A categorical framework for gradient-based learning

Geoff Cruttwell
Mount Allison University

Joint work with Bruno Gavranović, Neil Ghani,
Paul Wilson, and Fabio Zanasi

March 9th, 2021

Overall plan

Today:

- Describe some of the basic ideas behind gradient-based machine learning
- Start to think about how we can represent these ideas categorically through three structures:
  - A construction of a category of parameterized maps
  - A construction of a category of "bidirectional" maps, also known as lenses
  - Cartesian reverse differential categories (and how they relate to the more well-known structure of Cartesian differential categories)

Next time:

- Putting these these three ideas together and looking at how specific examples of gradient-based machine learning algorithms fit into this framework

Basic gradient-based learning

In general, we want a machine to "learn" the values of some objective function

$$O : X \to Y$$

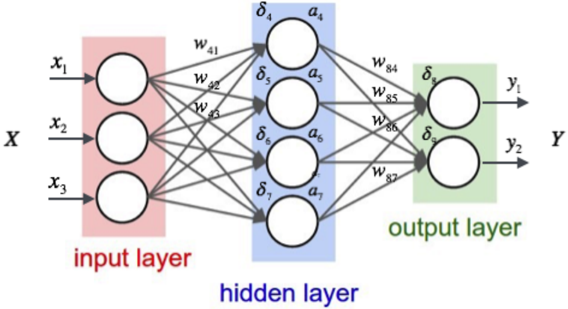
- For example, X might consist of the set of all images of a certain size (represented as some $\mathbb{R}^n$ for large n), Y might be the set $[0, 1]$, and we want $O(x)$ to be "how likely the image a contains a cat".
- Of course, searching over the entire space of all functions from X to Y is hard!
- So, instead one often fixes a particular "parameter space" $P = \mathbb{R}^k$ (usually, again, k is large), fix a particular function

  $$f : P \times X \to Y$$

  and try to find a good value of $p \in P$ so that for all x, $f(p, x)$ is sufficiently close to $O(x)$.

Neural networks

A standard choice of $f : P \times X \longrightarrow Y$ is given by a neural network, eg.[1],



- The values w_{ij}, δ_k, a_l, etc., are determined by the parameter $p \in P$.
- Sometimes they are combined in linear ways, sometimes in non-linear ways.
- While the choice of $f : P \times A \longrightarrow B$ is often of this form, it's been increasingly realized that looking more generally any function of type $P \times X \longrightarrow Y$ is useful.

[1] Image from https://cs231n.github.io/neural-networks-1/

Learning

How does one find the right choice of parameter $p \in P$?

- We start with some set of "training data": pairs

$$(x_0, y_0), (x_1, y_1), \ldots (x_n, y_n)$$

  where $y_i = O(x_i)$ is a known value of the function. (These are often found by some poor student "volunteer").
- We want to minimize how far $f(p, x_i)$ is from y_i.
- That is, some error function $e : Y \times Y \longrightarrow \mathbb{R}$ is chosen, and then we define a new function

$$J : P \longrightarrow \mathbb{R}$$

$$J(p) = e(f(p, a_0), b_0)$$

  which we want to minimize.

Learning continued

To do this, we follow the idea of gradient descent:

- Calculate a new parameter

$$p' = p - \epsilon \nabla J(p)$$

  (where ∇ is the gradient and ϵ is some constant).
- Repeat this process until we get $f(p, a)$ sufficiently close to $O(a)$.

This is the basic version, but there are many variants of it:

- Different error functions are used
- The update $p - \epsilon \nabla J(p)$ is often more complicated, sometimes involving previous update values and/or trying to "look ahead" to see what updates might be coming
- Recently, people have been trying this idea other than in the category of smooth maps, eg., the category of "boolean circuits".

Learning at a higher level

Let's think about this a bit more generally:

- We want to build a map
$$X \to Y.$$
- To do this, we instead look at maps of type
$$P \times X \to Y$$
(where P isn't fixed).
- The key step of the update process involves building a map of type
$$P \times X \times Y \to P$$
where P on the left is the current parameter, X is the current input, Y is the desired output, and P on the right is the updated parameter.
- (Note that this map has switched the role of input and output in some of its terms!)
- The update process itself always involves the gradient in some way.

Plan to set this up

We handle this via three categorical ideas:

1. The Para construction, which from a category with maps $X \longrightarrow Y$, builds a category whose maps are of type $P \times X \longrightarrow Y$
2. The Lens construction, which builds a category of maps with both a "forwards" and a "backwards" part, as happens in the parameter update
3. Reverse derivative categories, in which every map has an associated "reverse derivative" (which in the standard case is essentially the gradient)

And we'll see the process of taking a parameterized map

$$P \times X \longrightarrow Y$$

and producing a map of type

$$P \times X \times Y \longrightarrow P$$

as a (2-)functor.

Para construction

We want to consider maps of type $P \times A \longrightarrow B$ for varying P. This is handled by the following construction:

Definition (Gavranović 2019)

Given any Cartesian category $\mathcal{C}$, define $\mathrm{Para}(\mathcal{C})$ with the following data:

- An object is an object of $\mathcal{C}$
- A map from A to B is a pair (P, f) where P is an object of $\mathcal{C}$ and

$$f : P \times A \longrightarrow B.$$

- The composite of $(P, f) : A \longrightarrow B$ with $(Q, g) : B \longrightarrow C$ has object $Q \times P$ and map

$$Q \times P \times A \xrightarrow{1_Q \times f} Q \times B \xrightarrow{g} C$$

Note that in general this composition won't be strictly associative...we either need an equivalence on maps or, more generally, a 2-structure:

The Para construction continued

Definition

- A 2-cell from $(P, f), (P', f') : A \longrightarrow B$ is a map $\alpha : P' \longrightarrow P$ such that

$$\begin{tikzcd}
P' \times A \arrow[r, "\alpha \times 1"] \arrow[dr, "f'"'] & P \times A \arrow[d, "f"] \\
& B
\end{tikzcd}$$

- Most examples we will consider (such as the category of smooth maps between $\mathbb{R}^n$'s) are skeletal, and in those cases this construction will give a (2-)category.
- One could also build a category directly by having arrows be up to equivalence of 2-cells.
- But the 2-struture can also be useful in its own right.

Lenses

We also want to describe categories containing maps with both "forwards" and "backwards" components:

Definition (de Paiva, Rosebrugh, Johnson, Hedges...)

Given any Cartesian category $\mathcal{C}$, the category of **lenses** in $\mathcal{C}$ is a category $\mathrm{Lens}(\mathcal{C})$ with:

- An object is an object of $\mathcal{C}$
- A map from A to B is a *pair* of maps (f, f^*), where

$$f : A \longrightarrow B \text{ and } f^* : A \times B \longrightarrow A$$

Given $f : A \longrightarrow B$, $f^* : A \times B \longrightarrow A$ and $g : B \longrightarrow C$, $g^* : B \times C \longrightarrow B$, what is their composite?

Composition and identities of lenses

Definition

- The composite of $(f, f^*) : A \longrightarrow B$ with $(g, g^*) : B \longrightarrow C$ is the map

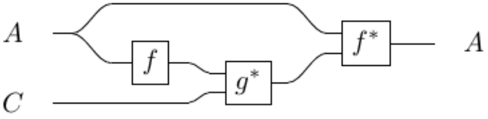
$$A \xrightarrow{f} B \xrightarrow{g} C$$

together with the map

$$A \times C \xrightarrow{\langle \pi_1, \pi_1 f, \pi_2 \rangle} A \times B \times C \xrightarrow{1 \times g^*} A \times B \xrightarrow{f^*} A$$

- The identity on A is the map $(1_A, \pi_2)$.

This project convinced me of the usefulness of string diagrams...in strings, the second component of the composite is



Lens variants

This is just one category which goes by the name of lenses. There are a variety of others (but all feature pairs of maps, one going forward, the other backwards):

- They have been used in abstract database theory, in which case one often imposes additional axioms on the maps (f, f^*).
- There is a more general version with objects pairs (A, A'), in which a map $(f,^*) : (A, A') \longrightarrow (B, B')$ has $f : A \longrightarrow B$ and

$$f^* : A \times B' \longrightarrow A'$$

- More general versions include categories of "optics".

Lens as a dual fibration

For those familiar with fibrations, there's a way to view categories of lenses in a less ad-hoc manner:

- For any fibration $F : \mathcal{E} \longrightarrow \mathcal{B}$, one can constructs its *dual fibration* $F^* : \mathcal{E}^* \longrightarrow \mathcal{B}$, in which the fibre over B of F^* is the opposite category of the fibre of F over B.
- For any Cartesian category $\mathcal{C}$, one can construct the simple fibration whose objects are pairs (C, C'), with maps $(f, f') : (C, C') \longrightarrow (D, D')$ consisting of an $f : C \longrightarrow D$ and a $f' : C \times C' \longrightarrow D'$.
- The second category of lenses on the previous slide is the dual fibration to the simple fibration.

More generally, the dual fibration of the codomain fibration is interesting and may provide insight on how to work with learning on manifolds.

Combining Para and Lens

If we do both of these constructions, that is, consider $\mathrm{Para}(\mathrm{Lens}(\mathcal{C}))$, we get a category with:

- Objects are those of $\mathcal{C}$
- A map from A to B is a pair $(P, (f, f^*))$ where P is an object of $\mathcal{C}$, and (f, f^*) is a lens from $P \times A \longrightarrow B$
- For such a pair $f : P \times A \longrightarrow B$ and

$$f^* : P \times A \times B \longrightarrow P \times A$$

This is actually more than we need to handle the maps that appear in gradient descent (for those, we just needed maps of type $P \times A \times B \longrightarrow P$), but keeping the A around is important to make everything compositional, and may well be useful for machine learning in its own right.

Neural networks and learners

So, given a Cartesian category $\mathcal{C}$, we have the categories Para($\mathcal{C}$) ("neural networks") which has a map from A to B a map of type

$$P \times A \longrightarrow B$$

and Para(Lens($\mathcal{C}$)), ("learners") which has a map from A to B containing a map of type

$$P \times A \times B \longrightarrow P \times A$$

We want to see gradient-based learning algorithms as functors

$$\mathrm{Para}(\mathcal{C}) \longrightarrow \mathrm{Para}(\mathrm{Lens}(\mathcal{C}))$$

and a key component of these functors should be a gradient operation. Thus, we need $\mathcal{C}$ to have some kind of differential structure.

Derivatives, categorically

Let's think about the type of the derivative of a smooth map

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}^m.$$

The Jacobian of f can be viewed as a map

$$J(f) : \mathbb{R}^n \longrightarrow \mathrm{Lin}(\mathbb{R}^n, \mathbb{R}^m)$$

But in general we don't want to assume our categories are closed. So instead we view the Jacobian as a map of type

$$D(f) : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}^m$$

satisfying certain properties. For example, the chain rule can be expressed as

$$D(fg) = \langle \pi_0 f, D(f) \rangle D(g)$$

Cartesian differential categories

Definition (Blute/Cockett/Sealy 2009)

A **Cartesian differential category** (CDC) is a Cartesian left additive category which has, for each map

$$f : A \longrightarrow B$$

a map

$$D[f] : A \times A \longrightarrow B$$

satisfying seven axioms, including the “chain rule”:

$$D[fg] = \langle \pi_0 f, D[f] \rangle D[g]$$

The canonical example is the category of smooth maps between $\mathbb{R}^n$’s, but there are many others, including categories in algebraic geometry, synthetic differential geometry, computer science, homotopy theory, etc.

Reverse derivatives, categorically

But actually, we don't exactly want the Jacobian. For example, given a map $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, the map

$$D[f] : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$$

takes $((x_i), (v_i))$ and gives you the point

$$\left(\frac{df}{dx_1}(x_1) + \frac{df}{dx_2}(x_2) + \ldots \frac{df}{dx_n}(x_n) \right) \cdot (v_1, v_2 \ldots v_n)$$

But what we want is the *transpose* of the Jacobian which gives a map

$$R[f] : \mathbb{R}^n \times \mathbb{R} \longrightarrow \mathbb{R}^n$$

which takes $((x_i), t))$ and gives you the vector

$$\left(\frac{df}{dx_1}(x_1), \frac{df}{dx_2}(x_2), \ldots \frac{df}{dx_n}(x_n) \right) \cdot t$$

So we want some structure which generalizes the transpose of the Jacobian.

Cartesian reverse differential categories

Definition (Cockett/Cruttwell/Gallagher/Lemay/MacAdam/Plotkin/Pronk 2020)

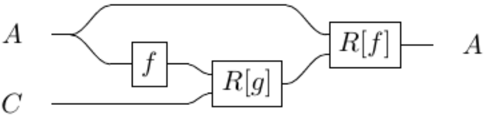
A **Cartesian reverse differential category** (CRDC) is a Cartesian left additive category which has, for each map $f : A \longrightarrow B$, a map

$$R[f] : A \times B \longrightarrow A$$

satisfying seven axioms, including the “reverse chain rule”:

$$R[fg] = \langle \pi_1, \pi_1 f, \pi_2 \rangle (1 \times R[g]) R[f]$$

In strings, the reverse chain rule is



which should look familiar - it's how composition in Lens($\mathcal{C}$) was defined!

More on CRDCs

- The category of smooth maps between $\mathbb{R}^n$'s is a Cartesian reverse differential category, with $R[f]$ the transpose of the Jacobian.
- The category of Boolean circuits (see the paper "Reverse derivative ascent") is a CRDC.
- Any Cartesian differential category with a "contextual linear dagger" (dagger category structure in each linear slice) is a Cartesian reverse differential category.
- In fact, this is an equivalence: CRDC's are exactly CDC's with a contextual linear dagger. See "Reverse derivative categories" for more details.

R as a functor

The important point for our setup, however, is that if $\mathcal{C}$ is a CRDC, then there is a functor

$$R^* : \mathcal{C} \longrightarrow \mathsf{Lens}(\mathcal{C})$$

which is the identity on objects, and maps $f : A \longrightarrow B$ in $\mathcal{C}$ to the pair

$$(f, R[f]) : A \longrightarrow B \text{ in } \mathsf{Lens}(\mathcal{C}).$$

And, moreover, Para is itself a 2-functor, so we get an induced 2-functor

$$\mathsf{Para}(R^*) : \mathsf{Para}(\mathcal{C}) \longrightarrow \mathsf{Para}(\mathsf{Lens}(\mathcal{C}))$$

This is the first component of gradient descent algorithms!

Conclusions

Thus, our setup initially consists of:

- A CRDC $\mathcal{C}$
- From this, we can build $\mathrm{Para}(\mathcal{C})$, the (bi-)category of parameterized functions, which include neural networks.
- From this, we can build $\mathrm{Para}(\mathrm{Lens}(\mathcal{C}))$, the (bi-)category of "learners".
- And we already have a 2-functor

  $$\mathrm{Para}(R^*) : \mathrm{Para}(\mathcal{C}) \longrightarrow \mathrm{Para}(\mathrm{Lens}(\mathcal{C}))$$

  which provides the gradient operation used in machine learning algorithms.

Next time, we'll see how to enhance the functor $\mathrm{Para}(R^*)$ with "error" and "update" maps; varying these error and update maps will then give many of the gradient descent algorithms used in practice.

References

The paper the talk is based on is

- Cruttwell, G., Gavranović, B., Ghani, N., Wilson, P., and Zanasi, F. **Categorical foundations of gradient-based learning**, arXiv:2103.01931, 2021.

The most closely related categorical work is

- Fong, B., Spivak, D.., and Tuyeras, R. **Backprop as functor**, LICS 2019.

Other references include

- Cockett, R., Cruttwell, G., Gallagher, J., Lemay, J-S., MacAdam, B., Plotkin, G., and Pronk, D. **Reverse derivative categories**. CSL 2020.
- Gavranović, B. **Composition deep learning**, arXiv:1907.08292, 2019.
- Hedges, J. *Lenses for philosophers* (blog post at https://julesh.com/2018/08/16/lenses-for-philosophers/, contains many other references and history about lenses)
- Wilson, P. and Zanasi, F. **Reverse derivative ascent: a categorical approach to learning boolean circuits**. Proceedings of ACT 2020.