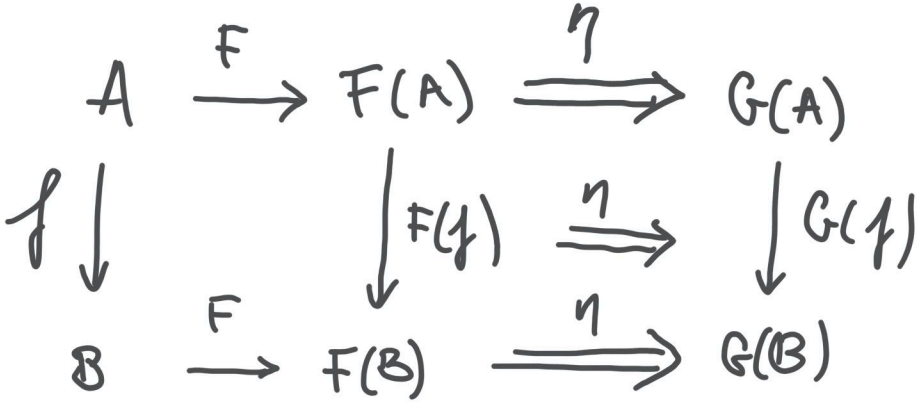
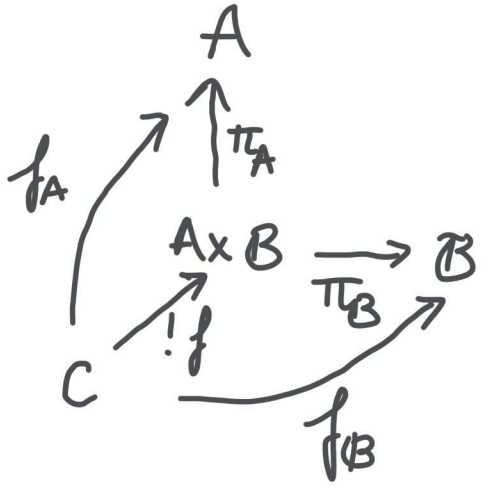
EAI Math Reading Group

Category Theory





(Why) Do you need Category Theory ?

Why do you need

- Type systems ?
  - Algebraic Data Types
  - Polymorphic types (generics)
  - typeclasses
- Functions ?
- Compilers ?

You don't <u>need</u> abstraction, but it is <u>useful</u>

Outline

1) Category theory 101

2) Example: Gradient-based learning

[3) Example: Automatic differentiation]

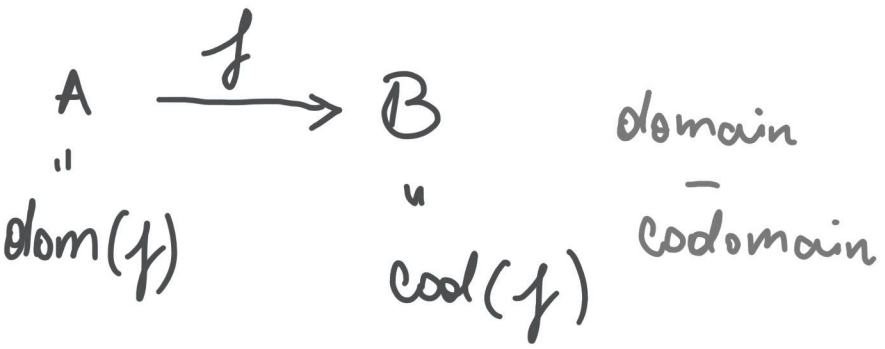
4) "A Monad is a Monoid in the category of endofunctors"

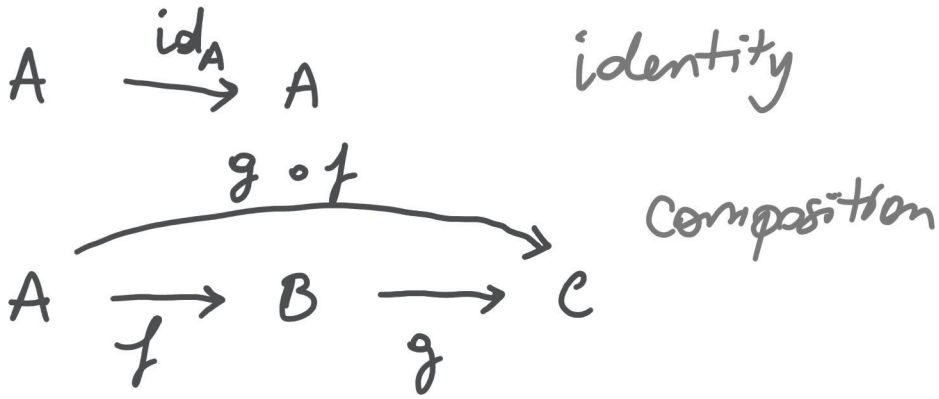
Categories

"$\mathcal{C} = (Ob(\mathcal{C}), Mor(\mathcal{C}))$"

objects $Ob(\mathcal{C})$ $A, B, C, \ldots$

morphisms $Mor(\mathcal{C})$ (arrows) $f, g, h, \ldots$





$$id_B \circ f = f = f \circ id_A$$

$$f \circ (g \circ h) = (f \circ g) \circ h$$

Examples

- **Sets**
  - objects: sets
  - arrows: functions

- $(\mathbb{R}, \leq)$
  - objects: $x \in \mathbb{R}$
  - arrows: "$x \leq y$"

→ more generally, any partially ordered Set

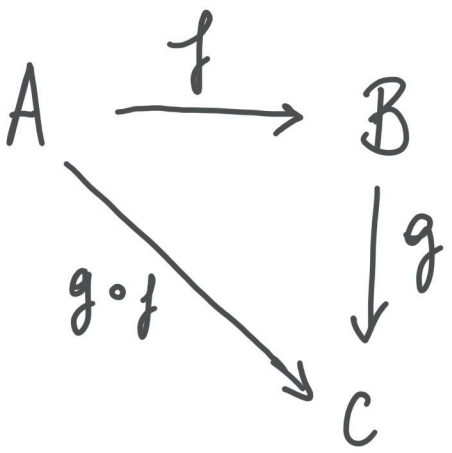
- $\mathrm{Vect}_{\mathbb{R}}$ ($\mathbb{R}^n$ $\mathbb{R}^d$)
  - objects: vector spaces over $\mathbb{R}$
  - arrows: linear maps

- "Structured sets"
  - groups
  - topological spaces
  - differential manifolds
  - ...

- (Haskell) types $a :: A$, $b :: B$, $f :: A \to B$

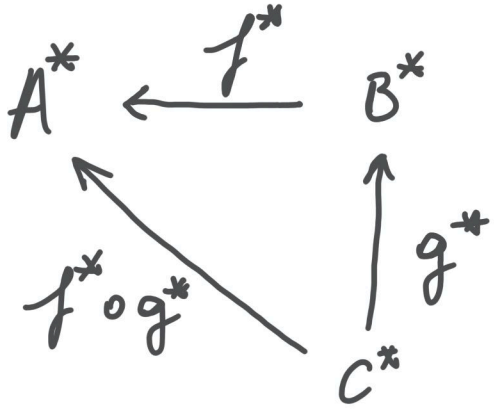
Opposite category (aka "dual category")

$\mathcal{C}$



$\mathcal{C}^{op}$

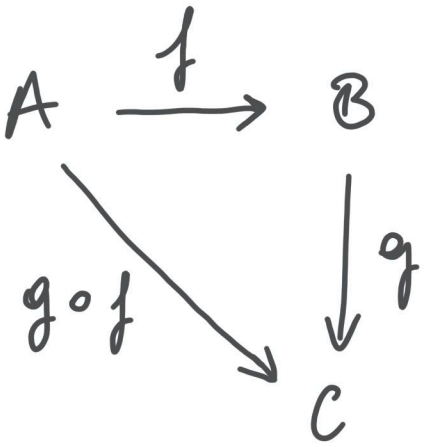
$Ob(\mathcal{C}^{op}) = Ob(\mathcal{C})$

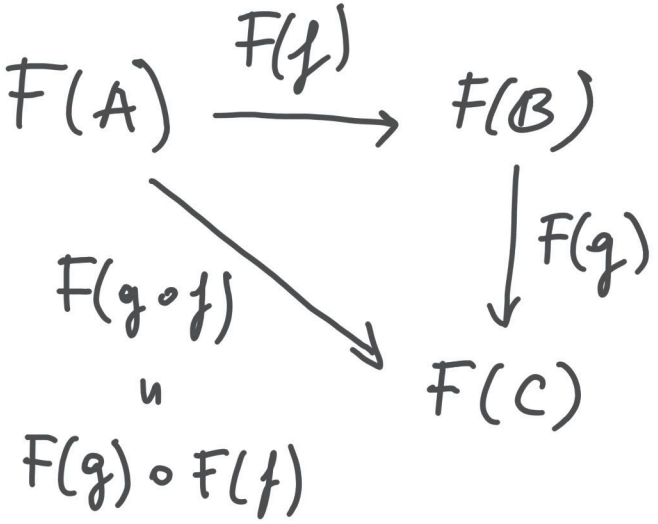


"Duality principle": Every definition, theorem, ... in $\mathcal{C}$ has a dual equivalent in $\mathcal{C}^{op}$

Functors

$$F : \mathcal{C} \longrightarrow \mathcal{D}$$



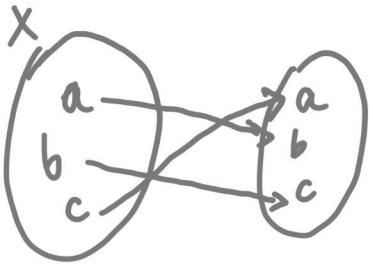


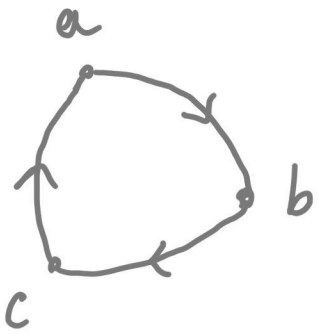
" Functors are just morphisms of categories "

Examples

- discrete dynamical systems $\longrightarrow$ directed graphs

$$(X, f: X \to X) \longmapsto (X, \{(x, f(x)) \mid x \in X\})$$





- Polymorphic types
  - List `[a]`, Maybe a

- dual vector space

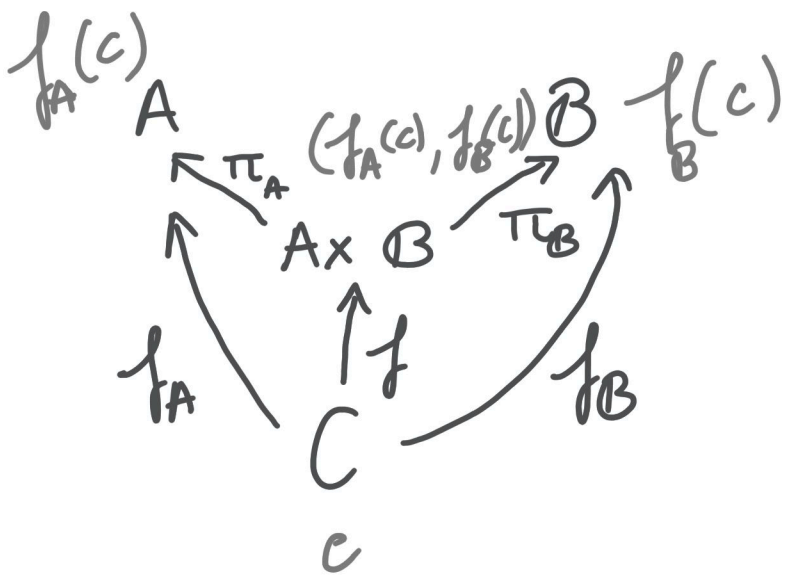
$$V \longmapsto V^* = \{g: V \to \mathbb{R}, f \text{ linear}\}$$

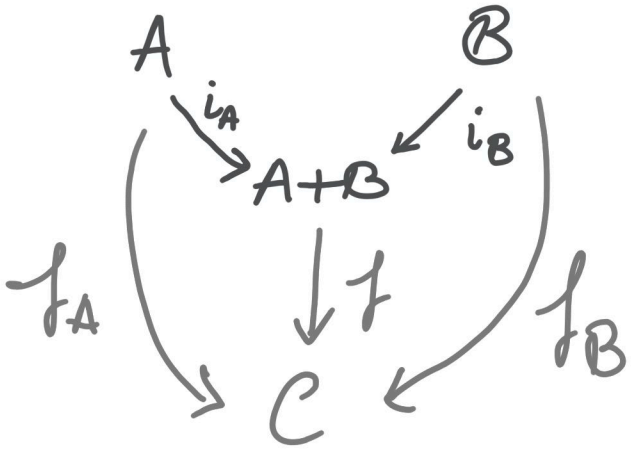
$$\mathrm{Vect}_{\mathbb{R}}^{op} \to \mathrm{Vect}_{\mathbb{R}}$$

$$\begin{array}{ccc} V & \longrightarrow & V^* \\ f \downarrow & & \uparrow f^* \\ W & \longrightarrow & W^* \end{array}$$

$$[f^*(g)](v) = g(f(v))$$

Products and coproducts





Examples

Sets: cartesian product

Types: "product" type (tuples)

(a, b)

disjoint union

"sum" types (unions, enums)

Either a b

Monoidal Categories

A (strict) monoidal category is a category $\mathcal{C}$ with:

- a functor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$
- a unit $I \in Ob(\mathcal{C})$

s.t.

$$A \otimes (B \otimes C) \simeq (A \otimes B) \otimes C$$

$$(a, (b, c)) \simeq ((a, b), c)$$

$$A \otimes I \simeq A \simeq I \otimes A$$

$$(a, ()) \quad a \quad ((), a)$$

Categories with finite products are monoidal

Application: Gradient Based learning

Goal : model Machine learning models trained by gradient descent

⇝ programs

We need :

- parameters
- data
- gradients (backprop)
- parameter updates

⇝ programs that take parameters, data, ... and modify them

Parameterized category

For a [strict] monoidal category $(\mathcal{C}, \otimes, I)$

$\mathbf{Para}(\mathcal{C})$ → parameterized over $\mathcal{C}$

objects: $Ob(\mathcal{C})$

arrows (P, f) $\quad f : P \otimes A \to B$

$\downarrow$

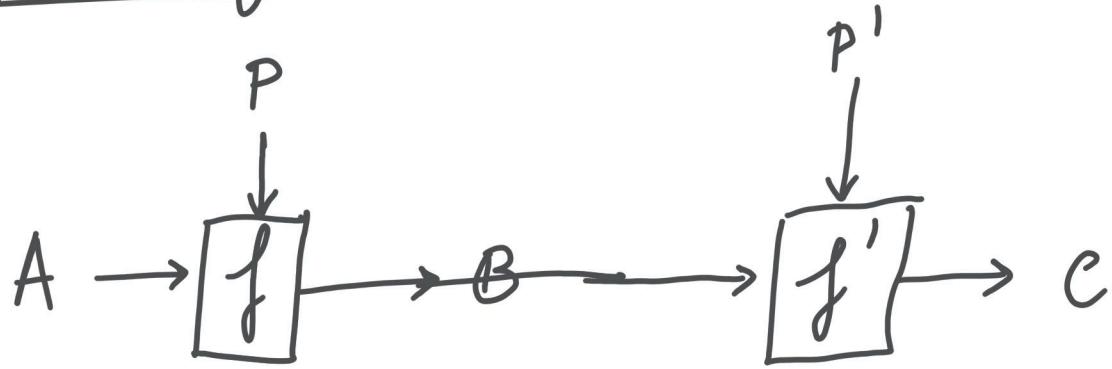
$\in \mathcal{C}$

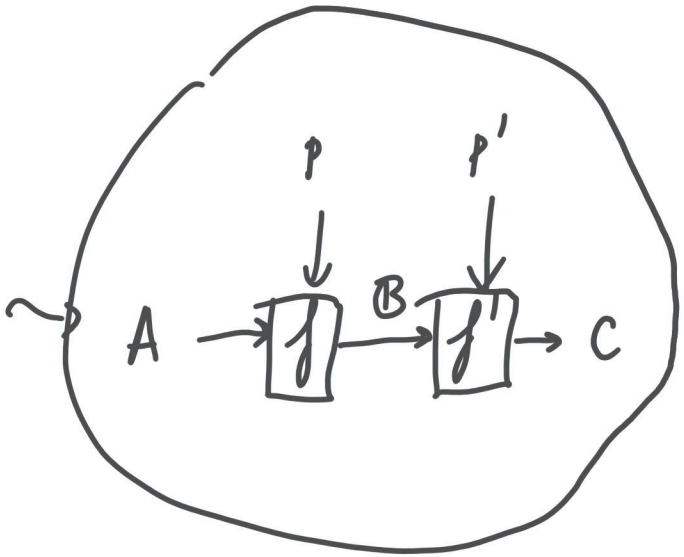
$id_A = (I, id_A)$

$(P, f) ; (P', g) = (P' \otimes P, (1 \otimes f) ; g)$

$(P', g) \circ (P, f)$

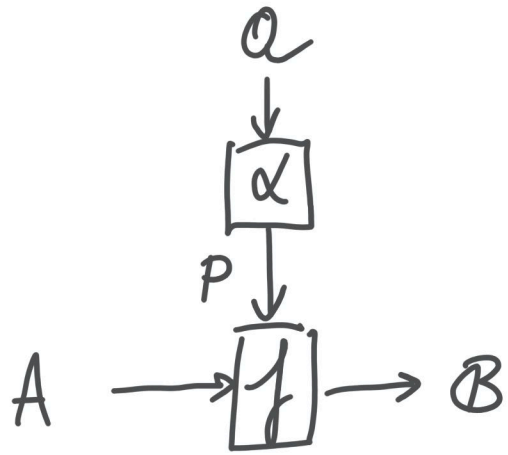
String diagrams





. Reparameterization

$$\left.\begin{array}{l}(P, f) : A \to B \\ \alpha : Q \to P\end{array}\right\} (Q, (\alpha \otimes 1_A); f)$$



Lenses (↝ parameter updates)

For a Cartesian category $\mathcal{C}$

$\mathbf{Lens}(\mathcal{C})$

objects (A, A') $\quad (A, A' \in \mathcal{C})$

A ↓ state, A' ↘ update

arrows $(f, f^*) : (A, A') \to (B, B')$

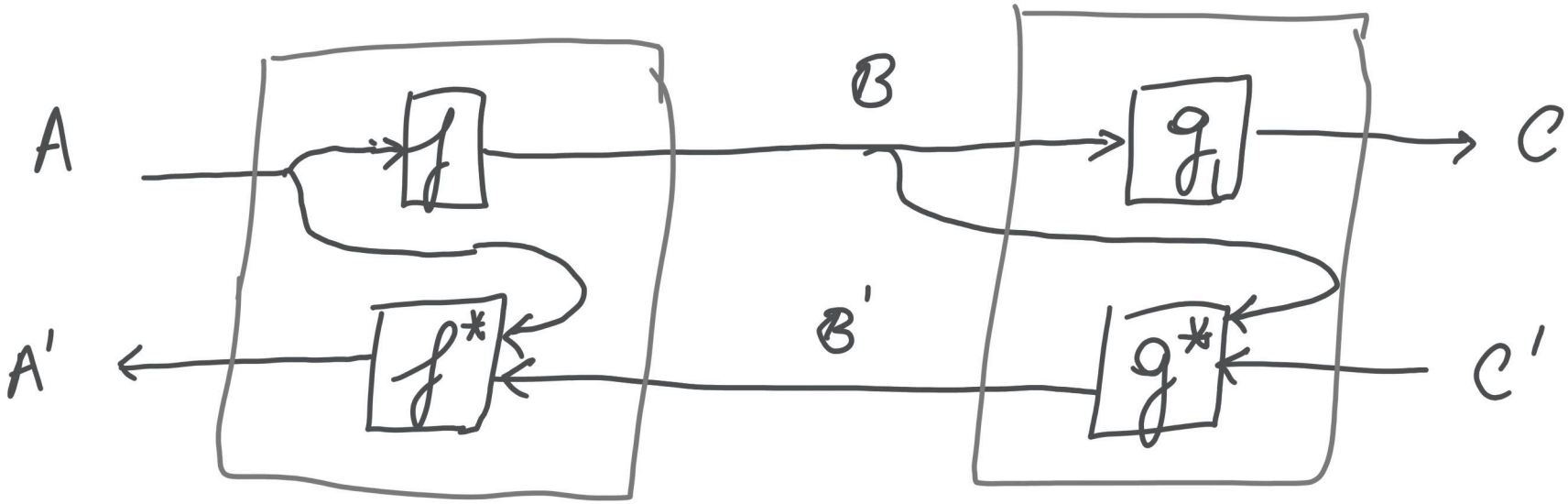
$f : A \to B$

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

Composition of morphisms in Lens(C)

$$(g, g^*) \circ (f, f^*) = (g \circ f,$$

$$\langle \pi_0, \langle \pi_0; f, \pi_1 \rangle; g^* \rangle; f^*$$



$$(a, c') \longmapsto f^*(a, g^*(f(a), c'))$$

Parametric lenses

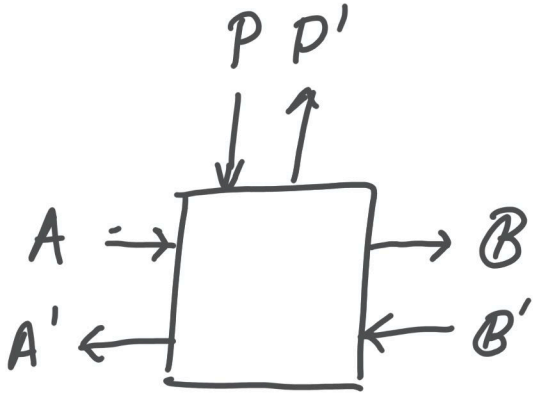
$\mathbf{Para}(\mathbf{Lens}(\mathcal{C}))$

objects : (A, A')

arrows : $(f, f^*) : (A, A') \times (P, P') \to (B, B')$

$f : P \times A \to B$

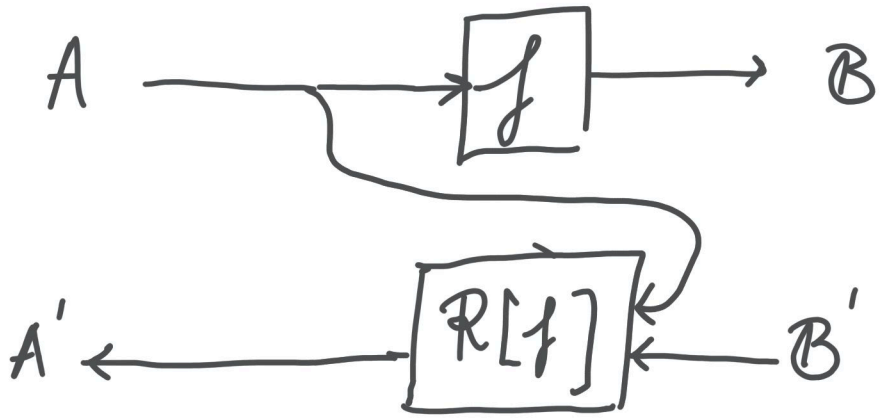
$f^* : P \times A \times B' \to P' \times A'$



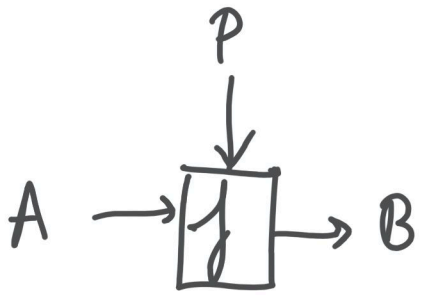
Cartesian Reverse Differential category

to each map $f: A \to B$, we have

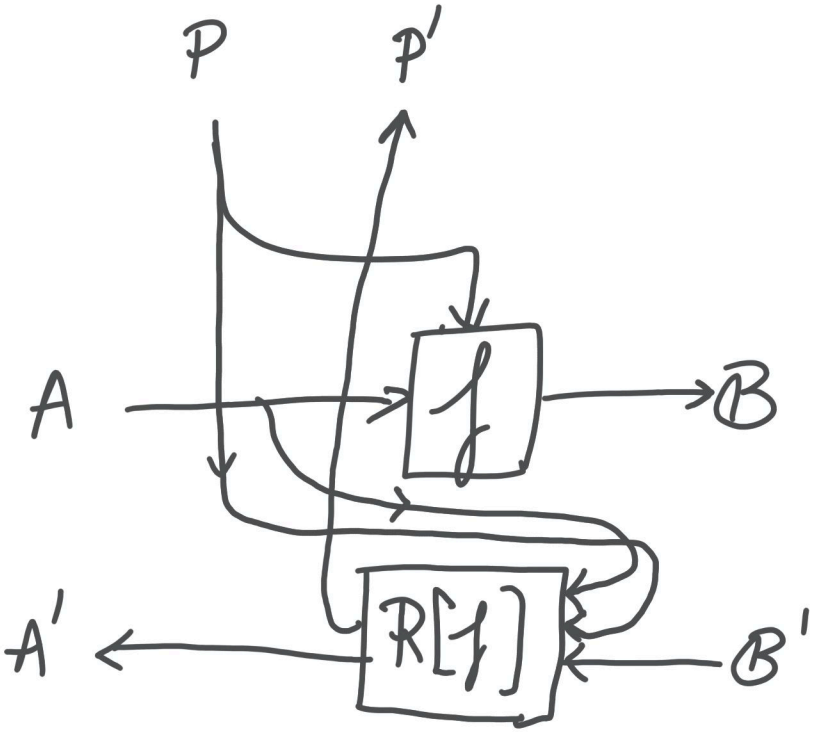
$$R[f]: A \times B' \to A' \qquad \text{(backpropagation)}$$



Models as parametric lenses

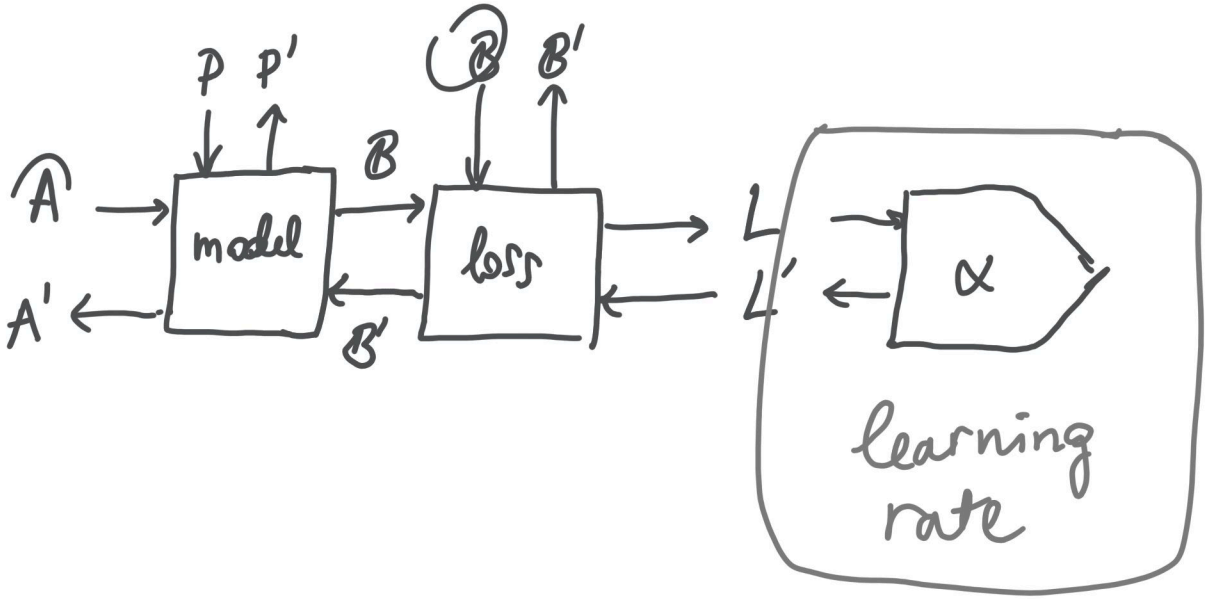


Para(R) ⟼



Components :

Loss map



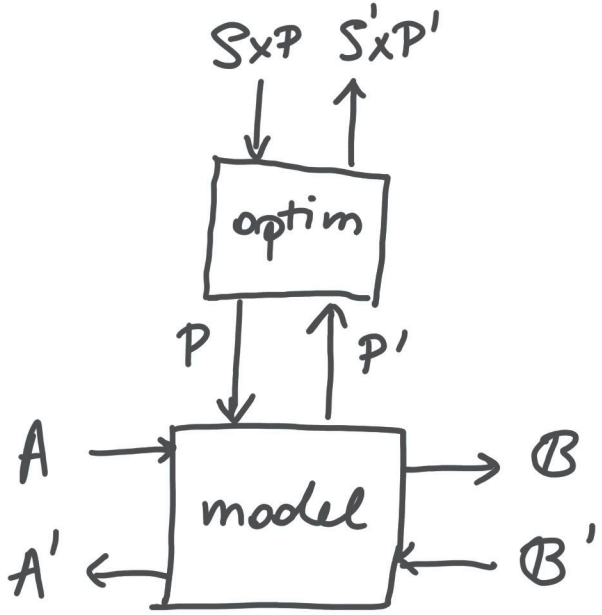
Optimizers as Reparameterizations

- gradient update

$$(G, G^*) : (P, P') \to (P, P')$$

$$G(\phi) = \phi$$

$$G^*(\phi, \phi') = \phi + \phi'$$



- Stateful parameter update

$$U : (S \times P, S \times P) \longrightarrow (P, P')$$

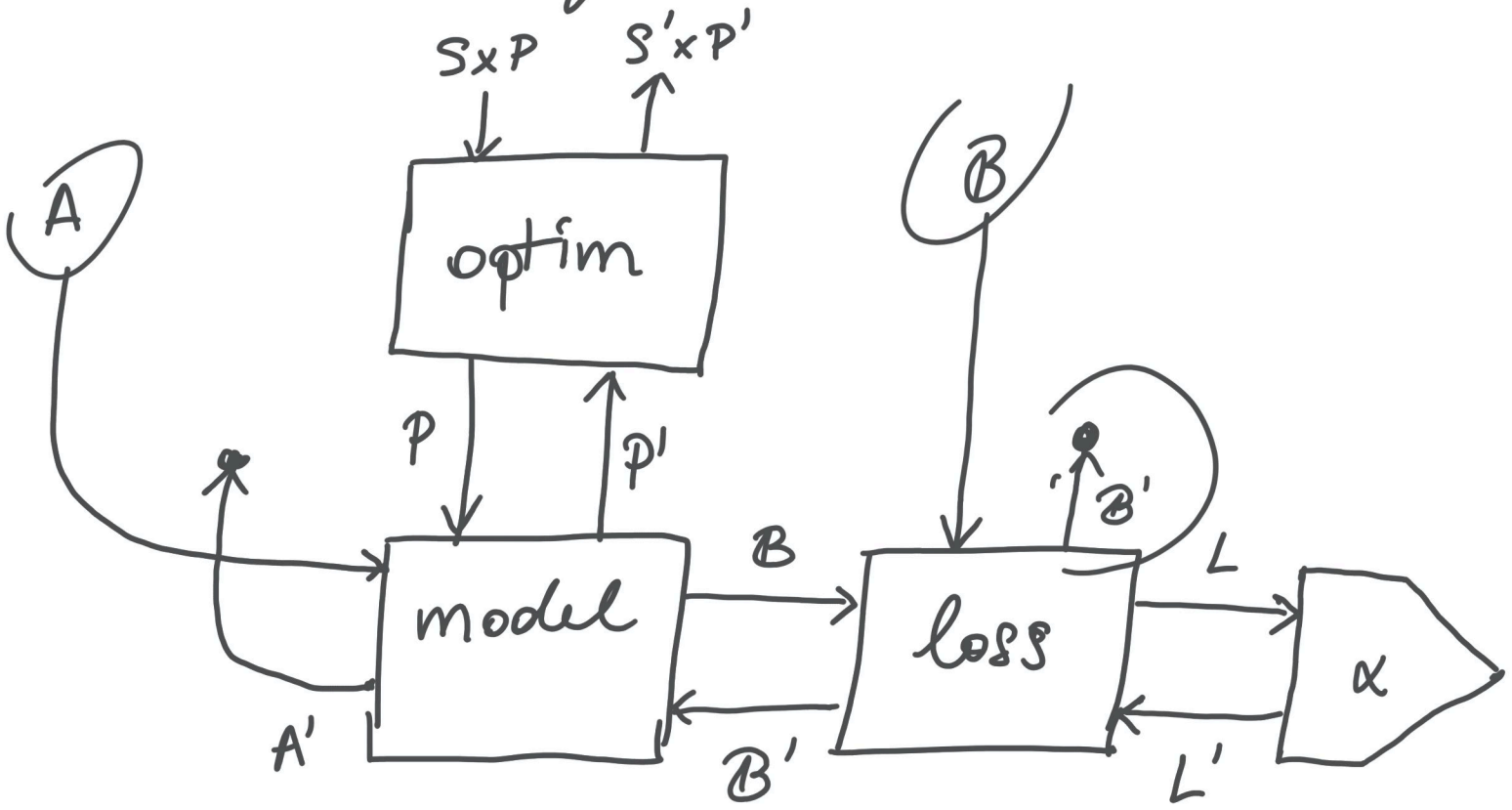
e.g. momentum $(U, U^*) : (P \times P, P \times P) \to (P, P')$

$$U(s, \phi) = \phi$$

$$U^*(s, \phi, \phi') = (s', \phi + s')$$

$$\hookrightarrow -\gamma s + \phi'$$

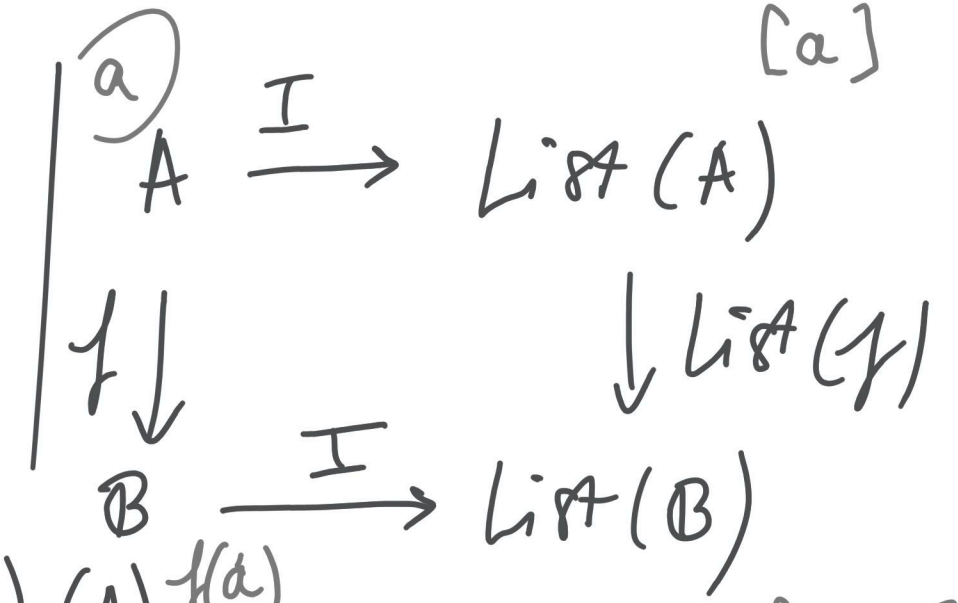
Supervised Learning



Bonus: "A Monad is a Monoid in the category of endofunctors"

List : A → List(A)

List(List(A))



(M, *, I)

↳ (a * b) * c = a * (b * c)

(List ∘ List)(A) → f(a) → flatMap

I * a = a = a * I