CATEGORICAL FOUNDATIONS OF GRADIENT-BASED LEARNING

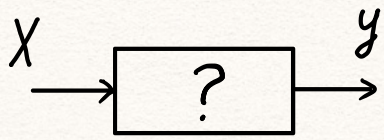
(CRUTTWELL, GAVRANOVIĆ, GHANI, WILSON, ZANASI)

GOAL:
PROVIDE A CATEGORICAL FRAMEWORK

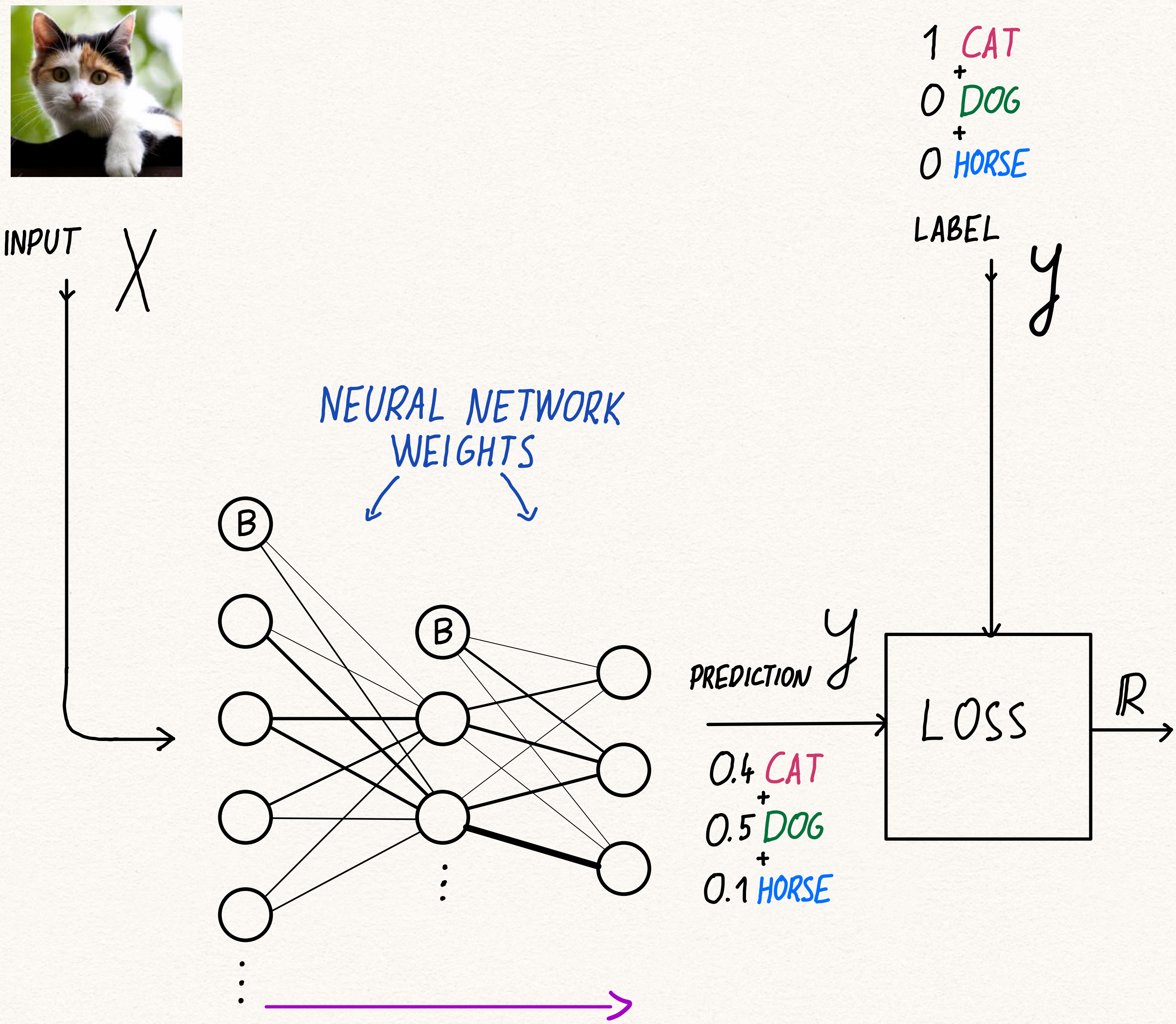
—

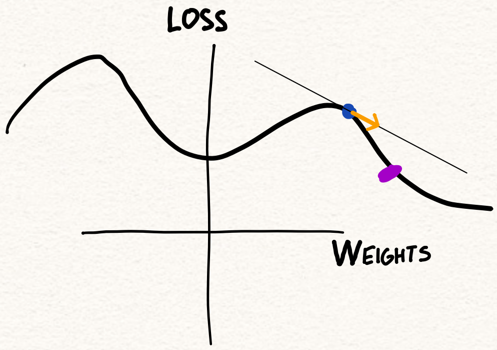
FOR DEEP LEARNING

SUPERVISED LEARNING WITH NEURAL NETWORKS
IN ONE SLIDE:



DATASET : List $X \times Y$





GRADIENT DESCENT
~
„OPTIMIZER"

- NN IS COMPUTATION PARAMETERIZED BY WEIGHTS
- BACKPROPAGATION OF CHANGES
- PARAMETER UPDATE – „OPTIMIZERS"

NEURAL NETWORKS {
- LINEAR LAYER
- BIAS TERM
- ACTIVATION FUNCTION

THIS SIMPLE STORY PERMEATES DEEP LEARNING!

PLAN FOR TODAY?

TAKE A BIRD'S EYE VIEW OF NEURAL NETWORKS

- TRACE OUT THE INFORMATION FLOW ABOVE
- PRECISELY WRITE DOWN ALL THE HIGH-LEVEL NOTIONS IN ISOLATION:
  - DIFFERENTIATION – REVERSE DERIVATIVE CATS.
  - BIDIRECTIONALITY – OPTICS/LENSES
  - PARAMETERIZATION – PARA

  AND STUDY THEIR INTERACTION.

PARAMETERIZED OPTICS

AS A COMMON STRUCTURE BEHIND

- NEURAL NETWORKS
- LOSS FUNCTIONS
- OPTIMIZERS

- PAUL: CONCRETE EXAMPLES OF NEURAL NETWORKS

DIFFERENTIATION

- CARTESIAN (FORWARD) DIFFERENTIAL CATEGORIES (Blute et. al.)
- CARTESIAN REVERSE DIFFERENTIAL CATEGORIES (CRDC) (Cockett et. al.)

DEFINITION.

A CRDC $\mathcal{C}$ is a Cartesian left-additive category which for every map

$$f : A \longrightarrow B$$

has a REVERSE DIFFERENTIAL COMBINATOR

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

(compare $D[f] : A \times A \longrightarrow B$)

subject to 7 axioms.

EXAMPLE. Smooth is a CRDC. $\mathrm{Poly}_{\mathbb{Z}_2}$ IS A CRDC.

EXAMPLE. Let $\mathbb{R}^2 \xrightarrow{f} \mathbb{R}$ in Smooth.

$$(x, y) \longmapsto x^2 + 3yx$$

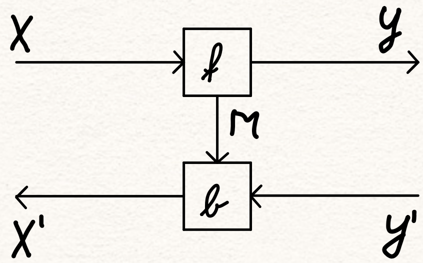
Then $R[f] : \mathbb{R}^2 \times \mathbb{R} \longrightarrow \mathbb{R}^2$

$$(x, y), w \longmapsto (2xw, 3xw)$$

PLAN: STUDY CRDC'S THROUGH OPTICS/LENSES

OPTICS / LENSES

DEFINITION. Let $\mathcal{C}$ be a SMC. Category $\mathrm{Optic}(\mathcal{C})$:



- Objects – pairs of objects $\binom{X}{X'}$ in $\mathcal{C}$

- $\mathrm{Optic}(\mathcal{C})\left(\binom{X}{X'}, \binom{Y}{Y'}\right) = \int^{M:\mathcal{C}} \mathcal{C}(X, Y \otimes M) \times \mathcal{C}(Y' \otimes M, X')$ ← COEND

(M, f, b) $\quad f: X \longrightarrow Y \otimes M$

$b: Y' \otimes M \longrightarrow X'$

PROP. If $\mathcal{C}$ is Cartesian,

$$\int^{M:\mathcal{C}} \mathcal{C}(X, M \times Y) \times \mathcal{C}(M \times Y', X')$$

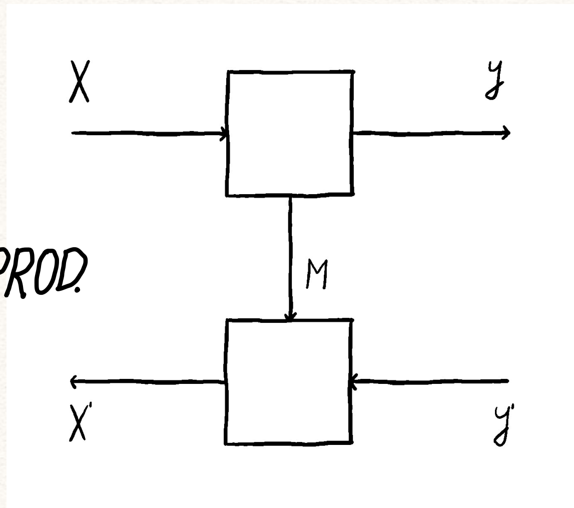
$\cong$ UNIV. PROPERTY OF PROD.

$$\int^{M:\mathcal{C}} \mathcal{C}(X, Y) \times \mathcal{C}(X, M) \times \mathcal{C}(M \times Y', X')$$

$\cong$ YONEDA REDUCTION

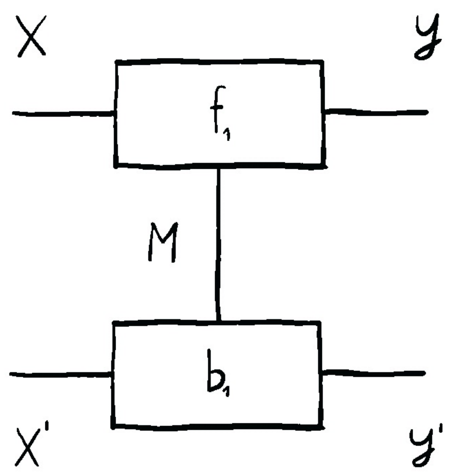
$$\int \underbrace{\mathcal{C}(X, Y)}_{get} \times \underbrace{\mathcal{C}(X \times Y', X')}_{put}$$

then $\mathrm{Optic}(\mathcal{C}) \cong \mathrm{Lens}(\mathcal{C})$



BIDIRECTIONAL INFORMATION FLOW

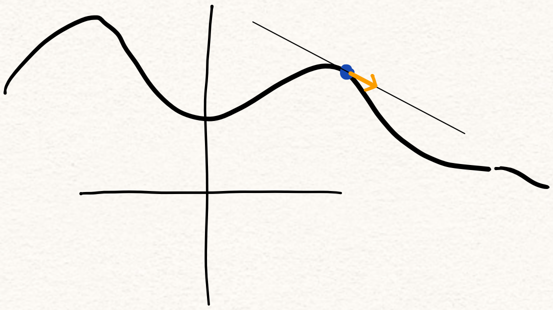
OPTICS CAN BE COMPOSED



PROPOSITION. Optic($\mathcal{C}$) is symmetric monoidal.

EXAMPLE.

GRADIENT DESCENT



$$P \times P' \xrightarrow{u} P$$
$$(p, \nabla p) \longmapsto p - \alpha \nabla p$$

is a lens, for $\mathcal{C} := \mathbf{Smooth}$

$$\begin{pmatrix} P \\ P \end{pmatrix} \xrightarrow{(id_P, u)} \begin{pmatrix} P \\ P' \end{pmatrix}$$

EXAMPLE. STATEFUL OPTIMIZERS

INTERNAL STATE

$$\begin{pmatrix} S \times P \\ S \times P \end{pmatrix} \longrightarrow \begin{pmatrix} P \\ P' \end{pmatrix}$$

- MOMENTUM,

  get: $P \times P \longrightarrow P$

  $(v, p) \longmapsto p$

  put: $P \times P \times P \longrightarrow P \times P$

  $(v, p, \nabla p) \longmapsto (v', p - v')$

  where $v' = \gamma v + \varepsilon p'$

- NESTEROV MOMENTUM

  get: $P \times P \longrightarrow P$

  $(v, p) \longmapsto p - \gamma v$

  put – same as above

- ADAGRAD
- ADAM

...

BACK TO CRDC's:

$f : A \longrightarrow B$ $\sim$ 'get' MAP OF A LENS

$R[f] : A \times B \longrightarrow A$ $\sim$ 'put' MAP OF A LENS

PROPOSITION.

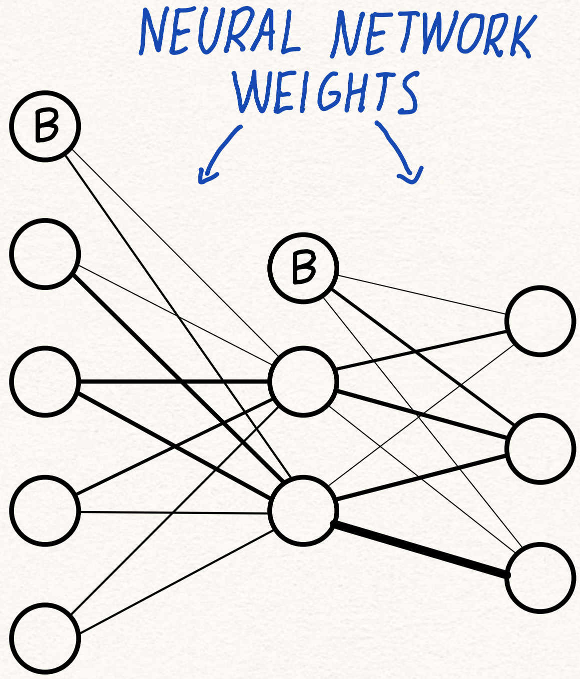
For each CRDC $\mathcal{C}$ there is a symmetric monoidal functor

$$\mathcal{C} \xrightarrow{\;F\;} \mathrm{Lens}(\mathcal{C}) \cong \mathrm{Optic}(\mathcal{C})$$

$$\begin{array}{ccc} A & \longmapsto & (A, A) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle (f, R[f])} \\ B & \longmapsto & (B, B) \end{array}$$

- THIS IS OUR FRAMEWORK FOR BACKPROPAGATION

PARAMETERIZATION



Fix a SMC $(\mathcal{C}, \otimes, I)$.

DEF. Bicategory $\mathrm{Para}(\mathcal{C})$

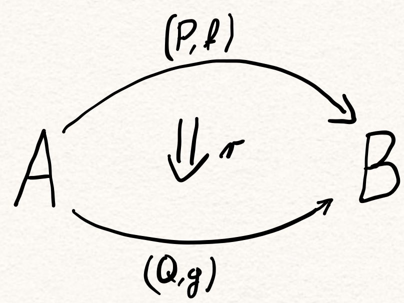
Objects – objects of $\mathcal{C}$

$$\mathrm{Para}(\mathcal{C})(A,B) = \int^{P:\mathcal{C}} \mathcal{C}(P\otimes A, B)$$

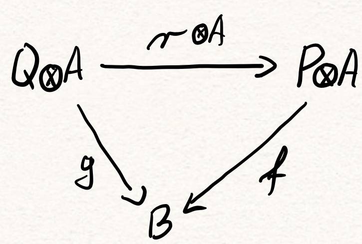
(CATEGORY OF ELEMENTS)

$$A \xrightarrow{(P:\mathcal{C},\ f:P\otimes A \to B)} B$$

2-cells are reparameterizations: a 2-cell



is a map $Q \xrightarrow{r} P$ such that



commutes.

EXAMPLE.

$(\mathbf{Set}, \times, 1)$ SETS AND

$\mathbf{Para}(\mathbf{Set})$ PARAMETERIZED FUNCTIONS

$(\mathbf{Smooth}, \times, 1)$ EUCLIDEAN SPACES AND

$\mathbf{Para}(\mathbf{Smooth})$ PARAMETERIZED SMOOTH FUNCTIONS

$(\mathbf{Optic}(\mathcal{C}), \otimes, 1)$ PAIRS OF OBJECTS AND

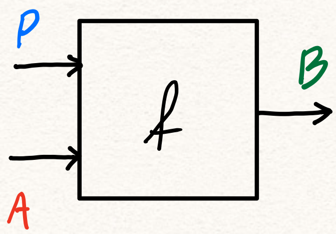
$\mathbf{Para}(\mathbf{Optic}(\mathcal{C}))$ PARAMETERIZED OPTICS

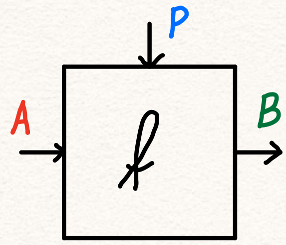
. . .

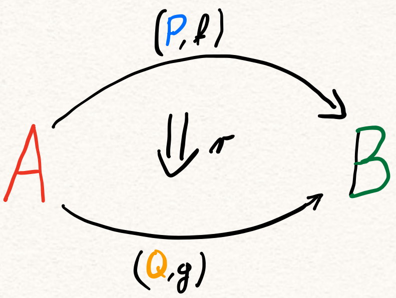
GRAPHICAL LANGUAGE

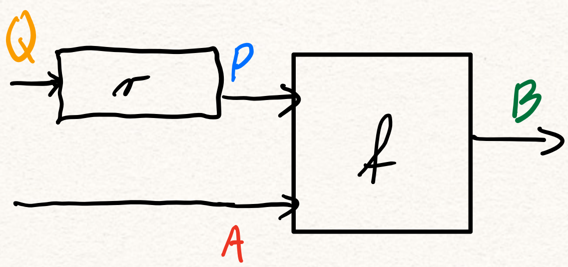
TEXTUAL NOTATION	STANDARD STRING DIAGRAM	2D STRING DIAGRAM

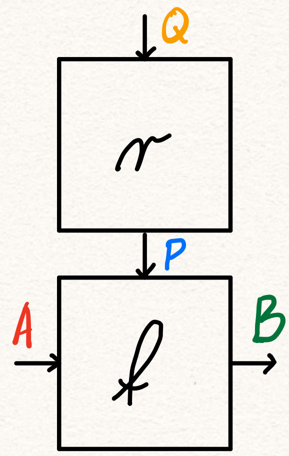
$f : P \otimes A \longrightarrow B$



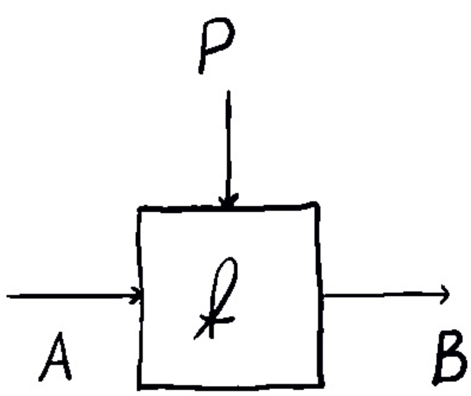




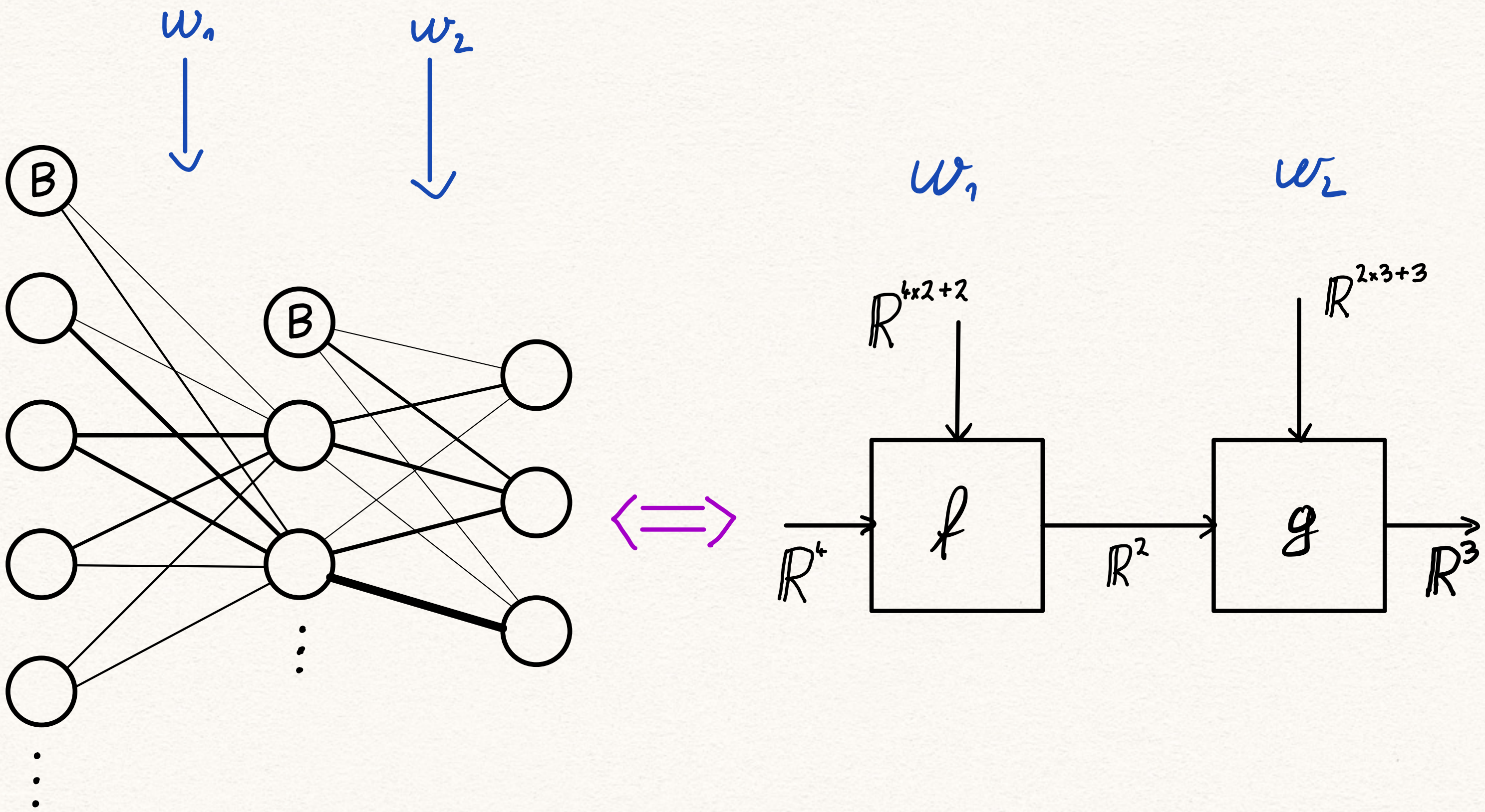




HOW DOES COMPOSITION WORK?



RECAP



Para IS NATURAL WITH RESPECT TO BASE CHANGE.

DEFINITION.

Let $G : \mathcal{C} \longrightarrow \mathcal{D}$ be a symm. monoidal functor. We define

$$\begin{array}{ccc} \mathbf{Para}(G) : \mathbf{Para}(\mathcal{C}) & \longrightarrow & \mathbf{Para}(\mathcal{D}) \\ A & \longmapsto & GA \\ {\scriptstyle (P,f)} \downarrow & & \downarrow {\scriptstyle (GP,\, f')} \\ B & \longmapsto & GB \end{array}$$

where f' is the composite

$$G(P) \otimes G(A) \xrightarrow{\mu_{P,A}} G(P \otimes A) \xrightarrow{G(f)} G(B)$$

+ MORE.

Para IS RICH IN CATEGORICAL STRUCTURE.

- CoKleisli category of a graded comonad
- Double category
- Actegorical Para

…

PARAMETERIZED OPTICS

$$\mathcal{C} \longmapsto \mathbf{Optic}(\mathcal{C}) \longmapsto \mathbf{Para}(\mathbf{Optic}(\mathcal{C}))$$

- Objects – objects of $\mathbf{Optic}(\mathcal{C})$ – pairs $\begin{pmatrix} X \\ X' \end{pmatrix}$ in $\mathcal{C}$

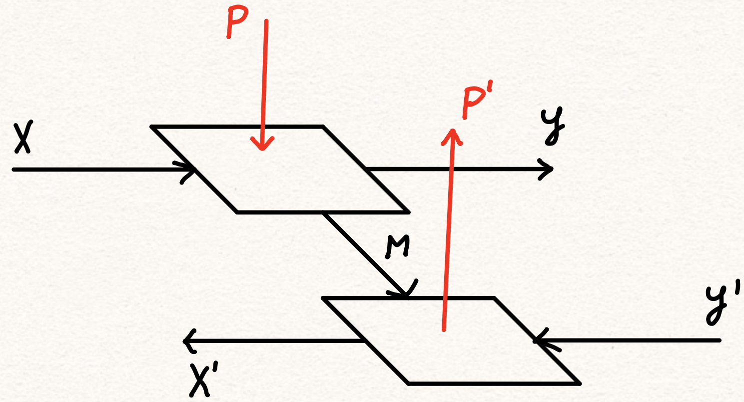
- Morphisms $\begin{pmatrix} X \\ X' \end{pmatrix} \xrightarrow{\left(\begin{pmatrix} P \\ P' \end{pmatrix}, f\right)} \begin{pmatrix} Y \\ Y' \end{pmatrix}$ where $f : \begin{pmatrix} P \otimes X \\ P' \otimes X' \end{pmatrix} \longrightarrow \begin{pmatrix} Y \\ Y' \end{pmatrix}$

  (M, f, b)

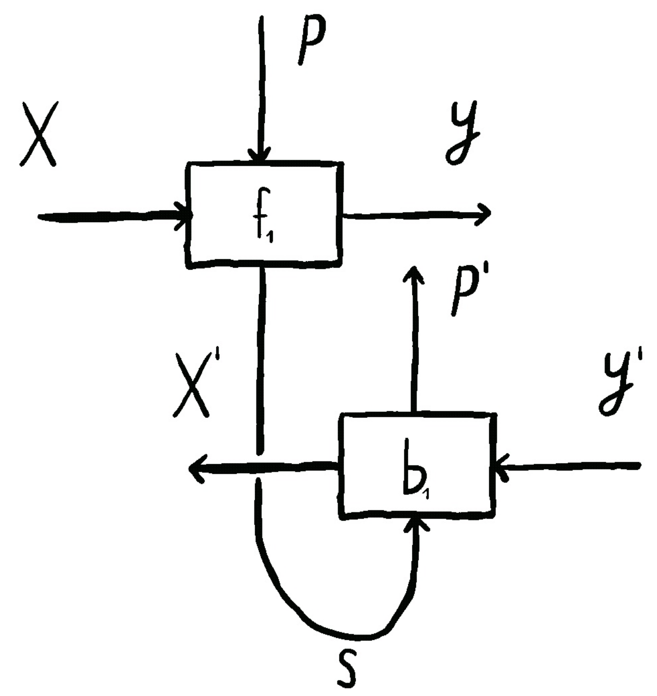
  $M : \mathcal{C}$

  $f : P \otimes X \longrightarrow Y \otimes M$

  $b : Y' \otimes M \longrightarrow P' \otimes X'$



- WE CAN COMPOSE PARAMETERIZED OPTICS



- We automatically get two parameter ports

- A 2-cell $\binom{X}{S} \overset{\binom{P}{Q},f}{\underset{\binom{Z}{W},g}{\Downarrow r}} \binom{Y}{R}$ is an <u>optic</u>

$$\binom{Z}{W} \xrightarrow{r} \binom{P}{Q}$$

THEOREM.

GRADIENT DESCENT IS A 2-cell IN $\mathrm{Para}(\mathrm{Optic}(\mathcal{C}))$.

(Since it is a lens)

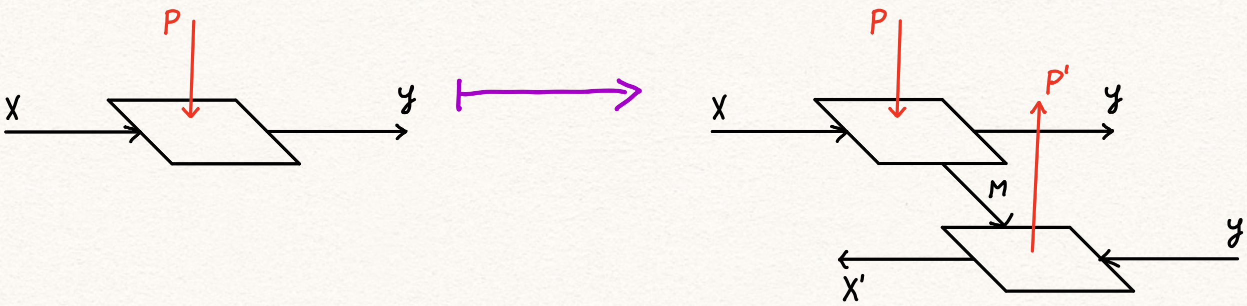
THEOREM.

APPLYING Para TO THE CRDC FUNCTOR

$$\mathcal{C} \xrightarrow{F} \mathbf{Optic}(\mathcal{C})$$

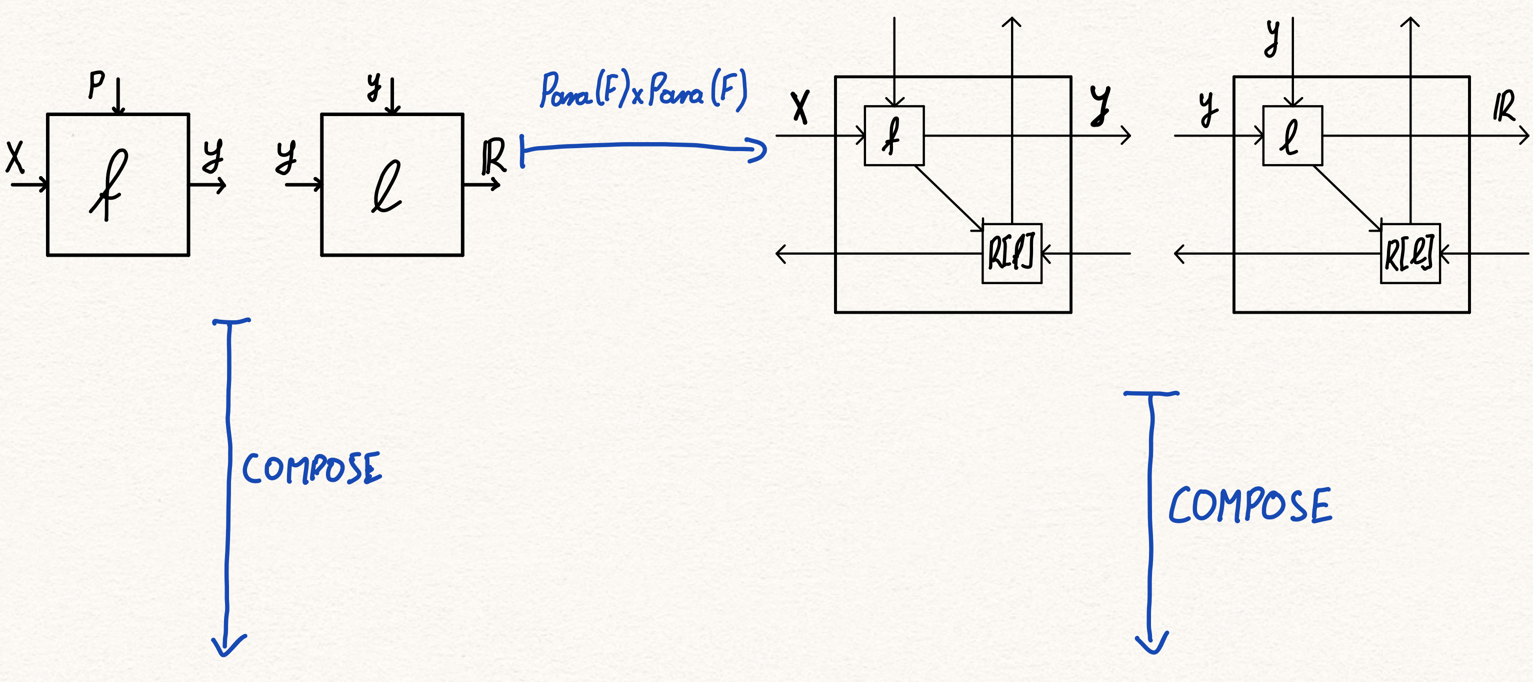
RESULTS IN A FUNCTOR

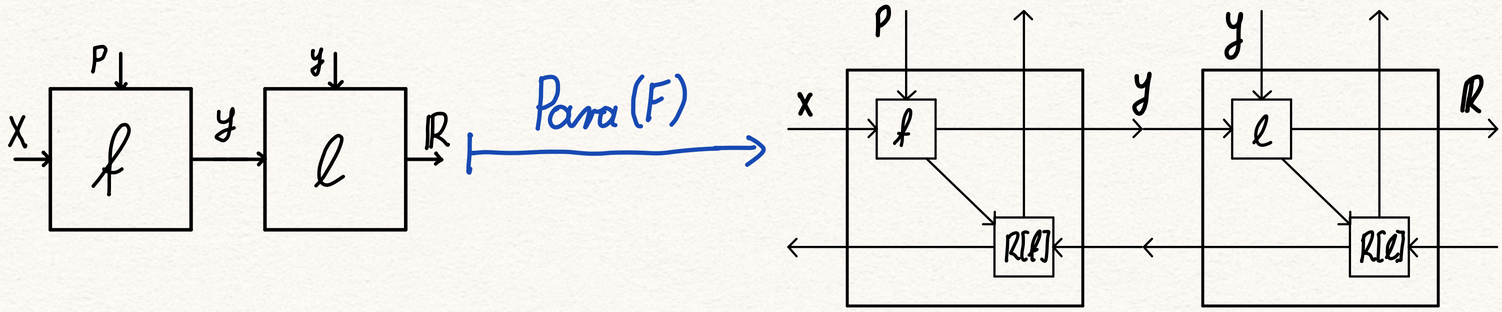
$$\mathbf{Para}(\mathcal{C}) \xrightarrow{\mathbf{Para}(F)} \mathbf{Para}(\mathbf{Optic}(\mathcal{C}))$$



• FUNCTORIALITY IS IMPORTANT!

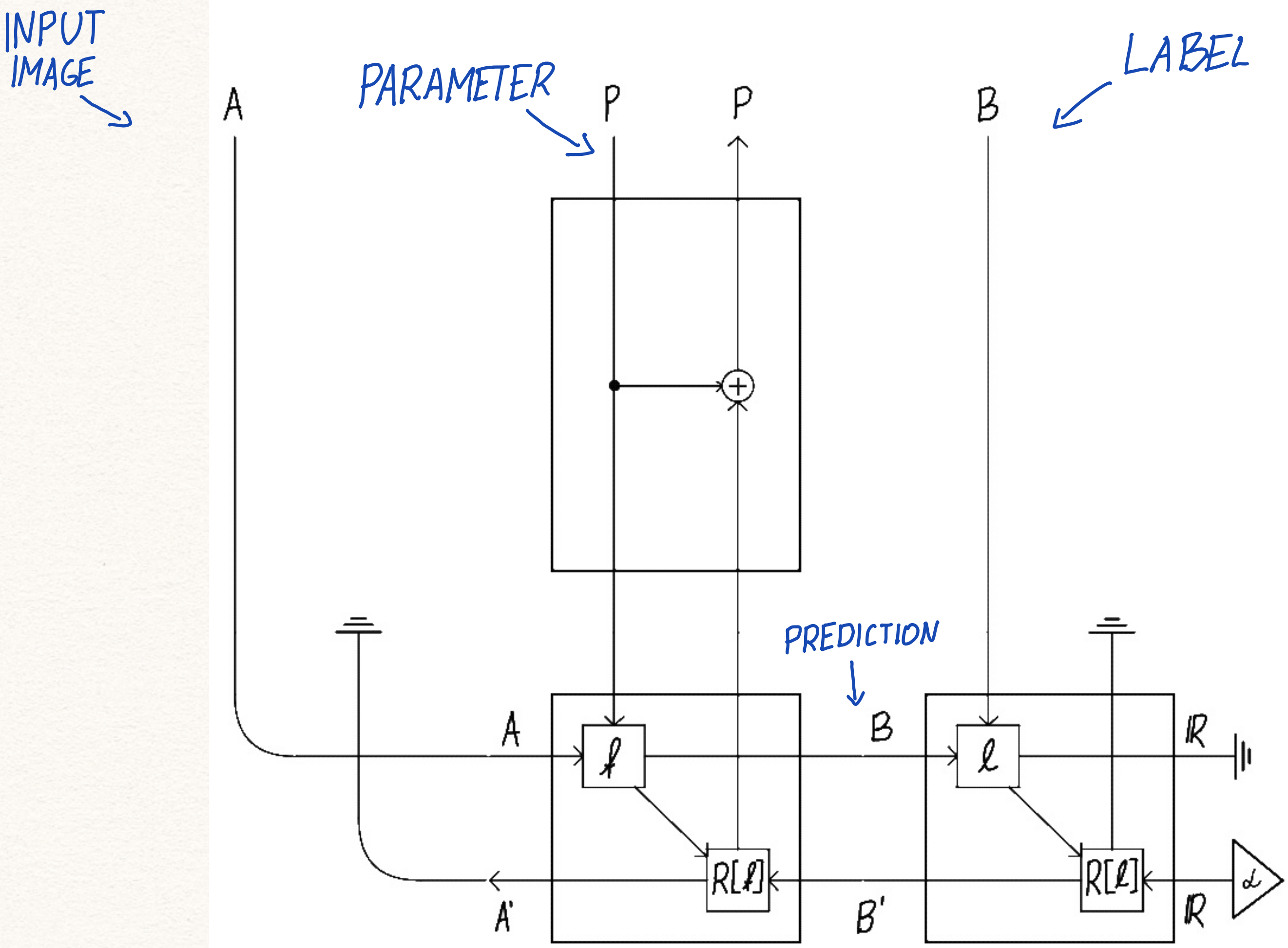
EXAMPLE. A NEURAL NETWORK + A LOSS FUNCTION



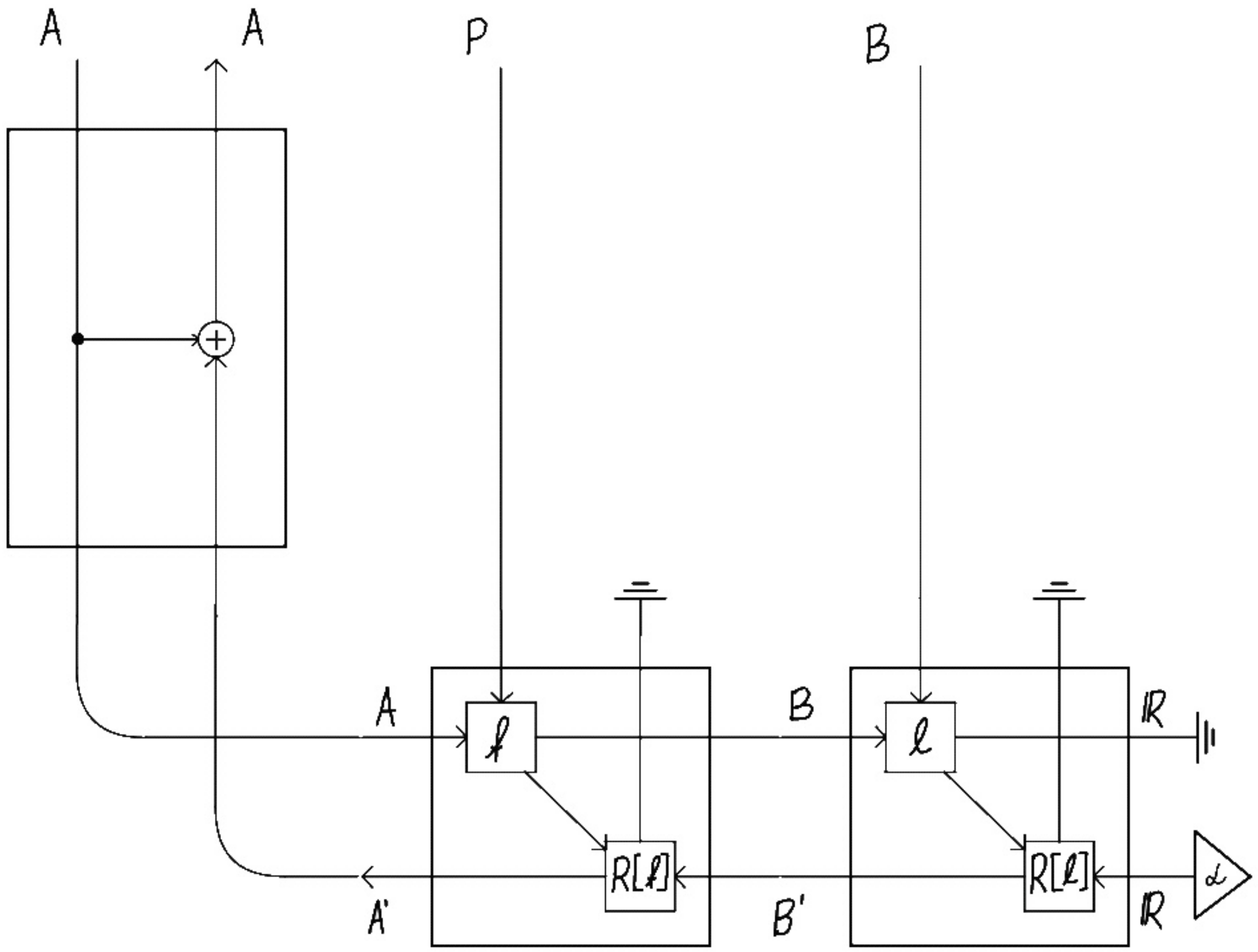


WE CAN PUT THE PIECES TOGETHER.

SUPERVISED LEARNING



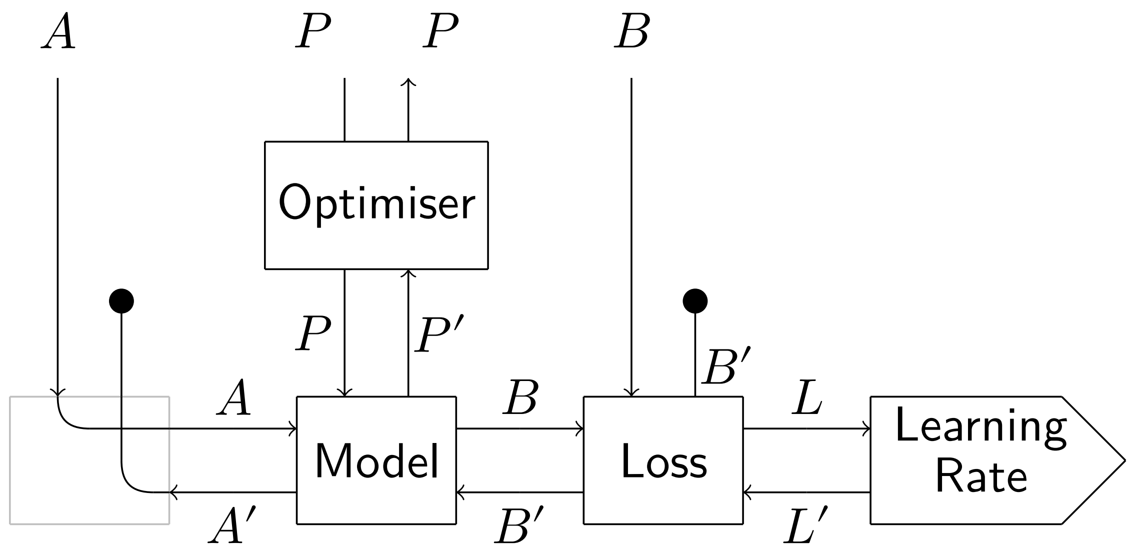
DEEP DREAMING



Categorical Foundations of Gradient-Based Learning

How to Build a Neural Network out of Lenses

Using Para and Lens we get a high level picture



Now we'll see some examples of what can be plugged into each of these boxes.

The Setting

- Each box in the diagram is a pair of maps
- Guiding example: Simple hidden layer neural network, basic gradient descent, MSE loss.
- We'll specify each pair of maps for each box
- Goal: you (roughly) understand how to translate this into code
- Implementation: github.com/statusfailed/numeric-optics-python/
  - examples include a convolutional image classifier for MNIST[1]

[1] Lecun et al., "Gradient-Based Learning Applied to Document Recognition."

Supervised Learning

In supervised learning, we want to learn a map

$$f : A \to B$$

from a dataset of examples

$$(a, b) \in A \times B$$

Now, based on our beliefs about the structure of A and B, we design a *parametrised* map:

$$\text{model} : P \times A \to B$$

and we search for some $\theta \in P$ such that model$(\theta, -)$ best represents the data.

Gradient-Based Learning

We want to use a datapoint $(a, b) \in A \times B$ to improve θ, so we need a map

$$??? : P \times A \times B \to P$$

The reverse derivative is almost what we want. For a map $f : A \to B$,

$$R[f] : A \times B' \to A'$$

(while in an RDC $A' = A$ and $B' = B$, it's useful think of the "primed" objects as representing **changes**)

So the reverse derivative of our model morphism has the following type:

$$R[\text{model}] : P \times A \times B' \to P' \times A'$$

Updates, "Displacement" and Reverse Derivatives

This is not quite enough: we have two problems:

1. We have a "true" value $b \in B$ and a "predicted" value model$(\theta, a) \in B$ but we need a B'
2. The reverse derivative gives us a P' and we want a P

This is exactly what the update and loss lenses are for:

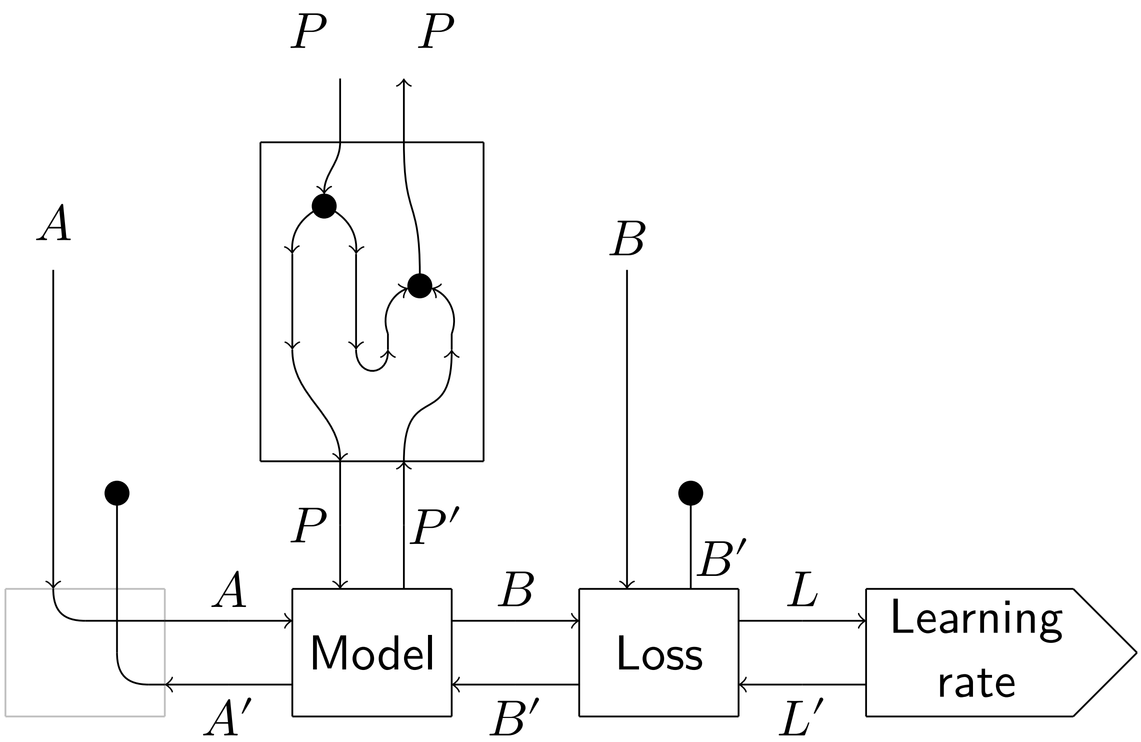
$$R[\text{model}] : P \times A \times B' \to P' \times A'$$

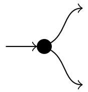
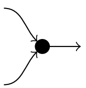
$$\text{loss}_{\text{put}} : B \times B \to B' \times B'$$

$$\text{update}_{\text{put}} : P \times P' \to P$$

Updates

Updates are like "generalised addition": add a vector to a point. The most obvious choice is just to add! That's basic gradient descent:



where  is copying and  is addition

Updates 2

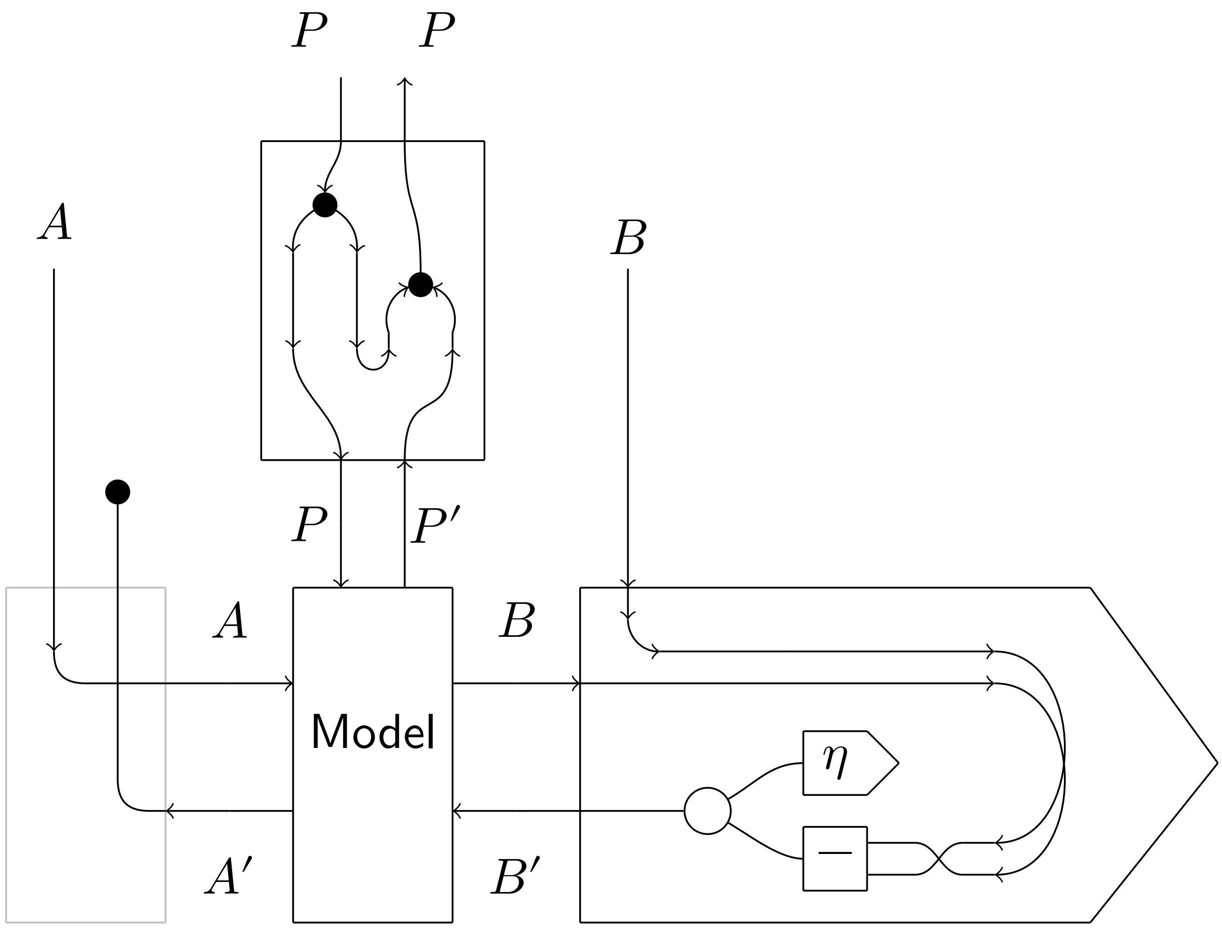
So basic gradient descent is comprised of this pair of maps:

$$\begin{aligned} \text{get} : P &\to P \\ \theta &\mapsto \theta \end{aligned}$$

$$\begin{aligned} \text{put} : P \times P' &\to P \\ \theta \qquad \theta' &\mapsto \theta + \theta' \end{aligned}$$

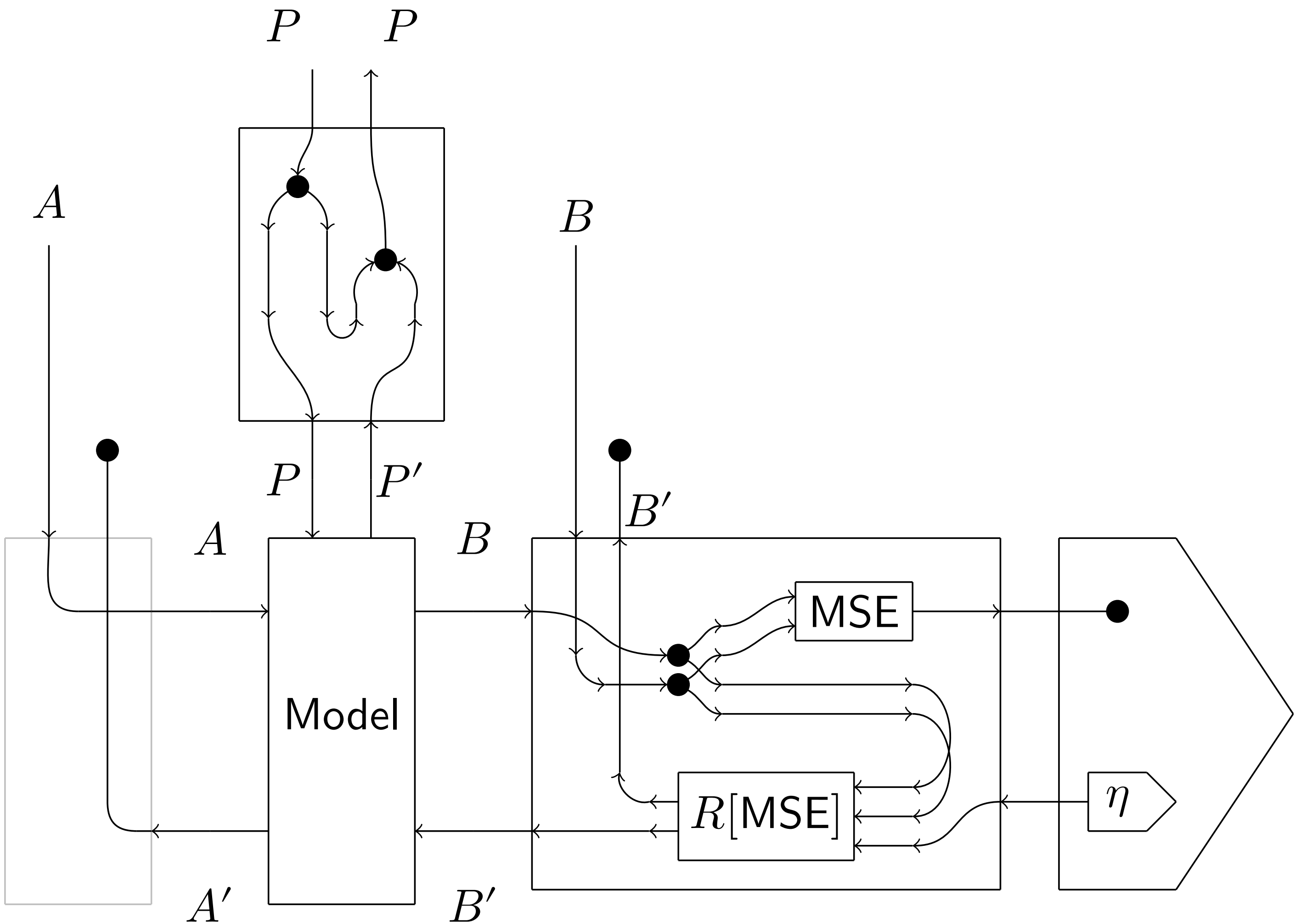
Loss + Learning Rate

Simple choice is just to subtract:



Loss + Learning Rate

This is just MSE Loss + fixed learning rate!



Loss + Learning Rate

We can think of MSE loss as the parametrised lens with maps

$$\begin{aligned} \text{get} : B \times B &\to \mathbb{R} \\ y \qquad \hat{y} &\mapsto \frac{1}{2n} \sum_{i}^{n} (y_i - \hat{y})^2 \end{aligned}$$

$$\begin{aligned} \text{put} : B \times B \times \mathbb{R} &\to P \\ y \quad \hat{y} \quad l' &\mapsto l'(\hat{y} - y) \end{aligned}$$

And the fixed learning rate as

$$\begin{aligned} \text{get} : \mathbb{R} &\to I \\ l &\mapsto \langle\rangle \end{aligned}$$

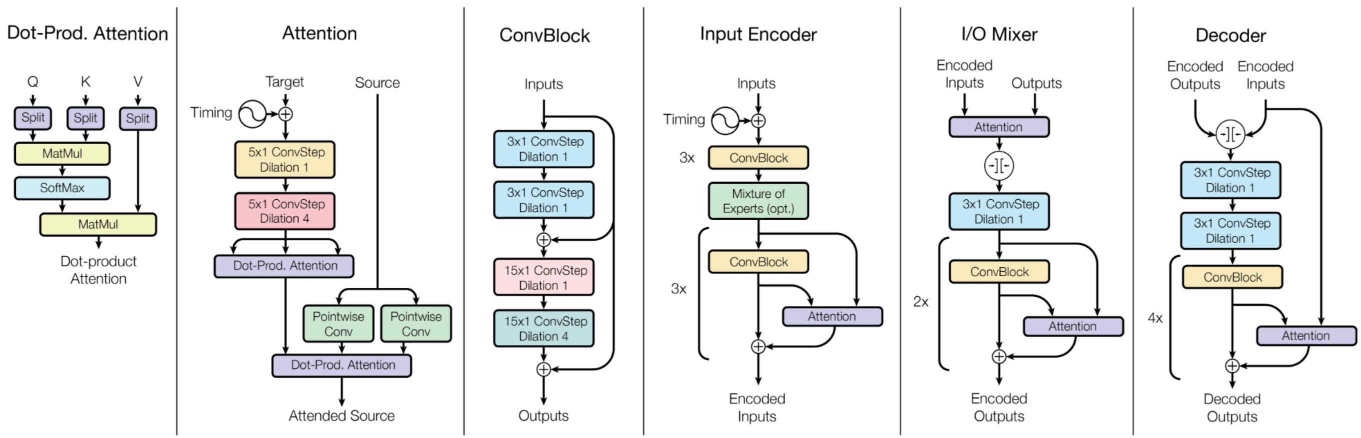
$$\begin{aligned} \text{put} : \mathbb{R} \times I &\to \mathbb{R} \\ l &\mapsto \eta \end{aligned}$$

Models, Architectures, and Layers

Two levels of detail in the model: “architecture” and “layers”.

- Architecture: the whole program as a collection of subroutines (a composition of parametrised lenses)
- Layer[2]: an individual subroutine (a parametrised lens / pair of maps)

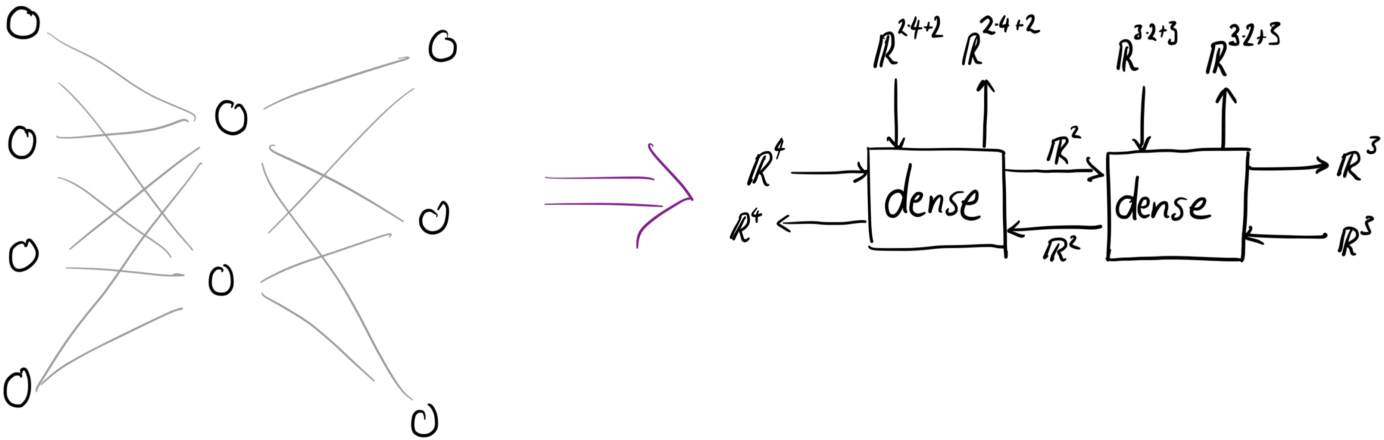
Example of a complicated architecture[3]:



[2] **ambiguous terminology warning**

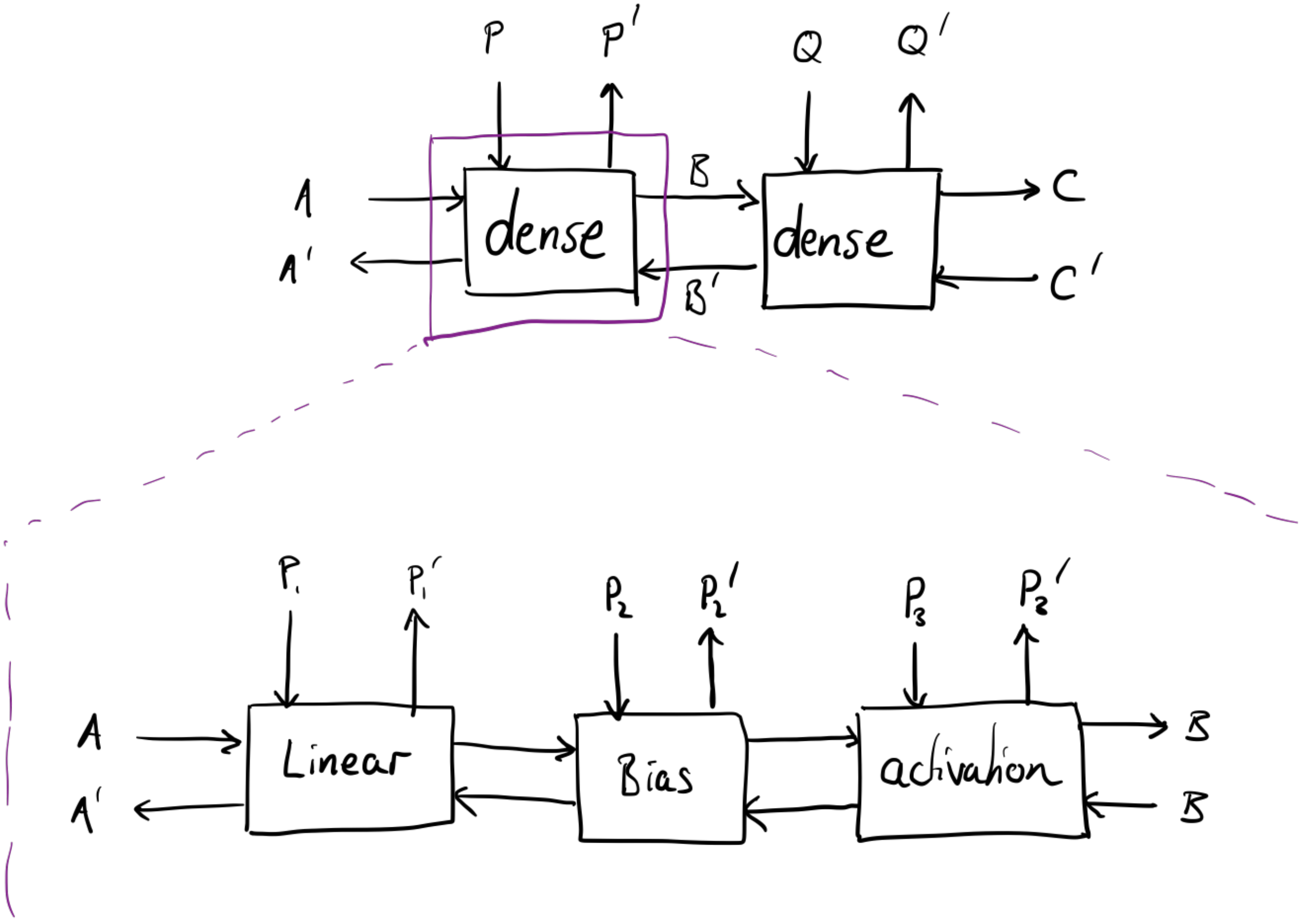
[3] Kaiser et al., “One Model to Learn Them All.”

The Old Ways

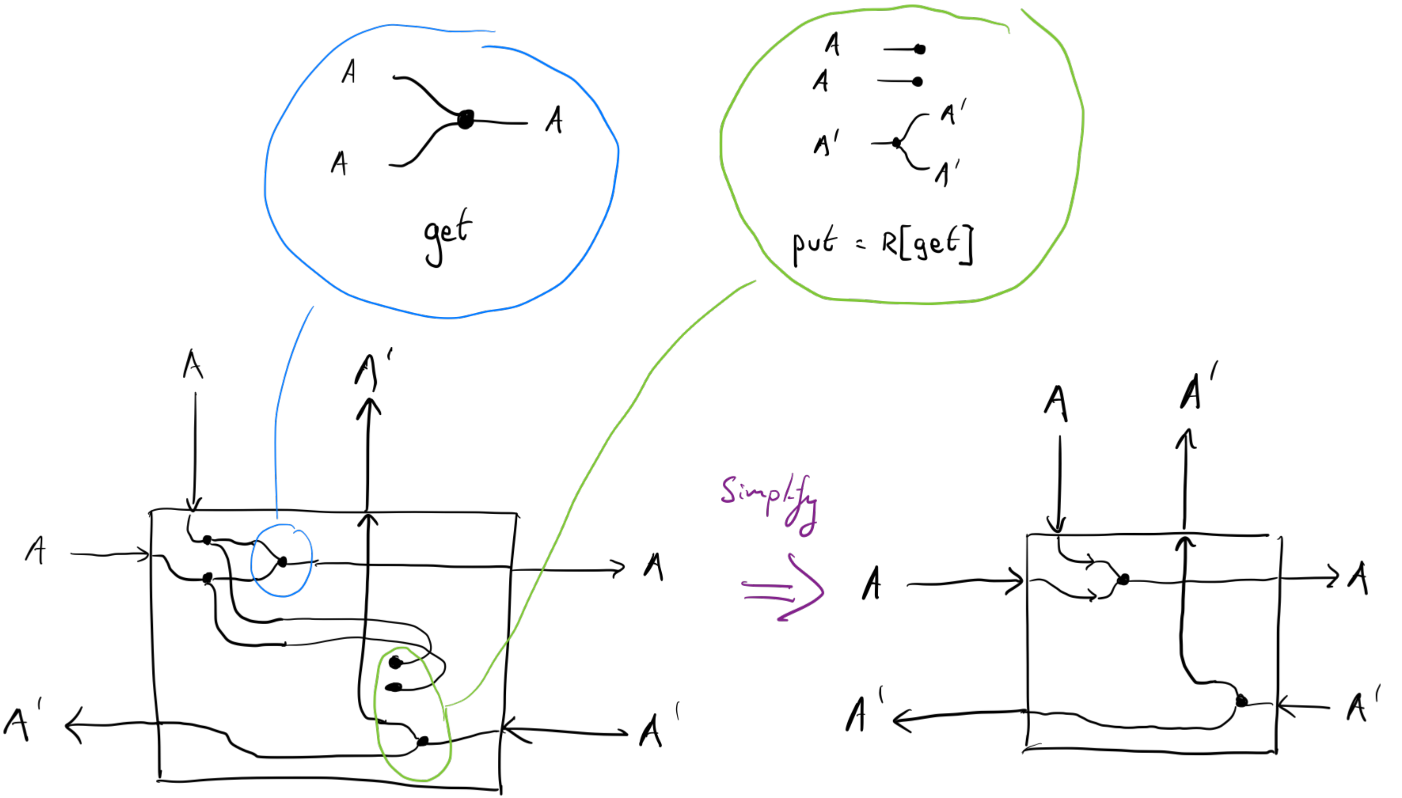


Dense Layers

A simple hidden layer neural network is a composition of two dense layers. Let's unpack a dense layer and see what's inside...



Bias Layers



Linear Layers

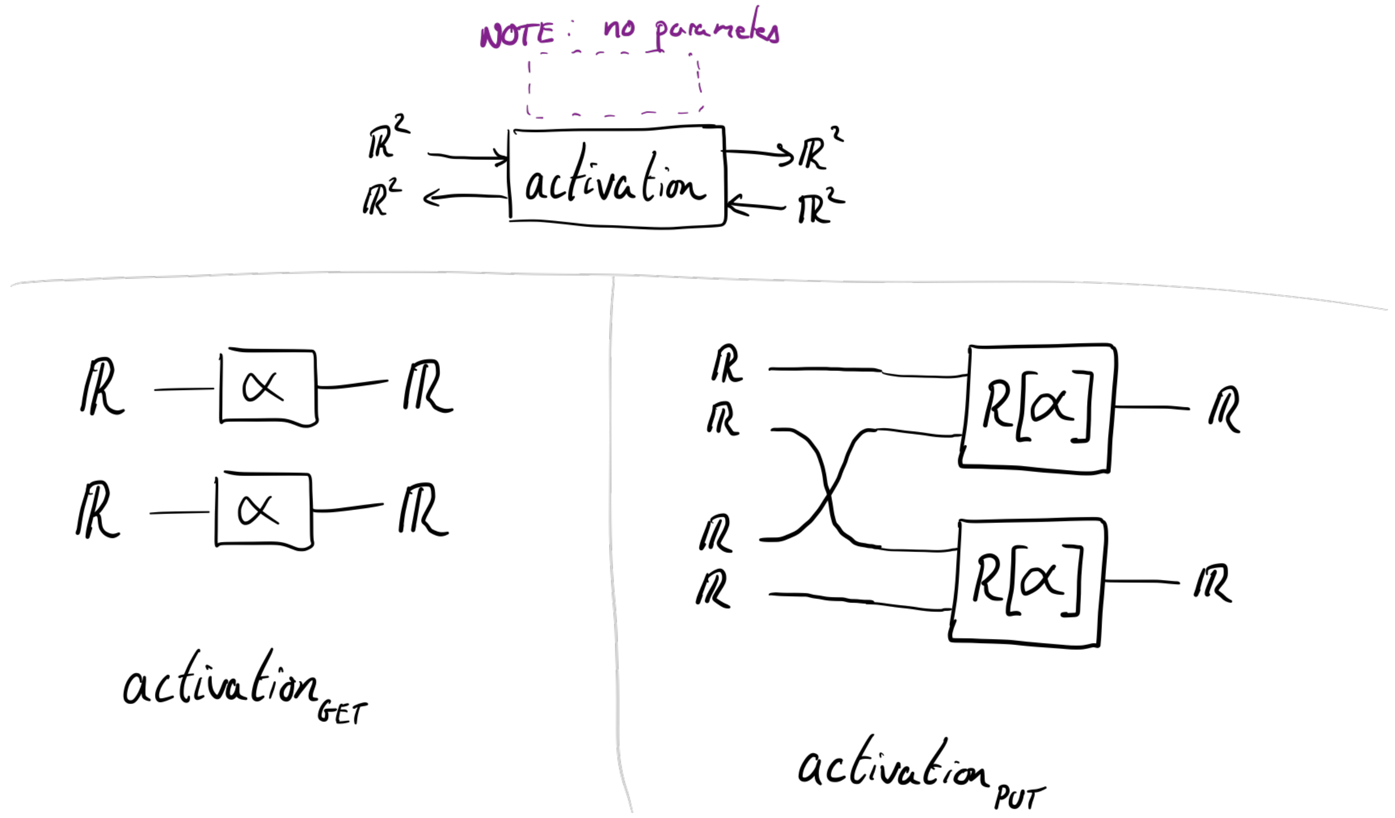
- Parameters $P = \mathbb{R}^{b \cdot a}$ are the coefficients of a matrix
- Input $A = \mathbb{R}^{a}$ is an a-dimensional vector
- Forward pass multiplies the matrix by the vector:

$$\begin{aligned} \mathsf{get} : \mathsf{Mat}(A, B) \times \mathsf{Vec}(A) &\to \mathsf{Vec}(B) \\ \mathsf{get}(M, x) &\mapsto Mx \end{aligned}$$

- Reverse pass does this (note that it typechecks!):

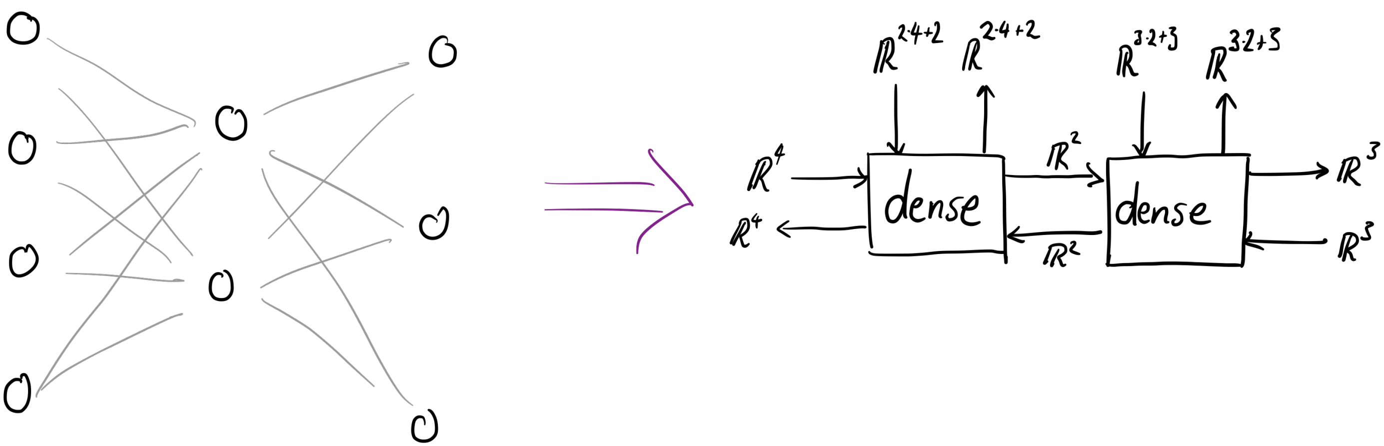
$$\begin{aligned} \mathsf{put} : \mathsf{Mat}(A, B) \times \mathsf{Vec}(A) \times \mathsf{Vec}(B) &\to \mathsf{Mat}(A, B) \times \mathsf{Vec}(A) \\ \mathsf{put}(M, x, y) &\mapsto \langle y \otimes x, M^T y \rangle \end{aligned}$$

Activation Layer

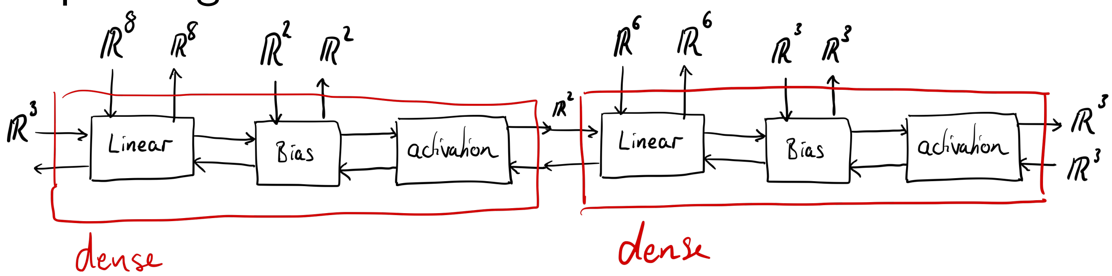


Hidden Layer Neural Network

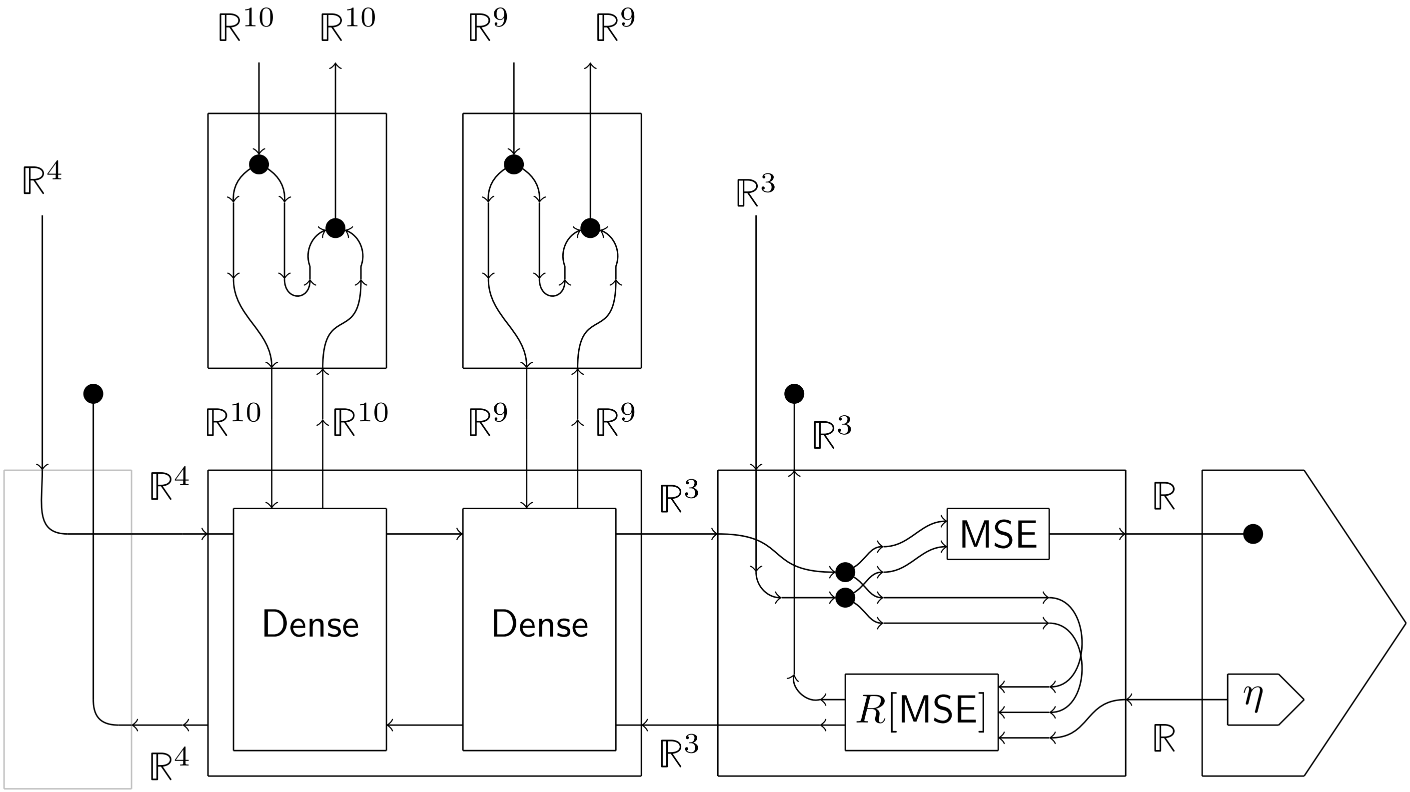
Returning to the "standard" picture of a neural network:



Expanding out "dense":



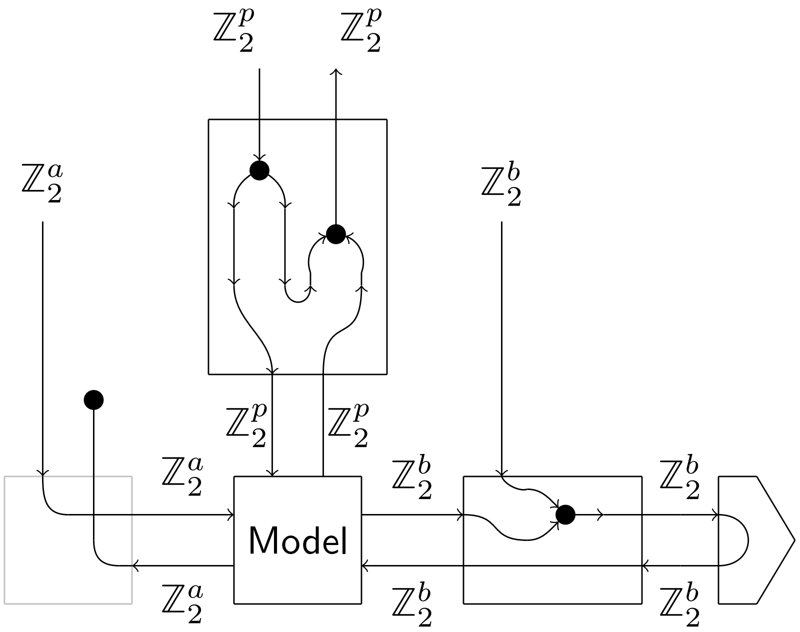
A Hidden Layer Neural Network as a Parametrised Lens



... with MSE loss, basic gradient descent, and fixed error rate

What else can we plug in?

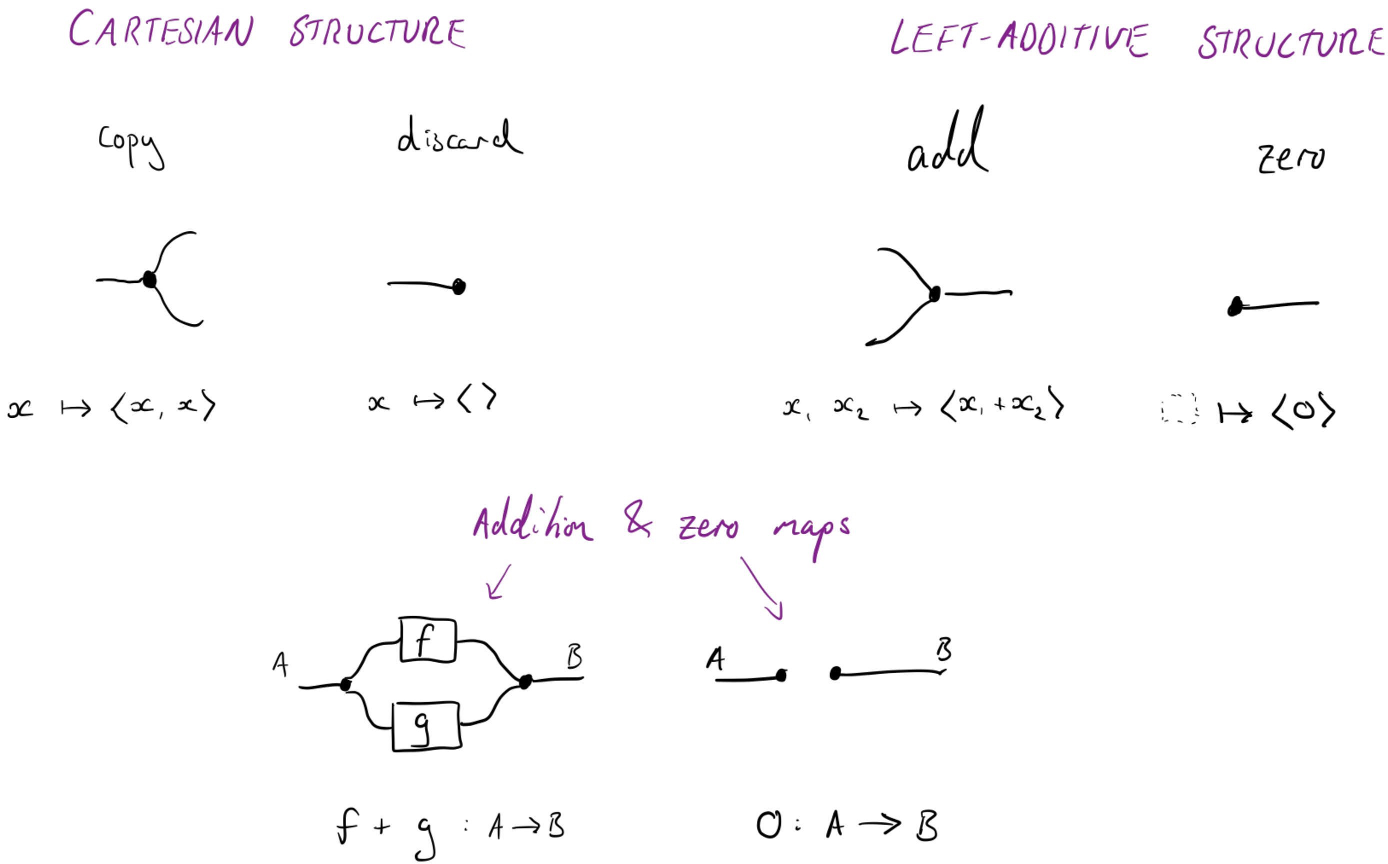
- So far we've only seen neural networks, where objects are $\mathbb{R}^n$ for $n \in \mathbb{N}$.
- We can do learning with boolean circuits too, as in Reverse Derivative Ascent[4]:



[4] Wilson and Zanasi, "Reverse Derivative Ascent."

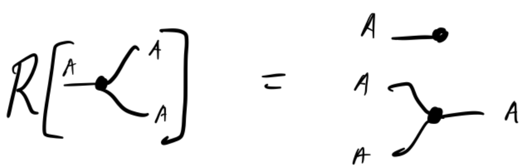
Questions?

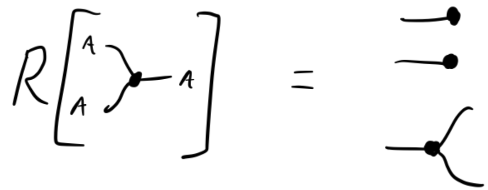
Reverse Derivatives, Graphically

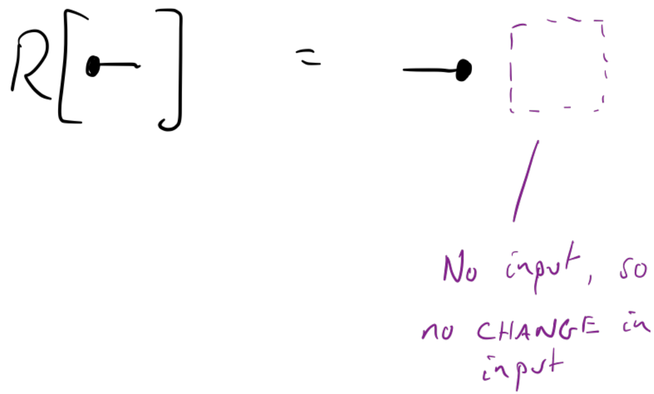


Reverse Derivatives, Graphically

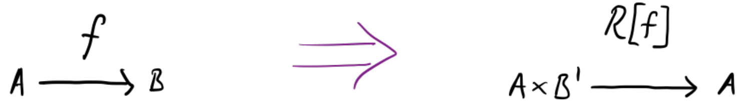
$$A \xrightarrow{f} B \quad \Longrightarrow \quad A \times B' \xrightarrow{R[f]} A'$$

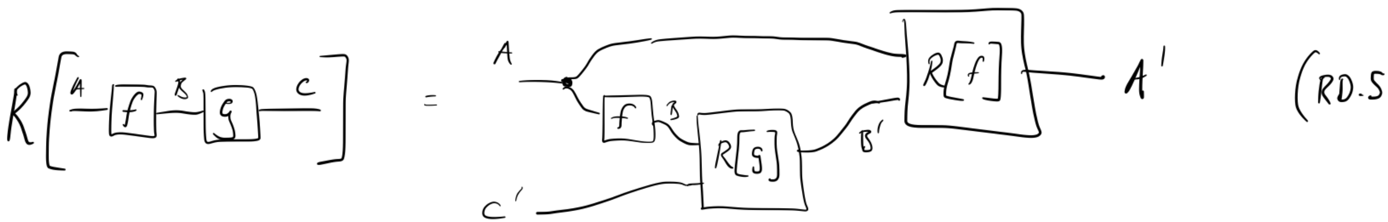


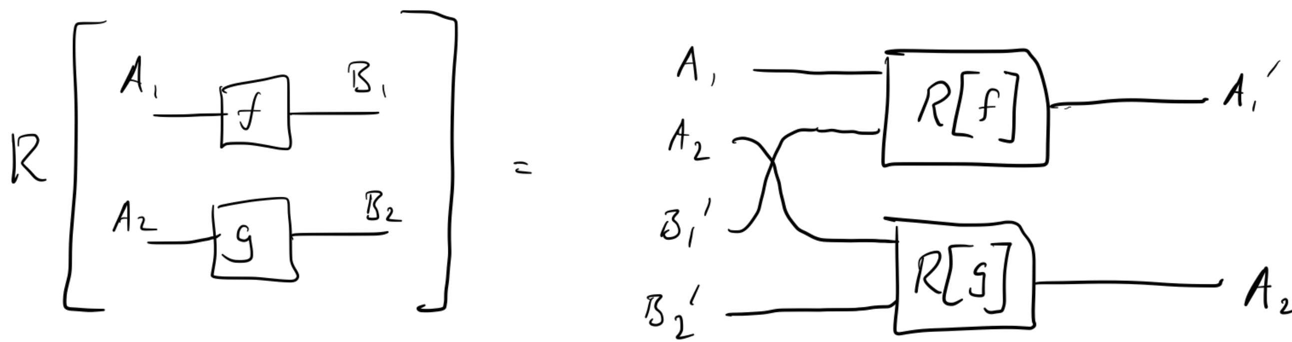




Reverse Derivatives, Graphically







References

Kaiser, Lukasz, Aidan N. Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. "One Model to Learn Them All," 2017. http://arxiv.org/abs/1706.05137.

Lecun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-Based Learning Applied to Document Recognition." In *Proceedings of the Ieee*, 2278–2324, 1998. https://doi.org/10.1109/5.726791.

Wilson, Paul, and Fabio Zanasi. "Reverse Derivative Ascent: A Categorical Approach to Learning Boolean Circuits." *Electronic Proceedings in Theoretical Computer Science* 333 (February 2021): 247–60. https://doi.org/10.4204/eptcs.333.17.