OpenQML - Technical manuscript

Xanadu
(Dated: April 27, 2018)

Compilation of formulas and conventions for the OpenQML framework.

I. BASIC BUILDING BLOCKS

A. Variational circuit, layers and gates

We are given a **variational circiut**

$$U(\theta[,x]) : \rho_0 \to \mathrm{tr}\{\rho\hat{O}_i\} := E_i.$$

The circuit takes a ground or vacuum state $\rho_0 = |0\rangle\langle 0|$ and maps it to the expectation value of an operator $\hat{O}_i$ from a set of operators $\mathcal{O} = \{\hat{O}_1 \cdots \hat{O}_{N_O}\}$ with respect to the final state ρ of the circuit. These operators correspond to measurements that are easy to make on the device, for example when $\hat{O}$ is the Pauli operator σ_z that corresponds to a computational basis measurement, or the quadrature operator $\hat{x}$ corresponding to homodyne detection in CV systems. The circuit can also potentially take an input x, which we treat as a placeholder that can be replaced with values during training.

The variational circuit can be composed of **layers**

$$U(\theta) = \mathcal{L}_{N_L} \cdots \mathcal{L}_1$$

which repeat a certain architecture. A layer $\mathcal{L}_l$, $l = 1, ..., N_L$, consists of a series of **gates**

$$\mathcal{L}_l = \mathcal{G}^l_{N_G}(\theta^l_{N_G}) \cdots \mathcal{G}^l_1(\theta^l_1).$$

Each gate is parametrised by a set of parameters θ^i_j, $i = 1, ..., N_L$, $j = 1, ..., N_G$, which can be empty if the gate is not parametrised.

We propose to use the following **elementary gate sets**:

a. Qubit architectures Three rotations

$$R_z(\alpha) = \mathrm{e}^{-\alpha\sigma_x/2} = \begin{pmatrix} \cos\frac{\alpha}{2} & -i\sin\frac{\alpha}{2} \\ -i\sin\frac{\alpha}{2} & \cos\frac{\alpha}{2} \end{pmatrix}$$
$$R_z(\alpha) = \mathrm{e}^{-\alpha\sigma_y/2} = \begin{pmatrix} \cos\frac{\alpha}{2} & -\sin\frac{\alpha}{2} \\ \sin\frac{\alpha}{2} & \cos\frac{\alpha}{2} \end{pmatrix}$$
$$R_z(\alpha) = \mathrm{e}^{-\alpha\sigma_z/2} = \begin{pmatrix} \mathrm{e}^{-i\frac{\alpha}{2}} & 0 \\ 0 & \mathrm{e}^{i\frac{\alpha}{2}} \end{pmatrix}$$

and the CNOT gate.

b. CV architectures Displacement, Squeezing, Phase rotation, beam splitter, Kerr nonlinearity, Cubix phase gate. See SF conventions.

B. Cost function

The **cost function** depends on the expectation vales $\{E_i\}$,

$$C(\theta[,\mathcal{D}]) = g(\{E_i\}),$$

and potentially on a data set $\mathcal{D}$ which either consists of multiple inputs, $\mathcal{D} = \{x\}$, or input-output pairs, $\mathcal{D} = \{(x, y)\}$. We assume that both x and y are real finite vectors or scalars.

For the following we will also assume that the cost function is a sum of functions of the expectations,

$$C(\theta[,\mathcal{D}]) = \sum_i g_i(\{E_i\}), \tag{1}$$

which is true for the standard use-cases of variational circuits.

Examples of cost functions are:

a. Expectation cost for a variational eigensolver The cost function of a variational eigensolver is a weighed sum of expectation values,

$$C(\theta) = \sum_i h_i E_i(\theta).$$

Usually, these are expectations of Pauli operators, and their sum is the energy expectation,

$$\langle H\rangle = \sum_{i,\alpha} h^i_\alpha \langle\sigma^i_\alpha\rangle + \sum_{\substack{i,j \\ \alpha,\beta}} h^{ij}_{\alpha,\beta}\langle\sigma^i_\alpha\sigma^j_\beta\rangle + \ldots,$$

where i, j denote the qubits that the Paulis act on, and $\alpha, \beta = x, y, z$ sum over all different combinations of Pauli operators.

b. Maximum likelihood cost for a generative model If the goal is to increase the likelihood of measuring a basis state $|x\rangle$ that represents an input in the data set $\mathcal{D} = \{x\}$, the expectations we are interested in are

$$E_x(\theta) = \mathrm{tr}\{\rho(\theta)|x\rangle\langle x|\},$$

and we can use maximum likelihood to define

$$C(\theta, \mathcal{D}) = \sum_{x\in\mathcal{D}} \log E_x(\theta).$$

Note that in this unsupervised learning task the data enters the cost function directly, and the circuit is data-independent.

c. Square loss cost for supervised learning If we interpret the expectation $E_x(\theta) = \mathrm{tr}\{\rho(\theta, x)\ \sigma_z\}$ of the computational basis state of a designated qubit for an input x as the prediction of a quantum classifier, the square loss cost function reads

$$C(\theta, \mathcal{D}) = \sum_{(x,y)\in\mathcal{D}} |E_x(\theta) - y|^2.$$

II. OPTIMIZATION

OpenQML deals with the optimization of the variational circuit with regards to the cost. We always minimize the cost. Our core method is (stochastic) gradient descent.

A. Gradient descent step

In every **step** of the optimization algorithm, the parameters are updated according to the following simple loop

1: **procedure** GRADIENT DESCENT STEP
2: [sample a batch $\mathcal{D}$ from the training data]
3: **for** $\mu \in \theta$ **do**
4: $\mu^{(t+1)} = \mu^{(t)} - \eta^{(t)}\partial_\mu C(\theta[,\mathcal{D}]) + S$

The learning rate $\eta^{(t)}$ can be adapted in each step, either depending on the gradient or on the step number. S is a potential additional term that can add a momentum to the update. The gradient $\partial_\mu C(\theta[,\mathcal{D}])$ has to be evaluated by automatic differentiation of the cost function, which is computed hybridly by the quantum device or simulator and a classical computer. We therefore need to define a hybrid automatic differentiation scheme.

B. Computing the gradient

For the cost of Eq. (1) and neglecting the dependency on inputs x for now, we have

$$\partial_\mu C(\theta[,\mathcal{D}]) = \sum_i \frac{dg(\{E_i(\theta)\})}{dE_i(\theta)}\partial_\mu E_i(\theta).$$

The first gradient $\frac{dg(\{E_i(\theta)\})}{dE_i(\theta)}$ is the derivative of the classically computed part of the cost. We evaluate it using standard computational tools for automatic differentiation. The second gradient $\partial_\mu E_i(\theta)$ is the derivative of the part that is computed by the quantum device. We need something like "quantum automatic differentiation" to compute these gradients.

Formally, we have

$$\begin{aligned}\partial_\mu E_i(\theta) &= \partial_\mu \mathrm{tr}\{\rho(\theta)\ \hat{O}_i\} \\ &= \mathrm{tr}\{(\partial_\mu \rho(\theta))\ \hat{O}_i\},\end{aligned}$$

and with $\rho(\theta) = U(\theta)\rho_0 U^\dagger(\theta)$ and the product rule of differentiation, we get

$$\partial_\mu \rho(\theta) = (\partial_\mu U(\theta))\,\rho_0 U^\dagger(\theta) + U(\theta)\rho_0\left(\partial_\mu U^\dagger(\theta)\right).$$

This expression contains the **circuit derivative**, $\partial_\mu U(\theta)$. Assume that only the lth layer depends on circuit parameter μ, then

$$\partial_\mu U(\theta) = \mathcal{L}_1^\dagger \cdots (\partial_\mu \mathcal{L}_l^\dagger)\cdots \mathcal{L}_D^\dagger.$$

We call the expression $\partial_\mu \mathcal{L}_d^\dagger$ the **layer derivatives**. The layer is decomposed into different gates from our universal gate set. Let μ be the parameter of the rth gate $\mathcal{G}_r(\mu)$ (where $\mathcal{G}$ can also depend on other parameters). The derivative of a layer is then the original layer, but instead of $\mathcal{G}_r$ we use $\partial_\mu \mathcal{G}_r(\mu)$. This expression is in general not a quantum gate, and in particular not necessarily a member of our elementary gate set. We therefore have to decompose it into a weighed sum of "allowed gates" $A_k(\mu)$,

$$\partial_\mu \mathcal{G}(\mu) = \sum_k a_k A_k(\mu), \tag{2}$$

with complex coefficients a_k. This is always possible, because any operator can be represented as a sum of unitaries, and if our elementary gate set is universal for quantum computing, it can represent any unitary. The derivative circuit hence becomes

$$\partial_\mu U(\theta) = \sum_k a_k U_{A_k}(\theta) \tag{3}$$

$$= \sum_k a_k \mathcal{G}_1^1 \cdots A_k(\mu)\cdots \mathcal{G}_{N_G}^{N_L}, \tag{4}$$

where $U_{A_k}(\theta)$ is the original circuit but with the gate $\mathcal{G}_r^l(\mu)$ replaced by $A_k(\mu)$.

An example is the special case for Equation (2) when $\mathcal{G}(\mu) = \mathrm{e}^{i\mu H}$ for a generator H. Then, formally,

$$\partial_\mu \mathcal{G}(\mu) = iH\mathrm{e}^{i\mu H}.$$

Since H might not be a unitary operator, we have to decompose it into unitaries H_k via $H = \sum_k a_k H_k$, so that

$$\partial_\mu \mathcal{G}(\mu) = \sum_k i\,\underbrace{H_k \mathrm{e}^{i\mu H}}_{A_k}.$$

Summarizing the above, we get the formulae

$$\begin{aligned}\partial_\mu E_i(\theta) &= \mathrm{tr}\{\sum_k \left[a_k U_{A_k}(\theta)\rho_0 U^\dagger(\theta) + h.c.\right]\ \hat{O}_i\}, \\ &= \sum_k \mathrm{tr}\{\left[a_k U_{A_k}(\theta)\rho_0 U^\dagger(\theta) + h.c.\right]\ \hat{O}_i\} \\ &= \sum_k \mathrm{tr}\{a_k U_{A_k}(\theta)\rho_0 U^\dagger(\theta)\hat{O}_i + h.c.\}.\end{aligned}$$

Altogether, the quantum computer has to evaluate the terms

$$\mathrm{tr}\{U_{A_k}(\theta)\rho_0 U^\dagger(\theta)\hat{O}\},$$

and

$$\mathrm{tr}\{U(\theta)\rho_0 U^\dagger_{A_k}(\theta)\hat{O}\},$$

from which we can construct the derivative of the expectation classically. For this, we have to prepare a quantum state of the form

$$\mathcal{A}\rho_0\mathcal{B},$$

where $\mathcal{A},\mathcal{B}$ are the derivative and the original variational circuit.

Maria: Nathan, you mentioned that you know a trick of how we can implement this easily? So far I am very sceptical. I only know the following trick: Decompose $\hat{O}$ into a sum of unitaries B_s,

$$\hat{O} = \sum_s b_s B_s,$$

so that we have to implement

$$\mathrm{tr}\{U(\theta)\rho_0 U^\dagger_{A_k}(\theta)B_s\}.$$

Define $A = U(\theta)$ and $D = U^\dagger_{A_k}(\theta)B_s$, then prepare

$$\frac{1}{\sqrt{2}}(|0\rangle\otimes A\rho_0 A^\dagger + |1\rangle\otimes D\rho_0 D^\dagger),$$

and apply a Hadamard on the ancilla. The probability of measuring the ancilla in 0 is

$$p(0) = \frac{1}{2} + \frac{1}{2}\mathrm{tr}\{A\rho_0 A^\dagger D\rho_0 D^\dagger + D\rho_0 D^\dagger A\rho_0 A^\dagger\}$$