Quantum simulation on a random tensor network

February 5, 2022

1 Forward and Reverse mode automatic differentiation

See [4, 5].

2 Differentiating single step time evolution

The m site Rydberg Hamiltonian is

$$H_{\text{Rydberg}} = \sum_{i,j=1,i>j}^{m} \frac{C}{|r_i - r_j|^6} n_i n_j + \Omega(t) \sum_{i=1}^{m} \frac{1}{2} \sigma_i^x + \Delta(t) \sum_{i=1}^{m} n_i \tag{1}$$

For simplicity, we consider the following general representation of a time-space dependent Hamiltonian with k terms

$$H = \sum_{k=1}^{K} c_k O_k \tag{2}$$

where c_k can be dependent on a set of parameters like locations $r_1, r_2, \ldots, r_m$, and pulses $\Omega(t)$ and $\Delta(t)$.

2.1 The ODE version

In each step of the ODE solver, it performs the following update

$$|\psi'\rangle = (1 - iH\Delta t)|\psi\rangle \tag{3}$$

We derive the backward rules for the gradients by inspecting the following differential form.

$$\begin{aligned}\overline{\mathcal{L}}\delta\mathcal{L} &= \overline{|\psi'\rangle} \circ \delta|\psi'\rangle \\ &= \sum_k \overline{c_k}\delta c_k + \overline{|\psi\rangle} \circ \delta|\psi\rangle + \overline{\Delta t}\delta\Delta t\end{aligned} \tag{4}$$

where $\overline{c_k} \equiv \frac{\partial L}{\partial c_k}$ is the adjoint of variable c_k, $\overline{\mathcal{L}} = 1$, $\circ$ is the Hadamard product applied on real numbers. In the definition of adjoint and Hadamard product, we treat a complex number as two real numbers for simplicity, one alternative way to understand the complex adjoints is the Wirtinger derivatives. We rewrite the above equations in a more standard linear algebra style as the following.

$$\begin{aligned}\overline{\mathcal{L}}\delta\mathcal{L} &= \overline{\langle\psi'|}\delta|\psi'\rangle \\ &= \sum_k \overline{c_k}\delta c_k + \overline{\langle\psi|}\delta|\psi\rangle + \overline{\Delta t}\delta\Delta t\end{aligned} \tag{5}$$

where we have used $\langle\psi|$ to represent the hermitian conjugate of $|\psi\rangle$.

$$\delta|\psi'\rangle = -i\sum_k \delta c_k O_k \Delta t|\psi\rangle - iH\delta\Delta t|\psi\rangle + (1 - iH\Delta t)\delta|\psi\rangle \tag{6}$$

By observing Eq. (5) and Eq. (6), one can see

$$\overline{\langle\psi|} = \overline{\langle\psi'|}(1 - iH\Delta t) \tag{7}$$

$$\overline{c_k} = \Re\left[-i\Delta t\overline{\langle\psi'|}O_k|\psi\rangle\right] \tag{8}$$

$$\overline{\Delta t} = \Re\left[-i\overline{\langle\psi'|}H|\psi\rangle\right] \tag{9}$$

After a step, a normalization procedure might be called on the wave functions, this is trivial so that we do not discuss it at this stage.

2.2 The `expmv` version

To differentiate the time evolution directly, one can use the Taylor expansion

$$\begin{aligned}|\psi'\rangle &= e^{-iHt}|\psi\rangle \\ &= \sum_{n=0}^{\infty}\frac{(-it)^n H^n}{n!}|\psi\rangle\end{aligned} \tag{10}$$

Similarly, we have

$$\delta|\psi'\rangle = e^{-iHt}\delta|\psi\rangle + \sum_{n=0}^{\infty}\frac{(-it)^n\delta(H^n)}{n!}|\psi\rangle + \left(e^{-iH(t+\delta t)} - e^{-iHt}\right)|\psi\rangle \tag{11}$$

$$\overline{\langle\psi|} = \overline{\langle\psi|}e^{-iHt}\delta \tag{12}$$

$$\overline{t} = \overline{\langle\psi'|} - iHe^{-iHt}|\psi\rangle \tag{13}$$

$$\overline{c_k} = \sum_n \frac{(-it)^n}{n!}\sum_{p=0}^{n-1}\overline{\langle\psi'|}H^p O_k H^{n-p-1}|\psi\rangle \tag{14}$$

3 How to differentiate an ODE solver

We need a time-space trade-off scheme to trace back intermediate states.

3.1 The adjoint state method (or neural ODE)

Since the time evolution is reversible, one can reverse it by doing inverse time evolution (or the adjoint state method [6, 2]).

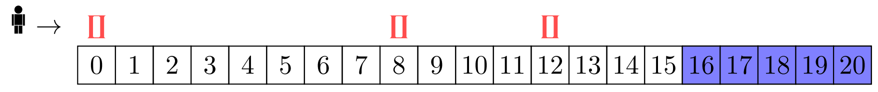
Algorithm 1: The continuous adjoint state method

```
input  : parameters $\theta$, initial time $t_0$, ending time $t_n$, final state $s_n$ and
         its adjoint $\overline{s_n}$
output: $\overline{s_0}$, $\overline{\theta}$
function aug_dynamics(($s$, $a$, -), $t$, $\theta$)
    $q = f(s,t,\theta)$                              # the augmented dynamics
    return ($q$, $-a^T\frac{\partial q}{\partial s}$, $-a^T\frac{\partial q}{\partial \theta}$)
end
$S_0 = (s_n, \overline{s_n}, 0)$         # initial state of the augmented dynamics
($s_0$, $\overline{s_0}$, $\overline{\theta}$) = ODESolve(aug_dynamics, $S_0$, $\theta$, $t_n$, $t_0$)  # integrate the
  augmented dynamics in the reversed time
```

One can easily check the second and third field in the return value of the augmented dynamics is nothing more than taking the $\mathrm{d}t \to 0^+$ in the backward rules of $\overline{s}$ and $\overline{\theta}$.

3.2 The treeverse algorithm (or optimal checkpointing)

For the cases reversibility is not guarented, one can use treeverse algorithm [3, 1] to trace back the states in a reversed order. We map the problem of checkpointing into a real world problem. Peter is a street maintainer. He is assigned a job by his employer to tile a one-way street to blue mosaic in the reversed order (do not ask me why!). This is a job can not be completed for normal people, because once he tiled the last untiled block, he can not move back. Lucky enough, Peter picked up some teleportation magics in Hogwarts, he can setup a teleportation gate at where he locates and teleport to this gate later. In the following illustration, we represent the street as N grids arranged in one dimension.



Peter is allowed take the following actions.

1. tile one block s_i only if he is in i-th block and s_{i+1} is tile,
2. move in one direction, i.e. $i \to i + 1$,

3. setup a teleportation gate (marked with red symbols) at where he locates. He can at most create δ gates at the same time. When this upper limit is reached, he must distroy an existing teleportation gate to create a new one.

4. teleport himself to any existing teleportation gate.

Given $N = 10000$, $\delta = 10$, can you please help Peter design a scheme so that he can move (take action 2) the least?

Here, we map a state to a grid, the program counter to Peter's location, the cached states to the teleportation gate and computing gradients as tiling street. Only if we have the state i and the adjoint of the succeeding state $\overline{s_{i+1}}$, one can back-propagate for one step to obtain $\overline{s_i}$. The total memory limits the number of cached states, which corresponds to the upper limit of the number teleportation gates that can be created at the same time.

The optimal solution is described in [3], where we recursively sweep and devide a block into different size specified by function $\eta(d, \tau - t)$. Here $\eta(d, t) \equiv \frac{(d+t)!}{d!t!}$ is the binomial function, d is the number of available checkpoints in current sweep and t is the number of sweeps in this block. We summarize the algorithm that can be directly implemented in programming languages in Table 2.

References

[1] TreeverseAlgorithm.jl. https://github.com/GiggleLiu/TreeverseAlgorithm.jl.

[2] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[3] Andreas Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and software*, 1(1):35–54, 1992.

[4] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[5] Laurent Hascoet and Valérie Pascual. The tapenade automatic differentiation tool: Principles, model, and specification. *ACM Transactions on Mathematical Software (TOMS)*, 39(3):20, 2013.

[6] R.-E. Plessix. A review of the adjoint-state method for computing the gradient of a functional with geophysical applications. *Geophysical Journal International*, 167(2):495–503, 11 2006.

Algorithm 2: The Treeverse algorithm

```
input : State cache $S = \{0 : s_0\}$, the initial adjoint $\overline{s_n} \equiv \frac{\partial \mathcal{L}}{\partial s_n}$, the
        maximum number of checkpoints $\delta$, the number of scan $\tau$, the
        starting location of current block $\beta = 0$, the end point of
        current block $\phi = n$, and the bisection point of current block
        $\sigma = 0$
output: Back-propagated adjoint $\overline{s_0} \equiv \frac{\partial \mathcal{L}}{\partial s_0}$
1  function treeverse($S$, $\overline{s_\phi}$, $\delta$, $\tau$, $\beta$, $\sigma$, $\phi$)
2      if $\sigma > \beta$ then
3          $\delta = \delta - 1$
4          $s = S[\beta]$                                   # load initial state $s_\beta$
5          for $j = \beta, \beta + 1, ..., \sigma - 1$ do
6              $s_{j+1} = f_j(s_j)$                         # compute $s_\sigma$
7          end
8          $S[\sigma] = s_\sigma$
9      end
10     # let $\kappa$ be the division point, call the treeverse algorithm recursively
11     while $\tau > 0$ and $\kappa = \mathrm{mid}(\delta, \tau, \sigma, \phi) < \phi$ do
12         $\overline{s_\kappa}$ = treeverse($S$, $\overline{s_\phi}$, $\delta$, $\tau$, $\sigma$, $\kappa$, $\phi$)
13         $\tau = \tau - 1$
14         $\phi = \kappa$
15     end
16     $\overline{s_\sigma} = \overline{f_\sigma}(\overline{s_{\sigma+1}}, s_\sigma)$                 # back propagate the gradient
17     if $\sigma > \beta$ then
18         remove($S[\sigma]$)                              # remove state $s_\sigma$ from cache
19     end
20     return $\overline{s_\sigma}$
21 end
22 function mid($\delta$, $\tau$, $\sigma$, $\phi$)
    # choose the bisection point
23     $\kappa = \lceil (\delta\sigma + \tau\phi)/(\tau + \delta) \rceil$
24     if $\kappa \geq \phi$ and $\delta > 0$ then
25         $\kappa = \max(\sigma + 1, \phi - 1)$
26     end
27 end
```