Backprop as Functor: A compositional perspective on supervised learning

David I. Spivak (joint with Brendan Fong and Rémy Tuyéras)

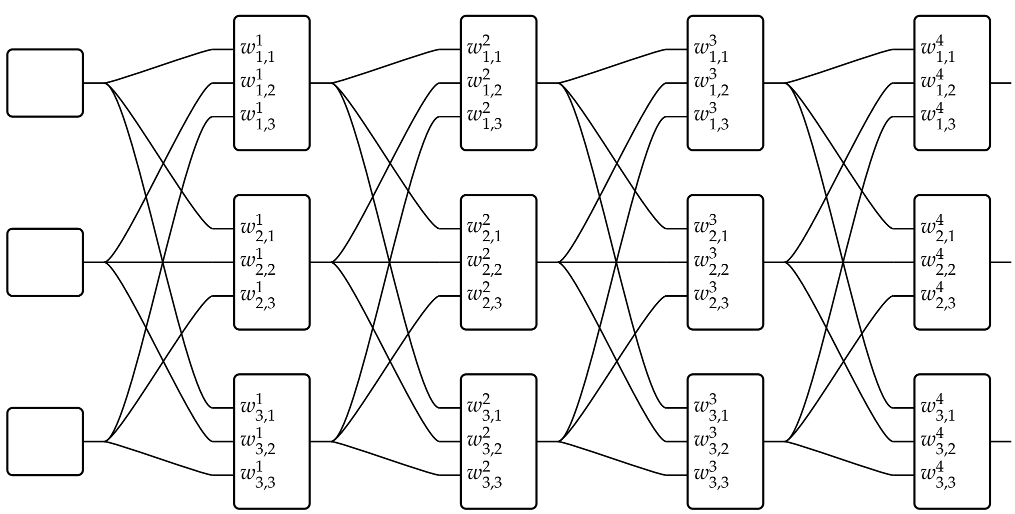
2018/02/01 at Oxford OASIS

Abstract

Abstract: Neural networks can be trained to perform functions, such as classifying images. The usual description of this process involves keywords like neural architecture, activation function, cost function, back propagation, training data, weights and biases, and weight-tying.

In this talk we will describe a symmetric monoidal category Learn, in which objects are sets and morphisms are roughly "functions that adapt to training data". The back propagation algorithm can then be viewed as a strong monoidal functor from a category of parameterized functions between Euclidean spaces to our category Learn.

This presentation is algebraic, not algorithmic; in particular it does not give immediate insight into improving the speed or accuracy of neural networks. The point of the talk is simply to articulate the various structures that one observes in this subject—including all the keywords mentioned above—and thereby obtain a categorical foothold for further study.



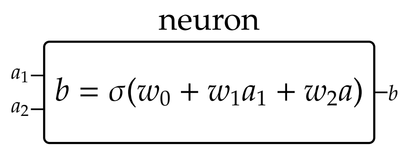
Backprop as Functor: A compositional perspective on supervised learning

I. Introduction

   A. What is supervised learning?

      1. The usual story:

         a. neural architecture, weights+biases, activation fn., cost fn.



         b. training data

         c. gradient descent, backpropagation

      2. Extra bells and whistles: Convolutional neural nets + weight tying

         a. Convolutional neural nets: image classification

         b. Idea: *Shift-invariance*: look for same patterns in different places

   B. Our goal: articulate the structures categorically

      1. A monoidal category setting Learn for general supervised learning

      2. Parameterized functions as special case: a functor $\mathsf{Para} \to \mathsf{Learn}$

      3. Neural nets as special case: a functor $\mathsf{NNet} \to \mathsf{Para}$

II. Compositional supervised learning

   A. Basic structure isn't compositional

      1. Basic structure of morphism $A \to B$:

         a. Parameter space P,

         b. Parameterized function $I\colon P \times A \to B$, and

         c. Update function: $U\colon P \times A \times B \to P$.

      2. Problem with composition: need local training data

   B. The fix: request function

      1. Add to the list: $r\colon P \times A \times B \to A$.

      2. Mainly used for compositionality: generalized backprop

      3. Also for *dreaming*: "make input more cat-like"

   C. The symmetric monoidal category Learn

      1. $\mathrm{Ob}(\mathsf{Learn}) = \mathrm{Ob}(\mathbf{Set})$

      2. $\mathsf{Learn}(A, B) = (P, I, U, r)$ up to equivalence

         a. Any $f\colon P \to P'$ commuting with I, I', U, U', and r, r' is equivalence.

3. Monoidal product: $\times$ across the board; unit: $\mathbf{1}$

III. The strong monoidal functor $\mathsf{Para} \to \mathsf{Learn}$

A. The prop Para

1. $\mathrm{Ob}(\mathsf{Para}) = \mathbb{N}$; I'll denote them $\mathbb{R}^n$
2. $\mathsf{Para}(m, n) = \{(k, I) \mid k \in \mathbb{N}, I\colon \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}^n\}$
3. Neural networks are special morphisms in Para

B. The functors $L_{\varepsilon,e}\colon \mathsf{Para} \to \mathsf{Learn}$

1. Choices
   a. Choose step size $\varepsilon > 0$
   b. Choose cost function $e\colon \mathbb{R}^{\{g,b\}} \to \mathbb{R}$ satisfying technical condition:
   c. $\frac{\partial e}{\partial g}(g_0, -)\colon \mathbb{R}^{\{b\}} \xrightarrow{\cong} \mathbb{R}$ an isomorphism for each "guess" g_0
2. Obtain a strong monoidal functor $L_{\varepsilon,e}\colon \mathsf{Para} \to \mathsf{Learn}$
   a. For simplicity, assume quadratic error: $e(g, b) = \frac{1}{2}(g - b)^2$.
   b. On objects: $n \mapsto \mathbb{R}^n$.
   c. On morphisms: $(P, I) \mapsto (P, I, U, r)$ where

$$U_I(p, a, b) := p - \varepsilon \nabla_p E_I(p, a, b)$$
$$r_I(p, a, b) := a - \nabla_a E_I(p, a, b)$$

   where $E_I(p, a, b) = \frac{1}{2}\|I(p, a) - b\|^2$.

C. What does strong monoidal functoriality mean?

1. We can compose string diagrams
2. Gradient descent can be done locally, and requests are back-propagated.
3. (Cheaper).

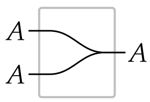
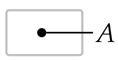
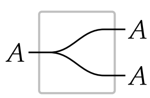
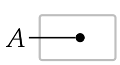
IV. Other structures

A. A bimonoid on each object $\mathbb{R}^n \in \mathsf{Learn}$

1. Let FVect denote category of f.d. vector spaces and linear maps.
2. Have $\mathsf{FVect} \hookrightarrow \mathsf{Para} \hookrightarrow \mathsf{Learn}$ (for any ϵ, e).
3. This gives a bimonoid structure on each object of the image. [see next page]

B. Weight-tying

1. Each morphism in Para can be factored.
2. $(P, I)\colon A \to B$ vs. $(P, \mathrm{id})\colon \mathbf{1} \to P$ and $(\mathbf{1}, I)\colon P \times A \to B$.
3. Weight tying [see next page]

	Implementation	Request
Multiplication, μ $(1, I_\mu, !, r_\mu)$  	$I_\mu\colon A \times A \longrightarrow A;$ $(a_1, a_2) \longmapsto a_1 + a_2$	$r_\mu\colon (A \times A) \times A \longrightarrow A \times A;$ $((a_1, a_2), a_3) \longmapsto (a_3 - a_2, a_3 - a_1)$
Unit, η $(1, I_\eta, !, r_\eta)$  	$I_\eta\colon \mathbb{R}^0 \longrightarrow A;$ $0 \longmapsto 0$	$r_\eta\colon A \longrightarrow \mathbb{R}^0;$ $a \longmapsto 0$
Comultiplication, δ $(1, I_\delta, !, r_\delta)$  	$I_\delta\colon A \longrightarrow A \times A;$ $a \longmapsto (a, a)$	$r_\delta\colon A \times (A \times A) \longrightarrow A;$ $(a_1, (a_2, a_3)) \longmapsto a_2 + a_3 - a_1$
Counit, ϵ $(1, I_\epsilon, !, r_\epsilon)$  	$I_\epsilon\colon A \longrightarrow \mathbb{R}^0;$ $a \longmapsto 0$	$r_\epsilon\colon A \longrightarrow A;$ $a \longmapsto 0$