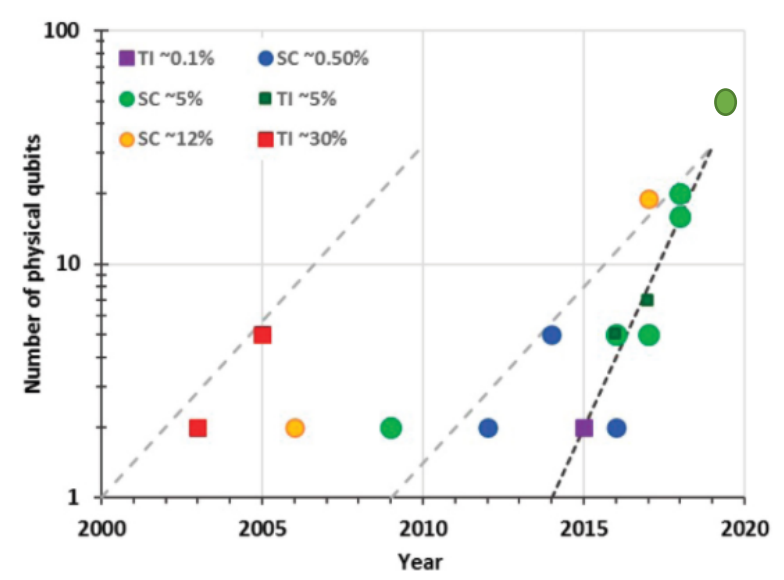
2020年度未踏ターゲット事業
成果報告会

テンソルネットワーク構造を用いた量子回路学習

真鍋秀隆 / Hidetaka Manabe

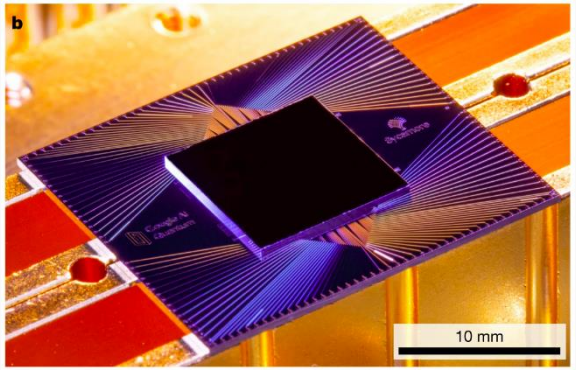
京都大学工学部情報学科
非線形物理学講座 計算物理学グループ B4

量子コンピュータの発展と量子機械学習



量子コンピュータの基本単位：量子ビット

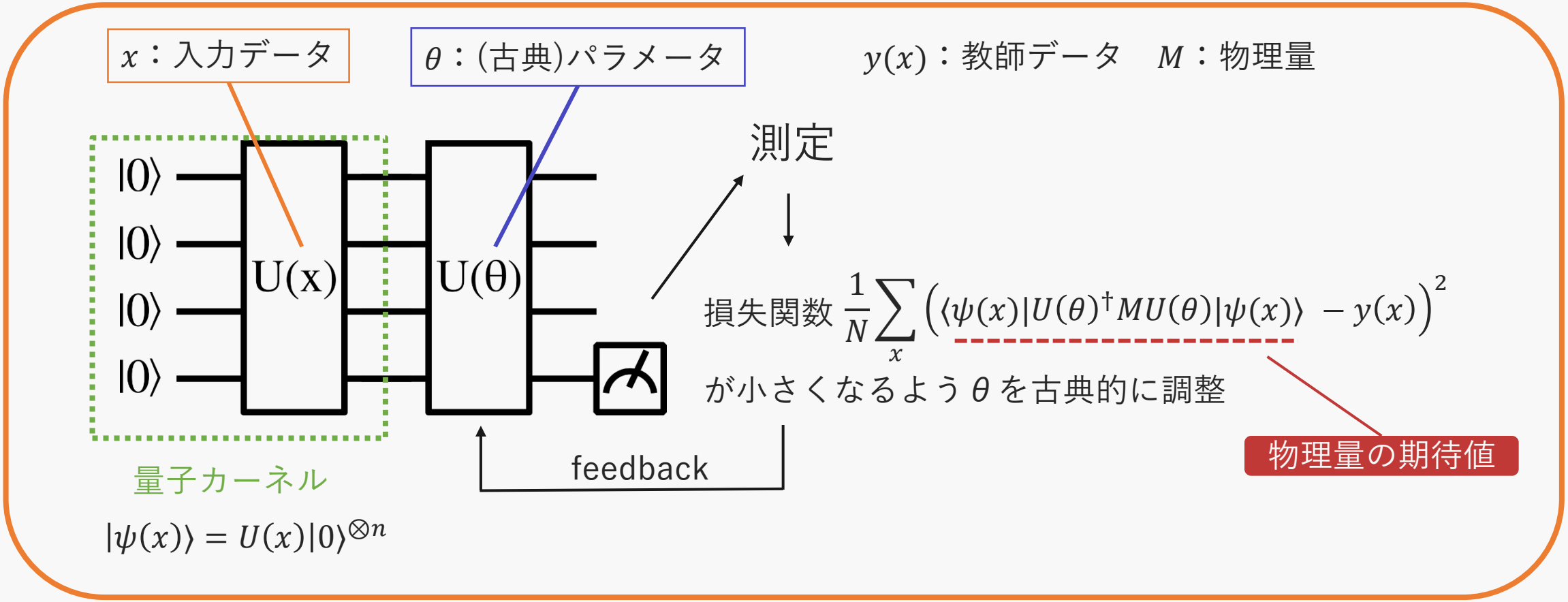
量子ビットが増えるごとに性能が劇的に向上する
→**量子超越性**の実現
　従来のコンピュータの限界を超えた計算能力を示す
　(例：Google(2019)，中国の研究チーム(2020))



今開発されているものは，**NISQ**と呼ばれる
誤り耐性機能のない量子コンピュータで，
ノイズが非常に強い
→アルゴリズムとしてノイズ耐性のある**量子機械学習**が
　注目されている．

量子回路学習アルゴリズム

量子回路学習：量子機械学習の１種.
古典コンピュータでは学習できないデータセットを学習できる可能性があるとして注目されている.



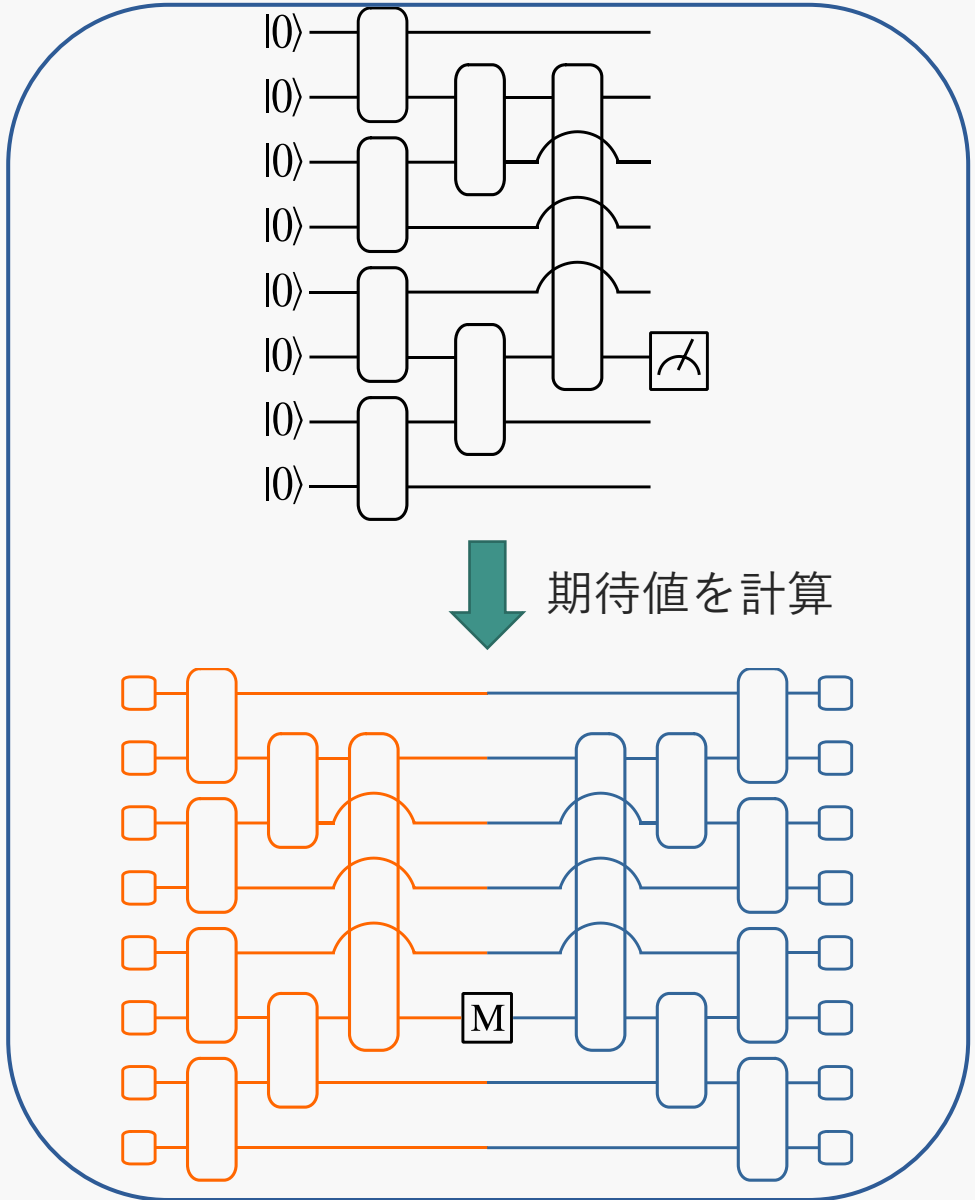
量子回路とテンソルネットワーク

量子回路を古典シミュレーション
→指数時間かかる.
(30量子ビット程度が限度)

量子回路をテンソルネットワークとみなすことで
計算時間を短縮.

右図のようなツリーテンソルネットワーク型の
量子回路を用いれば，物理量の期待値は
効率的に計算することができる.

一般の量子回路に対しても，最適な縮約順を見つけることで
従来手法より高速に計算できる.



本プロジェクトの提案

①**テンソルネットワークをバックエンド**とする量子回路学習ライブラリの作成

- **Pythonで作成**
- ライセンスをつけて**Github等で公開**
- **誰でも簡単に利用できるように**

→QTensorNetの開発

②テンソルネットワークを用いた量子回路学習についての**研究**

- 本プロジェクトで提供するライブラリを用いて，
  より効率的かつ再現可能な実験→研究を加速

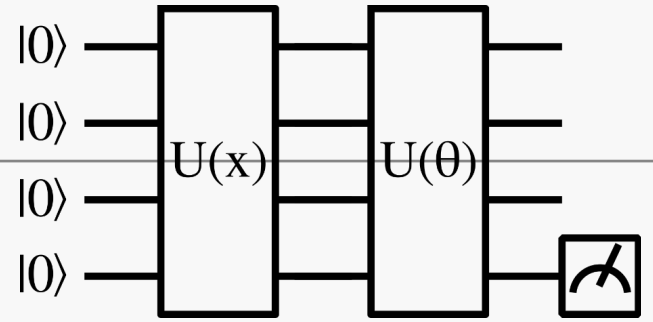
→大規模量子系における量子回路学習の性能調査

QTensorNet

- テンソルネットワーク・バックエンドの量子回路シミュレータ，量子回路学習フレームワーク
- 物理量の期待値をテンソルネットワークのアルゴリズムを用いて高速に計算
- 量子回路学習のシミュレートを低いコストで実現
- 通常の量子回路シミュレータとしても利用可能
- GPUにも対応

https://github.com/wotto27oct/QTensorNet

QTensorNetの使用例



大きさ8×8の"0"と"1"手書き文字認識をしてみる.

データは各ラベルにつき1000枚.

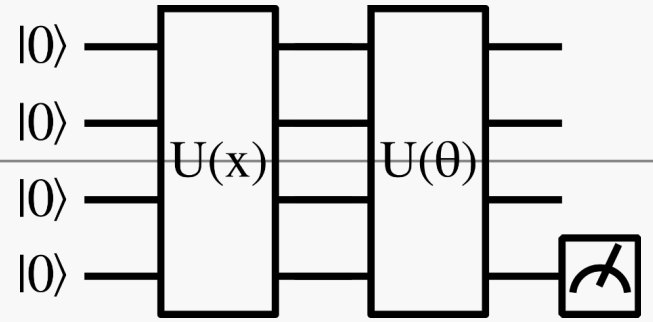
MNIST dataset.
0~9の数字の画像

```
import jax.numpy as np
import qtensornetwork.components
import qtensornetwork.circuit
import qtensornetwork.ansatz
import qtensornetwork.util
import qtensornetwork.optimizer
from qtensornetwork.gate import *
```

①各種モジュールのインポート

QTensorNetは，バックエンドとして
Googleの**Jax**を使用.
通常のnumpyと同様に扱える.

QTensorNetの使用例



大きさ8×8の"0"と"1"手書き文字認識をしてみる.

②量子回路を準備

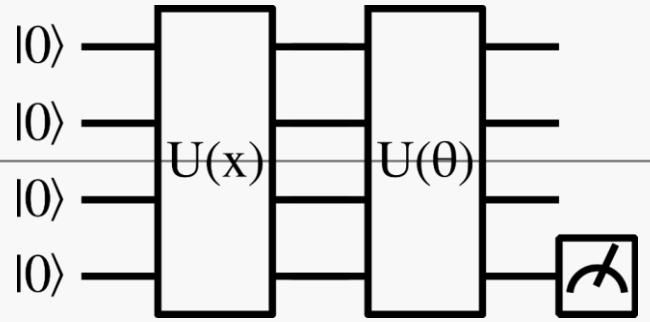
```
qnum = 64
circuit = qtensornetwork.circuit.Circuit(qnum)

xtrain, ytrain, xtest, ytest = generate_binary_mnist(0, 1, 1000, 200, 8, 8)

qxtrain = qtensornetwork.util.dtoq_miles(xtrain)
qxtest = qtensornetwork.util.dtoq_miles(xtest)

for i in range(qnum):
    circuit.set_init_state(qtensornetwork.components.State([i], None, train_idx=i))
```

古典データを量子状態にエンコードする
いくつかの標準的な関数も使用可能.

③データを準備

④量子データと量子ビットを対応付ける

QTensorNetの使用例



大きさ8×8の"0"と"1"手書き文字認識をしてみる.

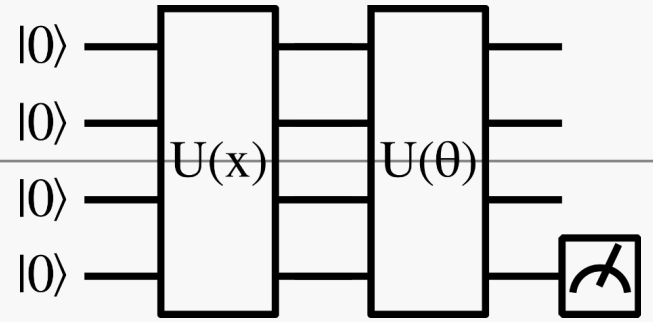
```
def complex_gate():
    Rz0 = RZ(0,0)
    Ry1 = RY(0,0)
    Rz2 = RZ(0,0)
    Rz3 = RZ(1,0)
    Ry4 = RY(1,0)
    Ry5 = RZ(1,0)
    CNOT6 = CNOT([0,1])
    U_gate = combine_gates([Rz0, Ry1, Rz2, Rz3, Ry4, Ry5, CNOT6])
    return U_gate

cgate = complex_gate()
layer = qtensornetwork.ansatz.TTN([i for i in range(qnum)], gate_input_num=2,
                                  gate_output_num=1, gate_func=cgate.func, gate_params_num=6)
circuit.append_layer(layer)
```

標準的に用いる量子ゲートも使用可能.

既存のゲートを組み合わせて
新しいゲートを簡単に構成可能.

⑤テンソルネットワーク構造を用いて
回路を自動的に構成する.

QTensorNetの使用例



大きさ8×8の"0"と"1"手書き文字認識をしてみる.

```
m_tensor = np.array([[1, 0], [0, 0]])
measurement1 = qtensornetwork.components.Measurement(None, m_tensor)
circuit.add_measurement(measurement1)

circuit.show_circuit_structure()

optimizer = qtensornetwork.optimizer.Adam(lr=0.01)

circuit.classify(qxtrain, None, ytrain, qxtest, None, ytest,
                 optimizer=optimizer, epoch=50, batch_size=20)
```

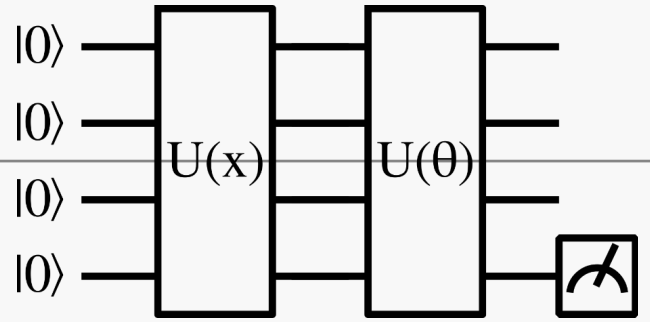
⑥測定を準備

回路のおおまかな構造を描画する関数.

⑦optimizerを準備

⑧データとオプションを指定して学習開始

QTensorNetの使用例



大きさ8×8の"0"と"1"手書き文字認識をしてみる.

```
compling loss and grad function......
time: 155.46103143692017
compiling accuracy function.......
time: 98.16039061546326
initial loss: 500.0056288987398
initial train_acc: 0.4985
initial test_acc: 0.5125
------optimization start------
epoch: 1 loss: 438.24718037690036 train_accuracy: 0.725 test_accuracy: 0.7075 elapsed time for epoch: 20.358287811279297
epoch: 2 loss: 318.6578997867182 train_accuracy: 0.782 test_accuracy: 0.77 elapsed time for epoch: 17.6671085357666
epoch: 3 loss: 289.9908798709512 train_accuracy: 0.805 test_accuracy: 0.7925 elapsed time for epoch: 17.705913543701172
epoch: 4 loss: 264.46283615425637 train_accuracy: 0.822 test_accuracy: 0.7925 elapsed time for epoch: 24.415674686431885
epoch: 5 loss: 243.79993093297526 train_accuracy: 0.8435 test_accuracy: 0.8175 elapsed time for epoch: 17.578953504562378
```

⋮

```
epoch: 48 loss: 222.65169605513802 train_accuracy: 0.847 test_accuracy: 0.83 elapsed time for epoch: 17.471848011016846
epoch: 49 loss: 221.93629252619576 train_accuracy: 0.859 test_accuracy: 0.835 elapsed time for epoch: 17.76170563697815
epoch: 50 loss: 222.5268678564462 train_accuracy: 0.8575 test_accuracy: 0.835 elapsed time for epoch: 17.947643995285034
------optimization end------
optimization time: 1356.517373085022 [sec]
whole elapsed time: 1610.1389956474304 [sec]
```

結果：テスト精度83.5%，学習時間1610秒

QTensorNetの使用例

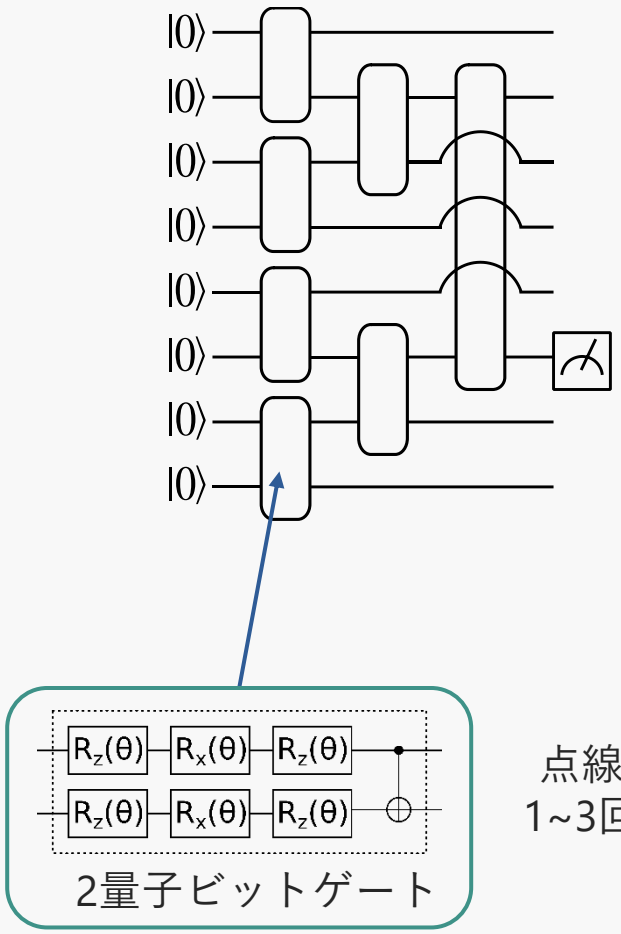
実際には，使用するテンソルネットワーク構造やゲート構成により
性能は大きく変化.

学習後のlossの値

繰り返し回数	1回	2回	3回
64qubit	262.32	30.26	37.22
256qubit	253.29	15.47	20.19

学習後のテストデータへの正答率

繰り返し回数	1回	2回	3回
64qubit	82.56%	99.56%	99.31%
256qubit	87.5%	100%	99.75%



まとめ

QTensorNet

- 通常の量子回路シミュレータと全く同じように使うことができる.
- ゲートのかけ方次第では，今までの量子回路シミュレータより**大規模な系**に対し，**高速に期待値を計算する**ことができる.
- 機械学習の**分類・回帰タスク**をこなすことができる.
- Jaxと組み合わせることで，**柔軟なモデル設計**が可能となる.
- これらの機能を，テンソルネットワークの知識なしで**誰でも簡単に利用**することができる.

今後の予定

- QTensorNetのメンテナンスは今後も続行
  ・不具合の対処やUI，アルゴリズムの改善
  ・最新手法の導入
- 京都大学第15回ICTイノベーションのパネル展示への参加
  (http://ict-nw.i.kyoto-u.ac.jp/ict-innovation/panel/266/)
- QTensorNetを用いた研究
  ・どのような量子回路の構成が性能が良いのか？
  ・テンソルネットワーク構造を用いたときどの程度の性能が出るのか？
  →近いうちにまとめ，適切な場所で発表