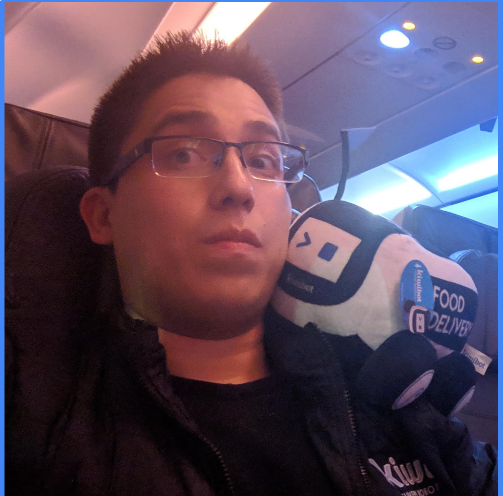
# Automatic differentiation with JAX

**David Cardozo**
ML Engineer
@_davidcardozo

Please reach out with questions



# Our Awesome new world

- TensorFlow, Stan, Theano, Edward, PyTorch, MinPy
- Only need to specify forward model
- Autodiff + optimization / inference done for you

- loops? branching? recursion? closures? data structures?
- debugger?
- a second compiler/interpreter to satisfy

## JAX Overview

- What is JAX?
- Why JAX?
- Overview of JAX

## Jacobians and the chain rule

**Forward and reverse accumulation**

## Autograd implementaation

**Fully closed tracing autodiff in python**

## References

**Models**
Image Transformer
DCGAN

# JAX

01

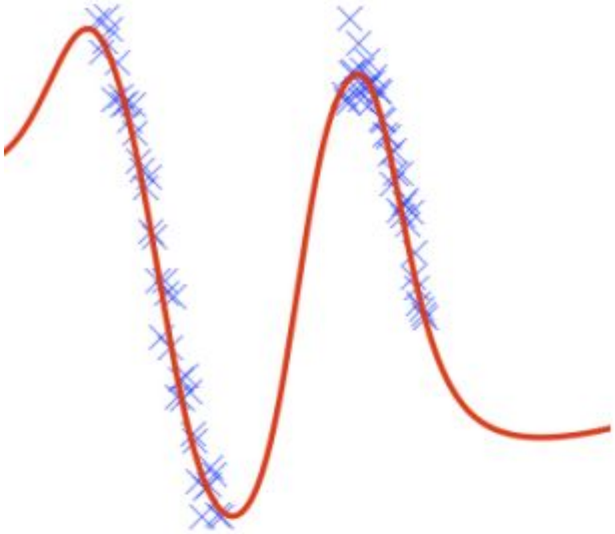
# JAX

- differentiates native Python code
- handles most of Numpy + Scipy
- loops, branching, recursion, closures
- arrays, tuples, lists, dicts, classes, …
- derivatives of derivatives
- a one-function API
- small and easy to extend

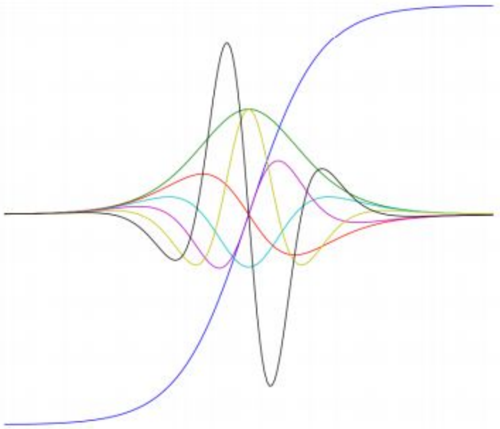
# XLA
# The Linear algebra compiler

ACCELERATED LINEAR ALGEBRA

```python
import jax.numpy as np
import jax.random as npr
from jax import grad

def predict(weights, inputs):
    for W, b in weights:
        outputs = np.dot(inputs, W) + b
        inputs = np.tanh(outputs)
    return outputs

def init_params(scale, sizes):
    return [(npr.randn(m, n) * scale,
             npr.randn(n) * scale)
            for m, n in zip(sizes[:-1], sizes[1:])]

def logprob_fun(params, inputs, targets):
    preds = predict(weights, inputs)
    return np.sum((preds - targets)**2)

gradient_fun = grad(logprob_fun)
```



# JAX Examples



```
import jax.numpy as np
from jax import grad, vmap
import matplotlib.pyplot as plt

x = np.linspace(-7, 7, 200)
plt.plot(x, np.tanh(x),
         x, vmap(grad(np.tanh))(x),
         x, vmap(grad(grad(np.tanh)))(x),
         x, vmap(grad(grad(grad(np.tanh))))(x),
         x, vmap(grad(grad(grad(grad(np.tanh)))))(x), )
```

# Hessians and HVP

```python
from jax import grad, jacobian

def hessian(fun, argnum=0):
    return jacobian(jacobian(fun, argnum), argnum)
def hvp(fun):
    def grad_dot_vector(arg, vector):
        return np.dot(grad(fun)(arg), vector)
    return grad(grad_dot_vector)
```

$$\nabla^2 f(x) \cdot v = \nabla_x (\nabla_x f(x) \cdot v)$$

# Black-box inference in a tweet

**Ryan Adams** @ryan_p_adams · 7 Nov 2015

@DavidDuvenaud

```
def elbo(p, lp, D, N):
 v=exp(p[D:])
 s=randn(N,D)*sqrt(v)+p[:D]
 return mvn.entropy(0, diag(v))+mean(lp(s))
gf = grad(elbo)
```

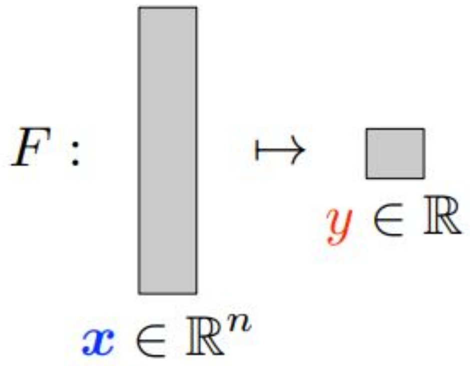
1 7 22

# Jacobians and the chain rule

Forward and reverse accumulation

02

$$F : \mathbb{R}^n \to \mathbb{R}$$




$$F = D \circ C \circ B \circ A \qquad y = F(\boldsymbol{x}) = D(C(B(A(\boldsymbol{x}))))$$

$$y = D(\boldsymbol{c}), \quad \boldsymbol{c} = C(\boldsymbol{b}), \quad \boldsymbol{b} = B(\boldsymbol{a}), \quad \boldsymbol{a} = A(\boldsymbol{x})$$

$$y = D(\boldsymbol{c}), \quad \boldsymbol{c} = C(\boldsymbol{b}), \quad \boldsymbol{b} = B(\boldsymbol{a}), \quad \boldsymbol{a} = A(\boldsymbol{x})$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

$$y = D(\boldsymbol{c}), \quad \boldsymbol{c} = C(\boldsymbol{b}), \quad \boldsymbol{b} = B(\boldsymbol{a}), \quad \boldsymbol{a} = A(\boldsymbol{x})$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

$$F'(\boldsymbol{x}) = \quad \frac{\partial y}{\partial \boldsymbol{c}} \; \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} \; \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}} \; \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}$$

$$y = D(\boldsymbol{c}), \quad \boldsymbol{c} = C(\boldsymbol{b}), \quad \boldsymbol{b} = B(\boldsymbol{a}), \quad \boldsymbol{a} = A(\boldsymbol{x})$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{c}} \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}$$

$$\frac{\partial y}{\partial \boldsymbol{c}} = D'(\boldsymbol{c})$$

$$y = D(\boldsymbol{c}), \quad \boldsymbol{c} = C(\boldsymbol{b}), \quad \boldsymbol{b} = B(\boldsymbol{a}), \quad \boldsymbol{a} = A(\boldsymbol{x})$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{x}} = \left[\frac{\partial y}{\partial x_1} \quad \cdots \quad \frac{\partial y}{\partial x_n}\right]$$

$$F'(\boldsymbol{x}) = \quad \frac{\partial y}{\partial \boldsymbol{c}} \; \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} \; \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}} \; \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}$$

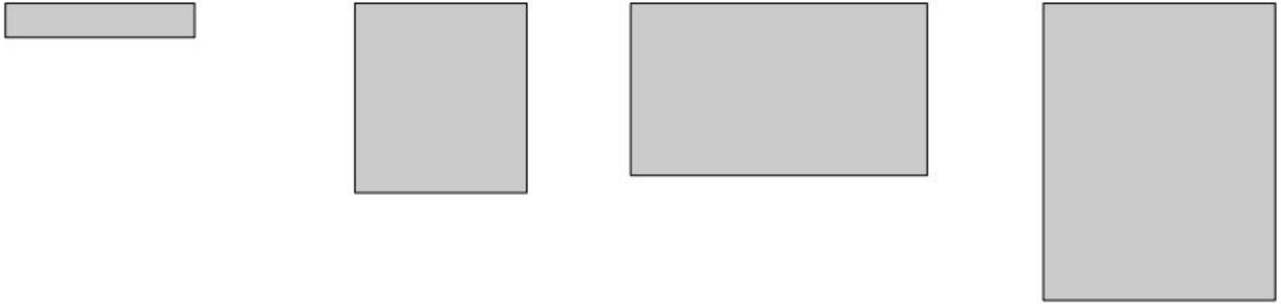
$$\frac{\partial y}{\partial \boldsymbol{c}} = D'(\boldsymbol{c}) \qquad \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} = C'(\boldsymbol{b})$$

$$y = D(\boldsymbol{c}), \quad \boldsymbol{c} = C(\boldsymbol{b}), \quad \boldsymbol{b} = B(\boldsymbol{a}), \quad \boldsymbol{a} = A(\boldsymbol{x})$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{c}} \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}$$

$$\frac{\partial y}{\partial \boldsymbol{c}} = D'(\boldsymbol{c}) \qquad \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} = C'(\boldsymbol{b}) \qquad \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}} = B'(\boldsymbol{a}) \qquad \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}} = A'(\boldsymbol{x})$$



$$F'(\boldsymbol{x}) = \qquad \frac{\partial y}{\partial \boldsymbol{c}} \Bigg( \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} \Bigg( \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}} \quad \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}} \Bigg)$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{c}} \Bigg( \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} \Bigg( \underbrace{\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}} \quad \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}}_{\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial b_1}{\partial x_1} & \cdots & \frac{\partial b_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_m}{\partial x_1} & \cdots & \frac{\partial b_m}{\partial x_n} \end{bmatrix}} \Bigg)$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{c}}\left(\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}}\underbrace{\left(\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\ \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right)}_{\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial b_1}{\partial x_1} & \cdots & \frac{\partial b_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_m}{\partial x_1} & \cdots & \frac{\partial b_m}{\partial x_n} \end{bmatrix}}\right)$$

Forward accumulation

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{c}}\left(\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}}\underbrace{\left(\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\ \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right)}\right)$$

$$\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial b_1}{\partial x_1} & \cdots & \frac{\partial b_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_m}{\partial x_1} & \cdots & \frac{\partial b_m}{\partial x_n} \end{bmatrix}$$

Forward accumulation

$$F'(\boldsymbol{x}) = \left(\left(\frac{\partial y}{\partial \boldsymbol{c}}\ \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}}\right)\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\right)\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{c}}\left(\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}}\underbrace{\left(\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\ \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right)}\right)$$

$$\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial b_1}{\partial x_1} & \cdots & \frac{\partial b_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_m}{\partial x_1} & \cdots & \frac{\partial b_m}{\partial x_n} \end{bmatrix}$$

Forward accumulation

$$F'(\boldsymbol{x}) = \left(\underbrace{\left(\frac{\partial y}{\partial \boldsymbol{c}}\ \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}}\right)}\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\right)\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}$$

$$\frac{\partial y}{\partial \boldsymbol{b}} = \begin{bmatrix} \frac{\partial y}{\partial b_1} & \cdots & \frac{\partial y}{\partial b_m} \end{bmatrix}$$

$$F'(\boldsymbol{x}) = \frac{\partial y}{\partial \boldsymbol{c}} \left( \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} \underbrace{\left( \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}} \right)}_{} \right)$$

$$\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial b_1}{\partial x_1} & \cdots & \frac{\partial b_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_m}{\partial x_1} & \cdots & \frac{\partial b_m}{\partial x_n} \end{bmatrix}$$

Forward accumulation

$$F'(\boldsymbol{x}) = \left( \underbrace{\left( \frac{\partial y}{\partial \boldsymbol{c}} \frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}} \right)}_{} \frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}} \right) \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}$$

$$\frac{\partial y}{\partial \boldsymbol{b}} = \begin{bmatrix} \frac{\partial y}{\partial b_1} & \cdots & \frac{\partial y}{\partial b_m} \end{bmatrix}$$

Reverse accumulation

$$F'(\boldsymbol{x})\,\boldsymbol{v} = \quad \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{c}}\;\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}}\;\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\;\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\,\boldsymbol{v}$$

$$F'(\boldsymbol{x})\,\boldsymbol{v} = \quad \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{c}}\left(\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}}\left(\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\left(\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\,\boldsymbol{v}\right)\right)\right)$$

Forward accumulation $\leftrightarrow$ Jacobian-vector products

Build Jacobian one column at a time

$$F'(\boldsymbol{x}) \quad = \quad \frac{\partial y}{\partial \boldsymbol{c}}\left(\frac{\partial \boldsymbol{c}}{\partial \boldsymbol{b}}\left(\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\left(\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\;\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{x}}\right)\right)\right)$$

$$v^{\mathsf{T}} F'(x) = \qquad v^{\mathsf{T}} \frac{\partial y}{\partial c} \quad \frac{\partial c}{\partial b} \quad \frac{\partial b}{\partial a} \quad \frac{\partial a}{\partial x}$$

$$v^{\mathsf{T}} F'(x) = \Bigg(\Bigg(\Bigg( v^{\mathsf{T}} \frac{\partial y}{\partial c} \Bigg) \frac{\partial c}{\partial b} \Bigg) \frac{\partial b}{\partial a} \Bigg) \frac{\partial a}{\partial x} \Bigg)\Bigg)\Bigg)$$

Reverse accumulation $\leftrightarrow$ vector-Jacobian products

Build Jacobian one row at a time

$$F'(x) = \Bigg(\Bigg(\Bigg( \frac{\partial y}{\partial y} \; \frac{\partial y}{\partial c} \Bigg) \frac{\partial c}{\partial b} \Bigg) \frac{\partial b}{\partial a} \Bigg) \frac{\partial a}{\partial x} \Bigg)\Bigg)\Bigg)$$

# Forward and reverse accumulation

- Forward accumulation
  - Jacobian-vector products
  - “push-forward”
  - build Jacobian matrix one column at a time
- Reverse accumulation
  - vector-Jacobian products
  - “pull-back”
  - build Jacobian matrix one row at a time

# Non-chain composition

Fan-in

$$y = F(\boldsymbol{x}_1, \boldsymbol{x}_2)$$

$$\frac{\partial y}{\partial \boldsymbol{x}_1} = F_1'(\boldsymbol{x}_1, \boldsymbol{x}_2) \qquad \frac{\partial y}{\partial \boldsymbol{x}_2} = F_2'(\boldsymbol{x}_1, \boldsymbol{x}_2)$$

Fan-out

$$G(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} I \\ I \end{bmatrix} \boldsymbol{x}$$

$$G'(\boldsymbol{x}) = \begin{bmatrix} I \\ I \end{bmatrix} \qquad \boldsymbol{v}^\top G'(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{v}_1{}^\top & \boldsymbol{v}_2{}^\top \end{bmatrix} \begin{bmatrix} I \\ I \end{bmatrix} = \boldsymbol{v}_1{}^\top + \boldsymbol{v}_2{}^\top$$

# Autodiff differentiation

## Read and generate source code ahead-of-time

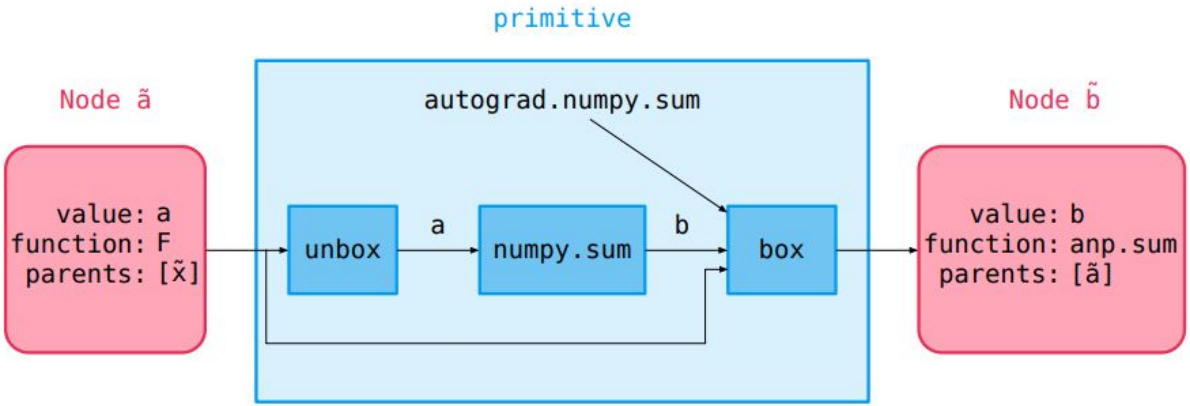
- Source and target language could be python
- Monitor function execution at runtime

## Monitor function execution at runtime

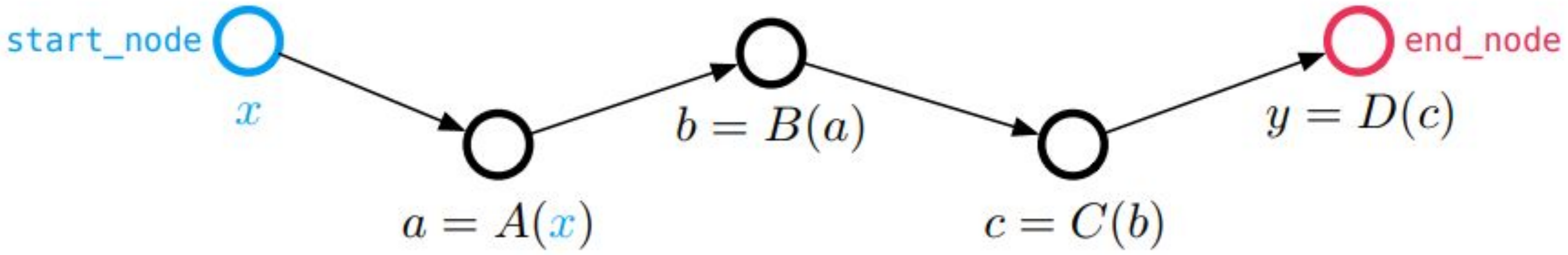
Strategy at Higher level languages

Pytorch mynpy

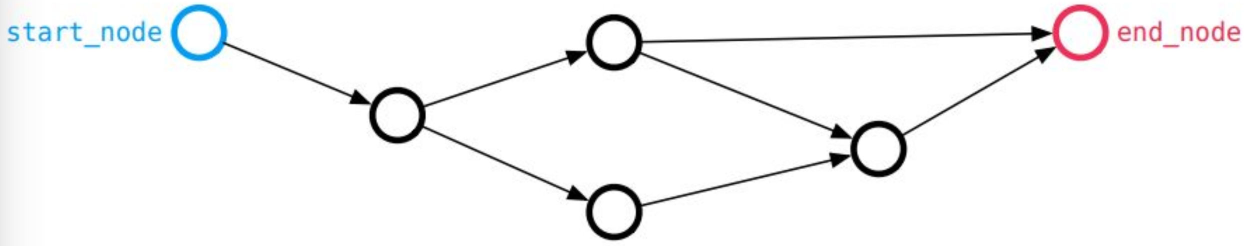
# Tracing the composition of primitive functions



# Tracing on chains

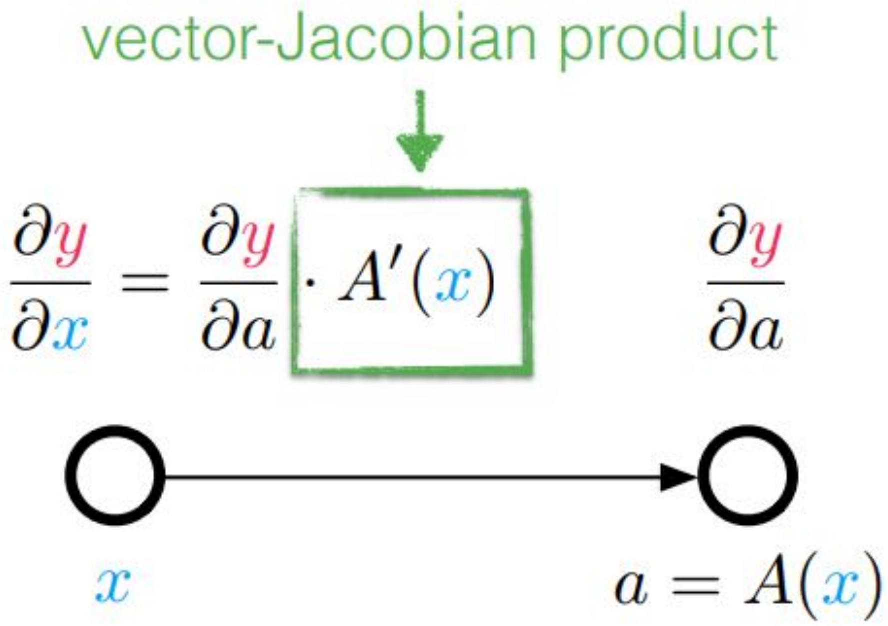


# Graphs on DAGs

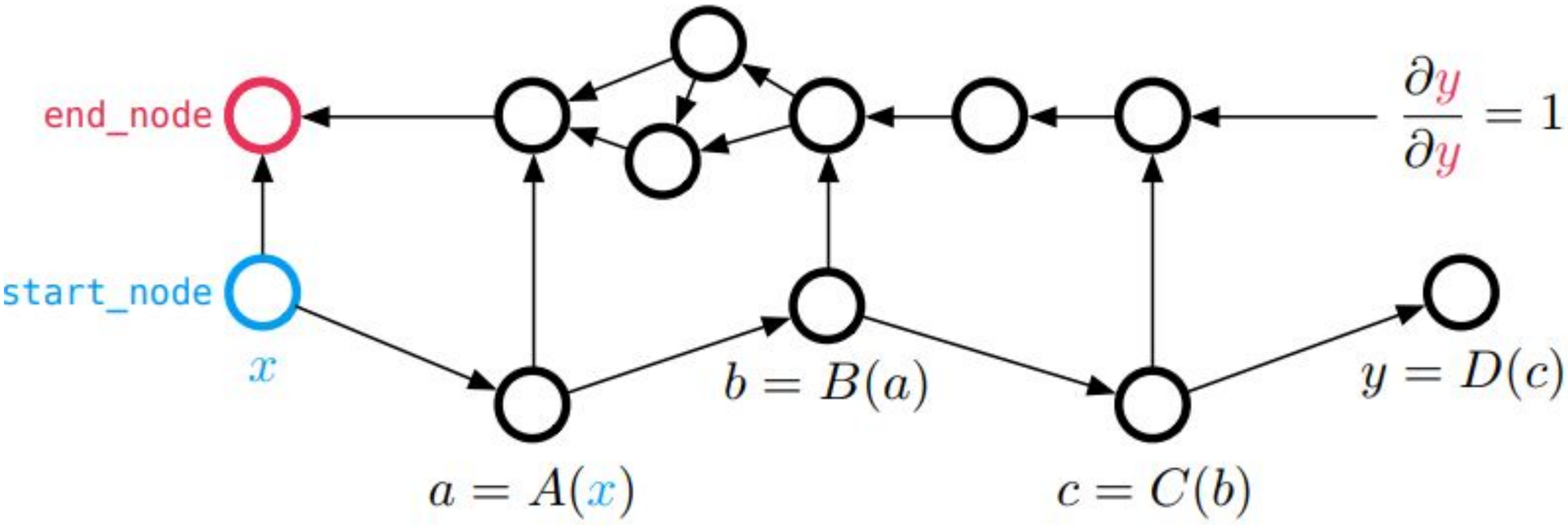



No control flow!

# Defining a vector-jacobian product for each primitive



# Composing VJPs backwards



# Autograd’s ingredients

1. Tracing the composition of primitive functions

   `Node`, `primitive`, `forward_pass`

2. Defining a vector-Jacobian product (VJP) operator for each primitive

   `defvjp`

3. Composing VJPs backward

   `backward_pass`, `make_vjp`, `grad`

# JAX examples

03

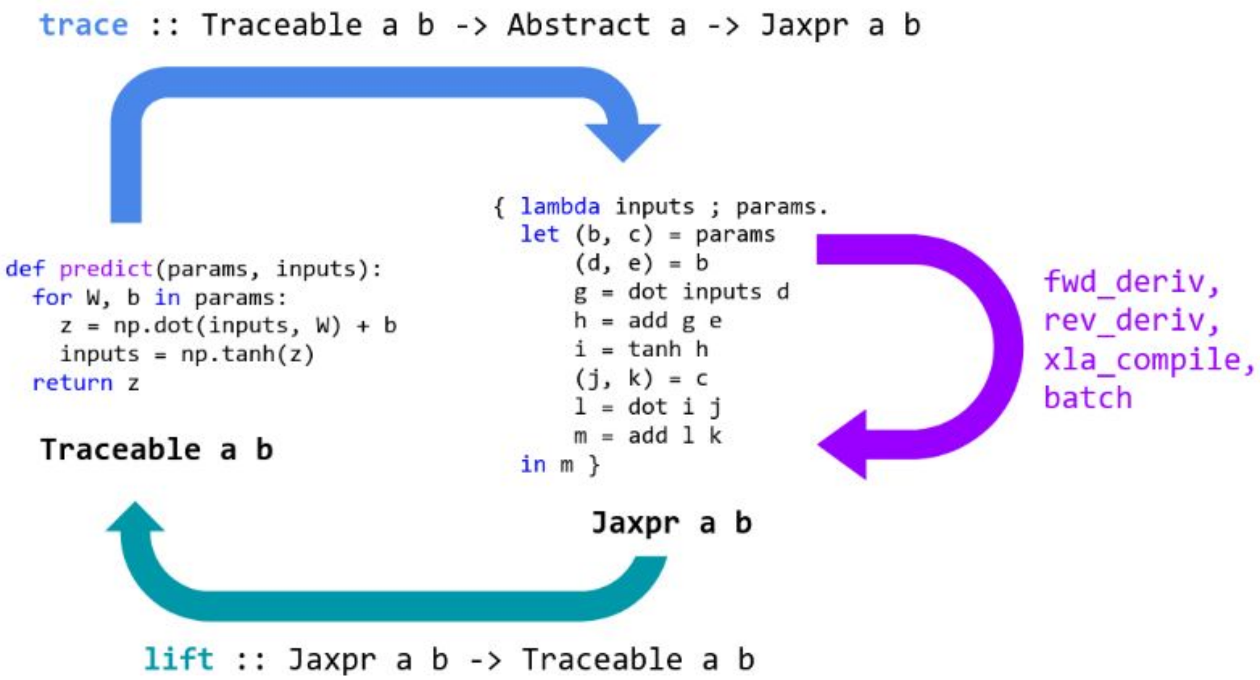
# JIT in JAX

```python
import jax.numpy as np
from jax import jit

def slow_f(x):
  # Element-wise ops see a large benefit from fusion
  return x * x + x * 2.0

x = np.ones((5000, 5000))
fast_f = jit(slow_f)
%timeit -n10 -r3 fast_f(x)  # ~ 4.5 ms / loop on Titan X
%timeit -n10 -r3 slow_f(x)  # ~ 14.5 ms / loop (also on GPU via JAX)
```

# Stax in JAX

```python
import jax.numpy as np
from jax import random
from jax.experimental import stax
from jax.experimental.stax import Conv, Dense, MaxPool, Relu, Flatten, LogSoftmax

# Use stax to set up network initialization and evaluation functions
net_init, net_apply = stax.serial(
    Conv(32, (3, 3), padding='SAME'), Relu,
    Conv(64, (3, 3), padding='SAME'), Relu,
    MaxPool((2, 2)), Flatten,
    Dense(128), Relu,
    Dense(10), LogSoftmax,
)

# Initialize parameters, not committing to a batch shape
rng = random.PRNGKey(0)
in_shape = (-1, 28, 28, 1)
out_shape, net_params = net_init(rng, in_shape)

# Apply network to dummy inputs
inputs = np.zeros((128, 28, 28, 1))
predictions = net_apply(net_params, inputs)
```

# Conclusions



# Thank you !

**David Cardozo**

ML Engineer

@_davidcardozo

Please reach out with questions