Hessian with Backprop.

Pushpendre Rastogi
January 25, 2021

It is possible to compute the action of the hessian of a function on an input vector. Let $\theta \in \mathbb{R}^d$ and let $f : \mathbb{R}^d \to \mathbb{R}$.

Let $J = \partial f$ and $H = \partial^2 f$ be the jacobian and hessian function of f. $H(\theta)$ is itself a linear operator and we'll use H_θ to denote this operator.

The trick is that

$$H_\theta \times \nu = \frac{\partial(J_\theta^T \times \nu)}{\partial \theta}.$$

The proof is trivial by just expanding the term on the r.h.s using the product rule and noting that $\frac{\partial \nu}{\partial \theta} = 0$.

This can be implemented in JAX as follows

```
from scipy.optimize import (
  rosen as scipy_rosen, rosen_hess as scipy_rosen_hess)
import jax
import jax.numpy as jnp

x = jnp.array([2.2, .21])
direction = jnp.array([1.2, 3.1])

def f(x):
    return (100.0 * (x[1:] - x[:-1]**2.0)**2.0 + (1 - x[:-1])**2.0)
    .sum()

J = jax.grad(f)

def Jproduct(x, direction):
    return J(x).dot(direction)

H_action = jax.grad(Jproduct)


assert f(x) == scipy_rosen(x), "rosenbrock function impl wrong."
gold_action = scipy_rosen_hess(x).dot(direction)
trick_action = H_action(x, direction)
print(trick_action - gold_action)
assert(jnp.linalg.norm(trick_action - gold_action)
      / jnp.linalg.norm(gold_action) < 1e-6)

# [-4.8828125e-04 -6.1035156e-05]
```

The second trick is the hutchinson method for extracting the diagonal values of a matrix just by computing the action of the matrix on Rademacher distributed input values. Rademacher R.V. are sampled from a uniform p.m.f. on $\{-1, 1\}^d$. We can see this is in JAX as

```
import numpy as np
def rademacher(n, d):
    return np.random.choice([-1, 1], size=n*d).reshape((n,d))

def approx_diag(x, samples=100):
    return sum(H_action(x, z) * z for z in rademacher(samples, 2))
    / samples

%time print(np.diag(scipy_rosen_hess(x)))
%time print(approx_diag(x, 30))

# [5726.0005  200.     ]
# CPU times: user .445 ms, sys: .081 ms, total: .526 ms
# Wall time: .464 ms

# [5667.3335   141.33333]
# CPU times: user 1 s, sys: 198 ms, total: 1.2 s
# Wall time: 1.03 s
```

See `rise.cs.berkeley.edu/projects/adahessian/` and google for adahessian and pyhessian for more details and references.

Now these noisy hessian computations can be used inside optimization methods. The adahessian method is a full-fledged optimization algorithm proposed in "ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning" based on this idea. See `arxiv.org/pdf/2006.00719.pdf`

1 Applying Hessian-By-Backprop to Trust Region methods

TODO

Well-known methods such as the trust-region method can also be implemented using the `H_action` operator.