Supplementary: qLEET - Visualizing Loss Landscapes, Expressibility, Entangling power and Training Trajectories for Parameterized Quantum Circuits

Utkarsh Azad* and Animesh Sinha†
Center for Computational Natural Sciences and Bioinformatics, International Institute of Information Technology, Hyderabad.
Center for Quantum Science and Technology, International Institute of Information Technology, Hyderabad.
(Dated: December 16, 2022)

S1. TUTORIAL: ENTAGLEMENT ABILITY ANALYSIS

In this section, we will learn how to calcualte expressibility of Parameterized Quantum Circuits (PQCs) using qLEET, which could thought of as traversing power of a PQC in the Hilbert space. We look at different parameterized states generated by the sampled ensemble of parameters for a given PQC. We then compare the resulting distribution of state fidelities ($\mathcal{F}$) generated by this sampled ensemble to that of the ensemble of Haar random states.

We currently support two expressibility measures - **Kullback–Leibler Divergence** and **Jensen–Shannon Divergence**

$$\text{Expressibility} = D_{\text{KL}}\Big(\hat{P}_{\text{PQC}}(\mathcal{F};\theta)\big|P_{\text{Haar}}(\mathcal{F})\Big)$$

$$\text{Expressibility} = D_{\sqrt{\text{JSD}}}\Big(\hat{P}_{\text{PQC}}(\mathcal{F};\theta)\big|P_{\text{Haar}}(\mathcal{F})\Big)$$

All circuit analysis using qleet begins with defining a parameterized quantum circuit using a library of choice, and then passing it into qleet's `CircuitDescriptor` interface.

```
params = [qiskit.circuit.Parameter(r"$\theta_1$")]

qiskit_circuit = qiskit.QuantumCircuit(1)
qiskit_circuit.h(0)
qiskit_circuit.rz(params[0], 0)
qiskit_descriptor = qleet.interface.circuit.CircuitDescriptor(
    circuit=qiskit_circuit, params=params, cost_function=None
)
```

The analyze the expressibility, we can use the corresponding analyzer. We can get the expressibility using either of the two supported measures.

```

qiskit_expressibility = qleet.analyzers.expressibility.Expressibility(
    qiskit_descriptor, samples=100
)
expr_jsd = qiskit_expressibility.expressibility("jsd")
print("JSD Expressibility:", expr_jsd)

expr_kld = qiskit_expressibility.expressibility("kld")
print("KLD Expressibility:", expr_kld)

plt_figure = qiskit_expressibility.plot()
```

We look at different parameterized states generated by the sampled ensemble of parameters for a given PQC. We then compare the resulting distribution of eigenvalues of the bipartite state generated by this sampled ensemble to that of the ensemble of eigenvalues of Haar random states.

* utkarsh.azad@research.iiit.ac.in; Corresponding Author
† animesh.sinha@research.iiit.ac.in

We currently support two measures to calculate entanglement spectrum divergence (ESD) - **Kullback–Leibler Divergence** and **Jensen–Shannon Divergence**

$$\text{ESD} = D_{\text{KL}}\Big(\hat{P}_{\text{PQC}}(H_{\text{ent}};\theta)\Big|P_{\text{Haar}}(H_{\text{ent}})\Big)$$

$$\text{ESD} = D_{\sqrt{\text{JSD}}}\Big(\hat{P}_{\text{PQC}}(H_{\text{ent}};\theta)\Big|P_{\text{Haar}}(H_{\text{ent}})\Big)$$

```
params = [
    qiskit.circuit.Parameter(r"$\theta_1$"),
    qiskit.circuit.Parameter(r"$\theta_2$")
]
qiskit_circuit = qiskit.QuantumCircuit(2)
qiskit_circuit.rx(params[0], 0)
qiskit_circuit.cx(0, 1)
qiskit_circuit.rx(params[1], 1)
qiskit_descriptor = qleet.interface.circuit.CircuitDescriptor(
    circuit=qiskit_circuit, params=params, cost_function=None
)
```

```
analyzer = (
    qleet.analyzers.entanglement.EntanglementCapability(
        qiskit_descriptor, samples=500
    )
)

entanglement_mw = analyzer.entanglement_capability("meyer-wallach")
print("Entanglement Capability (Meyer Wallach Measure):", entanglement_mw)

entanglement_scott = analyzer.entanglement_capability("scott")
print("Entanglement Capability (Scott Measure):", entanglement_scott)
```

In this section, we will plot the entanglement spectrum.

```
def ansatz(params, cparams=None):
    layers, num_qubits, depth = params.shape
    ansatz = qiskit.QuantumCircuit(num_qubits)
    for idx in range(layers):
        if idx:
            ansatz.barrier()
        for ind in range(num_qubits):
            ansatz.rx(params[idx][ind][0], ind)
            ansatz.rz(params[idx][ind][1], ind)
            ansatz.rx(params[idx][ind][2], ind)
        for ind in range(num_qubits-1):
            ansatz.cx(ind, ind+1)
    return ansatz
```

```
data = []
results = []
num_qubits = 12
for idx in range(1, 17):
    print(idx, end=' ')
    params = np.array([qiskit.circuit.Parameter(fr"$\theta_{idx}$")
                       for idx in range(idx*num_qubits*3)])
    qiskit_descriptor = qleet.CircuitDescriptor(
        circuit=ansatz(np.array(params).reshape((idx, num_qubits, 3))),
        params=params, cost_function=None
```

```
    )
    qiskit_entanglement_spectrum = \
        qleet.analyzers.entanglement_spectrum.EntanglementSpectrum(
            qiskit_descriptor, samples=100
        )
    pqc_esd, mean_eig = qiskit_entanglement_spectrum.entanglement_spectrum("jsd")
    results.append(pqc_esd)
    data.append(mean_eig)
data = np.array(data)

fig = qiskit_entanglement_spectrum.plot(data)
```

S2. LOSS LANDSCAPE AND TRAINING TRAJECTORY ANALYSIS

For this section of the tutorial, we shall be constructing our circuits in the **Cirq** library, which is also supported by our multi-backend analyzer. Using cirq, we define a parameterized quantum circuit, we define its parameters as sympy symbols, and we define a cost function as a Pauli measurement on the outputs of this circuits. All of this is passed into out `CircuitDescriptor` interface

```
graph = nx.gnm_random_graph(n=8, m=20)
qubits = cirq.GridQubit.rect(1, graph.number_of_nodes())
p = 5

params = sympy.symbols("q0:%d" % (2 * p))
qaoa_circuit = cirq.Circuit()
for qubit in qubits:
    qaoa_circuit.append(cirq.H(qubit))
for i in range(p):
    for edge in graph.edges():
        qaoa_circuit += cirq.CNOT(qubits[edge[0]], qubits[edge[1]])
        qaoa_circuit += cirq.rz(params[2 * i]).on(qubits[edge[1]])
        qaoa_circuit += cirq.CNOT(qubits[edge[0]], qubits[edge[1]])
    for j in range(len(qubits)):
        qaoa_circuit += cirq.rx(2 * params[2 * i + 1]).on(qubits[j])

qaoa_cost = cirq.PauliSum()
for edge in graph.edges():
    qaoa_cost += cirq.PauliString(1 / 2 * cirq.Z(qubits[edge[0]]) *
                                  cirq.Z(qubits[edge[1]]))

circuit = qleet.interface.circuit.CircuitDescriptor(
    qaoa_circuit, params, qaoa_cost)
solver = qleet.simulators.pqc_trainer.PQCSimulatedTrainer(circuit)
```

```
class MaxCutMetric(qleet.interface.metric_spec.MetricSpecifier):

    def __init__(self, graph):
        super().__init__("samples")
        self.graph = graph

    def from_samples_vector(self, samples_vector):
        return np.mean([nx.algorithms.cuts.cut_size(
            self.graph, np.where(cut)[0]) for cut in samples_vector])

    def from_density_matrix(self, density_matrix):
        raise NotImplementedError
```

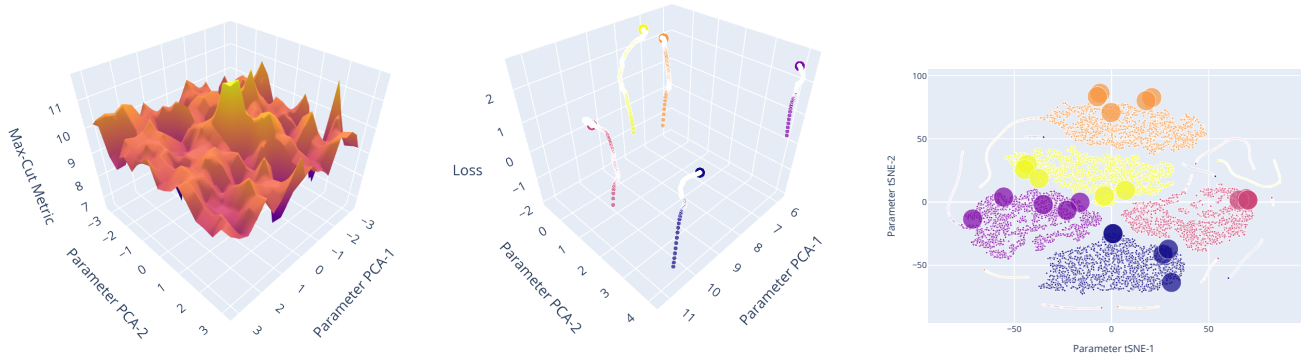
```
    def from_state_vector(self, state_vector):
        raise NotImplementedError

metric = MaxCutMetric(graph)
```

```
plot = qleet.analyzers.loss_landscape.LossLandscapePlotter(
    solver, metric, dim=2)
solver.train(n_samples=5000)
fig_loss_surface = plot.plot("surface", points=20)

trackers = qleet.interface.metas.AnalyzerList(
    qleet.analyzers.training_path.LossLandscapePathPlotter(plot),
    qleet.analyzers.training_path.OptimizationPathPlotter(mode="tSNE"),
)
for _i in range(5):
    solver.train(loggers=trackers, n_samples=5000)
    trackers.next()

fig_loss_traversal = trackers[0].plot()
fig_training_trace = trackers[1].plot()
```



(a) Metric Landscape (inverse of loss) around obtained optima

(b) PCA plot with loss for training trajectories of 5 runs

(c) 2-D tSNE of training trajectories from 5 runs

FIG. S1: Loss and Training Trajectory plots obtained on analyzing the circuit shown. Here, the analysis is shown for a circuit representing max-cut on a graph with 8 nodes and 20 edges.

S3. ENTANGLEMENT ANALYSIS FOR M_Z OPERATOR [1]

```
params = [qiskit.circuit.Parameter(r"$\theta_1$"),
          qiskit.circuit.Parameter(r"$\theta_2$"),
          qiskit.circuit.Parameter(r"$\theta_3$"),
          qiskit.circuit.Parameter(r"$\theta_4$")]

qiskit_circuit = qiskit.QuantumCircuit(4)
qiskit_circuit.rx(params[0], 0)
qiskit_circuit.rz(params[1], 0)
qiskit_circuit.rx(params[2], 2)
qiskit_circuit.rz(params[3], 2)
qiskit_circuit.cx(0, 1)
qiskit_circuit.cx(2, 3)
```

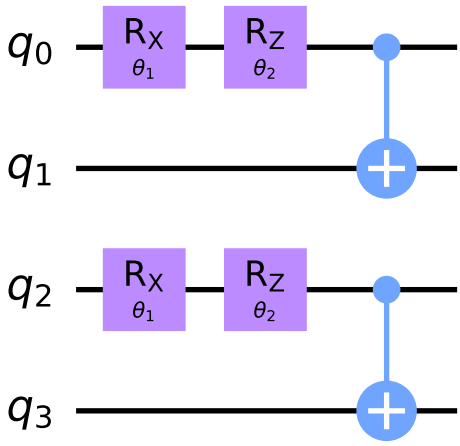


FIG. S2: M_Z operator used in the quantum computing model based on entanglement degree allows to differentiate between the non-orthogonal states of the form $e_1|0\rangle + e_2|1\rangle$, with arbitrary accuracy [1–4].

```
qiskit_descriptor = qleet.interface.circuit.CircuitDescriptor(
    circuit=qiskit_circuit, params=params, cost_function=None
)

qiskit_entg_capability = (
    qleet.analyzers.entanglement.EntanglementCapability(
        qiskit_descriptor, samples=1000
    )
)

entanglement_mw = qiskit_entg_capability.entanglement_capability("meyer-Wallach")
# >>> entanglement_mw = 0.5010648894421558

entanglement_scott = qiskit_entg_capability.entanglement_capability("scott")
# >>> entanglement_scott = array([0.4979689 , 0.38654991])
```

S4. QUANTUM CIRCUITS FROM THE EXPERIMENTS

A. Loss Landscape and Training Trajectories (Fig. S3 → Fig. 3)

B. Expressibility (Fig. S4 → Fig. 5)

C. Entangling Capability (Fig. S5 → Fig. 6)

D. Entanglement Spectrum (Fig. S6 → Fig. 7)

---

[1] M. Zidan, *A novel quantum computing model based on entanglement degree*, Modern Physics Letters B **34**, 2050401 (2020).
[2] S. Khan, M. Nauman, S. A. Alsaif, T. A. Syed, and H. A. Eleraky, *A quantum algorithm for evaluating the hamming distance*, Computers, Materials & Continua **71**, 1065 (2022).
[3] C. S. Punla and R. C. Farro, *Analysis of the quantum algorithm based on entanglement measure for classifying boolean multivariate function into novel hidden classes: Revisited*, Applied Mathematics & Information Sciences **15**, 643 (2021).
[4] B. Panda, N. K. Tripathy, S. Sahu, B. K. Behera, and W. E. Elhady, *Controlling remote robots based on zidan's quantum computing model*, Computers, Materials & Continua **73**, 6225 (2022).

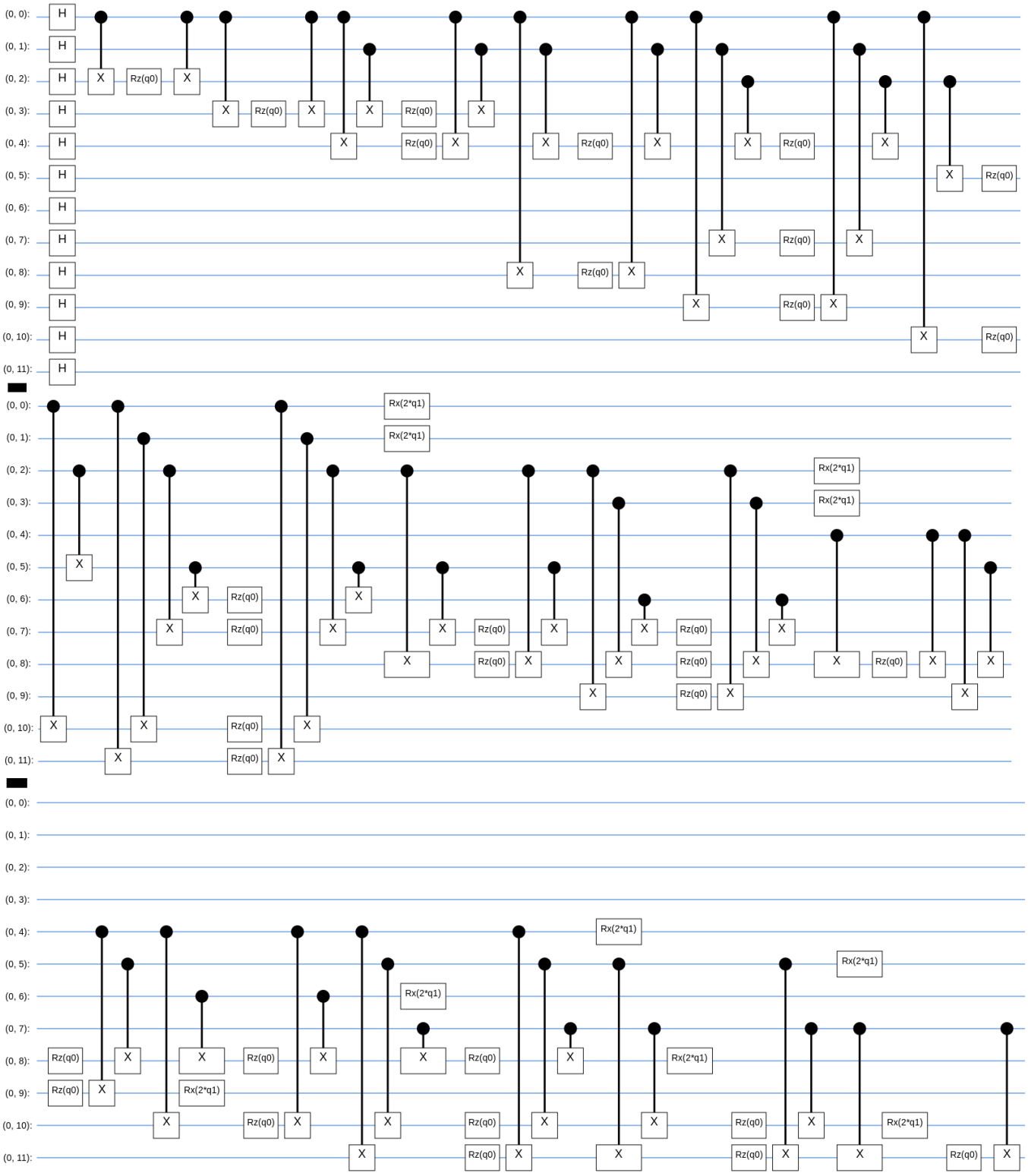


FIG. S3: QAOA circuit for p=1. This circuit (except the first Hadamard layer) will be repeated k times for $p = k$.

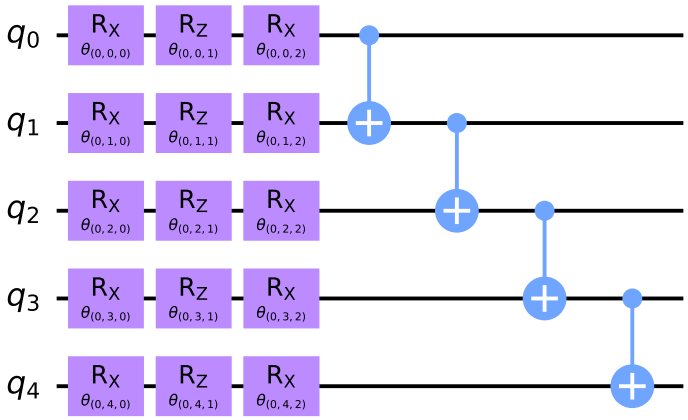


FIG. S4: Parameterized quantum circuit $U(\vec{\theta}) = \prod_1^L \left(\bigotimes_{i=1}^5 R_x(\theta_i^1) R_z(\theta_i^2) R_x(\theta_i^3) \dots \bigotimes_{i<j} CX(i,j) \right)$

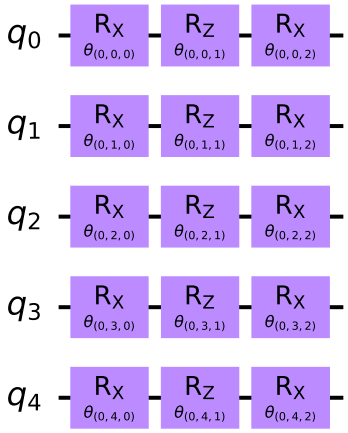


FIG. S5: Parameterized quantum circuit $U(\vec{\theta}) = \bigotimes_{i=1}^5 R_x(\theta_i^1) R_z(\theta_i^2) R_x(\theta_i^3)$

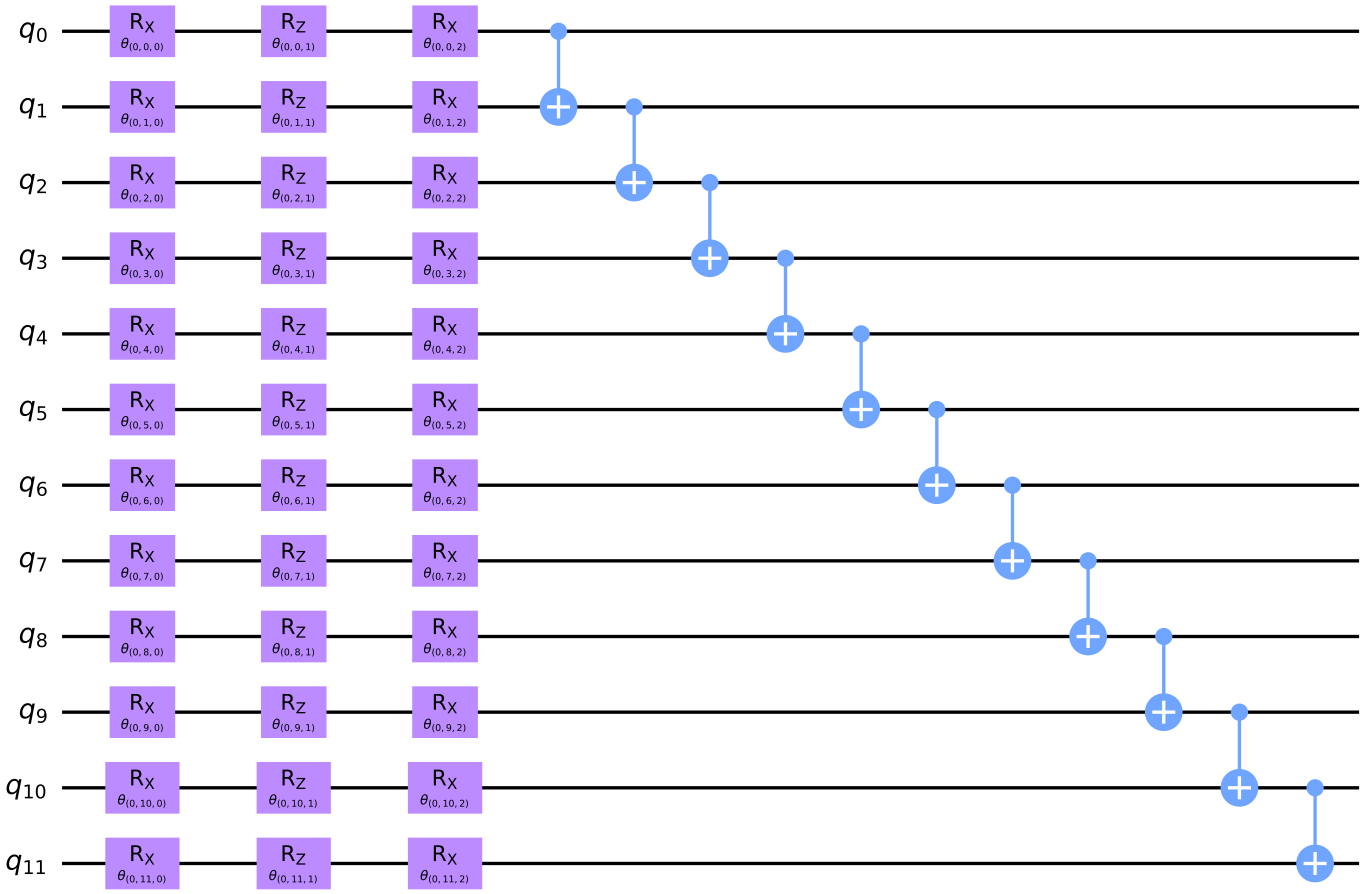


FIG. S6: Parameterized quantum circuit $U(\vec{\theta}) = \prod_1^L \left(\bigotimes_{i=1}^{12} R_x(\theta_i^1) R_z(\theta_i^2) R_x(\theta_i^3) \ldots \bigotimes_{i=1}^{11} CX(i, i+1) \right)$