Automatic differentiation: The most criminally underused tool in probabilistic numerics

David Duvenaud

VE RI TAS
HARVARD
School of Engineering
and Applied Sciences

Do we know the function we're integrating?

Do we know the function we’re integrating?

- Me: We treat the integrand as an unknown function –
- FB: But we do know what the function is.
- Me: OK, we know how to evaluate it, but we don’t know its integral.
- FB: If you can write it down and evaluate it, how is that different than knowing what the function is?
- Me: ...
- FB: ...

How to exploit our knowledge of integrand?

- Naive quadrature hopeless in high dimensions.
- Control flow can help break up the domain. Or, can find additivity. Can incorporate into custom kernels.
- What about continuous models, or physical simulations?
- Look mostly like compositions of many vector-valued, differentiable operations. f(g(h(x)))
- What to do with this sort of structure?
- Can model each function separately, but this gives deep (intractable) model.

Let's use gradient observations

- Bayesian Quadrature motivated by expense of evaluating function
- Gradients and higher derivatives are available cheaply!
- Even if function isn't everywhere differentiable, if it's continuous and differentiable almost everywhere, then gradients help find high/low regions.
- Chain rule combines info about all elements of composition
- Higher-order gradients give non-local information (Taylor expansion)

Common examples have (stochastic) gradients available

- Physical simulations
- Model hyperparameters (GPs, neural nets)
- Bayesian Quadrature itself! Might want to optimize based on output.
- Solar system models for expolanet detection
- Models of prawn behavior

Price List

	Time cost	num observations
function evaluation	$O(1)$	1
gradient evaluation	$O(2)$	D
Hessian-vector product	$O(4)$	D
full Hessian	$O(4D)$	D^2

How to condition on a Hessian-vector product?

- Automatic differentiation to the rescue again:

$$\mathrm{cov}(f(\mathbf{x}), Af(\mathbf{x}')) = A\,\mathrm{cov}(f(\mathbf{x}), f(\mathbf{x}'))$$

- Holds true for Hess-vec product: $A = \mathbf{v}^T \nabla_{\mathbf{x}} \nabla_{\mathbf{x}}^T$
- So

$$\mathrm{cov}(f(\mathbf{x}), \left[\mathbf{v}^T \nabla_{\mathbf{x}'} \nabla_{\mathbf{x}'}^T f(\mathbf{x}')\right]_d) = \left[\mathbf{v}^T \nabla_{\mathbf{x}'} \nabla_{\mathbf{x}'}^T k(\mathbf{x}, \mathbf{x}')|_{\mathbf{x}'}\right]_d$$

How to code a Hessian-vector product?

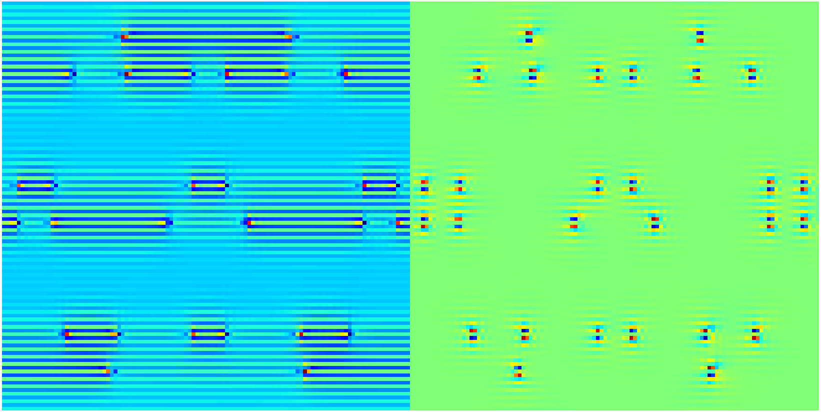
```python
from autograd import grad
from autograd.numpy import np

def hessian_vector_product(fun):
    fun_grad = grad(fun)
    def vector_dot_grad(arg, vector):
        return np.dot(vector, fun_grad(arg))
    return grad(vector_dot_grad)
```

- No explicit Hessian
- Can call HVP on complicated integrand
- Can call HVP on complicated kernel
- Can construct higher-order products

Example: Taking gradient through fluid sim

[Show code and demo]



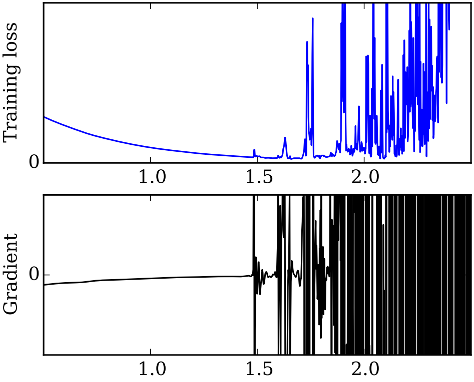
through fluid sim

Memory limitations for large simulations

- If you can reverse dynamics exactly, no need for intermediate storage.
- If you can reverse dynamics approximately, store only lost bits
- Obvious applications are physical simulations, learning dynamics

When will gradients be meaningful?

- Chaotic dynamics → gradient very noisy
- Smooth dynamics → meaningful gradient even for long sims
- Heteroskedastic noise model could gently back off to gradient-free case



learning rate

Autodiff makes the codebase small

- BBQ code was at least half derivatives
- Needed Hessians for Laplace approx
- Changing model or kernel means re-doing all derivatives
- Spearmint uses slice sampling for hyperparameters. Doesn’t scale to high dimensions. Use HMC or BBSVI instead. (more autodiff)

Summary

- Gradient or HVP provide D times more info at only twice the cost
- Ridiculous to say we care about expensive evaluations and ignore this
- HVPs and higher derivatives provide non-local-info
- Can ask which direction is most informative
- Autodiff makes code base simpler + less error prone