QCompute-QAPP Documentation

Institute for Quantum Computing, Baidu Inc.

Jun 30, 2021

API DOCS

CHAPTER

ONE

QAPP PACKAGE

1.1 qapp.algorithm package

class `qapp.algorithm.`**`VQE`**(*num*, *hamiltonian*, *ansatz*, *optimizer*, *backend*, *measurement='default'*)

Bases: `object`

Variational Quantum Eigensolver class

The constructor of the VQE class

Parameters

- **num** (`int`) – Number of qubits
- **hamiltonian** (`List`) – Hamiltonian whose minimum eigenvalue is to be solved
- **ansatz** (*ParameterizedCircuit*) – Ansatz used to search for the ground state of the Hamiltonian
- **optimizer** (*BasicOptimizer*) – Optimizer used to optimize the parameters in the ansatz
- **backend** (`str`) – Backend to be used in this task. Please refer to https://quantum-hub.baidu.com/quickGuide for details
- **measurement** (`str`) – Method chosen from 'default', 'ancilla', and 'SimMeasure' for measuring the expectation value, defaults to 'default'

`get_measure`(*shots=1024*)

Returns the measurement results

Parameters **shots** (`int`) – Number of measurement shots, defaults to 1024

Return type `dict`

Returns Measurement results in bitstrings with the number of counts

`get_gradient`(*shots=1024*)

Calculates the gradient with respect to current parameters in circuit

Parameters **shots** (`int`) – Number of measurement shots, defaults to 1024

Return type `ndarray`

Returns Gradient with respect to current parameters

`get_loss`(*shots=1024*)

Calculates the loss with respect to current parameters in circuit

Parameters **shots** (`int`) – Number of measurement shots, defaults to 1024

Return type `float`

Returns Loss with respect to current parameters

`run(shots=1024)`
Searches for the minimum eigenvalue of the input Hamiltonian with the given ansatz and optimizer

Parameters **shots** (int) – Number of measurement shots, defaults to 1024

property minimum_eigenvalue: Union[str, float]
The optimized minimum eigenvalue from last run

Return type Union[str, float]

Returns Optimized minimum eigenvalue from last run

`set_backend(backend)`
Sets the backend to be used

Parameters **backend** (str) – Backend to be used

class qapp.algorithm.SSVQE(*num, ex_num, hamiltonian, ansatz, optimizer, backend, measurement='default'*)
Bases: object

Subspace-Search Variational Quantum Eigensolver class

Please see https://journals.aps.org/prresearch/abstract/10.1103/PhysRevResearch.1.033062 for details on this algorithm.

The constructor of the SSVQE class

Parameters

- **num** (int) – Number of qubits
- **ex_num** (int) – Number of extra eignevalues to be solved. When ex_num = 0, only compute the minimum eigenvalue
- **hamiltonian** (List) – Hamiltonian whose eigenvalues are to be solved
- **ansatz** (*ParameterizedCircuit*) – Ansatz used to search for the eigenstates of the Hamiltonian
- **optimizer** (*BasicOptimizer*) – Optimizer used to optimize the parameters in the ansatz
- **backend** (str) – Backend to be used in this task. Please refer to https://quantum-hub.baidu.com/quickGuide for details
- **measurement** (str) – Method chosen from 'default', 'ancilla', and 'SimMeasure' for measuring the expectation value, defaults to 'default'

`get_gradient(shots=1024)`
Calculates the gradient with respect to current parameters in circuit

Parameters **shots** (int) – Number of measurement shots, defaults to 1024

Return type ndarray

Returns Gradient with respect to current parameters

`get_loss(shots=1024)`
Calculates the loss with respect to current parameters in circuit

Parameters **shots** (int) – Number of measurement shots, defaults to 1024

Return type float

Returns Loss with respect to current parameters

`run`(*shots=1024*)
Searches for the minimum eigenvalue of the input Hamiltonian with the given ansatz and optimizer

Parameters **shots** (`int`) – Number of measurement shots, defaults to 1024

`property minimum_eigenvalues: Union[str, List]`
The optimized minimum eigenvalue from last run

Return type `Union[str, List]`

Returns Optimized minimum eigenvalues from last run

`set_backend`(*backend*)
Sets the backend to be used

Parameters **backend** (`str`) – Backend to be used

`class qapp.algorithm.QAOA`(*num*, *hamiltonian*, *ansatz*, *optimizer*, *backend*, *measurement='default'*, *delta=0.1*)
Bases: `object`

Quantum Approximate Optimization Algorithm class

The constructor of the QAOA class

Parameters

- **num** (`int`) – Number of qubits
- **hamiltonian** (`List`) – Hamiltonian used to construct the QAOA ansatz
- **ansatz** (*QAOAAnsatz*) – QAOA ansatz used to search for the maximum eigenstate of the Hamiltonian
- **optimizer** (*BasicOptimizer*) – Optimizer used to optimize the parameters in the ansatz
- **backend** (`str`) – Backend to be used in this task. Please refer to https://quantum-hub.baidu.com/quickGuide for details
- **measurement** (`str`) – Method chosen from 'default', 'ancilla', and 'SimMeasure' for measuring the expectation value, defaults to 'default'
- **delta** (`float`) – Parameter used to calculate gradients, defaults to 0.1

`get_measure`(*shots=1024*)
Returns the measurement results

Parameters **shots** (`int`) – Number of measurement shots, defaults to 1024

Return type `dict`

Returns Measurement results in bitstrings with the number of counts

`get_gradient`(*shots=1024*)
Calculates the gradient with respect to current parameters in circuit

Parameters **shots** (`int`) – Number of measurement shots, defaults to 1024

Return type `ndarray`

Returns Gradient with respect to current parameters

`get_loss`(*shots=1024*)
Calculates the loss with respect to current parameters in circuit

Parameters **shots** (`int`) – Number of measurement shots, defaults to 1024

Return type `float`

Returns Loss with respect to current parameters

`run`(*shots=1024*)
Searches for the maximum eigenvalue of the input Hamiltonian with the given ansatz and optimizer

Parameters **shots** (int) – Number of measurement shots, defaults to 1024

property maximum_eigenvalue: Union[str, float]
The optimized maximum eigenvalue from last run

Return type Union[str, float]

Returns Optimized maximum eigenvalue from last run

`set_backend`(*backend*)
Sets the backend to be used

Parameters **backend** (str) – Backend to be used

class qapp.algorithm.**KernelClassifier**(*backend*, *encoding_style='IQP'*, *kernel_type='qke'*, *shots=1024*)
Bases: object

Kernel Classifier class

The constructor of the KernelClassifier class

Parameters

- **encoding_style** (str) – Encoding scheme to be used, defaults to 'IQP', which uses the default encoding scheme
- **kernel_type** (str) – Type of kernel to be used, defaults to 'qke', i.e., <x1|x2>
- **backend** (str) – Backend to be used in this task. Please refer to https://quantum-hub.baidu.com/quickGuide for details
- **shots** (int) – Number of measurement shots, defaults to 1024

`fit`(*X*, *y*)
Trains the classifier with known data

Parameters

- **X** (ndarray) – Set of classical data vectors as the training data
- **y** (ndarray) – Known labels of the training data

`predict`(*x*)
Predicts labels of new data

Parameters **x** (ndarray) – Set of data vectors with unknown labels

Return type ndarray

Returns Predicted labels of the input data

1.2 qapp.application package

1.2.1 qapp.application.chemistry package

class `qapp.application.chemistry.`**`MolecularGroundStateEnergy`**(*num_qubits=0*, *hamiltonian=None*)

Bases: `object`

Molecular Ground State Energy class

The constructor of the MolecularGroundStateEnergy class

Parameters

- **`num_qubits`** (`int`) – Number of qubits, defaults to 0
- **`hamiltonian`** (`Optional[List]`) – Hamiltonian of the molecular system, defaults to None

property num_qubits: int

The number of qubits used to encoding this molecular system

Return type `int`

Returns Number of qubits

property hamiltonian: List

The Hamiltonian of this molecular system

Return type `List`

Returns Hamiltonian of this molecular system

`compute_ground_state_energy`()

Analytically computes the ground state energy

Return type `float`

`load_hamiltonian_from_file`(*filename*, *separator=', '*)

Loads Hamiltonian from a file

Parameters

- **`filename`** (`str`) – Path to the file storing the Hamiltonian in Pauli terms
- **`separator`** (`str`) – Delimiter between coefficient and Pauli string, defaults to ', '

1.2.2 qapp.application.optimization package

class `qapp.application.optimization.`**`MaxCut`**(*num_qubits=0*, *hamiltonian=None*)

Bases: `object`

Max Cut Problem class

The constructor of the MaxCut class

Parameters

- **`num_qubits`** (`int`) – Number of qubits, defaults to 0
- **`hamiltonian`** (`Optional[List]`) – Hamiltonian of the target graph of the Max Cut problem, defaults to None

property num_qubits: int

The number of qubits used to encoding this target graph

Return type `int`

Returns Number of qubits used to encoding this target graph

property hamiltonian: List

The Hamiltonian of this target graph

Return type `List`

Returns Hamiltonian of this target graph

graph_to_hamiltonian(*graph*)

Constructs Hamiltonian from the target graph of the Max Cut problem

Parameters **graph** (`Graph`) – Undirected graph without weights

decode_bitstring(*bitstring*)

Decodes the measurement result into problem solution, i.e., set partition

Parameters **bitstring** (`str`) – Measurement result with the largest probability

Return type `dict`

Returns Solution to the Max Cut problem

1.3 qapp.circuit package

class qapp.circuit.**BasicCircuit**(*num*)

Bases: `abc.ABC`

Basic Circuit class

The constructor of the BasicCircuit class

Parameters **num** (`int`) – Number of qubits

abstract add_circuit(*q*)

Adds circuit to the register.

Parameters **q** (`QRegPool`) – Quantum register to which this circuit is added

class qapp.circuit.**IQPEncodingCircuit**(*num*, *inverse=False*)

Bases: *qapp.circuit.basic_circuit.BasicCircuit*

IQP Encoding Circuit class

The constructor of the IQPEncodingCircuit class

Parameters

- **num** (`int`) – Number of qubits
- **inverse** (`bool`) – Whether the encoding circuit will be inverted, i.e. U^dagger(x) if True, defaults to False

add_circuit(*q*, *x*)

Adds the encoding circuit used to map a classical data vector into its quantum feature state

Parameters

- **q** (`QRegPool`) – Quantum register to which this circuit is added
- **x** (`ndarray`) – Classical data vector to be encoded

class qapp.circuit.**BasisEncodingCircuit**(*num*, *bit_string*)

Bases: *qapp.circuit.basic_circuit.BasicCircuit*

Basis Encoding Circuit class

The constructor of the BasisEncodingCircuit class

Parameters

- **num** (int) – Number of qubits
- **bit_string** (str) – Bit string to be encoded as a quantum state

add_circuit(*q*)

Adds the basis encoding circuit to the register

Parameters **q** (QRegPool) – Quantum register to which this circuit is added

class qapp.circuit.**KernelEstimationCircuit**(*num*, *encoding_style*)

Bases: *qapp.circuit.basic_circuit.BasicCircuit*

Kernel Estimation Circuit class

The constructor of the KernelEstimationCircuit class

Parameters

- **num** (int) – Number of qubits
- **encoding_style** (str) – Encoding circuit, only accepts 'IQP' for now

add_circuit(*q*, *x1*, *x2*)

Adds the kernel estimation circuit used to evaluate the kernel entry value between two classical data vectors

Parameters

- **q** (QRegPool) – Quantum register to which this circuit is added
- **x1** (ndarray) – First classical vector
- **x2** (ndarray) – Second classical vector

class qapp.circuit.**ParameterizedCircuit**(*num*, *parameters*)

Bases: *qapp.circuit.basic_circuit.BasicCircuit*

Parameterized Circuit class

The constructor of the BasicCircuit class

Parameters

- **num** (int) – Number of qubits
- **parameters** (ndarray) – Parameters of parameterized gates

property parameters: numpy.ndarray

Parameters of the circuit

Return type ndarray

Returns Parameters of the circuit

set_parameters(*parameters*)

Sets parameters of the circuit

Parameters **parameters** (ndarray) – New parameters of the circuit

abstract add_circuit(*q*)

Adds the circuit to the register

Parameters **q** (QRegPool) – Quantum register to which this circuit is added

class qapp.circuit.**PauliMeasurementCircuit**(*num*, *pauli_terms*)

Bases: *qapp.circuit.basic_circuit.BasicCircuit*

Pauli Measurement Circuit class

The constructor of the PauliMeasurementCircuit class

Parameters

- **num** (int) – Number of qubits
- **pauli_terms** (str) – Pauli terms to be measured

add_circuit(*q*, *pauli_str*)

Adds the pauli measurement circuit to the register

Parameters

- **q** (QRegPool) – Quantum register to which this circuit is added
- **pauli_str** (str) – Pauli string to be measured

get_expectation(*preceding_circuits*, *shots*, *backend*)

Computes the expectation value of the Pauli terms

Parameters

- **preceding_circuit** – Circuit precedes the measurement circuit
- **shots** (int) – Number of measurement shots
- **backend** (str) – Backend to be used in this task

Return type float

Returns Expectation value of the Pauli terms

class qapp.circuit.**PauliMeasurementCircuitWithAncilla**(*num*, *pauli_terms*)

Bases: *qapp.circuit.basic_circuit.BasicCircuit*

Pauli Measurement Circuit with Ancilla class

The constructor of the PauliMeasurementCircuitWithAncilla class

Parameters

- **num** (int) – Number of qubits
- **pauli_terms** (str) – Pauli terms to be measured

add_circuit(*q*, *pauli_str*)

Adds the pauli measurement circuit to the register

Parameters

- **q** (QRegPool) – Quantum register to which this circuit is added
- **pauli_str** (str) – Pauli string to be measured

get_expectation(*preceding_circuits*, *shots*, *backend*)

Computes the expectation value of the Pauli terms

Parameters

- **preceding_circuit** – Circuit precedes the measurement circuit
- **shots** (int) – Number of measurement shots

- **backend** (str) – Backend to be used in this task

Return type float

Returns Expectation value of the Pauli terms

class qapp.circuit.**SimultaneousPauliMeasurementCircuit**(*num*, *pauli_terms*)
Bases: *qapp.circuit.basic_circuit.BasicCircuit*

Simultaneous Pauli Measurement Circuit for Qubitwise Commute Pauli Terms

The constructor of the SimultaneousPauliMeasurementCircuit class

Parameters

- **num** (int) – Number of qubits
- **pauli_terms** (List) – Pauli terms to be measured

add_circuit(*q*, *clique*)
Adds the simultaneous pauli measurement circuit to the register

Parameters

- **q** (QRegPool) – Quantum register to which this circuit is added
- **clique** (List) – Clique of Pauli terms to be measured together

get_expectation(*preceding_circuits*, *shots*, *backend*)
Computes the expectation value of the Pauli terms

Parameters

- **preceding_circuit** – Circuit precedes the measurement circuit
- **shots** (int) – Number of measurement shots
- **backend** (str) – Backend to be used in this task

Return type float

Returns Expectation value of the Pauli terms

class qapp.circuit.**QAOAAnsatz**(*num*, *parameters*, *hamiltonian*, *layer*)
Bases: *qapp.circuit.parameterized_circuit.ParameterizedCircuit*

QAOA Ansatz class

The constructor of the QAOAAnsatz class

Parameters

- **num** (int) – Number of qubits in this ansatz
- **parameters** (ndarray) – Parameters of parameterized gates in this ansatz
- **hamiltonian** (List) – Hamiltonian used to construct the QAOA ansatz
- **layer** (int) – Number of layers for this Ansatz

add_circuit(*q*)
Adds circuit to the register according to the given hamiltonian

Parameters **q** (QRegPool) – Quantum register to which this circuit is added

class qapp.circuit.**UniversalCircuit**(*num*, *parameters*)
Bases: *qapp.circuit.parameterized_circuit.ParameterizedCircuit*

Universal Circuit class

The constructor of the UniversalCircuit class

Parameters

- **num** (int) – Number of qubits in this ansatz
- **parameters** (ndarray) – Parameters of parameterized gates in this circuit, whose shape should be (3,) for single-qubit cases and should be (15,) for 2-qubit cases

add_circuit(*q*)

Adds the universal circuit to the register. Only support single-qubit and 2-qubit cases

Parameters **q** (QRegPool) – Quantum register to which this circuit is added

class qapp.circuit.**RealEntangledCircuit**(*num*, *layer*, *parameters*)

Bases: *qapp.circuit.parameterized_circuit.ParameterizedCircuit*

Real Entangled Circuit class

The constructor of the RealEntangledCircuit class

Parameters

- **num** (int) – Number of qubits in this ansatz
- **layer** (int) – Number of layers for this ansatz
- **parameters** (ndarray) – Parameters of parameterized gates in this circuit, whose shape should be (num * layer,)

add_circuit(*q*)

Adds the real entangled circuit to the register

Parameters **q** (QRegPool) – Quantum register to which this circuit is added

class qapp.circuit.**ComplexEntangledCircuit**(*num*, *layer*, *parameters*)

Bases: *qapp.circuit.parameterized_circuit.ParameterizedCircuit*

Complex Entangled Circuit class

The constructor of the ComplexEntangledCircuit class

Parameters

- **num** (int) – Number of qubits in this Ansatz
- **layer** (int) – Number of layer for this Ansatz
- **parameters** (ndarray) – Parameters of parameterized gates in this circuit, whose shape should be (num * layer * 2,)

add_circuit(*q*)

Adds the complex entangled circuit to the register

Parameters **q** (QRegPool) – Quantum register to which this circuit is added

class qapp.circuit.**RealAlternatingLayeredCircuit**(*num*, *layer*, *parameters*)

Bases: *qapp.circuit.parameterized_circuit.ParameterizedCircuit*

Real Alternating Layered Circuit class

The constructor of the RealAlternatingLayeredCircuit class

Parameters

- **num** (int) – Number of qubits in this Ansatz
- **layer** (int) – Number of layer for this Ansatz

- **parameters** (ndarray) – Parameters of parameterized gates in this circuit, whose shape should be `((2 * num - 2) * layer,)`

add_circuit(*q*)
Adds the real alternating layered circuit to the register

Parameters q (QRegPool) – Quantum register to which this circuit is added

class qapp.circuit.**ComplexAlternatingLayeredCircuit**(*num*, *layer*, *parameters*)
Bases: *qapp.circuit.parameterized_circuit.ParameterizedCircuit*

Complex Alternating Layered Circuit class

The constructor of the ComplexAlternatingLayeredCircuit class

Parameters

- **num** (int) – Number of qubits in this Ansatz
- **layer** (int) – Number of layer for this Ansatz
- **parameters** (ndarray) – Parameters of parameterized gates in this circuit, whose shape should be `((4 * num - 4) * layer,)`

add_circuit(*q*)
Adds the complex alternating layered circuit to the register

Parameters q (QRegPool) – Quantum register to which this circuit is added

1.4 qapp.optimizer package

class qapp.optimizer.**BasicOptimizer**(*iterations*, *circuit*)
Bases: abc.ABC

Basic Optimizer class

The constructor of the BasicOptimizer class

Parameters

- **iterations** (int) – Number of iterations
- **circuit** (*ParameterizedCircuit*) – Circuit whose parameters are to be optimized

set_circuit(*circuit*)
Sets the parameterized circuit to be optimized

Parameters circuit (*ParameterizedCircuit*) – Parameterized Circuit to be optimized

abstract **minimize**(*shots*, *loss_func*, *grad_func*)
Minimizes the given loss function

Parameters

- **shots** (int) – Number of measurement shots
- **loss_func** (Callable[[ndarray, int], float]) – Loss function to be minimized
- **grad_func** (Callable[[ndarray, int], ndarray]) – Function for calculating gradients

class qapp.optimizer.**SGD**(*iterations*, *circuit*, *learning_rate*)
Bases: *qapp.optimizer.basic_optimizer.BasicOptimizer*

SGD Optimizer class

The constructor of the SGD class

Parameters

- **iterations** (int) – Number of iterations
- **circuit** (*BasicCircuit*) – Circuit whose parameters are to be optimized

minimize(*shots*, *loss_func*, *grad_func*)
Minimizes the given loss function

Parameters

- **iterations** – Number of iterations
- **shots** (int) – Number of measurement shots
- **loss_func** (Callable[[ndarray, int], float]) – Loss function to be minimized
- **grad_func** (Callable[[ndarray, int], ndarray]) – Function for calculating gradients

class qapp.optimizer.**SLSQP**(*iterations*, *circuit*)
Bases: *qapp.optimizer.basic_optimizer.BasicOptimizer*

SLSQP Optimizer class

The constructor of the SLSQP class

Parameters

- **iterations** (int) – Number of iterations
- **circuit** (*BasicCircuit*) – Circuit whose parameters are to be optimized

minimize(*shots*, *loss_func*, *grad_func*)
Minimizes the given loss function

Parameters

- **shots** (int) – Number of measurement shots
- **loss_func** (Callable[[ndarray, int], float]) – Loss function to be minimized
- **grad_func** (Callable[[ndarray, int], ndarray]) – Function for calculating gradients

class qapp.optimizer.**SPSA**(*iterations*, *circuit*, *a=1.0*, *c=1.0*)
Bases: *qapp.optimizer.basic_optimizer.BasicOptimizer*

SPSA Optimizer class

The constructor of the SPSA class

Parameters

- **iterations** (int) – Number of iterations
- **circuit** (*BasicCircuit*) – Circuit whose parameters are to be optimized
- **a** (float) – Scaling parameter for step size, defaults to 1.0
- **c** (float) – Scaling parameter for evaluation step size, defaults to 1.0

minimize(*shots*, *loss_func*, *grad_func*)
Minimizes the given loss function

Parameters

- **shots** (int) – Number of measurement shots
- **loss_func** (Callable[[ndarray, int], float]) – Loss function to be minimized

- **grad_func** (`Callable[[ndarray, int], ndarray]`) – Function for calculating gradients

class `qapp.optimizer.`**SMO**(*iterations*, *circuit*)

Bases: *qapp.optimizer.basic_optimizer.BasicOptimizer*

SMO Optimizer class

Please see https://arxiv.org/abs/1903.12166 for details on this optimization method.

The constructor of the SMO class

Parameters

- **iterations** (`int`) – Number of iterations
- **circuit** (*BasicCircuit*) – Circuit whose parameters are to be optimized

minimize(*shots*, *loss_func*, *grad_func*)

Minimizes the given loss function

Parameters

- **shots** (`int`) – Number of measurement shots
- **loss_func** (`Callable[[ndarray, int], float]`) – Loss function to be minimized
- **grad_func** (`Callable[[ndarray, int], ndarray]`) – Function for calculating gradients

class `qapp.optimizer.`**Powell**(*iterations*, *circuit*)

Bases: *qapp.optimizer.basic_optimizer.BasicOptimizer*

Powell Optimizer class

The constructor of the Powell class

Parameters

- **iterations** (`int`) – Number of iterations
- **circuit** (*BasicCircuit*) – Circuit whose parameters are to be optimized

minimize(*shots*, *loss_func*, *grad_func*)

Minimizes the given loss function

Parameters

- **shots** (`int`) – Number of measurement shots
- **loss_func** (`Callable[[ndarray, int], float]`) – Loss function to be minimized
- **grad_func** (`Callable[[ndarray, int], ndarray]`) – Function for calculating gradients

1.5 qapp.utils package

`qapp.utils.`**grouping_hamiltonian**(*hamiltonian*, *coloring_strategy='largest_first'*)

Finds the minimum clique cover of the Hamiltonian graph, which is used for simultaneous Pauli measurement

Parameters

- **hamiltonian** (`List`) – Hamiltonian of the target system
- **coloring_strategy** (`str`) – Graph coloring strategy chosen from the following: 'largest_first', 'random_sequential', 'smallest_last', 'independent_set', 'connected_sequential_bfs', 'connected_sequential_dfs', 'connected_sequential', 'saturation_largest_first', and 'DSATUR'; defaults to 'largest_first'

Return type `List[List[str]]`

Returns List of cliques consisting of Pauli strings to be measured together

`qapp.utils.`**`pauli_terms_to_matrix`**(*pauli_terms*)

Converts Pauli terms to a matrix

Parameters **`pauli_terms`** (`List`) – Pauli terms whose matrix is to be computed

Return type `ndarray`

Returns Matrix form of the Pauli terms

PYTHON MODULE INDEX

q