Exploring explainable AI: category theory insights into machine learning algorithms

IOP Publishing Mach. Learn.: Sci. Technol. 4 (2023) 045061 https://doi.org/10.1088/2632-2153/ad1534

MACHINE LEARNING
Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
4 May 2023

REVISED
16 November 2023

ACCEPTED FOR PUBLICATION
11 December 2023

PUBLISHED
28 December 2023

PAPER

Exploring explainable AI: category theory insights into machine learning algorithms

Ares Fabregat-Hernández[1,*], **Javier Palanca**[1] **and Vicent Botti**[1,2]

[1] Valencian Research Institute for Artificial Intelligence (VRAIN), Universitat Politécnica de Valéncia, Camí de Vera s/n, 46022 Valencia, Spain
[2] valgrAI (Valencian Graduate School and Research Network of Artificial Intelligence), Valencia, Spain
[*] Author to whom any correspondence should be addressed.

E-mail: arfabher@etsii.upv.es

Keywords: explainability, category theory, Lipschitz functions, Yoneda embedding, compositionality

Abstract

Explainable artificial intelligence (XAI) is a growing field that aims to increase the transparency and interpretability of machine learning (ML) models. The aim of this work is to use the categorical properties of learning algorithms in conjunction with the categorical perspective of the information in the datasets to give a framework for explainability. In order to achieve this, we are going to define the enriched categories, with decorated morphisms, $\mathcal{L}earn$, $\mathcal{P}ara$ and $\mathcal{MN}et$ of learners, parameterized functions, and neural networks over metric spaces respectively. The main idea is to encode information from the dataset via categorical methods, see how it propagates, and lastly, interpret the results thanks again to categorical (metric) information. This means that we can attach numerical (computable) information via enrichment to the structural information of the category. With this, we can translate theoretical information into parameters that are easily understandable. We will make use of different categories of enrichment to keep track of different kinds of information. That is, to see how differences in attributes of the data are modified by the algorithm to result in differences in the output to achieve better separation. In that way, the categorical framework gives us an algorithm to interpret what the learning algorithm is doing. Furthermore, since it is designed with generality in mind, it should be applicable in various different contexts. There are three main properties of category theory that help with the interpretability of ML models: formality, the existence of universal properties, and compositionality. The last property offers a way to combine smaller, simpler models that are easily understood to build larger ones. This is achieved by formally representing the structure of ML algorithms and information contained in the model. Finally, universal properties are a cornerstone of category theory. They help us characterize an object, not by its attributes, but by how it interacts with other objects. Thus, we can formally characterize an algorithm by how it interacts with the data. The main advantage of the framework is that it can unify under the same language different techniques used in XAI. Thus, using the same language and concepts we can describe a myriad of techniques and properties of ML algorithms, streamlining their explanation and making them easier to generalize and extrapolate.

1. Introduction

As hinted in [BTV22], learning algorithms do not invent new properties: they merely learn to recognize properties of the data. That means that considering the data as a dataset, that is, just a **set** of data loses information. In [BTV22], the authors propose to model the data consisting of text with the language of enriched categories. In particular, they construct a category whose objects are strings of text and whose morphisms are inclusions of one string into the other. Such morphism exists if and only if the target string contains the source string. Then, they enrich the category over the interval $[0,1]$ to assign probabilities to that inclusion. That is, if we have a morphism $X \to Y$ then we have the data $p(Y|X)$ which is the probability that the string Y extends the string X.

Of course, this works because we are working with a category whose objects are strings of text and the 'extension' of strings makes sense. For instance, it makes sense to consider the extension from $x =$ 'the king' to the expression $y =$ 'By decree of the king'. This would be encoded by an arrow $x \to y$. However, the idea that one can use category theory to encode the structural information of the data set and then use it to understand how the algorithm uses it to learn these relations should be generalizable to other data structures. That is one part of the equation, the other is the algorithm itself. In [FST19], the authors consider the learning algorithms to be morphisms in a category. These morphisms are equipped with auxiliary functions that, in the right circumstances, implement gradient descent and backward pass. Furthermore, they add the structure of a symmetric monoidal category on top of the basic categorical structure. With that, they model the action of composing in series (via the standard categorical composition) and in parallel (via the monoidal product).

This rather theoretical, approach is used to model categorically both the dataset and the algorithm; in essence, answering the question '**What** are they?'. Again, that is only part of the story: no matter what they are, they perform an action. They classify or approximate things, they give criteria in order to make a decision, etc. There are two questions that immediately spawn from this: '**How** do they do it?' and '**Why** do they reach that result?'. Those are the kinds of questions that the area of explainable artificial intelligence (XAI) is trying to solve.

XAI has become an area of increasing interest among the machine learning (ML) community. This comes as no surprise since ML techniques have become ubiquitous in both research and industry applications. The main problem is that with the so-called black-box models, i.e. models that we do not know, nor understand their inner workings, possible after-effects cannot be predicted. In some cases, this is innocuous but, in others, such as medical applications, the side effects could be catastrophic, hence the need for explainability methods. The issue in this manuscript is mainly to answer the question of '**Why** do they reach that result?' by analyzing the structural information of the dataset and tracking its propagation through the algorithms. Thus, we mainly concern ourselves with the explainability of the models. For a survey on that matter, we refer the reader to [TG20].

This means that the explanations are meant to be for humans either the end-users of the algorithms or regulators. In the case of the end-users, ensure the results make sense by understanding the process of the algorithm and interpreting the results. In the case of regulators, they are responsible for ensuring that artificial intelligence (AI) systems adhere to ethical, legal, and regulatory standards; XAI allows them to assess whether AI models and applications comply with these standards by providing insights into the decision-making processes.

Another pressing issue in ML is the, apparently, inherent bias that the algorithms can contain (see [MMS+21]). This survey lists and explains the different types of biases. There are several distinctions among each type but it can be summed up in two types: bias from the data and external to the data. One key concept of the analysis made on [MMS+21, section 5.2] is the idea of fairness in classification. The idea relies heavily on making a comparison between data points: are they similar or dissimilar? That comparison is modeled by a 'distance' function that encodes the idea of being 'close' (similar) or 'far away' (dissimilar).

In this context, category theory has built in some desirable properties that make it a natural language to explain ML models:

- **Universal properties:** category theory provides a language for describing the universal properties of mathematical objects. Universal properties describe fundamental relationships between objects in categories. A universal property characterizes an object or morphism in terms of its interactions with other objects or morphisms in the category. It provides a concise and abstract way of specifying important structures or properties. Thus, understanding the algorithms in terms of their universal properties would yield a 'greatest common structure' of a collection of objects or morphisms which means that all algorithms of a certain type would have something in common that can be used to abstract and generalize its structure. This can help to formalize the idea of 'learning' as a functor between categories.
- **Compositionality:** category theory emphasizes the importance of compositionality: the ability to combine smaller structures to build larger ones. In ML, this translates to the idea of building complex models from simpler building blocks.

This has two main applications to ML:

- **Comparing models:** category theory can be used to identify the key features of a model that contribute to its behavior and performance and to extract these features in an interpretable way.
- **Formalizing model interpretability:** since there is no formal definition of what the 'interpretability' of a model is, category theory can be used to formalize the concept of interpretability itself, by providing a way to represent and reason about the information content of a model.

Most of what has been proposed to attack the explainability question is based on experimentation and statistical information. We believe that understanding the structure of the elements involved, the dataset and algorithm, and how it propagates information is a step to shed some light on the explainability issue. For that, we propose to unify the approaches of [FST19, BTV22] by means of the enriched version of the Yoneda embedding. This will allow us to keep track of the relationship between points in the dataset (input) and the results (output) of the algorithm. It is important to note that having a categorical framework allows us to have a way to systematically study the properties of the algorithms theoretically, as opposed to computationally, thus providing general rules ready to be applied. This, in turn, results in the bundling of different phenomena as different expressions of the same concept. Thus, this framework, by providing a succinct structural analysis of datasets, algorithms, and their mathematical interactions, facilitates the formulation of theoretical proofs with complete certainty in the explanations offered.

This is an example of the concepts of formality and universal properties in action and is by no means unique to this instance. Furthermore, once we have formalized a model we can incorporate it into existing theories or parts that are well understood to gain insight into more complex problems: this is the compositionality property that is emphasized in category theory. To summarize, the main question we solve in this manuscript is how to combine the categorical frameworks on the category of the algorithms and the category of the data set to track the distortion made by the algorithm. Additionally, we compare the metrics, obtained via enrichment, of the dataset with known data analysis methods. The rest of the paper is organized as follows.

In section 2 we introduce the categories Learn Para and NNet of learning algorithms, parameterized functions, and neural networks respectively. We also introduce some categorical notions that are needed to unify the frameworks mentioned above. In section 3 we define the categories $\mathcal{Learn}$, $\mathcal{Para}$ and $\mathcal{NNet}$ with decorated hom-categories of learners, parameterized functions, and neural networks. We also introduce the idea of a change of enrichment of the data categories. This way we showcase how, with the language of category theory, we can glance at different properties of the dataset. Lastly, in section 4 we analyze three examples of neural networks and datasets to acquire some intuition of the concepts we are dealing with. Furthermore, the analysis of the properties of the dataset will be equivalent to some metrics found in the analysis of the dataset thus demonstrating the ability of the framework to encompass different explainability methods. Thus, this method provides a structural explanation: by analyzing the structure of the data and how the algorithm interacts with it we can explain what the algorithm is seen and what it is emphasizing.

In conclusion regarding the manuscript's structure, the key takeaways are definitions 3.1 and 3.8, which elucidate the significance of enrichments on datasets, and theorem 3.19 in conjunction with definition 3.21, which clarify how to monitor changes in dataset metrics. In section 4, we then apply these concepts to specific datasets. The remainder of the manuscript is dedicated to establishing the necessary formalism for precisely defining the framework that yields these results. Therefore, for readers primarily interested in the framework's application, these are the primary sections to prioritize. An appendix has been added at the end of the manuscript giving the necessary categorical definitions to follow the the main arguments.

Lastly, it is worth noting that there are several problems that can be represented categorically, as demonstrated in works such as [Fon15], where the authors utilize categorical language to model electrical circuits; [HSH+23], which emphasizes the compositional nature of predictive control; and [ABFR23], which highlights the compositional aspects of algorithms and datasets. These works provide extensive examples of applying categorical language to more concrete scenarios.

We assume some familiarity with the basic constructions of category theory and its enriched counterpart. For an introduction to the subject we refer the reader to [Mil19, Rie17, Rie14].

2. Theoretical background

In this section, we review what has been attempted to explain how or what a neural network learns. For that, we analyze the structure of the learning algorithm (see [FST19]) and of the studied dataset (see [BTV22]). The study of the structure of these elements is carried by the same theoretical means, yet these are two separate approaches.

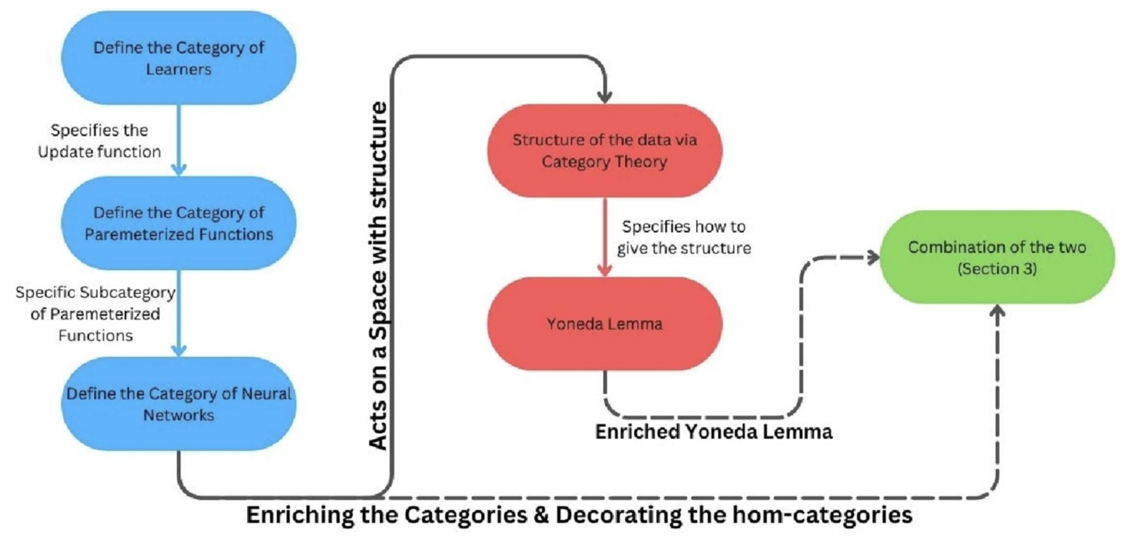

Define the Category of Learners
Specifies the Update function
Define the Category of Paremeterized Functions
Specific Subcategory of Paremeterized Functions
Define the Category of Neural Networks
Acts on a Space with structure
Structure of the data via Category Theory
Specifies how to give the structure
Yoneda Lemma
Enriched Yoneda Lemma
Combination of the two (Section 3)
Enriching the Categories & Decorating the hom-categories

Figure 1. Flowchart of section 2.

Figure 1 shows a flowchart of this section. There are two separate steps: the formalization of the algorithms (in blue) and the formalization of the datasets (in red). The bridge between those two steps is the action of the algorithms on the datasets: the algorithms take input points from the datasets. Then, both sides will be combined (via enrichment and decoration) in section 3. Through this chart we can see how this method works for XAI: we can interpret the relations (similarity, closeness, etc ...) of the data and the output of the algorithm thanks to the enriched version of the Yoneda embedding. This would correspond to data analysis of the input/output structures. Additionally, knowing that learners, viewed as parameterized functions, perform parameter updates given the relations between the input/output structures we can understand the decision-making that is been done by the algorithms. In particular, if we restrict ourselves to the case of neural networks we can understand how the structures of the data modify the structure of the algorithm. In other words, we can understand how it is learning.

In order to achieve that goal, we first review some of the results of [FST19] whose main punchline of [FST19] is that **learning is compositional**. In order to precisely define what is meant by that, the authors use the language of category theory which was built precisely to encode such behavior. A supervised algorithm is essentially a map from a space of inputs A to a space of outputs B that depends on some parameter space P. This supervised algorithm needs some kind of update of 'believes' that is, an update of the parameters, according to some labeled data which consists of pairs (a, b) with a being in the input space and b in the output space. The aim of such an algorithm is to find an approximation to an ideal function $f\colon A \to B$. As seen with the encoder–decoder architecture, it is often useful to chain such learning algorithms, in other words, to **compose them**. The problem is that having labeled pairs (a, c) for the complete encoder–decoder algorithm is different than having a couple of labeled pairs, (a, b) for the encoder and (b', c') for the decoder, and these two pairs should be related somehow. For that, an extra function is needed: a function that requests the parameter b' that would have been more useful for the second part of the algorithm. This is formalized in the following definition.

Definition 2.1 ([FST19, definition II.1]). Let A and B be sets. A **supervised learning algorithm** (or **learner**) $A \to B$ is a tuple (P, I, U, r) where P is a set and

$$\begin{aligned} I&\colon P \times A \to B \\ U&\colon P \times A \times B \to P \\ r&\colon P \times A \times B \to A \end{aligned} \tag{1}$$

are functions. The set P is called the **parameter space**. The functions I and U are called the **implement** and **update** functions respectively. Finally, the function r is called the **request** function.

Remark 2.2. We will use the term **learning algorithm** to signify the classical definition and **learner** to refer to the elements defined in definition 2.1.

Given a parameter $p \in P$, we think of $I(p, -)\colon A \to B$ as the function that implements p. The update function $U\colon P \times A \times B \to P$ is giving us a better parameter p' to approximate our ideal function $f\colon A \to B$.

That is to say that $I(p', -)$ should be more similar to f that $I(p, -)$. There is one notion of learning algorithms that needs to be formalized: when are two learners equivalent? With respect to learning algorithms with the same input and output spaces, we say they are equivalent if, even though their parameters might differ, their outputs do not. Thus, we say that two learners (P, I, U, r) and (P', I', U', r') are **equivalent** if there is a bijection $f\colon P \to P'$ such that

$$\begin{aligned} I'(f(p), a) &= I(p, a), \\ U'(f(p), a, b) &= f(U(p, a, b)), \\ r'(f(p), a, b) &= r(p, a, b). \end{aligned} \tag{2}$$

With all the information needed to define what a learner is, we can see that the class of these learners has a very nice structure.

Proposition 2.3 (**[FST19, proposition II.4]**). *There exists a symmetric monoidal category* Learn *whose objects are sets and whose morphisms are equivalence classes of learners.*

The *category* part of the result implies that one can concatenate learners, that is one can use the output of a learning algorithm as the input of another learner. Notice that the request function is necessary to have a composition (in series) of learners. The *symmetric monoidal* part implies that one can implement the algorithms in parallel.

That being said, when one works with learning algorithms, the update function comes from gradient descent (and backpropagation). For that, we need to define an additional structure: differential structure. Hence, we will consider the case $A = \mathbb{R}^n$ and $B = \mathbb{R}^m$ where we can do calculus. We have the following definition.

Definition 2.4. A **differentiable parametrized function** $A \to B$ is a pair (P, I) where P is an Euclidean space and

$$I\colon P \times A \to B \tag{3}$$

is a differentiable function. We call two such pairs (P, I) and (P', I') **equivalent** if there exists a differetiable bijection $f\colon P \to P'$ such that for all $p \in P$ and $a \in A$ we have $I(p, a) = I'(f(p), a)$.

The class of these differentiable parametrized functions forms a category.

Definition 2.5 (**[FST19, definition III.1]**). We denote by Para the strict symmetric monoidal category whose objects are Euclidean spaces and whose morphisms $\mathbb{R}^n \to \mathbb{R}^m$ are equivalence classes of differentiable parametrized functions $\mathbb{R}^n \to \mathbb{R}^m$. Composition in this category of $(P, I)\colon \mathbb{R}^n \to \mathbb{R}^m$ and $(Q, J)\colon \mathbb{R}^m \to \mathbb{R}^l$ is given by $(P \times Q, I * J)$ with

$$(I * J)(p, q, a) = J(q, I(p, a)). \tag{4}$$

The monoidal product of objects $\mathbb{R}^n, \mathbb{R}^m$ in Para is given by $\mathbb{R}^n \times \mathbb{R}^m = \mathbb{R}^{n+m}$. The monoidal product of morphisms $(P, I)\colon \mathbb{R}^n \to \mathbb{R}^m$ and $(Q, J)\colon \mathbb{R}^m \to \mathbb{R}^l$ is given by $(P \times Q, I \| J)$ with

$$(I \| J)(p, q, a, c) = (I(p, a), J(q, c)). \tag{5}$$

Remark 2.6. (1) Notice that the definition of equivalent differentiable parametrized functions is similar to the definition of equivalent discriminant functions.
(2) Composition in Para reflects the fact that we can compose in 'series' whereas the monoidal structure encodes that we can compose in 'parallel'.

This yields the main theorem of [FST19].

Theorem 2.7 (**[FST19, theorem III.2]**). *Fix a real number $\alpha > 0$ and $e(x, y)\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ a differentiable error function such that*

$$\frac{\partial e}{\partial x}(x_0, -)\colon \mathbb{R} \to \mathbb{R} \tag{6}$$

is invertible for all $x_0 \in \mathbb{R}$. Then we have a faithful, injective on objects, strong symmetric monoidal functor

$$L_{\alpha, e}\colon \text{Para} \to \text{Learn} \tag{7}$$

that sends each parameterized function $I\colon P\times A\to B$ to the learner (P, I, U_I, r_I) given by $U_I(p,a,b) = p-\alpha\nabla_p E_I(p,a,b)$ and $r_I(p,a,b) = f_a(\nabla_a E_I(p,a,b))$ with $E_I(p,a,b) = \sum_j e(I(p,a)_j, b_j)$ and f_a is the component-wise application of the inverse to $\frac{\partial e}{\partial x}(a_i, -)$ for each i.

The real number α is called **learning rate** or **step size**, the function $e(x,y)$ is the **error function** and E_I is the **total error function**.

Remark 2.8. (1) In the theorem 2.7, the fact that the sets A, B and P are vector spaces is not used. What is really relevant is the fact that those sets have differentiable structures, i.e. we can do calculus to compute the gradient. This has led to a generalization of this concept using differentiable manifolds.
(2) The update function U_I implements gradient descent while the request r_I function encodes backpropagation.

The next step is to understand how neural networks fit into this framework.

Definition 2.9. We define a **neural network layer** of type (m,n) as a subset of $[m]\times[n]$ with $[n] = \{1,\dots,n\}$.

The idea is that $C\subseteq[m]\times[n]$ encodes the connectivity of the layer. If $C = [m]\times[n]$ we say the layer is **dense**. With that, we define a ***k*-layer neural network** of type (m,n) as a sequence of neural network layers of types $(n_0,n_1)\cdots(n_{k-1},n_k)$, with $n_0 = m$ and $n_k = n$. A **neural network of type (m,n)** is a k-layer neural network of type (m,n) for some k.

Definition 2.10 ([FST19, definition IV.1]). The category NNet of neural networks has as objects natural numbers and as morphisms neural networks of type (m,n). Composition is given by the concatenation of neural networks.

With all of this defined we can finally integrate neural networks into the categorical framework.

Proposition 2.11 ([FST19, proposition IV.2]). *Given a differentiable function $\tau\colon\mathbb{R}\to\mathbb{R}$ we have a functor*

$$I^\tau\colon \mathrm{NNet}\to\mathrm{Para}. \tag{8}$$

On objects, this functor maps each natural number n to the Euclidean vector space $\mathbb{R}^n$. On morphisms, each one-layer neural network $C\colon m\to n$ is mapped to the parameterized function

$$I^\tau_C\colon \mathbb{R}^{|C|+n}\times\mathbb{R}^m\to\mathbb{R}^n \tag{9}$$

given by

$$\left(\left(w_{ji}, w_j\right), a\right)_{\substack{1\leqslant i\leqslant m\\ 1\leqslant j\leqslant n}} \mapsto \left(\tau\left(\sum_i w_{ji}\cdot a_i + w_j\right)\right). \tag{10}$$

Given a neural network $N = C_1\cdots C_k$, the image under the functor I^τ is the composition

$$I^\tau_N = I^\tau_{C_1} * \cdots * I^\tau_{C_k}. \tag{11}$$

The function $\tau\colon\mathbb{R}\to\mathbb{R}$ is called the **activation function**, the parameters w_{ji} are called the **weights** and w_j the **biases**.

Remark 2.12. (1) The composition $L_{\alpha,e}\circ I^\sigma$ yields an inclusion of NNet in Learn. Thus we can consider neural networks as learners.
(2) In practice, the activation function, or the error function for that matter, need not be differentiable but differentiable almost everywhere. Still, we can implement gradient descent, and consequently apply this framework, even if we are dealing with non-differentiable functions.

To summarize, we have constructed a framework in which every learning algorithm corresponds to a morphism in a certain category and we have produced a way to understand neural networks within the framework. Now, when we talk about neural networks and understanding what they do to the data, the dataset is represented as elements of a set $A = \mathbb{R}^n$.

The problem is that is not the whole story. The data has a structure in itself, and that structure is not considered here. That being said, the data does pertain to a set with added structure, usually an Euclidean vector space. As exemplified by [BTV22] if the data is semantical, we can consider the structure of a $[0,1]$-enriched category. That is, each object in the category is a string of words, that is, a sentence. We have a morphism between two sentences $x\to y$ if and only if $x\subseteq y$, that is, x is contained in y. Furthermore, language is not only algebraic (we can consider $y\subseteq z$ and thus $x\subseteq z$), but also statistical. We know there are certain phrases that are more common than others. In the same spirit, the probability of finding a certain word in a sentence depends on the context. Analogously, if we have a certain sentence x, what follows cannot be anything. It can be a lot of things, but not anything, and not every extension of that sentence would have the same probability.

For instance, if x is the word *red*, what follows might be *balloon* or *truck*, but not *dog*. Thus, the probability that $y = $ *red balloon* extends x should be higher than the probability that $z = $ *red dog* extends x. In other words, we can add to the data of a morphism $x \to y$, a number in $[0,1]$ representing $p(y|x)$ that is the probability that the expression y extends the expression x.

This makes a lot of sense, learning algorithms should not invent new properties. They should merely 'learn' existing information, and that information should be expressable somehow. The problem is that this way of expressing it does not fit with the framework developed in [FST19] and it is, *a priori*, hard to translate to the computational implementation. To give a first attempt solution to this we look at [BTV22, section 5] where the authors consider a $[0,+\infty]$-category, what is called a **generalized metric space** (see [Law73]). If we have $p(y|x)$ by taking the function

$$-\ln\colon\ [0,1] \to [0,+\infty] \tag{12}$$

we can pass from conditional probabilities to 'distances' and from distances to probabilities by

$$a \mapsto \exp\left(-a\right). \tag{13}$$

Thus, if we have some kind of structural information about the dataset that can be encoded in a $[0,1]$-enriched category, we can regard it as a peculiar metric space. This is good news since the usual computational implementation relies on some kind of embedding of the data into some Euclidean (metric) vector space $\mathbb{R}^n$. Furthermore, there is a result in category theory that states that two objects are isomorphic (that is, essentially the same) if and only if their hom sets, the sets of relations between any two objects, are isomorphic: the Yoneda embedding.

Lemma 2.13. *Let $\mathcal{C}$ be a category and x, y objects in $\mathcal{C}$. Then x is ismorphic to y if and only if* $\hom(x,-)$ *is ismorphic to* $\hom(y,-)$.

This means that we can substitute the object of the category by setting the morphisms from that object to all the objects. This seems rather convoluted but, by combining it with the enriched structure, and associating each morphism with a number, we can consider the set morphisms for every object as a vector with numerical values in it: the distance from the object to every object in the category.

Remark 2.14. (1) This would mean that one would need the enriched version of the Yoneda embedding, where the isomorphism is taken in the category of enrichment and not merely in the category of sets. This is reminiscent of the fact that any (locally small) category is a category enriched over the category of sets. Since the statement of the enriched version of the embedding is practically identical to lemma 2.13 we refer the reader to [Kel82, section 2.4] for a precise statement and proof.
(2) Notice that in the above paragraph '*are the same*' is in italics. The concept of sameness, which is related to the concept of closeness, is a subtle notion in mathematics (see www.quantamagazine.org/with-category-theory-mathematics-escapes-from-equality-20191010/).
(3) The functor $\hom(x,-)$ is sometimes denoted by $h^x(-)$. We will adopt this notation in the following sections.

To summarize, we have two categorical approaches, one for the learning algorithms and one for the datasets that are related by the categorical language yet studied separately. Furthermore, the approach to studying the structure of the dataset uses an enriched version of the categorical language that the other approach lacks. The goal of this paper is to unify both approaches to enhance the interpretability and explainability of the learners.

Remark 2.15. Attaching some geometry and category theoretical representations to learning algorithms is something being studied from various perspectives at the moment. For examples of this, see [BBCV21, BB21, SGW21].

We need to define what we mean by explainability. When we talk about *explainability* of a model we refer to the ability to explain why the model has reached a certain decision. We refer to *interpretability* to the ability to interpret the results of an algorithm. That is, interpretability, for us, is mainly concerned with the output space and what can we say about its points whereas explainability deals with the model itself and tries to answer the question of 'Why did the algorithm do that?'.

Thus, the compositional framework would help with explainability. Understanding how to decompose an algorithm into smaller, easier-to-explain parts, and knowing how to glue them would (theoretically) improve explainability. On the other hand, understanding the ambient spaces of inputs and outputs allows us to view an algorithm as a map between two mathematical structures. This extra structure allows us to interpret (mathematically) the results more precisely. We will devote the next section to the study of the structures to better understand the behavior of the algorithm.

3. Categorical modeling

Similar to the previous section, this one can be broadly categorized into two main components. The first involves enriching the data categories to derive properties of the data, while the second entails the enrichment (and decoration[3]) of the algorithm categories. By following this approach, we can introduce metrics that capture information about the datasets and subsequently imbue the algorithms with structure to analyze their impact on these metrics. We start by talking about what metrics on the data set carry important information.

3.1. Enriched datasets

The Yoneda embedding[4] gives us a new perspective of the features of the dataset: one should be able to determine a data point by its relation to all other data points. Since we are dealing with numerical values, the distance between data points can be computed and it can be thought of as the relationship between those two data points. If the distance is small, the data points are similar, if the distance is big, they are different.

Thus, we need to enrich our data, that is our input space, over $\mathbb{R}_+$ to turn it into a generalized metric space in order to measure distances between objects. Furthermore, we can do the same for the output: turn it into some sort of metric or generalized metric space in order to measure how the distances have been modified from input to output. This will depend on the task at hand since two outputs being similar might not mean that they are close in Euclidean distance.

Hence, in order to build a robust framework, we need to be able to accommodate as many ML tasks as possible. One such task is classification. The difficulty here is the metric in the classifying space. Indeed, in classification tasks, the output of the algorithms is not interpreted the same way as in approximation tasks. Thus, if we want to define an appropriate metric for the output space, the first thing to do is understand how a classification algorithm works. More precisely, how does it compute the error or loss?

Given a pair (a,b) where a is the output of the algorithm and b is a label (in this case a class) the CrossEntropyLoss is defined as

$$e(a,b) = l(a,b) = -w_b \log\left(\frac{\exp(a_b)}{\sum_{c=1}^{C}\exp(a_c)}\right) \tag{14}$$

where a_b is the coordinate of a corresponding to the label, that is if $b=1$ the coordinate of a would be the second, and a_c are the coordinates of a. First of all, we are going to ignore the weight w_b because it does not add any valuable information. The thing is, this expression measures how important is the coordinate of the label to the sum of the (exponential of the) coordinates of a. This makes sense, we do not want a vector to be in a concrete position, we want it to 'close' to one of the canonical basis of $\mathbb{R}^C$.

As stated before, being equal and being close is a difficult concept to define appropriately. Here, what we want, and what we are computing the error for, is that, for the output vector, most of its norm comes from the coordinate of the label. Notice that a_b is equivalent to $\langle a, e_b\rangle$ where e_b is the corresponding vector of the canonical basis of $\mathbb{R}^C$. In that case, the only coordinate that contributes to the norm is the coordinate of the label. Thus, we need to define an appropriate metric to reflect that this is what we want similar. For that, we have the following definition.

Definition 3.1. We define the **classification pseudodistance (or pseudometric)** in $\mathbb{R}^n$ to be $d_C\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ given by

$$d_C(x,y) = \|-\log(\sigma(x)) + \log(\sigma(y))\|_1 \tag{15}$$

where $\log(x) = (\log(x_1), \ldots, \log(x_n))$ and

$$\sigma(x) = \left(\frac{\exp(x_1)}{\sum_{i=1}^{n}\exp(x_i)}, \ldots, \frac{\exp(x_n)}{\sum_{i=1}^{n}\exp(x_i)}\right) \tag{16}$$

is the softmax function.

[3] As defined in definition 3.10.
[4] Lemma 2.13.

Remark 3.2. (1) Notice that we are using the softmax function to define a metric on the output space. Thus, it is independent of the structure of the category of the algorithms. In particular, it does not restrict the use of different activation functions for the ML algorithms.

(2) The pseudodistance is (up to constants) taking the differences of the entropy. The reason this is a good estimator can be found in [Sha48, section 6]. One of the important features is that the n-variable entropy has a maximum at $(1/n,\ldots,1/n)$ which corresponds to the most uncertain situation: the object is equally likely to be classified in any of the classes.

(3) Minimizing the error function, which in our case is the cross entropy, corresponds to reducing the uncertainty of the classification.

(4) This allows us to interpret the results since the pseudodistance in $\mathbb{R}^C$ allows us to make comparisons between distances in the outputs that are meaningful to the task at hand.

(5) In this case, the task is a classification task, and the metric defined in definition 3.1 is, *a priori*, only relevant for this task. However, this is just one instance of a metric space (X,d_X). Thus, considering learners as parameterized functions between metric spaces would yield information about the way the algorithm is working. Of course, this is dependent on the pseudodistance function used and should be adapted depending on the task at hand.

Now that we have established a distance for the output space we want to check if data points that are 'close' in $(\mathbb{R}^n,d_e)$ are mapped to the same class, points that are 'close' in $(\mathbb{R}^C,d_C)$. For that, we need to prove that a neural network with continuous activation functions, (or in general the learning algorithm) is a continuous map

$$(\mathbb{R}^n,d_e)\to\left(\mathbb{R}^C,d_C\right) \tag{17}$$

where $(\mathbb{R}^n,d_e)$ is $\mathbb{R}^n$ with the usual metric and $(\mathbb{R}^C,d_C)$ is $\mathbb{R}^C$ with the classification (pseudo)metric.

Proposition 3.3. *Any neural network with continuous activation functions is a continuous map from* $(\mathbb{R}^n,d_e)$ *to* $(\mathbb{R}^C,d_C)$, *where C is the number of classes.*

Proof. The map $(\mathbb{R}^n,d_e)$ to $(\mathbb{R}^C,d_C)$ can be viewed as the composition

$$(\mathbb{R}^n,d_e)\to\left(\mathbb{R}^C,d_e\right)\to\left(\mathbb{R}^C,d_C\right) \tag{18}$$

where the first map is the neural network and the second map is the composition $-\log\circ\sigma$, with log and σ as in definition 3.1. Then continuity follows by composition. Indeed, the neural network is the composition of linear, hence continuous, functions and activation functions that are by assumption continuous. The continuity with respect to the classification pseudodistance follows by the continuity of the logarithm and the softmax function. □

This essentially shows that the classification distance **depends continuously** on the (Euclidean) distance in the data. We now recall a definition from real analysis.

Definition. Let $f\colon\mathbb{R}\to\mathbb{R}$ be a real-valued function. We say f is **Lipschitz** if for all $x,y\in\mathbb{R}$ we have

$$|f(x)-f(y)|\leqslant L|x-y| \tag{19}$$

where $L>0$ is a constant.

This definition can be upgraded to any function between metric spaces.

Definition 3.5. Let $f\colon(X,d_X)\to(Y,d_Y)$ be a function between metric spaces. We say f is **Lipschitz** if for all $x,y\in X$ we have

$$d_Y(f(x),f(y))\leqslant Ld_X(x,y) \tag{20}$$

where $L>0$ is a constant.

The results in [AM21] suggest that learners can be seen as Lipschitz function from one (pseudo)metric space to another. Thus, the answer to the first question of this section seems to be in the very definition of the Lipschitz function: the constant. In order to check it out we need to understand how we can compute this number for a given neural network, or at least how to approximate it. For that, we recall some basic facts about Lipschitz functions.

(1) Any linear or affine mapping between two vector spaces with the Euclidean norm is Lipschitz. Indeed, such a map is of the form $x \mapsto Ax + b$ with A a matrix and b a vector. Then

$$\| (Ax+b) - (Ay+b) \|_2 = \|A(x-y)\|_2 \leqslant \|A\|_2 \|x-y\|_2. \tag{21}$$

Here, the Lipschitz constant is $L = \|A\|_2$.

(2) The composition of two Lischitz functions with constants L_1 and L_2 is Lipschitz with constant $L_3 = L_1 \cdot L_2$. Indeed, if $f\colon (A, d_A) \to (B, d_B)$ and $g\colon (B, d_B) \to (C, d_C)$ are Lischitz functions with constants L_1 and L_2 respectively we have

$$d_C(g(f(x)) - g(f(y))) \leqslant L_2 d_B(f(x) - f(y)) \leqslant L_1 \cdot L_2 d_A(x-y). \tag{22}$$

(3) Any bounded map with bounded derivative is Lipschitz. This is a consequence of the mean-value theorem.

The third fact implies that the most common activation functions of a neural network are Lipschitz: sigmoid, Relu, Leaky-Relu, and hyperbolic tangent are all Lipschitz. For a more detailed work on the bounds of the ReLu function see [AM21]. More relevant to us, the softmax function is Lipschitz as a map of Euclidean spaces as seen in [GP17, proposition 4]. We then have the following result.

Theorem 3.6. *A neural network (viewed as a parameterized function with a choice of parameters) classifier with linear layers and Lipschitz activation functions is a Lipschitz function of the metric spaces*

$$(B \subseteq \mathbb{R}^n, d_e) \to \left(\mathbb{R}^C, d_C\right), \tag{23}$$

where B is a bounded and closed subset of $\mathbb{R}^n$, C is the number of classes and d_C is the pseudodistance defined in definition 3.1.

Before the proof of the theorem, there is one thing to point out regarding the subset B. This seems to be a huge restriction from $\mathbb{R}^n$ and, in the general case, it is. However, since we are dealing with learning algorithms and the amount of data is finite we can always bound it. Furthermore, even if we have an infinite amount of data, the values of it cannot be anything. A lot of things, yes, but not anything. Thus, having an actual unbounded set of data is quite rare. In the literature, the kinds of sets considered to work with (local) Lipschitz functions are the so-called *pointed sets* or *pointed metric spaces* with finite diameters. Those are sets, o metric spaces, X with a distinguished point $x_0 \in X$ where the (local) Lipschitz constant is defined to be the following supremum

$$\sup_{y \in X, y \neq x_0} \frac{d_Y(f(x_0), f(y))}{d_X(x_0, y)}. \tag{24}$$

Doing this is another way to impose some bound on the diameter of the set X (see [AM21, vLB04]).

Proof. We begin the proof by noticing that this is not completely trivial. Indeed, the above remarks about Lipschitz functions imply that a neural network classifier with Lipschitz activation functions is itself a Lipschitz map between **Euclidean spaces**. It remains to prove that it satisfies the bounds for the distance d_C. For that, we are going to follow a similar strategy to the one we followed in proposition 3.3, that is we are going to prove that after we compose with the distance we get a Lipschitz function between Euclidean spaces.

First of all, notice that $(B \subseteq \mathbb{R}^n, d_e) \to (\mathbb{R}^C, d_C)$, since $\|x\|_2 \leqslant \|x\|_1$ for $x \in \mathbb{R}^C$, is equivalent to the composition

$$\begin{gathered}(B \subseteq \mathbb{R}^n, d_e) \to \left(\mathbb{R}^C, d_e\right) \to \left(\mathbb{R}^C, d_e\right) \\ x \mapsto I(x) = x' \mapsto v\end{gathered} \tag{25}$$

where I is the implement function of our algorithm, that is the neural network itself. The second function is the composition $(-\log) \circ \sigma$ with σ is the softmax function, i_v is the index that maximizes the norm of v, e_{i_v} is the vector of the canonical base of $\mathbb{R}^C$ and $\langle -, - \rangle$ denotes the standard scalar product in $\mathbb{R}^C$. Thus, to prove that the neural network is a Lipschitz function between metric spaces $(B \subseteq \mathbb{R}^n, d_e) \to (\mathbb{R}^C, d_C)$ is the same as to prove that the composition above is Lipschitz between euclidean spaces. By continuity of all the functions involved, what we actually have is a composition

$$(B \subseteq \mathbb{R}^n, d_e) \to \left(B^{(1)} \subseteq \mathbb{R}^C, d_e\right) \to \left(B^{(2)} \subseteq \mathbb{R}^C, d_e\right). \tag{26}$$

where each $B^{(i)}$ is a bounded and closed subspace. Thus, every function in the composition is bounded in those subspaces. Since the $B^{(i)}$ are bounded and closed the logarithms is Lipschitz and, by [GP17, proposition 4], the softmax is also Lipschitz. Thus $h = (-\log) \circ \sigma$ satisfies the Lipschitz condition between Euclidean spaces.

Hence, a neural network classifier with dense layers and Lipschitz activation functions is indeed a Lipschitz function of the metric spaces $(\mathbb{R}^n, d_e) \to (\mathbb{R}^C, d_C)$, where C is the number of classes and d_C is the pseudodistance defined in definition 3.1.

□

The preceding result tells us by how much the distance *might* get amplified. Another way of phrasing it is: that the Lipschitz constant is a bound on how much we allow the algorithm to strengthen the differences among points in the dataset. This shows how the framework incorporates known metrics, as shown in [vLB04] and [AM21], into the categorical setting. Thus, we are enlarging the describable phenomena with this unifying framework.

Since we have such a bound, we would like to interpret the bias of the network in relation to these bounds. For that, recall that the bias of a statistic T used to estimate a parameter θ is defined as

$$\text{bias}(T,\theta) = E(T) - \theta, \tag{27}$$

where $E(T)$ is the expected value of T. This encodes the difference between what is expected and what is measured. Following this, one might want to look at the difference in the 'expected relation between two data points' and the relation of a specific data point to the rest. In our case, this relation is encoded by the distance. We have the following definition.

Definition 3.7. Let $\{a^k, b^k\}_{k=1,\ldots,n}$, with $a^k \in \mathbb{R}^m$ and $b^k \in \mathbb{R}^C$, be our labeled data.

(1) We define the **mean distance of** a^k as

$$m_k = 1/n\|h^k\| = 1/n\|\left(h^k\left(a^1\right),\ldots,h^k\left(a^n\right)\right)\|_1 = \frac{1}{n}\sum_{i=1}^{n} d\left(a^k, a^i\right) \tag{28}$$

where $h^k(a^i) = d(a^k, a^i)$ denotes the Euclidean distance between a^k and a^i.

(2) We define the **bias** of a^k with respect to a^l as

$$b_{kl} = m_k - h^k\left(a^l\right). \tag{29}$$

This definition requires further explanation. The bias of an element with respect to another is the difference between the mean of the distances and the distance to that particular element. If the bias is greater than 0, then the distance between the elements is smaller than the average. This implies that a^l is closer than the average to a^l. That is say, a^l is more similar to a^k than the average. On the other hand, if the bias is smaller than 0 then the distance between the two elements is greater than the average and hence a^l is more different from a^k than the average. Finally, notice that if we have clusters, that is if we have subsets of vectors with mean distances similar among them but different to the mean distances of other subsets, by counting the cardinality of each of the subsets we can infer some sort of representation bias, that is, there is a subset with one particular trait that is underrepresented or overrepresented (see [MMS+21]).

One of the important ideas that follow from the Yoneda embedding is the idea of a representable functor. This is a functor between categories $F\colon \mathcal{C} \to \mathcal{D}$ such that there exists an object c of the category $\mathcal{C}$ satisfying $F(x) = h^c(x) = \text{hom}(c, x)$ for all objects x of $\mathcal{C}$. In the case of word embeddings, if one wishes to know if a certain embedded word c is gender biased one would look at the difference between the cosine distance between the word c and the words *man* and *woman* (see [CAC+22]). This can be translated, in the enriched setting, into

$$h^c(man) - h^c(woman). \tag{30}$$

As we can see, the functor h^c can give us important information about the elements. In particular, if the vector is a vector of attributes we can check the difference of each between the mean and a particular element for each coordinate. This would lead us to a vectorial definition of bias:

$$\overrightarrow{b_{kl}} = \overrightarrow{m_{kl}} - \overrightarrow{|a^k - a^l|}. \tag{31}$$

In a sense, this gives a how close each of the attributes is with respect to the mean.

If furthermore, the data is labeled we can consider for each element a^k of the dataset the differences between the mean distance of each coefficient for the elements in the same class and for the elements of other classes. That is, given a^k with class c we consider

$$\overrightarrow{m_{k|c}} - \overrightarrow{m_{k|c'}} \tag{32}$$

with $c' \neq c$, $|c|$ is the number of elements of class c and

$$\overrightarrow{m_{(k|c)i}} = \frac{1}{|c|} \sum_{a_l \in c} |a_i^k - a_i^l| \tag{33}$$

for i that ranges through the number of attributes. If attribute i is important to determine the class c, then the difference in mean distances should be noticeable. To summarize, we have the following definition.

Definition 3.8. Let $\{a^k, b^k\}_{k=1,\dots,n}$, with $a^k \in \mathbb{R}^m$ and $b^k \in \mathbb{R}^C$, be our labeled data. That is, k iterates over the instances of the data, m represents the dimension of each data point and C represents the dimension of the labels. We assume the vectors b^k are one-hot encodings of the classes of a^k thus considering the vector b^k a proxy for the class c_k of a^k.

(1) We define the **scalar mean distance of a^k to the class** c to be

$$m_{k|c} = \frac{1}{|c|} \sum_{a^l \in c} d\left(a^k, a^l\right) = \frac{1}{|c|} \sum_{a^l \in c} h^k\left(a^l\right). \tag{34}$$

of a^k to the class c.

(2) If the coordinates of a^k are attributes, we define the **vectorial mean distance of a^k to the class** c to be

$$\overrightarrow{m_{k|c}} = \frac{1}{|c|} \left(\sum_{a^l \in c} |a_1^k - a_1^l|, \dots, \sum_{a^l \in c} |a_m^k - a_m^l| \right). \tag{35}$$

This is the vector, of dimension m, of the mean distance of each attribute.

(3) We define the **mean scalar distance of the class c to the class c' to be**

$$m_{c,c'} = \frac{1}{|c'|} \sum_{a^k \in c'} m_{k|c} \tag{36}$$

(4) If the coordinates of a^k are attributes, we define the **vectorial mean distance of the class c to the class c'** to be

$$\overrightarrow{m_{c,c'}} = \frac{1}{|c'|} \sum_{a^k \in c'} \overrightarrow{m_{k|c}}. \tag{37}$$

If $c = c'$ we denote $\overrightarrow{m_{c,c'}} = \overrightarrow{m_c}$.

To complete definition 3.8 with the categorical picture, we need to specify a category of enrichment that yields the vector distances as the result of $h^c(k)$. This is achieved by enriching the category of the data over the product monoidal category $\mathcal{V}^A = [0, \infty]^A$, where A is the number of attributes or number of entries per point in the dataset. The vectorial mean distances can be used to infer some sort of relevance of the attributes: if the mean distance of an attribute from one class to another changes significantly then we can separate the classes by the attribute. The same, albeit with less precision can be said about the scalar mean distance between classes. This is essentially the idea behind the k nearest neighbors classifier. Notice that, by the definition of a distance function, these metrics cannot be obtained by considering the data as a metric space.

The next corollary follows from theorem 3.6 and definition 3.7.

Corollary 3.9. *Given a neural network classifier with Lipschitz constant K, we have the following bound on the biases:*

$$\overline{b_{kl}} \leqslant K \cdot b_{kl}, \tag{38}$$

where $\overline{b_{kl}}$ is the bias of the outputs w^l with respect to w^k and b_{kl} is the bias on the inputs x^l with respect to x^k.

The bias also gets amplified, which makes sense since the Lipschitz constant is a metric of how much we allow the differences to be strengthened. This implies that the bias, which in some sense is a metric of the relevance of a data point to the overall difference of a given element, will also get amplified by a factor at most the constant.

Thus, to any neural network with linear layers, fixed weights and biases, and Lipschitz activation functions we can attach a number: the Lipschitz constant. The constant establishes a maximal proportionality coefficient between the distances of the inputs and outputs. The constant is intrinsic to each algorithm with fixed weights and biases. To summarize, we are going to consider the categorical datasets to consist of a collection of objects (the data points) and the arrows between them transformations form one another[5]. On top of that, we will consider two main enrichments of the categorical datasets: over the monoidal category $[0,\infty]$ or over the product category $[0,\infty]^A$. The former turns the categorical dataset into a generalized metric space in the sense of [Law73]. The latter allows us to retrieve statistical properties of the dataset thanks to definition 3.8.

3.2. Decorated algorithms

Having enriched the category of the data over $\mathbb{R}_+$ or $\mathbb{R}_+^A$, two questions arise. First, is there a way to attach to each network a parameter or a number that predicts (or at least bounds) by how much the distance is going to be amplified? Having such a parameter is similar to rule-based interpretation techniques since it gives us an estimation of how important the relative position of a data point with respect to the others is to the algorithm. This means that the distance between elements of the dataset is a variable of the rules that the algorithm is using to reach its decision. Furthermore, if the is a change in the input, the parameter would be bound to the resulting change in the output which may change the prediction.

Secondly, if we consider the class of such networks, i.e. networks with a parameter attached to them, do they form a category in the mathematical sense? Knowing that this class of objects has a nice structure would allow us to exploit the structure to study the properties of these algorithms and propose modifications that then can be applied to each case.

We will start with the first question: how to attach to a learning algorithm a constant (or a function that gives us the constant for each choice of parameters) that bounds the distance amplification? The first thing to note is that the categories Learn, Para, and NNet are already enriched over the category of categories (as hinted in [FST19, section VII.C]). That is, the morphisms between two objects in those categories form a category.

We now show how to consider Learn an enriched category over Cat[6].

(1) Since $(\mathrm{Cat},\times)$ is a monoidal category we can consider it as a base of enrichment. In the case of $\mathrm{Para}_{\mathrm{ML}}$, given a pair of objects (X,d_X) and (Y,d_Y), the hom-space $\mathrm{Para}_{\mathrm{ML}}(X,Y)$ is the category whose objects are equivalence classes of parameterized functions $(P,I)\colon X\to Y$ (one-cells in the bicategory framework). Morphisms in $\mathrm{Para}_{\mathrm{ML}}(X,Y)$ are functions $f\colon P\to Q$ such that

$$J(f(p),x)=I(p,x) \tag{39}$$

for all $p\in P$ and $x\in X$ (that is, the two-cells in the bicategory framework). Composition is the functor:

$$\circ_{X,Y,Z}\colon \mathrm{Para}_{\mathrm{ML}}(X,Y)\times\mathrm{Para}_{\mathrm{ML}}(Y,Z)\to\mathrm{Para}_{\mathrm{ML}}(X,Z) \tag{40}$$

given by $((P,I)(Q,J))\mapsto(P\times Q,J\circ I)$ (called composition in series in [FST19]). Finally, the identity functor is given by $j_X\colon\{*\}\to\mathrm{Para}_{\mathrm{ML}}(X,X)$ sending $*\mapsto(\{*\},Id_X)$. The triangle and pentagon identities follow from the fact that $\mathrm{Para}_{\mathrm{ML}}(X,Y)$ follow from the observation made above that we can reinterpret a bicategory as an enriched category and that a bicategory satisfies triangle and pentagon identities.

(2) The case of $\mathrm{Learn}_{\mathrm{ML}}$, is analogous to the case of $\mathrm{Para}_{\mathrm{ML}}$. Indeed the same structure applies, just substituting the objects of $\mathrm{Learn}_{\mathrm{ML}}(X,Y)$ by tuples (P,I,U,r) as in [FST19]. Morphisms in $\mathrm{Learn}_{\mathrm{ML}}(X,Y)$ are $f\colon P\to Q$ satisfying equation (2). The rest of the properties are analogous to the previous case, just need to modify the spaces, and the relations are the ones found in [FST19].

What we want is to attach to each morphism some sort of parameter. This corresponds to the concept of a *decoration*, see [Fon15, BCV21]. In those manuscripts, the authors use the decorations to model electrical

[5] In the datasets considered in section 4, the objects are vectors and the arrow between two data points the vector from one point to the other.

[6] The necessary definitions have been added to appendix.

circuits and Petri Nets. One thing to note is that the precise definition of an F-decoration varies from one manuscript to another, as noted in [BCV21, section 2].

The main idea is that **decoration** consists of a way to attach (functorially) another structure to an object of a category. Thus, what we want is to find a suitable decoration for the categories mentioned in the previous section that helps us keep track of the structure of the data. Formally, for us, a decoration will have the following definition.

Definition 3.10. Let $\mathcal{A}$ a category and $\mathcal{M}$ a monoidal category. An F-**decoration** of $\mathcal{A}$ is a functor $F\colon \mathcal{A} \to \mathcal{M}$.

The definition of a decoration is different from the ones found in the bibliography. We have defined it as a functor to stress the fact that it preserves the structure and compositionality: it formalizes categorically the properties of the Lipschitz functions of equation (22). With that in mind, we want to decorate the categories over Lipschitz functions. In order to do that we need to consider more general spaces than just sets for the categories Learn, Para, and NNet.

Definition 3.11. We denote by $\mathrm{Para}_{\mathrm{ML}}$ the symmetric monoidal category whose objects are pseudometric spaces with a continuous injective map $(X, d_X) \to (\mathbb{R}^n, d_e)$ for some $n \in \mathbb{N}$. Morphism is this category are equivalences classes of parameterized functions $f = (P, I)\colon (X, d_X) \to (Y, d_Y)$ such that:

(1) The following diagram commutes

$$\begin{tikzcd}
(X, d_X) \arrow[r, "f"'] \arrow[d, "i"] & (Y, d_Y) \arrow[d, "i'"] \\
(\mathbb{R}^n, d_e) \arrow[r, "f'"] & (\mathbb{R}^m, d_e)
\end{tikzcd} \quad (41)$$

where $f = (P, I)$ is the parametrized function and $f' = (P, I')$ is a differentiable parametrized function as in [FST19, definition III.1].

(2) For all $p \in P$, the function $I(p, -)\colon (X, d_X) \to (Y, d_Y)$ is Lipschitz.

Composition and the monoidal structure in $\mathrm{Para}_{\mathrm{ML}}$ work as in Para.

Remark 3.12. (1) The function $f = (i')^{-1} \circ f' \circ i$. Since i' is injective continuous, $i'\colon X \to \mathrm{Im}(i')$ is bijective, where $\mathrm{Im}(i')$ denotes the image set of i'. Hence, it has a (local) inverse.

(2) This is indeed a symmetric monoidal category since composition and monoidal product work as in Para, that is, we can compose two diagrams of the form (41) to get a diagram of the same form and the product of pseudometric spaces is a pseudometric space with the product pseudometric.

(3) One would have to check that two equivalent parameterized functions have the same Lipschitz constant in order to see that $\mathrm{Para}_{\mathrm{ML}}$ is well-defined. But this follows immediately from the definition of equivalent parameterized functions.

(4) If we are dealing with a classification problem the space (Y, d_Y) would be $(\mathbb{R}^C, d_C)$ where C is the number of classes and d_C is the classification pseudodistance. Thus, this framework generalizes and axiomatizes what has been done before.

Since for all $p \in P$, the function $I(p, -)$ is Lipschitz, we can associate to every morphism in $\mathrm{Para}_{\mathrm{ML}}$ a parameterized function $(P, L_I)\colon \{*\} \to \mathbb{R}_+$, where $\{*\}$ is a one-point space and $\mathbb{R}_+ = \{x \in \mathbb{R} \colon x \geqslant 0\}$. That is, $L_I\colon P \times \{*\} \to \mathbb{R}$ sending $p \mapsto L_I(p, *) = L$, where L is the Lipschitz constant of the function $I(p, -)$.

Hence, we can decorate the category $\mathrm{Para}_{\mathrm{ML}}$ over the set of parameterized functions $(P, L_I)\colon \{*\} \to \mathbb{R}_+$, denoted by $\mathcal{L}$, up to the equivalence of parameterized functions. This set is actually a monoidal category. Indeed, we denote by $(\mathcal{L}, \otimes)$ the monoidal category whose objects are parameterized functions over the point up to equivalence, with the monoidal product given by $(P, L_1) \otimes (Q, L_2) = (P \times Q, L_1 \cdot L_2)$ where the product of functions is the pointwise product. The monoidal neutral element is the parameterized function $(\{*\}, e)$ with $e(*, *) = 1$. That is, given two parameterized functions $(P, L_I)\colon X \to Y$ and $(Q, L_J)\colon Y \to Z$ the product function is

$$(P, L_I) \otimes (Q, L_J) = (P \times Q, L_I \cdot L_J)\,(p, q, *) = L_I\,(p, *) \cdot L_J\,(q, *) \in \mathbb{R}_+ \quad (42)$$

for all $(p, q) \in P \times Q$. Morphisms are given by maps from one parameter space to the other $f\colon (P, L_I) \to (Q, L_J)$ by the rule

$$f((P, L_I)\,(p, *)) = f(P, L_I)\,(f(p)\,, *) = L_J\,(f(p)\,, *)\,. \quad (43)$$

Given two morphisms $f\colon (P,L_I)\to(Q,L_J)$ and $g\colon (Q,L_J)\to(R,L_K)$, their composition is given by

$$(g\circ f)\left((P,L_I)(p,*)\right)=L_K\left((g\circ f)(p),*\right). \tag{44}$$

In this setting, the neutral element for the monoidal operation $\otimes$, $(\{*\},e)$ is terminal in $(\mathcal{L},\otimes)$. Finally, it is worth noting that the category $\mathcal{L}$ can be seen as the categoy $\mathrm{Para}(\{*\},\mathbb{R}_+)$.

Remark 3.13. If the parameter spaces have extra structure, for instance, if they are topological spaces, we require the maps from parameter spaces to be continuous.

Lemma 3.14. *The assignment $[P,I]\mapsto[P,L_I]$ is indeed a decoration. That is, there is a functor $L\colon \mathrm{Para}_{\mathrm{ML}}(X,Y)\to\mathcal{L}$ sending $[P,I]\mapsto[P,L_I]$ and every morphism of parametrized functions $f\colon [P,I]\to[Q,J]$ to L_f such that $L_f(L_I(p,*))=L_J(f(p),*)$.*

Proof. In order to prove the result we must check that given two representatives of the same class, the functor L sends them to the same function. We have the following commutative diagram

$$\begin{tikzcd}
(P,I) \arrow[r, "f"'] \arrow[d, "L"] & (Q,J) \arrow[d, "L"] \\
(P,L_I) \arrow[r, "L_f"] & (Q,L_J)
\end{tikzcd} \tag{45}$$

with $J(f(p),*)=I(p,*)$, $L_f(P,L_I)(f(p))=L_I(p,*)$ by definition of $f\colon (P,I)\to(Q,J)$ and L. The commutativity of the diagram implies that the functor is well-defined for equivalence classes of parameterized functions $[P,I]$ since two representatives of the same class will have the same image (and the same image class). Finally, again by the commutativity of the diagram, L is indeed a functor which is the categorical counterpart of equation (22). □

Remark 3.15. To decorate parameterized functions with a function that provides their Lipschitz constant for specific parameter selections, it is essential for the categories Para and $\mathrm{Para}_{\mathrm{ML}}$ to be enriched within the category of categories: the categories we are decorating are the hom-categories between two spaces. Therefore, this enrichment serves as a prerequisite for the presence of the decoration.

Definition 3.16. We denote by $\mathcal{P}ara$ the symmetric monoidal category $\mathrm{Para}_{\mathrm{ML}}$ whose hom-categories are decorated over $\mathcal{L}$.

When a category-enriched category $\mathcal{C}$ has its hom-categories decorated we say $\mathcal{C}$ **is hom-decorated.** Notice that to recover the original definition from [Fon15] the only thing that is needed to decorate over $\mathbb{R}_+$ and make the choice of Lipschtz constant. However, since the choice can be made freely (with independence of the parameters) and the Lipschitz constant depends on those parameters we have opted to give this alternative definition. We can now do the analogous process to Learn.

Definition 3.17. (1) We define the symetric monoidal category $\mathrm{Learn}_{\mathrm{ML}}$ to be the category whose objects are pseudometric spaces (X,d_X) and whose morphism are equivalence calsses of learners (P,I,U,r) such that the implement function $I\colon P\times X\to Y$ is Lipschitz in the second variable, that is, $I(p,-)\colon (X,d_X)\to(Y,d_Y)$ is Lipshitz for all $p\in P$.
(2) We denote by $\mathcal{L}earn$ the category $\mathrm{Learn}_{\mathrm{ML}}$ whose hom-categories are decorated over $\mathcal{L}$.

Remark 3.18. It is important to notice that $\mathcal{L}$ is not the set of Lipschitz constants, but the monoidal category of functions that, to each choice of parameters of a parameterized function, assign a constant which is the Lipschitz constant for a specific choice of parameters. The fact that the Lipschitz constant depends on the choice of parameters is the reason why the categories have to be enriched over the monoidal category $(\mathcal{L})$.

We then have the decorated version of theorem 2.7.

Theorem 3.19. *Let $\alpha>0$ and $e\colon \mathbb{R}\times\mathbb{R}\to\mathbb{R}$ differentiable such that $\frac{\partial e}{\partial e}(x_0,-)\colon\mathbb{R}\to\mathbb{R}$ is differentiable for $x_0\in\mathbb{R}$. We have a faithful on objects, symmetric monoidal functor, that respects the decoration,*

$$\mathcal{L}_{\alpha,e}\colon \mathcal{P}ara\to\mathcal{L}earn \tag{46}$$

that sends each metric space to itself and each diagram

$$\begin{tikzcd}(X, d_X) \arrow[r, "f"'] \arrow[d] & (Y, d_Y) \arrow[d] \\ (\mathbb{R}^n, d_e) \arrow[r, "f'"] & (\mathbb{R}^m, d_e)\end{tikzcd} \tag{47}$$

with $f = (P, I)$ and $f' = (P, I')$ to

$$\begin{tikzcd}(X, d_X) \arrow[r, "F"'] \arrow[d] & (Y, d_Y) \arrow[d] \\ (\mathbb{R}^n, d_e) \arrow[r, "F'"] & (\mathbb{R}^m, d_e)\end{tikzcd} \tag{48}$$

with $F = (P, I, U_I, r_I)$ and $f' = (P, I', U_{I'}, r_{I'})$. The update $U_{I'}$ and request $r_{I'}$ functions are the same as in definition 2.7 for $f' = (P, I')$, that is:

$$U_{I'}(p, a, b) = p - \alpha \nabla_p E_{I'}(p, a, b) \tag{49}$$

and

$$r_{I'}(p, a, b) = f_a(\nabla_a E_{I'}(p, a, b)) \tag{50}$$

with $E_{I'}(p, a, b) = \sum_j e(I'(p, a)_j, b_j)$ and f_a is the component-wise application of the inverse to $\frac{\partial e}{\partial x}(a_i, -)$ for each i.

Proof. The functor is the identity of objects and on morphisms, it sends f' to F' which is injective by theorem 2.7. The existence of F follows by the commutativity of the diagram

$$\begin{tikzcd}(X, d_X) \arrow[r, "f"'] \arrow[d] & (Y, d_Y) \arrow[d] \\ (\mathbb{R}^n, d_e) \arrow[r, "f'"] & (\mathbb{R}^m, d_e).\end{tikzcd} \tag{51}$$

Finally, the fact that it respects the decoration follows from the fact that the Lipschitz function is attached to the implementation function which is the same in both categories. □

Remark 3.20. Notice that we are using the top and bottom parts of the diagram for two different purposes. The bottom part is used to implement the gradient descent algorithm whereas the top part is used to explain and interpret the results given by the algorithm. This means that the framework does not interfere with the optimization phase of the algorithm. It merely adds structure to understand how and why the algorithm has reached the results. Hence, the accuracy metrics of the algorithm are unaffected by the framework.

We would like to mention that there are cases and frameworks where one can define a non-euclidean version of gradient descent (see [HKM+18]). The only thing that remains to be done is to prove that NNet also fits in the decorated framework. Thus, we are going to consider its hom-decorated counterpart as a subcategory of the category $\mathcal{Para}$.

Definition 3.21. Let $\mathcal{MNet}$ be the symmetric monoidal subcategory of $\mathcal{Para}$ with hom-categories decorated over $\mathcal{L}$, whose objects are objects in $\mathcal{Para}$. A **simple morphism** between two objects is a diagram

$$\begin{tikzcd}(X, d_X) \arrow[r, "f"'] \arrow[d] & (Y, d_Y) \arrow[d] \\ (\mathbb{R}^n, d_e) \arrow[r, "f'"] & (\mathbb{R}^m, d_e)\end{tikzcd} \tag{52}$$

where f' is of the form

$$f' : \mathbb{R}^{|C|+m} \times \mathbb{R}^n \to \mathbb{R}^m \tag{53}$$

$$\left(\left(w_{ji}, w_j\right), a\right)_{\substack{1\leqslant i\leqslant m\\ 1\leqslant j\leqslant n}} \mapsto \left(\tau\left(\sum_i w_{ji}\cdot a_i + w_j\right)\right). \tag{54}$$

where $C \subseteq [n]\times[m]$ and τ is Lipschitz. The morphisms in this category are equivalent classes of compositions of simple morphisms.

Remark 3.22. (1) The choice of activation functions τ is independent of the metrics d_X and d_Y.
(2) The functor of theorem 2.11 is the inclusion $\mathcal{M}\mathcal{N}et \hookrightarrow \mathcal{P}ara$ in our case.
(3) We are implicitly working with a stronger version of neural nets called *acyclic directed graphs with interface* or 'idags' for short. See [FDC13] for more information on the subject.

One thing this approach can be used for is the determination of the relevance of attributes. To do so, one would decompose the learner into irreducible parts and analyze the relevance of each part. In the case of neural networks, all entries follow the same paths to the out with the exception of the path connecting the input layer to the first layer. Thus, by analyzing the weights and biases of the first layer one should be able to give a ranking by the relevance of the attributes. More generally, studying the non-equivalent parts of the different learners and how they treat the data should give us an idea of what the algorithm is seen as important (that might differ from what a user would consider important). This would yield similar results to other known methods such as [CMJG19]. These methods are reliant on a dimensionally large input set such as the ones found in sentiment analysis and are not really useful when confronted with datasets with fewer entries.

Finally, it is worth noting that we have attached to each algorithm a function that gives its Lipschitz constant for a choice of parameters. This gives a bound on how much the algorithm can distort the representation of the data. Not only that but since it bounds the distance between any pair of points, it yields the relative positions of all the points up to some deformation. It is a consequence of having, for each pair of points, a system of inequalities. Furthermore, in the same way, that having a general theory of systems of equations led to the solution, or at least a solvability criterion, having a general categorical framework could lead to criteria for a better understanding of the algorithms. We conclude this section by mentioning that this framework is more general than neural networks as is evidenced by the fact that the hom-decorated category $\mathcal{M}\mathcal{N}et$ is a subcategory of $\mathcal{P}ara$ and that we have a functor between $\mathcal{P}ara$ and $\mathcal{L}earn$. This means that any algorithm with a parameter space, implementation, update, and request functions fits into the framework.

In the next section, we are going to study two examples using the categorical framework in order to see what the algorithms are doing to the datasets.

4. Case of study

In order to showcase the framework we study three different examples. The first one will be a prediction algorithm: a neural network with two hidden layers tasked to predict the selling price of a house using the Boston dataset. The second example will be a classification algorithm: a neural network classifier with two hidden layers tasked to predict the species of a flower using the Iris dataset. The third example will be a weather forecasting algorithm used to predict the minimum, maximum, and mean temperature given numerical data.

We have chosen these examples for two main reasons: the first one is that they are both well-studied and understood, so we can check our predictions. The second is that we have two different metric spaces to test the framework. Furthermore, since the data is all numerical data representing distinct attributes, it is easier to explain and interpret the results. Finally, it is worth noting that with these examples we have three different situations: algorithms between Euclidean spaces and one between Euclidean space and a classification space. Furthermore, we have different types of enrichment (of the datasets) to showcase the flexibility of the framework.

Moreover, the categorical structure allows us to analyze and contextualize the results. This means that the precision of the algorithms is unaffected. In addition, since the framework exposed in section 3 encompasses a large set of algorithms and we will focus on neural networks, the main results that will be used in this section are the ones relating to the data and to neural networks specifically[7].

Considering that this is meant to be understood by humans and not by machines, giving a matrix whose rows are the vectors of distances of the element to all other elements would be useless. It does carry almost all the information that is needed about the points but it is not effectively readable by the end user. One method to make it clearer is to take the mean. Indeed, one loses information, due to the reduction of dimensionality,

[7] This, of course, with the knowledge that the framework can be applied to a wide range of ML algorithms represented by theorem 3.19.

Table 1. Lipschitz functions of our architectures for the Boston dataset.

Boston dataset	Parameter space (P_i)	Lipschitz function (L_i)
Architecture 1	$P_1 = \mathbb{R}^{12\times16+16} \times \mathbb{R}^{16^2+16} \times \mathbb{R}^{16+1}$	$L_1(p) = \|W_{1,1}\|_2 \cdot \|W_{1,2}\|_2 \cdot \|W_{1,3}\|_2$
Architecture 2	$P_2 = \mathbb{R}^{12\times32+32} \times \mathbb{R}^{32^2+32} \times \mathbb{R}^{32+1}$	$L_2(p) = \|W_{2,1}\|_2 \cdot \|W_{2,2}\|_2 \cdot \|W_{2,3}\|_2$
Architecture 3	$P_3 = \mathbb{R}^{12\times8+8} \times \mathbb{R}^{8^2+8} \times \mathbb{R}^{8+1}$	$L_3(p) = \|W_{3,1}\|_2 \cdot \|W_{3,2}\|_2 \cdot \|W_{3,3}\|_2$
Architecture 4	$P_4 = \mathbb{R}^{12\times16+16} \times \mathbb{R}^{16+1}$	$L_4(p) = \|W_{4,1}\|_2 \cdot \|W_{4,3}\|_2$

but it still carries some information that can be acted upon. For instance, if one has a point whose mean distance to the rest is d and another whose mean distance is $4d$, one can conclude that the first point is closer to the center of the data than the second (which can be an outlier). For this reason, we have plotted the mean of the distances of the inputs and outputs of the algorithm along with the quotient (which is bounded by the Lipschitz constant).

The quotient serves us as a metric of how much the distances are being amplified. That is, how much the algorithm is exaggerating the differences among elements to be sure it does not make a prediction error. Thus, it can help us measure the damage done by an unbalanced dataset: if there is an asymmetry in the dataset, the algorithm would amplify the asymmetry by a certain factor, which in our case is the quotient of the distances. This means that knowing to quotient can help us to know how potentially harmful a bias or asymmetry in the data would be to the algorithm.

4.1. Boston dataset

The Boston dataset has 506 rows and 14 columns. Of these 14 columns one is the target price, which acts as our label and one is the proportion of blacks that we have discarded for our analysis. Thus, our data consists of 12 vectors and one label. The entries of the vectors are various geographical, architectural, and social parameters.

Thus, the category $\mathcal{B}oston$ is the category whose objects are tuples of 12 entries, one per attribute. The morphisms are linear transformations from one tuple to another, the composition of morphism is given by the composition of linear transformations. This category can be enriched over the monoidal category $\mathcal{V} = \{[0, +\infty], \geqslant, +\}$, by taking each morphism to be the distance between two objects, making it a generalized metric space [Law73]. With this we can give a map from the set of objects of the category, denoted $\mathcal{B}oston_0$, to $\mathbb{R}^{12}$ which puts us in the situation of definition 3.21.

The chosen architecture for the learner of the Boston dataset was that of a two-hidden layer neural network with 16 neurons and a ReLU activation function. The rationale to do so is twofold: we want to have at least two hidden layers to have a matrix with most of the Lipschitz constant of the algorithm. The other reason is that we want to remove one of the layers for our testing and it would be better to have a hidden layer. Thus the algorithm can be seen as the composition

$$(\mathbb{R}^{12}, d_e) \to (\mathbb{R}^{16}, d_e) \to (\mathbb{R}^{16}, d_e) \to (\mathbb{R}, d_e). \tag{55}$$

Furthermore, the morphisms i, i' from definition 3.11 are identities of the Euclidean spaces. Notice that the Lipschitz constant for the ReLU is 1 and for the linear part is the norm of the weight matrix. We have considered four different architectures. the first one, the original, with 16 neurons on each of the hidden layers, the second and third ones with double and half the number of neurons in each layer respectively, and the last architecture with just one hidden layer with 16 neurons. Thus, we have the functions L_I of definition 3.21 on the table 1 where $W_{i,j}$ are the weight matrices corresponding to ith architecture and jth component of the parameter space. This means that changing the parameter space (shown in the second column of table 1), which is equivalent to changing the number of layers and neurons of the neural network, has a precise and measurable impact on the function L_i. This function in turn yields, for every choice $p_i \in P_i$, the Lipschitz constant of the algorithm $L_i(p_i)$ (shown in the third column of the table).

As we can see, the increase in parameter space increases the Lipschitz constant of the architecture. The Lipschitz constants for the implementations are summarized in table 3. With that, we can check the effect of the bounds on the distances.

Table 2 shows the maximum mean distance on the output space for each of the four architectures we have tested. This can also be appreciated in figure 2, showing the mean distances on the input space (in red, dotted line) in the middle, the distances on the output space (in blue, dashed lane) at the top, and the quotient of the distances (in green, continuous line) at the bottom. The quotient of distances (green, continuous) is what should be bounded by the Lipschitz constant. As we can see, the quotient does lie below the bound. However, the bound is not strict since for the first architecture the bound is 34, 80 and the quotient lies between 1 and

Table 2. Maximum of the mean distance for each architecture of the Boston dataset.

Boston dataset	Architecture 1	Architecture 2	Architecture 3	Architecture 4
Max mean distance	$24,75$	$24,02$	$21,78$	$25,67$

Table 3. Lipschitz constants of the linear part of each layer for the Boston dataset.

Boston dataset	Architecture 1	Architecture 2	Architecture 3	Architecture 4
2-norm of the first layer	5,20	5,80	5,14	8,45
2-norm of the hidden layer	4,78	6,08	3,50	—
2-norm of the output layer	1,40	1,20	1,44	2,83
Constant of the algorithm	34,80	42,32	27,22	23,91

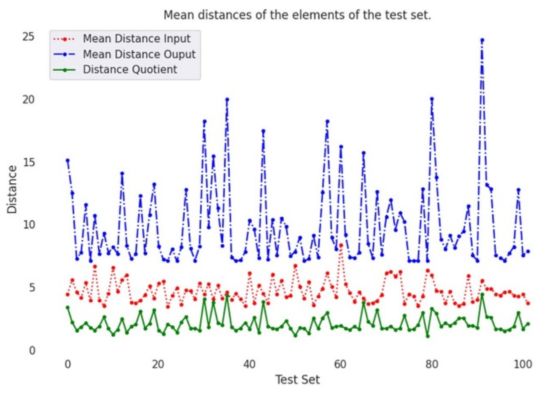


Figure 2. Mean distances of the test set for the Boston dataset with the first architecture.

5. This can be explained by looking at an alternative definition of the Lipschitz constant of a function $f\colon X \to Y$ as

$$\sup_{x,y\in X} \frac{d_Y\left(f(x), f(y)\right)}{d_X\left(x,y\right)}. \tag{56}$$

Hence, the constant is a worst-case scenario of the dilation caused by the function. Notice that this is consistent with the (strict) bounds of [AM21], that are of order 10^9. Furthermore, there seems to be little to no difference in distance dilation among the four different architectures, see table 2, even though the constant of the algorithm does change. Again, this is consistent with the worst-case scenario explanation: just by the fact that the function allows for more distance amplification does not imply that it is optimal to amplify it. The distances defined in definition 3.8, which are the result of a change in the enrichment category, correspond to the dispersion of the data per attribute. Since in this case, we do not have classes one could consider the whole dataset pertaining to the same class. Alternatively, one could consider price intervals as the classes and do the analysis for these classes.

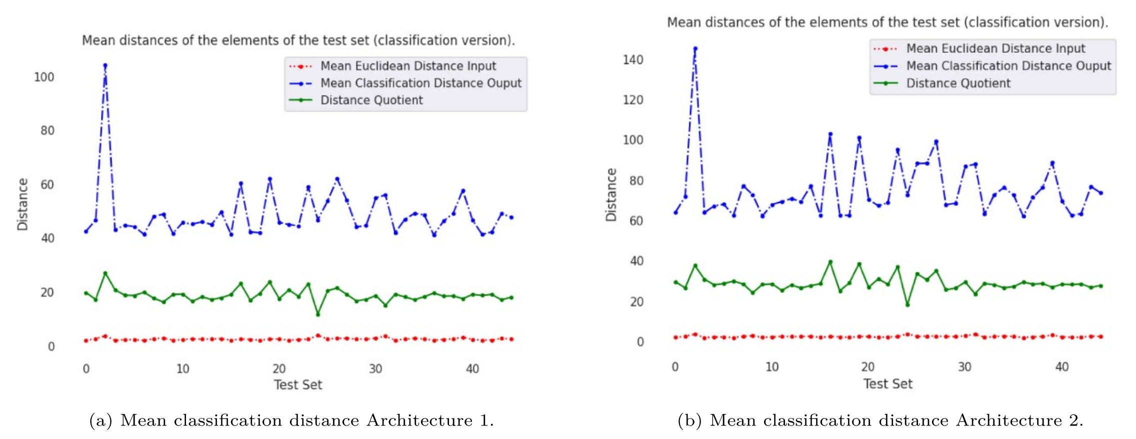
4.2. Iris dataset

We now turn our attention to the Iris dataset. This dataset consists of four dimensional $\mathbb{R}$ valued vectors plus a label categorizing the species of flower (a parameter that can take three different values). The vectors consist of the data: sepal length, sepal width, petal length, and petal width for each flower.

In this case the category $\mathscr{Iris}$ is the category whose objects are tuples of four attributes and morphisms are linear transformations from one object to another. As in the $\mathscr{Boston}$ the category can be enriched over $\mathscr{V}$ making it a generalized metric space thus allowing us to use the results from section 3.

We have used a neural network with two hidden layers with 64 neurons each. The activation functions are ReLU functions and the loss function, which is the total error function of theorem 2.7, is the CrossEntropyLoss function from PyTorch. The reasons to choose this specific architecture are analogous to the ones explained in the Boston dataset case. Here the algorithm can be seen as the composition

$$\left(\mathbb{R}^4, d_e\right) \to \left(\mathbb{R}^{16}, d_e\right) \to \left(\mathbb{R}^{16}, d_e\right) \to \left(\mathbb{R}^3, d_C\right). \tag{57}$$



(a) Mean classification distance Architecture 1.

(b) Mean classification distance Architecture 2.

Figure 3. Mean distances of the test set for the Iris dataset.

Table 4. Lipschitz functions of our architectures for the Iris dataset.

Iris dataset	Parameter space (P_i)	Lipschitz function (L_i)
Architecture 1	$P_1 = \mathbb{R}^{4\times64+64} \times \mathbb{R}^{64^2+64} \times \mathbb{R}^{64\times3+3}$	$L_1(p) = \|W_{1,1}\|_2 \cdot \|W_{1,2}\|_2 \cdot \|W_{1,3}\|_2$
Architecture 2	$P_2 = \mathbb{R}^{4\times128+128} \times \mathbb{R}^{128^2+128} \times \mathbb{R}^{128\times3+3+1}$	$L_2(p) = \|W_{2,1}\|_2 \cdot \|W_{2,2}\|_2 \cdot \|W_{2,3}\|_2$
Architecture 3	$P_3 = \mathbb{R}^{4\times32+32} \times \mathbb{R}^{32^2+32} \times \mathbb{R}^{32\times3+3}$	$L_3(p) = \|W_{3,1}\|_2 \cdot \|W_{3,2}\|_2 \cdot \|W_{3,3}\|_2$
Architecture 4	$P_4 = \mathbb{R}^{4\times64+64} \times \mathbb{R}^{64\times3+3}$	$L_4(p) = \|W_{4,1}\|_2 \cdot \|W_{4,3}\|_2$

Table 5. Lipschitz constants of the linear part of each layer for the Iris dataset.

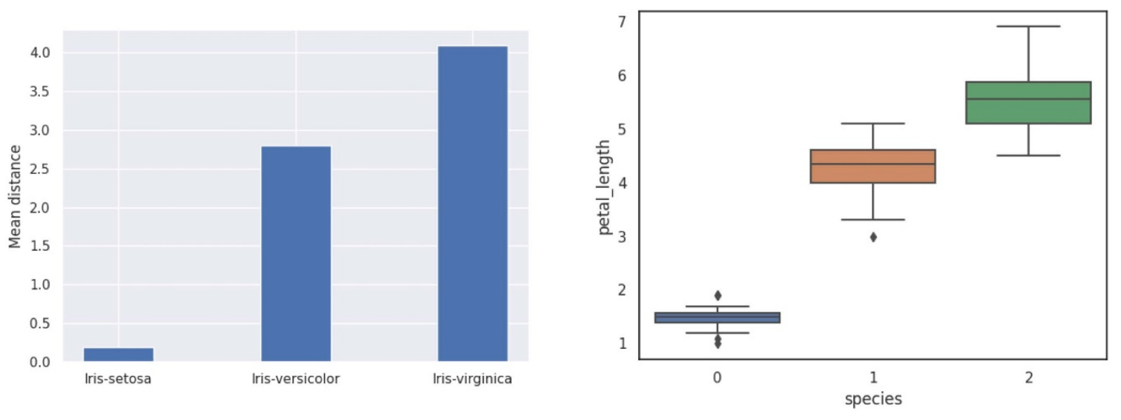
Iris dataset	Architecture 1	Architecture 2	Architecture 3	Architecture 4
2-norm of the first layer	6,40	7,92	4,86	6,69
2-norm of the hidden layer	11,10	16,89	7,45	—
2-norm of the output layer	2,88	3,02	2,69	4,14
Constant of the algorithm	204,60	403,98	80,70	27,70

Furthermore, the morphisms i, i' from definition 3.11 are the identity of the euclidean spaces and the morphism $(\mathbb{R}^3, d_C) \to (\mathbb{R}^3, d_e)$ that sends each element to itself. The fact that this is continuous follows from the continuity of the pseudodistance function.

One key difference with the Boston dataset is that, in this case, the architecture seems to have an effect on the distances even though the constants for the algorithms are considerably higher than the quotient of distances. This is due to the difference in the task: in the Boston dataset case, the task was the prediction of the price. This means that the exact number was the desired output. Whereas on the Iris dataset, the task is the classification of the species and the desired output is an index with the best certitude possible. This implies that the precise values of the output are more susceptible to the architecture of the learner. Indeed, the desired output is not a precise point in $\mathbb{R}^3$ but a vector with one of its entries significantly higher than the others. Thus, the specific value of its entries is going to be affected by the architecture of the algorithm in a way that makes the differences among the entries of the vector higher. As we can see by the mean distances of the Iris dataset, this is indeed affected by the architecture since both the mean distances of the output increase. The effect of this phenomenon can be seen in figure 3, where both the quotient and the maximum mean distances increased when we doubled the neurons of the hidden layer implying that the information of tables 4 and 5 seems to be a better descriptor of the algorithm's behavior. This would be analogous to the algorithm 'overfitting' the data. Notice that, since the Lipschitz constant of the softmax and logarithm functions are 1 we can compare directly the quotient of the distance with the Lipschitz constant which acts as a worst-case scenario for the quotient.

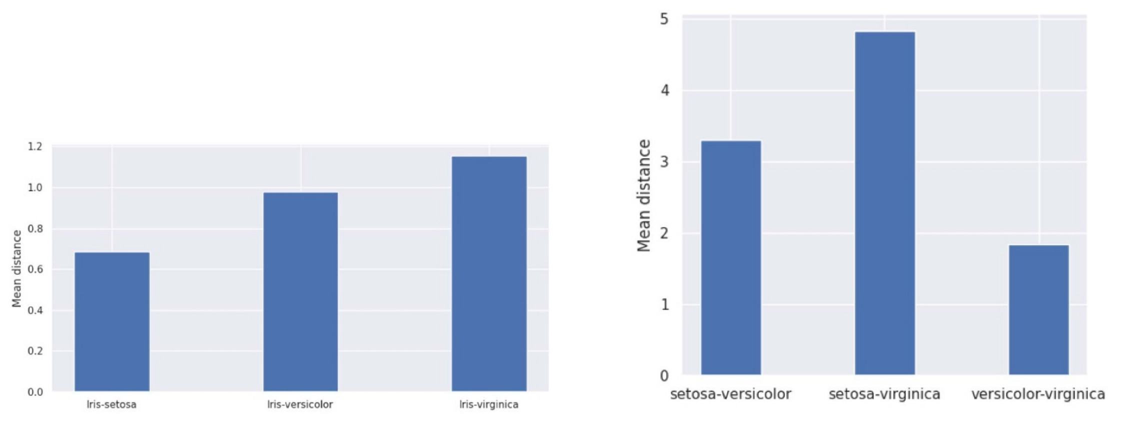
We would like to emphasize that the accuracy metrics are independent of the interpretation as explained in remark 3.20. Indeed, the algorithms would achieve the same accuracy with or without the framework since it is designed to explain and interpret not to optimize the algorithm.

If we restrict our attention to the dataset and consider the product category $\mathcal{V}^4$ as the category of enrichment of $\mathcal{I}ris$ we can compute the vectorial mean distance among elements of each class. Since we want



(a) Mean distance of the attribute petal length. (b) Box plot of petal length per class.

Figure 4. Vectorial mean distances of the petal length coordinate per class for the Iris dataset.



(a) Scalar mean distance for each class. (b) Scalar mean distances between classes.

Figure 5. Scalar mean distances per class for the Iris dataset.

to aggregate the results by species we use histograms to represent the information. We see in figure 4 that it yields the same information as the boxplot of the petal lengths. Thus, looking at the enriched version of the functor h^c over $\mathcal{V}^4$ for every element of the dataset yields interesting information that is equivalent to the statistical data of this dataset. Each of the bars of figure 4(a) corresponds to the mean dispersion of the petal lengths for each variety of flower that we can see in the boxplot. Thus, the mean distance of the attribute petal length corresponds to the average petal length difference for each class. This (roughly) corresponds to the amplitude of the boxes for each class and can be deduced from equation (35).

Furthermore, if we look at the scalar mean distances between classes $m_{cc'}$ (equation (37)), we see in figure 5, that the three species differ in how scattered the flowers in each species are, that is figure 5(a), and how separated they are from the other species, that is figure 5(b). This, in turn, partially corresponds to figure 6, where we can see not only how scattered the data points of each species are, but also how far apart from one another they are when we look at the projection to the coordinates representing the attributes sepal length and sepal width. Combining the results from figures 5 and 6 we see that the clusters of species also get further apart from each other as they pass through the algorithm. This is due to the fact that one wishes to differentiate each species clearly, so they get separated to emphasize the distinction. Finally, it is worth noting that having the vectorial mean distances for each species would lead to the reconstruction of the relative positions of the clusters of species since one would have a system of equations that can be solved up to parameters (which would yield the exact position, not just relative) to have the general distribution in space of the classes.

As we have seen, using the categorical approach, and by changing the category of enrichment if necessary, we can analyze the different properties of the datasets and keep track of their variation as they pass through the algorithm. Since the theory applied is abstract, the same strategy could be applied to other problems with

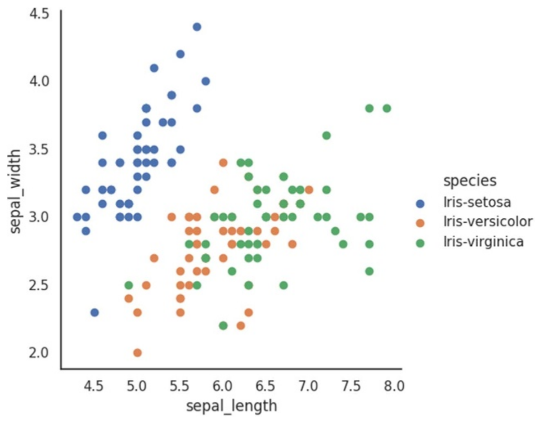


Figure 6. Scatter plot for the Iris dataset.

other datasets by repeating the steps of the manuscript: first, define the category of the dataset. Second, analyze the properties of interest and define the categories of enrichment $\mathcal{V}$ and then study their variation through the algorithm.

4.3. Weather forecast

Weather forecasting using ML is the application of ML algorithms and techniques to predict future weather conditions based on historical data and patterns. It involves analyzing vast amounts of meteorological data, such as temperature, humidity, wind speed, and atmospheric pressure, to generate accurate forecasts for various time frames. In our case, we are going to focus our attention on weather forecasting of India-specific regions[8]. This dataset provides the data from 1st May 2016 to 11 March 2018 of the specific city i.e. Jaipur in India in the form of a CSV file. We will use a neural network to predict the minimum, maximum, and mean temperature of a given day. We have chosen this dataset because it allows us to predict a vector, not a single variable. Thus, we have two different enrichments on the output space yielding different information.

More specifically, since the input space is similar to the input space of the Boston dataset, we can consider similar categorical structures: the objects of the category are vectors on a $\mathbb{R}$ vector space. The arrows are transformations from one vector to another. This category can be enriched either over $\mathbb{R}$ or over $\mathbb{R}^d$, with d the dimension of the $\mathbb{R}$ vector space. Analogously, since the task of the algorithm is the prediction of specific values, the output space can be modeled as a category whose objects are vectors in $\mathbb{R}^3$, and arrows are transformations from one vector to another. This category can be enriched either over $\mathbb{R}$ or over $\mathbb{R}^3$ turning it into a generalized metric space. Each enrichment yields different information: the enrichment over $\mathbb{R}$ measures the dispersion of the variables of temperature whereas the enrichment over $\mathbb{R}^3$ measures the dispersion of each of the temperature variables separately via the metrics defined in definition 3.8.

This specifies the possible metric spaces from definition 3.21. Now we focus on defining the neural network. In this case, we have chosen a neural network with one hidden layer of eight neurons. Thus, all the computations of the Lipschitz constant of the algorithm carry over from the previous examples. We then focus our attention on the effect the algorithm has on the distances.

Figure 7 shows the difference between the mean distance of the data vectors and the mean distance of the predicted temperature vectors. This is the result of enriching the output space over $\mathbb{R}$. Here there are two things to note: the first is that the distances are much bigger on the input space, this is due to the high dimensionality of the vectors (as in the Boston dataset). The second is that the variation of the distances is almost negligible. This makes sense since the specific region chosen for the forecasting has a tropical climate which implies some degree of stability of the mean temperature. If we concentrate on the mean distances of the three predicted values we have a zoomed-in picture of the effect of the algorithm.

Figure 8 shows the mean distance of the predicted values and the normalized mean distance of the input at the bottom (in cyan) which is the mean distance of the input normalized by the size of the vectors. This is the result of enriching the output space over $\mathbb{R}^3$. Here we can appreciate that the predicted maximum temperature has both more dispersion and more amplitude than the other values. Furthermore, the mean temperature seems to have the least dispersion of the three. This is consistent with the fact that the chosen

[8] The concrete dataset can be found in www.kaggle.com/datasets/rajatdey/jaipur-weather-forecasting?resource = download.

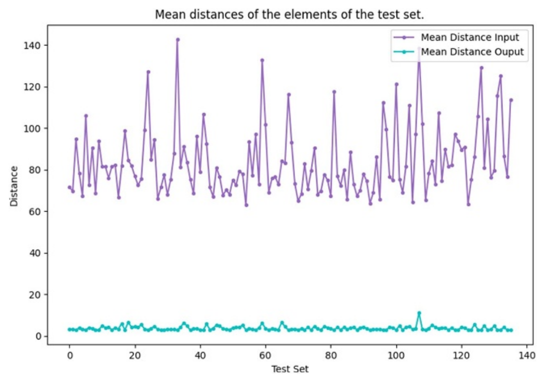


Figure 7. Mean distances of the test set.

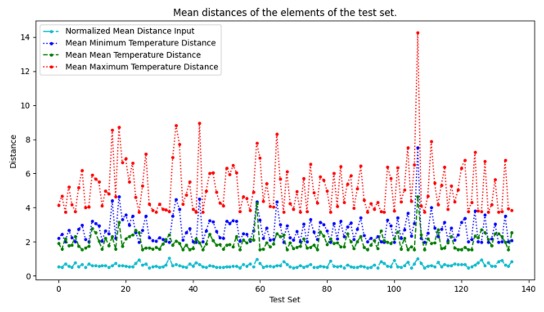


Figure 8. Mean distances of the minimum, maximum, and mean temperatures.

region is of tropical weather. Finally, it is worth mentioning that all of the predicted values have more dispersion and amplitude than the normalized mean distance input.

We can conclude that enriching over $\mathbb{R}$ (figure 7) yields information on how dispersed the output space is whereas enriching over $\mathbb{R}^3$ yields a more fine-tuned metric: how dispersed each of the predicted value is and how does it compare with the average dispersion of an attribute of the input set. Furthermore, as expressed in remark 3.20, this added structure does not affect the accuracy of the model.

Category theory allowed the unifying of the concepts of linear isomorphism, homeomorphism, and diffeomorphism as the concept of isomorphism in the categories of vector spaces, topological spaces, and differentiable manifolds respectively. In our case, it has allowed us to unify different metrics under the same concept given by the enrichment of the categories involved. As this framework offers a concise structural analysis of datasets, algorithms, and their mathematical interactions, it enables theoretical proofs without uncertainty in the provided explanations. Furthermore, the incorporation of established metrics within this framework reinforces our argument that category theory serves as a suitable language for modeling explainability. By providing precise yet comprehensive explanations, we can encompass various phenomena simultaneously. This approach allows us to derive a unified explanation from multiple perspectives, as demonstrated in this section, where we consider diverse metrics and enrichments on various datasets.

5. Conclusions

In this work, we have proposed how to monitor the relationship between the input dataset points and the output of the learning algorithm by employing the enriched version of the Yoneda embedding. This was achieved through the unification of the approaches proposed in [FST19, BTV22].

Decorating the hom-categories of learners (morphisms in the category Learn) over Lipschitz functions has allowed us to keep track of the distances between data points as a proxy of how relative information, what is similar or what is very different, propagates through the network. Moreover, we have observed a bound on the proportionality of the distances. This framework could help interpret the results and give a reference of what is the 'normal' behavior of the algorithm and when we are dealing with an outlier. This was accomplished by enriching the category of the data to create a metric space, and the category of learners to keep track of the information given to us by the enriched structure.

The first thing was to enrich the category of the dataset in order to reflect the structure of the data. This was done throughout section 3. With this, we encoded structural information (relation between objects) of the dataset. The different enriching categories served as a way to gather different kinds of information. Indeed, enriching over $[0,+\infty]$ yielded a metric of how similar or dissimilar two elements of the dataset were. In the case of the Iris dataset, enriching over $[0,+\infty]^4$ gave us dispersion metrics of the attributes of the dataset, thus having a hybrid picture of algebra (structure) and statistics (dispersion). Then we investigated how those metrics were affected by the algorithms.

In particular, definition 3.21 allows us to see neural networks as Lipschitz functions between metric spaces. Thus, it allows us to give a metric of how much the algorithm is going to exaggerate what it saw on the dataset. Furthermore, this metric depends continuously on the choice of parameters as seen in section 3. Lastly, the examples explored in section 4 give us another variable that will affect how much the algorithm will amplify the differences observed in the dataset: the number and size of the layers of the neural network. This is the other side of the coin of the phenomenon known as overfitting the data: it exaggerates everything to the point of unambiguity which leads to an excessively rigid model.

6. Future work

As we have seen throughout section 4 the bounds obtained via the Lipschitz constant are not strict since the quotient of distances fell significantly below. This can be explained by the effect of the ReLU functions that are shutting down some of the results. Running again the algorithms without the activation functions, thus having a linear model, yields bigger distance gaps but smaller weight matrix norms. Another thing to take into account is the effect of the pseudodistance in the Lipschitz constant associated with specific parameters of the algorithm, this is a possible line of future work.

A note on the Yoneda embedding in the enriched category setting: having our dataset be a category enriched over $\mathbb{R}_+$ as a means to encode distances has allowed us to view our dataset as points in a metric space. Then, decorating Learn over $\mathcal{L}$ has allowed us to keep track of the variation of distances and put an upper bound on it. Now, the space $\mathcal{L}(X,Y)$ of Lipschitz functions between bounded metric spaces can be viewed as a normed, and hence metric, space itself. This has been explored in [vLB04]. Thus, one could define the notion of 'close' learners by using its Lipschitz functions turning the category of learners into a metric or pseudometric space.

The other thing to consider is the distance itself. We chose the Iris and Boston datasets because they already gave us data as numerical values. Furthermore, the datasets consist of separated attributes. This made taking the distance the obvious choice for the enriched version of the hom-sets. However, applying this reasoning to more complex datasets, images for example, will fail since the data is much more intricate and taking the distance (pixel by pixel) loses too much information about the images. However, the same methodology can be readily applied to NLP problems where similitude metrics are a good estimator of distances. The same can be said about voice recognition: since the raw data is a wave that gets transformed into a vector of points, the algorithms presented in this manuscript capture all the necessary information. The thing is that how we define the morphisms in the category of the dataset and what we enrich it plays a fundamental role in the usefulness of the approach.

Additionally, since the categorical scheme is able to unify several explainability methods, another line of future work would be to make a more detailed comparison of these methods and they can be transcribed into this new language. This would allow the use of different kinds of explainability methods simultaneously and seamlessly.

Finally, we have used the enriched versions of the functors h^c to get statistical information about the dataset. In the Iris dataset, we have checked that the information given by, a combination of, those functors corresponds to the statistical information of the dataset. The important thing to notice is that the functors h^c can be computed in any category. Thus, having an appropriate category of enrichment would yield a generalization of the statistical analysis to datasets where it cannot be done so easily or cannot be done automatically. This can be understood as functors from the category of $[0,+\infty]^n$ enriched categories to the category of pseudometric spaces $\mathcal{PM}$:

$$\mathcal{M}_n\colon [0,+\infty]^n - \mathcal{C}at \to \mathcal{PM}. \tag{58}$$

Thus, the same setting can be applied to more general and complex cases to extract information that may not be apparent with other mathematical tools. One thing that can be done to implement this approach in more complex contexts is to consider learners between categories such that the following diagram commutes

$$\begin{tikzcd}
\{\text{Enriched category over } S\} \arrow[r, "F"'] \arrow[d, "h^X"] & \{\text{Enriched category over } R\} \arrow[d, "h^V"] \\
S \arrow[r, "f"] & R
\end{tikzcd} \tag{59}$$

where F is a 'functorial learner', h^X, h^V are Yoneda embeddings, and $f\colon\ S \to R$ is a map that keeps track of how the relative information between two objects (in our examples, the distances) evolve through the learning algorithm. That is, to consider the category of learners as the category of copresheaves which, theoretically, would help with both explainability and interpretability (see [MM12]). For instance, one can consider a metric space M to be a category $\mathcal{M}$ where the objects are the points of M and the morphisms are equivalence classes of paths connecting two points (if they exist) and composition is given by the composition of paths. This would work as long as each connected component of M is simply connected since the identity morphism for any point x_0 would be the fundamental group $\pi_1(M, x_0)$ which is trivial if and only if the space is simply connected. This category can be enriched over $\mathbb{R}_+$ by assigning to every morphism the infimum of the lengths of equivalent paths. This would yield a diagram of the form

$$\begin{tikzcd}
\mathcal{M} \arrow[r, "F"'] \arrow[d, "h^X"] & \{\text{Enriched category over } R\} \arrow[d, "h^V"] \\
\mathbb{R}_+ \arrow[r, "f"] & R
\end{tikzcd} \tag{60}$$

This approach can also be applied to more complex datasets than the ones we have studied in this paper using the categorical language and adapting the dissimilarity metrics. One example of this can be found in [SY21]. One thing to note about [SY21] is that the authors assume we have a vector representation of a certain category $\mathcal{C}$. The downside of this approach is that one does not have a way to check how biased the vector representation is. With that in mind, one could apply this method to the semantic dataset and have metrics of how biased, or how much bias is tolerated by, the algorithms. This is another line of future work.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://archive.ics.uci.edu/ml/datasets/iris.

Acknowledgments

This work is partially supported by the TAILOR project, a project funded by the EU Horizon 2020 research and innovation programme under GA No. 952215, and by the GUARDIA project, a project funded by Generalitat Valenciana GVA-CEICE Project PROMETEO/2018/002

Appendix. Enriched categories and bicategories

We give the definitions of bicategory, monoidal category, and enriched category. We prioritize the core concepts, setting aside the detailed presentation of the properties. Nevertheless, we have indicated where a concise statement of the properties can be found.

Definition A.1. A **bicategory** consists of the following data:

(1) A collection of objects also called **0-cells**.
(2) For every pair of objects A and B, a category $\mathcal{C}(A, B)$ whose objects are called *one-morphisms or one-cells* from A to B and whose morphisms are called *two-morphisms*.
(3) For every object A, a distinguished one-cell 1_A from A to A called identity one-cell.
(4) For every triple of objects A, B, and C, a composition functor

$$\circ_{A,B,C} : \mathcal{C}(B, C) \times \mathcal{C}(A, B) \to \mathcal{C}(A, C)$$

that assigns to each pair of one-morphisms $f \in \mathcal{C}(B, C)$ and $g \in \mathcal{C}(A, B)$ a one-morphism $f \circ g \in \mathcal{C}(A, C)$, and to each two-morphism $\alpha : f \Rightarrow f'$ and $\beta : g \Rightarrow g'$, a two-morphism $\alpha \circ \beta : f \circ g \Rightarrow f' \circ g'$, satisfying the usual associativity and identity axioms for composition of morphisms in categories.

(5) For every one-morphism $f: A \to B$, a two-morphism $1_f: 1_B \circ f \Rightarrow f \circ 1_A$ (unit isomorphism).
(6) For every triple of objects A, B, and C, and every pair of one-morphisms $f: A \to B$ and $g: B \to C$, a two-morphism $\alpha_{f,g}: (g \circ f) \circ 1_A \Rightarrow g \circ (f \circ 1_A)$ (associativity isomorphism).

These data must satisfy coherence axioms, including associativity and unit axioms for two-morphisms and the interchange law, which governs the interaction between two-morphisms.

Definition A.2. A **monoidal category** $(\mathcal{C}, \otimes)$ is a category $\mathcal{C}$ equipped with a functor

$$\otimes: \mathcal{C} \times \mathcal{C} \to \mathcal{C} \tag{A.1}$$

out of the product category of $\mathcal{C}$ with itself, called the **tensor product (or monoidal product)**. There is a distinguished object I of $\mathcal{C}$ called the **monoidal unit**. There are natural isomorphisms called **associator, left and right unitors** that satisfy the triangle and pentagon identities[9].

Definition A.3. Given a monoidal category $(\mathcal{V}, \otimes)$. An **enriched category** consists of the following data:

(1) A collection of objects.
(2) For every pair of objects A and B, an object in $\mathcal{V}\,\mathcal{C}(A,B)$ where the elements are called *enriched morphisms* or *arrows*.
(3) For every object A, an identity arrow 1_A in $\mathcal{C}(A,A)$.
(4) For every triple of objects A, B, and C, a composition operation

$$\circ_{A,B,C}: \mathcal{C}(B,C) \times \mathcal{C}(A,B) \to \mathcal{C}(A,C)$$

that assigns to each pair of enriched morphisms $f \in \mathcal{C}(B,C)$ and $g \in \mathcal{C}(A,B)$ an enriched morphism $f \circ g \in \mathcal{C}(A,C)$.

These data must satisfy the so-called triangle and pentagon identities.

The category Cat can be made into a monoidal category by taking the monoidal product of two categories $\mathcal{C}$ and $\mathcal{D}$ to be the product category. This, together with the fact that Cat is a 2-category[10], turns Cat into a monoidal category.

Thus, by definition A.3, a bicategory can be seen as a category enriched over (the monoidal category) Cat. Indeed, objects in the enriched setting correspond to 0-cells in the bicategory. For any two objects A, b in $\mathcal{C}$, the hom-space is a category $\mathcal{C}(A,B)$. The identity, associator, and unitors conditions on the enriched setting follow from the analogous properties of definition A.1.

ORCID iDs

Ares Fabregat-Hernández https://orcid.org/0009-0005-5104-514X
Vicent Botti https://orcid.org/0000-0002-6507-2756

References

[ABFR23] Althaus E, Merlin Bumpus B, Fairbanks J, and Rosiak D 2023 Compositional algorithms on compositional data: deciding sheaves on presheaves (arXiv:2302.05575)
[AM21] Avant T and Morgansen K A 2021 Analytical bounds on the local lipschitz constants of relu networks (arXiv:2104.14672)
[BB21] Belfiore J-C and Bennequin D 2021 Topos and stacks of deep neural networks (arXiv:2106.14587)
[BBCV21] Bronstein M M, Bruna J, Cohen T and Veličković P 2021 Geometric deep learning: grids, groups, graphs, geodesics, and gauges (arXiv:2104.13478)
[BCV21] Baez J C, Courser K and Vasilakopoulou C 2021 Structured versus decorated cospans (arXiv:2101.09363)
[BTV22] Bradley T-D, Terilla J and Vlassopoulos Y 2022 An enriched category theory of language: from syntax to semantics *La Mat.* **1** 1–30
[CAC+22] Caliskan A, Parth Ajay P, Charlesworth T, Wolfe R and Banaji M R 2022 Gender bias in word embeddings: a comprehensive analysis of frequency, syntax, and semantics (arXiv:2206.03390)
[CMJG19] Carter B, Mueller J, Jain S and Gifford D 2019 What made you do this? Understanding black-box decisions with sufficient input subsets *The 22nd Int. Conf. on Artificial Intelligence and Statistics* (PMLR) pp 567–76
[FDC13] Fiore M and Devesas Campos M 2013 The algebra of directed acyclic graphs *Computation, Logic, Games and Quantum Foundations. The Many Facets of Samson Abramsky* (Springer) pp 37–51
[Fon15] Fong B 2015 Decorated cospans *Theory Appl. Categories* **30** 1096–120

[9] The precise definition and properties can be found in https://ncatlab.org/nlab/show/monoidal+category.
[10] See definition https://ncatlab.org/nlab/show/2-category.

[FST19] Fong B, Spivak D and Tuyéras R 2019 Backprop as functor: a compositional perspective on supervised learning *2019 34th Annual ACM/IEEE Symp. on Logic in Computer Science (LICS)* (IEEE) pp 1–13

[GP17] Gao B and Pavel L 2017 On the properties of the softmax function with application in game theory and reinforcement learning (arXiv:1704.00805)

[HKM+18] Hsieh Y-P, Kao Y-C, Karimi Mahabadi R K, Yurtsever A, Kyrillidis A and Cevher V 2018 A non-Euclidean gradient descent framework for non-convex matrix factorization *IEEE Trans. Signal Process.* **66** 5917–26

[HSH+23] Hanks T, She B, Hale M, Patterson E, Klawonn M and Fairbanks J 2023 A compositional framework for convex model predictive control (arXiv:2305.03820)

[Kel82] Kelly M 1982 *Basic Concepts of Enriched Category Theory* vol 64 (CUP Archive)

[Law73] William Lawvere F 1973 Metric spaces, generalized logic and closed categories *Rendiconti del Semin. Mat. e Fis. di Milano* **43** 135–66

[Mil19] Milewski B 2019 *Category Theory for Programmers* (Bartosz Milewski)

[MM12] MacLane S and Moerdijk I 2012 *Sheaves in Geometry and Logic: A First Introduction to Topos Theory* (Springer)

[MMS+21] Mehrabi N, Morstatter F, Saxena N, Lerman K and Galstyan A 2021 A survey on bias and fairness in machine learning *ACM Comput. Surv.* **54** 1–35

[Rie14] Riehl E 2014 *Categorical Homotopy Theory* vol 24 (Cambridge University Press)

[Rie17] Riehl E 2017 *Category Theory in Context* (Courier Dover Publications)

[SGW21] Shiebler D, Gavranović B and Wilson P 2021 Category theory in machine learning (arXiv:2106.07032)

[Sha48] Elwood Shannon C E 1948 A mathematical theory of communication *Bell Syst. Tech. J.* **27** 379–423

[SY21] Sheshmani A and You Y-Z 2021 Categorical representation learning: morphism is all you need *Mach. Learn.: Sci. Technol.* **3** 015016

[TG20] Tjoa E and Guan C 2020 A survey on explainable artificial intelligence (XAI): toward medical XAI *IEEE Trans. Neural Netw. Learn. Syst.* **32** 4793–813

[vLB04] von Luxburg U and Bousquet O 2004 Distance-based classification with lipschitz functions *J. Mach. Learn. Res.* **5** 669–95