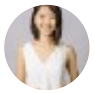


Lisha li Following
Feb 13 · 10 min read

Age of AI Talk:``Deep Learning est Mort! Vive Differentiable Programming”

Differentiable Programming Framework for Deep Learning and Machine Intelligence.

Here’s is a recording of the talk I gave at the Age of AI conference in San Francisco along with transcript.

(Rough) transcription below.

``Deep Learning est Morte! Vive Differentiable Programming”

I’ll be talking about Differentiable programming as a useful framework for deep learning and machine intelligence.

First a little about my background:

-I'm an investor with Amplify Partners.

-early venture firm that invests in technical founders solving technical problems.

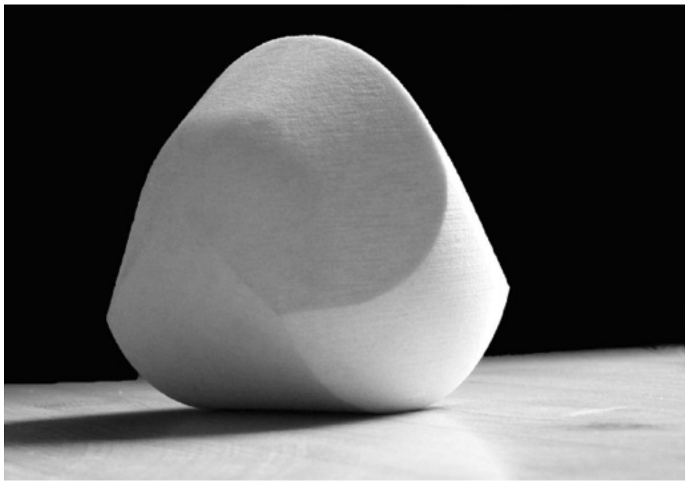
-our portfolio centers on compute infrastructure and applied AI

some examples are:

- *Primer (texts summarization from diverse modalities)*
- *DeterminedAI—which deploys deep learning models on prem.*
- *Embodied Intelligence—applies Deep Reinforcement Learning for robotics in manufacturing*

Prior to Amplify I was a PhD student at UC Berkeley focusing on mathematics and deep learning.

To start this story, I want to take what seems at first blush a detour.



This shape right here is known as a Gomboc. It solves a very peculiar and delicate optimization.

The optimization is best understood by considering these balancing dolls. They have one stable point of balance.When pushed, they will return to this stable point.

But the doll does this by having a weight near the bottom where it sits.

A mathematician may look at this property of the doll and ask, and ask, can this hold for a shape with no weighted bottom, for a completely homogenous solid?

Look at cube and sphere not so easy.

The solution is satisfyingly beautiful. The Gomboc.

This strangely named solid solves that delicate optimization. It has a single stable point of equilibrium. When placed on any side it will rebalance to that bottom point.

What is interesting for this talk is it took modern computational optimization to find this solution. In particular the ability to rapidly test hypothesis, iterate on ideas and get feedback.

Such optimization programs helps us explore a complex, nonintuitive space of solutions, and arrive at a better optimization then just analytically studying them.

I argue the power of differentiable programming is similar. But first, what is differentiable programming?

Yann LeCun claims that deep learning is dead. Differentiable programming is a more apt term to describe the advancements enabled by deep learning techniques.

"Deep learning est morte. Vive Differentiable Programming…people are now building a new kind of software by assembling networks of

> parameterized functional blocks and by training them from examples using some form of gradient-based optimization. An increasingly large number of people are defining the networks procedurally in a data-dependent way…allowing them to change dynamically as a function of the input data fed to them. It's really very much like a regular program, except it's parameterized, automatically differentiated, and trainable/optimizable."

As a start, let's establish that DP is not just machine learning, and it is not just another tool in the box. Rather, its a conceptual apparatus for how to apply these tools.

Results may seem diverse and unrelated. From AlphaGo to robotics, drug discovery to enterprise infrastructure, I hope to make the progress and future directions more comprehensible to a business audience beyond just being this bag of great results.

My goal is to explain why DP is a better framework to understand applications in machine intelligence and how we can use this understanding to better anticipate where applications will rise and thus direct our attention to the infrastructure that is needed to achieve this.

(Aside): Neural network, types and functional programming. Chris Olah

To study how this translates to applications, I will highlight three key areas that are particularly amenable to a differentiable programming approach. These three areas can really be viewed as principles that helps us understand the commonality between instances deep learning applications.

1. *Feature engineering is done by optimization, not you.*
2. *Very suitable for to end to end approaches.*
3. *Incredible potential as generative method.*

FEATURE ENGINEERING IS DONE BY OPTIMIZATION, NOT YOU.

This is the dream of differentiable programming.

And what I mean by: We get to be really dumb if we get a little smarter.

Key here is to have architectures that are expressive enough to create good representations of many different types of inputs. Ideally we can leave the algorithm design to neural networks.

So what are some examples?

1. ***NEURAL NETWORK ARCHITECTURE SEARCH***

Neural architecture search uses a sequence to sequence model to design from scratch a neural network for image data as well as language data.

It trains many models in parallel and uses accuracy on test data as a signal for whether the design choices are good.(The controller's samples architectures after learning from that signal and the process iterates.) Designing motifs of filters and other things is the learning problem.

Architectures created from scratch outperform most human invented architectures.

2. ***THE CASE FOR LEARNED INDEX STRUCTURES***

DIFFERENTIAL PROGRAMMING TO REPLACE CLASSICAL DATA STRUCTURES

One of the most attention grabbing results at NIPS this past December.

The key idea behind these results is that there are classical data structures and functions that are provably optimal for general distributions or worst case scenarios, but which may not be optimal if one has more information about the data.

Instead, we can have a model learn the sort order or structure of lookup keys and use this signal to effectively predict the position or existence of records. Initial results show that neural nets are able to outperform cache-optimized B-trees by up to 70% in speed while saving an order-of-magnitude in memory over several real-world datasets.

There have been criticisms about fair comparisons, such as not accounting for the training time. Regardless, the impact of the result is clear. From here, we are led to explore how differentiable programming could fundamentally change even the most basic building blocks of computer science

3. ***MATHEMATICS***

LEARNING SOPHISTICATED GRAPH ALGORITHMS

Even in mathematics.

Clustering on certain sparse graphs can be very difficult, especially in regimes where the sparsity reaches a point where it is informationally theoretically impossible to detect the communities. In this case, there are many mathematical algorithms that took a decade to figure out, based on deep connections with theoretical physics. My co author (Joan Bruna) and I noticed that the form of these algorithms, being spectral, can be expressed in approximate form with a type of differentiable network.

VERY SUITABLE FOR END TO END APPROACHES

Differential Programming allows us to move towards more end to end systems.

May be harder to train but may also avoid redundant information, since only optimizing for the outcome.

End to end approaches are conceptual and mathematical elegant: the system is trained in a holistic manner based on a single principle. Every *learning step is directed at the final goal, encoded by the overall objective function. There is no need to train modules on an "auxiliary" objective, unrelated to the task. One reason to prefer this is that it may be easier is that the representations may learn irrelevant information if we artificially break down the problem. If the system learns in an end to end way, we are better at optimizing for the entire pipeline.*

Examples:

1. ***WHAT CONTENT TO SERVE?***

RL for newsfeed.

Contextual bandits as a service.

RL to solve engagement in facebook. This is now not an army of data scientists doing A/B testing. Still in its infancy, but this allows the optimization surface to be far more comprehensive. If we have enough data, it can do better. With deep neural networks, we can better represent this data, so allows for multimodal representations (image data, multi language data).

70% lift against classical AB testing.

"A common methodology for exploration is A/B testing [31, 32]: policy A is tested against policy B by running both live on a small percentage of user traffic. Outcomes are measured statistically so as to determine whether B is better than A. This methodology requires data scaling linearly with the number of policies to be tested and it requires that the policy to be evaluated is known during data collection. "

"Contextual bandits [3, 34] is a line of work in machine learning allowing testing and optimization over exponentially more policies for a given number of events2 . We refer to this dramatic improvement in capability as Multiworld Testing (MWT). The essential reason for such an improvement is that each data point can be used to evaluate all the policies picking the same action for the same context (i.e., make the same decision for the same input features rather than just a single policy as in A/B testing. An

important property of MWT is that policies being tested do not need to be approved, implemented in production, and run live for a period of time (thus saving much business and engineering effort). Furthermore, the policies do not even need to be known during data collection."

2. ***ALPHAGO ZERO***

LEARNING FROM SCRATCH

One architecture to train both the policy and value. It is called the two headed model. Input raw pixels of the board. Good generalization with the same architecture to other games (Chess…etc).

In the case of games it is easier to encode the input. But for the real world, for robotics, both the input and output are harder. I encourage you to check out work in deep reinforcement learning for robotics.

3. ***AUTOMATED FARMING***

Robots move the seedlings around. Better sensors. The direction is to automate the entire process, collecting enough data to optimize for growing, nutrition, speed.etc.… So far it is not end to end.

But it is a natural problem to cast in this paradigm.

*4. **END-TO-END TRAINING OF DEEP VISUOMOTOR POLICIES***

More generally deep reinforcement learning for robotics, this end to end method by Sergey Levine, Chelsea Finn, Trevor Darrell and Pieter Abbeel at Berkeley.

INCREDIBLE POTENTIAL AS GENERATIVE METHOD

Examples

1. ***ALIGNING BOOKS AND MOVIES***

Towards Story-like Visual Explanations by Watching Movies and Reading Books.

This example is really interesting.

CNN-RNN to label scenes, we don't have really rich labels that gives a deeper description of the scene.

Unique dataset, books that were made to movies to train a network to align the text to scenes, since there was a source of truth.

Can do more, align other text to scenes, or even use scenes to generate text.

Potential for creative usecase, creative tooling.

"Books provide us with very rich, descriptive text that conveys both fine-grained visual details (how people or scenes look like) as well as high-level semantics (what people think and feel, and how their states evolve through a story). This source of knowledge, however, does not come with associated visual information that would enable us to ground it with descriptions. Grounding descriptions in books to vision would allow us to get textual explanations or stories behind visual information rather than simplistic captions available in current datasets. It can also provide us with extremely large amount of data (with tens of thousands books available online)."

2. ***DESIGN***

DIFFERENTIABLE PROGRAMMING AIDED WEB DESIGN

Really interesting case. We can do a image to sequence model. Due to modularity of these differentiable programming units. Input is a sketch of what we want, and output is the code that generates it.

3. ***DRAWING***

Collaborative interface.

Great for collaboration tools.

Neural network generated output fit into human workflow.

Put on investor hat: what can we create that integrates well with workflow.

4. ***MUSIC***

Input: midi files of real piano performances (to learn the expressivity of humans). output: generated music.

Test: Which one that follows is human generated, which one is computer generated?

Who thought it was A, B? Actually there is no trick, there was a human generated one and a machine generated one. Depending on the audience. Now two more because they are so good.

WHAT IS STILL HARD?

Now that we have heard all of these success stories, what is still hard?

Answer: Lots. Here are just a few things.

Adversarial examples.

Powerful and brittle representations.We need to understand these weakness better.

Causality

What makes us different from other animals?

Proposing counterfactuals. Intervene to test and then associate the correlates to Draw conclusions.

Bilingual model for causal inference.

Human-level AI cannot emerge from model-blind *learning machines. Ability to infer? Ability to create models of the world and test it.*

See Judea Pearl for more info.

Infrastructure 3.0

Majority of code in a ml/dl pipeline is dedicated to extraction, cleaning, shaping. Much less is dedicated to actual modeling. How do we made this infrastructure efficient for differentiable programming?

Compute substrate. To scale processes up to number of computations as the brain requires either a lot more power and better chips or an entirely new substrate that doesn't require such demanding energy consumptions.

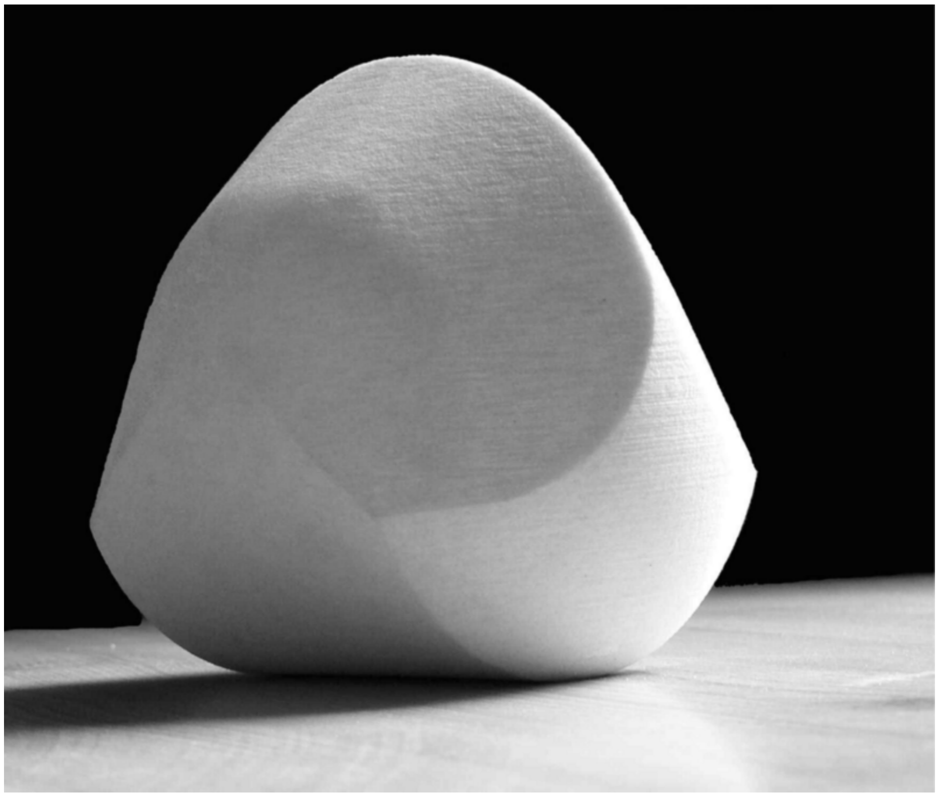
As an investor I am very interested in companies building this up.

SUMMARY

DIFFERENTIABLE PROGRAMMING ENABLED MACHINE INTELLIGENCE

- *Feature engineering is done by optimization, not you.*
- *Very suitable for to end to end approaches.*
- *Incredible potential as generative method.*

So in sum, I've shown you the three principles I find useful to understand differentiable programming, where it shines and instances of their applications.



Remember the Gomboc, where our story began? This strange yet beautiful convex object satisfying a delicate constraint. We needed the ability to iterate with compute to arrive at this shape.

GOMBOC IN NATURE

Interestingly enough there are actually Gomboc in nature. The Tortoise shell evolved to this shape.

Feedback signal in nature is survival.

.Nature gets to explore complex spaces because it can run in parallel many experiments with the computational power of reality. One can say that the feed back in nature was survival, it rans an evolutionary process. Using this as an inspiration, we can look forward to the framework of differential programming to enable us to give beautiful and exquisite optimizations.