What is differentiable programming?

Author: Mike Innes

taken from https://fluxml.ai/blogposts/2019-02-07-what-is-differentiable-programming/

> With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.
>
> -- John Von Neumann

The idea of "differentiable programming" is coming up a lot in the machine learning world. To many, it's not clear if this term reflects a real shift in how researchers think about machine learning, or is just (another) rebranding of "deep learning". This post clarifies what new things differentiable programming (or ∂P) brings to the machine learning table.

Most importantly, differentiable programming is actually a shift *opposite* from the direction taken by deep learning; from increasingly heavily parameterised models to simpler ones that take more advantage of problem structure.

Brute Force with Benefits

Differentiability is the core idea that makes deep learning so successful. Where a brute force search over even a few hundred model parameters would be far too expensive, gradients mean that we can take a pseudo-random walk around interesting parts of parameter space and find a good set. Doing the next-dumbest thing to brute force gives up surprisingly little generality; it's far from obvious that we can come up with a differentiable objective when, say, working with sequences in language translation, but it turns out to be straightforward with a little ingenuity.

What about biological neurons and \(y = σ(W \times x + b)\)? There's nothing particularly special about this formula; it's just a simple and flexible example of a highly-parameterised non-linear function. In fact, it's probably the worst such function in most cases. A single neural net layer can, in principle, classify cat pictures, but only by the relatively uninteresting trick of acting as a lookup table. It's *guaranteed* to work! -- but the small print warns that you may need more parameters than there are atoms in the universe. To actually make it work you need to *encode problem structure in the model* -- which is where it starts to look a lot more like traditional programming.

[...]

Encoding Structure, Redux

ML toolkits increasingly support algorithmic differentiation (AD), allowing us to differentiate models with for loops, branches and recursion -- or any program built on a set of differentiable mathematical primitives. This has led to more complex architectures: NLP models increasingly look more like classical grammar parsers with stack-augmented models, and one can even implement a Turing machine analog or a programming language interpreter in a differentiable way.

The final step taken by differentiable programming is to no longer see matrix multiplies, convolutions and RNNs as the fundamental building blocks of deep learning, but instead as mere special cases. We can apply the techniques of deep learning to *any* parameterised, differentiable function \(f(x)\). \(f\) could be a matrix multiply, but functions as complex as physics simulators or ray tracers can be differentiated and optimised just as well. Even quantum computation can fit into this framework.

Scientists have long used mechanistic models that sit between explicit programming and machine learning. Differential equations fit to data via sensitivities -- as in physics, epidemiology or pharmacodynamics -- are equivalent in all but terminology to neural networks. They are just far more constrained in what functions they can represent, making it easier to get a good result.

The really powerful advance is this: pervasive differentiability means all these techniques snap together like lego bricks. Rather than always writing new programs for ML, we can incorporate existing ones, enabling physics engines inside deep learning-based robotics models. Where current reinforcement learning algorithms need to build a detailed model of the external world from only a coarse-grained reward signal (which sounds like a brute force problem if anything does), we can instead just drop in detailed, precise knowledge of physical systems before training even begins.

[...]

As in the sciences, hybrid models can both be more effective and resolve some of the tradeoffs between deep learning and explicit programming. For example, a drone's flight-path planner might have a neural network component that can only make provably small adjustments to a reliable explicit program, making its overall behaviour analysable while still adapting to empirical data. This is also good for interpretability: parameters of mechanistic models and simulations typically have clear physical interpretations, so if a model estimates parameters internally, it's making a clear statement about what it thinks is happening outside.

If this is all so wonderful, why isn't everybody doing it already? Unfortunately, limitations of current frameworks make it difficult to build models of this complexity, and impossible to reuse the wealth of knowledge embedded in existing scientific code. The need to re-implement physics engines from scratch, in a constrained modelling language -- usually with significant limitations -- turns what should be a ten-line script into a multi-year research project. But advances in language and compiler technology, especially automatic differentiation, are bringing us closer to the holy grail: "just differentiate my game engine, please."

So, what is differentiable programming?

Differentiable programming applies the techniques of deep learning to *complex existing programs*, taking advantage of the huge amount of knowledge embedded within them. Deep learning, statistics, programming and the sciences all have something to say about modelling the world around us, and it's time to put these fields into a particle collider. This will both improve current models and enable ML to be applied in areas where its current limitations -- either interpretability or its computational and data requirements -- make it infeasible alone.

Thanks to Hannah Lepper and James Bradbury for feedback on this post.

-- Mike Innes