Software is eating the world, but ML is going to eat software

"Democratizing ML" is a hot topic these days - particularly in industry. Efficiency, composability and accessibility of machine learning technology are active areas of investment for many research and product groups. Unfortunately, while machine learning has the potential to fundamentally improve how software is constructed, opportunities to leverage machine learning to improve more conventional developer tools (languages, compilers, and IDEs for example) have largely gone untapped.

At Facebook we want to seize this opportunity. Our Developer Infrastructure team is on a mission to fundamentally rethink and retool Facebook's developer toolchain by applying machine learning at every layer in our stack. Our goal is to make our developers more productive, and our processes and infrastructure more efficient, by deeply integrating ML into our programming languages and developer tools (such as IDEs, version control, or continuous integration systems) in novel ways.

This talk will detail the work our team has done to improve developer efficiency and resource utilization at Facebook - from updating the Hack programming language to support probabilistic programming techniques, to developing a new suite of AI-driven developer tools. I'll describe the lessons we've learned along the way, as well as future opportunities we see to optimize or auto-tune other common pieces of developer infrastructure.
Sisyphus’ Curse

abstraction we need

abstraction we have

\[ A \xrightarrow{f} B \xleftarrow{g} Y \]

\[ \text{Hoare 1972} \]
Our Abstractions

invented our own math to model all this

$f \in A \rightarrow B$

mutable edge labeled graphs

polymorphic side-effecting procedures arbitrary nesting
Relational Databases

Table

Codd 1970

Leverage clever existing math (relational algebra)
Easy to reason about & optimize declarative & pure query language

OO/Relational Impedance mismatch

Copeland & Maier 1984
SQL

no SQL

co SQL

new SQL

can't beat the math
Can we abstract?

Kleisli Category

←

Relational Algebra

LINQ nice try
# Software 1.0

## Web
- Javascript
- React
- Vue
- Cycle
- Angular

## Mobile
- Android
- iOS
- Swift
- Objective C
- Java
- Kotlin
- C#

## Server
- Java
- Rust
- Go
- C++
- D
Software 2.0

In contrast, **Software 2.0** is written in neural network weights. No human is involved in writing this code because there are a lot of weights (typical networks might have millions), and coding directly in weights is kind of hard (I tried). Instead, we specify some constraints on the behavior of a desirable program (e.g., a dataset of input output pairs of examples) and use the computational resources at our disposal to search the program space for a program that satisfies the constraints. In the case of neural networks, we restrict the search to a continuous subset of the program space where the search process can be made (somewhat surprisingly) efficient with backpropagation and stochastic gradient descent.
Old

Coffee => human => Code

New

Data => Machine => Model
Learning Based Development

Code vs Model

- Code: deterministic, discrete
- Model: uncertainty, continuous

Old math:
- Bayes (1763)
- Calculus (1600)
Supervised Learning

leverage clever ancient math tricks to use error between $b$ and $b'$ to improve/train $f$ across all training examples.
Backpropagation

\[ f \in \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[ f = i \circ \ldots \circ h \quad \text{// differentiable} \]

\[ E(b, b') = \frac{1}{2} \| b - b' \|_2^2 \]

Wrong abstraction again!
Linear Regression

\[ f(a, b) = \lambda x \rightarrow ax + b \]

given training data \( \{ (x_i, y_i) \} \)

find \( a \) & \( b \) to minimize error

\[ \sum_i (y_i - f(a, b) x_i)^2 \]

desired value

observed value
Iterative/Stochastic Gradient Descent

\[ da_{-}error(a) = A \cdot b \quad (x, y) \rightarrow \]
\[ x \ast (f(a, b) \cdot x - y) \]

\[ db_{-}error(b) = A \cdot a \quad (x, y) \rightarrow \]
\[ (f(a, b) \cdot x - y) \]

gradient is the pair of these derivatives
train (data) {  
  var a = ... random ...
  var b = ... random ...

  foreach ((x, y) in data) {  
    update weights
    \[ a := \alpha \cdot da_{-}error \ a \ b \ (x, y) \]
    \[ b := \alpha \cdot db_{-}error \ b \ a \ (x, y) \]
  }

  \text{learning rate}
  \text{return} \ (a, b)
OK, Deep Learning has outlived its usefulness as a buzz-phrase. Deep Learning est mort. Vive Differentiable Programming!

Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers.

But the important point is that 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.
Differentiable Programming
To Define Models
Can we abstract

???

```
In [2]:
    mu = Variable(torch.zeros(1))  # mean zero
    sigma = Variable(torch.ones(1))  # unit variance
    x = dist.normal(mu, sigma)
    print(x)

Variable containing:
1.6869
[torch.FloatTensor of size 1]
```

```
In [7]:
def geometric(p, t=None):
    if t is None:
        t = 0
    x = pyro.sample("x_{}.format(t), dist.bernoulli, p)
    if torch.equal(x.data, torch.zeros(1)):
        return x
    else:
        return x + geometric(p, t+1)

print(geometric(Variable(torch.Tensor([0.5]))))

Variable containing:
0
[torch.FloatTensor of size 1]
```

Deep embedding
DEEP LEARNING WITH
DYNAMIC COMPUTATION GRAPHS

Moshe Looks, Marcello Herreshoff, DeLesley Hutchins & Peter Norvig
Google Inc.
{madscience,marcelloh,delesley,pnorvig}@google.com

ABSTRACT

Neural networks that compute over graph structures are a natural fit for problems in a variety of domains, including natural language (parse trees) and cheminformatics (molecular graphs). However, since the computation graph has a different shape and size for every input, such networks do not directly support batched training or inference. They are also difficult to implement in popular deep learning libraries, which are based on static data-flow graphs. We introduce a technique called dynamic batching, which not only batches together operations between different input graphs of dissimilar shape, but also between different nodes within a single input graph. The technique allows us to create static graphs, using popular libraries, that emulate dynamic computation graphs of arbitrary shape and size. We further present a high-level library\(^1\) of compositional blocks that simplifies the creation of dynamic graph models. Using the library, we demonstrate concise and batch-wise parallel implementations for a variety of models from the literature.
Backprop as Functor: A compositional perspective on supervised learning

Brendan Fong  David I. Spivak  Rémy Tuyéras

Department of Mathematics,
Massachusetts Institute of Technology

Abstract

A supervised learning algorithm searches over a set of functions $A \to B$ parametrised by a space $P$ to find the best approximation to some ideal function $f : A \to B$. It does this by taking examples $(a, f(a)) \in A \times B$, and updating the parameter according to some rule. We define a category where these update rules may be composed, and show that gradient descent—with respect to a fixed step size and an error function satisfying a certain property—defines a monoidal functor from a category of parametrised functions to this category of update rules. A key contribution is the notion of request function. This provides a structural perspective on backpropagation, as well as a broad generalisation of neural networks.
The Simple Essence of Automatic Differentiation (Invited Talk)

Track  
PEPM 2018

When  
Mon 8 Jan 2018 16:00 - 17:00 at Crocker - Session 1-3 Chair(s): Frank Pfenning

Abstract  
Automatic differentiation (AD) is often presented in two forms: forward mode and reverse mode. Forward mode is quite simple to implement and package via operator overloading but is inefficient for many problems of practical interest such as deep learning and other uses of gradient-based optimization. Reverse mode (including its specialization, back-propagation) is much more efficient for these problems, but is also typically given much more complicated explanations and implementations, involving mutation, graph construction, and “tapes”. This talk develops a very simple specification and Haskell implementation for mode-independent AD based on the vocabulary of categories (generalized functions). Although the categorical vocabulary would be difficult to write in directly, one can instead write regular Haskell programs to be converted to this vocabulary automatically (via a compiler plugin) and then interpreted as differentiable functions. The result is direct, exact, and efficient differentiation of Haskell programs with no notational overhead. The specification and implementation are then generalized considerably by parameterizing over an underlying category. This generalization is then easily specialized to forward and reverse modes, with the latter resulting from a simple dual construction for categories. Another instance of generalized AD is automatic incremental evaluation of functional programs, again with no notational impact to the programmer.

Conal Elliott  
Target, USA
### Some Principles of Differential Programming Languages

<table>
<thead>
<tr>
<th>Track</th>
<th>POPL 2018 Research Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>When</td>
<td>Thu 11 Jan 2018 08:30 - 09:30 at <strong>Bunker Hill / Watercourt</strong> - Keynote-II</td>
</tr>
<tr>
<td>Chair(s)</td>
<td>Andrew Myers</td>
</tr>
<tr>
<td>Abstract</td>
<td>Languages for learning pose interesting foundational challenges. We look at one case: the foundations of differentiable programming languages with a first-class differentiation operator. Graphical and linguistic facilities for differentiation have proved their worth over recent years for deep learning (deep learning makes use of gradient descent optimisation methods to choose weights in neural nets). Automatic differentiation, also known as algorithmic differentiation, goes much further back, at least to the early 1960s, and provides efficient techniques for differentiation, e.g., via source code transformation. It seems fair to say, however, that differentiable programming languages have begun to appear only recently. It seems further fair to say that, while there has certainly been some foundational study of differentiable programming languages, there is ample opportunity to do more.</td>
</tr>
</tbody>
</table>

---

Gordon Plotkin  
*University of Edinburgh, UK*
Duality Applied

minimize loss function

maximize probability
Probability Monad

$P[A]$ probability distribution

\[ \stackrel{\text{discrete or continuous}}{\Rightarrow} \]

\( \ast \in P[A] \times P[B|A] \)

\[ \Rightarrow P[B] \]

\[ \Rightarrow \text{conditional distribution} \]
Declarative, guess $a$ & $b$

```
from $a$ in guess \(\mathcal{N}\)
from $b$ in guess \(\mathcal{N}\)
let $f(x) = ax + b$
from $e$ in noise \(\mathcal{N}\)
from $(x,y)$ in training
where $f(x) \approx y$
select $f$
```

- Conditioning
- Bayesian
- Linear Regression
- Weigh choice of this $f$ by how close $f(x)$ is to $y$
var xs = [0, 1, 2, 3];
var ys = [0, 2, 4, 6];

var functions = Infer({method: 'MCMC', samples: 10000}, function() {
    var m = sample(Gaussian({ mu: 0, sigma: 2}));
    var b = sample(Gaussian({ mu: 0, sigma: 2}));

    var f = function(x) { return m * x + b }

    var sigma = sample(Gamma({shape: 1, scale: 1}));

    map2(
        function(x, y) {
            factor(Gaussian({mu: f(x), sigma: sigma}).score(y));
        }, xs, ys)

    return f;
})

var predict = function() {
    var f = sample(functions)
    return f(4)
}

viz.auto(Infer({method: 'MCMC', samples: 1000}, predict));
Probabilistic Programming
To Compose Models
Reinforcement Learning

What

user

predict what to serve user

(s, r)

max \Sigma r

WWW
Probabilistic Programming

\[ \text{www} \in (s, r) \rightarrow [P[A]] \]

program = imperative Hack/PHP code + machine learned models

Compute a probability distribution of possible actions

_convert that from theory into practice_
probabilistic function $f$ \(\text{img} : \text{Image}\): \([P[\text{ad}]]\) {
   $\text{name} = \text{sample} \ \text{NN} \ (\text{img})$
   
   \text{Awaitable}
   
   $\text{ent} = \text{wait query} \ (\text{name})$
   
   $\text{ad} = \text{sample} \ \text{DT} \ (\text{ent})$
   
   \text{condition} \quad \text{ad} \rightarrow \text{isAppropriate} \ (\text{ent} \rightarrow \text{age})$
   
   \text{return} \ \text{ad}
}

3

Compose uncertainty + latency

Cannot train as single model
Programming = Neural Nets + Probabilities

Read my lips!

Don't believe me
Can we abstract

Invited tutorial

Rif A. Saurous, Google, and Dustin Tran, Columbia University

Deep Probabilistic Programming: TensorFlow Distributions and Edward

Abstract: The TensorFlow Distribution and Edward libraries implement a vision of probability theory adapted to the modern deep-learning paradigm of end-to-end differentiable computation. We first introduce TensorFlow Distributions, an efficient low-level system for building and manipulating distributions. We focus on the non-obvious design choices in the library, paying particular attention to the Bijector abstraction, which supports composable volume-tracking transformations with automatic caching. We then provide an overview of Edward, a probabilistic programming system built on computational graphs and using Distributions as an efficient backend. In particular, we show how Edward and TensorFlow Distributions can be applied for expanding the frontier of deep generative models and variational inference.
Equational reasoning for probabilistic programming.

Track  POPL 2018 TutorialFest

When  Mon 8 Jan 2018 11:00 - 12:00 at Bradbury - Equational reasoning for probabilistic programming
      Mon 8 Jan 2018 09:00 - 10:30 at Bradbury - Equational reasoning for probabilistic programming

Abstract  Equations on programs are used to express domain knowledge, verify correctness, and improve performance, by people such as programmers and by tools such as compilers. It turns out that equations on probabilistic programs are a particularly good way to express Bayesian inference, verify distribution correctness, and improve sampler performance. In this way, this tutorial will introduce the mathematical reasoning principles that practitioners of probabilistic reasoning use to turn declarative models into efficient algorithms. These principles include integration and conjugacy, density and conditioning, and detailed balance.

Chung-chieh Shan  
Indiana University, USA
program =
      imperative code
+ ML model

Test
Build
Train
Version Control

Debug
Developers Are Users

Big Code

Use ML+ static analysis to optimize developer workflow
Autopatching

```javascript
if (algos.indexOf("Bayes")==-1) {
  // -1 is truthy
}
if (algos.indexOf("Bayes")>0) {
  //
  //
} else if (algos.includes("Bayes")) {
  // using machine learning
}
```
Sequence to Sequence Translation

encoder → decoder

the quick brown fox → el veloz zorro marrón

we know everything about sequences of tokens
does not understand EN nor SPA, yet can translate between
AI Complete

\[ x = \text{context} \]

\[
\Rightarrow \quad \text{translate} \quad \Rightarrow \quad \text{completion}
\]

works in theory but predictions must be 'beep' fast!
Neural Code Search

how do I hide the keyboard in React Native?

Predict

```javascript
<TouchableWithoutFeedback
    onPress={() => Keyboard.dismiss()}
    >
    <View style={styles.container}>
    <TextInput keyboardType='numeric'/>
    </View>
</TouchableWithoutFeedback>
& call `Keyboard.dismiss()`
```
Test Run Optimization

Murphy's Law:
- Test fails at the very end, 4 hrs wasted

Random:
- Test fails in the middle, 2 hrs wasted

Run suspicious tests first:
- Test fails immediately, 0 hrs wasted

Development
- Diff
- Pass/fail

Hot code many revisions...
and that is just the most trivial model

don't recompile if target is far away from change
Adaptive Infra

developer

or

1TB

(phone, mins spent)

contextual config

P[size]
Lots of juicy low hanging fruit
Tired of writing grant proposals with $P[\text{Accepted} \mid \text{Effort}] = 0$?

We are hiring!