TorchQuantum

Release 0.1

Hanrui Wang

Jul 19, 2022

API

CHAPTER

ONE

TORCHQUANTUM.FUNCTIONAL

1.1 Functions

apply_unitary_einsum(state, mat, wires)	Apply the unitary to the statevector using torch.einsum method.
apply_unitary_bmm(state, mat, wires)	Apply the unitary to the statevector using torch.bmm method.
gate_wrapper(name, mat, method, q_device, wires)	Perform the phaseshift gate.
reset(q_device, wires[, inverse])	
rx_matrix(params)	Compute unitary matrix for rx gate.
ry_matrix(params)	Compute unitary matrix for ry gate.
rz_matrix(params)	Compute unitary matrix for rz gate.
phaseshift_matrix(params)	
rot_matrix(params)	Compute unitary matrix for rot gate.
multirz_eigvals(params, n_wires)	Compute eigenvalue for multiqubit RZ gate.
multirz_matrix(params, n_wires)	Compute unitary matrix for multiqubit RZ gate.
rxx_matrix(params)	Compute unitary matrix for RXX gate.
ryy_matrix(params)	Compute unitary matrix for RYY gate.
rzz_matrix(params)	Compute unitary matrix for RZZ gate.
rzx_matrix(params)	Compute unitary matrix for RZX gate.
crx_matrix(params)	Compute unitary matrix for CRX gate.
cry_matrix(params)	Compute unitary matrix for CRY gate.
crz_matrix(params)	Compute unitary matrix for CRZ gate.
crot_matrix(params)	Compute unitary matrix for CRot gate.
u1_matrix(params)	Compute unitary matrix for U1 gate.
cu1_matrix(params)	Compute unitary matrix for CU1 gate.
u2_matrix(params)	Compute unitary matrix for U2 gate.
cu2_matrix(params)	Compute unitary matrix for CU2 gate.
u3_matrix(params)	Compute unitary matrix for U3 gate.
cu3_matrix(params)	Compute unitary matrix for CU3 gate.
qubitunitary_matrix(params)	Compute unitary matrix for Qubitunitary gate.
qubitunitaryfast_matrix(params)	Compute unitary matrix for Qubitunitary fast gate.
qubitunitarystrict_matrix(params)	Compute unitary matrix for Qubitunitary strict gate.
multicnot_matrix(n_wires)	Compute unitary matrix for Multi qubit CNOT gate.
multixcnot_matrix(n_wires)	Compute unitary matrix for Multi qubit XCNOT gate.
hadamard(q_device, wires[, params, n_wires, ...])	Perform the hadamard gate.

continues on next page

Table 1 – continued from previous page

shadamard(q_device, wires[, params, ...])	Perform the shadamard gate.
paulix(q_device, wires[, params, n_wires, ...])	Perform the Pauli X gate.
pauliy(q_device, wires[, params, n_wires, ...])	Perform the Pauli Y gate.
pauliz(q_device, wires[, params, n_wires, ...])	Perform the Pauli Z gate.
i(q_device, wires[, params, n_wires, ...])	Perform the I gate.
s(q_device, wires[, params, n_wires, ...])	Perform the s gate.
t(q_device, wires[, params, n_wires, ...])	Perform the t gate.
sx(q_device, wires[, params, n_wires, ...])	Perform the sx gate.
cnot(q_device, wires[, params, n_wires, ...])	Perform the cnot gate.
cz(q_device, wires[, params, n_wires, ...])	Perform the cz gate.
cy(q_device, wires[, params, n_wires, ...])	Perform the cy gate.
rx(q_device, wires[, params, n_wires, ...])	Perform the rx gate.
ry(q_device, wires[, params, n_wires, ...])	Perform the ry gate.
rz(q_device, wires[, params, n_wires, ...])	Perform the rz gate.
rxx(q_device, wires[, params, n_wires, ...])	Perform the rxx gate.
ryy(q_device, wires[, params, n_wires, ...])	Perform the ryy gate.
rzz(q_device, wires[, params, n_wires, ...])	Perform the rzz gate.
zz(q_device, wires[, params, n_wires, ...])	Perform the rzz gate.
rzx(q_device, wires[, params, n_wires, ...])	Perform the rzx gate.
zx(q_device, wires[, params, n_wires, ...])	Perform the rzx gate.
swap(q_device, wires[, params, n_wires, ...])	Perform the swap gate.
sswap(q_device, wires[, params, n_wires, ...])	Perform the sswap gate.
cswap(q_device, wires[, params, n_wires, ...])	Perform the cswap gate.
toffoli(q_device, wires[, params, n_wires, ...])	Perform the toffoli gate.
phaseshift(q_device, wires[, params, ...])	Perform the phaseshift gate.
p(q_device, wires[, params, n_wires, ...])	Perform the phaseshift gate.
cp(q_device, wires[, params, n_wires, ...])	Perform the cu1 gate.
rot(q_device, wires[, params, n_wires, ...])	Perform the rot gate.
multirz(q_device, wires[, params, n_wires, ...])	Perform the multi qubit RZ gate.
crx(q_device, wires[, params, n_wires, ...])	Perform the crx gate.
cry(q_device, wires[, params, n_wires, ...])	Perform the cry gate.
crz(q_device, wires[, params, n_wires, ...])	Perform the crz gate.
crot(q_device, wires[, params, n_wires, ...])	Perform the crot gate.
u1(q_device, wires[, params, n_wires, ...])	Perform the u1 gate.
u2(q_device, wires[, params, n_wires, ...])	Perform the u2 gate.
u3(q_device, wires[, params, n_wires, ...])	Perform the u3 gate.
u(q_device, wires[, params, n_wires, ...])	Perform the u3 gate.
cu1(q_device, wires[, params, n_wires, ...])	Perform the cu1 gate.
cu2(q_device, wires[, params, n_wires, ...])	Perform the cu2 gate.
cu3(q_device, wires[, params, n_wires, ...])	Perform the cu3 gate.
cu(q_device, wires[, params, n_wires, ...])	Perform the cu3 gate.
qubitunitary(q_device, wires[, params, ...])	Perform the qubitunitary gate.
qubitunitaryfast(q_device, wires[, params, ...])	Perform the qubitunitaryfast gate.
qubitunitarystrict(q_device, wires[, ...])	Perform the qubitunitarystrict = gate.
multicnot(q_device, wires[, params, ...])	Perform the multi qubit cnot gate.
multixcnot(q_device, wires[, params, ...])	Perform the multi qubit xcnot gate.
x(q_device, wires[, params, n_wires, ...])	Perform the Pauli X gate.
y(q_device, wires[, params, n_wires, ...])	Perform the Pauli Y gate.
z(q_device, wires[, params, n_wires, ...])	Perform the Pauli Z gate.
zz(q_device, wires[, params, n_wires, ...])	Perform the rzz gate.

continues on next page

Table 1 – continued from previous page

cx(q_device, wires[, params, n_wires, ...])	Perform the cnot gate.
ccnot(q_device, wires[, params, n_wires, ...])	Perform the toffoli gate.
ccx(q_device, wires[, params, n_wires, ...])	Perform the toffoli gate.
reset(q_device, wires[, inverse])	

1.1.1 apply_unitary_einsum

functional.**apply_unitary_einsum**(*mat*, *wires*)

Apply the unitary to the statevector using torch.einsum method.

Parameters

- **state** (*torch.Tensor*) – The statevector.
- **mat** (*torch.Tensor*) – The unitary matrix of the operation.
- **wires** (*int or List[int]*) – Which qubit the operation is applied to.

Returns
The new statevector.

Return type
torch.Tensor

1.1.2 apply_unitary_bmm

functional.**apply_unitary_bmm**(*mat*, *wires*)

Apply the unitary to the statevector using torch.bmm method.

Parameters

- **state** (*torch.Tensor*) – The statevector.
- **mat** (*torch.Tensor*) – The unitary matrix of the operation.
- **wires** (*int or List[int]*) – Which qubit the operation is applied to.

Returns
The new statevector.

Return type
torch.Tensor

1.1.3 gate_wrapper

functional.**gate_wrapper**(*mat*, *method*, *q_device: QuantumDevice*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*)

Perform the phaseshift gate.

Parameters

- **name** (*str*) – The name of the operation.
- **mat** (*torch.Tensor*) – The unitary matrix of the gate.

- **method** (*str*) – 'bmm' or 'einsum' to compute matrix vector multiplication.
- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

1.1.4 reset

functional.**reset**(*wires*, *inverse=False*)

1.1.5 rx_matrix

functional.**rx_matrix**() → Tensor

Compute unitary matrix for rx gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.6 ry_matrix

functional.**ry_matrix**() → Tensor

Compute unitary matrix for ry gate.

Parameters
params – The rotation angle.

Returns
The computed unitary matrix.

1.1.7 rz_matrix

functional.**rz_matrix**() → Tensor

Compute unitary matrix for rz gate.

Parameters
params – The rotation angle.

Returns
The computed unitary matrix.

1.1.8 phaseshift_matrix

functional.**phaseshift_matrix**()

1.1.9 rot_matrix

functional.**rot_matrix**()

Compute unitary matrix for rot gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.10 multirz_eigvals

functional.**multirz_eigvals**(*n_wires*)

Compute eigenvalue for multiqubit RZ gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed eigenvalues.

Return type
torch.Tensor

1.1.11 multirz_matrix

functional.**multirz_matrix**(*n_wires*)

Compute unitary matrix for multiqubit RZ gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.12 rxx_matrix

functional.**rxx_matrix**()

Compute unitary matrix for RXX gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.13 ryy_matrix

functional.**ryy_matrix**()

Compute unitary matrix for RYY gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.14 rzz_matrix

functional.**rzz_matrix**()

Compute unitary matrix for RZZ gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.15 rzx_matrix

functional.**rzx_matrix**()

Compute unitary matrix for RZX gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.16 crx_matrix

functional.**crx_matrix**()

Compute unitary matrix for CRX gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.17 cry_matrix

functional.**cry_matrix**()

Compute unitary matrix for CRY gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.18 crz_matrix

functional.**crz_matrix**()

Compute unitary matrix for CRZ gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.19 crot_matrix

functional.**crot_matrix**()

Compute unitary matrix for CRot gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.20 u1_matrix

functional.**u1_matrix**()

Compute unitary matrix for U1 gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.21 cu1_matrix

functional.**cu1_matrix**()

Compute unitary matrix for CU1 gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.22 u2_matrix

functional.**u2_matrix**()

Compute unitary matrix for U2 gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.23 cu2_matrix

functional.**cu2_matrix**()

Compute unitary matrix for CU2 gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.24 u3_matrix

functional.**u3_matrix**()

Compute unitary matrix for U3 gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.25 cu3_matrix

functional.**cu3_matrix**()

Compute unitary matrix for CU3 gate.

Parameters
params (*torch.Tensor*) – The rotation angle.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.26 qubitunitary_matrix

functional.**qubitunitary_matrix**()

Compute unitary matrix for Qubitunitary gate.

Parameters
params (*torch.Tensor*) – The unitary matrix.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.27 qubitunitaryfast_matrix

functional.**qubitunitaryfast_matrix**()

Compute unitary matrix for Qubitunitary fast gate.

Parameters
params (*torch.Tensor*) – The unitary matrix.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.28 qubitunitarystrict_matrix

functional.**qubitunitarystrict_matrix**()

Compute unitary matrix for Qubitunitary strict gate.
Strictly be the unitary.

Parameters
params (*torch.Tensor*) – The unitary matrix.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.29 multicnot_matrix

functional.**multicnot_matrix**()

Compute unitary matrix for Multi qubit CNOT gate.

Parameters
n_wires (*int*) – The number of wires.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.30 multixcnot_matrix

functional.**multixcnot_matrix**()

Compute unitary matrix for Multi qubit XCNOT gate.

Parameters
params (*torch.Tensor*) – The unitary matrix.

Returns
The computed unitary matrix.

Return type
torch.Tensor

1.1.31 hadamard

functional.**hadamard**(*wires: Union[List[int], int]*, *params: Optional[Tensor] = None*, *n_wires: Optional[int] = None*, *static: bool = False*, *parent_graph=None*, *inverse: bool = False*, *comp_method: str = 'bmm'*)

Perform the hadamard gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

1.1.32 shadamard

functional.**shadamard**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the shadamard gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

1.1.33 paulix

functional.**paulix**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the Pauli X gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.34 pauliy

functional.**pauliy**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the Pauli Y gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.35 pauliz

functional.**pauliz**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the Pauli Z gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.36 i

functional.**i**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the I gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.37 s

functional.s(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the s gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.38 t

functional.t(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the t gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.39 sx

functional.**sx**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the sx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.40 cnot

functional.**cnot**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cnot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.41 cz

functional.**cz**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cz gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.42 cy

functional.**cy**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cy gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.43 rx

functional.**rx**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.44 ry

functional.**ry**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the ry gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.45 rz

functional.**rz**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rz gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.46 rxx

functional.**rxx**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rxx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.47 ryy

functional.**ryy**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the ryy gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.48 rzz

functional.**rzz**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rzz gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.49 zz

functional.**zz**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rzz gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.50 rzx

functional.**rzx**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rzx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.51 zx

functional.**zx**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rzx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.52 swap

functional.**swap**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the swap gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.53 sswap

functional.**sswap**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the sswap gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.54 cswap

functional.**cswap**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cswap gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.55 toffoli

functional.**toffoli**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the toffoli gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.56 phaseshift

functional.**phaseshift**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the phaseshift gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.57 p

functional.**p**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the phaseshift gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.58 cp

functional.**cp**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cu1 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.59 rot

functional.**rot**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.60 multirz

functional.**multirz**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the multi qubit RZ gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.61 crx

functional.**crx**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the crx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.62 cry

functional.**cry**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cry gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.63 crz

functional.**crz**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the crz gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.64 crot

functional.**crot**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the crot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.65 u1

functional.**u1**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the u1 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.66 u2

functional.**u2**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the u2 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.67 u3

functional.**u3**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the u3 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.68 u

functional.**u**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the u3 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.69 cu1

functional.**cu1**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cu1 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.70 cu2

functional.**cu2**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cu2 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.71 cu3

functional.**cu3**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cu3 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.72 cu

functional.**cu**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cu3 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.73 qubitunitary

functional.**qubitunitary**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the qubitunitary gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.74 qubitunaryfast

functional.**qubitunaryfast**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the qubitunaryfast gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.75 qubitunitarystrict

functional.**qubitunitarystrict**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the qubitunitarystrict = gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.76 multicnot

functional.**multicnot**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the multi qubit cnot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.77 multixcnot

functional.**multixcnot**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the multi qubit xcnot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.78 x

functional.**x**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the Pauli X gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.79 y

functional.y(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the Pauli Y gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.80 z

functional.z(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the Pauli Z gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.81 cx

functional.**cx**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cnot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.82 ccnot

functional.**ccnot**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the toffoli gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

1.1.83 ccx

functional.**ccx**(*wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the toffoli gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

CHAPTER

TWO

TORCHQUANTUM.OPERATORS

2.1 Classes

WiresEnum(value)	Integer enumeration class to represent the number of wires an operation acts on.
NParamsEnum(value)	Integer enumeration class to represent the number of wires an operation acts on
AllWires	An enumeration which represents all wires in the subsystem.
AnyWires	An enumeration which represents any wires in the subsystem.
Operator([has_params, trainable, ...])	The class for quantum operators.
Observable([has_params, trainable, ...])	Class for Observables.
Operation([has_params, trainable, ...])	_summary_
DiagonalOperation([has_params, trainable, ...])	Class for Diagonal Operation.
Hadamard([has_params, trainable, ...])	Class for Hadamard Gate.
SHadamard([has_params, trainable, ...])	Class for SHadamard Gate.
PauliX([has_params, trainable, init_params, ...])	Class for Pauli X Gate.
PauliY([has_params, trainable, init_params, ...])	Class for Pauli Y Gate.
PauliZ([has_params, trainable, init_params, ...])	Class for Pauli Z Gate.
I([has_params, trainable, init_params, ...])	Class for Identity Gate.
S([has_params, trainable, init_params, ...])	Class for S Gate.
T([has_params, trainable, init_params, ...])	Class for T Gate.
SX([has_params, trainable, init_params, ...])	Class for SX Gate.
CNOT([has_params, trainable, init_params, ...])	Class for CNOT Gate.
CZ([has_params, trainable, init_params, ...])	Class for CZ Gate.
CY([has_params, trainable, init_params, ...])	Class for CY Gate.
RX([has_params, trainable, init_params, ...])	Class for RX Gate.
RY([has_params, trainable, init_params, ...])	Class for RY Gate.
RZ([has_params, trainable, init_params, ...])	Class for RZ Gate.
RXX([has_params, trainable, init_params, ...])	Class for RXX Gate.
RYY([has_params, trainable, init_params, ...])	Class for RYY Gate.
RZZ([has_params, trainable, init_params, ...])	Class for RZZ Gate.
RZX([has_params, trainable, init_params, ...])	Class for RZX Gate.
SWAP([has_params, trainable, init_params, ...])	Class for SWAP Gate.
SSWAP([has_params, trainable, init_params, ...])	Class for SSWAP Gate.
CSWAP([has_params, trainable, init_params, ...])	Class for CSWAP Gate.
Toffoli([has_params, trainable, ...])	Class for Toffoli Gate.

continues on next page

Table 1 – continued from previous page

PhaseShift([has_params, trainable, ...])	Class for PhaseShift Gate.
Rot([has_params, trainable, init_params, ...])	Class for Rotation Gate.
MultiRZ([has_params, trainable, ...])	Class for Multi-qubit RZ Gate.
CRX([has_params, trainable, init_params, ...])	Class for Controlled Rotation X gate.
CRY([has_params, trainable, init_params, ...])	Class for Controlled Rotation Y gate.
CRZ([has_params, trainable, init_params, ...])	Class for Controlled Rotation Z gate.
CRot([has_params, trainable, init_params, ...])	Class for Controlled Rotation gate.
U1([has_params, trainable, init_params, ...])	Class for Controlled Rotation Y gate.
U2([has_params, trainable, init_params, ...])	Class for U2 gate.
U3([has_params, trainable, init_params, ...])	Class for U3 gate.
CU1([has_params, trainable, init_params, ...])	Class for controlled U1 gate.
CU2([has_params, trainable, init_params, ...])	Class for controlled U2 gate.
CU3([has_params, trainable, init_params, ...])	Class for Controlled U3 gate.
QubitUnitary([has_params, trainable, ...])	Class for controlled Qubit Unitary gate.
QubitUnitaryFast([has_params, trainable, ...])	Class for fast implementation of controlled Qubit Unitary gate.
TrainableUnitary([has_params, trainable, ...])	Class for TrainableUnitary Gate.
TrainableUnitaryStrict([has_params, ...])	Class for Strict Unitary matrix gate.
MultiCNOT([has_params, trainable, ...])	Class for Multi qubit CNOT gate.
MultiXCNOT([has_params, trainable, ...])	Class for Multi qubit XCNOT gate.
Reset([has_params, trainable, init_params, ...])	Class for Reset gate.

2.1.1 WiresEnum

class torchquantum.operators.**WiresEnum**(*value*)

Bases: IntEnum

Integer enumeration class to represent the number of wires an operation acts on.

__init__()

Attributes

AnyWires = -1

AllWires = 0

2.1.2 NParamsEnum

class torchquantum.operators.**NParamsEnum**(*value*)

Bases: IntEnum

Integer enumeration class to represent the number of wires an operation acts on

__init__()

Attributes

AnyNParams = -1

2.1.3 AllWires

torchquantum.operators.**AllWires**

IntEnum: An enumeration which represents all wires in the subsystem. It is equivalent to an integer with value 0.

Attributes

2.1.4 AnyWires

torchquantum.operators.**AnyWires**

IntEnum: An enumeration which represents any wires in the subsystem. It is equivalent to an integer with value -1.

Attributes

2.1.5 Operator

class torchquantum.operators.**Operator**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: QuantumModule

The class for quantum operators.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

__init__ function for Operator.

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

forward(*q_device: QuantumDevice*, *wires=None*, *params=None*, *inverse=False*)

Apply the operator to the quantum device states.

Parameters

- **q_device** (*torchquantum.QuantumDevice*) – Quantum Device that the operator is applied to.
- **wires** (*Union[int, List[int]]*) – Qubits that the operator is applied to.
- **params** (*torch.Tensor*) – Parameters of the operator
- **inverse** (*bool*) – Whether inverse the unitary matrix of the operator.

Returns:

set_wires(*wires*)

Set which qubits the operator is applied to.

Parameters

wires (*Union[int, List[int]]*) – Qubits the operator is applied to.

Returns: None.

Attributes

eigvals

The eigenvalues of the unitary matrix of the operator.

Returns: Eigenvalues.

fixed_ops = ['Hadamard', 'SHadamard', 'PauliX', 'PauliY', 'PauliZ', 'I', 'S', 'T', 'SX', 'CNOT', 'CZ', 'CY', 'SWAP', 'SSWAP', 'CSWAP', 'Toffoli', 'MultiCNOT', 'MultiXCNOT', 'Reset']

matrix

The unitary matrix of the operator.

name

String for the name of the operator.

parameterized_ops = ['RX', 'RY', 'RZ', 'RXX', 'RYY', 'RZZ', 'RZX', 'PhaseShift', 'Rot', 'MultiRZ', 'CRX', 'CRY', 'CRZ', 'CRot', 'U1', 'U2', 'U3', 'CU1', 'CU2', 'CU3', 'QubitUnitary', 'QubitUnitaryFast', 'TrainableUnitary', 'TrainableUnitaryStrict']

training: bool

2.1.6 Observable

class torchquantum.operators.**Observable**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operator*

Class for Observables.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Init function of the Observable class

has_params (bool, optional): Whether the operations has parameters.

Defaults to False.

trainable (bool, optional): Whether the parameters are trainable
(if contains parameters). Defaults to False.

init_params (torch.Tensor, optional): Initial parameters.
Defaults to None.

n_wires (int, optional): Number of qubits. Defaults to None. wires (Union[int, List[int]], optional): Which qubit the operation

is applied to. Defaults to None.

Methods

diagonalizing_gates()

The diagonalizing gates when perform measurements.

Returns: None.

Attributes

training: bool

2.1.7 Operation

class torchquantum.operators.**Operation**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operator*

summary

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.

- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

build_params(*trainable*)

Build parameters.

Parameters
trainable (*bool*) – Whether the parameters are trainable.

Returns
Built parameters.

Return type
torch.Tensor

init_params()

Initialize the parameters.

Raises
NotImplementedError – The init param function is not implemented.

reset_params(*init_params=None*)

Reset parameters.

Parameters
init_params (*torch.Tensor, optional*) – Input the initialization parameters. Defaults to None.

Attributes

eigvals

"The eigenvalues of the unitary matrix of the operator.

Returns
Eigenvalues.

Return type
torch.Tensor

matrix

The unitary matrix of the operator.

training: bool

2.1.8 DiagonalOperation

class torchquantum.operators.**DiagonalOperation**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Diagonal Operation.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Attributes

eigvals

The eigenvalues of the unitary matrix of the operator.

Returns: Eigenvalues.

training: bool

2.1.9 Hadamard

class torchquantum.operators.**Hadamard**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Observable*

Class for Hadamard Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Init function of the Observable class

has_params (bool, optional): Whether the operations has parameters.

Defaults to False.

trainable (bool, optional): Whether the parameters are trainable
(if contains parameters). Defaults to False.

init_params (torch.Tensor, optional): Initial parameters.
Defaults to None.

n_wires (int, optional): Number of qubits. Defaults to None. wires (Union[int, List[int]], optional): Which qubit the operation

is applied to. Defaults to None.

Methods

diagonalizing_gates()

The diagonalizing gates when perform measurements.

Returns: None.

static func(*q_device: QuantumDevice*, *wires: Union[List[int], int]*, *params: Optional[Tensor] = None*, *n_wires: Optional[int] = None*, *static: bool = False*, *parent_graph=None*, *inverse: bool = False*, *comp_method: str = 'bmm'*)

Perform the hadamard gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

eigvals = tensor([1.+0.j, -1.+0.j])

matrix = tensor([[0.7071+0.j, 0.7071+0.j], [0.7071+0.j, -0.7071+0.j]])

num_params = 0

num_wires = 1

training: bool

2.1.10 SHadamard

class torchquantum.operators.**SHadamard**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for SHadamard Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the shadamard gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

Attributes

matrix = tensor([[0.9239+0.j, -0.3827+0.j], [0.3827+0.j, 0.9239+0.j]])

num_params = 0

num_wires = 1

training: bool

2.1.11 PauliX

class torchquantum.operators.**PauliX**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Observable*

Class for Pauli X Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Init function of the Observable class

has_params (bool, optional): Whether the operations has parameters.

Defaults to False.

trainable (bool, optional): Whether the parameters are trainable
(if contains parameters). Defaults to False.

init_params (torch.Tensor, optional): Initial parameters.
Defaults to None.

n_wires (int, optional): Number of qubits. Defaults to None. wires (Union[int, List[int]], optional): Which qubit the operation

is applied to. Defaults to None.

Methods

diagonalizing_gates()

The diagonalizing gates when perform measurements.

Returns: None.

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the Pauli X gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.

- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

eigvals = tensor([1.+0.j, -1.+0.j])

matrix = tensor([[0.+0.j, 1.+0.j], [1.+0.j, 0.+0.j]])

num_params = 0

num_wires = 1

training: bool

2.1.12 PauliY

class torchquantum.operators.**PauliY**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Observable*

Class for Pauli Y Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Init function of the Observable class

has_params (bool, optional): Whether the operations has parameters.

Defaults to False.

trainable (bool, optional): Whether the parameters are trainable
(if contains parameters). Defaults to False.

init_params (torch.Tensor, optional): Initial parameters.
Defaults to None.

n_wires (int, optional): Number of qubits. Defaults to None. wires (Union[int, List[int]], optional): Which qubit the operation

is applied to. Defaults to None.

Methods

diagonalizing_gates()

The diagonalizing gates when perform measurements.

Returns: None.

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the Pauli Y gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

eigvals = tensor([1.+0.j, -1.+0.j])

matrix = tensor([[0.+0.j, -0.-1.j], [0.+1.j, 0.+0.j]])

num_params = 0

num_wires = 1

training: bool

2.1.13 PauliZ

class torchquantum.operators.**PauliZ**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Observable*

Class for Pauli Z Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Init function of the Observable class

has_params (bool, optional): Whether the operations has parameters.

Defaults to False.

trainable (bool, optional): Whether the parameters are trainable
(if contains parameters). Defaults to False.

init_params (torch.Tensor, optional): Initial parameters.
Defaults to None.

n_wires (int, optional): Number of qubits. Defaults to None. wires (Union[int, List[int]], optional): Which qubit the operation

is applied to. Defaults to None.

Methods

`diagonalizing_gates()`

The diagonalizing gates when perform measurements.

Returns: None.

`static func(q_device, wires, params=None, n_wires=None, static=False, parent_graph=None, inverse=False, comp_method='bmm')`

Perform the Pauli Z gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

`eigvals = tensor([ 1.+0.j, -1.+0.j])`

`matrix = tensor([[ 1.+0.j, 0.+0.j], [ 0.+0.j, -1.+0.j]])`

`num_params = 0`

`num_wires = 1`

`training: bool`

2.1.14 I

class torchquantum.operators.I(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Observable*

Class for Identity Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Init function of the Observable class

has_params (bool, optional): Whether the operations has parameters.

Defaults to False.

trainable (bool, optional): Whether the parameters are trainable
(if contains parameters). Defaults to False.

init_params (torch.Tensor, optional): Initial parameters.
Defaults to None.

n_wires (int, optional): Number of qubits. Defaults to None. wires (Union[int, List[int]], optional): Which qubit the operation

is applied to. Defaults to None.

Methods

diagonalizing_gates()

The diagonalizing gates when perform measurements.

Returns: None.

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the I gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

eigvals = tensor([1.+0.j, 1.+0.j])

matrix = tensor([[1.+0.j, 0.+0.j], [0.+0.j, 1.+0.j]])

num_params = 0

num_wires = 1

training: bool

2.1.15 S

class torchquantum.operators.**S**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *DiagonalOperation*

Class for S Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the s gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.

- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

eigvals = tensor([1.+0.j, 0.+1.j])

matrix = tensor([[1.+0.j, 0.+0.j], [0.+0.j, 0.+1.j]])

num_params = 0

num_wires = 1

training: bool

2.1.16 T

class torchquantum.operators.**T**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *DiagonalOperation*

Class for T Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the t gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.

- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

eigvals = tensor([1.+0.j, 0.+1.j])

matrix = tensor([[1.0000+0.0000j, 0.0000+0.0000j], [0.0000+0.0000j, 0.7071+0.7071j]])

num_params = 0

num_wires = 1

training: bool

2.1.17 SX

class torchquantum.operators.**SX**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for SX Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the sx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

eigvals = tensor([1.+0.j, 0.+1.j])

matrix = tensor([[0.5000+0.5000j, 0.5000-0.5000j], [0.5000-0.5000j, 0.5000+0.5000j]])

num_params = 0

num_wires = 1

training: bool

2.1.18 CNOT

class torchquantum.operators.**CNOT**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for CNOT Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.

- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cnot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

Attributes

matrix = tensor([[1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j], [0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j]])

num_params = 0

num_wires = 2

training: bool

2.1.19 CZ

class torchquantum.operators.**CZ**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *DiagonalOperation*

Class for CZ Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cz gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

Attributes

eigvals = array([1, 1, 1, -1])

matrix = tensor([[1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, -1.+0.j]])

num_params = 0

num_wires = 2

training: bool

2.1.20 CY

class torchquantum.operators.**CY**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for CY Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cy gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.

- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

matrix = tensor([[1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, -0.-1.j], [0.+0.j, 0.+0.j, 0.+1.j, 0.+0.j]])

num_params = 0

num_wires = 2

training: bool

2.1.21 RX

class torchquantum.operators.**RX**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for RX Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.

- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 1

training: bool

2.1.22 RY

class torchquantum.operators.**RY**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for RY Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the ry gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 1

training: bool

2.1.23 RZ

class torchquantum.operators.**RZ**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *DiagonalOperation*

Class for RZ Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.

- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rz gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 1

training: bool

2.1.24 RXX

class torchquantum.operators.**RXX**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for RXX Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.

- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rxx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 2

training: bool

2.1.25 RYY

class torchquantum.operators.**RYY**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for RYY Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the ryy gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 2

training: bool

2.1.26 RZZ

class torchquantum.operators.**RZZ**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *DiagonalOperation*

Class for RZZ Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rzz gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

Attributes

num_params = 1

num_wires = 2

training: bool

2.1.27 RZX

class torchquantum.operators.**RZX**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for RZX Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rzx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 2

training: bool

2.1.28 SWAP

class torchquantum.operators.**SWAP**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for SWAP Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the swap gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.

- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

matrix = tensor([[1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j], [0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j]])

num_params = 0

num_wires = 2

training: bool

2.1.29 SSWAP

class torchquantum.operators.**SSWAP**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for SSWAP Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the sswap gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.

- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

matrix = tensor([[1.0000+0.0000j, 0.0000+0.0000j, 0.0000+0.0000j, 0.0000+0.0000j], [0.0000+0.0000j, 0.5000+0.5000j, 0.5000-0.5000j, 0.0000+0.0000j], [0.0000+0.0000j, 0.5000-0.5000j, 0.5000+0.5000j, 0.0000+0.0000j], [0.0000+0.0000j, 0.0000+0.0000j, 0.0000+0.0000j, 1.0000+0.0000j]])

num_params = 0

num_wires = 2

training: bool

2.1.30 CSWAP

class torchquantum.operators.**CSWAP**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for CSWAP Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cswap gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

matrix = tensor([[1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j]])

num_params = 0

num_wires = 3

training: bool

2.1.31 Toffoli

class torchquantum.operators.**Toffoli**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Toffoli Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the toffoli gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

Attributes

matrix = tensor([[1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j, 0.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j], [0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j, 0.+0.j]])

num_params = 0

num_wires = 3

training: bool

2.1.32 PhaseShift

class torchquantum.operators.**PhaseShift**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *DiagonalOperation*

Class for PhaseShift Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the phaseshift gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 1

training: bool

2.1.33 Rot

class torchquantum.operators.**Rot**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Rotation Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the rot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 3

num_wires = 1

training: bool

2.1.34 MultiRZ

class torchquantum.operators.**MultiRZ**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *DiagonalOperation*

Class for Multi-qubit RZ Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the multi qubit RZ gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.

- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = -1

training: bool

2.1.35 CRX

class torchquantum.operators.**CRX**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Controlled Rotation X gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the crx gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.

- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 2

training: bool

2.1.36 CRY

class torchquantum.operators.**CRY**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Controlled Rotation Y gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cry gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 2

training: bool

2.1.37 CRZ

class torchquantum.operators.**CRZ**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Controlled Rotation Z gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.

- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the crz gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 2

training: bool

2.1.38 CRot

class torchquantum.operators.**CRot**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Controlled Rotation gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.

- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the crot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 3

num_wires = 2

training: bool

2.1.39 U1

class torchquantum.operators.**U1**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *DiagonalOperation*

Class for Controlled Rotation Y gate. U1 is the same as phaseshift.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the u1 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 1

training: bool

2.1.40 U2

class torchquantum.operators.U2(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for U2 gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the u2 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns

None.

Attributes

num_params = 2

num_wires = 1

training: bool

2.1.41 U3

class torchquantum.operators.**U3**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for U3 gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the u3 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 3

num_wires = 1

training: bool

2.1.42 CU1

class torchquantum.operators.**CU1**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *DiagonalOperation*

Class for controlled U1 gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cu1 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.

- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 1

num_wires = 2

training: bool

2.1.43 CU2

class torchquantum.operators.**CU2**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for controlled U2 gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cu2 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.

- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 2

num_wires = 2

training: bool

2.1.44 CU3

class torchquantum.operators.**CU3**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Controlled U3 gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the cu3 gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = 3

num_wires = 2

training: bool

2.1.45 QubitUnitary

class torchquantum.operators.**QubitUnitary**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for controlled Qubit Unitary gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.

- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

build_params(*trainable*)

Build parameters.

Parameters
trainable (*bool*) – Whether the parameters are trainable.

Returns
Built parameters.

Return type
torch.Tensor

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the qubitunitary gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

reset_params(*init_params=None*)

Reset parameters.

Parameters
init_params (*torch.Tensor, optional*) – Input the initialization parameters. Defaults to None.

Attributes

num_params = -1

num_wires = -1

training: bool

2.1.46 QubitUnitaryFast

class torchquantum.operators.**QubitUnitaryFast**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for fast implementation of controlled Qubit Unitary gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

build_params(*trainable*)

Build parameters.

Parameters
trainable (*bool*) – Whether the parameters are trainable.

Returns
Built parameters.

Return type
torch.Tensor

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the qubitunitaryfast gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.

- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

reset_params(*init_params=None*)

Reset parameters.

Parameters
init_params (*torch.Tensor, optional*) – Input the initialization parameters. Defaults to None.

Attributes

num_params = -1

num_wires = -1

training: bool

2.1.47 TrainableUnitary

class torchquantum.operators.**TrainableUnitary**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for TrainableUnitary Gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

build_params(*trainable*)

Build the parameters for the gate.

Parameters
trainable (*bool*) – Whether the parameters are trainble.

Returns
Parameters.

Return type
torch.Tensor

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the qubitunitaryfast gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

reset_params(*init_params=None*)

Reset the parameters.

Parameters
init_params (*torch.Tensor, optional*) – Initial parameters.

Returns
None.

Attributes

num_params = -1

num_wires = -1

training: bool

2.1.48 TrainableUnitaryStrict

class torchquantum.operators.**TrainableUnitaryStrict**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *TrainableUnitary*

Class for Strict Unitary matrix gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the qubitunitarystrict = gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

num_params = -1

num_wires = -1

training: bool

2.1.49 MultiCNOT

class torchquantum.operators.**MultiCNOT**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Multi qubit CNOT gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the multi qubit cnot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.
- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

matrix

num_params = 0

num_wires = -1

training: bool

2.1.50 MultiXCNOT

class torchquantum.operators.**MultiXCNOT**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operation*

Class for Multi qubit XCNOT gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

summary

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device*, *wires*, *params=None*, *n_wires=None*, *static=False*, *parent_graph=None*, *inverse=False*, *comp_method='bmm'*)

Perform the multi qubit xcnot gate.

Parameters

- **q_device** (*tq.QuantumDevice*) – The QuantumDevice.
- **wires** (*Union[List[int], int]*) – Which qubit(s) to apply the gate.
- **params** (*torch.Tensor, optional*) – Parameters (if any) of the gate. Default to None.
- **n_wires** (*int, optional*) – Number of qubits the gate is applied to. Default to None.
- **static** (*bool, optional*) – Whether use static mode computation. Default to False.

- **parent_graph** (*tq.QuantumGraph, optional*) – Parent QuantumGraph of current operation. Default to None.
- **inverse** (*bool, optional*) – Whether inverse the gate. Default to False.
- **comp_method** (*bool, optional*) – Use 'bmm' or 'einsum' method to perform
- **'bmm'.** (*matrix vector multiplication. Default to*) –

Returns
None.

Attributes

matrix

num_params = 0

num_wires = -1

training: bool

2.1.51 Reset

class torchquantum.operators.**Reset**(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

Bases: *Operator*

Class for Reset gate.

__init__(*has_params: bool = False*, *trainable: bool = False*, *init_params=None*, *n_wires=None*, *wires=None*)

__init__ function for Operator.

Parameters

- **has_params** (*bool, optional*) – Whether the operations has parameters. Defaults to False.
- **trainable** (*bool, optional*) – Whether the parameters are trainable (if contains parameters). Defaults to False.
- **init_params** (*torch.Tensor, optional*) – Initial parameters. Defaults to None.
- **n_wires** (*int, optional*) – Number of qubits. Defaults to None.
- **wires** (*Union[int, List[int]], optional*) – Which qubit the operation is applied to. Defaults to None.

Methods

static func(*q_device: QuantumDevice*, *wires*, *inverse=False*)

Attributes

num_params = 0

num_wires = -1

training: bool

CHAPTER

THREE

INSTALLATION

To use TorchQuantum, first install it using pip:

```
(.venv) $ pip install torchquantum
```

CHAPTER

FOUR

EXAMPLES

4.1 Quanvolution

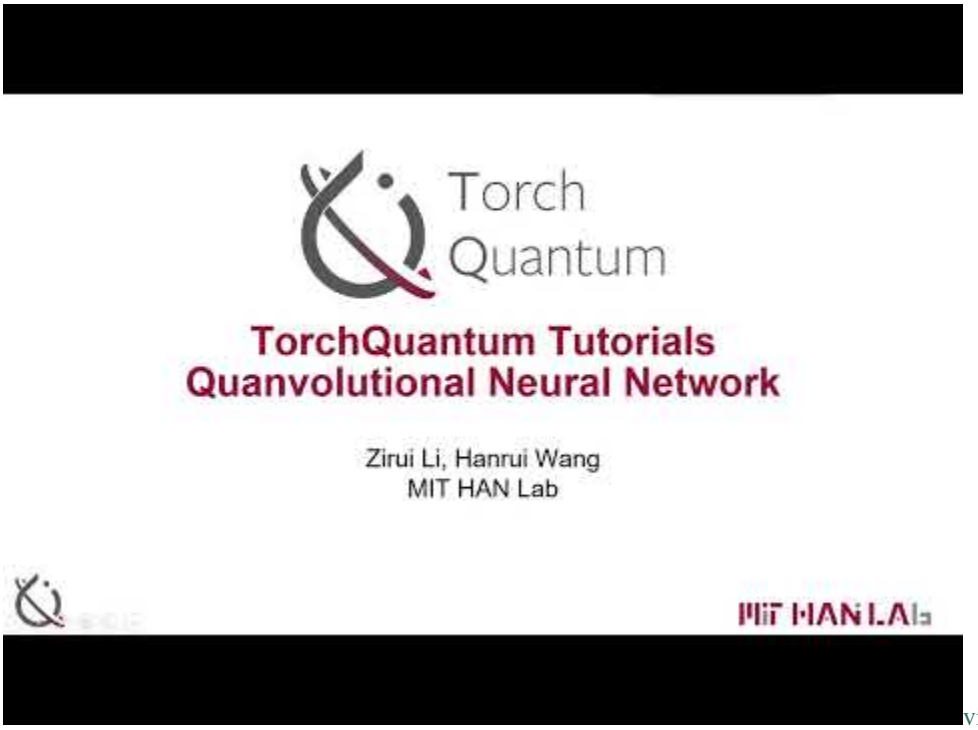
4.1.1 Quantum Convolution

Quantum Convolution (Quanvolution) for MNIST image classification

Authors: Zirui Li, Hanrui Wang

Use Colab to run this example:

See this tutorial video for detailed explanations:



video

Referece: Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits

Outline

1. Introduction to Quanvolutional Neural Network.
2. Build and train a Quanvolutional Neural Network.

- a. Compare Quanvolutional Neural Network with a classic model.
- b. Evaluate on real quantum computer.

1. Compare multiple models with or without a trainable quanvolutional filter.

[comment]: <> (#%% md)

In this tutorial, we use `tq.QuantumDevice`, `tq.GeneralEncoder`, `tq.RandomLayer`, `tq.MeasureAll`, `tq.PauliZ` class from TrochQuantum.

You can learn how to build, train and evaluate a quanvolutional filter using TorchQuantum in this tutorial.

[comment]: <> (#%% md)

Introduction to Quanvolutional Neural Network.

Convolutional Neural Network

Convolutional neural network is a classic neural network genre, mostly applied to anylize visual images. They are known for their convolutional layers that perform convolution. Typically the convolution operation is the Frobenius inner product of the convolution filter with the input image followed by an activation function. The convolution filter slides along the input image and generates a feature map. We can use the feature map for classification.

Quantum convolution

One can extend the same idea also to the context of quantum variational circuits. Replace the classical convolution filters with variational quantum circuits and we get quanvolutional neural networks with quanvolutional filters. The quanvolutional filters perform more complex operations in a higher dimension Hilbert space than Frobenius inner product. Therefore, quanvolutional filters have more potential than traditional convolution filters.

4.1.2 Quanvolution (Quantum convolution) for MNIST image classification with TorchQuantum.

Tutorial Author: Zirui Li, Hanrui Wang

Outline

1. Introduction to Quanvolutional Neural Network.
2. Build and train a Quanvolutional Neural Network.
   - a. Compare Quanvolutional Neural Network with a classic model.
   - b. Evaluate on real quantum computer.
3. Compare multiple models with or without a trainable quanvolutional filter.

In this tutorial, we use `tq.QuantumDevice`, `tq.GeneralEncoder`, `tq.RandomLayer`, `tq.MeasureAll`, `tq.PauliZ` class from TrochQuantum.

You can learn how to build, train and evaluate a quanvolutional filter using TorchQuantum in this tutorial.

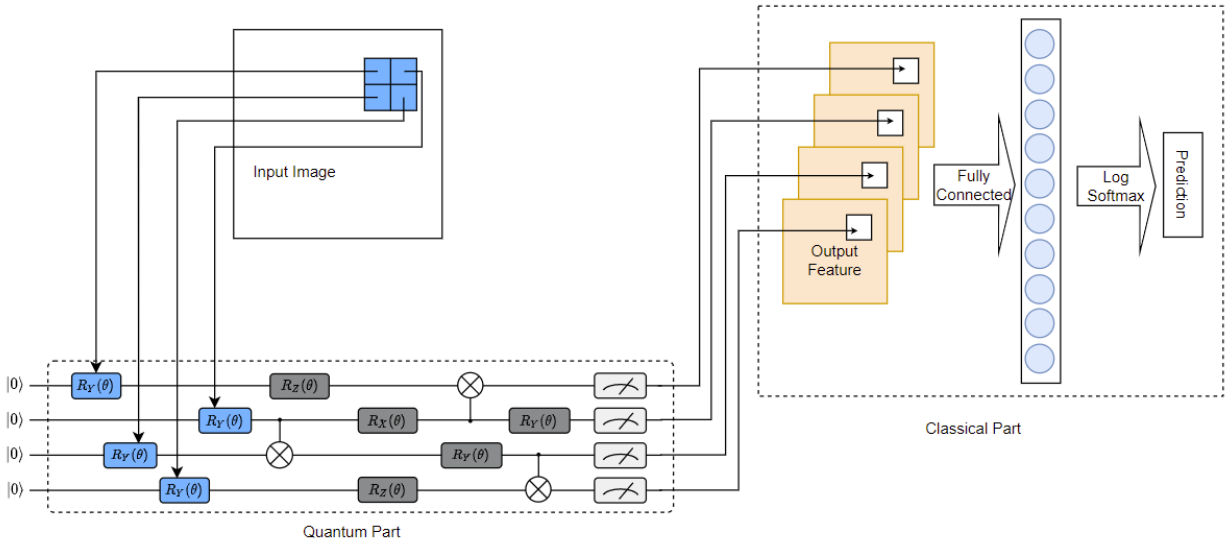
Introduction to Quanvolutional Neural Network.

Convolutional Neural Network

Convolutional neural network is a classic neural network genre, mostly applied to anylize visual images. They are known for their convolutional layers that perform convolution. Typically the convolution operation is the Frobenius inner product of the convolution filter with the input image followed by an activation function. The convolution filter slides along the input image and generates a feature map. We can use the feature map for classification.

Quantum convolution

One can extend the same idea also to the context of quantum variational circuits. Replace the classical convolution filters with variational quantum circuits and we get quanvolutional neural networks with quanvolutional filters. The quanvolutional filters perform more complex operations in a higher dimension Hilbert space than Frobenius inner product. Therefore, quanvolutional filters have more potential than traditional convolution filters.



Build and train a Quanvolutional Neural Network.

Installation

Install torchquantum and all the libs we need.

[ ]:
```
!pip install qiskit==0.32.1
```

```
Collecting qiskit==0.32.1
  Downloading qiskit-0.32.1.tar.gz (13 kB)
Collecting qiskit-terra==0.18.3
  Downloading qiskit_terra-0.18.3-cp37-cp37m-manylinux2010_x86_64.whl (6.1 MB)
     || 6.1 MB 4.1 MB/s
Collecting qiskit-aer==0.9.1
  Downloading qiskit_aer-0.9.1-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl␣
↪(17.9 MB)
     || 17.9 MB 633 kB/s
Collecting qiskit-ibmq-provider==0.18.1
  Downloading qiskit_ibmq_provider-0.18.1-py3-none-any.whl (237 kB)
     || 237 kB 73.3 MB/s
Collecting qiskit-ignis==0.6.0
  Downloading qiskit_ignis-0.6.0-py3-none-any.whl (207 kB)
     || 207 kB 65.4 MB/s
Collecting qiskit-aqua==0.9.5
  Downloading qiskit_aqua-0.9.5-py3-none-any.whl (2.1 MB)
     || 2.1 MB 60.5 MB/s
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aer==0.9.1->qiskit==0.32.1) (1.4.1)
Requirement already satisfied: numpy>=1.16.3 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aer==0.9.1->qiskit==0.32.1) (1.21.5)
Requirement already satisfied: h5py<3.3.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aqua==0.9.5->qiskit==0.32.1) (3.1.0)
Collecting quandl
  Downloading Quandl-3.7.0-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-aqua==0.9.5->qiskit==0.32.1) (1.0.2)
Collecting yfinance>=0.1.62
  Downloading yfinance-0.1.70-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.7/dist-packages (from␣
↪qiskit-aqua==0.9.5->qiskit==0.32.1) (5.4.8)
Collecting docplex>=2.21.207
  Downloading docplex-2.22.213.tar.gz (634 kB)
     || 634 kB 68.2 MB/s
Requirement already satisfied: setuptools>=40.1.0 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-aqua==0.9.5->qiskit==0.32.1) (57.4.0)
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aqua==0.9.5->qiskit==0.32.1) (1.7.1)
Collecting retworkx>=0.8.0
  Downloading retworkx-0.11.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.
↪manylinux_2_12_x86_64.manylinux2010_x86_64.whl (1.6 MB)
     || 1.6 MB 21.6 MB/s
Requirement already satisfied: fastdtw<=0.3.4 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aqua==0.9.5->qiskit==0.32.1) (0.3.4)
```

(continues on next page)

(continued from previous page)

```
Collecting dlx<=1.0.4
  Downloading dlx-1.0.4.tar.gz (5.5 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from␣
↪qiskit-aqua==0.9.5->qiskit==0.32.1) (1.3.5)
Requirement already satisfied: urllib3>=1.21.1 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-ibmq-provider==0.18.1->qiskit==0.32.1) (1.24.3)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-ibmq-provider==0.18.1->qiskit==0.32.1) (2.8.2)
Collecting requests-ntlm>=1.1.0
  Downloading requests_ntlm-1.1.0-py2.py3-none-any.whl (5.7 kB)
Requirement already satisfied: requests>=2.19 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-ibmq-provider==0.18.1->qiskit==0.32.1) (2.23.0)
Collecting websocket-client>=1.0.1
  Downloading websocket_client-1.2.3-py3-none-any.whl (53 kB)
     || 53 kB 2.7 MB/s
Collecting python-constraint>=1.4
  Downloading python-constraint-1.4.0.tar.bz2 (18 kB)
Collecting fastjsonschema>=2.10
  Downloading fastjsonschema-2.15.3-py3-none-any.whl (22 kB)
Collecting symengine>0.7
  Downloading symengine-0.8.1-cp37-cp37m-manylinux2010_x86_64.whl (38.2 MB)
     || 38.2 MB 116 kB/s
Collecting tweedledum<2.0,>=1.1
  Downloading tweedledum-1.1.1-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl␣
↪(943 kB)
     || 943 kB 22.1 MB/s
Collecting ply>=3.10
  Downloading ply-3.11-py2.py3-none-any.whl (49 kB)
     || 49 kB 8.3 MB/s
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.7/dist-packages (from␣
↪qiskit-terra==0.18.3->qiskit==0.32.1) (0.3.4)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-terra==0.18.3->qiskit==0.32.1) (4.3.3)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from␣
↪docplex>=2.21.207->qiskit-aqua==0.9.5->qiskit==0.32.1) (1.15.0)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages␣
↪(from h5py<3.3.0->qiskit-aqua==0.9.5->qiskit==0.32.1) (1.5.2)
Requirement already satisfied: importlib-resources>=1.4.0 in /usr/local/lib/python3.7/
↪dist-packages (from jsonschema>=2.6->qiskit-terra==0.18.3->qiskit==0.32.1) (5.4.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-
↪packages (from jsonschema>=2.6->qiskit-terra==0.18.3->qiskit==0.32.1) (3.10.0.2)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from jsonschema>=2.6->qiskit-terra==0.18.3->qiskit==0.32.1) (21.4.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-
↪packages (from jsonschema>=2.6->qiskit-terra==0.18.3->qiskit==0.32.1) (4.11.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/
↪local/lib/python3.7/dist-packages (from jsonschema>=2.6->qiskit-terra==0.18.3->
↪qiskit==0.32.1) (0.18.1)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from importlib-resources>=1.4.0->jsonschema>=2.6->qiskit-terra==0.18.3->qiskit==0.32.
↪1) (3.7.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-
↪packages (from requests>=2.19->qiskit-ibmq-provider==0.18.1->qiskit==0.32.1) (2021.10.
↪8)
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages␣
↪(from requests>=2.19->qiskit-ibmq-provider==0.18.1->qiskit==0.32.1) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-
↪packages (from requests>=2.19->qiskit-ibmq-provider==0.18.1->qiskit==0.32.1) (3.0.4)
Collecting cryptography>=1.3
  Downloading cryptography-36.0.1-cp36-abi3-manylinux_2_24_x86_64.whl (3.6 MB)
     || 3.6 MB 31.3 MB/s
Collecting ntlm-auth>=1.0.2
  Downloading ntlm_auth-1.5.0-py2.py3-none-any.whl (29 kB)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.7/dist-packages␣
↪(from cryptography>=1.3->requests-ntlm>=1.1.0->qiskit-ibmq-provider==0.18.1->qiskit==0.
↪32.1) (1.15.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from␣
↪cffi>=1.12->cryptography>=1.3->requests-ntlm>=1.1.0->qiskit-ibmq-provider==0.18.1->
↪qiskit==0.32.1) (2.21)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages␣
↪(from scikit-learn>=0.20.0->qiskit-aqua==0.9.5->qiskit==0.32.1) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
↪packages (from scikit-learn>=0.20.0->qiskit-aqua==0.9.5->qiskit==0.32.1) (3.1.0)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.7/dist-packages␣
↪(from sympy>=1.3->qiskit-aqua==0.9.5->qiskit==0.32.1) (1.2.1)
Collecting requests>=2.19
  Downloading requests-2.27.1-py2.py3-none-any.whl (63 kB)
     || 63 kB 814 kB/s
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-
↪packages (from yfinance>=0.1.62->qiskit-aqua==0.9.5->qiskit==0.32.1) (0.0.10)
Collecting lxml>=4.5.1
  Downloading lxml-4.7.1-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_
↪2_24_x86_64.whl (6.4 MB)
     || 6.4 MB 30.3 MB/s
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages␣
↪(from pandas->qiskit-aqua==0.9.5->qiskit==0.32.1) (2018.9)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.7/
↪dist-packages (from requests>=2.19->qiskit-ibmq-provider==0.18.1->qiskit==0.32.1) (2.0.
↪11)
Collecting inflection>=0.3.1
  Downloading inflection-0.5.1-py2.py3-none-any.whl (9.5 kB)
Requirement already satisfied: more-itertools in /usr/local/lib/python3.7/dist-packages␣
↪(from quandl->qiskit-aqua==0.9.5->qiskit==0.32.1) (8.12.0)
Building wheels for collected packages: qiskit, dlx, docplex, python-constraint
  Building wheel for qiskit (setup.py) ... done
  Created wheel for qiskit: filename=qiskit-0.32.1-py3-none-any.whl size=11777␣
↪sha256=911365fec91e5c648d2569b156af429c0aff7c3d95d453a7d24e6bf8d7d1a315
  Stored in directory: /root/.cache/pip/wheels/0f/62/0a/
↪c53eda1ead41c137c47c9730bc2771a8367b1ce00fb64e8cc6
  Building wheel for dlx (setup.py) ... done
  Created wheel for dlx: filename=dlx-1.0.4-py3-none-any.whl size=5718␣
↪sha256=cb913d8c2b19d87e8784f4220a02752c4858e3ebe531b0e80ab22c5625c1bd0b
  Stored in directory: /root/.cache/pip/wheels/78/55/c8/
↪dc61e772445a566b7608a476d151e9dcaf4e092b01b0c4bc3c
  Building wheel for docplex (setup.py) ... done
  Created wheel for docplex: filename=docplex-2.22.213-py3-none-any.whl size=696882␣
↪sha256=192bab0a25875036008ce12a090c9f129f2a6b0e88f3a41e568c07ca585b4e3ff
```

(continues on next page)

(continued from previous page)

```
  Stored in directory: /root/.cache/pip/wheels/90/69/6b/
↪1375c68a5b7ff94c40263b151c86f58bd72200bf0c465b5ba3
  Building wheel for python-constraint (setup.py) ... done
  Created wheel for python-constraint: filename=python_constraint-1.4.0-py2.py3-none-any.
↪whl size=24081 sha256=77684dcb7c666715267053a0114cf933c5c52cb7af9a30645de961f9af12c323
  Stored in directory: /root/.cache/pip/wheels/07/27/db/
↪1222c80eb1e431f3d2199c12569cb1cac60f562a451fe30479
Successfully built qiskit dlx docplex python-constraint
Installing collected packages: tweedledum, symengine, retworkx, python-constraint, ply,␣
↪fastjsonschema, requests, qiskit-terra, ntlm-auth, lxml, inflection, cryptography,␣
↪yfinance, websocket-client, requests-ntlm, quandl, qiskit-ignis, docplex, dlx, qiskit-
↪ibmq-provider, qiskit-aqua, qiskit-aer, qiskit
  Attempting uninstall: requests
    Found existing installation: requests 2.23.0
    Uninstalling requests-2.23.0:
      Successfully uninstalled requests-2.23.0
  Attempting uninstall: lxml
    Found existing installation: lxml 4.2.6
    Uninstalling lxml-4.2.6:
      Successfully uninstalled lxml-4.2.6
ERROR: pip's dependency resolver does not currently take into account all the packages␣
↪that are installed. This behaviour is the source of the following dependency conflicts.
google-colab 1.0.0 requires requests~=2.23.0, but you have requests 2.27.1 which is␣
↪incompatible.
datascience 0.10.6 requires folium==0.2.1, but you have folium 0.8.3 which is␣
↪incompatible.
Successfully installed cryptography-36.0.1 dlx-1.0.4 docplex-2.22.213 fastjsonschema-2.
↪15.3 inflection-0.5.1 lxml-4.7.1 ntlm-auth-1.5.0 ply-3.11 python-constraint-1.4.0␣
↪qiskit-0.32.1 qiskit-aer-0.9.1 qiskit-aqua-0.9.5 qiskit-ibmq-provider-0.18.1 qiskit-
↪ignis-0.6.0 qiskit-terra-0.18.3 quandl-3.7.0 requests-2.27.1 requests-ntlm-1.1.0␣
↪retworkx-0.11.0 symengine-0.8.1 tweedledum-1.1.1 websocket-client-1.2.3 yfinance-0.1.70
```

Download and cd to the repo.

```
[ ]: !git clone https://github.com/mit-han-lab/torchquantum.git
```

```
Cloning into 'torchquantum'...
remote: Enumerating objects: 10737, done.
remote: Counting objects: 100% (7529/7529), done.
remote: Compressing objects: 100% (3777/3777), done.
remote: Total 10737 (delta 3765), reused 7076 (delta 3348), pack-reused 3208
Receiving objects: 100% (10737/10737), 3.19 MiB | 12.92 MiB/s, done.
Resolving deltas: 100% (5732/5732), done.
Checking out files: 100% (50055/50055), done.
```

```
[ ]: %cd torchquantum
```

```
/content/torchquantum
```

Install torch-quantum.

```
[ ]: !pip install --editable .
```

```
Obtaining file:///content/torchquantum
Requirement already satisfied: numpy>=1.19.2 in /usr/local/lib/python3.7/dist-packages␣
↪(from torchquantum==0.1.0) (1.21.5)
Requirement already satisfied: torchvision>=0.9.0.dev20210130 in /usr/local/lib/python3.
↪7/dist-packages (from torchquantum==0.1.0) (0.11.1+cu111)
Requirement already satisfied: tqdm>=4.56.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from torchquantum==0.1.0) (4.62.3)
Requirement already satisfied: setuptools>=52.0.0 in /usr/local/lib/python3.7/dist-
↪packages (from torchquantum==0.1.0) (57.4.0)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from torchquantum==0.1.0) (1.10.0+cu111)
Collecting torchpack>=0.3.0
  Downloading torchpack-0.3.1-py3-none-any.whl (34 kB)
Requirement already satisfied: qiskit>=0.32.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from torchquantum==0.1.0) (0.32.1)
Collecting matplotlib>=3.3.2
  Downloading matplotlib-3.5.1-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (11.
↪2 MB)
     || 11.2 MB 6.5 MB/s
Collecting pathos>=0.2.7
  Downloading pathos-0.2.8-py2.py3-none-any.whl (81 kB)
     || 81 kB 12.1 MB/s
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.7/dist-
↪packages (from matplotlib>=3.3.2->torchquantum==0.1.0) (2.8.2)
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.7/dist-
↪packages (from matplotlib>=3.3.2->torchquantum==0.1.0) (3.0.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages␣
↪(from matplotlib>=3.3.2->torchquantum==0.1.0) (0.11.0)
Collecting fonttools>=4.22.0
  Downloading fonttools-4.29.1-py3-none-any.whl (895 kB)
     || 895 kB 55.2 MB/s
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from matplotlib>=3.3.2->torchquantum==0.1.0) (7.1.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-
↪packages (from matplotlib>=3.3.2->torchquantum==0.1.0) (1.3.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from matplotlib>=3.3.2->torchquantum==0.1.0) (21.3)
Collecting ppft>=1.6.6.4
  Downloading ppft-1.6.6.4-py3-none-any.whl (65 kB)
     || 65 kB 4.1 MB/s
Collecting pox>=0.3.0
  Downloading pox-0.3.0-py2.py3-none-any.whl (30 kB)
Requirement already satisfied: multiprocess>=0.70.12 in /usr/local/lib/python3.7/dist-
↪packages (from pathos>=0.2.7->torchquantum==0.1.0) (0.70.12.2)
Requirement already satisfied: dill>=0.3.4 in /usr/local/lib/python3.7/dist-packages␣
↪(from pathos>=0.2.7->torchquantum==0.1.0) (0.3.4)
Requirement already satisfied: six>=1.7.3 in /usr/local/lib/python3.7/dist-packages␣
↪(from ppft>=1.6.6.4->pathos>=0.2.7->torchquantum==0.1.0) (1.15.0)
Requirement already satisfied: qiskit-ibmq-provider==0.18.1 in /usr/local/lib/python3.7/
↪dist-packages (from qiskit>=0.32.0->torchquantum==0.1.0) (0.18.1)
Requirement already satisfied: qiskit-aqua==0.9.5 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit>=0.32.0->torchquantum==0.1.0) (0.9.5)
Requirement already satisfied: qiskit-aer==0.9.1 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit>=0.32.0->torchquantum==0.1.0) (0.9.1)
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: qiskit-ignis==0.6.0 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit>=0.32.0->torchquantum==0.1.0) (0.6.0)
Requirement already satisfied: qiskit-terra==0.18.3 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit>=0.32.0->torchquantum==0.1.0) (0.18.3)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aer==0.9.1->qiskit>=0.32.0->torchquantum==0.1.0) (1.4.1)
Requirement already satisfied: quandl in /usr/local/lib/python3.7/dist-packages (from␣
↪qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (3.7.0)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (1.0.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from␣
↪qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (1.3.5)
Requirement already satisfied: h5py<3.3.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (3.1.0)
Requirement already satisfied: fastdtw<=0.3.4 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (0.3.4)
Requirement already satisfied: dlx<=1.0.4 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (1.0.4)
Requirement already satisfied: retworkx>=0.8.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (0.11.0)
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (1.7.1)
Requirement already satisfied: docplex>=2.21.207 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (2.22.213)
Requirement already satisfied: yfinance>=0.1.62 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (0.1.70)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.7/dist-packages (from␣
↪qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (5.4.8)
Requirement already satisfied: urllib3>=1.21.1 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-ibmq-provider==0.18.1->qiskit>=0.32.0->torchquantum==0.1.0) (1.24.3)
Requirement already satisfied: requests-ntlm>=1.1.0 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-ibmq-provider==0.18.1->qiskit>=0.32.0->torchquantum==0.1.0) (1.1.
↪0)
Requirement already satisfied: websocket-client>=1.0.1 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-ibmq-provider==0.18.1->qiskit>=0.32.0->torchquantum==0.1.0) (1.2.
↪3)
Requirement already satisfied: requests>=2.19 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-ibmq-provider==0.18.1->qiskit>=0.32.0->torchquantum==0.1.0) (2.27.1)
Requirement already satisfied: symengine>0.7 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-terra==0.18.3->qiskit>=0.32.0->torchquantum==0.1.0) (0.8.1)
Requirement already satisfied: fastjsonschema>=2.10 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-terra==0.18.3->qiskit>=0.32.0->torchquantum==0.1.0) (2.15.3)
Requirement already satisfied: ply>=3.10 in /usr/local/lib/python3.7/dist-packages (from␣
↪qiskit-terra==0.18.3->qiskit>=0.32.0->torchquantum==0.1.0) (3.11)
Requirement already satisfied: python-constraint>=1.4 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-terra==0.18.3->qiskit>=0.32.0->torchquantum==0.1.0) (1.4.0)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.7/dist-packages␣
↪(from qiskit-terra==0.18.3->qiskit>=0.32.0->torchquantum==0.1.0) (4.3.3)
Requirement already satisfied: tweedledum<2.0,>=1.1 in /usr/local/lib/python3.7/dist-
↪packages (from qiskit-terra==0.18.3->qiskit>=0.32.0->torchquantum==0.1.0) (1.1.1)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages␣
↪(from h5py<3.3.0->qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (1.5.2)
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from jsonschema>=2.6->qiskit-terra==0.18.3->qiskit>=0.32.0->torchquantum==0.1.0) (21.
↪4.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-
↪packages (from jsonschema>=2.6->qiskit-terra==0.18.3->qiskit>=0.32.0->torchquantum==0.
↪1.0) (4.11.0)
Requirement already satisfied: importlib-resources>=1.4.0 in /usr/local/lib/python3.7/
↪dist-packages (from jsonschema>=2.6->qiskit-terra==0.18.3->qiskit>=0.32.0->
↪torchquantum==0.1.0) (5.4.0)
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/
↪local/lib/python3.7/dist-packages (from jsonschema>=2.6->qiskit-terra==0.18.3->qiskit>
↪=0.32.0->torchquantum==0.1.0) (0.18.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-
↪packages (from jsonschema>=2.6->qiskit-terra==0.18.3->qiskit>=0.32.0->torchquantum==0.
↪1.0) (3.10.0.2)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from importlib-resources>=1.4.0->jsonschema>=2.6->qiskit-terra==0.18.3->qiskit>=0.32.
↪0->torchquantum==0.1.0) (3.7.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.7/dist-packages␣
↪(from requests>=2.19->qiskit-ibmq-provider==0.18.1->qiskit>=0.32.0->torchquantum==0.1.
↪0) (2.10)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.7/
↪dist-packages (from requests>=2.19->qiskit-ibmq-provider==0.18.1->qiskit>=0.32.0->
↪torchquantum==0.1.0) (2.0.11)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-
↪packages (from requests>=2.19->qiskit-ibmq-provider==0.18.1->qiskit>=0.32.0->
↪torchquantum==0.1.0) (2021.10.8)
Requirement already satisfied: cryptography>=1.3 in /usr/local/lib/python3.7/dist-
↪packages (from requests-ntlm>=1.1.0->qiskit-ibmq-provider==0.18.1->qiskit>=0.32.0->
↪torchquantum==0.1.0) (36.0.1)
Requirement already satisfied: ntlm-auth>=1.0.2 in /usr/local/lib/python3.7/dist-
↪packages (from requests-ntlm>=1.1.0->qiskit-ibmq-provider==0.18.1->qiskit>=0.32.0->
↪torchquantum==0.1.0) (1.5.0)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.7/dist-packages␣
↪(from cryptography>=1.3->requests-ntlm>=1.1.0->qiskit-ibmq-provider==0.18.1->qiskit>=0.
↪32.0->torchquantum==0.1.0) (1.15.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.7/dist-packages (from␣
↪cffi>=1.12->cryptography>=1.3->requests-ntlm>=1.1.0->qiskit-ibmq-provider==0.18.1->
↪qiskit>=0.32.0->torchquantum==0.1.0) (2.21)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages␣
↪(from scikit-learn>=0.20.0->qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0)␣
↪(1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
↪packages (from scikit-learn>=0.20.0->qiskit-aqua==0.9.5->qiskit>=0.32.0->
↪torchquantum==0.1.0) (3.1.0)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.7/dist-packages␣
↪(from sympy>=1.3->qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (1.2.1)
Requirement already satisfied: tensorboard in /usr/local/lib/python3.7/dist-packages␣
↪(from torchpack>=0.3.0->torchquantum==0.1.0) (2.8.0)
Collecting toml
  Downloading toml-0.10.2-py2.py3-none-any.whl (16 kB)
Collecting tensorpack
```

(continues on next page)

(continued from previous page)

```
  Downloading tensorpack-0.11-py2.py3-none-any.whl (296 kB)
     || 296 kB 57.1 MB/s
Collecting multimethod
  Downloading multimethod-1.7-py3-none-any.whl (9.5 kB)
Collecting loguru
  Downloading loguru-0.6.0-py3-none-any.whl (58 kB)
     || 58 kB 2.6 MB/s
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (from␣
↪torchpack>=0.3.0->torchquantum==0.1.0) (3.13)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.7/dist-
↪packages (from yfinance>=0.1.62->qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.
↪0) (0.0.10)
Requirement already satisfied: lxml>=4.5.1 in /usr/local/lib/python3.7/dist-packages␣
↪(from yfinance>=0.1.62->qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (4.7.
↪1)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages␣
↪(from pandas->qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (2018.9)
Requirement already satisfied: more-itertools in /usr/local/lib/python3.7/dist-packages␣
↪(from quandl->qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (8.12.0)
Requirement already satisfied: inflection>=0.3.1 in /usr/local/lib/python3.7/dist-
↪packages (from quandl->qiskit-aqua==0.9.5->qiskit>=0.32.0->torchquantum==0.1.0) (0.5.1)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages␣
↪(from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (3.3.6)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/
↪dist-packages (from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (1.8.1)
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.7/dist-packages␣
↪(from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (1.0.0)
Requirement already satisfied: grpcio>=1.24.3 in /usr/local/lib/python3.7/dist-packages␣
↪(from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (1.43.0)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.7/dist-packages␣
↪(from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (0.37.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/
↪python3.7/dist-packages (from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (0.4.
↪6)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-
↪packages (from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (1.35.0)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/
↪python3.7/dist-packages (from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (0.6.
↪1)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-
↪packages (from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (1.0.1)
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (3.17.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-
↪packages (from google-auth<3,>=1.6.3->tensorboard->torchpack>=0.3.0->torchquantum==0.1.
↪0) (0.2.8)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-
↪packages (from google-auth<3,>=1.6.3->tensorboard->torchpack>=0.3.0->torchquantum==0.1.
↪0) (4.2.4)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages␣
↪(from google-auth<3,>=1.6.3->tensorboard->torchpack>=0.3.0->torchquantum==0.1.0) (4.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-
↪packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard->torchpack>=0.3.0->
↪torchquantum==0.1.0) (1.3.1)
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-
↪packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard->torchpack>=0.
↪3.0->torchquantum==0.1.0) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages␣
↪(from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard->
↪torchpack>=0.3.0->torchquantum==0.1.0) (3.2.0)
Requirement already satisfied: msgpack>=0.5.2 in /usr/local/lib/python3.7/dist-packages␣
↪(from tensorpack->torchpack>=0.3.0->torchquantum==0.1.0) (1.0.3)
Requirement already satisfied: pyzmq>=16 in /usr/local/lib/python3.7/dist-packages (from␣
↪tensorpack->torchpack>=0.3.0->torchquantum==0.1.0) (22.3.0)
Collecting msgpack-numpy>=0.4.4.2
  Downloading msgpack_numpy-0.4.7.1-py2.py3-none-any.whl (6.7 kB)
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.7/dist-packages␣
↪(from tensorpack->torchpack>=0.3.0->torchquantum==0.1.0) (0.8.9)
Requirement already satisfied: termcolor>=1.1 in /usr/local/lib/python3.7/dist-packages␣
↪(from tensorpack->torchpack>=0.3.0->torchquantum==0.1.0) (1.1.0)
Installing collected packages: msgpack-numpy, toml, tensorpack, ppft, pox, multimethod,␣
↪loguru, fonttools, torchpack, pathos, matplotlib, torchquantum
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.2.2
    Uninstalling matplotlib-3.2.2:
      Successfully uninstalled matplotlib-3.2.2
  Running setup.py develop for torchquantum
ERROR: pip's dependency resolver does not currently take into account all the packages␣
↪that are installed. This behaviour is the source of the following dependency conflicts.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is␣
↪incompatible.
Successfully installed fonttools-4.29.1 loguru-0.6.0 matplotlib-3.5.1 msgpack-numpy-0.4.
↪7.1 multimethod-1.7 pathos-0.2.8 pox-0.3.0 ppft-1.6.6.4 tensorpack-0.11 toml-0.10.2␣
↪torchpack-0.3.1 torchquantum-0.1.0
```

Data type cannot be displayed: application/vnd.colab-display-data+json

Change PYTHONPATH and install other packages.

[ ]:
```
%env PYTHONPATH=.
```

```
env: PYTHONPATH=.
```

Run the following code to store a qiskit token. You can replace it with your own token from your IBMQ account if you like.

[ ]:
```
from qiskit import IBMQ
IBMQ.save_account(
↪'0238b0afc0dc515fe7987b02706791d1719cb89b68befedc125eded0607e6e9e9f26d3eed482f66fdc45fdfceca3aab2edb9
↪', overwrite=True)
```

[ ]:
```
!pip install matplotlib==3.1.3
```

```
Collecting matplotlib==3.1.3
  Downloading matplotlib-3.1.3-cp37-cp37m-manylinux1_x86_64.whl (13.1 MB)
     || 13.1 MB 4.3 MB/s
```

(continues on next page)

(continued from previous page)

```
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/
↪lib/python3.7/dist-packages (from matplotlib==3.1.3) (3.0.7)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages␣
↪(from matplotlib==3.1.3) (1.21.5)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages␣
↪(from matplotlib==3.1.3) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-
↪packages (from matplotlib==3.1.3) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-
↪packages (from matplotlib==3.1.3) (1.3.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from␣
↪python-dateutil>=2.1->matplotlib==3.1.3) (1.15.0)
Installing collected packages: matplotlib
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.5.1
    Uninstalling matplotlib-3.5.1:
      Successfully uninstalled matplotlib-3.5.1
ERROR: pip's dependency resolver does not currently take into account all the packages␣
↪that are installed. This behaviour is the source of the following dependency conflicts.
torchquantum 0.1.0 requires matplotlib>=3.3.2, but you have matplotlib 3.1.3 which is␣
↪incompatible.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is␣
↪incompatible.
Successfully installed matplotlib-3.1.3
```

Data type cannot be displayed: application/vnd.colab-display-data+json

[ ]:
```
!ls artifact
```

```
aerbackend.py  example2  example4  example6  README.md
example1       example3  example5  example7
```

[ ]:
```
!cp artifact/aerbackend.py ../../usr/local/lib/python3.7/dist-packages/qiskit/providers/
↪aer/backends/ -r
```

Step

Our code requires torchquantum lib, mnist dataset, pytorch and numpy. We need torch and the logsoftmax function from `torch.nn.functional`, optimizers(`optim`), `torchquantum` module, MNIST dataset(`MNIST`), cosine annealing learning rate(`CosineAnnealingLR`).

[ ]:
```
import torch
import torch.nn.functional as F
import torch.optim as optim
import numpy as np

import torchquantum as tq
import random
```

(continues on next page)

(continued from previous page)

```
from examples.core.datasets import MNIST
from torch.optim.lr_scheduler import CosineAnnealingLR
```

Build a quanvolutional filter

Our quanvolution model is a hybrid model. It consists of two parts, the quanvolutional filter part and the classical layer part. To build the model, firstly we define our quanvolutional filter.

Our quanvolutional filter's structure is the same as the figure described above. It has four qubits. The `tq.QuantumDevice` module stores the state vector. Usually a Quantum Neural Network module consists of three parts: encoder, ansatz and measurement. We can create an encoder by passing a list of gates to `tq.GeneralEncoder`. Each entry in the list contains `input_idx`, `func`, and `wires`. Here, each qubit has a rotation-Y gate. 4 RY gates in total. They can encode the 2x2 input data to the quantum state. Then we decide our ansatz to be a random layer. We call `tq.RandomLayer` to create an ansatz composed by 8 basic gates with no more than 8 trainable parameters. And finally we perform Pauli-Z measurements on each qubit by creating a `tq.MeasureAll` module and passing `tq.PauliZ` to it. The measure function will return four expectation values from four qubits. The four results go to four channels.

Next look at how quanvolutional filter works. We get the batch size. Our image is 28x28. So we reshape our input data to `(bsz, 28, 28)`.

We initialize the `data_list`. The list stores the outputs in each stride.

The double loop is to iterate all the possible positions that the quanvolutional filter window may stride in. Here the stride is 2.

Then we catenate the data in the 2x2 window. Here we catenate four lists to one big list, so we need to reshape the list to `(4, bsz)` and transpose it to `(bsz, 4)`.

Next if you want to use qiskit's remote noise model or real quantum machine, you can set `use_qiskit=True` and pass these 5 parameters: `q_device`, `encoder`, `q_layer`, `measure`, and `data`. The `qiskit_processor` will receive these parameters, put the data in the encoder, run the while circuits and return the measurement result. Remember only when the model is doing an inference can you use qiskit remote.

If you are training or not using qiskit remote, you can run the three parts one by one on google colab's GPU.

After each stride, we append the measurement result to `data_list`.

Finally, we catenate the `data_list` along dimension 1 and return the result.

[ ]:
```
class QuanvolutionFilter(tq.QuantumModule):
    def __init__(self):
        super().__init__()
        self.n_wires = 4
        self.q_device = tq.QuantumDevice(n_wires=self.n_wires)
        self.encoder = tq.GeneralEncoder(
        [   {'input_idx': [0], 'func': 'ry', 'wires': [0]},
            {'input_idx': [1], 'func': 'ry', 'wires': [1]},
            {'input_idx': [2], 'func': 'ry', 'wires': [2]},
            {'input_idx': [3], 'func': 'ry', 'wires': [3]},])

        self.q_layer = tq.RandomLayer(n_ops=8, wires=list(range(self.n_wires)))
        self.measure = tq.MeasureAll(tq.PauliZ)

    def forward(self, x, use_qiskit=False):
        bsz = x.shape[0]
```

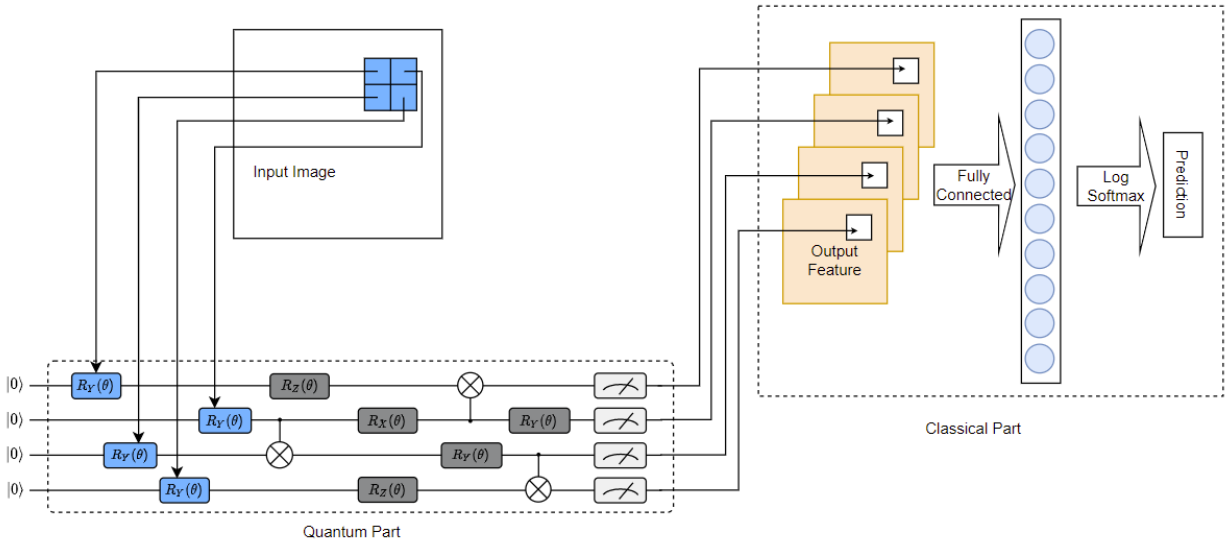
(continues on next page)

(continued from previous page)

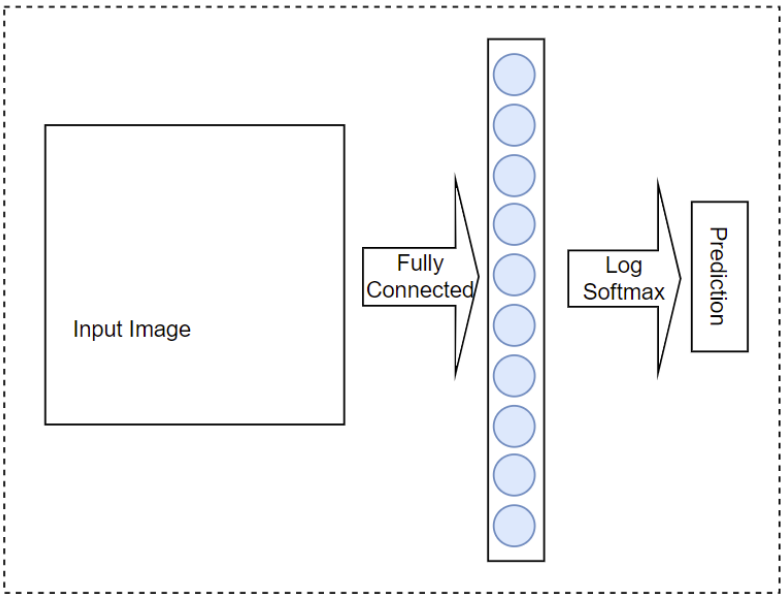
```
        size = 28
        x = x.view(bsz, size, size)

        data_list = []

        for c in range(0, size, 2):
            for r in range(0, size, 2):
                data = torch.transpose(torch.cat((x[:, c, r], x[:, c, r+1], x[:, c+1, r],
↪ x[:, c+1, r+1])).view(4, bsz), 0, 1)
                if use_qiskit:
                    data = self.qiskit_processor.process_parameterized(
                        self.q_device, self.encoder, self.q_layer, self.measure, data)
                else:
                    self.encoder(self.q_device, data)
                    self.q_layer(self.q_device)
                    data = self.measure(self.q_device)

                data_list.append(data.view(bsz, 4))

        result = torch.cat(data_list, dim=1).float()

        return result
```

Build the whole hybrid model.

Then we look at the whole model. The whole model consists of a `QuanvolutionFilter` and full connect layer(`torch.nn.Linear`). The size of input is 4*14*14 because a 28x28 image after quanvolutional filter turns into a 4 channel 14x14 feature. As the task is MNIST 10 digits classification, the size of output is 10. At last the model perform `F.logsoftmax` to the result for classification.



Here, we also has a model without quanvolutional filters used for comparison. Its full connect layer's input size is simple 28x28.



Classical Model

[ ]:
```
class HybridModel(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.qf = QuanvolutionFilter()
        self.linear = torch.nn.Linear(4*14*14, 10)

    def forward(self, x, use_qiskit=False):
        with torch.no_grad():
          x = self.qf(x, use_qiskit)
        x = self.linear(x)
        return F.log_softmax(x, -1)

class HybridModel_without_qf(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = torch.nn.Linear(28*28, 10)

    def forward(self, x, use_qiskit=False):
        x = x.view(-1, 28*28)
        x = self.linear(x)
        return F.log_softmax(x, -1)
```

Load the dataset MNIST

We use MNIST classification dataset(10 digits and 1000 training samples).

The `root` is the folder that stores the dataset. If there's no MNIST dataset in root, it will automatically download MNIST. Next, we set the `train_valid_split_ratio`, `n_test_samples`, and `n_train_samples`.

The dataset now contains three splits, 'train', 'valid' and 'test'. For each split, we create a dataloader with a random sampler, `batch_size` is 10, `num_workers` is 8 and `pin_memory` is true.

[ ]:
```
random.seed(42)
np.random.seed(42)
torch.manual_seed(42)
dataset = MNIST(
    root='./mnist_data',
    train_valid_split_ratio=[0.9, 0.1],
    n_test_samples=300,
    n_train_samples=500,
)
dataflow = dict()

for split in dataset:
    sampler = torch.utils.data.RandomSampler(dataset[split])
    dataflow[split] = torch.utils.data.DataLoader(
        dataset[split],
        batch_size=10,
        sampler=sampler,
        num_workers=8,
        pin_memory=True)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./mnist_data/
↪MNIST/raw/train-images-idx3-ubyte.gz
```

```
  0%|          | 0/9912422 [00:00<?, ?it/s]
```

```
Extracting ./mnist_data/MNIST/raw/train-images-idx3-ubyte.gz to ./mnist_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./mnist_data/
↪MNIST/raw/train-labels-idx1-ubyte.gz
```

```
  0%|          | 0/28881 [00:00<?, ?it/s]
```

```
Extracting ./mnist_data/MNIST/raw/train-labels-idx1-ubyte.gz to ./mnist_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./mnist_data/
↪MNIST/raw/t10k-images-idx3-ubyte.gz
```

```
  0%|          | 0/1648877 [00:00<?, ?it/s]
```

```
Extracting ./mnist_data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./mnist_data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./mnist_data/
↪MNIST/raw/t10k-labels-idx1-ubyte.gz
```

```
  0%|          | 0/4542 [00:00<?, ?it/s]
[2022-02-16 04:00:40.771] Only use the front 500 images as TRAIN set.
Extracting ./mnist_data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./mnist_data/MNIST/raw

[2022-02-16 04:00:40.868] Only use the front 300 images as TEST set.
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning:␣
↪This DataLoader will create 8 worker processes in total. Our suggested max number of␣
↪worker in current system is 2, which is smaller than what this DataLoader is going to␣
↪create. Please be aware that excessive worker creation might get DataLoader running␣
↪slow or even freeze, lower the worker number to avoid potential slowness/freeze if␣
↪necessary.
  cpuset_checked))
```

Then we set use_cuda, it depends on whether cuda is available.

Create a device.

Initialize the model, `n_epochs` to 15, Adam optimizer and cosine annealing learning rate scheduler.

[ ]:
```
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
model = HybridModel().to(device)
model_without_qf = HybridModel_without_qf().to(device)
n_epochs = 15
optimizer = optim.Adam(model.parameters(), lr=5e-3, weight_decay=1e-4)
scheduler = CosineAnnealingLR(optimizer, T_max=n_epochs)
```

Train the model.

When training the model, we iterate the dataloader. Get the `inputs` and `targets` data. Feed `inputs` to the model and get `outputs`. Calculate the negative loss likelihood loss(`F.nll_loss`). Reset all the gradients of parameters in the model to zero. Call `loss.backward()` to perform backpropagation. Call `optimizer.step()` to update all the parameters.

After each epoch, we will valid the model. In validation, we can use qiskit remote because we don't need to calculate gradients.

[ ]:
```
accu_list1 = []
loss_list1 = []
accu_list2 = []
loss_list2 = []

def train(dataflow, model, device, optimizer):
    for feed_dict in dataflow['train']:
        inputs = feed_dict['image'].to(device)
        targets = feed_dict['digit'].to(device)

        outputs = model(inputs)
        loss = F.nll_loss(outputs, targets)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

(continues on next page)

(continued from previous page)

```
        print(f"loss: {loss.item()}", end='\r')


def valid_test(dataflow, split, model, device, qiskit=False):
    target_all = []
    output_all = []
    with torch.no_grad():
        for feed_dict in dataflow[split]:
            inputs = feed_dict['image'].to(device)
            targets = feed_dict['digit'].to(device)

            outputs = model(inputs, use_qiskit=qiskit)

            target_all.append(targets)
            output_all.append(outputs)
        target_all = torch.cat(target_all, dim=0)
        output_all = torch.cat(output_all, dim=0)

    _, indices = output_all.topk(1, dim=1)
    masks = indices.eq(target_all.view(-1, 1).expand_as(indices))
    size = target_all.shape[0]
    corrects = masks.sum().item()
    accuracy = corrects / size
    loss = F.nll_loss(output_all, target_all).item()

    print(f"{split} set accuracy: {accuracy}")
    print(f"{split} set loss: {loss}")

    return accuracy, loss

for epoch in range(1, n_epochs + 1):
    # train
    print(f"Epoch {epoch}:")
    train(dataflow, model, device, optimizer)
    print(optimizer.param_groups[0]['lr'])

    # valid
    accu, loss = valid_test(dataflow, 'test', model, device, )
    accu_list1.append(accu)
    loss_list1.append(loss)
    scheduler.step()
```

```
Epoch 1:
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning:␣
↪This DataLoader will create 8 worker processes in total. Our suggested max number of␣
↪worker in current system is 2, which is smaller than what this DataLoader is going to␣
↪create. Please be aware that excessive worker creation might get DataLoader running␣
↪slow or even freeze, lower the worker number to avoid potential slowness/freeze if␣
↪necessary.
  cpuset_checked))
```

```
0.005
test set accuracy: 0.8066666666666666
test set loss: 0.6323180794715881
Epoch 2:
0.004945369001834514
test set accuracy: 0.7766666666666666
test set loss: 0.5900668501853943
Epoch 3:
0.004783863644106502
test set accuracy: 0.8466666666666667
test set loss: 0.48249581456184387
Epoch 4:
0.0045225424859373685
test set accuracy: 0.8133333333333334
test set loss: 0.5225163698196411
Epoch 5:
0.0041728265158971455
test set accuracy: 0.8033333333333333
test set loss: 0.6009621620178223
Epoch 6:
0.00375
test set accuracy: 0.8333333333333334
test set loss: 0.44394049048423767
Epoch 7:
0.0032725424859373687
test set accuracy: 0.84
test set loss: 0.4330306053161621
Epoch 8:
0.002761321158169134
test set accuracy: 0.8366666666666667
test set loss: 0.45171523094177246
Epoch 9:
0.002238678841830867
test set accuracy: 0.8633333333333333
test set loss: 0.4244077205657959
Epoch 10:
0.00172745751406263 2
test set accuracy: 0.8633333333333333
test set loss: 0.40085339546203613
Epoch 11:
0.0012500000000000007
test set accuracy: 0.8533333333333334
test set loss: 0.40397733449935913
Epoch 12:
0.0008271734841028553
test set accuracy: 0.87
test set loss: 0.3975270986557007
Epoch 13:
0.0004774575140626316 3
test set accuracy: 0.8566666666666667
test set loss: 0.4006715416908264
Epoch 14:
0.00021613635589349755
```

(continues on next page)

(continued from previous page)

```
test set accuracy: 0.8666666666666667
test set loss: 0.39790403842926025
Epoch 15:
5.463099816548578e-05
test set accuracy: 0.8666666666666667
test set loss: 0.3979119062423706
```

Train the model without quanvolutional filters.

[ ]:
```
optimizer = optim.Adam(model_without_qf.parameters(), lr=5e-3, weight_decay=1e-4)
scheduler = CosineAnnealingLR(optimizer, T_max=n_epochs)
for epoch in range(1, n_epochs + 1):
    # train
    print(f"Epoch {epoch}:")
    train(dataflow, model_without_qf, device, optimizer)
    print(optimizer.param_groups[0]['lr'])

    # valid
    accu, loss = valid_test(dataflow, 'test', model_without_qf, device)
    accu_list2.append(accu)
    loss_list2.append(loss)

    scheduler.step()
```

```
Epoch 1:
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning:␣
↪This DataLoader will create 8 worker processes in total. Our suggested max number of␣
↪worker in current system is 2, which is smaller than what this DataLoader is going to␣
↪create. Please be aware that excessive worker creation might get DataLoader running␣
↪slow or even freeze, lower the worker number to avoid potential slowness/freeze if␣
↪necessary.
  cpuset_checked))
0.005
test set accuracy: 0.7733333333333333
test set loss: 0.6043258905410767
Epoch 2:
0.004945369001834514
test set accuracy: 0.8166666666666667
test set loss: 0.5571645498275757
Epoch 3:
0.004783863644106502
test set accuracy: 0.8466666666666667
test set loss: 0.46128183603286743
Epoch 4:
0.0045225424859373685
test set accuracy: 0.8366666666666667
test set loss: 0.5158915519714355
Epoch 5:
0.004172826515897145 5
test set accuracy: 0.8666666666666667
test set loss: 0.45338067412376404
Epoch 6:
```

(continues on next page)

(continued from previous page)

```
0.00375
test set accuracy: 0.8466666666666667
test set loss: 0.4563254714012146
Epoch 7:
0.0032725424859373687
test set accuracy: 0.8566666666666667
test set loss: 0.4633018374443054
Epoch 8:
0.002761321158169134
test set accuracy: 0.86
test set loss: 0.46147480607032776
Epoch 9:
0.002238678841830867
test set accuracy: 0.85
test set loss: 0.45319321751594543
Epoch 10:
0.00172745751406263
test set accuracy: 0.84
test set loss: 0.46221110224723816
Epoch 11:
0.0012500000000000007
test set accuracy: 0.8533333333333334
test set loss: 0.4611275792121887
Epoch 12:
0.0008271734841028553
test set accuracy: 0.8533333333333334
test set loss: 0.4614029824733734
Epoch 13:
0.0004774575140626316
test set accuracy: 0.8533333333333334
test set loss: 0.4610340893268585
Epoch 14:
0.00021613635589349755
test set accuracy: 0.8533333333333334
test set loss: 0.46056315302848816
Epoch 15:
5.463099816548578e-05
test set accuracy: 0.8533333333333334
test set loss: 0.4606676697731018
```

Compare Quanvolutional Neural Network with classical model.

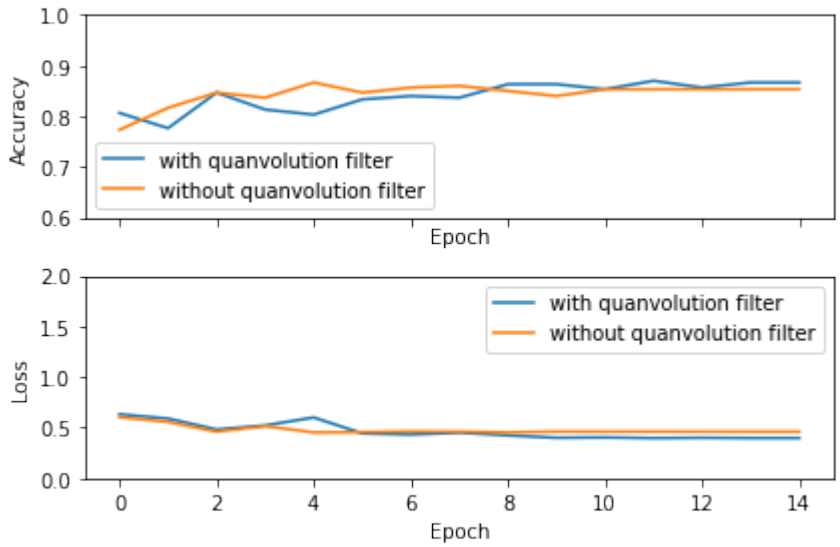
After training, we can plot the accuracy and loss curve. We can see that model with quanvolutional filter can achieve slightly higher accuracy than model without quanvolution.

[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import matplotlib

fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
```

(continues on next page)

(continued from previous page)

```
ax1.plot(accu_list1, label="with quanvolution filter")
ax1.plot(accu_list2, label="without quanvolution filter")
ax1.set_ylabel("Accuracy")
ax1.set_ylim([0.6, 1])
ax1.set_xlabel("Epoch")
ax1.legend()

ax2.plot(loss_list1, label="with quanvolution filter")
ax2.plot(loss_list2, label="without quanvolution filter")
ax2.set_ylabel("Loss")
ax2.set_ylim([0, 2])
ax2.set_xlabel("Epoch")
ax2.legend()
plt.tight_layout()
plt.show()
```



Here we can also see the image before quanvolutional filter and after quanvolutional filter.

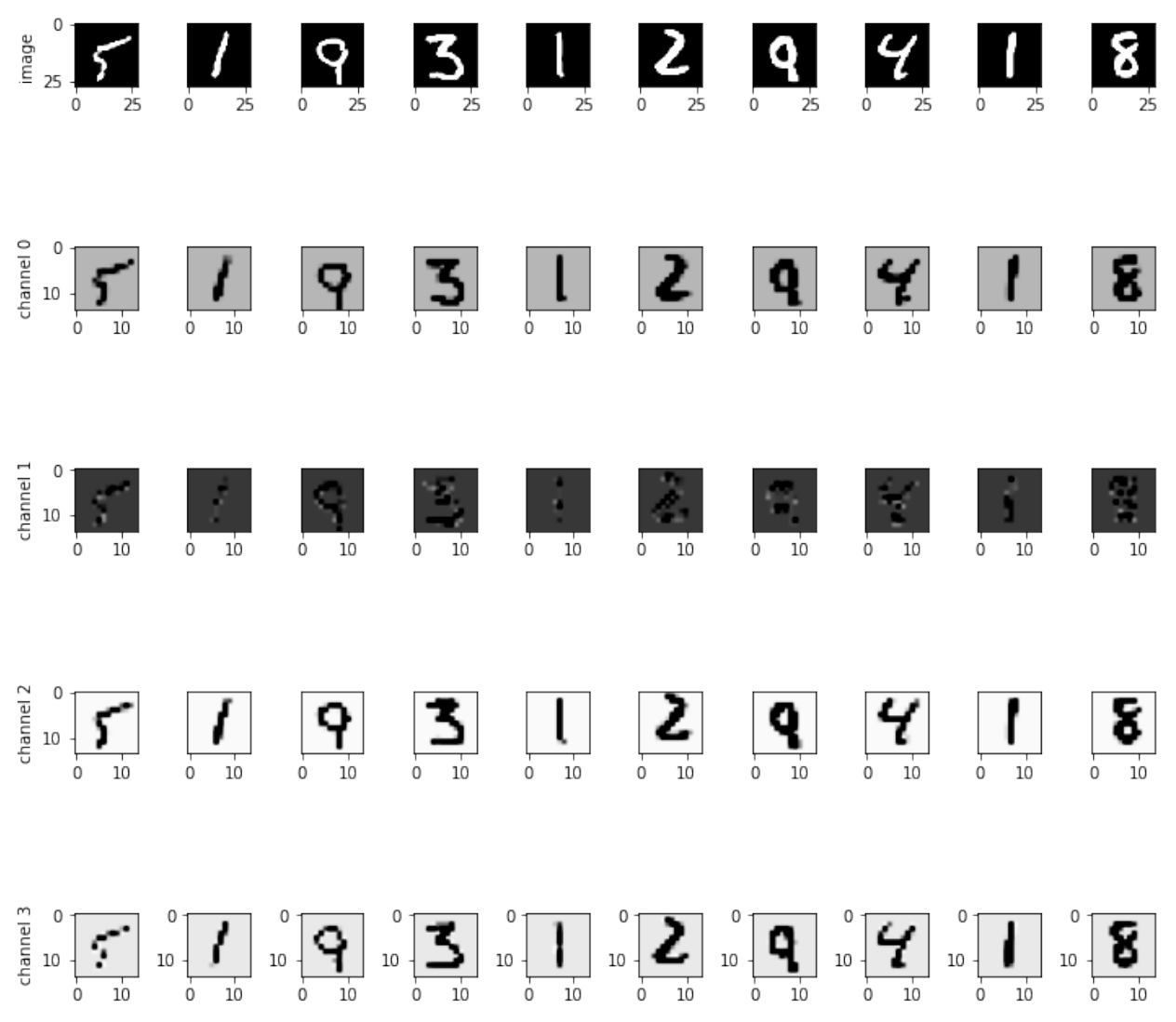
[ ]:
```
import matplotlib.pyplot as plt
import matplotlib

n_samples = 10
n_channels = 4
for feed_dict in dataflow['test']:
  inputs = feed_dict['image'].to(device)
  break
sample = inputs[:n_samples]
after_quanv = model.qf(sample).view(n_samples, 14*14, 4).cpu().detach().numpy()

fig, axes = plt.subplots(1 + n_channels, n_samples, figsize=(10, 10))
for k in range(n_samples):
    axes[0, 0].set_ylabel("image")
```

(continues on next page)

(continued from previous page)

```
    if k != 0:
        axes[0, k].yaxis.set_visible(False)

    norm = matplotlib.colors.Normalize(vmin=0, vmax=1)

    axes[0, k].imshow(sample[k, 0, :, :].cpu(), norm=norm, cmap="gray")

    for c in range(n_channels):
        axes[c + 1, 0].set_ylabel("channel {}".format(c))
        if k != 0:
            axes[c, k].yaxis.set_visible(False)
        axes[c + 1, k].imshow(after_quanv[k, :, c].reshape(14, 14), norm=norm, cmap="gray
↪")

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:481: UserWarning:␣
↪This DataLoader will create 8 worker processes in total. Our suggested max number of␣
↪worker in current system is 2, which is smaller than what this DataLoader is going to␣
↪create. Please be aware that excessive worker creation might get DataLoader running␣
↪slow or even freeze, lower the worker number to avoid potential slowness/freeze if␣
↪necessary.
  cpuset_checked))
```



Evaluate on real quantum computer.

At last, we can run our quanvolutional filter on IBMQ's real quantum machine. The process is really slow so I will not show it here. If you have higher priority access to IBMQ qiskit, you can check the code cell in installation and replace our token with your advance token. That will make the process faster.

[ ]:
```
# test
valid_test(dataflow, 'test', model, device, qiskit=False)

# run on Qiskit simulator and real Quantum Computers
try:
    from qiskit import IBMQ
    from torchquantum.plugins import QiskitProcessor
    # firstly perform simulate
    print(f"\nTest with Qiskit Simulator")
    processor_simulation = QiskitProcessor(use_real_qc=False)
    model.qf.set_qiskit_processor(processor_simulation)
```

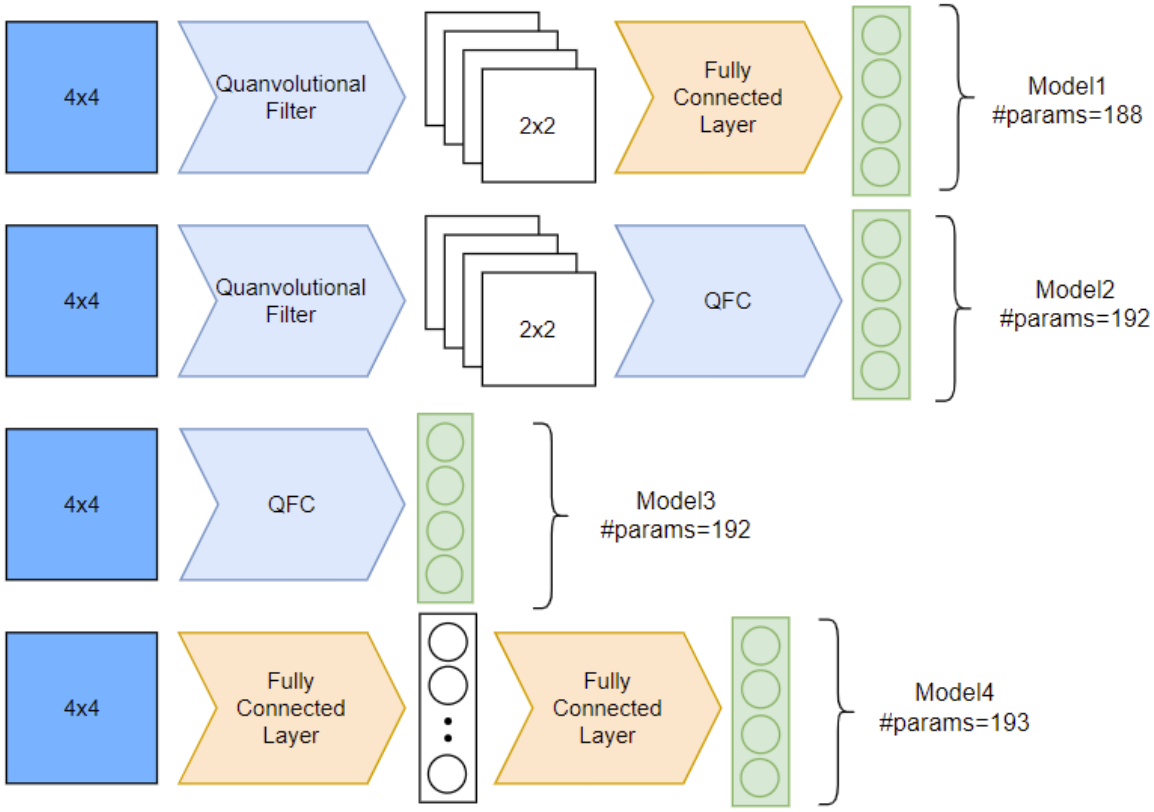
(continues on next page)

(continued from previous page)

```
    valid_test(dataflow, 'test', model, device, qiskit=True)
    # then try to run on REAL QC
    backend_name = 'ibmq_quito'
    print(f"\nTest on Real Quantum Computer {backend_name}")
    processor_real_qc = QiskitProcessor(use_real_qc=True,backend_name=backend_name)
    model.qf.set_qiskit_processor(processor_real_qc)
    valid_test(dataflow, 'test', model, device, qiskit=True)
except ImportError:
    print("Please install qiskit, create an IBM Q Experience Account and "
          "save the account token according to the instruction at "
          "'https://github.com/Qiskit/qiskit-ibmq-provider', "
          "then try again.")
```

Trainable Quanvolutional Filter

In this section, we consider the case that quanvolutional filters are trainable, and we compare various models with nearly the same number of trainale parameters. The four model compared here are described by the following figure.



The Model1 contains a trainable quanvolutional filter and a fully connected layer.

The Model2 contains a trainable quanvolutional filter and a quantum fully connected layer. We use `U3CU3Layer0` from `torchquantum.layers` to implement the QFC layer.

When building the ansatz part of the QFC, we need to pass a dict describing the architecture of the ansatz. Here the dict is `{'n_wires': self.n_wires, 'n_blocks': 4, 'n_layers_per_block': 2}`, which means the ansatz

contains n_wires qubits, there are 4 blocks and in each block are 2 layers. Passing the arch to `U3CU3Layer0` we will get a trainable ansatz with 4 blocks and in each block contains 4 U3 gates followed by 4 CU3 gates.

The Model3 is simply a QFC layer.

The Model4 is two fully connected layers.

[ ]:
```
from torchquantum.encoding import encoder_op_list_name_dict
from torchquantum.layers import U3CU3Layer0

class TrainableQuanvFilter(tq.QuantumModule):
    def __init__(self):
        super().__init__()
        self.n_wires = 4
        self.q_device = tq.QuantumDevice(n_wires=self.n_wires)
        self.encoder = tq.GeneralEncoder(
        [   {'input_idx': [0], 'func': 'ry', 'wires': [0]},
            {'input_idx': [1], 'func': 'ry', 'wires': [1]},
            {'input_idx': [2], 'func': 'ry', 'wires': [2]},
            {'input_idx': [3], 'func': 'ry', 'wires': [3]},])

        self.arch = {'n_wires': self.n_wires, 'n_blocks': 5, 'n_layers_per_block': 2}
        self.q_layer = U3CU3Layer0(self.arch)
        self.measure = tq.MeasureAll(tq.PauliZ)

    def forward(self, x, use_qiskit=False):
        bsz = x.shape[0]
        x = F.avg_pool2d(x, 6).view(bsz, 4, 4)
        size = 4
        stride = 2
        x = x.view(bsz, size, size)

        data_list = []

        for c in range(0, size, stride):
            for r in range(0, size, stride):
                data = torch.transpose(torch.cat((x[:, c, r], x[:, c, r+1], x[:, c+1, r],
↪ x[:, c+1, r+1])).view(4, bsz), 0, 1)
                if use_qiskit:
                    data = self.qiskit_processor.process_parameterized(
                        self.q_device, self.encoder, self.q_layer, self.measure, data)
                else:
                    self.encoder(self.q_device, data)
                    self.q_layer(self.q_device)
                    data = self.measure(self.q_device)

                data_list.append(data.view(bsz, 4))

        # transpose to (bsz, channel, 2x2)
        result = torch.transpose(torch.cat(data_list, dim=1).view(bsz, 4, 4), 1, 2).
↪float()

        return result
```

(continues on next page)

(continued from previous page)

```
class QuantumClassifier(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.n_wires = 4
        self.q_device = tq.QuantumDevice(n_wires=4)
        self.encoder = tq.GeneralEncoder(encoder_op_list_name_dict['4x4_ryzxy'])
        self.arch = {'n_wires': self.n_wires, 'n_blocks': 8, 'n_layers_per_block': 2}
        self.ansatz = U3CU3Layer0(self.arch)
        self.measure = tq.MeasureAll(tq.PauliZ)

    def forward(self, x, use_qiskit=False):
        bsz = x.shape[0]
        x = F.avg_pool2d(x, 6).view(bsz, 16)

        if use_qiskit:
            x = self.qiskit_processor.process_parameterized(
                self.q_device, self.encoder, self.q_layer, self.measure, x)
        else:
            self.encoder(self.q_device, x)
            self.ansatz(self.q_device)
            x = self.measure(self.q_device)

        return x

class QFC(tq.QuantumModule):
    def __init__(self):
        super().__init__()
        self.n_wires = 4
        self.q_device = tq.QuantumDevice(n_wires=self.n_wires)
        self.encoder = tq.GeneralEncoder(encoder_op_list_name_dict['4x4_ryzxy'])
        self.arch = {'n_wires': self.n_wires, 'n_blocks': 4, 'n_layers_per_block': 2}

        self.q_layer = U3CU3Layer0(self.arch)
        self.measure = tq.MeasureAll(tq.PauliZ)

    def forward(self, x, use_qiskit=False):
        bsz = x.shape[0]
        data = x
        if use_qiskit:
            data = self.qiskit_processor.process_parameterized(
                self.q_device, self.encoder, self.q_layer, self.measure, data)
        else:
            self.encoder(self.q_device, data)
            self.q_layer(self.q_device)
            data = self.measure(self.q_device)
        return data


class Model1(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.qf = TrainableQuanvFilter()
```

(continues on next page)

(continued from previous page)

```
        self.linear = torch.nn.Linear(16, 4)

    def forward(self, x, use_qiskit=False):
        x = x.view(-1, 28, 28)
        x = self.qf(x)
        x = x.reshape(-1, 16)
        x = self.linear(x)
        return F.log_softmax(x, -1)

class Model2(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.qf = TrainableQuanvFilter()
        self.qfc = QFC()

    def forward(self, x, use_qiskit=False):
        x = x.view(-1, 28, 28)
        x = self.qf(x)
        x = x.reshape(-1, 16)
        x = self.qfc(x)
        return F.log_softmax(x, -1)

class Model3(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.qfc = QuantumClassifier()

    def forward(self, x, use_qiskit=False):
        x = self.qfc(x)
        return F.log_softmax(x, -1)

class Model4(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = torch.nn.Linear(16, 9)
        self.linear2 = torch.nn.Linear(9, 4)

    def forward(self, x, use_qiskit=False):
        x = x.view(-1, 28, 28)
        bsz = x.shape[0]
        x = F.avg_pool2d(x, 6).view(bsz, 16)

        x = self.linear1(x)
        x = self.linear2(x)
        return F.log_softmax(x, -1)
```

Here we do the MNIST 4 classification tasks.

[ ]:
```
dataset = MNIST(
    root='./mnist_data',
    train_valid_split_ratio=[0.9, 0.1],
    digits_of_interest=[0, 1, 2, 3],
```

(continues on next page)

(continued from previous page)

```
    n_test_samples=300,
    n_train_samples=500,
)

dataflow = dict()
for split in dataset:
    sampler = torch.utils.data.RandomSampler(dataset[split])
    dataflow[split] = torch.utils.data.DataLoader(
        dataset[split],
        batch_size=10,
        sampler=sampler,
        num_workers=8,
        pin_memory=True)
```

[ ]:
```
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
accus = []

model_list = [Model1().to(device), Model2().to(device), Model3().to(device), Model4().
↪to(device)]
for model in model_list:
  n_epochs = 15

  optimizer = optim.Adam(model.parameters(), lr=5e-3, weight_decay=1e-4)
  scheduler = CosineAnnealingLR(optimizer, T_max=n_epochs)
  for epoch in range(1, n_epochs + 1):
      # train
      print(f"Epoch {epoch}:")
      train(dataflow, model, device, optimizer)
      print(optimizer.param_groups[0]['lr'])
      # valid
      accu, loss = valid_test(dataflow, 'test', model, device)
      scheduler.step()
  accus.append(accu)
```

[ ]:
```
for i, accu in enumerate(accus):
  print('accuracy of model{0}: {1}'.format(i+1, accu))
```

CHAPTER

FIVE

WELCOME TO TORCHQUANTUM'S DOCUMENTATION!

A PyTorch-based hybrid classical-quantum dynamic neural networks framework.

```
@inproceedings{hanruiwang2022quantumnas,
    title     = {Quantumnas: Noise-adaptive search for robust quantum circuits},
    author    = {Wang, Hanrui and Ding, Yongshan and Gu, Jiaqi and Lin, Yujun and Pan,␣
↪David Z and Chong, Frederic T and Han, Song},
    booktitle = {The 28th IEEE International Symposium on High-Performance Computer␣
↪Architecture (HPCA-28)},
    year      = {2022}
}
```

5.1 News

- Colab examples are available in the artifact folder.
- Our recent paper "QuantumNAS: Noise-Adaptive Search for Robust Quantum Circuits" is accepted to HPCA 2022. We are working on updating the repo to add more examples soon!
- Add a simple example script using quantum gates to do MNIST classification.
- v0.0.1 available. Feedbacks are highly welcomed!

5.2 Installation

```
git clone https://github.com/mit-han-lab/torchquantum.git
cd torchquantum
pip install --editable .
```

5.3 Usage

Construct quantum NN models as simple as constructing a normal pytorch model.

```
import torch.nn as nn
import torch.nn.functional as F
import torchquantum as tq
import torchquantum.functional as tqf

class QFCModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.n_wires = 4
    self.q_device = tq.QuantumDevice(n_wires=self.n_wires)
    self.measure = tq.MeasureAll(tq.PauliZ)

    self.encoder_gates = [tqf.rx] * 4 + [tqf.ry] * 4 + \
                         [tqf.rz] * 4 + [tqf.rx] * 4
    self.rx0 = tq.RX(has_params=True, trainable=True)
    self.ry0 = tq.RY(has_params=True, trainable=True)
    self.rz0 = tq.RZ(has_params=True, trainable=True)
    self.crx0 = tq.CRX(has_params=True, trainable=True)

  def forward(self, x):
    bsz = x.shape[0]
    # down-sample the image
    x = F.avg_pool2d(x, 6).view(bsz, 16)

    # reset qubit states
    self.q_device.reset_states(bsz)

    # encode the classical image to quantum domain
    for k, gate in enumerate(self.encoder_gates):
      gate(self.q_device, wires=k % self.n_wires, params=x[:, k])

    # add some trainable gates (need to instantiate ahead of time)
    self.rx0(self.q_device, wires=0)
    self.ry0(self.q_device, wires=1)
    self.rz0(self.q_device, wires=3)
    self.crx0(self.q_device, wires=[0, 2])

    # add some more non-parameterized gates (add on-the-fly)
    tqf.hadamard(self.q_device, wires=3)
    tqf.sx(self.q_device, wires=2)
    tqf.cnot(self.q_device, wires=[3, 0])
    tqf.qubitunitary(self.q_device0, wires=[1, 2], params=[[1, 0, 0, 0],
                                                           [0, 1, 0, 0],
                                                           [0, 0, 0, 1j],
                                                           [0, 0, -1j, 0]])

    # perform measurement to get expectations (back to classical domain)
    x = self.measure(self.q_device).reshape(bsz, 2, 2)
```

(continues on next page)

(continued from previous page)

```
    # classification
    x = x.sum(-1).squeeze()
    x = F.log_softmax(x, dim=1)

    return x
```

5.4 Features

- Easy construction of parameterized quantum circuits in PyTorch.
- Support **batch mode** inference and training on CPU/GPU.
- Support **dynamic computation graph** for easy debugging.
- Support easy **deployment on real quantum devices** such as IBMQ.

5.5 TODOs

- Support more gates
- Support compile a unitary with descriptions to speedup training
- Support other measurements other than analytic method
- In einsum support multiple qubit sharing one letter. So that more than 26 qubit can be simulated.
- Support bmm based implementation to solve scalability issue
- Support conversion from torchquantum to qiskit

5.6 Dependencies

- Python >= 3.7
- PyTorch >= 1.8.0
- configargparse >= 0.14
- GPU model training requires NVIDIA GPUs

5.7 MNIST Example

Train a quantum circuit to perform MNIST task and deploy on the real IBM Yorktown quantum computer as in mnist_example.py script:

```
python mnist_example.py
```

5.8 Files

File	Description
devices.py	QuantumDevice class which stores the statevector
encoding.py	Encoding layers to encode classical values to quantum domain
functional.py	Quantum gate functions
operators.py	Quantum gate classes
layers.py	Layer templates such as RandomLayer
measure.py	Measurement of quantum states to get classical values
graph.py	Quantum gate graph used in static mode
super_layer.py	Layer templates for SuperCircuits
plugins/qiskit*	Convertors and processors for easy deployment on IBMQ
examples/	More examples for training QML and VQE models

5.9 More Examples

The `examples/` folder contains more examples to train the QML and VQE models. Example usage for a QML circuit:

```
# train the circuit with 36 params in the U3+CU3 space
python examples/train.py examples/configs/mnist/four0123/train/baseline/u3cu3_s0/rand/
↪param36.yml

# evaluate the circuit with torchquantum
python examples/eval.py examples/configs/mnist/four0123/eval/tq/all.yml --run-dir=runs/
↪mnist.four0123.train.baseline.u3cu3_s0.rand.param36

# evaluate the circuit with real IBMQ-Yorktown quantum computer
python examples/eval.py examples/configs/mnist/four0123/eval/x2/real/opt2/300.yml --run-
↪dir=runs/mnist.four0123.train.baseline.u3cu3_s0.rand.param36
```

Example usage for a VQE circuit:

```
# Train the VQE circuit for h2
python examples/train.py examples/configs/vqe/h2/train/baseline/u3cu3_s0/human/param12.
↪yml

# evaluate the VQE circuit with torchquantum
python examples/eval.py examples/configs/vqe/h2/eval/tq/all.yml --run-dir=runs/vqe.h2.
↪train.baseline.u3cu3_s0.human.param12/

# evaluate the VQE circuit with real IBMQ-Yorktown quantum computer
python examples/eval.py examples/configs/vqe/h2/eval/x2/real/opt2/all.yml --run-dir=runs/
↪vqe.h2.train.baseline.u3cu3_s0.human.param12/
```

Detailed documentations coming soon.

5.10 QuantumNAS

Quantum noise is the key challenge in Noisy Intermediate-Scale Quantum (NISQ) computers. Previous work for mitigating noise has primarily focused on gate-level or pulse-level noise-adaptive compilation. However, limited research efforts have explored a higher level of optimization by making the quantum circuits themselves resilient to noise. We propose QuantumNAS, a comprehensive framework for noise-adaptive co-search of the variational circuit and qubit mapping. Variational quantum circuits are a promising approach for constructing QML and quantum simulation. However, finding the best variational circuit and its optimal parameters is challenging due to the large design space and parameter training cost. We propose to decouple the circuit search and parameter training by introducing a novel SuperCircuit. The SuperCircuit is constructed with multiple layers of pre-defined parameterized gates and trained by iteratively sampling and updating the parameter subsets (SubCircuits) of it. It provides an accurate estimation of SubCircuits performance trained from scratch. Then we perform an evolutionary co-search of SubCircuit and its qubit mapping. The SubCircuit performance is estimated with parameters inherited from SuperCircuit and simulated with real device noise models. Finally, we perform iterative gate pruning and finetuning to remove redundant gates. Extensively evaluated with 12 QML and VQE benchmarks on 10 quantum comput, QuantumNAS significantly outperforms baselines. For QML, QuantumNAS is the first to demonstrate over 95% 2-class, 85% 4-class, and 32% 10-class classification accuracy on real QC. It also achieves the lowest eigenvalue for VQE tasks on H2, H2O, LiH, CH4, BeH2 compared with UCCSD. We also open-source torchquantum for fast training of parameterized quantum circuits to facilitate future research.

QuantumNAS Framework overview:

QuantumNAS models achieve higher robustness and accuracy than other baseline models:

5.11 Contact

Hanrui Wang (hanrui@mit.edu)

PYTHON MODULE INDEX

t

INDEX

Symbols

A

B

C

D

E

F

G

H

I

M

N

O

P

Q

R

S

T

U

W

X

Y

Z