Variational Quantum Algorithms

Quantum Information And Computation

Giacomo Vittori (1811571)

January 15, 2024

SAPIENZA
UNIVERSITÀ DI ROMA

Table of Contents

NISQ Devices

Noisy Intermediate-Scale Quantum (NISQ) devices are the current state of the art quantum computers.

NISQ devices

- Quantum processor up to 1000 qubits
- Not advanced enough for quantum error correction
- Not large enough for quantum supremacy
- Noisy gates
- Prone to quantum decoherence

Any strategy to achieve quantum advantage with these devices must take into account

- Limited number of qubits
- Coherent and incoherent errors that limits depth

VQA: Basic Concepts

Accounting for the constraints on NISQ a strategy would require an optimization/learning based approach.

Variational Quantum Algorithms

VQAs[a] use parametrized quantum circuits to be run on a quantum computer and then outsource the parameter optimization to a classical optimizer.

[a] VQAs are the quantum analog of machine learning methods.

Advantages:

- keep the quantum circuit depth shallow
- mitigate noise
- versatility

Challenges:

- trainability
- accuracy
- efficiency

VQA: Basic Concepts

Consider a task one wishes to solve, this implies having access to:

- description of the problem
- training data (possibly)

General Framework

1. Encode the solution $\rightarrow$ Define a **Cost Function** [a]
2. Propose an **ansatz** $\rightarrow$ quantum operation depending on a set of parameters $\boldsymbol{\theta}$ that can be optimized
3. Train in a hybrid quantum-classical loop to solve the optimization task[b]

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta})$$

[a] The cost or its gradients is estimated using a quantum computer.

[b] To train $\boldsymbol{\theta}$ are used classical optimizer.

VQA: Cost Function

Cost Function

The cost maps trainable $\boldsymbol{\theta}$ to real numbers defining a hypersurface (*landscape*) where the optimizator navigate to find the global minimum.

$$C(\boldsymbol{\theta}) = \sum_k f_k \left(\mathrm{Tr}\left[O_k U(\boldsymbol{\theta}) \rho_k U^\dagger(\boldsymbol{\theta})\right]\right)$$

Requirements:

- The minimum should corresponds to the solution of the problem
- Efficiently computable with measurement on a quantum computer[1]
- Operationally meaningful
- Trainability

[1] But not efficiently computable on a classical computer.

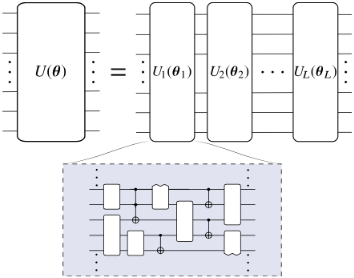
Ansatz

- Determine what $\boldsymbol{\theta}$s are and how to train them
- The structure depends on the task $\rightarrow$ *problem-inspired*
- The structure doesn't depend on the task $\rightarrow$ *problem-agnostic*

The parameters $\boldsymbol{\theta}$ are encoded in a unitary applied on the input states:

$$U(\boldsymbol{\theta}) = U_L(\boldsymbol{\theta}_L) \cdots U_2(\boldsymbol{\theta}) U_1(\boldsymbol{\theta}_1)$$

with $U_l(\boldsymbol{\theta}_l) = \prod_m e^{-i\theta_m H_m} W_m$



VQA: Gradients

To solve the optimization problem, under certain conditions there is an hardware-friendly protocol to evaluate the partial derivative of $C(\boldsymbol{\theta})$ with respect to θ_l referred as *parameter-shift rule* shifting θ_l by α.

Parameter-Shift rule

Consider $f_k(x) = x$, θ_l be the l-th element in $\boldsymbol{\theta}$ parametrizing $e^{i\theta_l \sigma_l}$, the parameter-shift rule states the equality holding for any real number α:

$$\frac{\partial C}{\partial \theta_l} = \sum_k \frac{1}{2\sin\alpha}\left(\mathrm{Tr}[O_k U^\dagger(\boldsymbol{\theta}_+)\rho_k U(\boldsymbol{\theta}_+)] - \mathrm{Tr}[O_k U^\dagger(\boldsymbol{\theta}_-)\rho_k U(\boldsymbol{\theta}_-)]\right)$$

with $\boldsymbol{\theta}_\pm = \boldsymbol{\theta} \pm \alpha \boldsymbol{e}_l$ and $\boldsymbol{e}_l$ being 1 as l-th element and 0 otherwise.

The accuracy is maximized at $\alpha = \frac{\pi}{4}$ (minimize $\frac{1}{2\sin\alpha}$). The observations relate to the fact that $C(\boldsymbol{\theta})$ can be expanded in a trigonometric series.

VQA Optimizer: Gradient Descent Methods

One of the most common approach is to make iterative steps in directions indicated by the gradient.

$$\downarrow$$

Stochastic Gradient Descent[2]

The true gradient is approximated by a gradient at a single sample:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}, x_i, y_i)$$

An alternative approach is *quantum natural GD* that works on a space with a metric tensor that encodes the sensitivity of the quantum state to parameters changes.

[2] E.g. Adam takes steps in the steepest descent direction in l_2 geometry adapting the size of the steps to allow more efficient and precise solution.

The **Barren Plateau** phenomenon in the cost function landscape is one of the main bottleneck of VQA

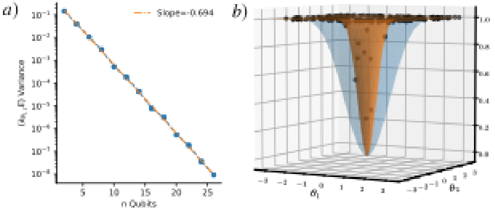


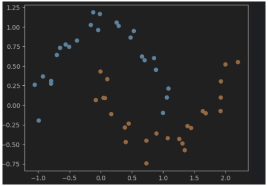
Figure: As the number of qubits increases the landscape is flatter and variance vanish exponentially.

Magnitude of the partial derivatives is exponentially vanishing with system size (flat landscape). It is needed an exponentially large precision to determine direction.

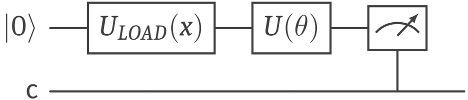
Table of Contents
2 QISKIT

Single Qubit Variational Classifier

Task: **Binary Classification** on *make moons* Scitkit-Learn dataset



The simplest operation:



- Loading can be trainable or not
- $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$ (± 1 for $|0\rangle, |1\rangle$)
- **Data Reuploading** solve the problem without more qubits repeating L times the building block[a]

[a] Single qubit means that $U \in SU(2)$, that can be parametrized by three real numbers

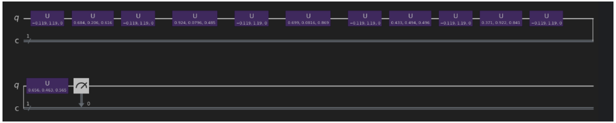
Implementation: Circuit

Qiskit is an open-source software to manipulating quantum circuits and running them on IBM quantum devices on classical simulators.

```python
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister

# data re-uploading classifier circuit
def circuit(params,x):
    L1 = int(params.size/3)
    qr = QuantumRegister(1, name="q")
    cr = ClassicalRegister(1, name='c')
    qc = QuantumCircuit(qr, cr)
    for k in range(L1):
        qc.u(x[0],x[1],0.0,qr[0])
        qc.u(params[3*k+0], params[3*k+1], params[3*k+2], qr[0]) # su(2) rotation
    qc.measure(qr, cr[0])
    return qc

L = 6 # number of layers
params = np.random.rand(L*3)
qc = circuit(params, X[1])
qc.draw('mpl')
```




```python
from qiskit import Aer, transpile, assemble
# try different simulators
# https://qiskit.org/documentation/tutorials/simulators/1_aer_provider.html
backend = Aer.get_backend("qasm_simulator")
NUM_SHOTS = 1000

t_qc = transpile(qc, backend)
t_qc.draw('mpl')
```

Executed at 2024.01.08 18:28:34 in 68ms

Implementation: Training

The Fidelity $F = |\langle y_i | \psi(x_i, \theta_i) \rangle|^2$ (the higher the better) suggests to use as a cost the **Infidelity**:

$$loss(x, y) = 1 - F \quad EmpRisk = \frac{1}{T} \sum_{i=1}^{T} loss(x_i, y_i)$$

Training: find $\arg\min_{\theta}(Risk)$ using a classical optimizator like COBYLA [3]

Prediction

[3] Constrained Optimization BY Linear Approximation (COBYLA) algorithm.

Run Prediction On A Real Hardware

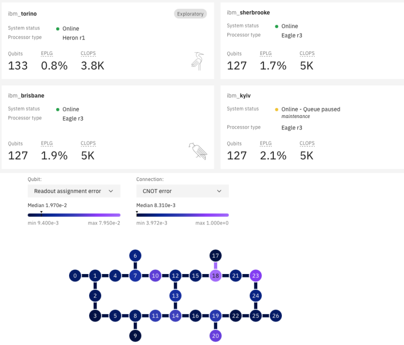
You can choose a quantum hardware to run you prediction from

https://quantum-computing.ibm.com/services?services=systems

```python
from qiskit_ibm_provider import IBMProvider

# load account
provider = IBMProvider()

# see https://quantum-computing.ibm.com/services?services=systems
#for a list of backends... we use armonk as we need a single qubit
backend = provider.get_backend('ibmq_mumbai')
```



Bibliography

- Cerezo, Marco, et al. ”Variational quantum algorithms.” Nature Reviews Physics 3.9 (2021): 625-644.
- Qiskit: An Open-source Framework for Quantum Computing, Doi: 10.5281/zenodo.2573505

Variational Quantum Algorithms *Thank you*

for listening!

Any questions?