Mathematical Structures in Computer (2026), 1–1
doi:10.1017/placeholder

CAMBRIDGE
UNIVERSITY PRESS

ARTICLE

Functorial Gradient-Based Learning

Takumu Nakamura, Kazuyuki Asada, andKeisuke Nakano

Graduate School of Information Sciences, Tohoku University
nakamura.takumu.q8@dc.tohoku.ac.jp
Research Institute of Electrical Communication Tohoku University
asada@tohoku.ac.jp, ksk@riec.tohoku.ac.jp

(Received xx xxx xxxx; revised xx xxx xxxx; accepted xx xxx xxxx)

Abstract
Two prior studies apply category theory to the construction of gradient-based learning algorithms, both using parametrized lenses to model learning. The construction of Fong et al. provides functoriality, whereas that of Cruttwell et al. offers modularity and greater generality. In this paper, we modularize and generalize the functorial construction of Fong et al. by adopting the modular and generic principles underlying the approach by Cruttwell et al. This generalization allows us to accommodate optimizers and loss functions used by Cruttwell et al., which were not supported in the original functorial setting. We also present a modular proof method for establishing a consistency property of our learning algorithms, namely the GetPut law as lenses, which also serves as an application of our modularization. Moreover, the modular design enables us to readily adapt our framework to handle a loss function for classification tasks, namely softmax cross-entropy.

1. Introduction

Gradient-based learning algorithms—exemplified by the backpropagation training of neural networks and other modern differentiable models—have achieved remarkable success, but their complex mechanics can be difficult to analyze and compose in a principled way. Among the possible approaches, category theory offers a high-level, compositional perspective on such learning algorithms, aiming to reveal their structure and improve our understanding of their behavior.

Fong et al. (2019) and Cruttwell et al. (2022) provided categorical frameworks for the same computations of gradient-based learning algorithms from different perspectives. The two propose different ways to construct a learning algorithm from an architecture. We call the construction/algorithm of Fong et al. (2019) the *FST construction/algorithm*, and that of Cruttwell et al. (2022) the *CGGWZ construction/algorithm*.

The main contribution of this paper is an integration of the two constructions. We first compare them in terms of the common features and the differences, and then explain how they can be integrated.

1.1 Common features: para construction and lens

Both constructions have the following points in common: *para construction* (Fong et al., 2019; Gavranović, 2019) and *lens* (Foster et al., 2007; Riley, 2018). We discuss each in turn.

The para construction is a simple categorical mechanism for adjoining parameters to morphisms, as illustrated in fig. 1, thereby yielding the notion of a *parametrized function*, or an

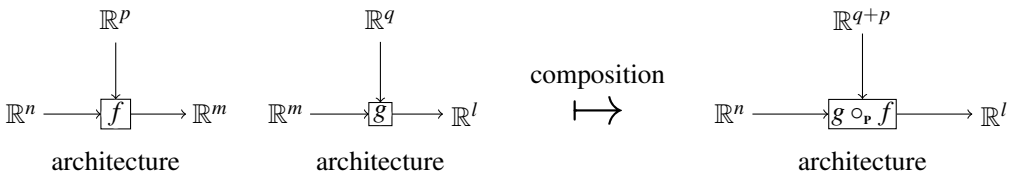


Figure 1. Architectures and their composition. For the left architecture (the function $f : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^m$), the lines labeled $\mathbb{R}^p, \mathbb{R}^n, \mathbb{R}^m$ indicate the flows of parameter values, input values, and output (prediction) values, respectively.

architecture in machine learning. This allows complex training pipelines to be composed in a uniform and modular way, by treating parameters as special inputs and carrying them through compositions.

Lenses have been used as an abstract framework for bidirectional transformations. A lens represents a computation (function) equipped with another computation in the opposite direction, and hence it can be used to express abstract backpropagation, which is an efficient calculation method widely used in machine learning. Lenses parametrized by the para construction are called *para lenses*.

1.2 Differences: generality, modularity, and functoriality.

One clear advantage of the CGGWZ construction is its generality on the base category. The FST construction uses the concrete category of Euclidean spaces $\mathbb{R}^n$ and smooth functions as a base category, while in the CGGWZ construction a base category can be an arbitrary *cartesian reverse differential category*, an abstract categorical structure that captures the essential properties of reverse-mode automatic differentiation. We adopt this level of generality in the present paper.

Another main difference between the two constructions is their approach to modularization. In the CGGWZ construction, a learning algorithm is represented as a para lens, which is constructed by four lenses, as shown on the left in fig. 2. Moreover, the four module lenses are constructed from the following, respectively:

- (arch) an architecture with the calculation of backpropagation,
- (opti) an optimizer (such as gradient descent),
- (lr) a learning rate, and
- (loss) a calculation of loss gradient.

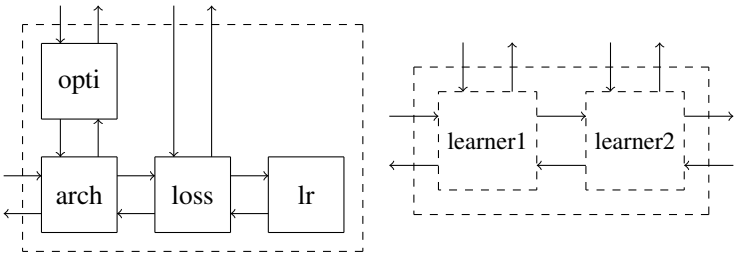


Figure 2. Modularizations of CGGWZ algorithms (left) and FST algorithms (right). All boxes represent lenses. Among them, a box drawn with a dotted outline indicates a lens that can be regarded as a learning algorithm, fully capturing the computation involved in gradient-based learning.

This construction is faithful to a traditional and practical approach. On the other hand, in the FST construction, although it also uses all of the four notions above, they are not represented as lenses, and the optimizer function is not parametric but fixed to gradient descent, while the loss function is parametric with some constraint.

In the FST approach, the notion of a lens is used to describe each learning algorithm itself. Specifically, they first provided an abstract (composable) learning algorithm called a *learner* as a para lens, and then showed a concrete construction of learners, which perform gradient-based learning. The prominent aspect of the notion of a learner is that it can backpropagate not only gradients but also desired outputs, by which two learners can be composed.

Thus, one benefit of the learner framework is that we can construct a complicated learner not only from an architecture but also as a composition of existing simpler learners, as shown on the right in fig. 2. Therefore, we can regard learners themselves as modules of a bigger learner. This modularization is completely different from that in the CGGWZ construction, where modules, the four lenses above, are not learning algorithms but more fundamental elements from which we can construct a learning algorithm.

The other main difference is the *compositionality* of the construction. Namely, the FST construction is (monoidally) *functorial*, which is denoted by the following equation:

$$\overline{\mathrm{Lf}}(g \circ_{\mathbf{P}} f) = \overline{\mathrm{Lf}}(g) \circ_{\mathbf{P}} \overline{\mathrm{Lf}}(f)$$

where $\overline{\mathrm{Lf}}(-)$ is the FST construction from an architecture to a learner, $\circ_{\mathbf{P}}$ on the left-hand side is the composition of architectures in fig. 1, and $\circ_{\mathbf{P}}$ on the right-hand side is the composition of learners in fig. 2. Here, functoriality means compositionality realized by a single functor from architectures to learners, as witnessed by the equation above. Given a desired output for $\overline{\mathrm{Lf}}(g \circ_{\mathbf{P}} f)$ and $\overline{\mathrm{Lf}}(g)$, the learner $\overline{\mathrm{Lf}}(g)$ can backpropagate the output, by which $\overline{\mathrm{Lf}}(f)$ can train further. The functoriality ensures that the two ways of training yield identical results, and that learners coherently subsume architectures. In connection with the modularity discussed in the previous paragraph, we also note that learner composition is strictly more expressive: $\overline{\mathrm{Lf}}(g)$ or $\overline{\mathrm{Lf}}(f)$ can be replaced with any learner L, even if L does not arise from the FST construction.

When we consider the functoriality equation above also for the CGGWZ construction, not only does it fail to hold, but the right-hand side does not even make sense as a learning algorithm. In other words, the interpretation of the CGGWZ para lenses as learning algorithms differs from that of learners, and does not respect the composition of para lenses. However, the CGGWZ construction can be reorganised so that it does respect composition, and our integrated construction below can also be viewed as being obtained in this way.

1.3 The integration.

The modularity in the CGGWZ construction plays a role in defining the construction itself, whereas in the FST construction the modularity arises from the property of the construction (namely, functoriality), based on the composability of learners. In the present paper, we mainly follow the CGGWZ-style modularity, and the FST-style modularity serves as our goal.

Our central contribution is to obtain the generalization and the CGGWZ-style modularization of the functorial FST construction by adopting the generic and modular principles underlying the CGGWZ construction. As a result of the modularization, we can use various optimizers used in CGGWZ construction but not in the FST construction. Also, we relax the constraint on a loss function for the functoriality, so that we can allow more loss functions (such as Ex. 44).

For the notion of lens, there is the most basic consistency law called the GetPut law. For learners, this law indicates a kind of convergence of training, which was first discussed by Fong and Johnson (2019). Proving this property for our learners should serve as one natural sanity check. As an application of our CGGWZ-style modularization, we demonstrate how the modularization helps to prove properties of learners, such as the GetPut law, in a modular way. Specifically, we

prove that the GetPut law for the learners obtained by our construction reduces to certain simple properties of the module lenses associated with the loss function and the optimizer, and then we examine whether each of the loss functions and the optimizers in this paper satisfies the reduced property.

Lastly, we extend our functorial framework so that it can handle (multi-class) classification tasks, where (softmax) cross-entropy loss is commonly used. The functorial construction thus far is mainly for regression tasks, where inputs and outputs are both vectors, while, for classification tasks, outputs are not general vectors but probability distributions (that are represented as vectors). Then we find that it is difficult to maintain both the functoriality and the consistency law GetPut (fig. 4), and thus, instead of functoriality, we develop a "jointly-functorial" construction of the following form:

$$\overline{\mathrm{Gb}}^{*}(g \circ_{\mathbf{P}} f) = \overline{\mathrm{Gb}}^{*}(g) \circ_{\mathbf{P}} \overline{\mathrm{Gb}}(f), \tag{1}$$

where $\overline{\mathrm{Gb}}$ and $\overline{\mathrm{Gb}}^{*}$ are the constructions for regression tasks and for classification tasks, respectively. Technically, this extension is a simple modification, but the simplicity is due to our CGGWZ-style modularization, and further extensions to other tasks are expected to be promising.

While concrete applications of functoriality are left for future work, we believe that functoriality itself already provides an important theoretical understanding of the mechanisms and behaviour of gradient-based learning, as the construction subsumes a broad range of modern gradient-based learning methods.

In setting up our functorial construction, we must also address a technical well-definedness issue arising from the interaction between the para construction and optimizers. Cruttwell et al. (2022) observed that the FST construction (Fong et al., 2019) is not well-defined with respect to the equivalence relation used there. They avoided this issue by assuming the underlying symmetric monoidal category to be strict, but this strictness assumption is not sufficient for the functoriality pursued by Fong et al. and in the present paper. We therefore use a modified para construction based on a finer equivalence relation.

1.4 Contribution

Our main contributions can be summarized as follows.

- We define the modularized FST construction (Def. 55), through the categorical generalization and the CGGWZ-style modularization of the FST construction. This is primarily aimed at regression tasks.
  - The base category **Smooth** of the construction of Fong et al. (2019) is generalized to an arbitrary cartesian reverse differential category.
  - As a technical foundation for the well-definedness of the construction, we use a modified para construction based on a finer equivalence relation (Def. 17; Rmk. 50 and Rmk. 53).
  - The modularization of the learner construction involves the following generalizations of its components:
    - The gradient-descent optimizer of Fong et al. (2019) is generalized and parametrized as a module, subsuming three other optimizers (Thm. 52).
    - The constraint on loss functions required for the functoriality is relaxed by axiomatizing loss computation as a loss lens, which enables an additional concrete example (Ex. 44).
  - The modularization also facilitates reasoning about learner properties:
    - The mechanism for the functoriality—especially, the constraints on the loss function and the optimizer—is clarified (Def. 40, Def. 46).
    - The well-definedness of the learner construction is explained by a kind of naturality of an optimizer. (Def. 46).

  - The GetPut law for learners is reduced to simple properties of the loss lens and the optimizer, yielding a modular proof of the consistency (Thm. 58).
- We adapt the above framework to classification tasks, which accommodates the (softmax) cross-entropy loss (section 6).

Organization of the paper. In section 2, we present preliminaries for the basic learning mechanism of gradient-based learning and category theory used in this paper. In section 3, we introduce the modified notion of para construction, to address the ill-definedness described above. In section 4, we review the two methods that construct gradient-based learning algorithms using category theory: the method of Fong et al. (2019) and that of Cruttwell et al. (2022). Then we also review the notion of the GetPut law. In section 5, we reorganize and simplify FST construction using the methods in CGGWZ construction, which modularizes the functorial FST construction. Then we give the modular proof method of the GetPut law for our learner construction. In section 6, we adapt our learner construction so that we can handle the (softmax) cross-entropy loss function. In section 7, we discuss related work, including further comparison between the present work and the two preceding works.

2. Preliminaries

In this section, we review fundamental concepts in machine learning and category theory.

2.1 Machine learning

This subsection provides a concise review of the fundamental mechanisms of gradient-based learning, which are the targets of our categorical development; see, e.g., (Goodfellow et al., 2016) for details.

We review supervised learning, which is a fundamental framework for learning, along with unsupervised learning and reinforcement learning. In this paper, we focus on gradient-based learning, specifically within the context of supervised learning. We also review two methods commonly used in gradient-based learning. The first is gradient descent, an optimization method that serves as a fundamental component of learning. The second is backpropagation, an algorithm to efficiently compute a gradient, which is used in gradient descent.

Supervised learning (with a parameter) In supervised learning, we have a smooth function (with a parameter) $f : \mathbb{R}^l \times \mathbb{R}^n \to \mathbb{R}^m$ called an *architecture*, and a (set of) *training data* $(a, b) \in \mathbb{R}^n \times \mathbb{R}^m$. A pair of an architecture and a *parameter* $p \in \mathbb{R}^l$ is called a *model*. The goal of supervised learning is to find a good parameter p by updating it from an initial parameter $p_0 \in \mathbb{R}^l$ with a learning algorithm and the training dataset so that we obtain a function $f(p, -) : \mathbb{R}^n \to \mathbb{R}^m$. A training data is a pair consisting of an input (example) a and a desired output (label) b, and we want to predict b from a, i.e., with ideal p we expect $b = f(p, a)$. However, for p that has not trained enough yet, there should be a "loss" between b and $f(p, a)$, and we want the loss to be minimized. Therefore, in order to update a parameter p by a better one p' with training data (a, b), we consider the minimization of the *objective function* $L(p) = e(f(p, a), b)$, where $e : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a fixed function called a *loss function*, which calculates a distance between two arguments (such as the *quadratic error* $e^{\mathrm{QE}}(x, y) = \frac{1}{2}\sum_i (x_i - y_i)^2$). Towards the minimization of L, we update p to p' so that $L(p') < L(p)$. One major method to solve the updating problem is the *gradient descent*.

Gradient descent *Gradient descent* is an example of an *optimizer*, i.e., optimization methods minimizing a value of a function. For any real-valued smooth function $L : \mathbb{R}^l \to \mathbb{R}$, we aim to find a value $p \in \mathbb{R}^l$ that minimizes $L(p)$. Given a value $p \in \mathbb{R}^l$, gradient descent updates it by $p' = \mathrm{Op}(p, \nabla_p L(p)) = p - \varepsilon \nabla_p L(p)$ where (i) $\varepsilon \in \mathbb{R}$ is a constant called a *learning rate* (a.k.a.

a *stepsize*), (ii) $\mathrm{Op}(x, y) = x - \varepsilon y$, and (iii) $\nabla_p L$ is the gradient of L with respect to p, that is, $\nabla_p L = (\frac{\partial L(x)}{\partial x_1}(p), \cdots, \frac{\partial L(x)}{\partial x_l}(p))$.[a] Note that Cruttwell et al. (2022) do not incorporate the learning rate into an optimizer.

Backpropagation *Backpropagation* is one of the methods for computing gradients. The computation uses automatic differentiation based on the chain rule of differentiation. Specifically, suppose that we want to calculate gradients of $c = g(q, f(p, a))$ with respect to q and p. Here, $f : \mathbb{R}^l \times \mathbb{R}^n \to \mathbb{R}^m$, $g : \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}$ are smooth functions, and $a \in \mathbb{R}^n$, $p \in \mathbb{R}^l$, $q \in \mathbb{R}^k$ are arbitrary values. First, we calculate the gradient $\nabla_{(q,f(p,a))} c$ by differentiating g where $(q, f(p, a)) \in \mathbb{R}^k \times \mathbb{R}^m$ is the concatenation of q and $f(p, a)$. These gradients can be divided into two components $\nabla_q c$ and $\nabla_{f(p,a)} c$ because of the definition of the gradients. Next, given $\nabla_{f(p,a)} c$, the gradient $\nabla_{(p,a)} c$ is computed as $(J_f(p, a))^\top \cdot \nabla_{f(p,a)} c$ by the chain rule where $J_f(p, a)$ is the Jacobian of f at p and a. These gradients can also be divided into two components $\nabla_p c$ and $\nabla_a c$. In this way, gradients with respect to parameters $\nabla_p c$, $\nabla_q c$ are calculated in reverse order, compared to the order in which p and q are used in $g(q, f(p, a))$.

Gradient-based learning We call a *(gradient-based learning) algorithm* an architecture f with a loss function e and an optimizer Op, since these components together constitute an algorithm of (gradient-based) supervised learning as above. A loss function and an optimizer can be chosen independently, and other examples than the above ones are given later.

2.2 Category Theory

After reviewing the basics of category theory, we review two categorical components that are fundamental to this paper. A category of lenses is a category whose morphisms are lenses. Lenses are important for abstracting the structure of the backpropagation algorithm. A cartesian reverse differential category is a category that abstracts the differentiation of functions, particularly the chain rule used in backpropagation.

2.2.1 Basics of category theory

We review monoidal categories, which are fundamental algebraic structures in category theory that abstracts the notions of parallel and sequential computations. If readers are not familiar with category theory, see the textbook by Mac Lane (1998) and the paper by Selinger (2011) for details.

Definition 1 (Monoidal category)**.** *A* monoidal category $\mathbf{C}$ *is a category equipped with a monoidal product, associators, a unit object, and unitors, as follows:*

- *a functor* $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ *called the* monoidal product*;*
- *an object* I *called the* unit object*;*
- *for any* $A, B, C \in \mathrm{Obj}(\mathbf{C})$*, a natural isomorphism* $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ *called the* associator*;*
- *for any* $A \in \mathrm{Obj}(\mathbf{C})$*, natural isomorphisms* $\lambda_A : I \otimes A \to A$ *and* $\rho_A : A \otimes I \to A$ *called the* left unitor *and the* right unitor*, respectively.*

The definition of a monoidal category requires coherence axioms*, but we omit them, because the axioms are auxiliary and not important.* ◁

[a] As is usual, $\nabla_p L$ is also written as $\nabla_p L(p)$ where one should easily infer L from "$L(p)$" and "p" syntactically.

A functor $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ enables us to handle parallel computations. Since the monoidal product is a functor, we can take products of objects and morphisms. Thus, the parallel composition of morphisms $f : A \to B$ and $g : C \to D$ is denoted as $f \otimes g : A \otimes C \to B \otimes D$.

The monoidal product has a kind of associativity. Although $(A \otimes B) \otimes C$ is not equal to $A \otimes (B \otimes C)$, there exist an associator $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ as a mutual conversion between them.

The monoidal product also has a kind of unitority. Although $I \otimes A$ and $A \otimes I$ are not equal to A, there exist unitors $\lambda_A : I \otimes A \to A$ and $\rho_A : A \otimes I \to A$ as mutual conversions between them.

Definition 2 (Symmetric monoidal category)**.** *A* symmetric monoidal category $\mathbf{C}$ *is a monoidal category equipped with natural isomorphisms* $\gamma_{A,B} : A \otimes B \to B \otimes A$ *called the* symmetry. *The definition of a symmetric monoidal category also requires some axioms, but we omit them.* ◁

Note that we often omit a subscript of a natural transformation such as $\alpha_{A,B,C}$, and we write α.

Definition 3 (Strict symmetric monoidal category)**.** *A symmetric monoidal category is* strict *if the associators* α *and the unitors* λ *and* ρ *are identities.* ◁

Note that we often omit the natural isomorphisms themselves in some expressions, such as the associators α, the unitors λ and ρ, and the symmetries γ, whether the category is strict or not.

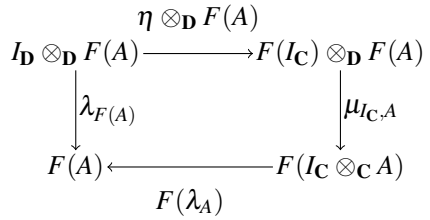
Next, we review strong monoidal functors, which are functors compatible with monoidal products.

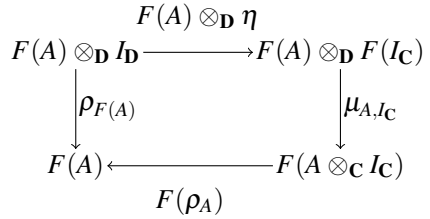
Definition 4 (Strong symmetric monoidal functor)**.** *For any symmetric monoidal categories* $\mathbf{C}$ *and* $\mathbf{D}$*, a* strong symmetric monoidal functor *is a functor* $F : \mathbf{C} \to \mathbf{D}$ *equipped with a morphism* η *and natural isomorphisms* μ*, as follows:*

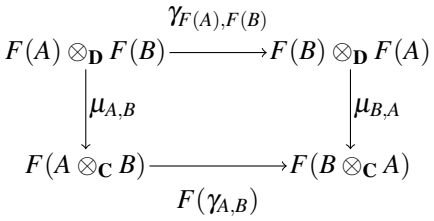
- $\eta : I_{\mathbf{D}} \to F(I_{\mathbf{C}})$ *where* $I_{\mathbf{C}}$ *and* $I_{\mathbf{D}}$ *are the unit objects in* $\mathbf{C}$ *and* $\mathbf{D}$*, respectively;*
- $\mu_{A,B} : FA \otimes_{\mathbf{D}} FB \to F(A \otimes_{\mathbf{C}} B)$ *where* $\otimes_{\mathbf{C}}$ *and* $\otimes_{\mathbf{D}}$ *are the monoidal products in* $\mathbf{C}$ *and* $\mathbf{D}$*, respectively.*

The following diagrams are required to commute.

$$
\begin{array}{ccc}
 & \overset{\alpha_{F(A),F(B),F(C)}}{\longrightarrow} & \\
(F(A) \otimes_{\mathbf{D}} F(B)) \otimes_{\mathbf{D}} F(C) & & F(A) \otimes_{\mathbf{D}} (F(B) \otimes_{\mathbf{D}} F(C)) \\
\downarrow{\scriptstyle \mu_{A,B} \otimes_{\mathbf{D}} F(C)} & & \downarrow{\scriptstyle F(C) \otimes_{\mathbf{D}} \mu_{B,C}} \\
F(A \otimes_{\mathbf{C}} B) \otimes_{\mathbf{D}} F(C) & & F(A) \otimes_{\mathbf{D}} F(B \otimes_{\mathbf{C}} C) \\
\downarrow{\scriptstyle \mu_{A \otimes_{\mathbf{C}} B,C}} & & \downarrow{\scriptstyle \mu_{A,B \otimes_{\mathbf{C}} C}} \\
F((A \otimes_{\mathbf{C}} B) \otimes_{\mathbf{C}} C) & \xrightarrow[F(\alpha_{A,B,C})]{} & F(A \otimes_{\mathbf{C}} (B \otimes_{\mathbf{C}} C))
\end{array}
$$







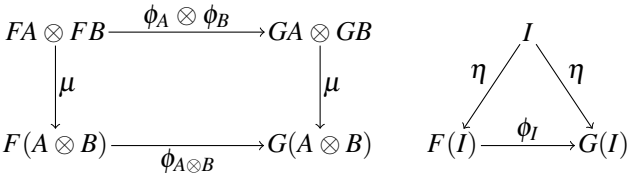
◁

The role of the morphisms $\eta : I_{\mathbf{D}} \to F(I_{\mathbf{C}})$, $\mu_{A,B} : FA \otimes_{\mathbf{D}} FB \to F(A \otimes_{\mathbf{C}} B)$ and the diagrams is to provide compatibility between the components of monoidal structure in $\mathbf{C}$ and the components of monoidal structure in $\mathbf{D}$.

Definition 5 (Strict symmetric monoidal functor)**.** *A strong symmetric monoidal functor $F : \mathbf{C} \to \mathbf{D}$ is* strict *if η and μ are identities.* ◁

Next, we review the monoidal natural transformation, which is a natural transformation compatible with the monoidal structure.

Definition 6 (Monoidal natural transformation)**.** *Let $\mathbf{C}$, $\mathbf{D}$ be symmetric monoidal categories and let $F, G : \mathbf{C} \to \mathbf{D}$ be symmetric monoidal functors. A natural transformation $\phi : F \Rightarrow G$ is* monoidal *if the following diagrams commute for any A and B in $\mathbf{C}$.*



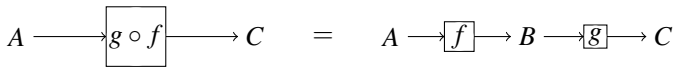
◁

These diagrams indicate that the morphisms $\phi_A \otimes \phi_B$ and $\phi_{A\otimes B}$ are compatible and the morphisms id_I, ϕ_I are compatible.

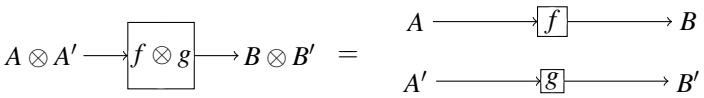
Since there are cases where the naturality of a monoidal natural transformation can be derived from its monoidality (as in Prop. 16), we also define a monoidal natural transformation without assuming naturality.

Definition 7. *We call a family of morphisms* $\{\phi_A : FA \to GA\}$ *indexed by objects* $A \in \mathbf{C}$ *the* transformation. *Moreover, a transformation is* monoidal *if the above diagram with the morphisms in the transformation commutes.* ◁

Throughout this paper, we use *string diagrams* (Selinger, 2011) of a monoidal category. They allow us to depict complicated morphisms consisting of other simple morphisms. Therefore, they are suitable for depicting a module structure. In the diagram, the objects are represented as wires with labels. The morphisms are represented as boxes with labels or points on some wires. For example, we can depict the composition of two arbitrary morphisms $f : A \to B$ and $g : B \to C$, as follows.



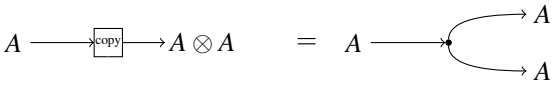
The monoidal product of two arbitrary morphisms $f : A \to B$ and $g : A' \to B'$ can be depicted as follows.



Following Selinger (2011), we add arrowheads to the wires in the diagrams.

Definition 8 (Cartesian category)**.** *A* cartesian category *is a monoidal category equipped with the following morphisms:*

- *for any* $A \in \mathrm{Obj}(\mathbf{C})$*, the morphism* $\mathrm{copy}_A : A \to A \otimes A$ *with naturality, called* copying *morphism, depicted as follows;*



- *for any* $A \in \mathrm{Obj}(\mathbf{C})$*, the morphism* $!_A : A \to I$ *with naturality, called* discarding *morphism, depicted as follows.*

◁

The definition of a cartesian category requires commutative comonoid axioms and coherence axioms, but we omit them; see (Selinger, 2011) for details.

For a cartesian category, we often write the monoidal product $\otimes$ as $\times$ and the unit object I as 1.

Intuitively, a copying morphism $\mathrm{copy}_A : A \to A \times A$ can be regarded as an operation that takes a value and returns the duplicated value. A discarding morphism $!_A : A \to 1$ can be regarded as an operation that takes a value and returns the constant value in 1, which can be removed by the unitors λ, ρ.

2.2.2 Categorical components for machine learning

We define a lens (Foster et al., 2007; Riley, 2018) that can be understood as a computation accompanied by an auxiliary computation. The auxiliary computation propagates certain information in the reverse direction. Note that the forward and backward computations in a lens are not symmetrical. Intuitively, the forward computation is executed first, followed by the backward computation. Therefore, the backward computation can also use an input value used in the forward computation.

Definition 9 (Lens)**.** *For a cartesian category* $\mathbf{C}$ *and objects* A *and* B *in the category* $\mathbf{C}$*, a* lens *from* A *to* B *is a pair of morphisms* $(\!|\, \mathbf{g}, \mathbf{p} \,|\!) : A \rightleftarrows B$ *consisting of a* get *morphism* $\mathbf{g} : A \to B$ *and a* put *morphism* $\mathbf{p} : A \times B \to A$*.* ◁

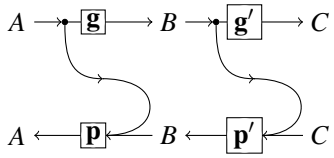
To simultaneously compose two forward computations $\mathbf{g}, \mathbf{g}'$ and two backward computations $\mathbf{p}, \mathbf{p}'$, respectively, we define a composition of lenses, as the composition in a monoidal category.

Definition 10 (Category of lenses)**.** *For any cartesian category* $\mathbf{C}$*, the* category of lenses $\mathbf{Lens}(\mathbf{C})$ *(Riley, 2018) is defined as follows:*

Objects are the same as those of $\mathbf{C}$*. Morphisms from* A *to* B *are lenses. The identity on* A *is* $(\!|\, \mathrm{id}_A, !_A \times \mathrm{id}_A \,|\!)$*. The composition of two lenses* $(\!|\, \mathbf{g}, \mathbf{p} \,|\!) : A \rightleftarrows B$*,* $(\!|\, \mathbf{g}', \mathbf{p}' \,|\!) : B \rightleftarrows C$ *is defined by*

$$(\!|\, \mathbf{g}', \mathbf{p}' \,|\!) \circ (\!|\, \mathbf{g}, \mathbf{p} \,|\!) : A \rightleftarrows C =$$
$$(\!|\, \mathbf{g}' \circ \mathbf{g},\ \mathbf{p} \circ (\mathrm{id}_A \times \mathbf{p}') \circ (\mathrm{id}_A \times \mathbf{g} \times \mathrm{id}_C) \circ (\mathrm{copy} \times \mathrm{id}_C) \,|\!).$$

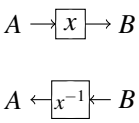
The composite lens can be depicted as follows.



The monoidal product of objects A *and* B*, written as* $A \otimes_L B$*, is simply* $A \times B$*, and that of lenses* $(\!|\, \mathbf{g}, \mathbf{p} \,|\!) : A_1 \rightleftarrows B_1$ *and* $(\!|\, \mathbf{g}', \mathbf{p}' \,|\!) : A_2 \rightleftarrows A_2$*, written as* $(\!|\, \mathbf{g}, \mathbf{p} \,|\!) \otimes_L (\!|\, \mathbf{g}', \mathbf{p}' \,|\!)$ *is* $(\!|\, \mathbf{g} \times \mathbf{g}', (\mathbf{p} \times \mathbf{p}') \circ$

swap$\rrbracket$ *where* swap$: (A_1 \times A_2) \times (B_1 \times B_2) \to (A_1 \times B_1) \times (A_2 \times B_2)$ *is the trivial morphism. The unit* I_L *is* 1.

The associators, left unitors, right unitors, and symmetries are defined as $\llparenthesis \alpha, ! \times \alpha^{-1} \rrbracket$, $\llparenthesis \lambda, ! \times \lambda^{-1} \rrbracket$, $\llparenthesis \rho, ! \times \rho^{-1} \rrbracket$, $\llparenthesis \gamma, ! \times \gamma^{-1} \rrbracket$*, respectively.*[b] *In string diagrams, these lenses are the pairs of morphisms and their inverse, which are depicted as follows* ($x \in \{\alpha, \lambda, \rho, \gamma\}$).



Note that the form of $\llparenthesis x, ! \times x^{-1} \rrbracket$ *is preserved by the composition and the monoidal products of lenses. Namely, the following equations hold.*

$$\llparenthesis x, ! \times x^{-1} \rrbracket \circ \llparenthesis y, ! \times y^{-1} \rrbracket = \llparenthesis x \circ y, ! \times (x \circ y)^{-1} \rrbracket.$$

$$\llparenthesis x, ! \times x^{-1} \rrbracket \otimes_L \llparenthesis y, ! \times y^{-1} \rrbracket = \llparenthesis x \otimes_L y, ! \times (x \otimes_L y)^{-1} \rrbracket.$$

◁

Note that the diagrams that depict lenses are not strictly string diagrams of a category of lenses. In general, a string diagram cannot depict a morphism and another morphism in the reverse direction in the same diagram. We use these diagrams because we want to simultaneously depict the composition of lenses, as well as get functions and put functions. If the readers want to depict this diagram exactly as a string diagram, they should use the *free cornering* of categories described in (Nester, 2021; Boisseau et al., 2023; Nester, 2023)

We now define the monoidal category that abstracts differentiable functions and the operation of differentiation.

Definition 11 (Cartesian reverse differential category)**.** *For any cartesian category* **C***, a* cartesian reverse differential category *(Cockett et al., 2020), abbreviated as* CRDC *is a cartesian category* **C** *with* commutative monoid structures *and a* reverse differential combinator*. A commutative monoid structure is defined as* $(\mathbf{C}(A,B), + : \mathbf{C}(A,B) \times \mathbf{C}(A,B) \to \mathbf{C}(A,B), 0 : 1 \to \mathbf{C}(A,B))$ *for any homsets* $\mathbf{C}(A,B)$*. Additionally, the reverse differential combinator maps an arbitrary morphism* $f : A \to B$ *to* $Rf : A \times B \to A$*.*

The commutative monoid structures satisfy $(f+g) \circ h = (f \circ h) + (g \circ h)$ *for any morphisms* f *and* g*. Moreover, the projections in the cartesian category satisfy* $\pi \circ (f+g) = (\pi \circ f) + (\pi \circ g)$ *for any morphisms* f, g*. The reverse differential combinator is also required to satisfy some axioms, but we omit them; see (Cockett et al., 2020) for details. We describe some important axioms later.* ◁

Example 12 (**Smooth**)**.** *A strict cartesian category* **Smooth** *(Cockett et al., 2020) whose objects are natural numbers (including zero) and whose morphisms from* n *to* m *are smooth functions* $f : \mathbb{R}^n \to \mathbb{R}^m$ *is a CRDC. If it does not cause confusion, we identify* $n \in \mathbb{N}$ *with* $\mathbb{R}^n \in \mathrm{Obj}(\mathbf{Set})$*. Note that* $\mathbb{R}^0$ *is a singleton* $\{*\}$*, which is a set containing only one element and which is the unit object in* **Smooth**.

The commutative monoid structures in homsets are defined by $(f+g)(x) = f(x) + g(x)$ *and* $0(x) = 0 \in \mathbb{R}$.

[b] In these definitions, we omit the left unitors from the put functions. The left unitors play the role of deleting the unit object that arises from discarding the first argument. Therefore, we write $\llparenthesis \alpha, \lambda \circ (! \times \alpha^{-1}) \rrbracket$ as $\llparenthesis \alpha, ! \times \alpha^{-1} \rrbracket$.

Meanwhile, the reverse differential combinator R is defined by $Rf(x,y) = (J_f(x))^\top \cdot y$.
In particular, when m is one, a morphism $f : \mathbb{R}^n \to \mathbb{R}$ *is mapped to* $Rf : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$, *where* $Rf(x,y) = (\nabla_x f(x)) \cdot y$. ◁

Using **Smooth**, we describe an axiom that characterizes the behavior of the reverse differential combinator when it maps a composite function as follows.

For any morphisms and values $f : A \to B, g : B \to C,\ a \in A,\ c \in C$, there is an axiom written as $R(g \circ f)(a,c) = Rf(a, Rg(f(a),c))$. Assuming $A, B, C = \mathbb{R}$, we can quickly realize that the axiom is just the chain rule of derivatives. The key point is that the chain rule has the same form as the composition of lenses. Therefore, a morphism and its reverse derivative together form a lens $(\!| f, Rf |\!)$, and the following equation holds.

$$(\!| g, Rg |\!) \circ (\!| f, Rf |\!) = (\!| g \circ f, R(g \circ f) |\!).$$

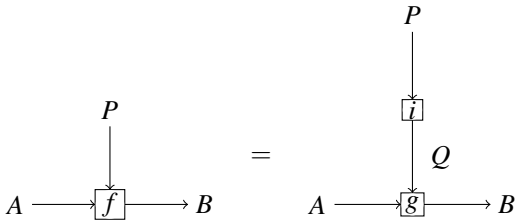
Thus, the construction of these lenses becomes a functor:

Definition 13 (Reverse differential functor)**.** *For any CRDC* **C**, *a* reverse differential functor $\mathrm{Rd} : \mathbf{C} \to \mathbf{Lens}(\mathbf{C})$ *(Cruttwell et al., 2022; Cockett et al., 2020) is an identity-on-objects strict symmetric monoidal functor that maps an arbitrary morphism* $f : A \to B$ *to lens* $(\!| f, Rf |\!) : A \rightleftarrows B$. ◁

3. Para construction

As in section 2.1, the notion of parameter plays a central role in (a certain class of) machine learning. For this, Cruttwell et al. used a general technique called *para construction* (Gavranović, 2019; Cruttwell et al., 2022; Capucci et al., 2022; Shiebler, 2021*a*), and a similar construction already appears in Fong et al. (2019). The para construction yields a category whose morphisms can be considered as architectures. We also describe some examples of architectures and their construction through para construction.

Identification of architectures One difference between the para constructions is that a morphism in the latter one is defined as an equivalence class under some equivalence relation, while one in the former is not. The general motivation for this identification comes from categorical convenience, but it is reasonable in the context of machine learning. The equivalence relation between architectures is depicted as follows,



where $f : P \otimes A \to B$ and $g : Q \otimes A \to B$ are architectures and $i : P \to Q$ is an isomorphism. The identification by the relation means that the architectures f and g are the same up to the existence of an isomorphism $i : P \to Q$ appended on the side of parameter object Q. In practice, a parameter value in machine learning is treated as a black box. We do not directly operate the parameter, and it is operated only when predicting and training. Therefore, we do not distinguish between

architectures based on how to use a parameter. We represent the difference as the existence of an isomorphism $i : P \to Q$.

Existing problem However, Cruttwell et al. (2022) point out that this equivalence relation makes a main result of Fong et al. ill-defined. Intuitively, the ill-definedness implies that even if two architectures are equivalent, results of training can be different. The cause of the problem is that they identify architectures f and g by an arbitrary isomorphism i between them. Therefore, we restrict the class of isomorphisms to be as small as possible.

3.1 Subcategory for restricting identification

We restrict these isomorphisms by defining a symmetric monoidal subcategory. A morphism in the subcategory is constructed only from the monoidal structure, such as identities, associators, unitors, or symmetries. We call a subcategory $\mathbf{D}$ of a category $\mathbf{C}$ a *wide subcategory* if the inclusion is surjective on objects, and a *subgroupoid* if $\mathbf{D}$ is a groupoid.

Given a symmetric monoidal category $\mathbf{C}$, among symmetric monoidal wide subgroupoids of $\mathbf{C}$, there exists the smallest one $\mathbf{C}_c$ with respect to the inclusion ordering, given by the intersection of all those. To examine examples etc., it is convenient to have a more concrete description:

Definition 14 (Smallest symmetric monoidal wide subcategory)**.** *The smallest symmetric monoidal wide subcategory* $\mathbf{C}_c$ *of* $\mathbf{C}$ *is given as follows:*

- ***Objects*** $\mathrm{Obj}(\mathbf{C}_c) = \mathrm{Obj}(\mathbf{C})$.
- ***Morphisms*** *Let*

$$\mathrm{Mor}(\mathbf{C}_c)_0 = \bigcup_{A,B,C \in \mathrm{Obj}(\mathbf{C})} \{\mathrm{id}_A, \alpha_{A,B,C}, \alpha^{-1}_{A,B,C}, \lambda_A, \lambda_A^{-1}, \rho_A, \rho_A^{-1}, \gamma_{A,B}, \gamma^{-1}_{A,B}\},$$

and, for any $n \in \mathbb{N}$*, let*

$$\begin{aligned}\mathrm{Mor}(\mathbf{C}_c)_{n+1} = \{g \circ f \mid f, g \in \mathrm{Mor}(\mathbf{C}_c)_n, \mathrm{cod}(f) = \mathrm{dom}(g)\} \\ \cup \{f \otimes g \mid f, g \in \mathrm{Mor}(\mathbf{C}_c)_n\}.\end{aligned}$$

Then we define the set of all morphisms $\mathrm{Mor}(\mathbf{C}_c)$ *as* $\bigcup_{n \in \mathbb{N}} \mathrm{Mor}(\mathbf{C}_c)_n$.

The structures that make this a symmetric monoidal wide subgroupoid of $\mathbf{C}$ *are given in an obvious way. Moreover, we can define the inclusion functor* $\iota : \mathbf{C}_c \to \mathbf{C}$. ◁

We say that a morphism in $\mathbf{C}$ is *canonical* if it is a morphism in $\mathbf{C}_c$.

Example 15 ($\mathbf{Smooth}_c$ and $\mathbf{Lens}(\mathbf{C})_c$)**.** *The morphisms in* $\mathbf{Smooth}_c$ *are constructed only from* γ *and* id *because* $\mathbf{Smooth}$ *is strict. Therefore,* $\mathbf{Smooth}_c(\mathbb{R}^n, \mathbb{R}^n)$ *can be regarded as a symmetric group* S_n *(in group theory).*

By the definition of the monoidal structure of $\mathbf{Lens}(\mathbf{C})$*, the base-case morphisms in* $\mathbf{Lens}(\mathbf{C})_c$ *have the form* $(\!| x, ! \times x^{-1} |\!)$ *where* x *is a canonical morphism in* $\mathbf{C}_c$*, and this form is closed under composition and monoidal product. The converse also holds similarly, so we have*

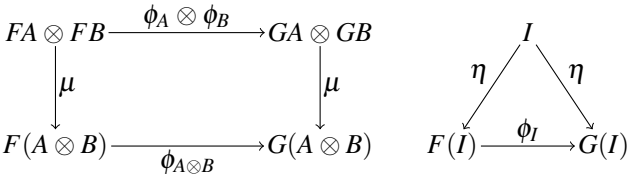
$$\mathbf{Lens}(\mathbf{C})_c(A, B) = \{(\!| x, !_A \times x^{-1} |\!) \mid x \in \mathbf{C}_c(A, B)\}.$$

◁

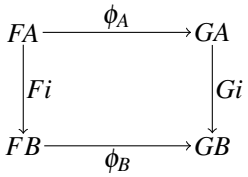
Restricting morphisms to canonical ones is also useful to derive the naturality of a family of morphisms. Suppose we restrict $\mathbf{C}$ to $\mathbf{C}_c$ in a monoidal natural transformation $\phi : F \Rightarrow G$:

$\mathbf{C} \to \mathbf{D}$, namely, the equations of naturality $G(f) \circ \phi_A = \phi_B \circ F(f)$ are required only for canonical morphisms $f : A \to B$. Then, the restricted naturality can be derived from the monoidality of ϕ:

Proposition 16. *For any symmetric monoidal categories* $\mathbf{C}, \mathbf{D}$ *and strong symmetric monoidal functors* $F, G : \mathbf{C}_c \to \mathbf{D}$*, let a transformation* $\{\phi_A : FA \to GA\}$ *be monoidal. Namely, the following diagrams commute for any* A *and* B *in* $\mathbf{C}_c$*.*



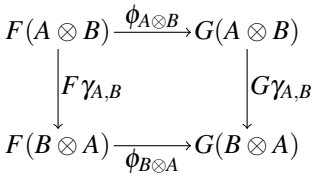
Then the transformation ϕ *is natural, namely, the following diagrams commute for any isomorphism* $i : A \to B$ *in* $\mathbf{C}_c$*.*



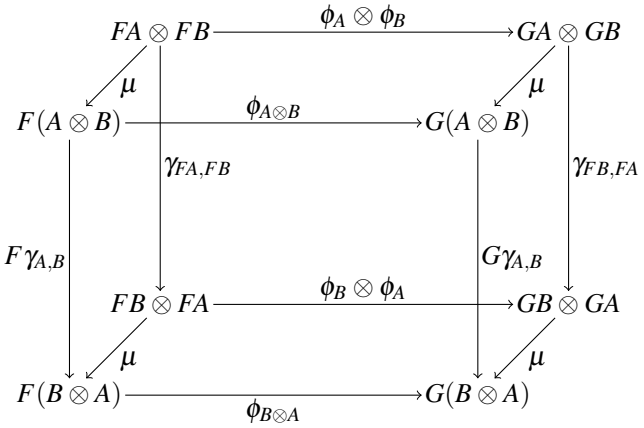
As a consequence, the family becomes a monoidal natural transformation. ◁

Proof. Since the morphism $i : A \to B$ in the diagram is canonical and defined recursively, we can prove the commutativity of the diagram by induction.

As base cases, we need to prove the equations of naturality in the cases $i = \alpha,\ i = \lambda,\ i = \rho,\ i =$ id, and $i = \gamma$. We only prove the equation with $i = \gamma$, which can be depicted as follows.

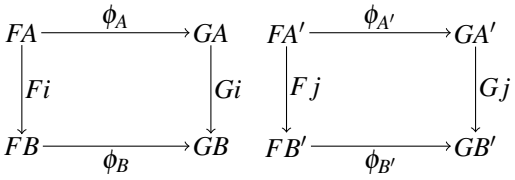


The commutativity can be proved by the following diagram.

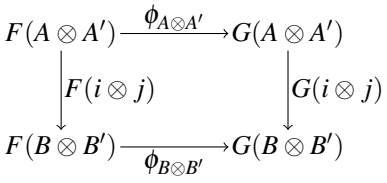


The equation we aim to show is depicted as the front square. Because the other squares commute, and μ are epimorphisms, the front square commutes.

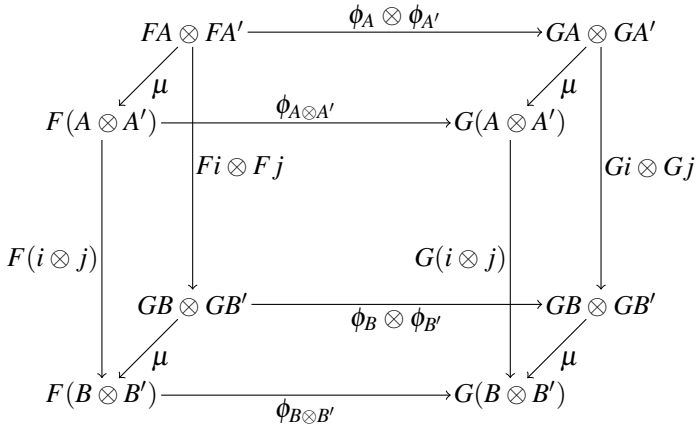
As the induction step, we need to prove that if the diagrams of naturality with canonical morphisms i, j commute, then the diagrams with $i \circ j$ and $i \otimes j$ commute. We prove only the commutativity of $i \otimes j$. For any canonical morphism $i : A \to B$, $j : A' \to B'$ such that the following diagrams commute,



the following diagram commutes.



The commutativity can be proved by the following diagram.



As in the base case, the front square commutes, which we aim to prove. □

3.2 Para construction and its use

Now we introduce our Para construction.

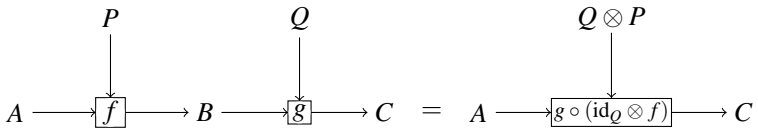
Definition 17 (Para construction)**.** *Given a symmetric monoidal category* $\mathbf{C}$ *and a symmetric monoidal wide subgroupoid* $\mathbf{D}$*, a symmetric monoidal category* $\mathbf{Para}_{\mathbf{D}}(\mathbf{C})$ *is defined as follows.*

The objects are the same as those in $\mathbf{C}$*. For objects* A *and* B*, let* $\mathbf{P}(A,B)$ *be the set of pairs* (P,f) *consisting of an object* P *in* $\mathbf{C}$*, called a* parameter object*, and a morphism* $f : P \otimes A \to B$*, called a* para morphism*. Then we define the homset by* $\mathbf{Para}_{\mathbf{D}}(\mathbf{C})(A,B) = \mathbf{P}(A,B)/\sim$ *where the equivalence relation* $\sim$ *on* $\mathbf{P}(A,B)$ *is given by:*

$$(P,f) \sim (Q,g) \iff \text{there exists a } \mathbf{D}\text{-isomorphism } i : P \cong Q \text{ such that } f = g \circ (i \otimes \mathrm{id}_A).$$

For a morphism $[(P,f)]$*, an equivalence class, we often write for it its representative* (P,f)*, or simply* f *if* P *is clear.*

The composition of two morphisms $(P,f) : A \to B$ *and* $(Q,g) : B \to C$ *is defined by* $(Q,g) \circ_{\mathbf{P}} (P,f) = (Q \otimes P, g \circ (\mathrm{id}_Q \otimes f)) : A \to C$*, which is depicted as follows*[c].



We call the composition para composition*. The identity morphism on* A *is* $(I, \lambda : I \otimes A \to A)$*. The monoidal product of objects* A *and* B*, written as* $A \otimes_{\mathbf{P}} B$*, is just* $A \otimes B$*, and that of morphisms* $(P_1,f_1) : A_1 \to B_1$ *and* $(P_2,f_2) : A_2 \to B_2$*, written as* $(P_1,f_1) \otimes_{\mathbf{P}} (P_2,f_2)$*, is* $(P_1 \otimes P_2, (f_1 \otimes f_2) \circ \mathrm{swap})$ *where* $\mathrm{swap} : (P_1 \otimes P_2) \otimes (A_1 \otimes A_2) \to (P_1 \otimes A_1) \otimes (P_2 \otimes A_2)$ *is the trivial one. The unit* $I_{\mathbf{P}}$ *is* I*. The associators, the unitors, and the symmetries in* $\mathbf{Para}(\mathbf{C})$ *are non-parameter (*$P = I$*)*

[c] In Fong et al. (2019), the order of P and Q in the parameter object of the composite morphism is $P \otimes Q$, whereas in Cruttwell et al. (2022) it is $Q \otimes P$; we adopt the latter.

morphisms arising from those in $\mathbf{C}$*. As an example, the associators in* $\mathbf{Para}(\mathbf{C})$ *are* $(I, \alpha \circ \lambda)$*, where the left unitor* $\lambda : I \otimes A \to A$ *deals with the parameter object* I*.*

◁

Note that if we do not use the appropriate equivalence relation nor the strictness constraint, $\mathbf{Para}_{\mathbf{D}}(\mathbf{C})$ does not form a category (though forms a bicategory).

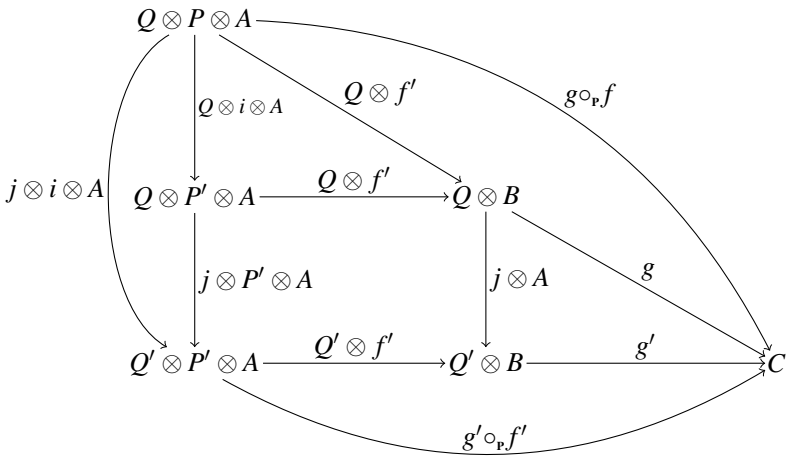
In this paper, we always refer to the smallest symmetric monoidal wide subgroupoid $\mathbf{C}_{\mathrm{c}}$ as $\mathbf{D}$. Therefore, we omit the subscript of $\mathbf{Para}_{\mathbf{D}}(\mathbf{C})$ when $\mathbf{D} = \mathbf{C}_{\mathrm{c}}$, and simply write $\mathbf{Para}(\mathbf{C})$.

Proposition 18. *For any symmetric monoidal category* $\mathbf{C}$*,* $\mathbf{Para}(\mathbf{C})$ *is a category.* ◁

Proof. First, we confirm that the para composition in $\mathbf{Para}(\mathbf{C})$ preserves the equivalence relation. Namely, we prove that $(P, f) \sim (P', f'), (Q, g) \sim (Q', g') \Longrightarrow (Q \otimes P, g \circ_{\mathrm{P}} f) \sim (Q' \otimes P', g' \circ_{\mathrm{P}} f')$ for any $P, Q, P', Q', A, B, C \in \mathrm{Obj}(\mathbf{C})$,

$$\begin{aligned} &f : P \otimes A \to B, && g : Q \otimes B \to C, \\ &f' : P' \otimes A \to B, \text{ and} && g' : Q' \otimes B \to C. \end{aligned}$$

If $(P, f) \sim (P', f')$ and $(Q, g) \sim (Q', g')$ hold with canonical morphisms $i : P \to P'$, $j : Q \to Q'$ in $\mathbf{C}$, the following diagram commutes.

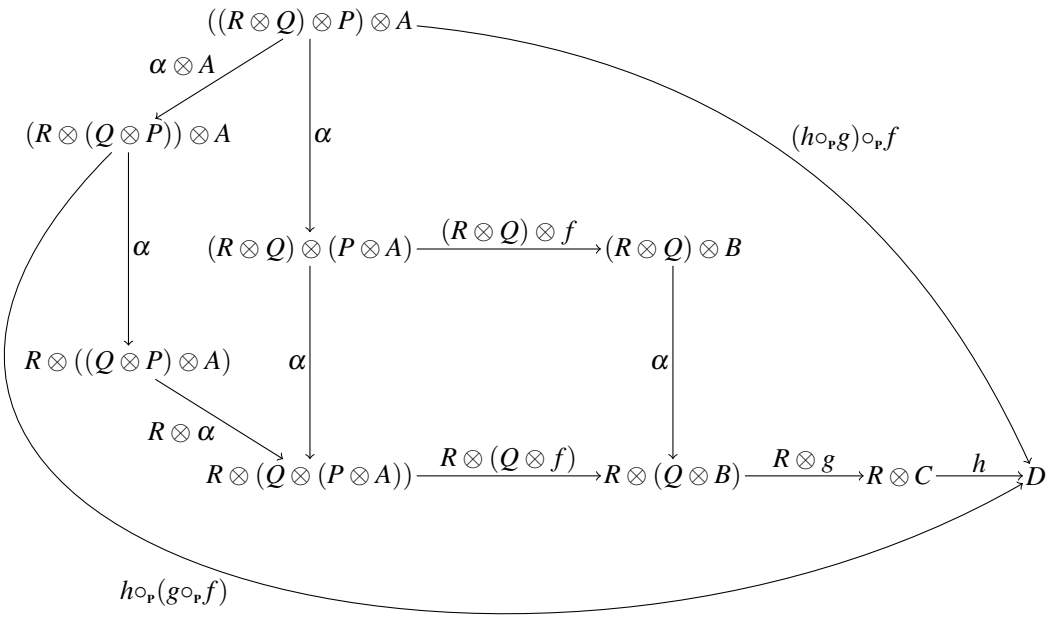


Because the morphism $j \otimes i : Q \otimes P \to Q' \otimes P'$ is canonical, $(Q \otimes P, g \circ_{\mathrm{P}} f) \sim (Q' \otimes P', g' \circ_{\mathrm{P}} f')$ holds.

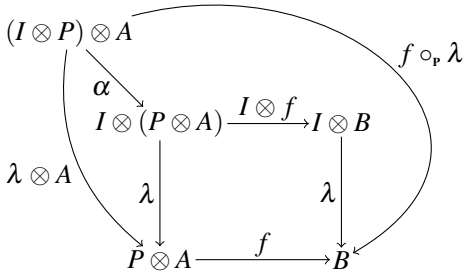
Next, we confirm the associativity of the composition of morphisms. For any objects $P, Q, R, A, B, C \in \mathrm{Obj}(\mathbf{C})$ and para morphisms

$$(P, f) : A \to B, (Q, g) : B \to C, (R, h) : C \to D,$$

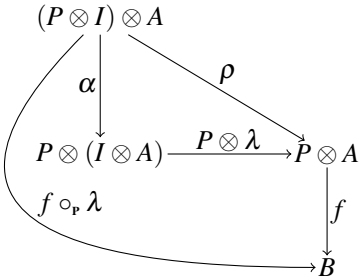
the equation $(h \circ_{\mathrm{P}} g) \circ_{\mathrm{P}} f = (h \circ_{\mathrm{P}} (g \circ_{\mathrm{P}} f)) \circ (\alpha \otimes A)$ holds, because the following diagram commutes. Therefore, because $\alpha \otimes A$ is canonical, the morphisms $(h \circ_{\mathrm{P}} g) \circ_{\mathrm{P}} f$ and $h \circ_{\mathrm{P}} (g \circ_{\mathrm{P}} f)$ are equivalent.



The left identity law is confirmed as follows. For any morphisms $(P, f) : A \to B$, the morphisms $f \circ_{\mathbf{P}} \lambda$ and f are equivalent because the following diagram commutes and λ is canonical.



Similarly, the right identity law can be proved by the following diagram.



□

Next, we consider the lifting of functor from $F : \mathbf{C} \to \mathbf{C}'$ to $\mathbf{Para}(F) : \mathbf{Para}(\mathbf{C}) \to \mathbf{Para}(\mathbf{C}')$.

Proposition 19. *If a strong symmetric monoidal functor $F : \mathbf{C} \to \mathbf{C}'$ satisfies the following two conditions:*

(1) $F(f) \in \mathbf{D}'$ if $f \in \mathbf{D}$,
(2) $\mu_{A,B} : FA \otimes FB \to F(A \otimes B)$, $\mu_I : I \to F(I) \in \mathbf{D}'$,

then we can define a strong symmetric monoidal functor $\mathbf{Para}(F) : \mathbf{Para}(\mathbf{C}) \to \mathbf{Para}(\mathbf{C}')$ as follows:

- $\mathbf{Para}(F)(A) := FA$,
- $\mathbf{Para}(F)(P, f) := (FP, Ff \circ \mu_{P,A})$.

◁

If $\mathbf{D} = \mathbf{C}_{\mathrm{c}}$ and $\mathbf{D}' = \mathbf{C}'_{\mathrm{c}}$, we have the following implications:

$$F \text{ is strict} \implies \text{the condition 2} \implies \text{the condition 1.}$$

In gradient-based learning, an architecture generally consists of *layers*, which calculate an affine transformation of a vector with some kind of non-linear function called *activation function*. As an example of an architecture, we review the fully-connected layer as a morphism in $\mathbf{Para}(\mathbf{Smooth})$:

Example 20 (Fully-connected layer)**.** *A* fully-connected layer *(Cruttwell et al., 2022; Goodfellow et al., 2016) in which the input size is n and the output size is m is a morphism in $\mathbf{Para}(\mathbf{Smooth})$ as follows. The morphism is denoted by $(\mathbb{R}^{m \times n} \times \mathbb{R}^m, f) : \mathbb{R}^n \to \mathbb{R}^m$. These parameters consist of a weight matrix $W \in \mathbb{R}^{m \times n}$ as a $m \times n$ matrix and a bias $b \in \mathbb{R}^m$ as a vector. We define the layer by $f((W, b), x) = \varphi^m(W \cdot x + b)$ where φ^m is an m-times product of a morphism $\varphi : \mathbb{R} \to \mathbb{R}$, denoted as $\overbrace{\phi \times \cdots \times \phi}^{m\text{-times}} : \mathbb{R}^m \to \mathbb{R}^m$.* ◁

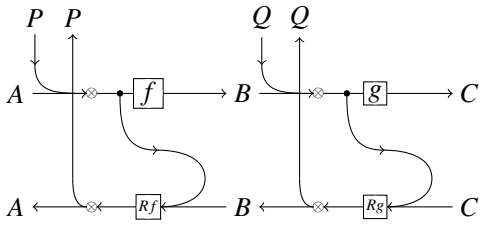
Since the category of lenses $\mathbf{Lens}(\mathbf{C})$ is a symmetric monoidal category, we can apply para construction to the category $\mathbf{Lens}(\mathbf{C})$.

Definition 21. *For any cartesian category $\mathbf{C}$, we call a pair $(P, (\!|\, \mathbf{g}, \mathbf{p} \,|\!))$* para lens*, consisting of an object P in $\mathbf{C}$ and a lens $(\!|\, \mathbf{g}, \mathbf{p} \,|\!) : P \otimes_L A \leftrightarrows B$.* ◁

Example 22 (Cruttwell et al., 2022)**.** *We can easily define an identity-on-objects symmetric strict monoidal functor $\mathbf{Para}(\mathrm{Rd}) : \mathbf{Para}(\mathbf{C}) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$ because* Rd *is a strict monoidal functor. Additionally, the preservation of the para composition is denoted by*

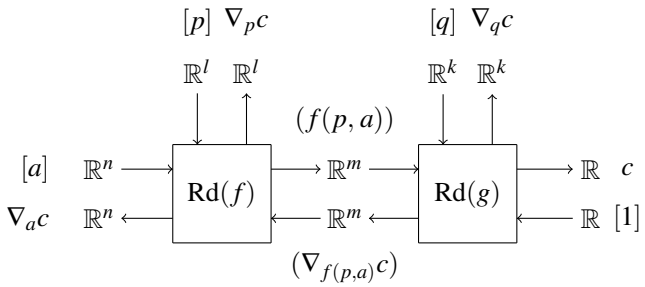
$$(Q \times P, \mathrm{Rd}(g \circ_P f)) = (Q, \mathrm{Rd}(g)) \circ_P (P, \mathrm{Rd}(f)).$$

Note that $\circ_P$ on the left side of the equation is the para composition in $\mathbf{Para}(\mathbf{C})$, while $\circ_P$ on the right side is the para composition of lenses defined in $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$, depicted as follows,



where the symbol $\begin{matrix} A \\ B \end{matrix} \!\!\rightarrow\!\!\otimes\!\!\rightarrow A \otimes B$ *denotes that we abbreviate multiple wires on the right side of* $\otimes$ *to a single wire. This notation is borrowed from (Cockett and Seely, 2017; Nester, 2023).* ◁

Example 23. *The calculation of backpropagation in section 2.1 can be constructed as lenses in* **Lens**(**Smooth**) *by the functor* Rd*, which is depicted as follows,*



where $a \in \mathbb{R}^n$, $p \in \mathbb{R}^l$, $q \in \mathbb{R}^k$, *and* $1 \in \mathbb{R}$ *are inputs written in brackets,* $c = g(q, f(p,a)) \in \mathbb{R}$, $\nabla_p c \in \mathbb{R}^l$, $\nabla_q c \in \mathbb{R}^k$, $\nabla_a c \in \mathbb{R}^n$ *are outputs written without decorations, and* $f(p,a) \in \mathbb{R}^m$, $\nabla_{f(p,a)} c \in \mathbb{R}^m$ *are intermediate values written in the parentheses. In this way, we also simultaneously depict the morphisms and correspondences between values.*

Thus, we can construct the "backpropagation" algorithm by the composition of lenses ◁

Remark 24. By functoriality, the equation $(Q \times P, \mathrm{Rd}(g \circ_{\mathbf{P}} f)) = (Q, \mathrm{Rd}(g)) \circ_{\mathbf{P}} (P, \mathrm{Rd}(f))$ holds. The equation is extensional, namely, it only requires the results of calculations to be the same; the equation does not account for the computational cost. The right-hand side of the equation performs backpropagation, but the left-hand side does not necessarily perform it.

◁

Finally, we present an example of a learning method defined by a combination of architectures. In machine learning, *(mini-batch) stochastic gradient descent* is regarded as an optimization method, which minimizes a loss using an averaged gradient computed over a mini-batch of randomly sampled training data. However, in categorical discussions, the learning algorithm of stochastic gradient descent can be emulated through the construction of an architecture.

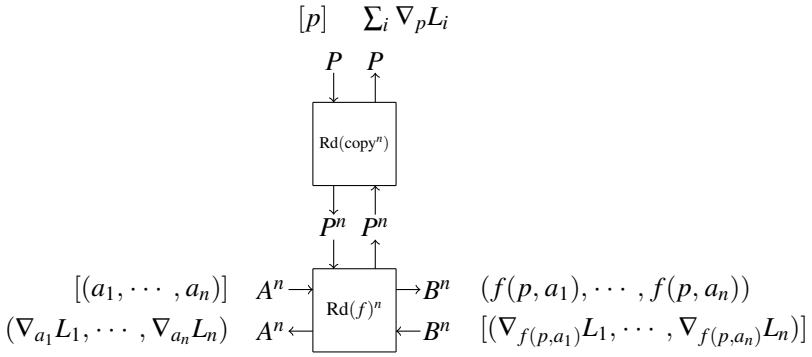
Example 25 (Stochastic gradient descent; cf. Gavranović, 2024; Goodfellow et al., 2016)**.** *For any architecture* $(P, f) : A \to B$ *in* **Para**(**Smooth**) *and* batch size $n \in \mathbb{N}$*, an architecture that emulates*

stochastic gradient descent is defined as follows:

$$(P, (\mathrm{copy}^n \times \mathrm{id}_{A^n}) \circ f^n) : A^n \to B^n$$

where $(P^n, f^n) = \overbrace{f \otimes_P \cdots \otimes_P f}^{n\text{-times}} : A^n \to B^n$ *and* $\mathrm{copy}^n : P \to P^n$ *is a morphism in* $\mathbf{C}$*, which duplicates an input n-times and returns the copies. Note that the function* copy^n *can be defined from identity morphisms and morphisms* $\mathrm{copy} : X \to X \times X$*. Applying the functor* Rd *to this architecture, we obtain a para lens:*

$$\mathbf{Para}(\mathrm{Rd})(P, f^n \circ (\mathrm{copy}^n \times \mathrm{id}_{A^n})) = (P, \mathrm{Rd}(f)^n \circ (\mathrm{Rd}(\mathrm{copy}^n) \times \mathrm{id}_{A^n}))$$



where $(P^n, \mathrm{Rd}(f)^n) = \overbrace{\mathrm{Rd}(f) \otimes_P \cdots \otimes_P \mathrm{Rd}(f)}^{n\text{-times}} : A^n \leftrightarrows B^n$*. We describe the behaviors of the lenses:*

- ***Para lens*** $(P^n, \mathrm{Rd}(f)^n)$ *The get morphism computes predictions* $(f(p, a_1), \cdots, f(p, a_n))$ *in parallel, from different inputs* $(a_1, \cdots a_n)$*, using the same architecture* f *and the same parameter* p*. The put morphisms also backpropagate in parallel, from different inputs and gradients, using the same architecture and the same parameter.*
- ***Lens*** $\mathrm{Rd}(\mathrm{copy}^n)$ *The get morphism duplicates the parameter* p *into* $(p, \cdots, p)$*. Meanwhile, the put morphism outputs the sum of the gradients of the loss* $\sum_i \nabla_p L_i$ *from the gradients of the loss* $(\nabla_p L_1, \cdots, \nabla_p L_n)$*. This is because the equation*

$$R(\mathrm{copy}^n)(p, x_1 \cdots x_n) = \sum_i x_i$$

holds; see (Gavranović, 2024, 7.1.1) for details.

However, it cannot divide the sum by the batch size n, as part of normalization for averaging. Fortunately, by adjusting a learning rate in an optimization, we can multiply a gradient by such a scalar. ◁

4. CGGWZ construction and FST construction

There are two categorical methods that construct learning algorithms from architectures. For ease of explanation, we first review the construction of Cruttwell et al. (2022), even though Fong et al.

(2019) was published earlier. The first construction, reviewed in section 4.1, is the CGGWZ construction introduced by Cruttwell et al. (2022), which is more faithful to actual computations than the FST construction. The second construction, reviewed in section 4.3, is the FST construction introduced by Fong et al. (2019), which appends an original operation to an algorithm and has functoriality, compared with the CGGWZ construction. Finally, we consider a property called the GetPut law for the learning algorithms.

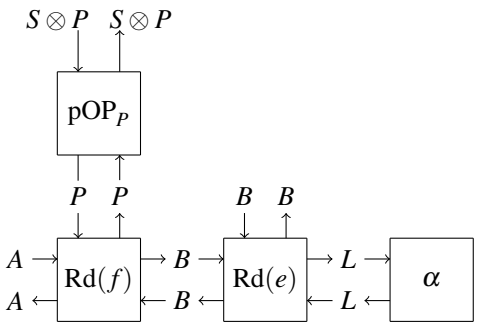
This whole section is mainly preparation, just a review or reorganization of existing work, and our contributions are not presented here (except for some remarks).

4.1 CGGWZ construction

Cruttwell et al. (2022) show that a gradient-based learning algorithm can be constructed as a morphism in $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$. This construction is faithful to the actual computation, namely, there are no computations and constraints that are unnecessary to training. As an application of category theory, the algorithms by this construction are modularized. It decomposes each feature of a learning algorithm, such as a calculation of loss or an optimization, into lenses.

Definition 26 (CGGWZ algorithm)**.** *For any CRDC* $\mathbf{C}$*, para morphisms* $(P, f) : A \to B$*,* $(B, e) : B \to L$*, lenses* $\alpha : L \leftrightarrows I_L$*, and* $\mathrm{pOP}_P : S \otimes_L P \leftrightarrows P$*, the CGGWZ algorithm is defined as follows.*

$$(S \otimes_L P \otimes_L B,\ \alpha \circ \mathrm{Rd}(e) \circ (\mathrm{id}_B \otimes_L \mathrm{Rd}(f)) \circ (\mathrm{pOP}_P \otimes_L \mathrm{id}_{B \otimes_L A})) : A \leftrightarrows I_L$$



We call the algorithms the CGGWZ algorithms *and call the construction the* CGGWZ construction. ◁

For training, these morphisms and lenses play the following roles.

Architecture $(P, f) : A \to B$ is a para morphism in $\mathbf{Para}(\mathbf{C})$ as an architecture of a learning algorithm. Note that if $f = g \circ_{\mathbf{r}} h$, we can use $\mathrm{Rd}(g) \circ_{\mathbf{r}} \mathrm{Rd}(h)$ instead of $\mathrm{Rd}(f)$ because of the functoriality of $\mathbf{Para}(\mathrm{Rd}) : \mathbf{Para}(\mathbf{C}) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$. For the para composition of $\mathrm{Rd}(g)$ and $\mathrm{Rd}(h)$, the algorithm performs backpropagation.

Loss function $(B, e) : B \to L$ is a para morphism in $\mathbf{Para}(\mathbf{C})$ that calculates the loss between two values in B. This para morphism is defined by a loss function $e : B \times B \to L$, where the parameter object is the first argument and L is an arbitrary object. Note that we provide a desired output as a parameter. The put function of the lens $\mathrm{Rd}(e)$ calculates a loss gradient.

Learning rate lens $\alpha : L \leftrightarrows I_{\mathrm{L}}$ is a lens that outputs a learning rate in L to the backward direction. Since $I_{\mathrm{L}} = 1$ and the codomain of a lens is I_{L}, the get function is the terminal morphism $!_L : L \to I_{\mathrm{L}}$ and the put function is a morphism $\mathbf{p} : L \to L$.

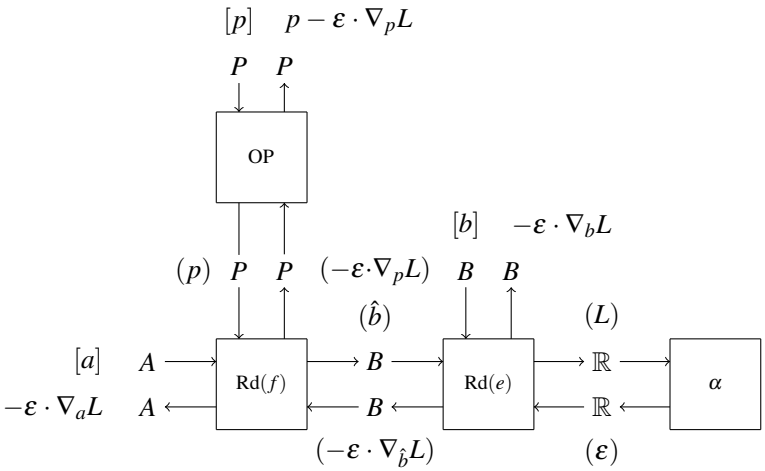
Preoptimizer lens We call $\mathrm{pOP}_P : S \otimes_{\mathrm{L}} P \leftrightarrows P$ a *preoptimizer*[d], and the object S is called *state object*. The put function $\mathbf{p} : S \times P \times P \to S \times P$, mapping (s, p, x) to (s', p'), updates parameter p to p' and state s to s' by "the loss gradient" x. In the gradient descent in the form of preoptimizer, the state is not used (i.e., $S = 1 = \mathbb{R}^0$), and examples that utilize states are given in Ex. 28.

We can construct a concrete learning algorithm from backpropagation and gradient descent described in section 2.1 as a CGGWZ algorithm:

Example 27. *Suppose* $\mathbf{C} = \mathbf{Smooth}$, $L = \mathbb{R}$, $S = 1 = \mathbb{R}^0$, $\mathrm{pOP} = (\!|\, \mathrm{id}_P, +_P \,|\!) : I_L \times P \leftrightarrows P$, *and* $\alpha = (\!|\, !_L, \Delta\varepsilon \,|\!) : \mathbb{R} \leftrightarrows I_L$ *where* $\Delta\varepsilon$ *is a constant function that outputs some* $-\varepsilon \in \mathbb{R}$. *For any para function* $f : A \to B$ *and loss function* $(B, e) : B \times B \to \mathbb{R}$, *the CGGWZ algorithm is defined as a lens in* $\mathbf{Para}(\mathbf{Lens}(\mathbf{Smooth}))$. *For any training data of input* $a \in A$, *training data of desired output* $b \in B$, *and parameter* $p \in P$, *the computation of the put function*[e] *is denoted by*

$$\mathbf{p}(b, p, a) = (\varepsilon \cdot \nabla_b L,\ p - \varepsilon \cdot \nabla_p L,\ \varepsilon \cdot \nabla_a L)$$

where $\widehat{b} = f(p, a),\ L = e(\widehat{b}, b) = e(f(p, a), b)$.



◁

Not only in this example, but we also often implicitly suppose that a CRDC $\mathbf{C}$ is the category **Smooth** only when explaining the diagram, since we want to use elements to explain the behavior of a morphism by the correspondences of elements.

To explain the use of state objects, we review the examples of preoptimizers:

[d] In Cruttwell et al. (2022), a preoptimizer is called an optimizer, but in this paper we reserve the term for another notion given in (PyTorch Foundation, 2024), which is constructed from preoptimizer and learning rate, since the notion of optimizer usually contains that of learning rate.

[e] In Cruttwell et al. (2022), the training data and initial parameter are provided to the algorithm through the parameter objects.

Example 28 (Cruttwell et al., 2022; Goodfellow et al., 2016)**.** *For any* $P \in \mathbf{Smooth}$*, there are preoptimizers as lenses in* $\mathbf{Para}(\mathbf{Lens}(\mathbf{Smooth}))$*:*

Momentum *Momentum lens* $\mathrm{pOP}_P^{\mathrm{Mom}} : P \otimes_L P \rightrightarrows P$ *is defined by*

$$\mathbf{g}(s,p) = p,\ \ \mathbf{p}(s,p,x) = (\gamma s + x,\ \ p + \gamma s + x)$$

where $\gamma > 0$ *is a constant;*

Nesterov Momentum *Nesterov momentum lens* $\mathrm{pOP}_P^{\mathrm{Nest}} : P \otimes_L P \rightrightarrows P$ *is defined by*

$$\mathbf{g}(s,p) = p + \gamma s,\ \ \mathbf{p}(s,p,x) = (\gamma s + x,\ \ p + \gamma s + x)$$

where $\gamma > 0$ *is a constant;*

Adagrad *Adagrad lens* $\mathrm{pOP}_P^{\mathrm{Adag}} : P \otimes_L P \rightrightarrows P$ *is defined by*

$$\mathbf{g}(s,p)_i = p_i$$

$$\mathbf{p}(s,p,x)_i = (s_i + {x_i}^2,\ p_i + \frac{1}{\delta + \sqrt{s_i + {x_i}^2}} x_i)$$

where δ *is a positive real constant (near zero). Note that the put function of the Adagrad lens is not total, but we refrain from delving into the fact.* ◁

These optimizers help parameter updates converge by using a state to store gradient information calculated in past training (updating a parameter). Specifically, states in the Momentum and the Nesterov Momentum are used as a velocity for moving a parameter, which is accelerated by a gradient. A state in the Adagrad is used to adapt learning rates individually for each parameter component.

4.2 Learner and Output backpropagation

Fong et al. (2019) introduce the notion of a learner, together with a construction of gradient-based learning algorithms as a functor from **Smooth** to $\mathbf{Para}(\mathbf{Lens}(\mathbf{Smooth}))$. We call such an algorithm an *FST algorithm*.

Intuitively, the backpropagation described in section 2.1 and the CGGWZ algorithm backpropagate gradients between architectures. In contrast, the composition of learners can be considered to backpropagate desired outputs between learners. To distinguish them, we call the latter backpropagation method *output backpropagation* (which might better be referred to as *desired-output backpropagation*, but for brevity we call it so).

The collection of learners is an abstract framework, with which output backpropagation is equipped, and in which FST algorithms live.

Definition 29 (Learner, Fong et al., 2019)**.** *Let* $\mathbf{C}$ *be a cartesian category.*[f] *A* learner *is just a para lens in* $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$. ◁

We explain the intuition of a learner $(P, (\!|\, \mathbf{g}, \mathbf{p} \,|\!)) : A \to B$, i.e., how to interpret it as a learning algorithm, supposing $\mathbf{C} = \mathbf{Set}$. For any values $p \in P, a \in A, b \in B$,

(1) when we apply $\mathbf{p}$ to p, a, b or ($\mathbf{g}$ to p, a), the values p, a, b are regarded as, respectively, a given parameter (to be trained), a training data of input, and a training data of output,
(2) let $\hat{b} = \mathbf{g}(p,a)$, then $\hat{b}$ is regarded as the prediction computed from p and a by the architecture $\mathbf{g}$,

[f] In Fong et al. (2019), $\mathbf{C}$ was fixed to be **Set**.

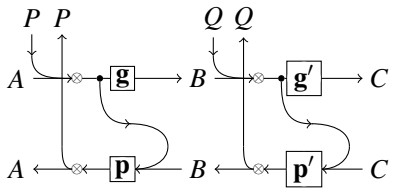
(3) let $(p', a') = \mathbf{p}(p, a, b)$, then p' is regarded as the updated parameter trained by the learning algorithm $\mathbf{p}$, while $\mathbf{p}$ also modifies the input a to a' so that hopefully $\mathbf{g}(p, a')$ is closer to the desired output b than $\mathbf{g}(p, a)$ is.

If we call a para lens a learner, we suppose the above interpretation as a learning algorithm, and we write a learner $(P, \langle\!| \mathbf{g}, \mathbf{p} |\!\rangle) : A \rightleftarrows B$ also as $\langle\!| \mathbf{g}, \mathbf{p} |\!\rangle : A \overset{P}{\Rrightarrow} B$. Note that the interpretation of a para lens by the CGGWZ construction is different from the above one; e.g., a desired output b is contained in a parameter. Also note that, a learner has an architecture as its get function, while a para lens by the CGGWZ construction does not have an architecture as its structure (though it does use during the construction).

In item 3 above, the viewpoint is that a' is a modification of a by b (and p), but we can also regard a' as a "backpropagation" from b by a (and p). Anyway, the role of a' might still be unclear, but it can be used in the next:

Definition 30 (Output backpropagation)**.** *Output backpropagation is just the composition in the category* $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$. ◁

For $\mathbf{C} = \mathbf{Set}$, we explain the interpretation of the output backpropagation of two learners $\langle\!| \mathbf{g}, \mathbf{p} |\!\rangle : A \overset{P}{\Rrightarrow} B$ and $\langle\!| \mathbf{g}', \mathbf{p}' |\!\rangle : B \overset{Q}{\Rrightarrow} C$. This is nothing but the unfolding of the definition of composition of $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$ with the terminology of learning, but the role of a' above, which corresponds to b' below, should be clarified. The string diagram of the composite should be helpful:



The type of the composite is $: A \overset{Q \times P}{\Rrightarrow} C$, and let $(q, p) \in Q \times P$ be a given pair of parameters, $a \in A$ and $c \in C$ be a pair of input and output training data.

In output backpropagation, the computation of the prediction $\hat{c} = \mathbf{g}'(q, \hat{b}) = \mathbf{g}'(q, \mathbf{g}(p, a))$ is performed in the same way as in the CGGWZ algorithm. However, the computation of training is performed differently, in two steps as follows:

1. Training by $\langle\!| \mathbf{g}', \mathbf{p}' |\!\rangle : B \overset{Q}{\Rrightarrow} C$**:**

- **preparation:** There is no input training data for $\langle\!| \mathbf{g}', \mathbf{p}' |\!\rangle$, so we prepare it as the prediction $\hat{b} = \mathbf{g}(p, a)$ by $\mathbf{g}$.
- **training:** Then $\mathbf{p}'$ updates the parameter q to q' by using the training data $\hat{b}$ of input and c of output, i.e., q' is the Q-component of $\mathbf{p}'(q, \hat{b}, c)$.

2. Training by $\langle\!| \mathbf{g}, \mathbf{p} |\!\rangle : A \overset{P}{\Rrightarrow} B$**:**

- **preparation:** There is no output training data for $(\!|\, \mathbf{g}, \mathbf{p} \,|\!)$, so we prepare it as the "backpropagation" b' from c (by $\mathbf{p}'$ with q and $\hat{b}$), i.e., b' is the B-component of $\mathbf{p}'(q, \hat{b}, c)$.
- **training:** Then $\mathbf{p}$ updates the parameter p to p' by using the training data a of input and b' of output, i.e., p' is the P-component of $\mathbf{p}(p, a, b')$.

Above, we do not use the A-component of $\mathbf{p}(p, a, b')$, which is used if we further precompose a learner.

Thus, the concepts of a learner and output backpropagation are nothing but the category $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$, but we emphasize two points. One is, as explained above, it is important how to interpret a para lens as a learning algorithm; recall that CGGWZ algorithms have the different interpretation. The other is that we may further restrict the category $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$ to some subcategory, i.e., we may consider some condition on learners that is closed under composition (output backpropagation). In fact, we consider the GetPut law as such a condition in section 4.4.

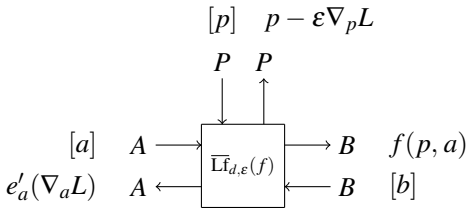
An important advantage of output backpropagation is that we can compose different styles of learning algorithms, such as a learner by quadratic error and a learner by softmax cross-entropy, or perhaps a Bayesian learner (which is not yet known to exist). We here emphasize that the concept of output backpropagation does not presuppose functoriality of a learner construction (such as the FST construction), although functoriality is helpful with output backpropagation, as used in section 6.

4.3 FST construction

We now define the FST algorithm, which is based on gradient-based learning, as follows:

Definition 31 (FST algorithms)**.** *Suppose a function $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ satisfies the condition that the derivative $\frac{\partial d(x,y)}{\partial x}(z, -) : \mathbb{R} \to \mathbb{R}$ has its inverse for any $z \in \mathbb{R}$. Also suppose we are given a learning rate ε and an architecture $(P, f) : A \to B$ where $(P, A, B) = (\mathbb{R}^l, \mathbb{R}^n, \mathbb{R}^m)$. Then the FST algorithm $\overline{\mathrm{Lf}}_{d,\varepsilon}(f)$ is the para lens $(P, (\!|\, f, \langle U_f, r_f \rangle \,|\!)) : A \overset{P}{\Rrightarrow} B$ where*

- *$U_f : P \times A \times B \to P$ is defined by $U_f(p, a, b) = p - \varepsilon \nabla_p L$,*
- *$r_f : P \times A \times B \to A$ is defined by $r_f(p, a, b) = e'_a(\nabla_a L)$ where $L = e(f(p, a), b)$,*



- *$e : B \times B \to \mathbb{R}$ is defined by $e(\widehat{b}, b) = \sum_i^m d(\widehat{b}_i, b_i)$,*
- *$e'_a : A \to A$ ($e'_a : \mathbb{R}^n \to \mathbb{R}^n$) is defined by $e'_a(x)_i = (\frac{\partial d(x,y)}{\partial x}(a_i, -))^{-1}(x_i)$ for each i where $(\frac{\partial d(x,y)}{\partial x}(a_i, -))^{-1} : \mathbb{R} \to \mathbb{R}$ is the inverse function of $\frac{\partial d(x,y)}{\partial x}(a_i, -) : \mathbb{R} \to \mathbb{R}$.*

We call the construction of these algorithms FST construction. ◁

The function $e : B \times B \to \mathbb{R}$ is used as a loss function in section 2.1 where the loss is a sum of losses calculated by $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ for each component. Therefore, the function $U_f : P \times A \times B \to P$ performs gradient descent. Regarding the behavior of $r_f : P \times A \times B \to A$, if $a' = e'_a(z)$, then $\frac{\partial d(x,y)}{\partial x}(a_i, a'_i) = z_i$. It means that the function $e'_a : A \to A$ modifies the input a to a' using z such that we can restore the gradient z from comparing a and a'. As we explain later, the intuition that the modification attempts to minimize the loss, just like gradient descent in U_f, is justified through output backpropagation.

In summary, for any training data of inputs $a \in A$, training data of desired output $b \in B$, and parameter $p \in P$, the computation of training is denoted by

$$\mathbf{p}(p, a, b) = (e'_a(\nabla_a L),\ p - \varepsilon \nabla_p L)$$

where $L = e(f(p, a), b)$.

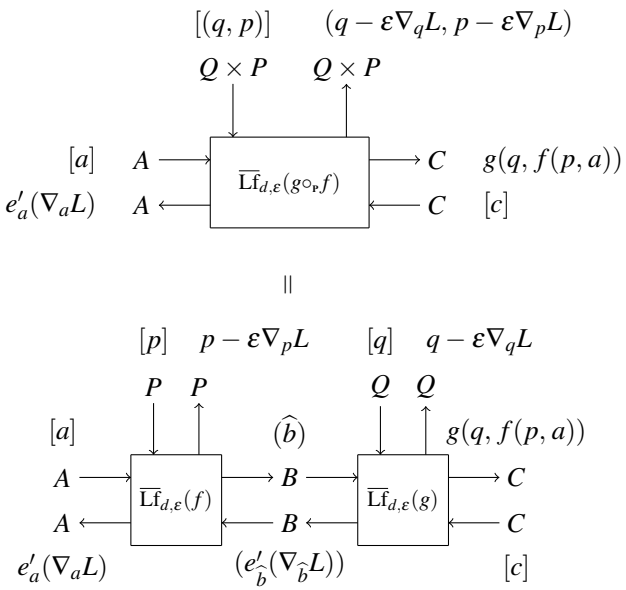
Definition 32 (Gradient descent/backpropagation functor)**.** *The correspondences from an architecture $(P, f) : A \to B$ to the FST algorithm $\overline{\mathrm{Lf}}_{d,\varepsilon}(f) : A \overset{P}{\Rightarrow} B$ become an identity-on-objects functor $\overline{\mathrm{Lf}}_{d,\varepsilon} : \mathbf{Para}(\mathbf{Smooth}) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{Smooth}))$, which Fong et al. (2019) call the* Gradient descent/backpropagation functor. ◁

To distinguish between learners and para lenses that are not learners, we use the following convention: we write the symbol of a functor that returns a learner with an overline, such as $\overline{\mathrm{Lf}}_{d,\varepsilon}$.

Regarding functoriality, the following equation holds for any architectures $(P, f) : A \to B$ and $(Q, g) : B \to C$.

$$\overline{\mathrm{Lf}}_{d,\varepsilon}(g \circ_{\mathbf{P}} f) = \overline{\mathrm{Lf}}_{d,\varepsilon}(g) \circ_{\mathbf{P}} \overline{\mathrm{Lf}}_{d,\varepsilon}(f).$$

The equation is depicted as follows,



where $\widehat{b} = f(p,a)$ and $L = e(g(q,f(p,a)),c)$.

The equation implies that even if the computation of the output backpropagation is complicated compared with the gradient descent described in section 2.1, the updated parameters are equal to those calculated by the gradient descent with architecture $g \circ_{\mathbf{r}} f$.

Note that the backpropagation described in section 2.1 and used in CGGWZ algorithms is also performed in the output backpropagation among FST algorithms; details are explained in Rmk. 39.

Remark 33. The functoriality of output backpropagation can explain the intuition of the input modification. In the above diagram, the modified prediction $e'_{\widehat{b}}(\nabla_{\widehat{b}}L)$ is used as a desired output in $\overline{\mathrm{Lf}}_{d,\varepsilon}(f)$. It is a strange situation, but the training is correctly done because of the functoriality. Therefore, it can be considered that $e'_{\widehat{b}}(\nabla_{\widehat{b}}L)$ is a modified value of $\widehat{b}$ to minimize the loss L, and we can regard $e'_{\widehat{b}}(\nabla_{\widehat{b}}L)$ as a desired output. ◁

We have to mention the problem of this functor in the original paper by Fong et al. (2019). As Cruttwell et al. (2022, Section 6) point out, the definition of the para lens in Fong et al. (2019) does not become a functor. Therefore, we avoid the problem by modifying the definition of the para construction.

Proposition 34. *For our definition of the para construction in Def. 17, the gradient descent/backpropagation functor preserves the equivalence relation.* ◁

Proof. We describe it in Ex. 49 by the generalized and modularized definition of the FST construction in Thm. 47 □

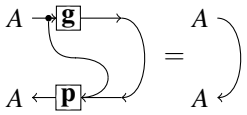
The algorithms defined by us and those defined by Fong et al. (2019) are the same except for the differences between the equivalence relations in the categories to which they belong. For a detailed explanation of well-definedness, see Rmk. 50.

4.4 GetPut law for a learning algorithm

In the bidirectional transformation, there are many laws on a lens such as the GetPut law (Foster et al., 2007; Riley, 2018). The *GetPut law* for a lens $(\!|\, \mathbf{g}, \mathbf{p} \,|\!)$ in an arbitrary cartesian category is defined by

$$\mathbf{p} \circ (\mathrm{id}_A \times \mathbf{g}) \circ \mathrm{copy} = \mathrm{id}_A : A \to A.$$

This is illustrated by the following diagram:



The GetPut law can be extended to a para lens $(P,\ (\!|\, \mathbf{g}, \mathbf{p} \,|\!) : P \otimes_{\mathbf{l}} A \to B)$: it satisfies *GetPut* if $(\!|\, \mathbf{g}, \mathbf{p} \,|\!)$ does. Fong and Johnson (2019) consider the GetPut law, which they call the I-UR law. We also introduce a weaker version of the GetPut law focusing only on the parameter object: A para lens $(P,\ (\!|\, \mathbf{g}, \mathbf{p} \,|\!) : P \otimes_{\mathbf{l}} A \to B)$ satisfies *pGetPut* if the following equation holds:

$$(\mathrm{id}_P \times !_A) \circ \mathbf{p} \circ (\mathrm{id}_{P \times A} \times \mathbf{g}) \circ \mathrm{copy}_{P \times A} = \mathrm{id}_P \times !_A.$$

When we consider a para lens in the cartesian category **Set**, the GetPut law says that, for any p, a, and b,

$$\mathbf{g}(p,a) = b \implies \mathbf{p}((p,a),b) = (p,a),$$

Meanwhile, the pGetPut law says that, for any p, a, and b, there exists a' such that

$$\mathbf{g}(p,a) = b \implies \mathbf{p}((p,a),b) = (p,a').$$

Fong and Johnson (2019) show that a para lens $\overline{\mathrm{Lf}}_{d,\varepsilon}(f)$ by the FST construction whose loss function is the quadratic error always satisfies the GetPut law. Recall that, in this case, the get function $\mathbf{g}$ is an architecture, and the put function $\mathbf{p}$ is a learning algorithm. Then the GetPut law (resp. pGetPut law) says that, if the prediction $\mathbf{g}(p,a)$ equals the desired output b, then the algorithm does not change the parameter p and the input a (resp. the parameter p). As a condition for a learning algorithm, pGetPut is natural, while GetPut is also important for output backpropagation. It is easy to check that the GetPut law is closed under the composition of para lenses, and also note that the pGetPut law is not closed.

Thus, we can consider the GetPut law as a condition that a learner must satisfy for a kind of sanity check. However, we do not say that GetPut must be satisfied absolutely in every learning situation, nor that this is sufficient for a sanity check.

Remark 35. We remark that, for a para lens given by the CGGWZ construction with quadratic error, (i) it does not satisfy pGetPut nor GetPut (ii) it satisfies the obvious condition that is essentially the same as pGetPut (iii) it does not satisfy the obvious condition that is essentially the same as GetPut. The difference on the third point between the FST and the CGGWZ constructions is that an FST algorithm uses e'_a while the CGGWZ algorithm does not and hence returns a gradient as the A-value, which is always zero when $\mathbf{g}(p,a) = b$. ◁

5. Modularized FST construction

In this section, we define a new construction of learning algorithms that combines the benefits of the CGGWZ construction of Cruttwell et al. (2022) with those of the FST construction. Thus, the construction is modularized in the CGGWZ-style and functorial.

First, we carefully observe differences between the CGGWZ algorithms and the FST algorithms. Next, as a decomposition step in the modularization process, we simplify and reorganize the construction of the FST algorithms and their properties such as functoriality and well-definedness in section 5.1.

In section 5.2, we generalize and formalize this modularization to the level of abstraction considered by Cruttwell et al. (2022), i.e., we define the construction on an arbitrary CRDC. We call the new construction *modularized FST construction*.

Using the modularization of the construction, we also consider the GetPut law for the modularized FST construction in section 5.3.

5.1 Reorganization of FST algorithm through CGGWZ construction

To reconfirm the differences between the CGGWZ algorithms and the FST algorithms, we consider the example of CGGWZ algorithm that outputs the same updated parameters as an FST algorithm:

Proposition 36. *Let a learning rate $\varepsilon \in \mathbb{R}$, an architecture $(P, f) : A \to B$, and a component-wise error function $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be given. Consider the following two learning algorithms:*

- ***The FST algorithm****: Defined from the error function $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, which gives rise to a total error function $e : B \times B \to \mathbb{R}$ and a put function $\mathbf{p}' : P \times A \times B \to P \times A$.*
- ***The CGGWZ algorithm*** *(Ex. 27): For this comparison, we define this algorithm with an error function $e' : B \times B \to \mathbb{R}$ where the arguments of e are swapped, i.e., $e'(b, \hat{b}) := e(\hat{b}, b)$. This yields the put function $\mathbf{p} : B \times P \times A \times 1 \to B \times P \times A$.*

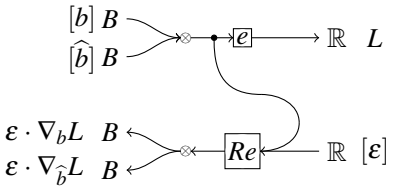
Then, for any parameter $p \in P$, input $a \in A$, and desired output $b \in B$, the parameter updates from both algorithms are identical. Specifically, their respective put functions satisfy:

$$\pi_1(\mathbf{p}(b, p, a)) = p - \varepsilon \cdot \nabla_p L = \pi_0(\mathbf{p}'(p, a, b))$$

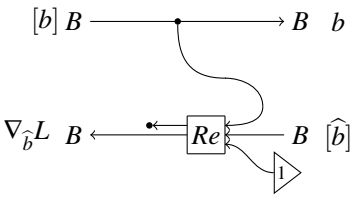
where $L = e(\hat{b}, b)$ with $\hat{b} = f(p, a)$. ◁

Note that the first variable of the loss functions in the CGGWZ algorithm receives a desired output, whereas the first variable of the loss functions in the FST algorithm receives a prediction.

This difference between the treatment of a loss function in CGGWZ algorithms and FST algorithms is important. In the CGGWZ construction, the lens $\langle\!| e, Re |\!\rangle : B \otimes_{\mathrm{L}} B \rightleftarrows \mathbb{R}$ computes the loss between prediction and desired output. For **Smooth**, for any $b, \widehat{b} \in B$ and $\varepsilon \in \mathbb{R}$, the get function outputs the loss $L = e(b, \widehat{b})$ and the put function outputs the gradients $\varepsilon \cdot \nabla_b L, \varepsilon \cdot \nabla_{\widehat{b}} L$.



This lens is easy to obtain by the functor Rd, but the two outputs L, $\varepsilon \cdot \nabla_b L$ by the lens are never used in prediction or training, and the input ε is just a constant value in gradient-based learning. Certainly, these input and outputs are omitted from the FST algorithms. Instead, an FST algorithm outputs a prediction rather than a loss, and it receives a desired output at the codomain of the algorithm, not at the parameter object. Notice that we can reshape $\mathrm{Rd}(e)$ into such a lens for calculating a loss gradient, which is similar to an FST algorithm, as follows.



From these observations, for any FST algorithm $\overline{\mathrm{Lf}}_{d,\varepsilon}(f) : A \overset{P}{\Rrightarrow} B$ (given in Def. 31), we can decompose the algorithm into the following four lenses in **Smooth**:

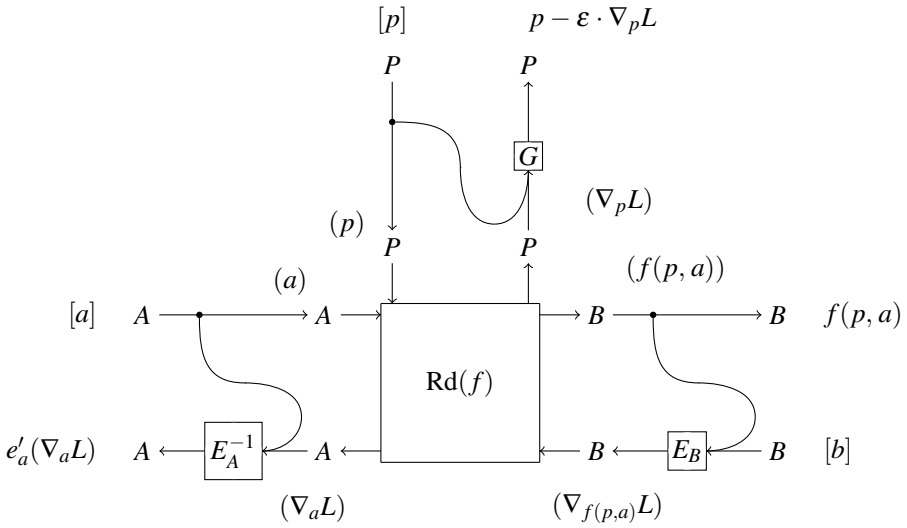
Proposition 37. *Suppose* $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, $\varepsilon \in \mathbb{R}$, *and* $(P, f) : A \to B$ *satisfy the assumptions of Def. 31. Then we have the following equation:*

$$\overline{\mathrm{Lf}}_{d,\varepsilon}(f) = \left(P, \langle\!| \mathrm{id}_B, E_B |\!\rangle \circ \mathrm{Rd}(f) \circ (\langle\!| \mathrm{id}_p, G |\!\rangle \otimes_\iota \langle\!| \mathrm{id}_A, E_A^{-1} |\!\rangle)\right) : A \overset{P}{\Rightarrow} B$$

where

$$E_B(\widehat{b}, b) = \nabla_{\widehat{b}} e(\widehat{b}, b) \text{ for any } b, \widehat{b} \in B,$$
$$E_A^{-1}(a, x) = e'_a(x) \text{ for any } a, x \in A,$$
$$G(p, x) = p - \varepsilon \cdot x \text{ for any } p, x \in P.$$

The construction of the right-hand side is depicted as follows where $L = e(f(p, a), b)$.



◁

Proof. This can be easily confirmed by the element-wise comparison of the algorithms. □

The lens $\langle\!| \mathrm{id}_P, G |\!\rangle$ is similar to a preoptimizer lens in Def. 26, but the learning rate is applied to the parameter during optimization. Thus, when we refer to an optimizer, the application of a learning rate is included.

In this way, we accomplish reorganizing the FST construction by dividing the features into lenses. Moreover, in considering the decomposition of properties, the following proposition is important:

Proposition 38. *For any FST construction from a functor* $\overline{\mathrm{Lf}}_{d,\varepsilon}$ *in Def. 32, two lenses* $\langle\!| \mathrm{id}_B, E_B |\!\rangle$, $\langle\!| \mathrm{id}_B, E_B^{-1} |\!\rangle : B \rightleftarrows B$ *are inverse to each other.* ◁

Proof. For any values $b, b' \in B$, the condition that $\langle\!| \mathrm{id}_B, E_B |\!\rangle$ and $\langle\!| \mathrm{id}_B, E_B^{-1} |\!\rangle$ are inverse to each other can be written as

$$E_B^{-1}(b, E_B(b, b')) = b', \; E_B(b, E_B^{-1}(b, b')) = b'.$$

Suppose $B = \mathbb{R}^n$ for some $n \in \mathbb{N}$, the i-th component of this equation can be proved as follows.

$$\begin{aligned} E_B^{-1}(b, E_B(b,b'))_i &= e'_{b_i}\left(\frac{\partial}{\partial b_i}\sum_k^n d(b_k, b'_k)\right) \\ &= e'_{b_i}\left(\frac{\partial}{\partial b_i} d(b_i, b'_i)\right) \end{aligned}$$

This formula can be written as follows because e_{b_i} is an inverse function.

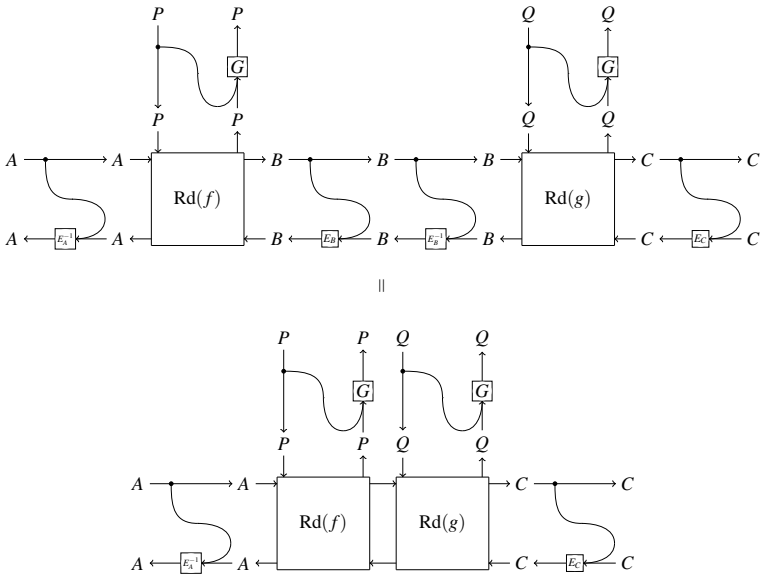
$$\begin{aligned} &= \left(\frac{\partial}{\partial b_i} d(b_i, -)\right)^{-1}\left(\frac{\partial}{\partial b_i} d(b_i, b'_i)\right) \\ &= b'_i. \end{aligned}$$

Similarly, the other equation can be proved as follows:

$$\begin{aligned} E_B(b, E_B^{-1}(b,b'))_i &= \frac{\partial}{\partial b_i}\sum_k^n d(b_k, e'_{b_k}(b'_k)) \\ &= \left(\frac{\partial}{\partial b_i} d\left(b_i, \left(\frac{\partial}{\partial b_i} d(b_i, -)\right)^{-1}(b'_i)\right)\right) \\ &= b'_i. \end{aligned}$$

□

By these isomorphisms, the unnecessary loss function in output backpropagation between $\mathrm{Rd}(f)$ and $\mathrm{Rd}(g)$ is canceled out as follows.



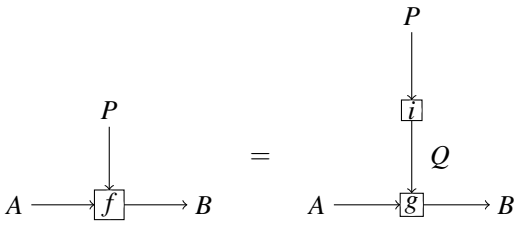
Remark 39. From the above equation, we can confirm that output backpropagation among FST algorithms performs backpropagation through the lens $\mathrm{Rd}(g) \circ_{\mathbf{P}} \mathrm{Rd}(f)$, which is also used in CGGWZ algorithms in Def. 26 and Ex. 23. ◁

Due to this isomorphism, we can easily describe the constraints on a loss function required for the construction to be a functor, and we can easily prove the functoriality of the FST construction from the isomorphisms and the functoriality of Rd.

Regarding the well-definedness, the functor $\overline{\mathrm{Lf}}_{d,\varepsilon} : \mathbf{Para}(\mathbf{Smooth}) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{Smooth}))$ must satisfy the following condition:

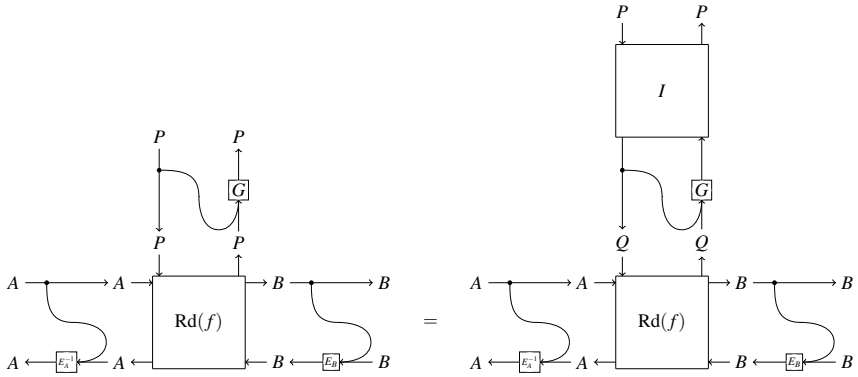
$$f \sim g \Longrightarrow \overline{\mathrm{Lf}}_{d,\varepsilon}(f) \sim \overline{\mathrm{Lf}}_{d,\varepsilon}(g).$$

We can also simplify and reorganize this discussion. If the equation in eq. (2) holds for some morphism $i : P \to Q$,



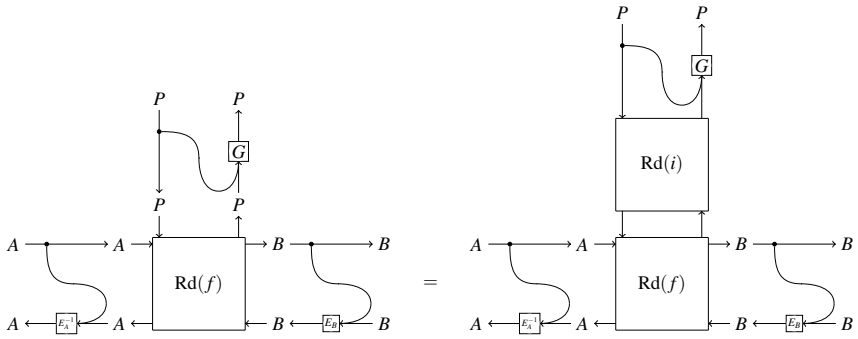
(2)

we need to prove that the equation in eq. (3) also holds for some lens $I : P \rightleftarrows Q$.



(3)

Let us apply the functor $\overline{\mathrm{Lf}}_{d,\varepsilon}$ to the architectures on both sides of eq. (2). Then, the following equation holds.



(4)

However, eq. (3) and eq. (4) differ in the position of $(\!|\, \mathrm{id}_p, G \,|\!)$. Therefore, the well-definedness of the learner construction arises from the compatibility between i, I (defined in para construction), and lens $(\!|\, \mathrm{id}, G \,|\!)$.

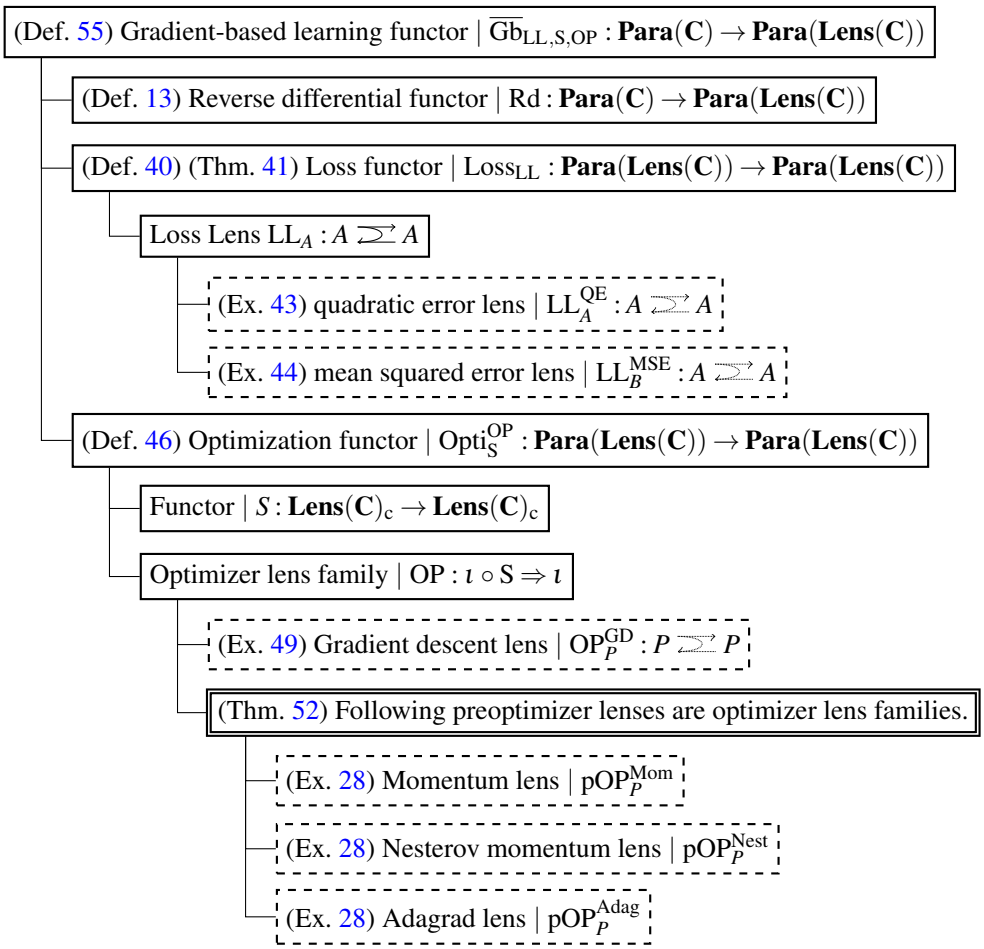
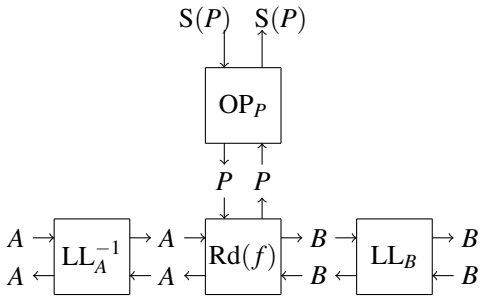


Figure 3. Components of the gradient-based learning functor

5.2 Generalization of FST construction

We aim to abstract the decomposed computations in FST algorithm using the results in Prop. 37. Specifically, we generalize the lenses $(\!|\, \mathrm{id}_P, G \,|\!)$, $(\!|\, \mathrm{id}_A, E_A^{-1} \,|\!)$, and $(\!|\, \mathrm{id}_B, E_B \,|\!)$ as follows.



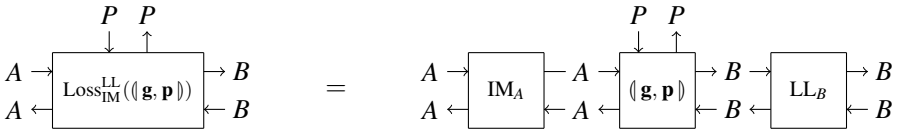
However, the mathematical definition and discussion of the new construction are complicated. Therefore, we also decompose the construction into three functors; the structure of the construction is summarized in fig. 3. As shown in the figure, we can now utilize the optimizers described in Ex. 28 in our construction. Of the three functors, the first is the reverse differential functor $\mathbf{Para}(\mathrm{Rd}) : \mathbf{Para}(\mathbf{C}) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$ defined in Ex. 22, and the second is a functor $\mathrm{Loss} : \mathbf{Para}(\mathbf{Lens}(\mathbf{C})) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$ that appends a lens calculating a loss gradient and its inverse, and the third is a functor $\mathrm{Opti} : \mathbf{Para}(\mathbf{Lens}(\mathbf{C})) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$ that appends an optimizer forming a lens. Note that, in constructing a learning algorithm, the order of application of the two functors, Loss and Opti, is interchangeable. This is because the two functors append lenses to an input para lens on different sides.

5.2.1 Loss functor

We define the second functor, which arises from the abstraction of E_B and E_A^{-1} in Prop. 37, as follows. We start with a slightly general definition; the extra generality is not used here but used from section 5.3 for the discussion of the Softmax cross-entropy in section 6.

Definition 40 (Loss functor)**.** *For any CRDC* $\mathbf{C}$*, we suppose two families of lenses* $\mathrm{LL}_A : A \overset{\rightarrow}{\leftarrow} A$ *(called* Loss Lens*) and* $\mathrm{IM}_A : A \overset{\rightarrow}{\leftarrow} A$ *(called* Input Modifier*) in* $\mathbf{Lens}(\mathbf{C})$ *indexed by objects* A *in* $\mathbf{Lens}(\mathbf{C})$*. Then, a family of maps* $(\mathrm{Loss}^{\mathrm{LL}}_{\mathrm{IM}})_{A,B} : \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))(A,B) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))(A,B)$ *is defined by*

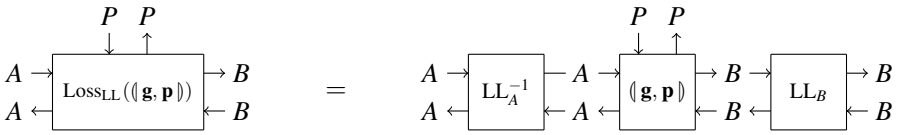
$$(\mathrm{Loss}^{\mathrm{LL}}_{\mathrm{IM}})_{A,B}(P, (\!|\, \mathbf{g}, \mathbf{p} \,|\!)) \quad = \quad (I_L, \mathrm{LL}_B) \circ_P (P, (\!|\, \mathbf{g}, \mathbf{p} \,|\!)) \circ_P (I_L, \mathrm{IM}_A)$$



for $(P, (\!|\, \mathbf{g}, \mathbf{p} \,|\!)) : A \overset{\rightarrow}{\leftarrow} B$ *in* $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$*. We may write* $(\mathrm{Loss}^{\mathrm{LL}}_{\mathrm{IM}})_{A,B}$ *as* $\mathrm{Loss}^{\mathrm{LL}}_{\mathrm{IM}}$ *for brevity.*

Then, for a family of isomorphic lenses $\mathrm{LL}_A : A \overset{\rightarrow}{\leftarrow} A$ *in* $\mathbf{Lens}(\mathbf{C})$ *indexed by objects* A *in* $\mathbf{Lens}(\mathbf{C})$*, an identity-on-objects symmetric strong monoidal functor* $\mathrm{Loss}_{\mathrm{LL}} : \mathbf{Para}(\mathbf{Lens}(\mathbf{C})) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$ *is defined as follows:*

$$\mathrm{Loss}_{\mathrm{LL}} = \mathrm{Loss}^{\mathrm{LL}}_{\mathrm{LL}^{-1}}.$$



The isomorphisms μ *and* η *of the strong monoidal functor are defined as*

$$\mu = \mathrm{LL}_{A \otimes_P B} \circ (\mathrm{LL}_A^{-1} \otimes_P \mathrm{LL}_B^{-1}) : \mathrm{Loss}_{\mathrm{LL}}(A) \otimes_P \mathrm{Loss}_{\mathrm{LL}}(B) \overset{\rightarrow}{\leftarrow} \mathrm{Loss}_{\mathrm{LL}}(A \otimes_P B),$$

$$\eta = \mathrm{LL}_{I_P} : I_P \overset{\rightarrow}{\leftarrow} \mathrm{Loss}_{\mathrm{LL}}(I_P).$$

◁

Note that we do not assume the naturality for the lens families LL and IM. In Section 6, Cruttwell et al. (2022) give a construction similar to $\mathrm{Loss}^{\mathrm{LL}}_{\mathrm{IM}}$ in relation to *deep dreaming*. In mathematics, the enclosing operation $\mathrm{Loss}_{\mathrm{LL}}$ is akin to the conjugate action.

Theorem 41. *The loss functor* $\mathrm{Loss}_{\mathrm{LL}}$ *preserves the para composition of lenses, the identities, and the monoidal product.* ◁

Proof. We confirm the functoriality of the functor. For any para lenses $(P, F) : A \rightrightarrows B$ and $(Q, G) : B \rightrightarrows C$, the equation

$$\begin{aligned}
&\mathrm{Loss}_{\mathrm{LL}}(G) \circ \mathrm{Loss}_{\mathrm{LL}}(F) \\
=&\mathrm{LL}_C \circ_{\mathbf{P}} G \circ_{\mathbf{P}} \mathrm{LL}_B^{-1} \circ_{\mathbf{P}} \mathrm{LL}_B \circ_{\mathbf{P}} F \circ_{\mathbf{P}} \mathrm{LL}_A^{-1} \\
=&\mathrm{LL}_C \circ_{\mathbf{P}} G \circ_{\mathbf{P}} F \circ_{\mathbf{P}} \mathrm{LL}_A^{-1} \\
=&\mathrm{Loss}_{\mathrm{LL}}(G \circ F)
\end{aligned}$$

holds. Also, the equation

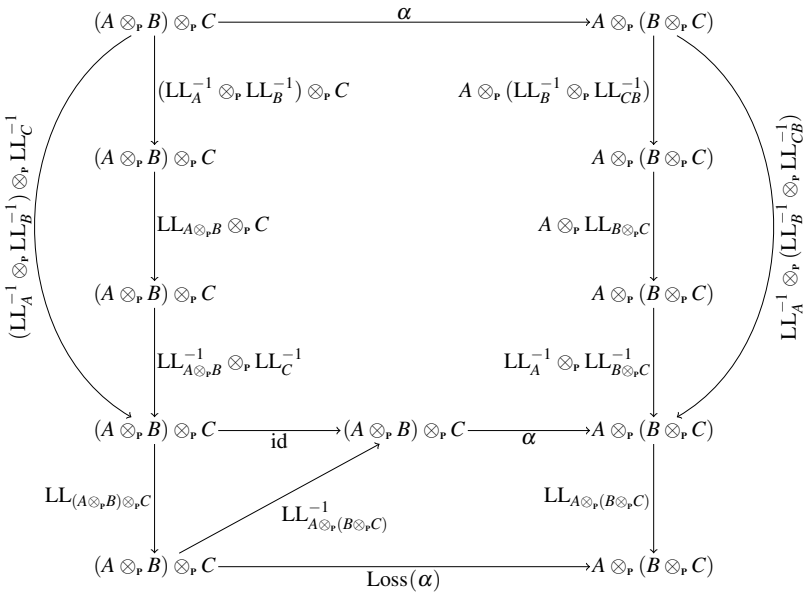
$$\mathrm{Loss}_{\mathrm{LL}}(\mathrm{id}_A) = \mathrm{LL}_A \circ_{\mathbf{P}} \mathrm{LL}_A^{-1} = \mathrm{id}_A$$

holds.

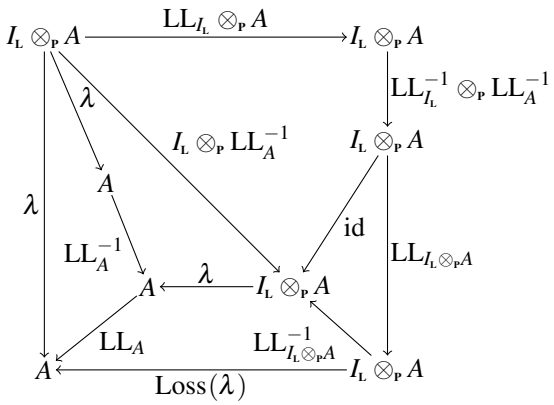
Next, we confirm the naturality of the isomorphism μ. For any para lenses $(P, F) : A \rightrightarrows B$ and $(Q, G) : C \rightrightarrows D$, the following equation holds.

$$\begin{aligned}
&(\mathrm{LL}_B \otimes_{\mathbf{P}} \mathrm{LL}_D) \circ_{\mathbf{P}} \mathrm{LL}^{-1}_{B \otimes_{\mathbf{P}} D} \circ_{\mathbf{P}} \mathrm{Loss}_{\mathrm{LL}}(F \otimes_{\mathbf{P}} G) \\
&= (\mathrm{LL}_B \otimes_{\mathbf{P}} \mathrm{LL}_D) \circ_{\mathbf{P}} (F \otimes_{\mathbf{P}} G) \circ_{\mathbf{P}} \mathrm{LL}^{-1}_{A \otimes_{\mathbf{P}} C} \\
&= (\mathrm{Loss}_{\mathrm{LL}}(F) \otimes_{\mathbf{P}} \mathrm{Loss}_{\mathrm{LL}}(G)) \\
&\circ_{\mathbf{P}} (\mathrm{LL}_A \otimes_{\mathbf{P}} \mathrm{LL}_C) \circ_{\mathbf{P}} \mathrm{LL}^{-1}_{A \otimes_{\mathbf{P}} C}.
\end{aligned}$$

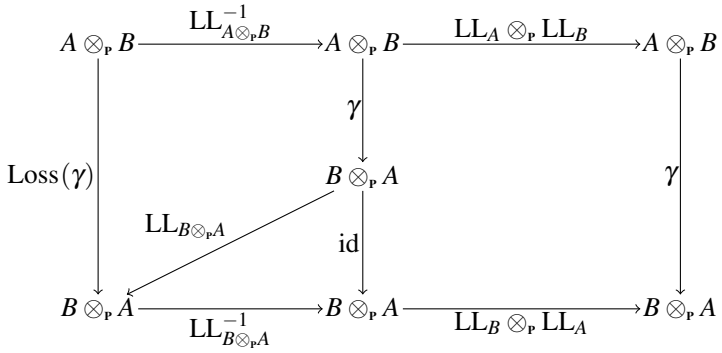
This functor preserves the associator because the following diagram commutes.



This functor preserves the left unitor because the following diagram commutes.



The right unitors are also preserved. This functor is symmetric, namely, the functor preserves the symmetries because the following diagram commutes.



□

Remark 42. We can derive that the loss functor is strong monoidal from the invertibility of loss lenses, even if the loss lenses do not respect monoidal product. However, if we assume that the loss lenses respect monoidal product, namely, a family of loss lenses becomes a monoidal transformation from some strict monoidal functor to some strict monoidal functor, the loss functor becomes strict monoidal. ◁

Example 43 (Quadratic error (loss) lens)**.** *We call the loss lenses defined by* $E_B : B \rightleftarrows B$ *and* $E_A^{-1} : A \rightleftarrows A$ *in Prop. 37 whose loss is the quadratic error the* quadratic error (loss) lens $\mathrm{LL}^{\mathrm{QE}}$. *Since the put function is defined by*

$$\mathbf{p}(\widehat{b}, b) = \nabla_{\widehat{b}} e^{\mathrm{QE}}(\widehat{b}, b) = \widehat{b} - b,$$

the quadratic error lens induces the functor, and μ *and* η *are the identities; see (Fong et al., 2019, Example III.4) for details.* ◁

Example 44 (Mean squared error (loss) lens)**.** *As a benefit of the modularization of the FST construction, we can utilize the* mean squared error *in a natural way. The mean squared error*

$e^{\mathrm{MSE}} : B \times B \to \mathbb{R}$ *is almost the same as the quadratic error, which is defined as follows:*

$$e^{\mathrm{MSE}}(x, y) = \frac{2}{m} e^{\mathrm{QE}}(x, y),$$

where $m \in \mathbb{N}$ *such that* $B = \mathbb{R}^m$. *Specifically, we can define the* mean squared error (loss) lens *as follows:*

$$\mathrm{LL}_B^{\mathrm{MSE}} = (\!| \, \mathrm{id}_B, \mathbf{p} \, |\!),$$

where $\mathbf{p}(\widehat{b}, b) = \nabla_{\widehat{b}} e^{\mathrm{MSE}}(\widehat{b}, b) = \frac{2}{m} \cdot (\widehat{b} - b)$. *Note that we cannot define mean squared error through the component-wise loss calculation* $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ *because of the dimension* m *of the vector space* B. *Therefore, mean squared error cannot be used in the FST construction. Indeed, these lenses do not respect monoidal product, unlike the quadratic error loss lens, as follows:*

$$\mathrm{LL}_A^{\mathrm{MSE}} \otimes_\iota \mathrm{LL}_B^{\mathrm{MSE}} \neq \mathrm{LL}_{A \otimes_\iota B}^{\mathrm{MSE}}, \textit{ in general.}$$

However, the lens $\mathrm{LL}_A^{\mathrm{MSE}}$ *has an inverse. Therefore, although it is not strict monoidal, we can obtain a strong monoidal loss functor using this lens.* ◁

Remark 45. Theorem A.1 of Fong et al. (2019) shows that the FST construction is generalized to handle losses with objectwise scalar multiplication such as $1/m$ where m is the dimension of the vector space. This generalization allows an FST construction to utilize the mean squared error. However, our construction naturally extends this fact by using loss lenses that do not necessarily respect the monoidal product. ◁

5.2.2 Optimization functor

We define the third functor, which arises from the generalization of $G : P \rightrightarrows P$ in Prop. 37, and we also consider the state objects of the preoptimizers:

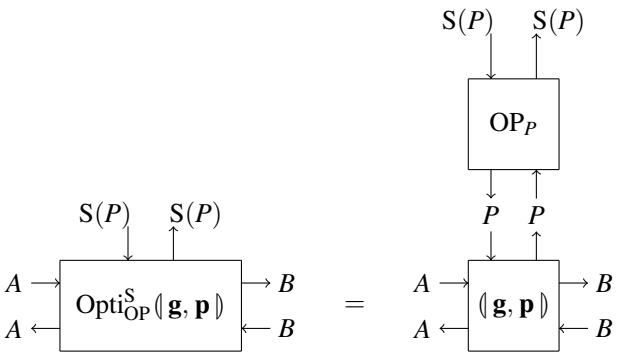
Definition 46 (Optimization functor)**.** *For CRDC* $\mathbf{C}$, *let* $\mathbf{Lens}(\mathbf{C})_{\mathrm{c}}$ *be the wide subgroupoid of* $\mathbf{Lens}(\mathbf{C})$ *in Ex. 15, and let* $\iota : \mathbf{Lens}(\mathbf{C})_{\mathrm{c}} \to \mathbf{Lens}(\mathbf{C})$ *be the inclusion functor in Def. 14. We also assume the following functor and the lenses (without parameters),*

- *a symmetric strong monoidal functor* $\mathrm{S} : \mathbf{Lens}(\mathbf{C})_{\mathrm{c}} \to \mathbf{Lens}(\mathbf{C})_{\mathrm{c}}$ *called* state functor,
- *a monoidal natural transformation of lenses* $\mathrm{OP} : \iota \circ \mathrm{S} \Rightarrow \iota$, *called* optimizer lens family,

Then, an identity-on-objects symmetric strict monoidal functor $\mathrm{Opti}_{\mathrm{OP}}^{\mathrm{S}} : \mathbf{Para}(\mathbf{Lens}(\mathbf{C})) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$, *which is called the* Optimization functor, *is defined as follows:*

$$\mathrm{Opti}_{\mathrm{OP}}^{\mathrm{S}}(P, (\!| \, \mathbf{g}, \mathbf{p} \, |\!)) = (\mathrm{S}(P), (\!| \, \mathbf{g}, \mathbf{p} \, |\!) \circ (\mathrm{OP}_P \otimes_\iota \mathrm{id}_A))$$

where $(P, (\!| \, \mathbf{g}, \mathbf{p} \, |\!)) : A \rightrightarrows B$ *is a morphism in* $\mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$.



◁

By restricting the domain and the codomain of the functor S, natural isomorphisms $\eta : I_{\text{L}} \to \mathrm{S}(I_{\text{L}})$ and $\mu_{A,B} : \mathrm{S}(A) \otimes_{\text{L}} \mathrm{S}(B) \to \mathrm{S}(A \otimes_{\text{L}} B)$ are restricted to canonical ones. Similarly, the type of the optimizer lens family $\mathrm{OP} : \iota \circ \mathrm{S} \Rightarrow \iota$ ensures that the lenses have naturality with respect to canonical morphisms in $\mathbf{Lens}(\mathbf{C})$. The monoidality of the family indicates compatibility between $\mathrm{OP}_P \otimes_{\text{L}} \mathrm{OP}_Q$ and $\mathrm{OP}_{P \otimes_{\text{L}} Q}$.

We need to check that the mapping of morphisms in the functor is well-defined:

Theorem 47. *The optimization functor preserves the equivalence relations.* ◁

Proof. To prove the preservation of the equivalence relations, we suppose a lens (canonical morphism) $i : P \leftrightarrows Q$ such that $g \circ (i \otimes_{\text{L}} A) = f$ for given $f : P \times A \leftrightarrows B, g : Q \times A \leftrightarrows B$. It is required to show the existence of a lens $i' : \mathrm{S}(P) \leftrightarrows \mathrm{S}(Q)$ such that $\mathrm{Opti}^{\mathrm{OP}}_{\mathrm{S}}(g) \circ (i' \otimes_{\text{L}} A) = \mathrm{Opti}^{\mathrm{OP}}_{\mathrm{S}}(f)$. By the definition of the functor, this equation is expanded to

$$g \circ (\mathrm{OP}_Q \otimes_{\text{L}} A) \circ (i' \otimes_{\text{L}} A) = f \circ (\mathrm{OP}_P \otimes_{\text{L}} A).$$

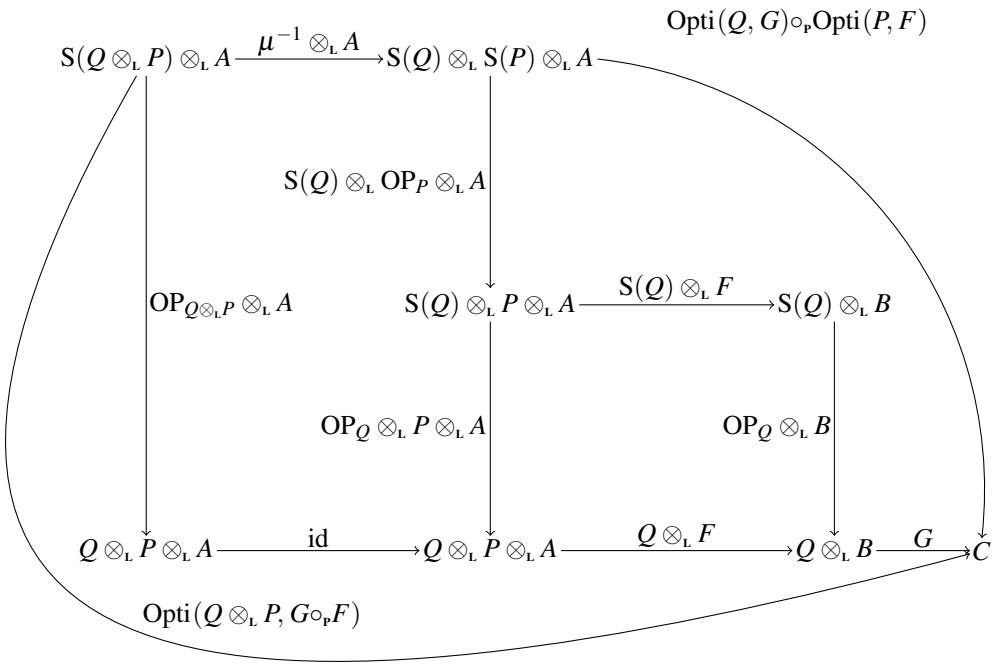
This equation holds for $i' = \mathrm{S}(i)$, because of the naturality of OP:

$$\mathrm{OP}_Q \circ \mathrm{S}(i) = i \circ \mathrm{OP}_P. \tag{5}$$

□

Theorem 48. *The optimization functor preserves the para composition, the identities, and the monoidal structure.* ◁

Proof. We confirm that the optimization functor preserves para composition. One issue is that the parameter object $\mathrm{S}(Q \otimes_{\text{L}} P)$ in $\mathrm{Opti}^{\mathrm{OP}}_{\mathrm{S}}(Q \otimes_{\text{L}} P, G \circ_{\text{P}} F)$ and the parameter object $\mathrm{S}(P) \otimes_{\text{L}} \mathrm{S}(Q)$ in $\mathrm{Opti}^{\mathrm{OP}}_{\mathrm{S}}(Q, G) \circ_{\text{P}} \mathrm{Opti}^{\mathrm{OP}}_{\mathrm{S}}(P, F)$ are not equal. However, because μ of the strong monoidal functor S are canonical, the parameter objects $\mathrm{S}(Q) \otimes_{\text{L}} \mathrm{S}(P)$ and $\mathrm{S}(Q \otimes_{\text{L}} P)$ in the para morphisms are identified by the equivalence relation. Precisely, for any para lens $(P, F) : A \leftrightarrows B$ and $(Q, G) : B \leftrightarrows C$, the equation
$\mathrm{Opti}^{\mathrm{OP}}_{\mathrm{S}}(Q \otimes_{\text{L}} P, G \circ_{\text{P}} F) =$
$\mathrm{Opti}^{\mathrm{OP}}_{\mathrm{S}}(Q, G) \circ_{\text{P}} \mathrm{Opti}^{\mathrm{OP}}_{\mathrm{S}}(P, F) \circ (\mu_{Q,P} \otimes_{\text{L}} A)$ holds because the following diagram commutes.



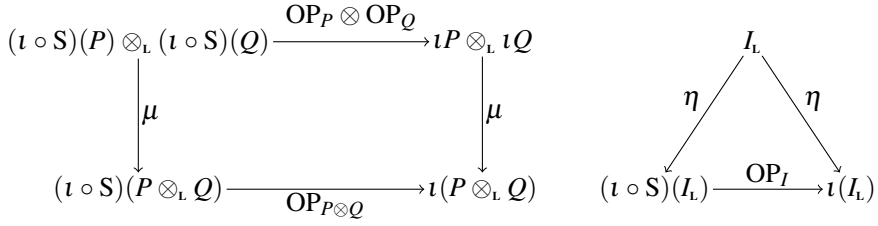
Similarly, preservation of identity is also satisfied. The mappings of the associators α, the unitors λ, ρ, and the symmetries γ are identities because their parameter objects are the unit I_{L} and $\eta : I_{\mathrm{L}} \to \mathrm{S}(I_{\mathrm{L}})$ is canonical. For these reasons, this functor is strict monoidal. □

Example 49 (Gradient descent lens)**.** *Given $P = \mathbb{R}^n \in$ **Smooth**, we define $\mathrm{OP}_P^{\mathrm{GD}}$ as the lens $(\!| \mathrm{id}_P, G |\!) : P \rightleftarrows P$ where $G : P \times P \to P$ is given in Prop. 37, and we call this a* gradient descent lens*. These lenses form an optimizer lens family, and the family gives rise to the optimization functor $\mathrm{Opti}^{\mathrm{id}_{\mathbf{C}}}_{\mathrm{OP}^{\mathrm{GD}}}$. The details are explained below.* ◁

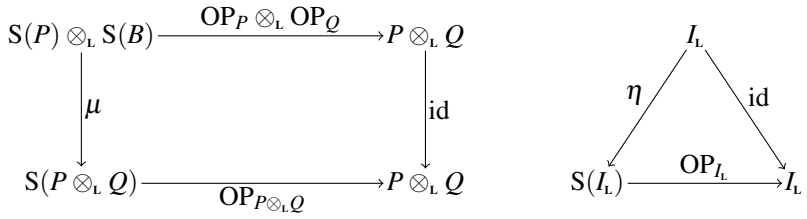
As in the case of the FST construction, the optimization method is restricted to gradient descent. However, we use various optimizers if they form optimizer lens families.

Remark 50. As explained in section 3 and section 4.3, unlike the CGGWZ construction, the FST construction has the problem of well-definedness. Specifically, the original FST construction of Fong et al. (2019) does not preserve the equivalence relation. The key point is the use of the naturality in eq. (5) in the proof of Thm. 47 above. With our definition, the gradient descent lenses satisfy the naturality as explained in Ex. 49, where the restriction of the naturality by the inclusion $\iota : \mathbf{Lens}(\mathbf{C})_{\mathrm{c}} \to \mathbf{Lens}(\mathbf{C})$ was important (recall that $\mathrm{OP} : \iota \circ \mathrm{S} \Rightarrow \iota$ from Def. 46). On the other hand, the construction of Fong et al. (2019) requires the naturality of $\mathrm{OP}^{\mathrm{GD}}$ on a lens $(\!| f, !_P \times f^{-1} |\!)$ for any isomorphism f in $\mathbf{C}$ (not only the canonical one), which fails. This is why we defined the new para construction with the canonical isomorphisms in $\mathbf{Lens}(\mathbf{C})_{\mathrm{c}}$. ◁

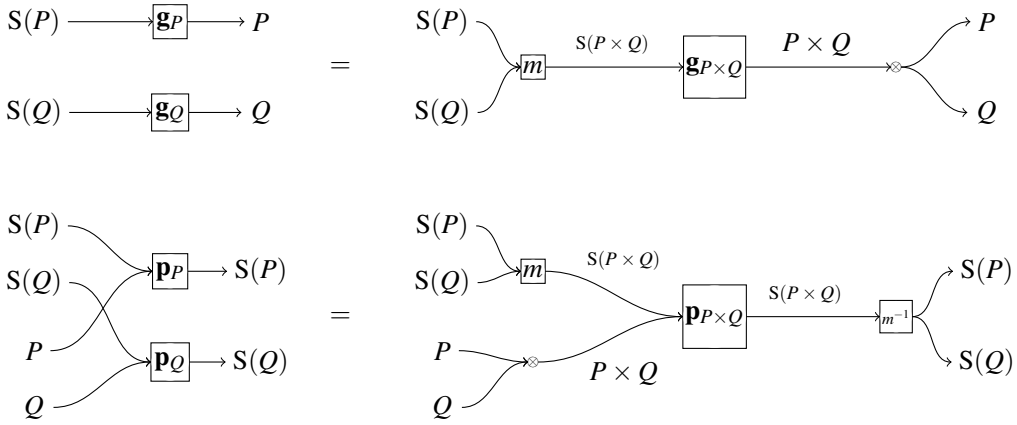
Remark 51. It is tedious to show that a transformation is monoidal and natural. However, we can use Prop. 16 to derive naturality from monoidality. Therefore, it suffices to show only that the family of lenses OP is monoidal, as follows.



The diagram can be expanded as follows.



Because the lens $\mu_{A,B} : \mathrm{S}(A) \otimes_{\mathrm{L}} \mathrm{S}(A) \to \mathrm{S}(A \otimes_{\mathrm{L}} B)$ is canonical, we can write $\mu_{A,B}$ as $(\!|\, m, !_A \times m^{-1} \,|\!)$ and also η as $(\!|\, e, !_A \times e^{-1} \,|\!)$ (by Ex. 15). Therefore, the equations consisting of get morphisms and put morphisms to satisfy the diagram on the left above diagram can be depicted as follows, (the right above diagram obviously commutes because I_{L} is the terminal object in $\mathbf{C}$.)



where $(\!|\, \mathbf{g}_X, \mathbf{p}_X \,|\!) = \mathrm{OP}_X$ for any $X \in \mathbf{C}$. The above equation in the string diagram indicates that $\mathbf{g}_P \times \mathbf{g}_Q$ and $\mathbf{g}_{P\times Q}$ are compatible through m. The below equation in the string diagram indicates that $\mathbf{p}_P \times \mathbf{p}_Q$ and $\mathbf{p}_{P\times Q}$ are compatible through m, m^{-1} and γ (γ comes from the monoidal product of lenses in Def. 10). Specifically, we append canonical isomorphisms m and m^{-1} as type conversions to the input and the output of $\mathbf{g}_{P\times Q}$ and $\mathbf{p}_{P\times Q}$. Then these morphisms are equal to the monoidal product of get morphisms or put morphisms. ◁

Although we explain the monoidality of an optimizer lens family, the monoidality of (pre)optimizers in this paper can be easily shown. As in the case of the *gradient descent lens* in Ex. 49, the monoidality of an optimizer lens family often holds if each lens consists of an n-times product of a function in **Smooth**. In this way, we can define optimizer lens families from the preoptimizers in Ex. 28 (Momentum, Nesterov Momentum, Adagrad) with the application of a learning rate.

Theorem 52. *For any learning rate* $\varepsilon \in \mathbb{R}$ *and preoptimizer* $\mathrm{pOP}^* \in \{\mathrm{pOP}^{\mathrm{Mom}}, \mathrm{pOP}^{\mathrm{Nest}}, \mathrm{pOP}^{\mathrm{Adag}}\}$ *in Ex. 28, we can construct a state functor* S *and an optimizer lens family* OP^* *(Def. 46) as follows:*

$$\begin{aligned}
&\mathrm{S}(P) = P \otimes_{\mathrm{L}} P \qquad \text{for any object } P \text{ in } \mathbf{Lens}(\mathbf{Smooth})_{\mathrm{c}},\\
&\mathrm{S}(F) = F \otimes_{\mathrm{L}} F \qquad \text{for any canonical isomorphism } F : P \xrightarrow{\sim} Q \text{ in } \mathbf{Lens}(\mathbf{Smooth})_{\mathrm{c}},\\
&\mathrm{OP}^*_P = (\!|\, \mathrm{id}_P, \mathbf{p}_\varepsilon \,|\!) \circ \mathrm{pOP}^*_P : \mathrm{S}(P) \xrightarrow{\sim} P,
\end{aligned}$$

where $\mathrm{pOP}^*_P : P \otimes_{\mathrm{L}} P \xrightarrow{\sim} P$, $(\!|\, \mathrm{id}_P, \mathbf{p}_\varepsilon \,|\!) : P \xrightarrow{\sim} P$, *and* $\mathbf{p}_\varepsilon(p, p') = -\varepsilon \cdot p'$. *The strong monoidal structure* μ *of the functor* S *is given by the "swap" canonical isomorphism*

$$\mu_{P,Q} : (P \otimes_{\mathrm{L}} P) \otimes_{\mathrm{L}} (Q \otimes_{\mathrm{L}} Q) \xrightarrow{\sim} (P \otimes_{\mathrm{L}} Q) \otimes_{\mathrm{L}} (P \otimes_{\mathrm{L}} Q).$$

◁

Proof. To prove that the lenses are optimizer lens families, we first show that the lenses are monoidal transformations, and then we use Prop. 16. For any $P = \mathbb{R}^n$ and optimizer lens $\mathrm{OP}^*_P : \mathrm{S}(P) \xrightarrow{\sim} P$, the i-th output of the get function $\mathbf{g}(s, p)_i$ and the i-th outputs of the put function s'_i and p'_i ($(s', p') = \mathbf{p}(s, p, x)$) are calculated from the i-th inputs s_i, p_i, and x_i for each $i \leq n$. Since the calculations of these lenses are independent of each component, analogous to parallel computation, the following equation holds up to canonical isomorphisms μ and η.

$$\mathrm{OP}^*_P = \overbrace{\mathrm{OP}^*_{\mathbb{R}} \otimes_{\mathrm{L}} \cdots \otimes_{\mathrm{L}} \mathrm{OP}^*_{\mathbb{R}}}^{n \text{ times}} : \mathrm{S}(P) \xrightarrow{\sim} P$$

Therefore, the optimizer lenses are monoidal transformations. In this way, they become optimizer lens families. □

Remark 53. One might wonder whether, as in Cruttwell et al. (2022), we could take the approach that we assume **C** to be a symmetric *strict* monoidal category and do not take the quotient by the equivalence relation in Def. 17. Although the strict approach works well for their purpose, namely, the non-functorial construction of algorithms, it does not work for our functorial construction. As in the proof of Thm. 48, the functoriality requires us to equate a morphism with parameter object $\mathrm{S}(P) \otimes_{\mathrm{L}} \mathrm{S}(Q)$ and that with $\mathrm{S}(P \otimes_{\mathrm{L}} Q)$. In our approach taking the quotient, this is done by composing the strong monoidal structure, which is the swap $\mu_{P,Q} : (P \otimes_{\mathrm{L}} P) \otimes_{\mathrm{L}} (Q \otimes_{\mathrm{L}} Q) \xrightarrow{\sim} (P \otimes_{\mathrm{L}} Q) \otimes_{\mathrm{L}} (P \otimes_{\mathrm{L}} Q)$ in the case of Thm. 52. On the other hand, in the approach without taking the quotient, this requires the two morphisms to be equal on the nose; for this, the strong monoidal structure must be taken as the identity, but this is not guaranteed merely by the strictness assumption, since the swap need not be the identity. ◁

Remark 54. Cruttwell et al. (2022) gave five examples of optimizers for the category **Smooth**, and the remaining one that has not yet been discussed so far is an optimizer called *Adam*, where the state has a record of the number of training iterations. Unfortunately, we cannot currently handle this last example in our functorial approach; for this we may need a more sophisticated way to handle such a history state. ◁

5.2.3 Gradient-based learning functor

Using the three functors defined above, we introduce a new functor that constructs modularized FST algorithms from arbitrary para morphisms:

Definition 55 (Modularized FST construction)**.** *For any CRDC* $\mathbf{C}$*, we define the family of maps*

$$(\overline{\mathrm{Gb}}^{\mathrm{LL}}_{\mathrm{IM},\mathrm{S},\mathrm{OP}})_{A,B} = (\mathrm{Opti}^{\mathrm{S}}_{\mathrm{OP}})_{A,B} \circ (\mathrm{Loss}^{\mathrm{LL}}_{\mathrm{IM}})_{A,B} \circ \mathrm{Rd}_{A,B}$$
$$: \mathbf{Para}(\mathbf{C})(A,B) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))(A,B)$$

where $\mathrm{Opti}^{\mathrm{S}}_{\mathrm{OP}}$ *is the optimization functor in Def. 46,* $\mathrm{Loss}^{\mathrm{LL}}_{\mathrm{IM}}$ *is the family of maps in Def. 40, and* Rd *is the reverse differential functor in Def. 13. We may write* $(\overline{\mathrm{Gb}}^{\mathrm{LL}}_{\mathrm{IM},\mathrm{S},\mathrm{OP}})_{A,B}$ *as* $\overline{\mathrm{Gb}}^{\mathrm{LL}}_{\mathrm{IM},\mathrm{S},\mathrm{OP}}$ *for brevity. Then, gradient-based learning functors* $\overline{\mathrm{Gb}}_{\mathrm{LL},\mathrm{S},\mathrm{OP}} : \mathbf{Para}(\mathbf{C}) \to \mathbf{Para}(\mathbf{Lens}(\mathbf{C}))$ *are defined by* $\overline{\mathrm{Gb}}_{\mathrm{LL},\mathrm{S},\mathrm{OP}} = \overline{\mathrm{Gb}}^{\mathrm{LL}}_{\mathrm{LL}^{-1},\mathrm{S},\mathrm{OP}}$. ◁

We often omit the subscripts LL, S, OP of $\overline{\mathrm{Gb}}_{\mathrm{LL},\mathrm{S},\mathrm{OP}}$ and write $\overline{\mathrm{Gb}}$. We call the construction by the mapping of morphisms *modularized FST construction*, and we call the algorithm by this construction *modularized FST algorithm*.

As a final check, we confirm that gradient-based learning functor is a generalization of gradient descent/backpropagation functor in Def. 32.

Theorem 56. *Gradient descent/backpropagation functor in Def. 32 is a gradient-based learning functor, where* $\mathrm{OP} = \mathrm{OP}^{\mathrm{GD}}$*,* $\mathrm{LL} = \mathrm{LL}^{\mathrm{QE}}$*, and* $\mathrm{S} = \mathrm{id}$*.* ◁

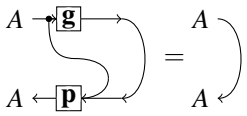
Due to the modularized FST construction, we accomplish modularization and generalization of FST algorithms while keeping their functoriality.

5.3 GetPut law in modularized FST algorithms

We also discuss the GetPut law of the modularized FST algorithms. In particular, through the CGGWZ-style modularizations, we clarify the necessary and sufficient conditions for modularized FST algorithms to satisfy the GetPut law. First, building on a result of Boisseau et al. (2023), we can write the GetPut law as a lens equation:

Proposition 57. *For any lens* $(\!| \mathbf{g}, \mathbf{p} |\!) : A \rightleftarrows B$*, the following statements are equivalent.*

- $(\!| !_B, \mathrm{id}_B |\!) \circ (\!| \mathbf{g}, \mathbf{p} |\!) = (\!| !_A, \mathrm{id}_A |\!)$.
- $(\!| \mathbf{g}, \mathbf{p} |\!)$ *satisfies the GetPut law.*



◁

Proof.

(⇑) This is straightforward to prove because lenses that satisfy the GetPut law and that the

codomain is the terminal, are only $(\!|\, !_A, \mathrm{id}_A \,|\!)$.
($\Downarrow$) We prove this by transforming the formula $\mathbf{p} \circ (\mathrm{id}_A \times \mathrm{id}_A) \circ (\mathrm{id}_A \times \mathbf{g} \times \mathrm{id}_1) \circ (\mathrm{copy} \times \mathrm{id}_1) = \mathbf{p} \circ (\mathrm{id}_A \times \mathbf{g}) \circ \mathrm{copy} = \mathrm{id}_A$. □

This proposition indicates that the lenses in the form of folded wires are important for considering the GetPut law. Moreover, the proposition indicates that the GetPut law can be defined as the interaction between arbitrary lens $(\!|\, \mathbf{g}, \mathbf{p} \,|\!)$ and specific lens $(\!|\, !_A, \mathrm{id}_A \,|\!)$. Such an observation is intriguing in terms of the modularization of lenses.

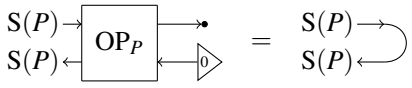
Next, we decompose the condition for a learning algorithm to satisfy the GetPut law into conditions for the three lenses OP_P, IM_A and LL_B.

Theorem 58. *Let (1), (2), (3) and (4) be the following statements:*

- ***(1)*** *for any object $P \in \mathrm{Obj}(\mathbf{C})$, the equation of lenses*

$$\mathrm{Rd}(!_P) \circ \mathrm{OP}_P = (\!|\, !_{\mathrm{S}(P)}, \mathrm{id}_{\mathrm{S}(P)} \,|\!)$$

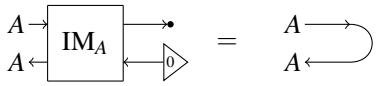
holds;



- ***(2)*** *for any object $A \in \mathrm{Obj}(\mathbf{C})$, the equation of lenses*

$$\mathrm{Rd}(!_A) \circ \mathrm{IM}_A = (\!|\, !_A, \mathrm{id}_A \,|\!)$$

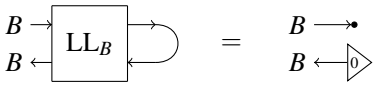
holds;



- ***(3)*** *for any $B \in \mathrm{Obj}(\mathbf{C})$, the equation of lenses*

$$(\!|\, !_B, \mathrm{id}_B \,|\!) \circ \mathrm{LL}_B = \mathrm{Rd}(!_B)$$

holds;



- ***(4)*** *for any morphism $(P, f) : A \to B$, $\overline{\mathrm{Gb}}^{\mathrm{LL}}_{\mathrm{IM,S,OP}}(f)$ satisfies the GetPut law.*

Then, $(1) \wedge (2) \wedge (3) \Longrightarrow (4)$ and $(4) \Longrightarrow (1) \wedge (2)$ hold. ◁

Proof. ((1) ∧ (2) ∧ (3) ⟹ (4)) For any $(P, f) : A \to B$, we simplify the lens $(\!|\, !_B, \mathrm{id}_B \,|\!) \circ \overline{\mathrm{Gb}}^{\mathrm{LL}}_{\mathrm{IM},\mathrm{S},\mathrm{OP}}(f)$ to $(\!|\, !_{\mathrm{S}(P)\times A}, \mathrm{id}_{\mathrm{S}(P)\times A} \,|\!)$ by the given equations.

$$
\begin{aligned}
&(\!|\, !_B, \mathrm{id}_B \,|\!) \circ \mathrm{LL}_B \circ \mathrm{Rd}(f) \circ (\mathrm{OP} \otimes_{\mathrm{L}} \mathrm{IM}_A) \\
&= \mathrm{Rd}(!_B) \circ \mathrm{Rd}(f) \circ (\mathrm{OP} \otimes_{\mathrm{L}} \mathrm{IM}_A) \\
&= \mathrm{Rd}(!_{P\times A}) \circ (\mathrm{OP} \otimes_{\mathrm{L}} \mathrm{IM}_A) \\
&= (\mathrm{Rd}(!_P) \circ \mathrm{OP}) \otimes_{\mathrm{L}} (\mathrm{Rd}(!_A) \circ \mathrm{IM}_A) \\
&= (\!|\, !_{\mathrm{S}(P)}, \mathrm{id}_{\mathrm{S}(P)} \,|\!) \otimes_{\mathrm{L}} (\!|\, !_A, \mathrm{id}_A \,|\!) \\
&= (\!|\, !_{\mathrm{S}(P)\times A}, \mathrm{id}_{\mathrm{S}(P)\times A} \,|\!).
\end{aligned}
$$

□

Proof. ((4) ⟹ (1) ∧ (2)) For any architecture $(P, f) : A \to B$, the GetPut law can be expressed as follows:

$$
(\!|\, !_B, \mathrm{id}_B \,|\!) \circ \overline{\mathrm{Gb}}^{\mathrm{LL}}_{\mathrm{IM},\mathrm{S},\mathrm{OP}}(f) = (\!|\, !_{\mathrm{S}(P)\times A}, \mathrm{id}_{\mathrm{S}(P)\times A} \,|\!). \tag{6}
$$

In particular, using this form of the GetPut law for the following two architectures (morphisms in **C**)

$$
0_B \circ !_A \circ \lambda_A : 1 \times A \to B \quad \text{and} \tag{7}
$$

$$
0_B \circ !_P \circ \rho_P : P \times 1 \to B, \tag{8}
$$

we can derive the following two equations in (1) and (2):

$$
\mathrm{Rd}(!_A) \circ \mathrm{IM}_A = (\!|\, !_A, \mathrm{id}_A \,|\!) \quad \text{and}
$$

$$
\mathrm{Rd}(!_P) \circ \mathrm{OP}_P = (\!|\, !_{\mathrm{S}(P)}, \mathrm{id}_{\mathrm{S}(P)} \,|\!), \quad \text{respectively.}
$$

We only show the former by substituting Equation (7) for f in Equation (6) as follows, because the latter derivation is similar. The left side of the Equation (6) can be simplified as follows:

$$
(\!|\, !_B, \mathrm{id}_B \,|\!) \circ \overline{\mathrm{Gb}}^{\mathrm{LL}}_{\mathrm{IM},\mathrm{S},\mathrm{OP}}(0_B \circ !_A \circ \lambda_A) \tag{9}
$$

$$
= (\!|\, !_B, \mathrm{id}_B \,|\!) \circ \mathrm{Rd}(0_B) \circ \mathrm{Rd}(!_A) \circ \mathrm{Rd}(\lambda_A) \circ (\mathrm{OP}_I \otimes_{\mathrm{L}} \mathrm{IM}_A) \tag{10}
$$

Because the lens from I_{L} to I_{L} is only the identity lens, we can transform the formula as follows.

$$
= \mathrm{Rd}(!_A) \circ \mathrm{Rd}(\lambda_A) \circ (\mathrm{id}_I \otimes_{\mathrm{L}} \mathrm{IM}_A) \tag{11}
$$

$$
= \mathrm{Rd}(!_{1\times A}) \circ (\mathrm{id}_I \otimes_{\mathrm{L}} \mathrm{IM}_A) \tag{12}
$$

$$
= \mathrm{id}_{I_{\mathrm{L}}} \otimes_{\mathrm{L}} (\mathrm{Rd}(!_A) \circ \mathrm{IM}_A). \tag{13}
$$

Similarly, the right side of the Equation (6) can be simplified as follows:

$$
(\!|\, !_{1\times A}, \mathrm{id}_{1\times A} \,|\!) = \mathrm{id}_{I_{\mathrm{L}}} \otimes_{\mathrm{L}} (\!|\, !_A, \mathrm{id}_A \,|\!). \tag{14}
$$

By Equation (6), (13) and (14), we have

$$
\mathrm{id}_{I_{\mathrm{L}}} \otimes_{\mathrm{L}} (\mathrm{Rd}(!_A) \circ \mathrm{IM}_A) = \mathrm{id}_{I_{\mathrm{L}}} \otimes_{\mathrm{L}} (\!|\, !_A, \mathrm{id}_A \,|\!). \tag{15}
$$

Therefore, by appending λ and λ^{-1} to Equation (15) to delete $\mathrm{id}_{I_{\mathrm{L}}}$, we get

$$
\mathrm{Rd}(!_A) \circ \mathrm{IM}_A = (\!|\, !_A, \mathrm{id}_A \,|\!). \tag{16}
$$

□

Remark 59. We cannot derive (3) from (4), because there exists a trivial counterexample. Let OP_P be $\mathrm{Rd}(!_P) \circ (\!|\, !_P, \mathrm{id}_{\mathrm{S}(P)} \,|\!) : \mathrm{S}(P) \rightleftarrows P$ and LL_A be $\mathrm{Rd}(!_A) \circ (\!|\, !_A, \mathrm{id}_A \,|\!) : A \rightleftarrows A$. These lenses do not modify a parameter or an input. Therefore, for any architecture and LL_B, the learning algorithm constructed from them always satisfies the GetPut law, even if LL_B does not satisfy

(3). However, this is not a negative result, since the objective in an application is to construct a learning algorithm that satisfies the GetPut law. Therefore, the counterexample means that it is possible to compensate for the unsuitable property of LL_B, namely that it does not satisfy (3), by the selection of lenses IM_A and OP_P. ◁

Proposition 60. *If we additionally suppose that* $\mathrm{IM}_A = \mathrm{LL}_A^{-1}$*, then* (1) ∧ (2) ∧ (3) *is equivalent to* (4). ◁

Proof. Under the assumption, we can derive (3) from (2). By appending a lens LL_A to both sides of the equation

$$\mathrm{Rd}(!_A) \circ \mathrm{IM}_A = (\!|\, !_A, \mathrm{id}_A \,|\!)$$

in (2), we obtain the equation

$$\mathrm{Rd}(!_A) = (\!|\, !_A, \mathrm{id}_A \,|\!) \circ \mathrm{LL}_A$$

in (3). So, (4) $\Longrightarrow$ (1) ∧ (2) ∧ (3) holds, and then (1) ∧ (2) ∧ (3) is equivalent to (4). □

Therefore, if we use the functorial modularized FST construction, (1) ∧ (2) ∧ (3) $\Longleftrightarrow$ (4) holds, and this statement can be simplified to (1) ∧ (2) $\Longleftrightarrow$ (4).

Example 61. *We can easily prove that FST algorithms whose loss function is the quadratic error satisfy the GetPut law since the FST construction is a modularized FST construction whose* $\mathrm{S} = \mathrm{id}$, $\mathrm{LL} = \mathrm{LL}^{\mathrm{QE}}$*, and* $\mathrm{OP} = \mathrm{OP}^{\mathrm{GD}}$*. The equation* $\mathrm{Rd}(!_P) \circ \mathrm{OP}_P^{\mathrm{GD}} = (\!|\, !_P, \mathrm{id}_P \,|\!)$ *indicates that if a gradient is the zero vector, the lens* $\mathrm{OP}_P^{\mathrm{GD}}$ *does not modify a parameter. While the equation* $(\!|\, !_A, \mathrm{id}_A \,|\!) \circ \mathrm{LL}_A^{\mathrm{QE}} = \mathrm{Rd}(!_A)$ *indicates that if a prediction and a desired output are equal, the lens* $\mathrm{LL}_A^{\mathrm{QE}}$ *outputs the zero vector as a gradient.* ◁

Example 62. *Mean squared error lens* $\mathrm{LL}^{\mathrm{MSE}}$ *(Ex. 44) also satisfies the condition* (3). ◁

Remark 63. The three optimizers other than gradient descent do not satisfy the condition in Thm. 58 for GetPut, because their states propagate information about the gradient from the past training. Therefore, a state and a parameter may be modified even if a prediction agrees with a desired output only in the current training. We believe that this problem is simply that the GetPut law is too simple to be compatible with the three "stateful" optimizers. ◁

6. Softmax cross-entropy

Here, we consider loss lenses for the *softmax cross-entropy* loss function (Cruttwell et al., 2022; Goodfellow et al., 2016), which is used in classification tasks. First, we claim that there is no straightforward functorial construction. Instead of a functorial construction, we present a "jointly-functorial" one to fit constructed learning algorithms into the intuition of learner. For the construction, we utilize the CGGWZ-style modularization of the learner construction. We justify this construction through the properties of the learner construction, such as functoriality, the GetPut law and monoidality. Moreover, there are two candidates for loss lenses called cross-entropy loss lens $\mathrm{LL}^{\mathrm{CE}}$ and softmax cross-entropy lens $\mathrm{LL}^{\mathrm{SCE}}$. We evaluate two loss lenses by comparing their constructed learners, specifically examining whether the learner satisfies the GetPut law and whether the learner preserves monoidal product. Finally, we describe the approaches of Cruttwell et al. (2022) and Fong et al. (2019) to handling (softmax) cross-entropy, and compare them with our approach.

loss lens LL	(a) get of LL $\mathbf{g} : B \to B =$	(b) put of LL $\mathbf{p} : B \times B \to B$ $\mathbf{p}(\widehat{b}, b)_i =$	(c) invertibility of LL	(d) monoidality of LL	(e) Does LL satisfy the condition on LL for $\overline{\mathrm{Gb}}(f)$'s GetPut?	(f) By LL, is $\overline{\mathrm{Gb}}$ defined with GetPut?
$\mathrm{LL}^{(\mathrm{S})\mathrm{CE}}$	id_B	$-b_i + \mathrm{Sof}(\widehat{b})_i \cdot \sum_k b_k$	×	×	×	undefined
$\mathrm{LL}^{\mathrm{SCE}}$	Sof	$-b_i + \mathrm{Sof}(\widehat{b})_i \cdot \sum_k b_k$	×	×	◯	undefined
$\mathrm{LL}^{\mathrm{CE}}$	id_B	$-b_i / \widehat{b}_i$	◯	◯	×	△†

- **(c)** indicates whether LL has the inverse lens (i.e., whether we can obtain a *functorial* algorithm $\overline{\mathrm{Gb}}$ in the way of Def. 40 and Def. 55).
- **(d)** indicates whether LL satisfies $\mathrm{LL}_A \otimes_{\mathrm{L}} \mathrm{LL}_B = \mathrm{LL}_{A \otimes_{\mathrm{L}} B}$.
- **(e)** indicates whether LL satisfies $(\!|\, !_A, \mathrm{id}_A \,|\!) \circ \mathrm{LL}_A = \mathrm{Rd}(!_A)$, the condition on LL for $\overline{\mathrm{Gb}}(f)$ to satisfy GetPut law given in Thm. 58. This column does not require that (c) holds, so $\overline{\mathrm{Gb}}$ might not be well-defined.
- **(f)** indicates whether both (c) and (d) hold (i.e., whether LL can be used to define $\overline{\mathrm{Gb}}$ and every $\overline{\mathrm{Gb}}(f)$ satisfies the GetPut law).
- † Given suitable OP as in Thm. 58, if the softmax function is post-composed at the end of the composite of architectures, then the learner $\overline{\mathrm{Gb}}(\mathrm{Sof} \circ_{\mathbf{P}} f_n \circ_{\mathbf{P}} \ldots \circ_{\mathbf{P}} f_1)$ satisfies pGetPut, rather than GetPut.

Figure 4. The loss lenses for classification tasks.

For classification tasks, an input and an output in a training dataset are, respectively, an arbitrary vector and a finite discrete probability distribution. The finite discrete probability distribution is given as a vector x such that $0 \leq x_i \leq 1$ for any i and the sum of all the components of x is 1.

For learning classification tasks, we use softmax cross-entropy loss function, rather than quadratic error, which is used for regression tasks or approximation tasks.

Definition 64 (Softmax cross-entropy)**.** *For any B in* **Smooth**, Softmax cross-entropy $e^{\mathrm{SCE}} : B \times B \to \mathbb{R}$ *is defined as follows:*

$$e^{\mathrm{SCE}}(\widehat{b}, b) = e^{\mathrm{CE}}(\mathrm{Sof}(\widehat{b}), b),$$

where $e^{\mathrm{CE}} : B \times B \to \mathbb{R}$ is defined by

$$e^{\mathrm{CE}}(b', b) = -\sum_i b_i \log b'_i,$$

which is called cross-entropy*, and $\mathrm{Sof}_B : B \to B$ is defined by*

$$\mathrm{Sof}(x)_i = e^{x_i} / \sum_j e^{x_j},$$

which is called softmax function. ◁

Cross-entropy is a distance between two probability distributions. The softmax function can be regarded as a normalizing function that takes a vector x and returns the probability distribution $\mathrm{Sof}(x)$. Thus, the softmax cross-entropy compares a vector output by an architecture with a probability distribution given as a desired output. Note that, while cross-entropy is a partial function, softmax cross-entropy is total because every component in $\mathrm{Sof}(x)$ is greater than 0.

Basic idea of a construction for classification tasks

First, we think that we cannot present a straightforward functorial construction for classification

tasks. This is because we do not consider the composition of models for classification. The composition means providing a probability distribution, which is output by the first model, to the second model as an input. However, the second model requires a vector as input, not a probability distribution. Therefore, in practice, it is desirable for the first model to be a model for regression tasks, because the model outputs a vector, which is desirable for the second model. Namely, when composing architectures, all architectures are for regression tasks, or all but the last one are for regression tasks and the last one is for a classification task. Hence, if we follow such an existing style, what we need for a classification task is not the functoriality on its own but some variant, i.e., the following property for some $\overline{\mathrm{Gb}}^*$ for classification tasks where $\overline{\mathrm{Gb}}$ is used for regression tasks:

$$\overline{\mathrm{Gb}}^*(g) \circ \overline{\mathrm{Gb}}(f) = \overline{\mathrm{Gb}}^*(g \circ f). \tag{17}$$

6.1 Learner construction for classification tasks

To construct learners, we define two loss lenses that calculate a loss gradient by cross-entropy and softmax cross-entropy as follows:

Definition 65 (Cross entropy loss lens)**.** *For any B in* **Smooth***, the* cross-entropy loss lens *is defined as follows:*

$$\mathrm{LL}_B^{\mathrm{CE}} = (\!|\, \mathrm{id}_B, \mathbf{p} \,|\!) : B \rightleftarrows B$$

where $\mathbf{p}(\widehat{b}, b)_i = (\nabla_{\widehat{b}} e^{\mathrm{CE}}(\widehat{b}, b))_i = -b_i/\widehat{b}_i$. ◁

Definition 66 (Softmax cross-entropy loss lens)**.** *For any B in* **Smooth***, the* softmax cross-entropy loss lens *is defined as follows:*

$$\mathrm{LL}_B^{\mathrm{SCE}} = (\!|\, \mathrm{Sof}, \mathbf{p} \,|\!) : B \rightleftarrows B$$

where $\mathbf{p}(\widehat{b}, b)_i = (\nabla_{\widehat{b}} e^{\mathrm{SCE}}(\widehat{b}, b))_i = -b_i + \mathrm{Sof}(\widehat{b})_i \cdot \sum_k b_k$. ◁

Although the cross-entropy loss lens and the softmax cross-entropy loss lens seem to be entirely different, the following equation holds:

$$\mathrm{LL}_B^{\mathrm{CE}} \circ \mathrm{Rd}(\mathrm{Sof}) = \mathrm{LL}_B^{\mathrm{SCE}}.$$

Thus, the softmax cross-entropy loss lens not only compares a prediction with a desired output, but also appends the softmax function to the architecture. This is why the get function in the softmax cross-entropy loss lens is not the identity function.

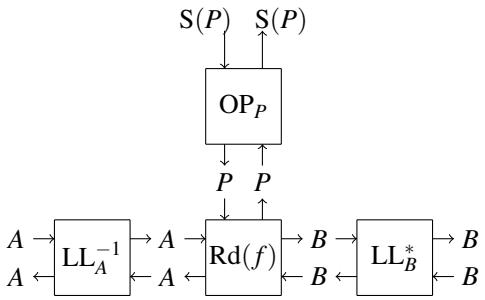
Remark 67. As noted by Fong et al. (2019), the get function and the put function of cross-entropy loss lens are partial functions. However, the get function and the put function of softmax cross-entropy loss lens are total functions if desired output is a probability distribution. ◁

The categorical behaviors of learners utilizing these loss lenses are different. Generally speaking, a learner using the cross-entropy loss lens is compatible with the monoidal product, and a learner using the softmax cross-entropy loss lens is compatible with the GetPut law; more details are discussed later.

To consistently define learners for classification tasks, we define a modularized FST construction for classification tasks. In the next definition, lenses IM are intended as input modifiers for modifying (input) vectors; so a typical example is the inverse of the quadratic error lens in Ex. 43.

Definition 68. *For any family of invertible lenses* $\mathrm{LL}_A : A \rightleftarrows A$ *and* $* \in \{\mathrm{CE}, \mathrm{SCE}\}$*, we define:*

$$\overline{\mathrm{Gb}}^{*}_{\mathrm{LL},\mathrm{S},\mathrm{OP}} = \overline{\mathrm{Gb}}^{\mathrm{LL}^{*}}_{\mathrm{LL}^{-1},\mathrm{S},\mathrm{OP}}$$



◁

The family $\overline{\mathrm{Gb}}^{*}_{\mathrm{LL},\mathrm{S},\mathrm{OP}}$ is a family of maps on morphisms, and hence—equipped with the identity on objects—has the same structure as a functor, but does not preserve composition nor identity in general, so this is actually not a functor.

The reason we use this unusual definition stems from the asymmetry in classification tasks, which is the difference between an input being a vector and an output being a probability distribution. Recall that the role of the module $(\mathrm{LL}^{\mathrm{QE}})^{-1}$, QE is a loss function comparing two vectors, and hence the lens $(\mathrm{LL}^{\mathrm{QE}})^{-1}$ modifies an input vector to be suitable for comparison by the quadratic error. Insofar as an input in classification tasks is a vector, the use of $(\mathrm{LL}^{\mathrm{QE}})^{-1}$ is reasonable. Moreover, we can justify the construction by not only the role of the module, but also the behavior of the learner provided by the construction. Specifically, the behavior is explained through output backpropagation. As promised, we have the following "jointly-functorial" property:

Theorem 69.

$$\overline{\mathrm{Gb}}^{*}_{\mathrm{LL},\mathrm{S},\mathrm{OP}}(g) \circ_{\mathrm{P}} \overline{\mathrm{Gb}}_{\mathrm{LL},\mathrm{S},\mathrm{OP}}(f) = \overline{\mathrm{Gb}}^{*}_{\mathrm{LL},\mathrm{S},\mathrm{OP}}(g \circ_{\mathrm{P}} f) \quad (* \in \{\mathrm{CE}, \mathrm{SCE}\})$$

◁

Proof. This follows from the following analogous property:

$$\mathrm{Loss}^{\mathrm{LL}^{*}}_{\mathrm{LL}^{-1}}(g) \circ_{\mathrm{P}} \mathrm{Loss}^{\mathrm{LL}}_{\mathrm{LL}^{-1}}(f) = \mathrm{Loss}^{\mathrm{LL}^{*}}_{\mathrm{LL}^{-1}}(g \circ_{\mathrm{P}} f),$$

which holds clearly by the construction. □

Moreover, although specific conditions for an architecture and a loss lens are required, we can expect that learning algorithms constructed by $\overline{\mathrm{Gb}}^{*}_{\mathrm{LL},\mathrm{S},\mathrm{OP}}$ satisfy the GetPut law, as explained later.

6.2 Loss lenses for classification

We discuss the categorical properties of the two loss lenses defined above. The properties of these lenses (with $\mathrm{LL}^{(\mathrm{S})\mathrm{CE}}$, which is $\mathrm{LL}^{\mathrm{SCE}}$ such that the get function is the identity) are summarized in

fig. 4. Due to differences between these lenses, the behaviors of the constructions $\overline{\mathrm{Gb}}^{*}_{\mathrm{LL,S,OP}}$ also change.

The mathematical difference between the two lenses is whether they include the softmax function. Specifically, the equation $\mathrm{LL}^{\mathrm{CE}}_{B} \circ \mathrm{Rd}(\mathrm{Sof}) = \mathrm{LL}^{\mathrm{SCE}}_{B}$ holds. Intuitively, softmax cross-entropy loss lens performs normalization of an output of an architecture within the loss lens. In contrast, cross-entropy loss lens assumes that normalization is already performed in an architecture. For these reasons, the normalization method called softmax function is important to analyze, but it does not possess categorically good properties. In practice, the softmax function $\mathrm{Sof}_B : B \to B$ (indexed by object B) does not always respect the monoidal product, as follows:

$$\mathrm{Sof}_A \times \mathrm{Sof}_B \neq \mathrm{Sof}_{A \times B} : A \times B \to A \times B, \text{ in general.}$$

Moreover, although it does not relate to our construction $\overline{\mathrm{Gb}}^{*}_{\mathrm{LL,S,OP}}$, softmax function $\mathrm{Sof}_B : B \to B$ does not have its inverse. Therefore, $\mathrm{Rd}(\mathrm{Sof})$ does not have an inverse, and thus $\mathrm{LL}^{\mathrm{SCE}}_{B}$ also lacks its inverse.

Preservation of monoidal product Cross-entropy loss lens respects monoidal product, as follows:

$$\mathrm{LL}^{\mathrm{CE}}_{A} \otimes_{\mathrm{L}} \mathrm{LL}^{\mathrm{CE}}_{B} = \mathrm{LL}^{\mathrm{CE}}_{A \otimes_{\mathrm{L}} B}.$$

In contrast, softmax cross-entropy lens does not, because softmax function $\mathrm{Sof} : B \to B$ does not respect monoidal product. These properties of the loss lenses also affect the constructions. For our construction with cross-entropy loss lens $\overline{\mathrm{Gb}}^{\mathrm{CE}}_{\mathrm{LL,S,OP}}$, the following equation holds:

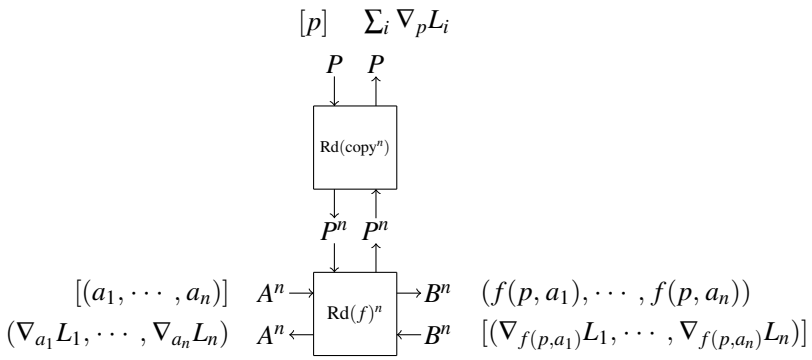
$$\overline{\mathrm{Gb}}^{\mathrm{CE}}_{\mathrm{LL,S,OP}}(f) \otimes_{\mathrm{L}} \overline{\mathrm{Gb}}^{\mathrm{CE}}_{\mathrm{LL,S,OP}}(g) = \overline{\mathrm{Gb}}^{\mathrm{CE}}_{\mathrm{LL,S,OP}}(f \otimes_{\mathrm{L}} g),$$

but the case of softmax cross-entropy lens $\overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}$ does not preserve monoidal product, as follows:

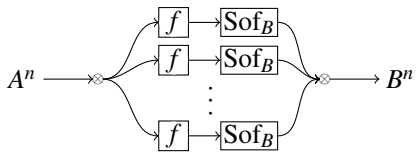
$$\overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}(f) \otimes_{\mathrm{L}} \overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}(g) \neq \overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}(f \otimes_{\mathrm{L}} g), \text{ in general.}$$

In particular, the property that $\overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}$ does not preserve monoidal product is problematic in the situation of stochastic gradient descent in Ex. 25. Here, we apply $\overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}$ to the following architecture:

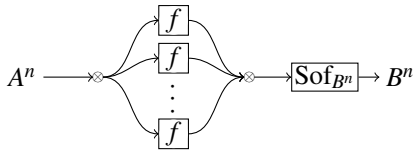
$$\mathbf{Para}(\mathrm{Rd})(P, (\mathrm{copy}^n \times \mathrm{id}_{A^n}) \circ f^n) = (P, (\mathrm{Rd}(\mathrm{copy}^n) \times \mathrm{id}_{A^n}) \circ \mathrm{Rd}(f)^n)$$



This construction appends the loss lens $\mathrm{LL}^{\mathrm{SCE}}_{B^n} : B^n \rightleftarrows B^n$ to the architecture. However, in practice, we want to append the loss lens $\mathrm{LL}^{\mathrm{SCE}^n}_B$ where $\mathrm{LL}^{\mathrm{SCE}^n}_B = \overbrace{\mathrm{LL}^{\mathrm{SCE}}_B \otimes_{\mathrm{L}} \cdots \otimes_{\mathrm{L}} \mathrm{LL}^{\mathrm{SCE}}_B}^{n\text{-times}} : B^n \rightleftarrows B^n$. Moreover, the two lenses $\mathrm{LL}^{\mathrm{SCE}}_{B^n}$ and $\mathrm{LL}^{\mathrm{SCE}^n}_B$ are generally not equal, because $\mathrm{LL}^{\mathrm{SCE}}_{B^n} = \mathrm{Rd}(\mathrm{Sof}_{B^n}) \circ \mathrm{LL}^{\mathrm{CE}}_{B^n}$ and $\mathrm{LL}^{\mathrm{SCE}^n}_B = \mathrm{Rd}(\mathrm{Sof}^n_B) \circ \mathrm{LL}^{\mathrm{CE}}_{B^n}$. To explain the difference between the two lenses using string diagrams on **Para**(**Smooth**), we consider only the get functions of these lenses. The get function of learner using $\mathrm{LL}^{\mathrm{SCE}^n}_B$ can be depicted as follows.



For the original architecture $(P, f) : A \to B$, we want to train the architecture (or model) f with normalization using softmax function $\mathrm{Sof} : B \to B$. Therefore, we should individually normalize the outputs of architectures, and obtain n probability distributions. However, the get function of a learner using $\mathrm{LL}^{\mathrm{SCE}}_{B^n}$, which is appended by our new construction $\overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}$, can be depicted as follows.



In the diagram, the softmax function $\mathrm{Sof}_{B^n} : B^n \to B^n$ normalizes the concatenated outputs of the architectures. Thus, an output of the diagram is a single probability distribution, which does not make sense.

Remark 70. It might be thought that we should take the n-times monoidal product of learner $\overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}(f)$ instead of architecture f, because the normalization of the learner is performed by Sof^n_B rather than Sof_{B^n}. However, in this construction, we cannot append $\mathrm{Rd}(\mathrm{copy}^n)$, because the presence of the optimizer prevents it. Thus, applying the stochastic gradient descent method in this situation is unrealistic, even if we apply it either after or before constructing learners. ◁

Therefore, for the construction $\overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}$, the architecture cannot emulate stochastic gradient descent.

The GetPut law We can use Thm. 58 to check that learning algorithms produced by some constructions satisfy the GetPut law, because the theorem does not assume the invertibility of loss lenses. Therefore, the following equation in (3) is important.

$$(\!| !_B, \mathrm{id}_B |\!) \circ \mathrm{LL}^*_B = \mathrm{Rd}(!_B). \tag{18}$$

Cross-entropy loss lens does not satisfy Equation 18 in general. Therefore, we cannot derive that the learner using cross-entropy loss lens satisfies the GetPut law (or pGetPut law). In contrast, softmax cross-entropy loss lens satisfies this equation. Therefore, with an appropriate loss lens

such as quadratic error lens $\mathrm{LL}^{\mathrm{QE}}$, a learner using softmax cross-entropy loss lens satisfies the GetPut law.

However, it is not the case that a learner using cross-entropy loss lens cannot learn at all; it works the same as softmax cross-entropy loss lens when we assume an output of an architecture is normalized by softmax function.

Remark 71. The use of $\mathrm{LL}^{\mathrm{QE}}$ as a loss lens for classification tasks is necessary for a learner to satisfy the GetPut law, compared with functorial construction. First, the loss lens $\mathrm{LL}_B^{\mathrm{SCE}}$ does not have an inverse. So, it cannot define functorial learner to check the GetPut law. While the loss lens $\mathrm{LL}_B^{\mathrm{CE}}$ has an inverse, the "inverse" does not satisfy the condition (2) in Thm. 58. It means that even if an output of an architecture is normalized, the learning algorithms do not satisfy the GetPut law. Therefore, GetPut law also justifies our construction $\overline{\mathrm{Gb}}^{*}_{\mathrm{LL,S,OP}}$. ◁

6.3 Comparison with existing results

Without going into detail, both Fong et al. (2019) and Cruttwell et al. (2022) also use softmax cross-entropy (cross-entropy). For the CGGWZ construction, the above issues are not problematic because they do not consider the functoriality of correspondences from architectures to learning algorithms. Therefore, we can directly use e^{CE} or e^{SCE} for the CGGWZ construction, and they do not need to use $(\mathrm{LL}^{\mathrm{QE}})^{-1}$. For the FST construction, they use cross-entropy as an example of a loss function. However, their mathematical definition differs from our cross-entropy. They define the cross-entropy loss $e^{\mathrm{CE}'} : B \times B \to \mathbb{R}$ as follows:

$$d(x, y) = y \log x + (1 - y) \log(1 - x),\ e^{\mathrm{CE}'}(\widehat{b}, b) = \sum_i d(\widehat{b}_i, b_i).$$

The loss function $e^{\mathrm{CE}} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is used for the n-class classification with a batch size of 1, while the loss function $e^{\mathrm{CE}'} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is used for the 2-class classification with a batch size of n. Fortunately, the loss lenses $\mathrm{LL}^{\mathrm{CE}'}$ constructed from the above loss function $e^{\mathrm{CE}'}$ give rise to the FST construction, which is functorial and monoidal. However, learners by the construction do not always satisfy the GetPut law, as in the case of Rmk. 71. In this situation, the invalidity is problematic for the interpretation of learners. In particular, for the modification of an input by a learner, a lens $(\mathrm{LL}^{\mathrm{CE}'})^{-1}$ is required to modify an input that forms a probability distribution to a probability distribution. This is because the lens $\mathrm{LL}^{\mathrm{CE}'}$ is required to compare two probability distributions. However, the lens $(\mathrm{LL}^{\mathrm{CE}'})^{-1}$ does not output a probability distribution even if an input is also a probability distribution.

Remark 72. We do not aim to define a functorial construction for classification tasks by adjusting the actual computations of a learner. However, the obstacle to our new construction $\overline{\mathrm{Gb}}^{\mathrm{SCE}}_{\mathrm{LL,S,OP}}$ being functorial is that we can compose two models for classification tasks. Here, although it is not a practical contribution, we can prevent such a composition by distinguishing between the set of vectors and the set of probability distributions. Fabregat-Hernández et al. (2023) distinguish the two sets by decorating an object $\mathbb{R}^n$ in **Smooth** with a metric $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. Using the formalization, we can type the softmax function as follows:

$$\mathrm{Sof} : (\mathbb{R}^n, e^{\mathrm{QE}}) \to (\mathbb{R}^n, e^{\mathrm{CE}}).$$

It indicates that the inputs of softmax function are vectors, which are compared using quadratic error, and the outputs of softmax function are probability distributions, which are compared using cross-entropy. So, we can also type architectures for classification as follows:

$$\mathrm{Sof} \circ_{\mathbf{r}} f : (\mathbb{R}^n, e^{\mathrm{QE}}) \to (\mathbb{R}^m, e^{\mathrm{CE}}),$$

$$\mathrm{Sof} \circ_{\mathfrak{p}} g : (\mathbb{R}^m, e^{\mathrm{QE}}) \to (\mathbb{R}^l, e^{\mathrm{CE}}).$$

Obviously, we cannot compose the two architectures for classification tasks.

In constructing a learning algorithm, we can also use information about the metrics. Unlike the learner construction proposed by Fabregat-Hernández et al. (2023), we can directly construct a learner from the architecture $\mathrm{Sof} \circ_{\mathfrak{p}} f : (\mathbb{R}^n, e^{\mathrm{QE}}) \to (\mathbb{R}^m, e^{\mathrm{CE}})$, where $\mathrm{LL} = \mathrm{LL}^{\mathrm{CE}}$ and $\mathrm{IM} = (\mathrm{LL}^{\mathrm{QE}})^{-1}$, which may be functorial. ◁

7. Related work

In addition to this study, there are various results regarding applications of category theory to machine learning. We can review these results through survey papers by Shiebler et al. (2021), Jia et al. (2025), and Crescenzi (2024).

7.1 Category theory and machine learning

First, we review existing categorical formalizations of machine learning which are closely related to the present work. Specifically, we focus on learning algorithms, learners, and architectures.

Learning algorithm We review categorical formalizations of gradient-based learning other than the frameworks of Fong et al. (2019), Cruttwell et al. (2022), and the present work. Shiebler (2021*b*) extended architectures, the notion of a learner, and their construction to stochastic variants. As mentioned in Rmk. 72, Fabregat-Hernández et al. (2023) also extended these concepts to handle metric spaces instead of vector spaces. Jacobs and Sprunger (2019) introduced formalizations of learning algorithms distinct from those of Fong et al. (2019) and Cruttwell et al. (2022); they used the notion of a functor as the operation of updating parameters.

Learner Building on the notion of the para lens (learner) introduced by Fong et al. (2019), Diskin (2020), Fong and Johnson (2019), and Riley (2018) focused on the abstract structure in para lenses and developed it in the direction of bidirectional transformations. They also discussed lens laws other than the GetPut law.

Architecture Architectures, which are the target of the application of our construction, have also been studied in category theory. Gavranović and Villani (2022) formalized a kind of architecture called *Graph Convolutional Neural Networks*. Sprunger and Katsumata (2021) also introduced the *delayed trace*, which can formalize *Recurrent Neural Networks* and differentiate through them while keeping their structure. Regarding the general construction of architectures, Gavranović et al. (2024) formalized recursive constructions of architectures and the notion of *parameter sharing*, which is also used in Ex. 25. Moreover, Xu and Maruyama (2022) and Khatri et al. (2024) have applied string diagrams to the graphical construction of architectures.

7.2 Comparison with existing frameworks

Among these topics, our contribution is based on the application of category theory to learning algorithms, building upon two existing frameworks (Fong et al., 2019; Cruttwell et al., 2022). Since we have already compared our contribution with these two works in detail, we summarize the comparison concisely here.

- **Base category** In Fong et al. (2019), the base category is restricted to **Smooth**, while in Cruttwell et al. (2022) and in our work, the base category is generalized to cartesian reverse differential categories. Examples of cartesian reverse differential categories in Cruttwell

et al. (2022) include not only **Smooth** but also the category $\mathrm{POLY}_{\mathbb{Z}_2}$, which is used for the formalization of boolean circuits.

- **Modules** As mentioned in section 2.1, gradient-based learning algorithms consist of an architecture, a loss function, and an optimizer.
  - **Architecture** In Fong et al. (2019) and Cruttwell et al. (2022), as well as in our work, an architecture is an arbitrary morphism in a base category.
  - **Loss function** In Fong et al. (2019), examples of loss functions are the quadratic error e^{QE} and the cross-entropy loss $e^{\mathrm{CE}'}$ (discussed in (Fong et al., 2019, VII. Discussion)). In Cruttwell et al. (2022), examples of loss functions are the quadratic error e^{QE} and the softmax cross-entropy loss e^{SCE}. Moreover, the dot product is discussed there as a loss function for deep dreaming. In our work, we formalized loss functions as loss lenses, which include the quadratic error e^{QE} in Ex. 43, the mean squared error e^{MSE} in Ex. 44, the cross-entropy loss e^{CE} in Def. 65, and the softmax cross-entropy loss e^{SCE} in Def. 66. Because of practicality and redundancy, we did not explicitly formalize the dot product and $e^{\mathrm{CE}'}$ as loss lenses. Note that these loss functions can also be utilized in the CGGWZ construction, as it does not require functoriality.
  - **Optimizer** In Fong et al. (2019), only the gradient descent optimizer is considered. In Cruttwell et al. (2022), various optimizers such as gradient descent, momentum, Nesterov momentum, Adagrad, and Adam are formalized as lenses. In our work, we reformulated these optimizers as optimizer lens families in Thm. 52, which include the gradient descent optimizer in Ex. 49; however, as discussed in Rmk. 54, we did not consider the Adam optimizer.
- **GetPut law** In Fong and Johnson (2019), the GetPut law was discussed only for the FST algorithm with the quadratic error e^{QE} (with the gradient descent optimizer). In Cruttwell et al. (2022), the GetPut law was not discussed for CGGWZ algorithms because these algorithms were misaligned with the notion of a lens in terms of bidirectional transformation. In our work, we discussed the GetPut law for learners constructed from the various modules reviewed above.

8. Conclusions

In section 5, we defined the modularized FST construction. As a consequence, we obtained the two construction methods of modularized FST algorithms. The first method is by appending lenses $\mathrm{LL}, \mathrm{LL}^{-1}, \mathrm{OP}$ such as in the CGGWZ construction. The second method is by output backpropagation such as in the FST construction. The typical example is the learner for classification tasks in section 6. Through the CGGWZ-style modularization, we can simultaneously use lenses derived from different loss functions, utilizing $(\mathrm{LL}^{\mathrm{QE}})^{-1}$ as an input modifier and $\mathrm{LL}^{\mathrm{CE}}$ as a loss lens in the construction of the learners. Through FST-style modularization, we can construct a learner for classification via the composition $\overline{\mathrm{Gb}}^*(g) \circ \overline{\mathrm{Gb}}(f) = \overline{\mathrm{Gb}}^*(g \circ f)$.

While we formulated these computations into modules, we also observed their behaviors (properties). These behaviors can be used as the sufficient conditions for the desirable properties of the modularized FST construction, such as functoriality, the GetPut law, and well-definedness. As a result, by checking these properties, we can easily justify the anomalous learner construction, such as the construction for classification tasks.

8.1 Future work

There remain some limitations regarding the full incorporation of specific instances from previous works, such as the Adam optimizer. Beyond addressing these, we can consider the following future works:

- **Quantification** The properties considered in this paper are of limited practical utility for actual learning algorithms since these properties are qualitative rather than quantitative. By extending these properties to quantitative counterparts, we can analyze the performance of learning algorithms more precisely.
- **Other learning mechanisms** In this paper, we focused on gradient-based learning algorithms. However, the framework of learners is not limited to gradient-based learning. For example, Smithe (2020) gives a formulation using an optic (generalized lens) for Bayesian learning algorithms. Hedges and Rodríguez Sakamoto (2023) and Hedges and Rodríguez Sakamoto (2025) also give optic formulations of reinforcement learning algorithms. We believe that these learning algorithms can also be formulated within the learner framework.
- **Specific training methods and tasks** There are training methods (or techniques) which do not depend on specific learning algorithms (gradient-based or not). These specific training methods often give rise to specific tasks, such as input reconstruction in autoencoders. In the training of autoencoders, we use input data as the desired output data. Therefore, to formalize such training methods and tasks, the framework of learners is particularly suitable because it is generalized from specific learning algorithms, and it explicitly formalizes provision of desired output data.

Acknowledgments

In conducting the research and writing this paper, we would like to thank Kentaro Kikuchi for his support. We also extend our thanks to Eijiro Sumii and Hiroshi Unno for their valuable comments.

References

Boisseau, G., Nester, C., and Román, M. 2023. Cornering optics. *Electronic Proceedings in Theoretical Computer Science*, 380:97–110.

Capucci, M., Gavranović, B., Hedges, J., and Rischel, E. F. 2022. Towards foundations of categorical cybernetics. *Electronic Proceedings in Theoretical Computer Science*, 372:235–248.

Cockett, J. R. B. and Seely, R. A. G. 2017. Proof Theory of the Cut Rule. In *Categories for the Working Philosopher*, pp. 223–261. Oxford University Press.

Cockett, R., Cruttwell, G., Gallagher, J., Lemay, J.-S. P., MacAdam, B., Plotkin, G., and Pronk, D. 2020. Reverse Derivative Categories. In **Fernández, M. and Muscholl, A.**, editors, *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, volume 152 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 18:1–18:16, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Crescenzi, F. R. 2024. Towards a categorical foundation of deep learning: A survey.

Cruttwell, G. S. H., Gavranović, B., Ghani, N., Wilson, P., and Zanasi, F. 2022. Categorical foundations of gradient-based learning. In **Sergey, I.**, editor, *Programming Languages and Systems*, pp. 1–28, Cham. Springer International Publishing.

Diskin, Z. 2020. General supervised learning as change propagation with delta lenses. In **Goubault-Larrecq, J. and König, B.**, editors, *Foundations of Software Science and Computation Structures*, pp. 177–197, Cham. Springer International Publishing.

Fabregat-Hernández, A., Palanca, J., and Botti, V. 2023. Exploring explainable ai: category theory insights into machine learning algorithms. *Machine Learning: Science and Technology*, 4(4):045061.

Fong, B. and Johnson, M. 2019. Lenses and learners. In **Cheney, J. and Ko, H.-S.**, editors, *Proceedings of the Eighth International Workshop on Bidirectional Transformations*.

Fong, B., Spivak, D., and Tuyeras, R. 2019. Backprop as functor: A compositional perspective on supervised learning. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–13, Los Alamitos, CA, USA. IEEE Computer Society.

Foster, J. N., Greenwald, M. B., Moore, J. T., Pierce, B. C., and Schmitt, A. 2007. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3):17–es.

Gavranović, B. 2019. Compositional deep learning.

Gavranović, B. 2024. Fundamental components of deep learning: A category-theoretic approach.

Gavranović, B., Lessard, P., Dudzik, A., von Glehn, T., Araújo, J. G. M., and Veličković, P. 2024. Categorical deep learning: An algebraic theory of architectures.

Gavranović, B. and Villani, M. 2022. Graph convolutional neural networks as parametric cokleisli morphisms.

Goodfellow, I., **Bengio, Y.**, **and Courville, A.** 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hedges, J. and Rodríguez Sakamoto, R. 2023. Value iteration is optic composition. *Electronic Proceedings in Theoretical Computer Science*, 380:417–432.

Hedges, J. and Rodríguez Sakamoto, R. 2025. Reinforcement learning in categorical cybernetics. *Electronic Proceedings in Theoretical Computer Science*, 429:270–286.

Jacobs, B. and Sprunger, D. 2019. Neural nets via forward state transformation and backward loss transformation. *Electronic Notes in Theoretical Computer Science*, 347:161–177. Proceedings of the Thirty-Fifth Conference on the Mathematical Foundations of Programming Semantics.

Jia, Y., **Peng, G.**, **Yang, Z.**, **and Chen, T.** 2025. Category-theoretical and topos-theoretical frameworks in machine learning: A survey.

Khatri, N., **Laakkonen, T.**, **Liu, J.**, **and Wang-Mańscianica, V.** 2024. On the anatomy of attention.

Mac Lane, S. 1998. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer, 2 edition.

Nester, C. 2021. The structure of concurrent process histories. In **Damiani, F. and Dardha, O.**, editors, *Coordination Models and Languages*, pp. 209–224, Cham. Springer International Publishing.

Nester, C. 2023. Concurrent Process Histories and Resource Transducers. *Logical Methods in Computer Science*, Volume 19, Issue 1.

PyTorch Foundation 2024. Torch.optim — PyTorch 2.3 documentation.

Riley, M. 2018. Categories of optics.

Selinger, P. 2011. *A Survey of Graphical Languages for Monoidal Categories*, pp. 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg.

Shiebler, D. 2021a. Categorical Stochastic Processes and Likelihood. *Compositionality*, 3.

Shiebler, D. 2021b. Categorical stochastic processes and likelihood. *Compositionality*, Volume 3 (2021).

Shiebler, D., **Gavranović, B.**, **and Wilson, P.** 2021. Category theory in machine learning.

Smithe, T. S. C. 2020. Bayesian updates compose optically.

Sprunger, D. and Katsumata, S.-y. 2021. Differentiable causal computations via delayed trace. In *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '19. IEEE Press.

Xu, T. and Maruyama, Y. 2022. Neural string diagrams: A universal modelling language for categorical deep learning. In **Goertzel, B.**, **Iklé, M.**, **and Potapov, A.**, editors, *Artificial General Intelligence*, pp. 306–315, Cham. Springer International Publishing.