CATEGORICAL FOUNDATIONS OF GRADIENT-BASED LEARNING

(CRUTTWELL, GAVRANOVIC, GHANI, WILSON, ZANASI)

GOAL: PROVIDE A CATEGORICAL FRAMEWORK FOR DEEP LEARNING
SUPERVISED LEARNING WITH NEURAL NETWORKS IN ONE SLIDE:

\[ X \rightarrow \square \rightarrow y \]

**DATASET:** List \( X \times Y \)

**INPUT** \( X \)

**NEURAL NETWORK WEIGHTS**

**PREDICTION** \( Y \)

**LABEL**
- 1 CAT
- 0 DOG
- 0 HORSE

**LOSS**
- 0.4 CAT
- 0.5 DOG
- 0.1 HORSE
LOSS

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
  - BIDIRECTIONALITY
  - PARAMETERIZATION

  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 $C$ is a Cartesian left-additive category which for every map $f: A \rightarrow B$

has a REVERSE DIFFERENTIAL COMBINATOR $R[f]: A \times B \rightarrow A$ \hspace{1cm} (compare $D[f]: A \times A \rightarrow B$)

subject to 7 axioms.

**Example.** Smooth is a CRDC. $\text{Poly}_{Z}$ IS A CRDC.

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

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

\[(x,y), w \mapsto (2xw, 3xw)\]

Plan: study CRDC’s through optics/lenses
**OPTICS/LENSES**

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

- **Objects** — pairs of objects $(x, x')$ in $C$

$$\text{Optic}(C)(x, x') = \int_{n:e} C(x, y \otimes M) \times C(y \otimes M, x')$$

\[ (M, f, b) \quad f : X \to y \otimes M \]

\[ b : y' \otimes M \to X' \]

**PROP.** If $C$ is Cartesian,

$$\int_{n:e} C(x, M \times y) \times C(M \times y, x')$$

$$\overset{\text{UNIV. PROPERTY OF PROD.}}{=} \int_{n:e} C(x, y) \times C(x, M) \times C(M \times y, x)$$

$$\overset{\text{YONEDA REDUCTION}}{=} \int_{n:e} C(x, y) \times C(x \times y', x')$$

get \[ \quad \text{put} \]

then $\text{Optic}(C) \cong \text{Lens}(C)$

**BIDIRECTIONAL INFORMATION FLOW**
OPTICS CAN BE COMPOSED

PROPOSITION. \( \text{Optic}(C) \) is symmetric monoidal.
**Example.**

**Gradient Descent**

\[ p_x p' \xrightarrow{\mu} p \]
\[ (p, \nabla p) \xrightarrow{} p - \alpha \nabla p \]

\[
\text{is a lens, for } E := \text{Smooth} \\
\left( p \right) \xrightarrow{\text{id}, \mu} \left( p' \right)
\]

**Example.** **Stateful Optimizers**

- **Momentum**
  
  **get:** \( p_x p \xrightarrow{} p \)
  
  \[ (v, p) \xrightarrow{} p \]
  
  **put:** \( p_x p_x p \xrightarrow{} p_x p \)
  
  \[ (v, p, \nabla p) \xrightarrow{} (v', p - v') \]
  
  where \( v' = \gamma v + \epsilon p' \)

- **Nesterov Momentum**

  **get:** \( p_x p \xrightarrow{} p \)
  
  \[ (v, p) \xrightarrow{} p - \gamma v \]

  **put:** same as above

- **Adagrad**

- **Adam**

  ...
BACK TO CRDC's:

\[ f: A \rightarrow B \]
\[ R[f]: A \times B \rightarrow A \]

\[ \sim \]

\[ \text{'get' MAP OF A LENS} \]
\[ \sim \]

\[ \text{'put' MAP OF A LENS} \]

**PROPOSITION.**

For each CRDC \( C \) there is a symmetric monoidal functor

\[ C \xrightarrow{F} \text{Lens}(C) \cong \text{Optic}(C) \]

\[ A \xrightarrow{(A, A)} \]

\[ \downarrow \]

\[ B \xrightarrow{(B, B)} \]

\[ (f, R[f]) \]

\[ \Downarrow \]

\[ \text{• THIS IS OUR FRAMEWORK FOR BACKPROPAGATION} \]
Fix a SMC \((\mathcal{C}, \otimes, I)\).

**DEF.** Bicategory \(\text{Para}(\mathcal{C})\)

- **Objects** - objects of \(\mathcal{C}\)
  \[
  \text{Para}(\mathcal{C})(A, B) = \int_{P,E} \mathcal{C}(P \otimes A, B)
  \]

- **2-cells** - reparameterizations: a 2-cell is a map \(Q \Rightarrow P\) such that
  \[
  Q \otimes A \Rightarrow P \otimes A
  \]
  commutes.
EXAMPLE.

$(\text{Set}, x, 1)$  
Para$(\text{Set})$

$(\text{Smooth}, x, 1)$  
Para$(\text{Smooth})$

$(\text{Optic}(E), \otimes, 1)$  
Para$(\text{Optic}(E))$

SETS AND PARAMETERIZED FUNCTIONS

EUCLIDEAN SPACES AND PARAMETERIZED SMOOTH FUNCTIONS

PAIRS OF OBJECTS AND PARAMETERIZED OPTICS
GRAPHICAL LANGUAGE

TEXTUAL NOTATION  STANDARD STRING DIAGRAM  2D STRING DIAGRAM

\( f : P \times A \rightarrow B \)

\( \Downarrow \pi \)

\( A \rightarrow B \)

\( (Q, q) \)

\( A \rightarrow B \)

HOW DOES COMPOSITION WORK?
RECAP

Para IS NATURAL WITH RESPECT TO BASE CHANGE.
DEFINITION.

Let $G : C \to D$ be a symm. monoidal functor. We define

$$\text{Para}(G) : \text{Para}(C) \to \text{Para}(D)$$

where $f'$ is the composite

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

MORE.

Para is rich in categorical structure.

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

...
PARAMETERIZED OPTICS

\[ E \rightarrow \text{Optic}(E) \rightarrow \text{Para}(\text{Optic}(E)) \]

- Objects - objects of \( \text{Optic}(E) \) - pairs \((x, x')\) in \( E \)

- Morphisms \((x, x') \xrightarrow{(p, f)} (y, y')\) where \( f: (p \circ x) \rightarrow (y, y') \)

\( (M, f, b) \)

- \( M: E \)
- \( f: P \circ x \rightarrow y \circ M \)
- \( b: y \circ M \rightarrow P \circ x' \)

- We can compose parameterized optics
We automatically get two parameter ports

\[ \begin{pmatrix} p \\ q \end{pmatrix} \]

A 2-cell \((s: x \Rightarrow y)\) is an optic

\[ \begin{pmatrix} z \\ w \end{pmatrix} \xrightarrow{\alpha} \begin{pmatrix} p \\ q \end{pmatrix} \]

**THEOREM.**

**Gradient Descent** is a 2-cell in \(\text{Para}(\text{Optic}(C))\).

(Since it is a lens)
**THEOREM.**

APPLYING \( \text{Para} \) TO THE CRDC FUNCTOR

\[
C \xrightarrow{F} \text{Optic}(C)
\]

RESULTS IN A FUNCTOR

\[
\text{Para}(C) \xrightarrow{\text{Para}(F)} \text{Para}(\text{Optic}(C))
\]

**EXAMPLE.** A NEURAL NETWORK + A LOSS FUNCTION
WE CAN PUT THE PIECES TOGETHER.

SUPERVISED LEARNING
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

$$f : A \rightarrow B$$

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:

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

and we search for some $\theta \in P$ such that $\text{model}(\theta, -)$ best represents the data.
Gradient-Based Learning

We want to use a datapoint \((a, b) \in A \times B\) to improve \(\theta\), 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 $\text{model}(\theta, a) \in B$ 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:

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

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

$$\text{update}_{\text{put}} : P \times P' \rightarrow P$$
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
So basic gradient descent is comprised of this pair of maps:

get : $P \rightarrow P$

$\theta \mapsto \theta$

put : $P \times P' \rightarrow P$

$\theta \quad \theta' \mapsto \theta + \theta'$
Loss + Learning Rate

Simple choice is just to subtract:

\[
A \xrightarrow{P} \text{Model} \xrightarrow{P'} B
\]
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

$\text{get} : B \times B \to \mathbb{R}$

$y \quad \hat{y} \mapsto \frac{1}{2n} \sum_{i}^{n} (y_{i} - \hat{y})^{2}$

$\text{put} : B \times B \times \mathbb{R} \to P$

$y \quad \hat{y} \quad l' \mapsto l' (\hat{y} - y)$

And the fixed learning rate as

$\text{get} : \mathbb{R} \to I$

$l \mapsto \langle \rangle$

$\text{put} : \mathbb{R} \times I \to \mathbb{R}$

$l \mapsto \eta$
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**: an individual subroutine (a parametrised lens / pair of maps)

Example of a complicated architecture:

---

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...

\[
\begin{align*}
  & A \quad A' \\
  & \quad \downarrow \quad \uparrow \\
  & \quad \text{dense} \\
  & B \quad B' \\
  & \quad \downarrow \quad \uparrow \\
  & C \quad C' \\
\end{align*}
\]

\[
\begin{align*}
  & A \quad A' \\
  & \quad \downarrow \quad \uparrow \\
  & \quad \text{Linear} \\
  & \quad \downarrow \quad \uparrow \\
  & \quad \text{Bias} \\
  & \quad \downarrow \quad \uparrow \\
  & \quad \text{activation} \\
  & B \quad B' \\
\end{align*}
\]
Bias Layers

\[ \text{get} \]

\[ \text{put} = R[\text{get}] \]

Simplify
Linear Layers

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

$$
\text{get} : \text{Mat}(A, B) \times \text{Vec}(A) \to \text{Vec}(B) \\
\text{get}(M, x) \mapsto Mx
$$

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

$$
\text{put} : \text{Mat}(A, B) \times \text{Vec}(A) \times \text{Vec}(B) \to \text{Mat}(A, B) \times \text{Vec}(A) \\
\text{put}(M, x, y) \mapsto \langle y \otimes x, M^T y \rangle
$$
Activation Layer

activation\_GET

NOTE: no parameters

activation\_PUT
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$^4$:

\[\begin{array}{c}
\mathbb{Z}_2^a \\
\mathbb{Z}_2^b \\
\mathbb{Z}_2^p \\
\mathbb{Z}_2^p \\
\mathbb{Z}_2^a \\
\mathbb{Z}_2^p \\
\mathbb{Z}_2^p \\
\mathbb{Z}_2^b \\
\end{array}\]

\[\begin{array}{c}
\text{Model} \\
\end{array}\]

\[\begin{array}{c}
\mathbb{Z}_2^b \\
\mathbb{Z}_2^b \\
\mathbb{Z}_2^b \\
\end{array}\]

$^4$Wilson and Zanasi, “Reverse Derivative Ascent.”
Questions?
Reverse Derivatives, Graphically

**Cartesian Structure**

- **Copy**: \( x \mapsto (x, x) \)
- **Discard**: \( x \mapsto \langle \rangle \)

**Left-Additive Structure**

- **Add**: \( x, x_2 \mapsto \langle x_1 + x_2 \rangle \)
- **Zero**: \( \langle 0 \rangle \)

**Addition & Zero maps**

- **Function**: \( f + g : A \rightarrow B \)
- **Zero**: \( 0 : A \rightarrow B \)
Reverse Derivatives, Graphically

\[
\begin{align*}
R[\begin{array}{c}
A \xleftarrow{A} A
\end{array}] &= A \\
R[\begin{array}{c}
A \xrightarrow{A} A
\end{array}] &= A \\
R[\begin{array}{c}
A \xrightarrow{A} A
\end{array}] &= A \\
R[\begin{array}{c}
\bullet
\end{array}] &= A
\end{align*}
\]
Reverse Derivatives, Graphically

\[ f : A \to B \quad \Rightarrow \quad R[f] : A \times B' \to A' \]

\[ R \left[ \begin{array}{c} f \\ \sigma \\ \xi \\ c \end{array} \right] = \begin{array}{c}
A \\
R[f] \\
A' \\
\end{array} \]

\[ R \left[ \begin{array}{c} A_1 \\ A_2 \\ B_1 \\ B_2 \end{array} \right] = \begin{array}{c}
A_1 \\
\sigma \\
R[f] \\
A_1' \\
A_2 \\
\xi \\
\sigma \\
R[s] \\
A_2' \\
B_1' \\
B_2' \end{array} \]
References

