Neural Networks, Types, and Functional Programming

Posted on September 3, 2015

An Ad-Hoc Field

Deep learning, despite its remarkable successes, is a young field. While models called artificial neural networks have been studied for decades, much of that work seems only tenuously connected to modern results.

It's often the case that young fields start in a very ad-hoc manner. Later, the mature field is understood very differently than it was understood by its early practitioners. For example, in taxonomy, people have grouped plants and animals for thousands of years, but the way we understood what we were doing changed a lot in light of evolution and molecular biology. In chemistry, we have explored chemical reactions for a long time, but what we understood ourselves to do changed a lot with the discovery of irreducible elements, and again later with models of the atom. Those are grandiose examples, but the history of science and mathematics has seen this pattern again and again, on many different scales.

It seems quite likely that deep learning is in this ad-hoc state.

At the moment, deep learning is held together by an extremely successful tool. This tool doesn't seem fundamental; it's something we've stumbled on, with seemingly arbitrary details that change regularly. As a field, we don't yet have some unifying insight or shared understanding. In fact, the field has several competing narratives!

I think it is very likely that, reflecting back in 30 years, we will see deep learning very differently.

Deep Learning 30 Years in the Future

If we think we'll probably see deep learning very differently in 30 years, that suggests an interesting question: how are we going to see it? Of course, no one can actually know how we'll come to understand the field. But it is interesting to speculate.

At present, three narratives are competing to be the way we understand deep learning. There's the neuroscience narrative, drawing analogies to biology. There's the representations narrative, centered on transformations of data and the manifold hypothesis. Finally, there's a probabilistic narrative, which interprets neural networks as finding latent variables. These narratives aren't mutually exclusive, but they do present very different ways of thinking about deep learning.

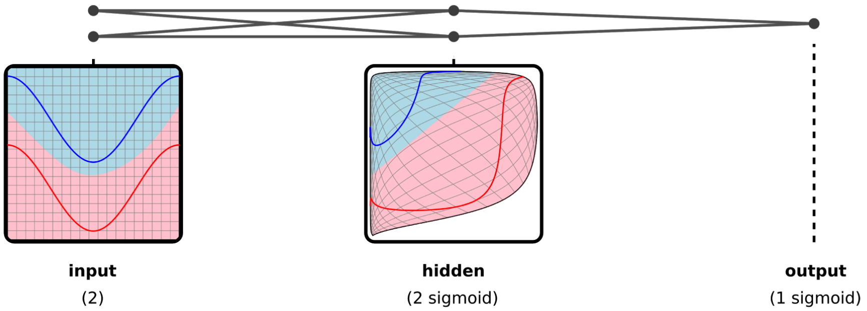
This essay extends the representations narrative to a new answer: deep learning studies a connection between optimization and functional programming.

In this view, the representations narrative in deep learning corresponds to type theory in functional programming. It sees deep learning as the junction of two fields we already know to be incredibly rich. What we find, seems so beautiful to me, feels so natural, that the mathematician in me could believe it to be something fundamental about reality.

This is an extremely speculative idea. I am not arguing that it is true. I wish to argue only that it is plausible, that one could imagine deep learning evolving in this direction. To be clear: I am primarily making an aesthetic argument, rather than an argument of fact. I wish to show that this is a natural and elegant idea, encompassing what we presently call deep learning.

Optimization & Function Composition

The distinctive property of deep learning is that it studies deep neural networks – neural networks with many layers. Over the course of multiple layers, these models progressively bend data (http://colah.github.io/posts/2015-01-Visualizing-Representations/#neural-networks-transform-space), warping it into a form where it is easy to solve the given task.



The details of these layers change every so often.[1] What has remained constant is that there is a sequence of layers.

Each layer is a function, acting on the output of a previous layer. As a whole, the network is a chain of composed functions. This chain of composed functions is optimized to perform a task.

Every model in deep learning that I am aware of involves optimizing composed functions. I believe this is the heart of what we are studying.

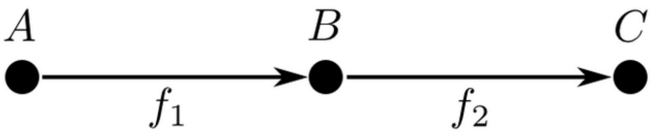
Representations are Types

With every layer, neural networks transform data, molding it into a form that makes their task easier to do. We call these transformed versions of data "representations."

Representations correspond to types.

At their crudest, types in computer science are a way of embedding some kind of data in n bits. Similarly, representations in deep learning are a way to embed a data manifold in n dimensions.

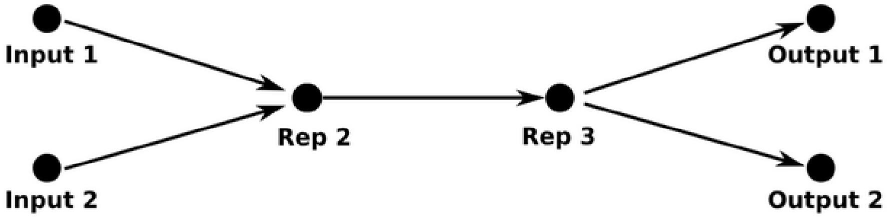
Just as two functions can only be composed together if their types agree, two layers can only be composed together when their representations agree. Data in the wrong representation is nonsensical to a neural network. Over the course of training, adjacent layers negotiate the representation they will communicate in; the performance of the network depends on data being in the representation the network expects.



A layer f_1 followed by a layer f_2. The output representation of f_1 is the input of f_2.

In the case of very simple neural network architectures, where there's just a linear sequence of layers, this isn't very interesting. The representation of one layer's output needs to match the representation of the next layer's input – so what? It's a trivial and boring requirement.

But many neural networks have more complicated architectures where this becomes a more interesting constraint. For a very simple example, let's imagine a neural network with multiple similar kinds of inputs, which performs multiple, related tasks. Perhaps it takes in RGB images and also grayscale images. Maybe it's looking at pictures of people, and trying to predict age and gender. Because the similarities between the kinds of inputs and between the kinds of tasks, it can be helpful to do all of this in one model, so that training data helps them all. The result is multiple input layers mapping into one representation, and multiple outputs mapping from the same representation.



Perhaps this example seems a bit contrived, but mapping different kinds of data into the same representation can lead to some pretty remarkable things (http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/). For example, by mapping words from two languages into one representation, we can find corresponding words that we didn't know were translations when we started. And by mapping images and words into the same representation, we can classify images of classes we've never seen!

Representations and types can be seen as the basic building blocks for deep learning and functional programming respectively. One of the major narratives of deep learning, the manifolds and representations narrative, is entirely centered on neural networks bending data into new representations. The known connection between geometry, logic, topology, and functional programming suggests that the connections between representations and types may be of fundamental significance.

Deep Learning & Functional Programming

One of the key insights behind modern neural networks is the idea that many copies of one neuron can be used in a neural network.

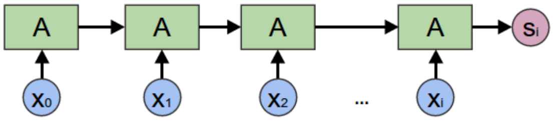
In programming, the abstraction of functions is essential. Instead of writing the same code dozens, hundreds, or even thousands of times, we can write it once and use it as we need it. Not only does this massively reduce the amount of code we need to write and maintain, speeding the development process, but it also reduces the risk of us introducing bugs, and makes the mistakes we do make easier to catch.

Using multiple copies of a neuron in different places is the neural network equivalent of using functions. Because there is less to learn, the model learns more quickly and learns a better model. This technique – the technical name for it is 'weight tying' – is essential to the phenomenal results we've recently seen from deep learning.

Of course, one can't just arbitrarily put copies of neurons all over the place. For the model to work, you need to do it in a principled way, exploiting some structure in your data. In practice, there are a handful of patterns that are widely used, such as recurrent layers and convolutional layers.

These neural network patterns are just higher order functions – that is, functions which take functions as arguments. Things like that have been studied extensively in functional programming. In fact, many of these network patterns correspond to extremely common functions, like fold. The only unusual thing is that, instead of receiving normal functions as arguments, they receive chunks of neural network.[2]

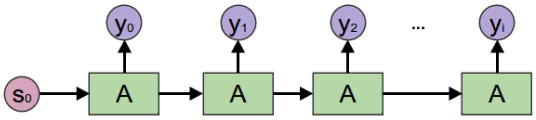
- **Encoding Recurrent Neural Networks** are just folds. They're often used to allow a neural network to take a variable length list as input, for example taking a sentence as input.



fold = Encoding RNN

Haskell: `foldl a s`

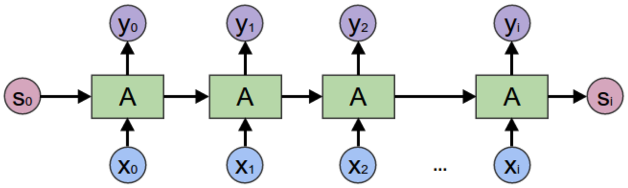
- **Generating Recurrent Neural Networks** are just unfolds. They're often used to allow a neural network to produce a list of outputs, such words in a sentence.



unfold = Generating RNN

Haskell: `unfoldr a s`

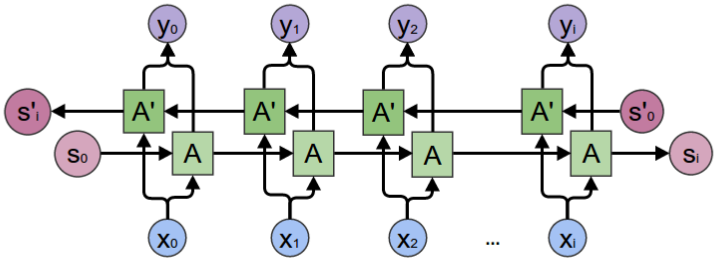
- **General Recurrent Neural Networks** are accumulating maps. They're often used when we're trying to make predictions in a sequence. For example, in voice recognition, we might wish to predict a phenome for every time step in an audio segment, based on past context.



Accumulating Map = RNN

Haskell: `mapAccumR a s`

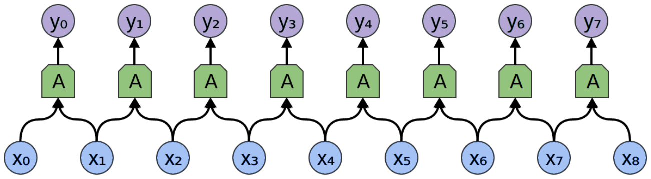
- **Bidirectional Recursive Neural Networks** are a more obscure variant, which I mention primarily for flavor. In functional programming terms, they are a left and a right accumulating map zipped together. They're used to make predictions over a sequence with both past and future context.



Zipped Left & Right Accumulating Map = Bidirectional RNN

Haskell: `zip (mapAccumR a s xs) (mapAccumL a` s` xs)`

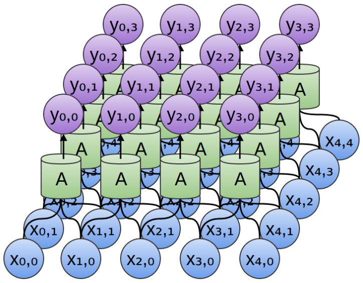
- **Convolutional Neural Networks** are a close relative of map. A normal map applies a function to every element. Convolutional neural networks also look at neighboring elements, applying a function to a small window around every element.[3]



Windowed Map = Convolutional Layer

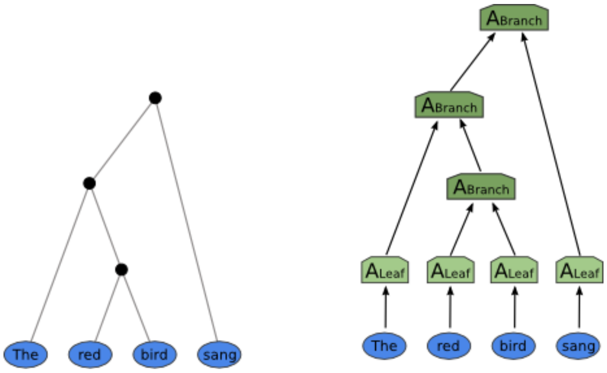
Haskell: `zipWith a xs (tail xs)`

Two dimensional convolutional neural networks are particularly notable. They have been behind recent successes in computer vision. (More on conv nets. (http://colah.github.io/posts/2014-07-Conv-Nets-Modular/))



Two Dimensional Convolutional Network

- **Recursive Neural Networks** ("TreeNets") are catamorphisms, a generalization of folds. They consume a data structure from the bottom up. They're mostly used for natural language processing, to allow neural networks to operate on parse trees.



Catamorphism = TreeNet

Haskell: `cata a`

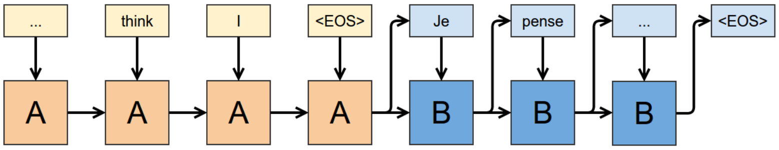
The above examples demonstrate how common patterns in neural networks correspond to very natural, simple functional programs.

A New Kind of Programming

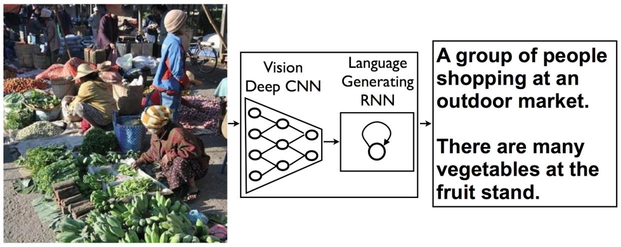
These patterns are building blocks which can be combined together into larger networks. Like the building blocks, these combinations are functional programs, with chunks of neural network throughout. The functional program provides high level structure, while the chunks are flexible pieces that learn to perform the actual task within the framework provided by the functional program.

These combinations of building blocks can do really, really remarkable things. I'd like to look at a few examples.

- Sutskever, *et al.* (2014) (http://arxiv.org/pdf/1409.3215.pdf) perform state of the art English to French translation by combining an encoding RNN and a generating RNN. In functional programming terms, they essentially fold over the (backwards) English sentence and then unfold to produce the French translation.



- Vinyals, *et al.* (2014) (http://arxiv.org/pdf/1411.4555.pdf) generate image captions with a convolutional network and a generating RNN. Essentially, they consume the image with a convolutional network, and then unfold the resulting vector into a sentence describing it.


Vision
Deep CNN
Language
Generating
RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

(Vinyals, *et al.* (2014) (http://arxiv.org/pdf/1411.4555.pdf))

These sorts of models can be seen as a kind of differentiable functional programming.

But it's not just an abstract thing! They're imbued with the flavor of functional programming, even if people don't use that language. When I hear colleagues talk at a high level about their models, it has a very different feeling to it than people talking about more classical models. People talk about things in lots of different ways, of course – there's lots of variance in how people see deep learning – but there's often an undercurrent that feels very similar to functional programming conversations.

It feels like a new kind of programming altogether, a kind of differentiable functional programming. One writes a very rough functional program, with these flexible, learnable pieces, and defines the correct behavior of the program with lots of data. Then you apply gradient descent, or some other optimization algorithm. The result is a program capable of doing remarkable things that we have no idea how to create directly, like generating captions describing images.

It's the natural intersection of functional programming and optimization, and I think it's beautiful.

Conclusion

I find this idea really beautiful. At the same time, this is a pretty strange article and I feel a bit weird posting it. I'm very strongly presenting a speculative idea with little support behind it besides my own enthusiasm. Honestly, adopting the most objective perspective I can, I expect this idea is wrong, because most untested ideas are wrong. But it could be right, and I think it's worth talking about.

Beyond that, I'm not really the right person to explore a lot of the directions this suggests – for example, one of the obvious things to do is to analyze neural networks from a homotopy type theory perspective, but I don't have the relevant background. This is an idea that's begging for broader discussion. It really seems like publishing this is the right thing to do.

Finally, I'm hoping this essay might stir up more discussion and thoughts about what deep learning is really about. I think there's an important discussion waiting to be had.

Besides, what's the point of writing a blog if I can't speculate? Hopefully, I've been able to appropriately balance my excitement with my uncertainty.

Acknowledgments

Firstly, I'm incredibly grateful to Greg Corrado (http://research.google.com/pubs/GregCorrado.html) and Aaron Courville (https://aaroncourville.wordpress.com/). They are the deep learning researchers I know who most strongly empathize with these ideas, and I'm really grateful for their intellectual support.

Several other people have had really extended and helpful conversations with me. Sebastian Zany spent several hours talking about type theory and neural networks with me. Michael Nielsen (http://michaelnielsen.org/) gave thorough feedback on a draft of this essay. Chas Leichner (https://twitter.com/flower_snark) thought really deeply about these ideas, and was very encouraging. A big thank you to the three of them!

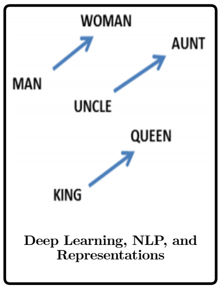
I'm also thankful for the comments of James Koppel (http://www.jameskoppel.com/), Luke Vilnis (http://people.cs.umass.edu/~luke/), Sam Bowman (http://stanford.edu/~sbowman/), Greg Brockman (https://gregbrockman.com/) and Rob Gilson (https://github.com/D1plo1d). Finally, I've spoken with a number of other people about these ideas in the last few months – thanks to all of them!

Appendix: Functional Names of Common Layers

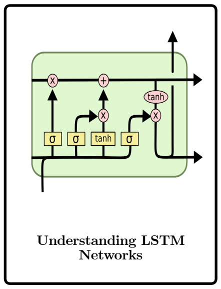
Deep Learning Name	**Functional Name**
Learned Vector	Constant
Embedding Layer	List Indexing
Encoding RNN	Fold
Generating RNN	Unfold
General RNN	Accumulating Map
Bidirectional RNN	Zipped Left/Right Accumulating Maps
Conv Layer	"Window Map"
TreeNet	Catamorphism
Inverse TreeNet	Anamorphism

1. For example, the once ubiquitous sigmoid layer has been substantially replaced by ReLU layers.↩
2. I think it's actually kind of surprising that these sort of models are possible at all, and it's because of a surprising fact. Many higher order functions, given differentiable functions as arguments, produce a function which is differentiable almost everywhere. Further, given the derivatives of argument functions, you can simply use chain rule to calculate the derivative of the resulting function.↩
3. This operation is also closely related to stencil/convolution functions, which are their linear version. They're typically implemented using those. However, in modern neural net research, where "MLP convolution layers" are becoming more popular, it seems preferable to think of this as an arbitrary function.
↩

More Posts

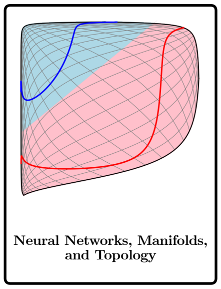


Deep Learning, NLP, and Representations



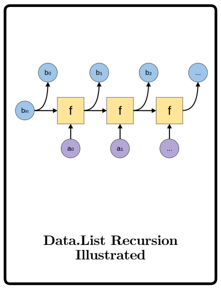
Understanding LSTM Networks

(../../posts/2014-07-NLP-RNNs-Representations/) (../../posts/2015-08-Understanding-LSTMs/)



Neural Networks, Manifolds, and Topology

(../../posts/2014-03-NN-Manifolds-Topology/)



Data.List Recursion Illustrated

(../../posts/2015-02-DataList-Illustrated/)

36 Comments (/posts/2015-09-NN-Types-FP/#disqus_thread)

36 Comments colah's blog 1 Login ▾

Recommend 10 Share Sort by Best ▾

Join the discussion…

Yann LeCun · 8 months ago

For a complete treatise (with type theory) of automatic differentiation in a functional setting (for neural nets and other things), have a look at Jeff Siskind and Barak Pearlmutter's work (here: http://www.bcl.hamilton.ie/~ba...)

Barak A. Pearlmutter and Jeffrey Mark Siskind. Reverse-Mode AD in a functional framework: Lambda the ultimate backpropagator. TOPLAS 30(2):1-36, Mar 2008, doi:10.1145/1330017.1330018.

Alexey Radul, Barak A. Pearlmutter, and Jeffrey Mark Siskind, AD in Fortran, Part 2: Implementation via Prepreprocessor, Advances in Automatic Differentiation, Lecture Notes in Computational Science and Engineering, Springer, 2012. Also arXiv:1203.1450.

Barak A. Pearlmutter and Jeffrey Mark Siskind. Using programming language theory to make AD sound and efficient. In Proceedings of the 5th International Conference on Automatic Differentiation, pages 79-90, Bonn, Germany, Aug 2008, doi:10.1007/978-3-540-68942-3_8. Springer-Verlag.

9 ^ | v • Reply • Share ›

Steven Krouse · 9 months ago

So this would make neural nets the "opposite" of QuickCheck, right?

QuickCheck goes from types and functions to test data, and neural nets go from types and test data to functions.

2 ⌃ | ⌄ • Reply • Share ›

chrisolah Mod ➦ Steven Krouse • 9 months ago

Something like that. :)

⌃ | ⌄ • Reply • Share ›

Andrew Tulloch • 9 months ago

@chrisolah have you seen https://github.com/ajtulloch/d... That's something I put together a few months ago one long weekend that embodies some of these ideas. It's essentially an EDSL in Haskell that uses monadic versions of these combinators to construct the module graph, which then gets dumped to Caffe/Torch/Purine/graphviz/etc. Thought you might find it interesting.

2 ⌃ | ⌄ • Reply • Share ›

chrisolah Mod ➦ Andrew Tulloch • 9 months ago

Very neat, thanks for sharing!

⌃ | ⌄ • Reply • Share ›

ZygmuntZ • 9 months ago

"DL is a young field – perhaps ten years old." Hinton, LeCun, Bengio, especially Schmidhuber, would probably disagree.

To elaborate a bit, let's consider convolutional networks and recurrent networks, both bread-and-butter of deep learning.

1. Yann LeCun's page on LeNet (the first well-known CNN) lists papers from 1989 to 1998.
http://yann.lecun.com/exdb/len...

2. LSTM, currently the most popular flavour of RNN, was proposed in 1997.
http://people.idsia.ch/~juerge...

Having said that, I think 10 years ago is a good estimate of when deep learning became mainstream in machine learning.

2 ⌃ | ⌄ • Reply • Share ›

Dev Nag ➦ ZygmuntZ • 9 months ago

It sounds like you both converged to saying the same thing (with which I agree) -- that Deep Learning became mainstream/branded around 2006, most specifically with Hinton's paper "A fast learning algorithm for deep belief nets."

But that doesn't make Deep Learning a young field, unless you're going by PR alone. **@ZygmuntZ** notes that two of the dominant models popular within DL were proposed (and implemented!) 20-25 years ago. This wasn't da Vinci drawing up a (flawed) helicopter that couldn't be built for another 400+ years -- these were working, validated, benchmarked models running on the hardware of the day.

Selecting unsupervised autoencoder pretraining as a more substantial beginning is even more problematic because of what's happened since -- namely, that unsupervised pretraining has been greatly de-emphasized over the last 3-4 years. Even the Big Three (Bengio/Hinton/LeCunn) state in their 2015 Nature paper that "unsupervised learning (refs 91-98) had a catalytic effect in reviving interest in deep learning, but has since been overshadowed by the successes of purely supervised learning."

Note that they also talk about "reviving" interest in DL, meaning that they themselves think of DL as preceding the 2006 Hinton paper. Schmidhuber is on the same page with this view, of course, drawing on the heritage going back to the 1940's here: http://arxiv.org/pdf/1404.7828...

If you read Schmidhuber's paper, I think you'll come to see that the strands of research that fed into the current field are not narrow at all, but foundational and even prophetic.

⌃ | ⌄ • Reply • Share ›

chrisolah Mod ➦ Dev Nag • 9 months ago

(I've dropped any specification of the year until some sort of agreement can be reached.)

• Reply • Share ›

chrisolah Mod ➦ ZygmuntZ • 9 months ago

I picked ten years because that's around when autoencoder pre-training started, and people were able to generally train networks with more than one or two hidden layers. It's true that certain weight-tying schemes made it possible to train deeper models, but those seem like narrow cases. The deep learning brand was explicitly introduced around that time as well.

I'm not particularly tied to the ten year figure. If you felt like "around twenty years" was more fair, I'd be happy to change to that.

Originally I had a much more caveated paragraph, but I cut it down to keep the writing tight. There are some strands of research going back some ways that fed into deep learning, but they seem narrow. There was a lot of research called neural networks and connectionism, but only a fraction of it seems to have become what we now call deep learning. There was a big change between what was being done in the sixties and what we do now, and the community seems to have agreed it was sufficient to call a different area. I'd like to draw the line somewhere.

• Reply • Share ›

hakimkse • 9 months ago

I like the idea. Now develop the mathematics!

1 • Reply • Share ›

Zenna Tavares • 9 months ago

It's interesting to see this post as I am working on something pretty similar at the moment. I think the best formalism to understand neural network in PL terms are John Hugh's "Arrows". Arrows compose together in a way which is more like a general graph structure than the tree structure of normal functional composition. It is actually very natural to express all these neural network structures using arrow combinators.

As for the idea of layers, types and representations, I think there's something to it, but it has to be a lot more complex than that . The idea of a layer does not seem fundamental to me.

1 • Reply • Share ›

chrisolah Mod ➦ Zenna Tavares • 9 months ago

Sounds interesting! Let me know what happens!

> The idea of a layer does not seem fundamental to me.

Hm. The idea of individual neurons doesn't seem fundamental to me! :) The thing about layers is that they have a very nice interpretation as a single geometric entity rather than a collection of neurons.

1 • Reply • Share ›

Gonçalo Lopes • a month ago

I really liked your table of equivalences between NN topologies and FP operators. Very conspicuously the monadic "bind" operator is missing... what would "bind" look like in NN topology space?

• Reply • Share ›

Qian Hong • 3 months ago

Hello! could we train a big Recurrent Neural Networks superN so that it works like a general Turing machine? Consider we have a small size input x, a small recurrent neural network N, network N takes x as input and generates N(x) as output. Now we encode N into another small size input n, combine x and n together as a large size input [n, x], network superN takes [n, x] as input and generates superN([n,x]) as output. Is there anyway to train superN so that for any possible x (with limited size) and any possible N (with limited size), superN([n,x]) is always closed enough to N(x)?

• Reply • Share ›

bobpoekert • 3 months ago

All of the major deep learning tools (Theano, Torch, mxnet, TensorFlow) are DSLs built on top of well-defined dataflow-based abstract machines and compilers for those machines. I wouldn't call that "ad-hoc", and I'm not sure that functional programming has anything to add that couldn't be done by adding some more type information to the existing dataflow abstractions.

• Reply • Share ›

Paul-Olivier Dehaye • 9 months ago

You might enjoy what Joshua Tan is currently thinking about, here: http://www.joshuatan.com/wp-co...

• Reply • Share ›

PMG • 9 months ago

Are Neural Networks recursive functions? Of course they are. They are a subset of all recursive functions with N elements for NNs with N neurons, with the architecture of the NN giving you the class of functions.
Is functional programming based on recursion? Yup. This idea has been used to generate code (in LISP, mostly) using genetic algorithms to solve problems, not really novel. I've no idea if somebody already used NNs but I'd bet on it.

• Reply • Share ›

Shane • 9 months ago

This is actually really spooky to me as I have been intensely focused on deep learning, but recently have found my way to functional programming (Scala specifically) and I keep thinking "there must be a relationship here". I am also starting to think that both FP and DL adhere to the Principle of Least Effort in a radical way. They are both composed of lazy but efficient algorithms. Much like myself! This is exciting to see a much more formalized discussion of my "hunch". I am hopeful that FP can help push these advances past their current constraints, including the curse of dimensionality and the (somewhat related) issues with allowing computers separated by our puny ethernet networks to collaborate and increase training time. Final note: I think they both just share a fanatical devotion to mathematic principles, which makes them natural partners.

• Reply • Share ›

Vitaly • 9 months ago

Thanks for sharing, Chris! I had similar thoughts, though this is the first post I've seen where someone mentions it explicitly.
I'd like to add that some deep learning frameworks actually employ functional approach to a certain extent:
1) plain Theano (everything is defined as a function: output, loss, weights' updates);
2) Caffe2 (refactoring of Caffe, where a model is defined as a graph of simple operations applied to input data and parameters; there are stateless operations instead of stateful layers);
3) Computational Network Toolkit [CNTK] (similar to Caffe2).

• Reply • Share ›

chrisolah Mod ➦ Vitaly • 9 months ago

Absolutely! A lot of deep learning frameworks have a functional flavor, and I don't think that's a coincidence.

• Reply • Share ›

alexisvallet • 9 months ago

I have been thinking along the same lines for the past few months, here are my modest contributions. :)

I started from the realization that recurrent nets, as you point out, are basically folds and unfolds of sequences. As has been done for trees, perhaps one can generalize to arbitrary data structures by using ana- and catamorphisms of the corresponding algebraic data types. This led me to the idea of a "category of neural nets", where objects are representations and morphisms are functions with forward pass, backward pass and weights. I started playing around in Haskell with the ideas over there: https://github.com/alexisValle... (very experimental at the moment).

Turns out there are 2 meaningful ways to compose such neural net "layers" to form a category structure (haven't written a proof of it though). Either requiring all layers to share the same weights, or keeping them separate. The former category's anamorphisms and catamorphisms seem to correspond very closely to recurrent neural nets when applied to the Haskell list datatype. The latter leads to traditional feed-forward nets.

I'm similarly hoping that allowing neural nets to work with progressively more powerful patterns of computation - feed-

I'm similarly hoping that allowing neural nets to work with progressively more powerful patterns of computation - feed forward, recurrent, anamorphisms and catamorphisms - will help getting us out of the "ad-hoc" state of deep learning. Perhaps the functional programming approach will lead to a dead-end, but it's worth a shot I think :p .

• Reply • Share ›

tksfz alexisvallet • 6 months ago

The category theory interpretation really feels like the most natural path forward here.
Representations are the objects.
Neurons and neural nets are arrows between representations.
Neural network "patterns" (described in the post as analogous to FP higher order functions) are functors.
Finally, the natural transformations are the compositions of neural network layers.

I'd love to see this idea worked out in greater detail.

• Reply • Share ›

Paul-Olivier Dehaye tksfz • 6 months ago

David Spivak's "Category Theory for the Sciences" goes a long way towards that, although it is for essentially static data. Joshua Tan's ideas are close too http://www.joshuatan.com/wp-co... . The two have started to work together.

• Reply • Share ›

Rong Ou • 9 months ago

Traditionally probability theory is formalized as measure theory based on set theory (ZFC). Since HoTT aims to extend the foundations of mathematics, perhaps if one formalizes probability theory in HoTT we'll see your second and third narratives on deep learning are equivalent.

• Reply • Share ›

chrisolah Mod Rong Ou • 9 months ago

I've been wondering if one can get a fuzzy logic version of the Curry-Howard Correspondence. Instead of having elements of a set be a proof, we have samples of distributions...

• Reply • Share ›

sky7drake chrisolah • 9 months ago

the "soft" types...

• Reply • Share ›

This comment was deleted.

chrisolah Mod Guest • 9 months ago

[I think the deleted comment was a really good one. I'd encourage the author to post it again.]

For what it's worth, I don't think most deep learning researchers know much functional programming, at least explicitly. I had to tell some people what a fold is. :P

> What would convince me otherwise would be someone coming up with (useful) deep learning results by 'converting' results from type theory.

Agreed! That would make me feel a lot more confident as well! It's kind of weird: what's the right heuristic for judging a narrative of a field? It isn't really a question of fact, it's more like "is this a productive story for researchers to tell themselves?" I think this is a pretty story, but the only way we'll know if it's productive is if results seem to come out of it. I guess we'll have to reflect on it in a decade.

2 • Reply • Share ›

Andy chrisolah • 9 months ago

Hah! Deleted my original comment because I decided I hadn't done enough thinking about it yet to say anything, but you caught me.

In case anyone is wondering what Chris is replying to, I said that there are two possible reasons for this isomorphism: either it's a natural way to structure computation emerging in two unrelated fields, or this isomorphism: either it's a natural way to structure computation emerging in two unrelated fields, or it's the ideas of one field (FP) unconsciously influencing how computation is thought about in the other (DL).

What I then went on to say - and what I decided was ill thought out - is the bit Chris has quoted. The problem is that on second thought, whether results from one field can be mapped onto the other really doesn't say anything about which of the two above cases holds. I'm now wondering what would be a good way to distinguish one from the other, if it's possible at all.

⌃ | ⌄ • Reply • Share ›

Mark Ettinger • 9 months ago

Very stimulating! On an application level, there may also be an intersection of homotopy type theory and deep learning. One of the motivations for HoTT is formal proof verification but this is quite labor intensive. Could we use deep learning to translate "informal" (journal style) mathematics to formal mathematics suitable for proof-checking (e.g. Coq)?

⌃ | ⌄ • Reply • Share ›

zitterbewegung ↪ Mark Ettinger • 9 months ago

People have already applied machine learning to Coq / ACL2 see http://arxiv.org/abs/1404.3034 , http://arxiv.org/abs/1410.5467 . But I don't see anything where you use deep learning on it.

I suppose that if you basically took a github download of every Coq program and attempted to learn on that you might get something. Downloading every ACL2 program is quite easy since they have their own journal attached to it. I don't know about Coq.

⌃ | ⌄ • Reply • Share ›

chrisolah Mod ↪ Mark Ettinger • 9 months ago

Formalizing mathematics in this sense sounds like a machine learning task, an deep learning may be an excellent approach. It's more at the "application level" of deep learning, though, and I'd be surprised if it worked particularly well because of low level possible connections between the subjects.

On a related topic: Deep learning is really good at pattern recognition. It feels like the hard part of automating proving theorems is magic "human intuition" which seems a lot like pattern recognition. There's been a couple research results exploring the use of neural networks to help in theorem proving, and I suspect it's a fruitful direction...

⌃ | ⌄ • Reply • Share ›

Paul-Olivier Dehaye ↪ chrisolah • 9 months ago

For proof *assistants*, the difficulty is in search: how do you find a result that you need and has already been proved, among millions of others?

⌃ | ⌄ • Reply • Share ›

Jack Scully ↪ Paul-Olivier Dehaye • 8 months ago

That may be the wrong question. Deep learning may solve the problem a different way. "What proofs does this share similar features with? What proven results did those proofs depend on? If we apply the same transformation (look at difference vectors) does that give us the answer or put us in the right neighborhood?"

⌃ | ⌄ • Reply • Share ›

Mark Ettinger ↪ chrisolah • 9 months ago

Before applying DL to theorem proving, I'd start with something "easier" like verification. And before verification I'd start with something still "easier" like informal to formal translation.

⌃ | ⌄ • Reply • Share ›

Cybernetic1 • 6 months ago

Very interesting.... I am thinking about something very similar, perhaps exactly the same -- a recurrent neural network acting on a cognitive state, as a general and universal model of "thinking". Let me send you my paper and see if you're interested...

⌃ | ⌄ • Reply • Share ›

Reply • Share ›

Benoit Descamps • 8 months ago

Amazing. I have started studying neural networks myself. I come from a particular field of condensed matter physics. Convolutional neural networks look very similar to Matrix Products States (MPS) and Projected Entangled Pair States (PEPS) studied in my field (http://arxiv.org/abs/0907.2796....

• Reply • Share ›

ALSO ON **COLAH'S BLOG**

Visual Information Theory

23 comments • 8 months ago

Avat **Heng** — Great article! Just be curious about what tool you use to plot these figures?

Calculus on Computational Graphs: Backpropagation

1 comment • 9 months ago

Avat **Jedd Haberstro** — Thank you for the insightful article. If it were possible not to do so in an overly complicating way, the expression tree …

Neural Networks, Types, and Functional Programming

2 comments • 9 months ago

Avat **zitterbewegung** — This is really interesting. I think that if also if you apply machine learning to attempt to generate programs you may be …

Conv Nets: A Modular Perspective

1 comment • 9 months ago

Avat **hkh** — Am an outsider to the field. What does sigma mean here? Is it the abstraction of operations applied to (Wx+b) such as ReLUs …

Subscribe Add Disqus to your site Add Disqus Add Privacy