Deep learning on C^∞ Manifolds

Jake Bian*

April 2017

Abstract

The intention of this note is to explain what it means to "train a machine learning model" to a mathematical audience. As a by-product we give a description of "training/back-prop/gradient descent" in the category of smooth manifolds. I also explain briefly how going beyond the category of vector spaces can lead to novel classes of neural networks.

Introduction

Existing machine learning literature is not very accessible to people coming from a geometry/topology background. We explain in this note the perspective that machine learning problems are fascinating problems about flows on moduli spaces of maps.

Outline Section 1 describes the program of "machine learning" in general. Section 2 gives a construction which is a geometric formalization of "back propagation". Section 3 discusses neural networks and how our construction applies. The final section describes a simple example of how this framework allows one to do deep learning without linear maps.

Acknowledgements This note is a version of what I circulated for the past year to explain what I've been working on to some of my colleagues in math/physics, I thank them for comments and discussion. In particular I thank Raouf Dridi for comments on this draft for spending time years ago to convince me that there are interesting geometry problems in machine learning.

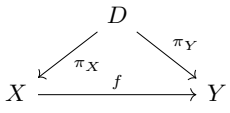
1 Machine Learning as Variation Problem

I'm going to explain the central problem in machine learning. For the remainder of this we stay in the category of smooth manifolds.

Let X, Y be smooth manifolds. We're given a finite set of points

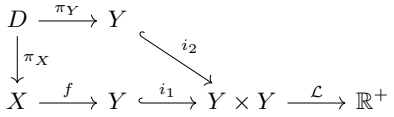
$$D \subset X \times Y, \quad |D| \equiv N \in \mathbb{N}$$

and we want to find a function $f : X \to Y$ that "approximates" D. In the sense that, **ideally**, we want this to commute



*jake@keplr.io

But sometimes dreams have to be dreams. We will settle for a modification of this diagram, that measures how well f approximates D through some non-negative function $\mathcal{L} : Y \times Y \to \mathbb{R}^+$:



A measure of how well f approximates D is obtained by summing the output of this diagram across the finite set D:

$$S[f] \equiv \sum_{(x_k, y_k)\ \in D} \mathcal{L}(f(x_k), y_k)$$

So to summarize, given inputs and D, $\mathcal{L}$, our goal is to find a f that minimizes the functional $S_{D,\mathcal{L}}$ across $C^\infty(X \to Y)$.

Phrased this way machine learning is just the usual problem in variational calculus, with an energy functional given by summing the Lagrangian over a discrete domain. This is difficult to tackle analytically because because we're interested in cases where the following numbers are huge:

$$|D|,\ dim(X) >> 0$$

So we swallow our pride and resort to dumber approaches.

2 The Framework of "Training"

Turns out the key insight is to feed into the second graph above elements of D one at a time, and obtain a sequence of functions f_i that gradually minimizes the functional S.

To make a problem a little less hopelessly difficult we need to constrain it a little more. $C^\infty(X, Y)$ is too big. Instead say we have a smooth family of functions parametrized by a smooth manifold Σ. That is, a map:

$$\alpha : \Sigma \times X \to Y \tag{1}$$

which is smooth in both arguments.

2.1 Smooth Intution: Functional Minimization from Integral Curves

Intuitively, what we're about to do next is a discrete version of the following: find a curve on Σ:

$$\gamma : [0, 1] \to \Sigma \tag{2}$$

such that the induced 1-parameter flow of functions

$$f_t \equiv \alpha(\gamma(t), \cdot) \tag{3}$$

flows to a minimum of the functional $S_{D,\mathcal{L}}$.

One way to obtain such a curve γ is the following: evaluate the derivatives of S, which induces a vector field on Σ. Integrate this vector field starting from some $\lambda_0 \in \Sigma$ in the negative direction of dS to obtain a integral curve γ.

2.2 The Training Diagram

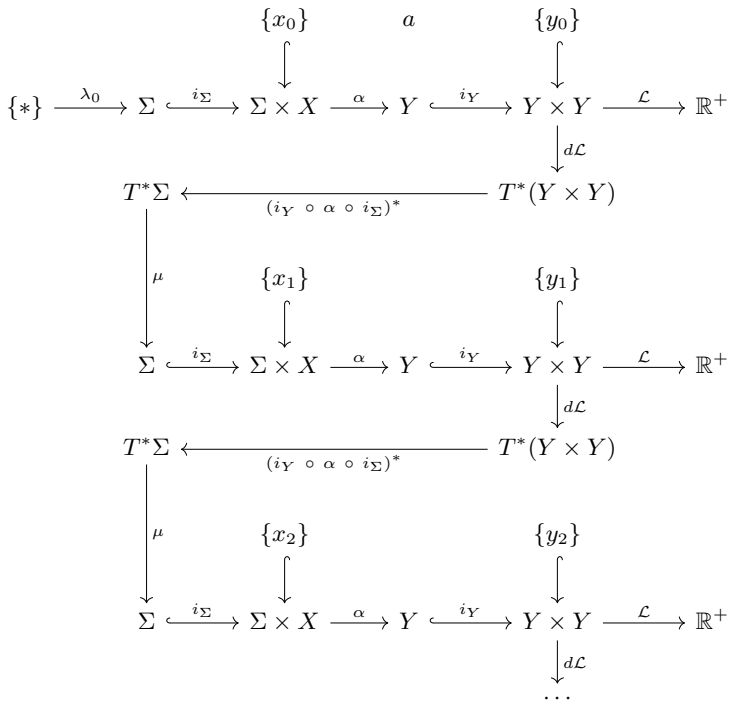
Now we just proceed to give the discrete version of the procedure described above.

- Instead of the smooth parameter t we're going to produce a finite sequence of functions f_i
- We take some function μ which produces a point in parameter space from a tangent vector

$$\mu : T^*\Sigma \to \Sigma \tag{4}$$

  Intuitively, one should think of this as the local, point-wise version of taking the integral curve of a vector field

We will give explicit examples of the function μ later. Note there are versions of this which makes use of higher derivatives, in which case we'd replace the tangent bundles with higher jet bundles. Finally everything fits together into this diagram:



where we gave D and order and wrote its points as $(x_0, y_0), (x_1, y_1), \cdots$.

A few observations

- Our discrete version of the integral curve in parameter space is given by the sequence of values obtained in this diagram at Σ
- This diagram goes on until all of D is exhausted, at which point one might go through the diagram again with λ_0 given by the value at Σ in the last row. This is the essence of training a machine learning model.
- Suppose our parameter space admits a splitting as a product

$$\Sigma \simeq \Sigma_0 \times \cdots \times \Sigma_k$$

each right-moving row can be extended on the left by an inclusion of the factors, and in the left-moving rows each new inclusion arrow here induces a pullback on the left. Something like this happens in neural networks, as we will see in a bit.

3 Neural Networks by Stacking Diagrams

3.1 Parameter Spaces

A neural network (see, e.g. [1]) is a particular ansatz for the general problem described above. The intuition is as follows.

Consider the special case that the spaces X, Y admits linear structures as vector spaces of dimensions m, n respectively. The first step to carrying out the procedure described above is to choose a parameter space Σ. But since we're in the category of vector spaces it feels natural to try the space of linear maps:

$$\Sigma = \{\text{Linear } X \xrightarrow{f} Y\} \simeq GL(m, n)$$

This is a horribly small set of functions in $C^\infty(X \to Y)$. In fact this approach is known to physics undergrads as "finding a line of best fit", not a particularly versatile technique.

Here's one way to enlarge this space. Let V be some other vector space, and consider morphisms that are compositions

$$X \xrightarrow{f} V \xrightarrow{g} Y$$

But if we leave f and g in the linear category, this doesn't enlarge our set of morphism at all. Hence we forcefully take the diagram out of the linear category by inserting a non-linear link in the middle

$$X \xrightarrow{f} V \xrightarrow{\sigma} V \xrightarrow{g} Y$$

for good choices of σ, the set of such compositions becomes

$$GL(m, k) \oplus GL(k, n)$$

.

The ansatz $g \circ \sigma \circ f$ is the first example of a multi-layer neural network. More generally, one chooses a sequence of vector spaces $(V_0, \cdots V_n)$, and non-linear automorphisms σ_i. (Fully-connnected) neural networks are anstaz' of the form

$$f_n \circ \cdots \sigma_1 \circ f_1 \circ \sigma_0 \circ f_0$$

where f_i are linear. Your parameter space becomes the direct sum of the morphism sets of all the arrows in the diagram

$$X \to V_0 \to \cdots \to V_{n-1} \to Y$$

.

3.2 Training

We now describe training a neural network in the framework laid out above. We define a parameter space at each intermediate space

$$\Sigma_i \equiv Hom_{Vect}(V_i, V_{i+1})$$

taking as definition $V_0 \equiv X$, $V_n \equiv Y$. And define the maps α_i as the evaluation maps

$$\alpha_i : \Sigma_i \times V_i \to V_{i+1}, \quad (A, v) \mapsto A(v)$$

In our training diagrams, we replace

$$\Sigma \times X \to Y$$

with

$$\Sigma_0 \times V_0 \xrightarrow{\alpha_0} \Sigma_1 \times V_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{n-2}} \Sigma_{n-1} \times V_{n-1} \xrightarrow{\alpha_{n-1}} V_n$$

We then suppose we have the functions

$$\mu_i : T^*\Sigma_i \to \Sigma_i$$

It is then straightforward to draw the training diagrams above for a neural network. $d\mathcal{L}$ now pulls back to each parameter space Σ_i, and for the next right moving row we evaluate inputs to each Σ_i with μ_i. We leave drawing an example of this for a 1-layer network as an exercise.

4 Discussion

Deep learning on the torus Let's pause and summarize the story so far. We described our framework in the category of smooth manifolds, but had to go into the category of vector spaces in order to discuss neural networks. What the category of vector spaces does for us is simply that linear maps gives a natural, and "small" class of morphisms, that we can compose with some fixed non-linear maps to gradually probe more and more of the space of smooth morphisms.

I want to now discuss a simple example of how we can repeat this story, without resorting to linear maps. Let's take

$$X = T^2, \quad Y = \mathbb{R} \tag{5}$$

That is, we're trying to approximate some function on the torus, given some dataset D. Datasets like this can arise quite naturally: one simply a problem where the domain cpntains 2 periodic variables $\in S^1$.

If one were to build a neural network for this, the natural thing to do is probably to pass to the universal cover $\mathbb{R}^2$ and do linear algebra there. Or one might even have a case where the domain variables are embedded in $\mathbb{R}^3$, then one would try to approximate a function $\mathbb{R}^3 \to \mathbb{R}$ and restricts to the correct approximation on the torus.

What the machinery developed here allows one to do the following. Instead of passing to some space where it makes sense to consider linear maps, we stay in T^2. Instead of considering linear maps, consider intermediate spaces of tori, and consider the action of the modular group $PSL(2, Z)$ on the torus. This is the small, manageable class of maps that we'll use. We can then compose these again with component-wise activation functions to generate more homeomorphisms of the torus. It is completely unclear whether this approach will produce better experimental results, but it should be clear that this is a natural analogue of a fully connected neural network on the torus.

Some open problems Here we have spelled out machine learning is a differential geometry problem. There are then natural questions on how tools from geometry can help us understand neural networks better. Here are some of my favorites:

- When points in D mostly lie on some submanifold $S \subset X$, can we use Morse theory to deduce the topology of S?
- It should be clear from the description above, the data required for specifying a neural network is more or less that of a representation of some quiver (I only discussed feed-forward networks here but neural networks with more interesting graph topologies are common, e.g. [5]). Can we use this to classify neural networks in a useful way?

- There's long been some intuition that deep learning is somehow in a precise way analogous to renormalization in physical systems [2][3]. Now in algebraic geometry we have an understanding of certain renormalization procedures (namely on D-branes) in terms of derived categories of coherent sheaves [4]. Can we use this to understand deep learning in a geometric and rigorous way?

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Henry W. Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well?, 2016.

[3] Pankaj Mehta and David J. Schwab. An exact mapping between the variational renormalization group and deep learning, 2014.

[4] E. Sharpe. Derived categories and stacks in physics, 2006.

[5] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.