Reinforcement Learning with Neural Networks for Quantum Feedback

Due on February 7, 2019

Section A

Jinlai Ning

Introuduction

This paper use two-stage learning with teacher and student networks and a reward quantifying the capability to recover the quantum information stored in a multiqubit system to show how a network-based agent can discover complete quantum-error-correction strategies, protecting a collection of qubits against noise.

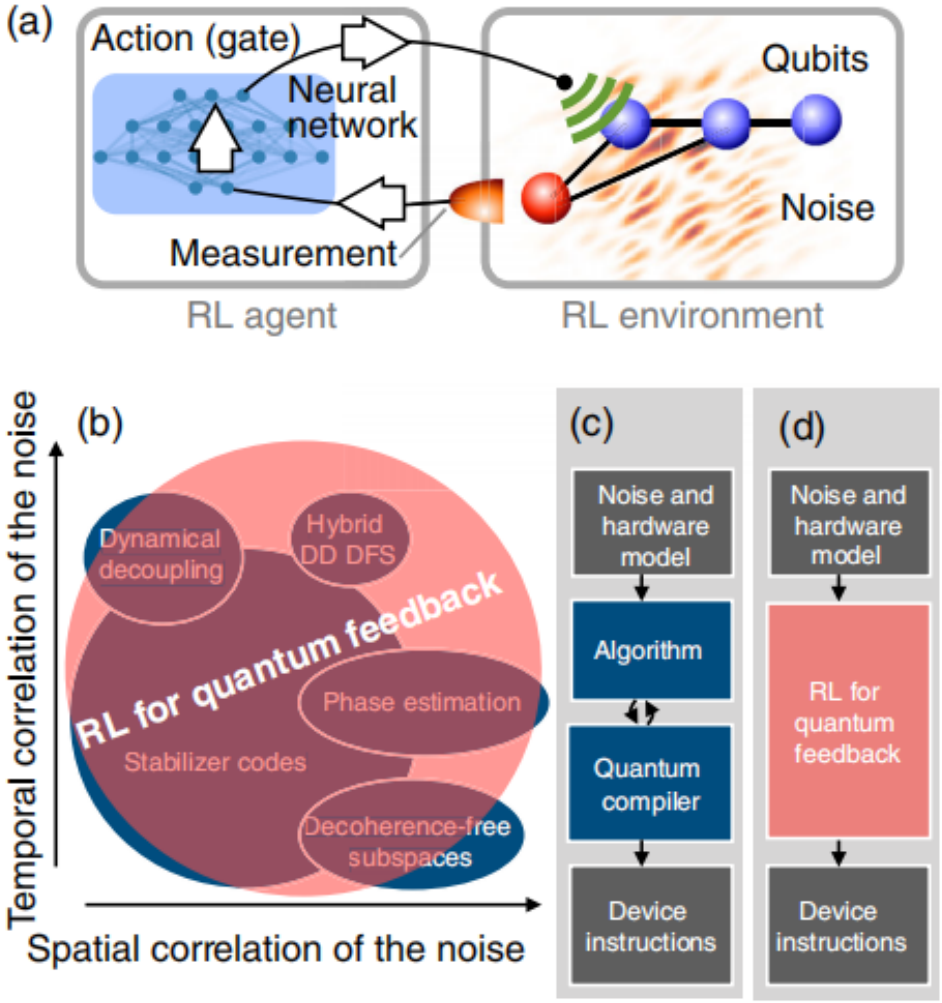

Figure 1: (a) shows how network-based RL works; (b) lists common quantum-error-correction(QEC) strategies; (c)(d) compare conventional procedure to the approach this paper use.

1. two-stage learning with a RL-trained network receiving maximum input acting as a teacher for a second network
2. a measure of the recoverable quantum information hidden inside a collection of qubits being used as a reward

Thoughts
to do list

Reinforcement Learning

The purpose of RL is to find an optimal set of actions (here, quantum gates and measurements) that an agent can perform in response to the changing state of an environment (here, the quantum memory). The objective is to maximize the expected return R, i.e., a sum of rewards.
The network is fed s_t as an input vector and outputs the probabilities $\pi_\theta(a_t \mid s_t)$. The expected return can then be maximized by applying the policy gradient RL update rule:

$$\delta\theta_j = \eta \frac{\partial \mathbb{E}[R]}{\partial \theta_j} = \eta \mathbb{E}\left[R \sum_t \frac{\partial}{\partial \theta_j} ln \pi_\theta(a_t \mid s_t) \right], \tag{1}$$

with η the learning rate and $\mathbb{E}$ the expectation over all gate sequences and measurement outcomes. $\pi_\theta(a_t \mid s_t)$ is the probability to apply action a_t, given the state s_t of the RL environment. As we use a neural network to compute π_θ, the multidimensional parameter θ stands for all the networks weights and biases.
In practice, authors improve (2) that they employ a baseline, natural policy gradient, and entropy regularization. That is to see, they use following equation for learning gradient

$$g = \lambda_{pol} F^{-1} \mathbb{E}\left[\sum_t (R_t - b_t) \frac{\partial}{\partial \theta_j} ln \pi_\theta(a_t \mid s_t) \right] - \lambda_{entr} \mathbb{E}\left[\sum_a \frac{\partial}{\partial \theta_j} (\pi_\theta(a \mid s) ln \pi_\theta(a \mid s) \right], \tag{2}$$

where R_t is the (discounted) return [compare protection reward and recovery reward]. This return is corrected by an (explicitly time-dependent) baseline b_t, which we choose as the exponentially decaying average of R_t; i.e. for the training update in epoch N, we use $b_t = (1-k)\Sigma_{n=0}^{N-1} k^n \bar{R}_t^{(N-1-n)}$, where k is the baseline discount rate, and $\bar{R}_t^{(n)}$ is the mean return at time step t in epoch n. We compute the natural gradient by multiplying F^{-1}, the (Moore-Penrose) inverse of the Fisher information matrix $F = \mathbb{E}\left\{ [(\partial/\partial\theta) ln \pi_\theta(a \mid s)] [(\partial/\partial\theta) ln \pi_\theta(a \mid s)]^T \right\}$. The second term is entropy regularization; we use it only to train the state-aware network. As update rule, we use adaptive moment estimation (Adam) (see following subsection) without bias correction.

Protection Reward and Recovery Reward

The goal of the protection reward is to maximize RQ at the end of the simulation. Based on this reward, we choose the return (the function of the reward sequence used to compute the policy gradient) as

$$R_t = (1-\gamma)\left(\Sigma_{k=0}^{T-t-1} \gamma^k r_{t+k}^{(l)} \right) + r_t^{(2)}, \tag{3}$$

where γ is the return discount rate. r_t is a suitable (immediate) reward, which is given by

$$r_t = \begin{cases} \underbrace{1 + \frac{\bar{R}_Q(t+\Delta t) - R_Q(t)}{2\Delta t / T_{single}}}_{=:r_t^{(1)}} & \underbrace{+0}_{=:r_t^{(2)}} & if\ \bar{R}_Q(t+\Delta t) > 0, \\ 0 & -P & if\ R_Q(t) \neq 0 \wedge R_Q(t+\Delta t) = 0, \\ 0 & +0 & if\ R_Q(t) = 0 \end{cases} \tag{4}$$

to do