worlds. *Theoretical Computer Science*, 430:3–22, April 2012. URL: https://www.sciencedirect.com/science/article/pii/S0304397512000163, doi:10.1016/j.tcs.2012.01.001.

[Hut18] Matthew Hutson. AI researchers allege that machine learning is alchemy, May 2018. URL: https://www.science.org/content/article/ai-researchers-allege-machine-learning-alchemy.

[HVHV19] Chris Heunen, Jamie Vicary, Chris Heunen, and Jamie Vicary. *Categories for Quantum Theory: An Introduction*. Oxford Graduate Texts in Mathematics. Oxford University Press, Oxford, New York, November 2019.

[HWG13] Ralf Hinze, Nicolas Wu, and Jeremy Gibbons. Unifying structured recursion schemes. *ACM SIGPLAN Notices*, 48(9):209–220, September 2013. doi:10.1145/2544174.2500578.

[HY21] Reinhard Heckel and Fatih Furkan Yilmaz. Early Stopping in Deep Networks: Double Descent and How to Eliminate it. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=tlV90jvZbw.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. ISSN: 1063-6919. URL: https://ieeexplore.ieee.org/document/7780459, doi:10.1109/CVPR.2016.90.

[IL23] Sacha Ikonicoff and Jean-Simon Pacaud Lemay. Cartesian Differential Comonads and New Models of Cartesian Differential Categories. *Cahiers de topologie et géométrie différentielle catégoriques*, LXIV-2(2):198–239, January 2023. arXiv:2108.04304 [math]. URL: http://arxiv.org/abs/2108.04304, doi:10.48550/arXiv.2108.04304.

[Irp18] Alex Irpan. Deep Reinforcement Learning Doesn't Work Yet, 2018. URL: https://www.alexirpan.com/2018/02/14/rl-hard.html.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456, Lille, France, July 2015. JMLR.org.

[Jac98] Bart Jacobs. *Categorical logic and type theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1998. URL: https://linkinghub.elsevier.com/retrieve/pii/S0049237X98X80286, doi:10.1016/S0049-237X(98)X8028-6.

[JK01] G Janelidze and G M Kelly. A NOTE ON ACTIONS OF A MONOIDAL CATEGORY. *Theory and Applications of Categories*, 9(4), 2001. URL: http://www.tac.mta.ca/tac/volumes/9/n4/9-04abs.html.

[JS86] Andre Joyal and Ross Street. Braided Monoidal Categories. Technical report, Macquarie University, November 1986. URL: http://maths.mq.edu.au/~street/JS1.pdf.

[JS14] Bart Jacobs and Alexandra Silva. Automata Learning: A Categorical Perspective. In Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten, editors, *Horizons of the Mind. A Tribute to Prakash Panangaden: Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science, pages 384–406. Springer International Publishing, Cham, 2014. doi:10.1007/978-3-319-06880-0_20.

[JYJY21] Niles Johnson, Donald Yau, Niles Johnson, and Donald Yau. *2-Dimensional Categories*. Oxford University Press, Oxford, New York, January 2021.

[Kas22] Rohan V. Kashyap. A survey of deep learning optimizers-first and second order methods, November 2022. arXiv:2211.15596 [cs, math]. URL: http://arxiv.org/abs/2211.15596, doi:10.48550/arXiv.2211.15596.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning*

Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015. URL: http://arxiv.org/abs/1412.6980.

[Kel82] Max Kelly. Basic Concepts of Enriched Category Theory | Logic, categories and sets, April 1982. URL: https://www.cambridge.org/us/academic/subjects/mathematics/logic-categories-and-sets/basic-concepts-enriched-category-theory,https://www.cambridge.org/us/academic/subjects/mathematics/logic-categories-and-sets.

[KMH+20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models, January 2020. arXiv:2001.08361 [cs, stat]. URL: http://arxiv.org/abs/2001.08361, doi:10.48550/arXiv.2001.08361.

[Kni23] Will Knight. OpenAI's CEO Says the Age of Giant AI Models Is Already Over. *Wired*, April 2023. Section: tags. URL: https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/.

[Kur99] Ray Kurzweil. *Age of Spiritual Machines: When Computers Exceed Human Intelligence*. Penguin USA, USA, 1st edition, November 1999.

[KV06] Jevgeni Kabanov and Varmo Vene. Recursion Schemes for Dynamic Programming. In Tarmo Uustalu, editor, *Mathematics of Program Construction*, Lecture Notes in Computer Science, pages 235–252, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11783596_15.

[Lac10] Stephen Lack. Icons. *Applied Categorical Structures*, 18(3):289–307, June 2010. doi:10.1007/s10485-008-9136-5.

[Lau06] Martin Laubinger. DIFFEOLOGICAL SPACES. *Proyecciones (Antofagasta)*, 25(2), August 2006. URL: https://www.revistaproyecciones.cl/article/view/1542, doi:10.4067/S0716-09172006000200003.

[Lei11] Tom Leinster. The Magnitude of an Enriched Category, June 2011. URL: https://golem.ph.utexas.edu/category/2011/06/the_magnitude_of_an_enriched_c.html.

[Lei14] Tom Leinster. *Basic Category Theory*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, 2014. URL: https://www.cambridge.org/core/books/basic-category-theory/A72533879BBC7BD956CC415777B7DA99, doi:10.1017/CBO9781107360068.

[Lei21] Tom Leinster. *Entropy and Diversity: The Axiomatic Approach*. Cambridge University Press, April 2021. URL: https://www.cambridge.org/core/books/entropy-and-diversity/496CF94AEA7B33F15904BD4FC8CC2369, doi:10.1017/9781108963558.

[Lev13] Paul Blain Levy. *Call-By-Push-Value*. Thesis, University of London, December 2013. Accepted: 2013-12-10T14:42:27Z ISSN: 1470-5559. URL: https://qmro.qmul.ac.uk/xmlui/handle/123456789/4742.

[Lew19] Martha Lewis. Compositionality for Recursive Neural Networks. *FLAP*, 6(4):709–724, 2019. URL: https://collegepublications.co.uk/ifcolog/?00033.

[LKN+22] Ziming Liu, Ouail Kitouni, Niklas S Nolte, Eric Michaud, Max Tegmark, and Mike Williams. Towards Understanding Grokking: An Effective Theory of Representation Learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 34651–34663. Curran Associates, Inc., 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/dfc310e81992d2e4cedc09ac47eff13e-Paper-Conference.pdf.

[Llo18] Kirsten Lloyd. Bias Amplification in Artificial Intelligence Systems, September 2018. arXiv:1809.07842 [cs]. URL: http://arxiv.org/abs/1809.07842, doi:10.48550/arXiv.1809.07842.

[Lor21] Fosco Loregian. *(Co)end Calculus*. London Mathematical Society Lecture Note Series. Cambridge University Press, Cambridge, 2021. URL: https://www.cambridge.org/core/books/coend-calculus/C662E90767358B336F17B606D19D8C43, doi:10.1017/9781108778657.

[Lyn23] Owen Lynch. Category Theory is like Java - General, April 2023. Section: General. URL: https://www.localcharts.org/t/category-theory-is-like-java/1313.

[Maa13] Andrew L. Maas. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, 2013. URL: https://www.semanticscholar.org/paper/Rectifier-Nonlinearities-Improve-Neural-Network-Maas/367f2c63a6f6a10b3b64b8729d601e69337ee3cc.

[McK06] James McKinna. Why dependent types matter. *ACM SIGPLAN Notices*, 41(1):1, January 2006. URL: https://www.cs.nott.ac.uk/~psztxa/publ/ydtm.pdf, doi:10.1145/1111320.1111038.

[MDG13] Samuel Mimram and Cinzia Di Giusto. A Categorical Theory of Patches. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 298:283–307, November 2013. doi:10.1016/j.entcs.2013.09.018.

[Mel22a] Paul-André Mellies. Parametric monads and enriched adjunctions, June 2022.

[Mel22b] Luckeciano C. Melo. Transformers are Meta-Reinforcement Learners. In *Proceedings of the 39th International Conference on Machine Learning*, pages 15340–15359. PMLR, June 2022. ISSN: 2640-3498. URL: https://proceedings.mlr.press/v162/melo22a.html.

[MGY+21] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, June 2021. Number: 7862 Publisher: Nature Publishing Group. URL: https://www.nature.com/articles/s41586-021-03544-w, doi:10.1038/s41586-021-03544-w.

[Mil19] Bartosz Milewski. *Category Theory for Programmers*. Blurb, 2019. URL: https://github.com/hmemcpy/milewski-ctfp-pdf.

[Mil21a] Bartosz Milewski. Dependent Optics, September 2021. URL: https://bartoszmilewski.com/2021/09/04/dependent-optics/.

[Mil21b] Bartosz Milewski. PolyLens, December 2021. URL: https://bartoszmilewski.com/2021/12/07/polylens/.

[Mil22] Bartosz Milewski. Compound Optics, March 2022. arXiv:2203.12022 [math]. URL: http://arxiv.org/abs/2203.12022, doi:10.48550/arXiv.2203.12022.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. Number: 7540 Publisher: Nature Publishing Group. URL: https://www.nature.com/articles/nature14236, doi:10.1038/nature14236.

[MM22] Gary Marcus and Raphael Milliere. The Challenge of Compositionality for AI, June 2022. URL: https://compositionalintelligence.github.io/.

[MOT15] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going Deeper into Neural Networks, June 2015. URL: https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

[MP23] Sean Moss and Paolo Perrone. A category-theoretic proof of the ergodic decomposition theorem. *Ergodic Theory and Dynamical Systems*, pages 1–27, February 2023. arXiv:2207.07353 [cs, math]. URL: http://arxiv.org/abs/2207.07353, doi:10.1017/etds.2023.6.

[MRCA18] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A Simple Neural Attentive Meta-Learner. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=B1DmUzWAW.

[MSV23] Lyne Moser, Maru Sarazola, and Paula Verdugo. Internal Grothendieck construction for enriched categories, August 2023. arXiv:2308.14455 [math]. URL: http://arxiv.org/abs/2308.14455, doi:10.48550/arXiv.2308.14455.

[MSW22] Joshua Meyers, David I. Spivak, and Ryan Wisnesky. Fast Left Kan Extensions Using the Chase. *Journal of Automated Reasoning*, 66(4):805–844, November 2022. doi:10.1007/s10817-022-09634-2.

[MU22] Dylan McDermott and Tarmo Uustalu. Flexibly Graded Monads and Graded Algebras. In Ekaterina Komendantskaya, editor, *Mathematics of Program Construction*, Lecture Notes in Computer Science, pages 102–128, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-16912-0_4.

[MV15] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196, June 2015. Conference Name: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) ISBN: 9781467369640 Place: Boston, MA, USA Publisher: IEEE. URL: http://ieeexplore.ieee.org/document/7299155/, doi:10.1109/CVPR.2015.7299155.

[Mye22a] David Jaz Myers. *Categorical Systems Theory*. unpublished book draft, February 2022. URL: http://davidjaz.com/Papers/DynamicalBook.pdf.

[Mye22b] David Jaz Myers. The Para Construction as a Distributive Law, December 2022. URL: https://www.youtube.com/watch?v=zB_ifewP8Yk.

[Nes93] Yurii Evgenievich Nesterov. A method of solving a convex programming problem with convergence rate O^(1/k^2). *Dokl. Akad. Nauk SSSR*, 269(3):543–547, 1993. URL: https://mathscinet.ams.org/mathscinet-getitem?mr=0701288.

[New17] Max New. Closure Conversion as CoYoneda, 2017. URL: https://prl.khoury.northeastern.edu/blog/2017/08/28/closure-conversion-as-coyoneda/.

[NKB+21] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: where bigger models and more data hurt*. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, December 2021. Publisher: IOP Publishing and SISSA. URL: https://dx.doi.org/10.1088/1742-5468/ac3a74, doi:10.1088/1742-5468/ac3a74.

[nLa23] nLab authors. quasi-limit, June 2023. URL: https://ncatlab.org/nlab/show/quasi-limit.

[NS23] Nelson Niu and David I. Spivak. *Polynomial Functors: A Mathematical Theory of Interaction*. arXiv, December 2023. arXiv:2312.00990 [math]. URL: http://arxiv.org/abs/2312.00990.

[NW22] Minh Nguyen and Nicolas Wu. Folding over Neural Networks. In Ekaterina Komendantskaya, editor, *Mathematics of Program Construction*, Lecture Notes in Computer Science, pages 129–150, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-16912-0_5.

[NYC15] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, June 2015. ISSN: 1063-6919. URL: https://ieeexplore.ieee.org/document/7298640, doi:10.1109/CVPR.2015.7298640.

[OC17] Chris Olah and Shan Carter. Research Debt. *Distill*, 2017. doi:10.23915/distill.00005.

[Ola15] Christopher Olah. Neural Networks, Types, and Functional Programming – colah's blog, September 2015. URL: https://colah.github.io/posts/2015-09-NN-Types-FP/.

[Ope22] OpenAI. Introducing ChatGPT, November 2022. URL: https://openai.com/blog/chatgpt.

[OWEI20] Dominic Orchard, Philip Wadler, and Harley Eades III. Unifying graded and parameterised monads. *Electronic Proceedings in Theoretical Computer Science*, 317:18–38, May 2020. arXiv:2001.10274 [cs, math]. URL: http://arxiv.org/abs/2001.10274, doi:10.4204/EPTCS.317.2.

[Pav14] Dusko Pavlovic. Chasing Diagrams in Cryptography. In Claudia Casadio, Bob Coecke, Michael Moortgat, and Philip Scott, editors, *Categories and Types in Logic, Language, and Physics: Essays Dedicated to Jim Lambek on the Occasion of His 90th Birthday*, Lecture Notes in Computer Science, pages 353–367. Springer, Berlin, Heidelberg, 2014. doi:10.1007/978-3-642-54789-8_19.

[Per22] Paolo Perrone. Markov Categories and Entropy, December 2022. arXiv:2212.11719 [cs, math, stat]. URL: http://arxiv.org/abs/2212.11719, doi:10.48550/arXiv.2212.11719.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[PGW17] Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor Optics: Modular Data Accessors. *The Art, Science, and Engineering of Programming*, 1(2):7, April 2017. arXiv:1703.10857 [cs]. URL: http://arxiv.org/abs/1703.10857, doi:10.22152/programming-journal.org/2017/1/7.

[PH22] Mary Phuong and Marcus Hutter. Formal Algorithms for Transformers, July 2022. arXiv:2207.09238 [cs]. URL: http://arxiv.org/abs/2207.09238, doi:10.48550/arXiv.2207.09238.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318, Atlanta, GA, USA, June 2013. JMLR.org.

[Pol64] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, January 1964. URL: https://www.sciencedirect.com/science/article/pii/0041555364901375, doi:10.1016/0041-5553(64)90137-5.

[Pre18] Preserve Knowledge. NIPS 2017 Test of Time Award "Machine learning has become alchemy." | Ali Rahimi, Google, March 2018. URL: https://www.youtube.com/watch?v=x7psGHgatGM.

[PRS22] João Paixão, Lucas Rufino, and Paweł Sobociński. High-level axioms for graphical linear algebra. *Science of Computer Programming*, 218:102791, June 2022. URL: https://www.sciencedirect.com/science/article/pii/S0167642322000247, doi:10.1016/j.scico.2022.102791.

[PS08] Barak A. Pearlmutter and Jeffrey Mark Siskind. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems*, 30(2):7:1–7:36, March 2008. URL: https://dl.acm.org/doi/10.1145/1330017.1330018, doi:10.1145/1330017.1330018.

[PSHM23] Mathilde Papillon, Sophia Sanborn, Mustafa Hajij, and Nina Miolane. Architectures of Topological Deep Learning: A Survey on Topological Neural Networks, April 2023. arXiv:2304.10031 [cs]. URL: http://arxiv.org/abs/2304.10031, doi:10.48550/arXiv.2304.10031.

[PVM+22] Wei Peng, Tuomas Varanka, Abdelrahman Mostafa, Henglin Shi, and Guoying Zhao. Hyperbolic Deep Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):10023–10044, December 2022. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. URL: https://ieeexplore.ieee.org/document/9658224, doi:10.1109/TPAMI.2021.3136921.

[PYY+19] Zhaoqing Pan, Weijie Yu, Xiaokai Yi, Asifullah Khan, Feng Yuan, and Yuhui Zheng. Recent Progress on Generative Adversarial Networks (GANs): A Survey. *IEEE Access*, 7:36322–36333, 2019. Conference Name: IEEE Access. doi:10.1109/ACCESS.2019.2905015.

[QGL+20] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary Neural Networks: A Survey. *Pattern Recognition*, 105:107281, September 2020. arXiv:2004.03333 [cs]. URL: http://arxiv.org/abs/2004.03333, doi:10.1016/j.patcog.2020.107281.

[RBL+22] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685, Los Alamitos, CA, USA, June 2022. IEEE Computer Society. URL: https://

doi.ieeecomputersociety.org/10.1109/CVPR52688.2022.01042, doi:10.1109/CVPR52688.2022.01042.

[RDN+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents, April 2022. arXiv:2204.06125 [cs]. URL: http://arxiv.org/abs/2204.06125, doi:10.48550/arXiv.2204.06125.

[Rie20] Bastion Rieck. Machine Learning Needs a Langlands Programme, January 2020. URL: https://bastian.rieck.me/blog/posts/2020/langlands/.

[Ril18] Mitchell Riley. Categories of Optics. *arXiv:1809.00738 [math]*, September 2018. arXiv: 1809.00738. URL: http://arxiv.org/abs/1809.00738.

[Rut00] J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, October 2000. URL: https://www.sciencedirect.com/science/article/pii/S0304397500000566, doi:10.1016/S0304-3975(00)00056-6.

[SA19] Florian Schaefer and Anima Anandkumar. Competitive Gradient Descent. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/hash/56c51a39a7c77d8084838cc920585bd0-Abstract.html.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, October 2018.

[Sch19] Robin M. Schmidt. Recurrent Neural Networks (RNNs): A gentle Introduction and Overview, November 2019. arXiv:1912.05911 [cs, stat]. URL: http://arxiv.org/abs/1912.05911, doi:10.48550/arXiv.1912.05911.

[Sel11] P. Selinger. A Survey of Graphical Languages for Monoidal Categories. In Bob Coecke, editor, *New Structures for Physics*, Lecture Notes in Physics, pages 289–355. Springer, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-12821-9_4.

[SGW21] Dan Shiebler, Bruno Gavranović, and Paul Wilson. Category Theory in Machine Learning, June 2021. arXiv:2106.07032 [cs]. URL: http://arxiv.org/abs/2106.07032, doi:10.48550/arXiv.2106.07032.

[Shi22] Dan Shiebler. Kan Extensions in Data Science and Machine Learning, July 2022. arXiv:2203.09018 [cs, stat]. URL: http://arxiv.org/abs/2203.09018, doi:10.48550/arXiv.2203.09018.

[Shi23] D. Shiebler. *Compositionality and functorial invariants in machine learning*. http://purl.org/dc/dcmitype/Text, University of Oxford, 2023. URL: https://ora.ox.ac.uk/objects/uuid:ec72e338-d95e-4bd6-9412-7ac76b7ddc15.

[SHS+18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, December 2018. Publisher: American Association for the Advancement of Science. URL: https://www.science.org/doi/10.1126/science.aar6404, doi:10.1126/science.aar6404.

[SK16] Tim Salimans and Durk P Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/hash/ed265bc903a5a097f61d3ec064d96d2e-Abstract.html.

[SK21] David Sprunger and Shin-ya Katsumata. Differentiable causal computations via delayed trace. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '19, pages 1–12, Vancouver, Canada, June 2021. IEEE Press.

[SKvOG23] Minkyu Shin, Jin Kim, Bas van Opheusden, and Thomas L. Griffiths. Superhuman Artificial Intelligence Can Improve Human Decision Making by Increasing Novelty. *Proceedings of the National Academy of Sciences*, 120(12):e2214840120, March 2023. arXiv:2303.07462 [cs, econ, q-fin]. URL: http://arxiv.org/abs/2303.07462, doi:10.1073/pnas.2214840120.

[Smi22] Toby St Clere Smithe. Mathematical Foundations for a Compositional Account of the Bayesian Brain, December 2022. arXiv:2212.12538 [cs, math, q-bio, stat]. URL: http://arxiv.org/abs/2212.12538.

[Spi20] David I. Spivak. Poly: An abundant categorical setting for mode-dependent dynamics, June 2020. arXiv:2005.01894 [math]. URL: http://arxiv.org/abs/2005.01894, doi:10.48550/arXiv.2005.01894.

[Spi22a] David I. Spivak. Generalized Lens Categories via functors $\mathcal{C}^{\rm op}\to\mathsf{Cat}$, March 2022. arXiv:1908.02202 [cs, math]. URL: http://arxiv.org/abs/1908.02202, doi:10.48550/arXiv.1908.02202.

[Spi22b] David I. Spivak. Learners' Languages. *Electronic Proceedings in Theoretical Computer Science*, 372:14–28, November 2022. arXiv:2103.01189 [cs, math]. URL: http://arxiv.org/abs/2103.01189, doi:10.4204/EPTCS.372.2.

[Spi23a] David I. Spivak. Functorial aggregation, April 2023. arXiv:2111.10968 [cs, math]. URL: http://arxiv.org/abs/2111.10968, doi:10.48550/arXiv.2111.10968.

[Spi23b] David I. Spivak. A reference for categorical structures on $\mathbf{Poly}$, December 2023. arXiv:2202.00534 [math]. URL: http://arxiv.org/abs/2202.00534.

[SPW+13] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL: https://aclanthology.org/D13-1170.

[SS23] Hisham Sati and Urs Schreiber. Entanglement of Sections: The pushout of entangled and parameterized quantum information, September 2023. arXiv:2309.07245 [math-ph, physics:quant-ph]. URL: http://arxiv.org/abs/2309.07245, doi:10.48550/arXiv.2309.07245.

[Str12] Ross Street. Monoidal categories in, and linking, geometry and algebra. *Bulletin of the Belgian Mathematical Society - Simon Stevin*, 19, January 2012. doi:10.36045/bbms/1354031551.

[Stu15] Kirk Sturtz. Categorical Probability Theory, March 2015. arXiv:1406.6030 [math]. URL: http://arxiv.org/abs/1406.6030, doi:10.48550/arXiv.1406.6030.

[SVZ14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, 2014. URL: http://arxiv.org/abs/1312.6034.

[TDBM22] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient Transformers: A Survey. *ACM Computing Surveys*, 55(6):109:1–109:28, December 2022. URL: https://dl.acm.org/doi/10.1145/3530811, doi:10.1145/3530811.

[Tea23] CCRL Team. CCRL 40/15 - Index, December 2023. URL: https://computerchess.org.uk/ccrl/4040/index.html.

[THJ+19] Javier Turek, Alex Huth, Shailee Jain, Chris Honey, Tai Linzen, Emma Strubell, Leila Wehbe, Kyunghyun Cho, and Alan Yuille. Context and Compositionality in Biological and Artificial Neural Systems, December 2019. URL: https://context-composition.github.io/context-composition.github.io/.

[TUR50] A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, October 1950. doi:10.1093/mind/LIX.236.433.

[TYdF21] Alexis Toumi, Richie Yeung, and Giovanni de Felice. Diagrammatic Differentiation for Quantum Machine Learning. *Electronic Proceedings in Theoretical Computer Science*, 343:132–144, September 2021. arXiv:2103.07960 [quant-ph]. URL: http://arxiv.org/abs/2103.07960, doi:10.4204/EPTCS.343.7.

[US20] Henning Urbat and Lutz Schröder. Automata Learning: An Algebraic Approach. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 900–914, New York, NY, USA, July 2020. Association for Computing Machinery. doi:10.1145/3373718.3394775.

[VBC+19] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James

Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, November 2019. Number: 7782 Publisher: Nature Publishing Group. URL: https://www.nature.com/articles/s41586-019-1724-z, doi:10.1038/s41586-019-1724-z.

[VC22] Andre Videla and Matteo Capucci. Lenses for Composable Servers, March 2022. arXiv:2203.15633 [cs]. URL: http://arxiv.org/abs/2203.15633, doi:10.48550/arXiv.2203.15633.

[Vel22] Petar Veličković. Message passing all the way up, February 2022. arXiv:2202.11097 [cs, stat]. URL: http://arxiv.org/abs/2202.11097, doi:10.48550/arXiv.2202.11097.

[Vel23] Petar Veličković. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, April 2023. URL: https://www.sciencedirect.com/science/article/pii/S0959440X2300012X, doi:10.1016/j.sbi.2023.102538.

[Ver23] Pietro Vertechi. Dependent Optics. *Electronic Proceedings in Theoretical Computer Science*, 380:128–144, August 2023. arXiv:2204.09547 [cs, math]. URL: http://arxiv.org/abs/2204.09547, doi:10.4204/EPTCS.380.8.

[VG15] Tamara Von Glehn. *Polynomials and models of type theory*. PhD thesis, University of Cambridge, 2015. Publisher: Apollo - University of Cambridge Repository. URL: https://www.repository.cam.ac.uk/handle/1810/254394.

[VOFA24] Apostol Vassilev, Alina Oprea, Alie Fordyce, and Hyrum Anderson. Adversarial Machine Learning: A Taxonomy and Terminology of Attacks and Mitigations. Technical Report NIST Artificial Intelligence (AI) 100-2 E2023, National Institute of Standards and Technology, January 2024. URL: https://csrc.nist.gov/pubs/ai/100/2/e2023/final, doi:10.6028/NIST.AI.100-2e2023.

[VONR+23] Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-

context by gradient descent. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *ICML'23*, pages 35151–35174, Honolulu, Hawaii, USA, July 2023. JMLR.org.

[VS22] Matthijs Vákár and Tom Smeding. CHAD: Combinatory Homomorphic Automatic Differentiation. *ACM Transactions on Programming Languages and Systems*, 44(3):20:1–20:49, August 2022. doi:10.1145/3527634.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[Vá21] Matthijs Vákár. Reverse AD at Higher Types: Pure, Principled and Denotationally Correct. *Programming Languages and Systems*, 12648:607, 2021. Publisher: Nature Publishing Group. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7984537/, doi:10.1007/978-3-030-72019-3_22.

[Win23] Glynn Winskel. Making Concurrency Functional. In *LICS*, pages 1–14. IEEE Computer Society, June 2023. URL: https://www.computer.org/csdl/proceedings-article/lics/2023/10175727/1OM4V7dKgsU, doi:10.1109/LICS56636.2023.10175727.

[WMLC23] Vincent Wang-Mascianica, Jonathon Liu, and Bob Coecke. Distilling Text into Circuits, January 2023. arXiv:2301.10595 [cs, math]. URL: http://arxiv.org/abs/2301.10595, doi:10.48550/arXiv.2301.10595.

[Woo76] R. J. Wood. *Indicial methods for relative categories*. PhD thesis, Dalhouse University, 1976. URL: https://dalspace.library.dal.ca//handle/10222/55465.

[Woo78] R. J. Wood. V-indexed categories. In *Indexed Categories and Their Applications*, volume 661, pages 126–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978. Series Title: Lecture Notes in Mathematics. URL: http://link.springer.com/10.1007/BFb0061362, doi:10.1007/BFb0061362.

[WPC+21] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, January 2021. URL: https://ieeexplore.ieee.org/document/9046288/, doi:10.1109/TNNLS.2020.2978386.

[WZ21] Paul Wilson and Fabio Zanasi. Reverse Derivative Ascent: A Categorical Approach to Learning Boolean Circuits. *Electronic Proceedings in Theoretical Computer Science*, 333:247–260, February 2021. arXiv:2101.10488 [cs]. URL: http://arxiv.org/abs/2101.10488, doi:10.4204/EPTCS.333.17.

[WZ22] Paul Wilson and Fabio Zanasi. Categories of Differentiable Polynomial Circuits for Machine Learning. In Nicolas Behr and Daniel Strüber, editors, *Graph Transformation*, Lecture Notes in Computer Science, pages 77–93, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-09843-7_5.

[XUD+19] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges. In Jie Tang, Min-Yen Kan, Dongyan Zhao, Sujian Li, and Hongying Zan, editors, *Natural Language Processing and Chinese Computing*, Lecture Notes in Computer Science, pages 563–574, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-32236-6_51.

[YSHZ19] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, July 2019. doi:10.1162/neco_a_01199.

[ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2242–2251, October 2017. Conference Name: 2017 IEEE International Conference on Computer Vision (ICCV) ISBN: 9781538610329 Place: Venice Publisher: IEEE. URL: http://ieeexplore.ieee.org/document/8237506/, doi:10.1109/ICCV.2017.244.