A Functional Reboot for Deep Learning

Conal Elliott

Target

August 2019

Goal

- Extract the essence of DL.
- Shed accidental complexity and artificial limitations,
  i.e., simplify *and* generalize.

Essence

- Optimization: best element of a set (by objective function).
  Usually via differentiation and gradient following.
- For machine learning, sets of *functions*.
- Objective function is defined via set of input/output pairs.

Accidental complexity in deep learning

Accidental complexity in DL (overview)

- Imperative programming
- Weak typing
- Graphs (neural *networks*)
- Layers
- Tensors/arrays
- Back propagation
- Linearity bias
- Hyper-parameters
- Manual differentiation

Imperative programming

- Thwarts correctness/dependability (usually “not even wrong”).
- Thwarts efficiency (parallelism).
- Unnecessary for expressiveness.
- Poor fit. DL is math, so express in a math language.

Weak typing

- Requires people to manage detail & consistency.
- Run-time errors.

Graphs (neural *networks*)

- Clutters API, distracting from purpose.
- Purpose: a representation of functions.
- We already have a better one: programming language.
- Can we differentiate?
  - An issue of *implementation*, not language or library definition.
  - Fix accordingly.

Layers

- Strong bias toward sequential composition.
- Neglects equally important forms: parallel & conditional.
- Awkward patches: "skip connections", ResNet, HighwayNet.
- Don't patch the problem; eliminate it.
- Replace with binary sequential, parallel, conditional composition.

“Tensors”

- Really, multi-dimensional arrays.
- Awkward: imagine you could program only with arrays (Fortran).
- Unsafe without dependent types.
- Multiple intents / weakly typed
- Even as linear maps: meaning of $m \times n$ array?
- Limited: missing almost all differentiable types.
- Missing more natural & compositional data types, e.g., trees.

Back propagation

- Specialization and rediscovery of reverse-mode auto-diff.
- Described in terms of graphs.
- Highly complex due to graph formulation.
- Stateful:
  - Hinders parallelism/efficiency.
  - High memory use, limiting problem size.

Linearity bias

- "Dense" & "fully connected" mean arbitrary *linear* transformation.
- Sprinkle in "activation functions" as exceptions to linearity.
- Misses simpler and more efficient architectures.

Hyper-parameters

- Same essential purpose as parameters.
- Different mechanisms for expression and search.
- Inefficient and ad hoc

A functional reboot

Values

- *Precision*: meaning, reasoning, correctness.
- *Simplicity*: practical rigor/dependability.
- *Generality*: room to grow; design guidance.

Essence

- Optimization: best element of a set (by objective function). Usually via differentiation and gradient following.
- For machine learning, sets of *functions*.
- Objective function is defined via set of input/output pairs.

Optimization

- Describe a set of values as range of function: $f :: p \to c$.
- Objective function: $q :: c \to \mathbb{R}$.
- Find $argMin\ (q \circ f) :: p$.
- When $q \circ f$ is differentiable, gradient descent can help.
- Otherwise, other methods.
- Consider also global optimization, e.g., with interval methods.

Learning *functions*

- Special case of optimization, where $c = a \to b$, i.e., $f :: p \to (a \to b)$, and $q :: (a \to b) \to \mathbb{R}$.
- Objective function often based on sample set $S \subseteq a \times b$. Measure mis-predictions (loss).
- Additivity enables parallel, log-time learning step.

Differentiable functional programming

- Directly on Haskell (etc) *programs*:
  - Not a library/DSEL
  - No graphs/networks/layers
- Differentiated at compile time
- Simple, principled, and general
  (*The simple essence of automatic differentiation*)
- Generating efficient run-time code
- Amenable to massively parallel execution (GPU, etc)

Beyond "tensors"

- Most differentiable types are *not* vectors (uniform n-tuples), and most derivatives (linear maps) are not matrices.
- A more general alternative:
  - *Free vector space* over s: $i \to s \cong f\ s$ ("i indexes f")
  - Special case: $Fin_n \to s \cong Vec_n\ s$
  - *Algebra of representable functors*: $f \times g$, 1, $g \circ f$, Id
  - Your (representable) functor via **deriving** *Generic*
- Linear map $(f\ s \multimap g\ s) \cong g\ (f\ s) \cong (g \circ f)\ s$ (generalized matrix). Other representations for efficient reverse-mode AD (w/o tears).
- Use with *Functor*, *Foldable*, *Traversable*, *Scannable*, etc. No need for special/limited array "reshaping" operations.
- Compositional and naturally parallel-friendly (*Generic parallel functional programming*)

Modularity

- How to build function families from pieces, as in DL?
- Category of indexed sets of functions.
- Extract monolithic function after composing.
- Other uses, including satisfiability.
- Prototyped, but problem with GHC type-checker.

Progress

- Simple & efficient reverse-mode AD.
- Some simple regressions, simple DL, and CNN.
- Some implementation challenges with robustness.
- Looking for collaborators, including
  - GHC internals (compiling-to-categories plugin)
  - Background in machine learning and statistics

Summary

- Generalize & simplify DL (more for less).
- Essence of DL: pure FP with *minarg*.
- Generalize from "tensors" (for composition & safety).
- Collaboration welcome!