# Qiskit Finance Tutorial

Yuma Nakamura

Qiskit Advocate

Last updated: 2021-04-09

Quantum Tokyo

# Agenda

1. Option Pricingとは？
2. qGANによる確率分布の再現
3. Payoff関数のエンコード
4. まとめ

# Option Pricingとは?

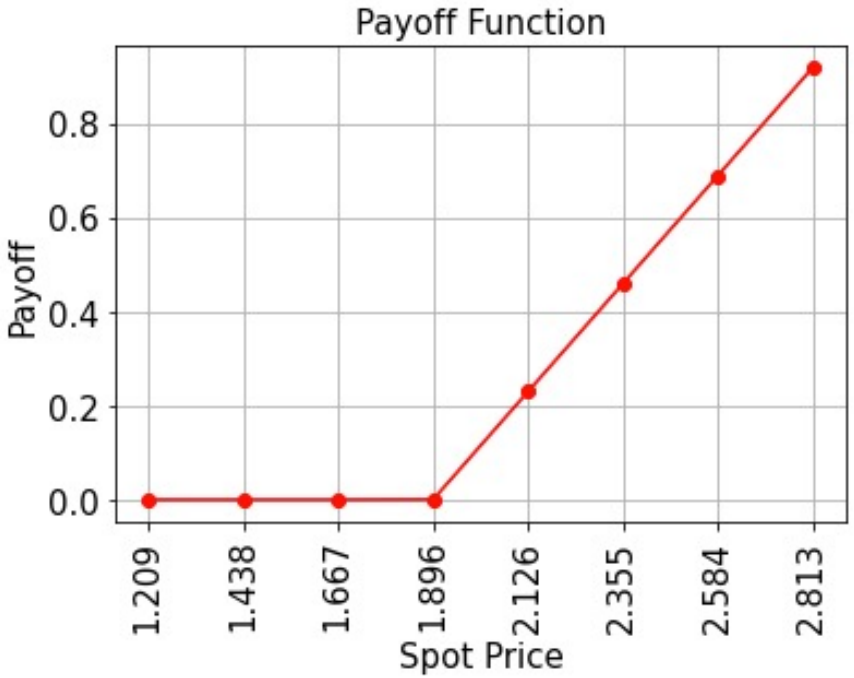
Optionとは満期に株を権利行使価格で買う/売る権利(義務ではない)

権利行使価格: K
満期のスポット価格: $S_T$

このときPayoff関数は

$$\max\{S_T - K, 0\}$$

つまり買い(Call Option)では
・スポット価格が権利行使価格より高い場合、利益が発生
・スポット価格が権利行使価格より低い場合、損益なし(買う義務はないので)




この権利をいくらで取引するかを決める問題がOption Pricing

# Option Pricingとは?　--例題--

Question:

$10で株AのCall Option(満期 2021年10月、権利行使価格$100)を買ったとする

シナリオ1

2021年10月の株Aの価格が$130になった

→利益/損失は？

シナリオ2

2021年10月の株Aの価格が$80になった

→利益/損失は？

*ここでは利息は考えていない。実際は今買った場合の利息分を利益から差引く。
　利息分の計算は古典計算で十分なのでチュートリアル・原著論文では割愛されている

# Option Pricingとは?　--例題--

Question:

$10で株AのCall Option(満期 2021年10月、権利行使価格$100)を買ったとする

シナリオ1

2021年10月の株Aの価格が$130になった

→利益/損失は？

max($130 - $100,0) - $10= $20

$20の利益

シナリオ2

2021年10月の株Aの価格が$80になった

→利益/損失は？

max($80 - $100, 0) - $10= -$10

$10の損失

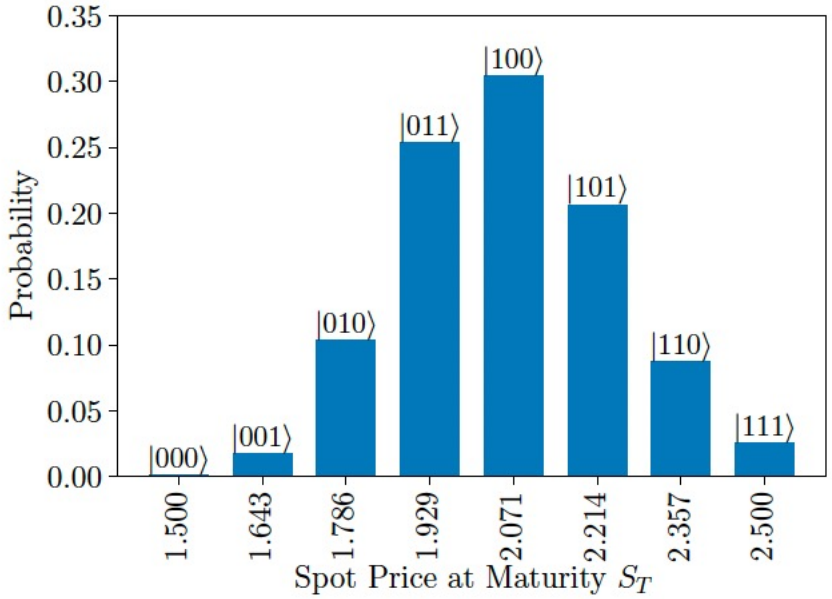
*ここでは利息は考えていない。実際は今買った場合の利息分を利益から差引く。
利息分の計算は古典計算で十分なのでチュートリアル・原著論文では割愛されている

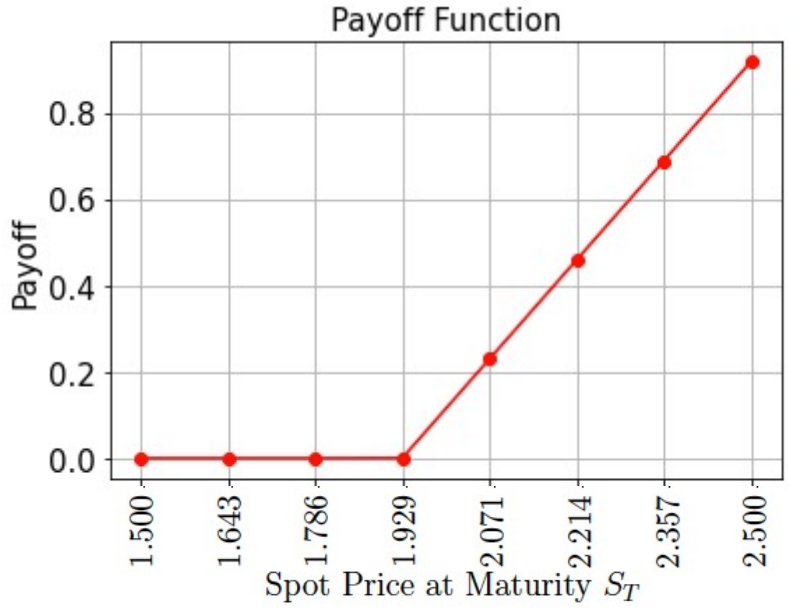
# Option Pricingとは?

## 満期のスポット価格の確率分布が与えられたとき、最適なOption価格は?

スポット価格の分布は対数正規分布で近似できる
(Black-Scholes model)

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x}\exp\left(-\frac{(\log x - \mu)^2}{2\sigma^2}\right)$$




→Payoff関数(右図)の期待値を価格の確率分布(左図)をもとに積分計算

# Option Pricing計算フロー

(1) 確率分布$p_i$を量子状態にエンコード

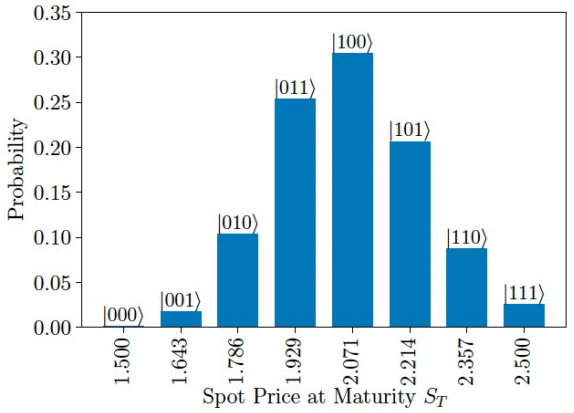
$$|\psi_1\rangle = \sum_{i=0}^{2^n-1} \sqrt{p_i}|i\rangle_n$$

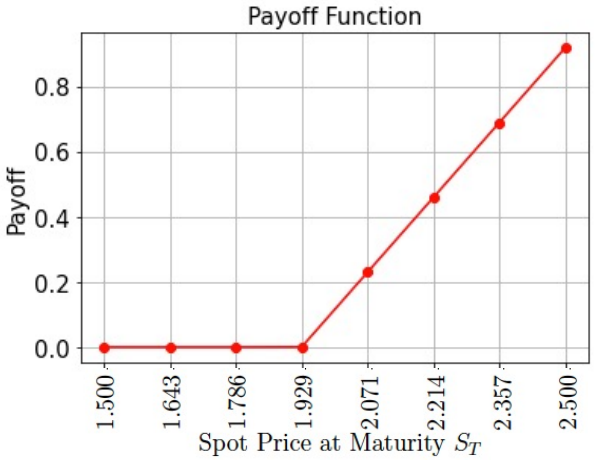
(2) Payoff関数 $f_i$を量子状態にエンコード

$$|\psi_2\rangle = \sum_{i=0}^{2^n-1} \sqrt{1-f_i}\,\sqrt{p_i}|i\rangle_n|0\rangle + \sum_{i=0}^{2^n-1} \sqrt{f_i}\sqrt{p_i}|i\rangle_n\,|1\rangle$$

(3) Payoffの期待値として振幅$\sum_{i=0}^{2^n-1}\left(\sqrt{f_i}\sqrt{p_i}\right)^2$ を計算

$$|\psi_2\rangle = \sum_{i=0}^{2^n-1} \sqrt{1-f_i}\,\sqrt{p_i}|i\rangle_n|0\rangle + \sum_{i=0}^{2^n-1} \sqrt{f_i}\sqrt{p_i}|i\rangle_n\,|1\rangle$$



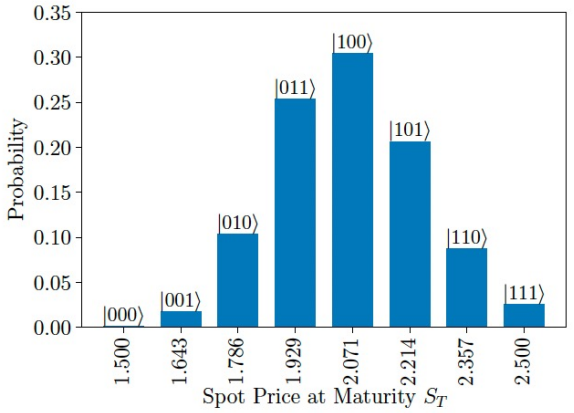



# Option Pricing計算フロー

(1) 確率分布$p_i$を量子状態にエンコード

$$|\psi_1\rangle = \sum_{i=0}^{2^n-1} \sqrt{p_i}|i\rangle_n$$

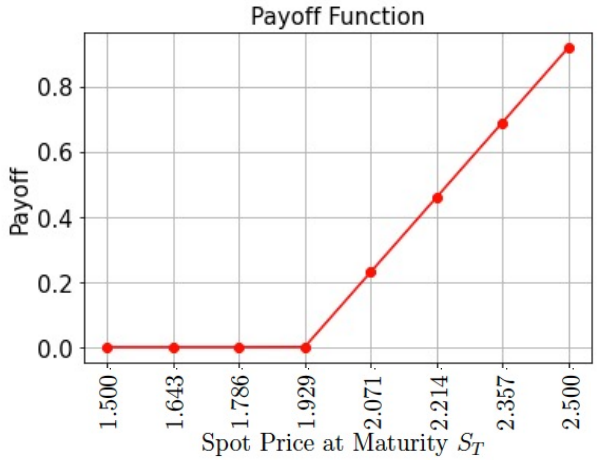
本日のメイン
Quantum GAN
(Machine Learning)




(2) Payoff関数 $f_i$を量子状態にエンコード

Controlled-Ry

$$|\psi_2\rangle = \sum_{i=0}^{2^n-1} \sqrt{1-f_i}\sqrt{p_i}|i\rangle_n|0\rangle + \sum_{i=0}^{2^n-1} \sqrt{f_i}\sqrt{p_i}|i\rangle_n|1\rangle$$




(3) Payoffの期待値として振幅$\sum_{i=0}^{2^n-1}\left(\sqrt{f_i}\sqrt{p_i}\right)^2$を計算

$$|\psi_2\rangle = \sum_{i=0}^{2^n-1} \sqrt{1-f_i}\sqrt{p_i}|i\rangle_n|0\rangle + \sum_{i=0}^{2^n-1} \sqrt{f_i}\sqrt{p_i}|i\rangle_n|1\rangle$$

Amplitude Estimation
(詳しくは田中さんが後日発表)

# Quantum GAN 価格確率分布の量子状態作成

量子敵対的生成ネットワークなるものを使って確率分布を作成する

npj | Quantum Information

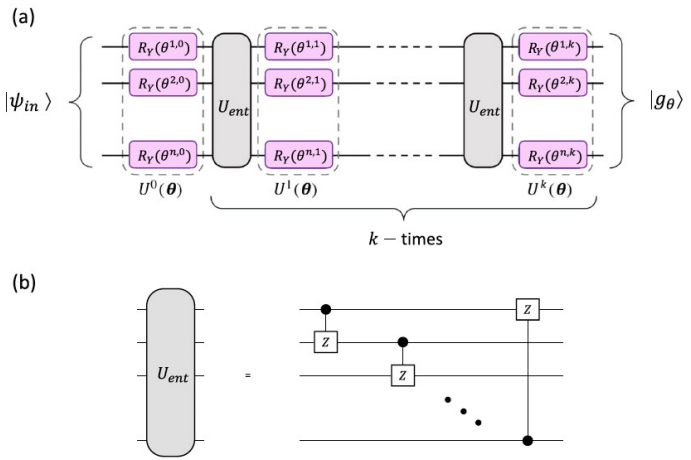
www.nature.com/npjqi

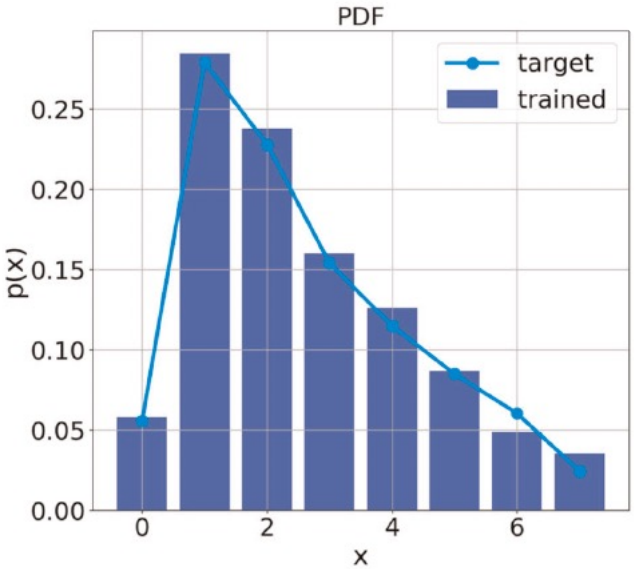
ARTICLE OPEN

## Quantum Generative Adversarial Networks for learning and loading random distributions

Christa Zoufal[1,2*], Aurélien Lucchi[2] and Stefan Woerner[1]

変分量子回路(Variational Quantum Circuit; VQC)




ref: C. Zoufal et al., 2019

所望の確率分布を実現するようにパラメータ$\theta_{ij}$を決定

# Quantum GAN 価格確率分布の量子状態作成

量子部分のGeneratorと古典部分のDiscriminatorを競合させながら学習をする

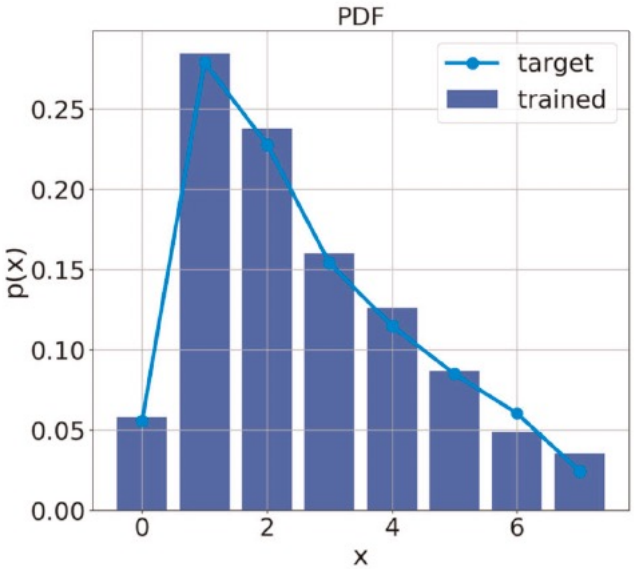
npj | Quantum Information

www.nature.com/npjqi

ARTICLE OPEN

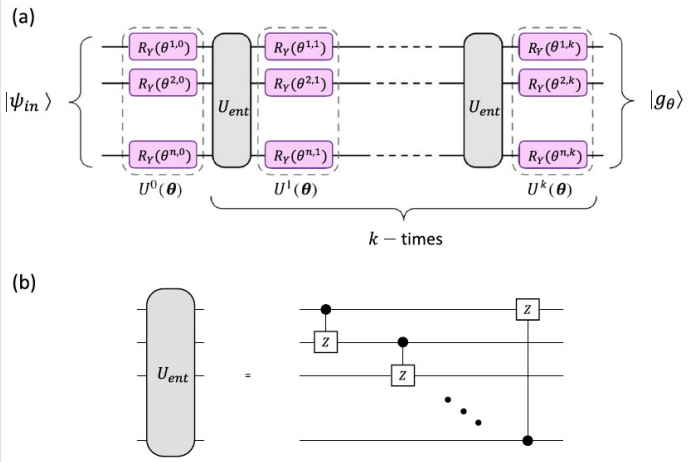
## Quantum Generative Adversarial Networks for learning and loading random distributions

Christa Zoufal[1,2*], Aurélien Lucchi[2] and Stefan Woerner[1]




ref: C. Zoufal et al., 2019

### VQC Generator circuit




ダミーの確率分布を生成

### Classical Discriminator

PYTORCH

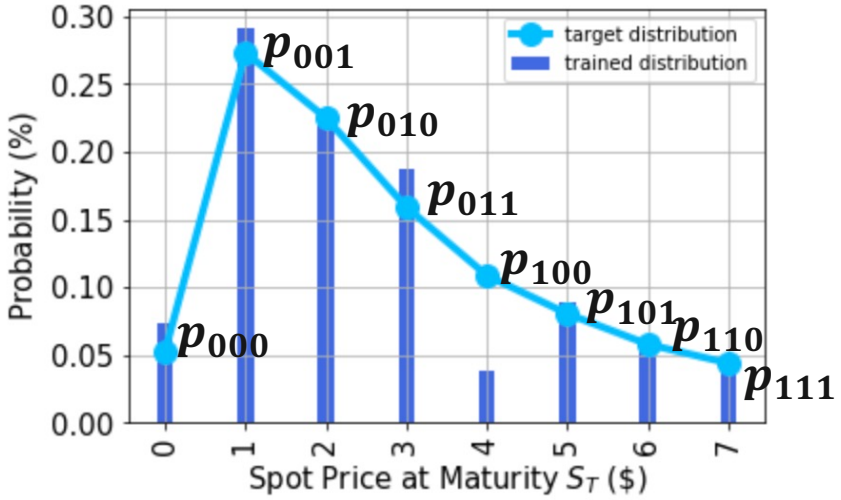
入力がダミーの確率分布か
本来の確率分布かを判定

競合させながら学習することで
高度なダミーの確率分布を生成

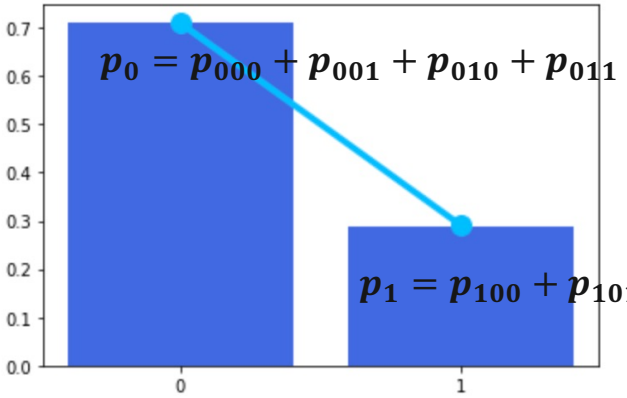
# なぜqGANを使うか？

価格の確率分布は制御Ryゲートを使うことで系統的にエンコードできるが$O(2^n)$のゲートが必要
qGANを使うと$O(poly(n))$のゲートで実装可能

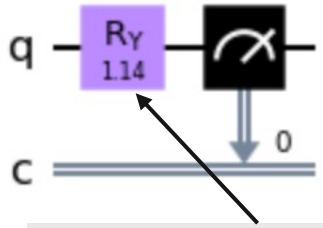
## 系統的アプローチ 1/3




確率分布を2分割に集約して再現




$$RY(\theta) = exp(-i\frac{\theta}{2}Y) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$




$$\theta_0 = 2\arccos(\sqrt{p_0})$$

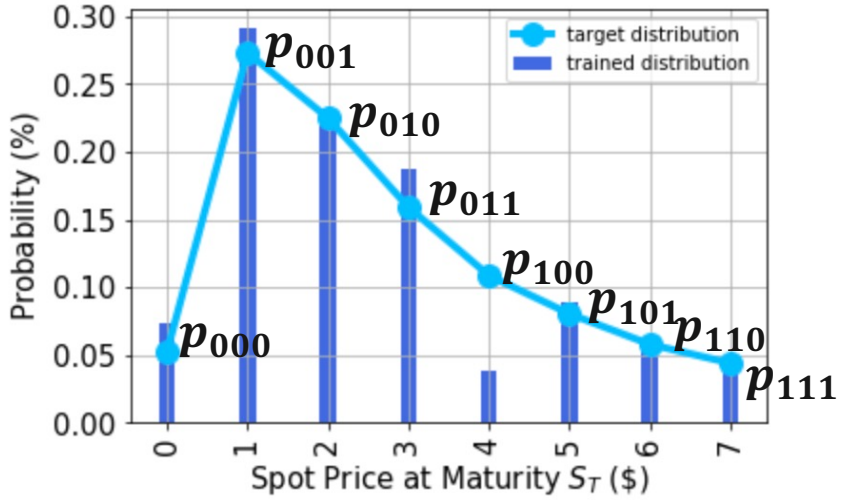
$$|\psi_1\rangle = Ry(\theta_0)|0\rangle = \cos\left(\frac{\theta_0}{2}\right)|0\rangle + \sin\left(\frac{\theta_1}{2}\right)|1\rangle$$

ref: https://qiskit.org/documentation/locale/ja_JP/tutorials/finance/10_qgan_option_pricing.html

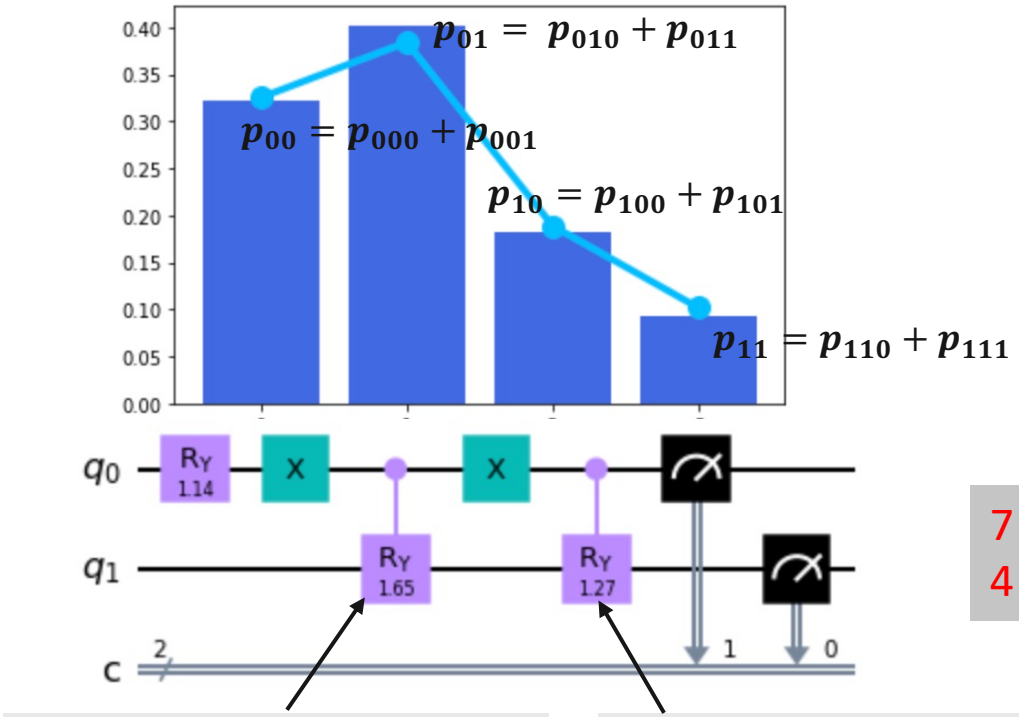
# なぜqGANを使うか？

価格の確率分布は制御Ryゲートを使うことで系統的にエンコードできるが$O(2^n)$のゲートが必要
qGANを使うと$O(poly(n))$のゲートで実装可能

## 系統的アプローチ 2/3




確率分布を4分割に集約し再現




7 u3 gates
4 cx gates

$\theta_1 = 2\arccos\left(\sqrt{p_{00}/p_0}\right)$

$\theta_2 = 2\arccos\left(\sqrt{p_{10}/p_1}\right)$

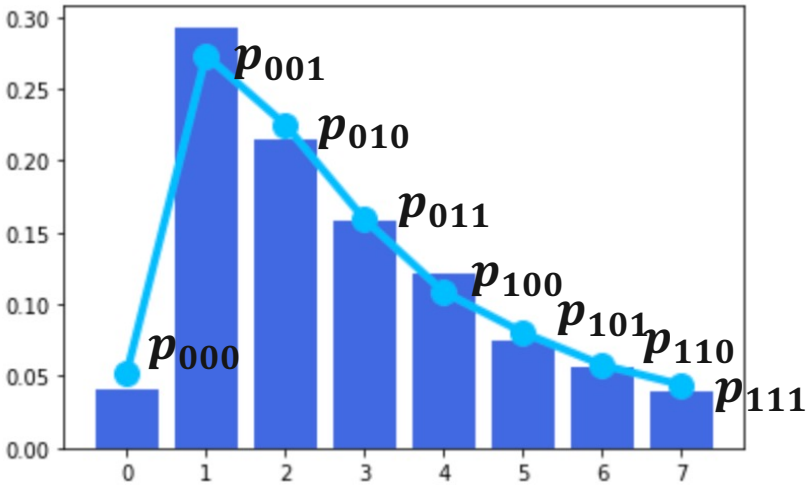
$$|\psi_1\rangle = \cos\left(\frac{\theta_0}{2}\right)|0\rangle\left(\cos\left(\frac{\theta_1}{2}\right)|0\rangle + \sin\left(\frac{\theta_1}{2}\right)|1\rangle\right) + \sin\left(\frac{\theta_0}{2}\right)|1\rangle\left(\cos\left(\frac{\theta_2}{2}\right)|0\rangle + \sin\left(\frac{\theta_2}{2}\right)|1\rangle\right)$$

ref: https://qiskit.org/documentation/locale/ja_JP/tutorials/finance/10_qgan_option_pricing.html

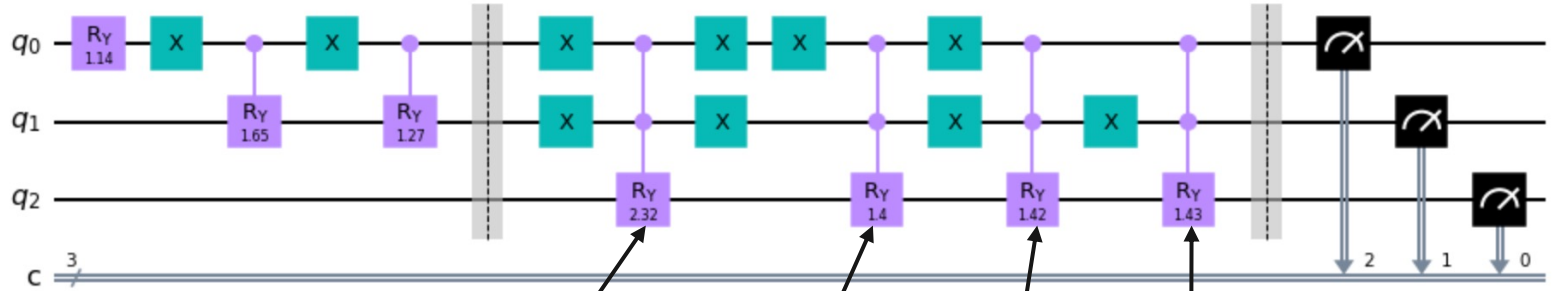
# なぜqGANを使うか？

価格の確率分布は制御Ryゲートを使うことで系統的にエンコードできるが$O(2^n)$のゲートが必要
qGANを使うと$O(poly(n))$のゲートで実装可能

## 系統的アプローチ 3/3




確率分布を8分割に集約し再現




$\theta_3 = 2\arccos\left(\sqrt{p_{000}/p_{00}}\right)$

$\theta_4 = 2\arccos\left(\sqrt{p_{010}/p_{01}}\right)$

$\theta_5 = 2\arccos\left(\sqrt{p_{100}/p_{10}}\right)$

$\theta_6 = 2\arccos\left(\sqrt{p_{100}/p_{11}}\right)$

95 u3 gates
52 cx gates

$$|\psi_1\rangle = \cos\left(\frac{\theta_0}{2}\right)\cos\left(\frac{\theta_1}{2}\right)|00\rangle\left(\cos\left(\frac{\theta_3}{2}\right)|0\rangle + \sin\left(\frac{\theta_3}{2}\right)|1\rangle\right) + \cos\left(\frac{\theta_0}{2}\right)\sin\left(\frac{\theta_1}{2}\right)|01\rangle\left(\cos\left(\frac{\theta_4}{2}\right)|0\rangle + \sin\left(\frac{\theta_4}{2}\right)|1\rangle\right)$$

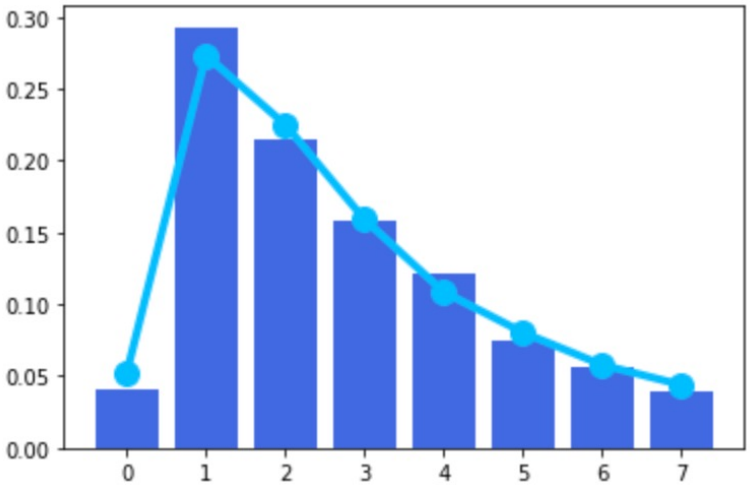
$$+ \sin\left(\frac{\theta_0}{2}\right)\cos\left(\frac{\theta_2}{2}\right)|10\rangle\left(\cos\left(\frac{\theta_5}{2}\right)|0\rangle + \sin\left(\frac{\theta_5}{2}\right)|1\rangle\right) + \sin\left(\frac{\theta_0}{2}\right)\sin\left(\frac{\theta_2}{2}\right)|11\rangle\left(\cos\left(\frac{\theta_6}{2}\right)|0\rangle + \sin\left(\frac{\theta_6}{2}\right)|1\rangle\right)$$

ref: https://qiskit.org/documentation/locale/ja_JP/tutorials/finance/10_qgan_option_pricing.html

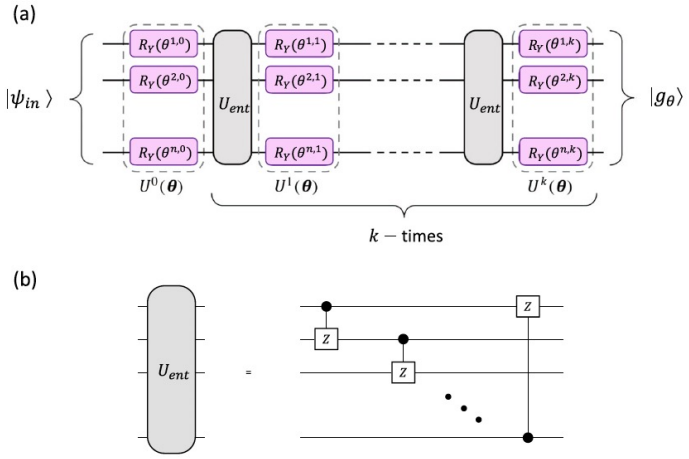
# なぜqGANを使うか？

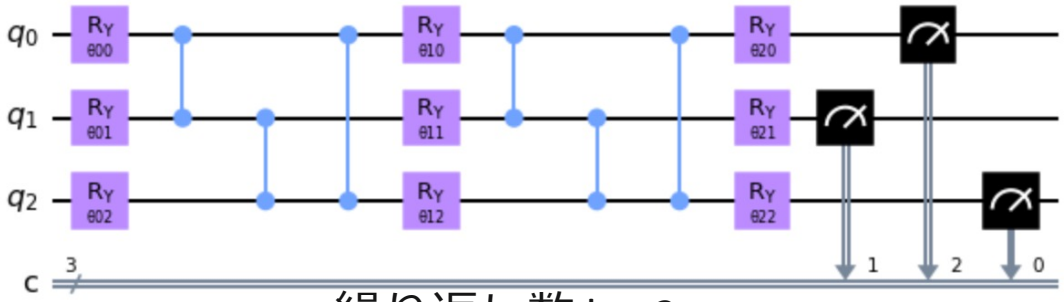
価格の確率分布は制御Ryゲートを使うことで系統的にエンコードできるが$O(2^n)$のゲートが必要
qGANを使うと$O(poly(n))$のゲートで実装可能

qGANを使った場合




VQC Generator circuit




繰り返し数 k = 2

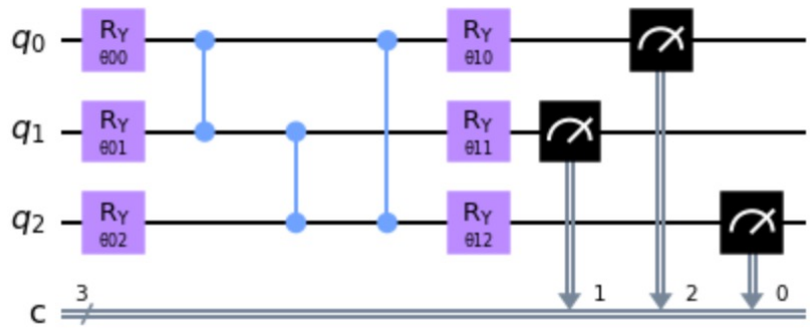
適切なパラメータ$\theta_{ij}$で確率分布を実現可能
→変分法により$\theta_{ij}$を決定

n(k+1) Ry gates
nk cz gates

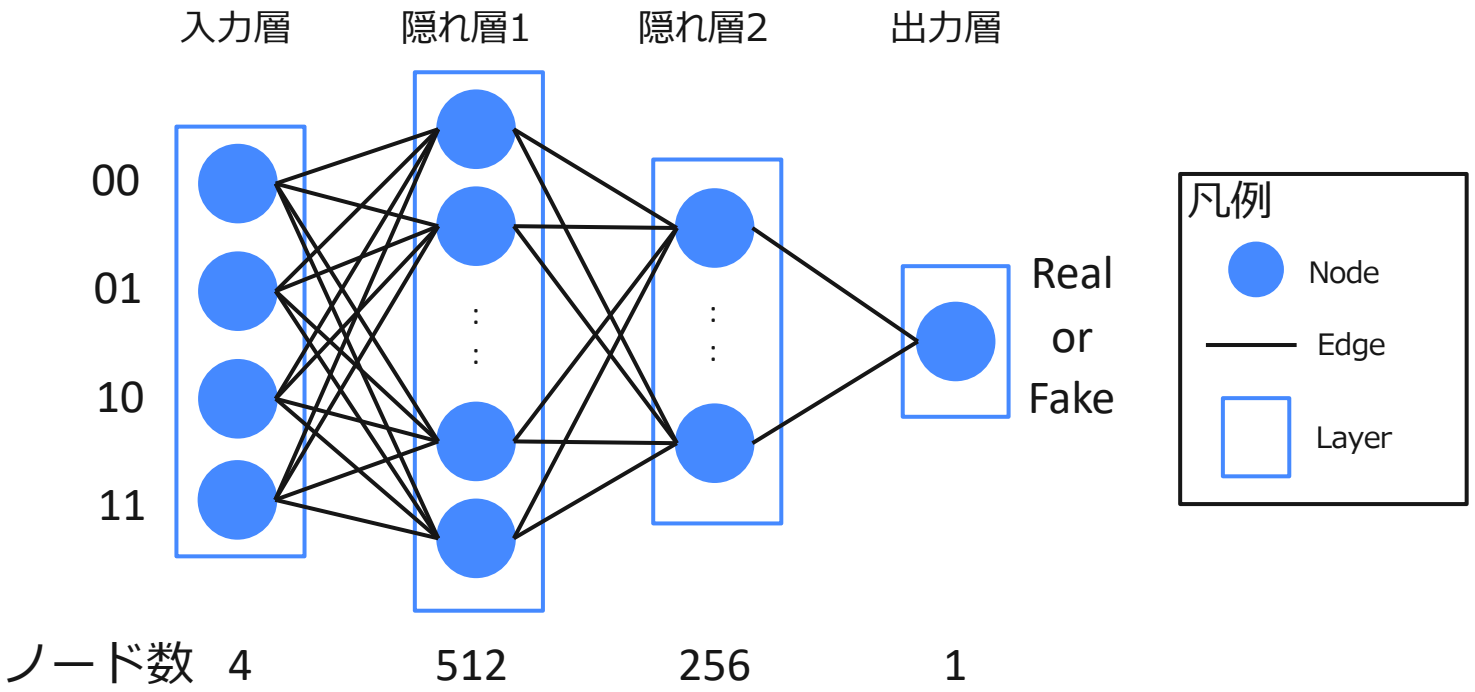
ref: https://qiskit.org/documentation/locale/ja_JP/tutorials/finance/10_qgan_option_pricing.html

# qGanのアーキテクチャー

Q-Generator




Discriminator




(1) Q-Generatorで確率分布を出力(shots数=batch size)

(2) 本来の確率分布とQ-Generatorが作る確率分布を区別できるようにDiscriminatorを訓練

最適化関数: $L_D(D_{\boldsymbol{\phi}}, G_{\boldsymbol{\theta}}) = \frac{1}{m}\sum_{l=1}^{m}[\log D_{\boldsymbol{\phi}}(\boldsymbol{x}^l) + \log(1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}^l)))]$

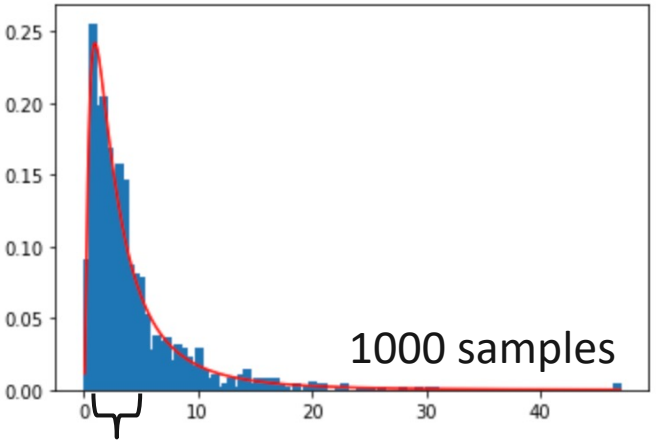
(3) Discriminatorがダミーの確率分布を正しいと判定するようにQ-Generatorを訓練

最適化関数: $L_G(\boldsymbol{\phi}, \boldsymbol{\theta}) = -\frac{1}{m}\sum_{l=1}^{m}[\log(D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(z^l)))]$

$D_\phi$: Discriminatorの分類結果
$x$: 本来の確率分布からの
サンプリング度数分布
$G_\theta(z)$: Generatorによる度数分布
$m$: Batchサイズ

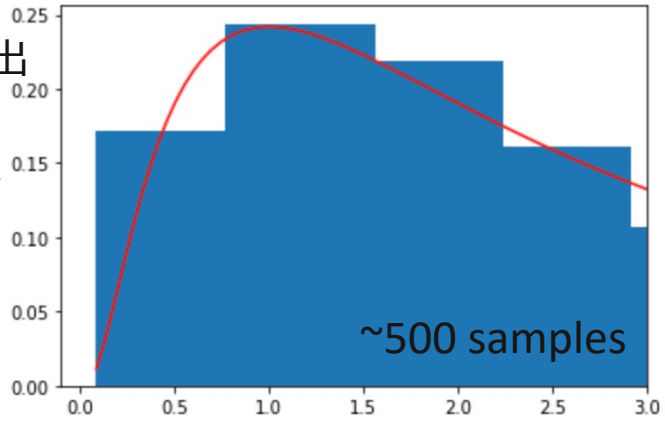
# qGanのアーキテクチャー –Discriminatorが使う正解データ

目的の確率分布からサンプリング

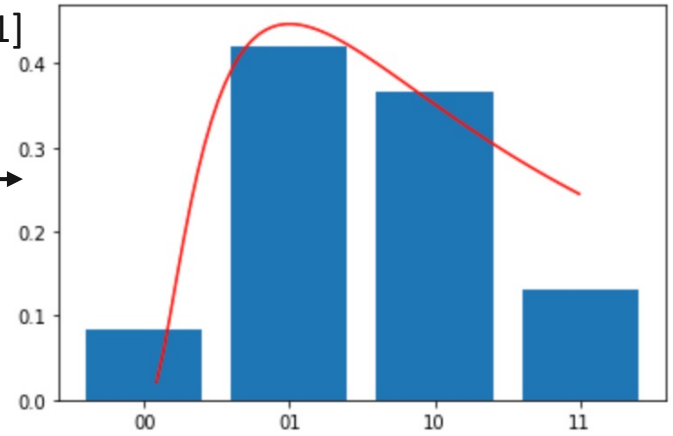



対象区間を[0,3]とする

対象区間の抽出
(truncate)




[00, 01, 10, 11]
で離散化




*確率分布曲線は区間内で再度規格化

サンプリング例: [1,2, 1, 0, 0, 1, 2, 3, ….]

~500 samples

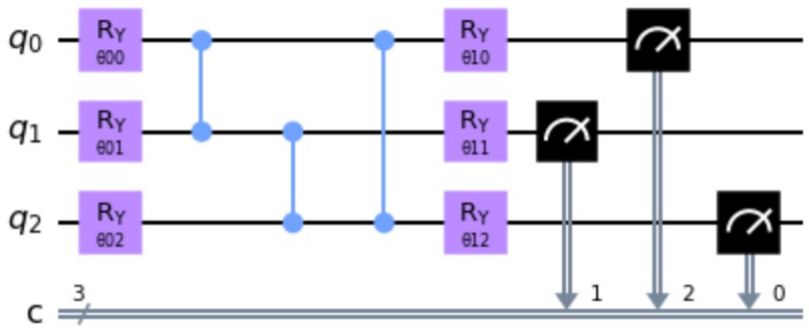
Batch*毎の度数分布を予測の特徴量としてDiscriminatorで学習

| Prob | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| batch_1 | 0.13 | 0.45 | 0.31 | 0.11 |
| batch_2 | 0.15 | 0.45 | 0.32 | 0.08 |
| batch_3 | 0.00 | 0.43 | 0.42 | 0.15 |
| batch_4 | 0.12 | 0.40 | 0.35 | 0.13 |
| batch_5 | 0.05 | 0.37 | 0.42 | 0.15 |
| batch_6 | 0.03 | 0.41 | 0.38 | 0.17 |

*batch size =100

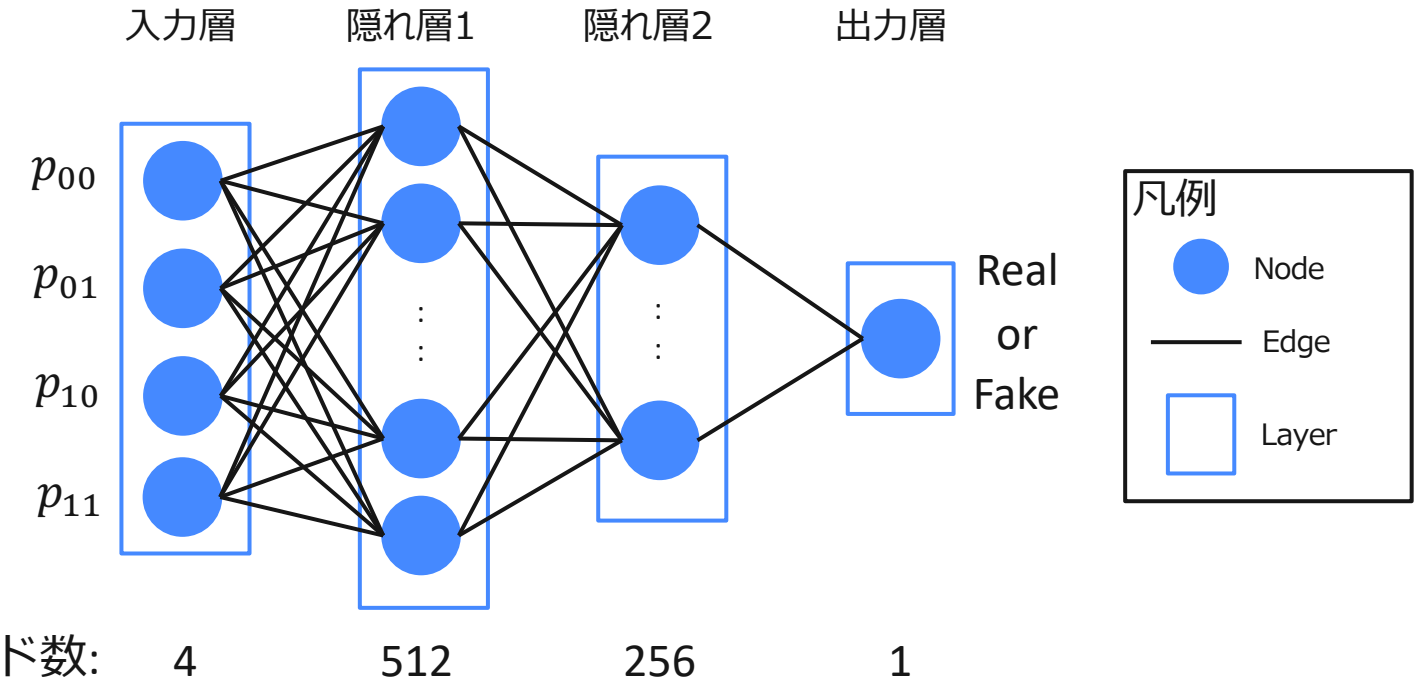
# qGanのアーキテクチャー

Q-Generator




RyとCZを使うのは$\theta = 0$で位相が変わらないため

Discriminator




(1) Q-Generatorで確率分布を出力(shots数=batch size)
(2) 本来の確率分布とQ-Generatorが作る確率分布を区別できるようにDiscriminatorを訓練

最適化関数: $L_D(D_{\phi}, G_{\theta}) = \frac{1}{m}\sum_{l=1}^{m}[\log D_{\phi}(\boldsymbol{x}^l) + \log(1 - D_{\phi}(G_{\theta}(\boldsymbol{z}^l)))]$

(3) Discriminatorがダミーの確率分布を正しいと判定するようにQ-Generatorを訓練

最適化関数: $L_G(\boldsymbol{\phi}, \boldsymbol{\theta}) = -\frac{1}{m}\sum_{l=1}^{m}[\log(D_{\phi}(G_{\theta}(z^l)))]$

$D_{\phi}$: Discriminatorの分類結果
$x$: 本来の確率分布からのサンプリング度数分布
$G_{\theta}(z)$: Generatorによる度数分布
$m$: Batchサイズ

# 再現度合いの指標

相対エントロピーによって本来の確率分布と生成された確率分布の差異を定量化

相対エントロピー

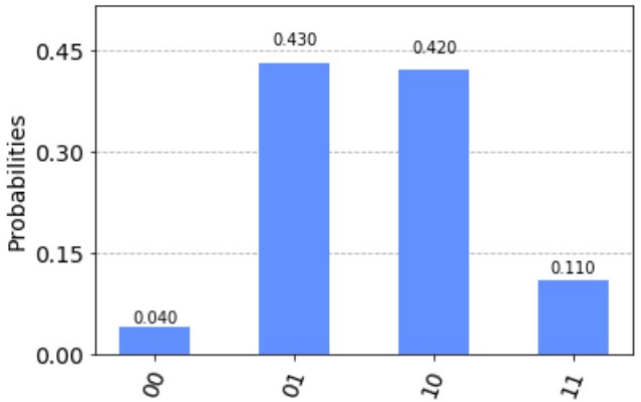
$$D_{\mathrm{KL}}(P\|Q) = \sum_j p_j \log \frac{p_j}{q_j}.$$

$p_j$ : 生成された確率分布

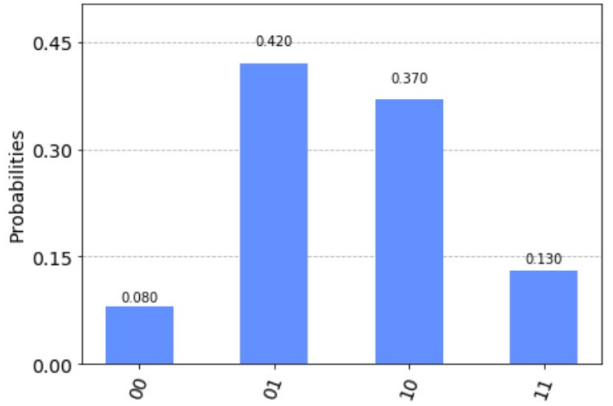
$q_j$ : 本来の確率分布

| Probability | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| Generated | 0.04 | 0.43 | 0.42 | 0.10 |
| Original | 0.08 | 0.42 | 0.37 | 0.13 |

Generated




Original Sampling

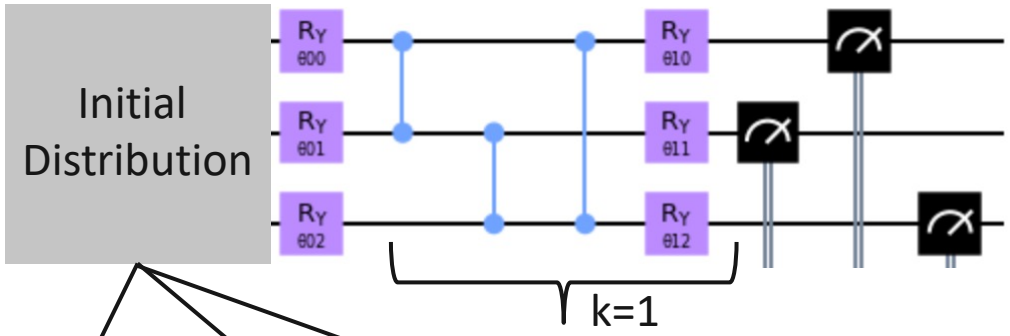



$$D_{KL} = 0.04 \log \frac{0.04}{0.08} + 0.43 \log \frac{0.43}{0.42} + 0.42 \log \frac{0.42}{0.37} + 0.10 \log \frac{0.10}{0.13} = 0.02$$

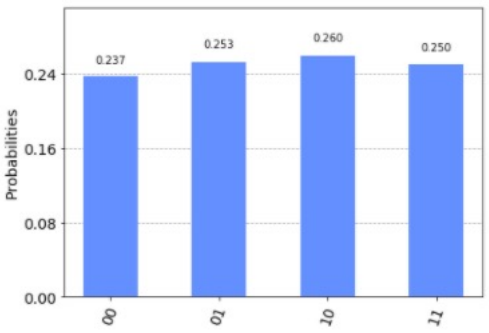
小さい値になるほど類似

# 変分量子回路Tips

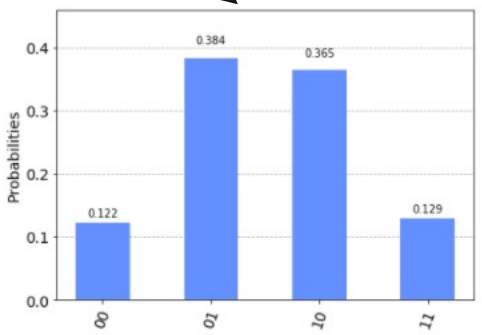
- 繰り返し数kを増やすと性能向上する傾向がある
- 初期化する分布が結果に大きく影響(正規分布が◎)




一様分布




正規分布




Random

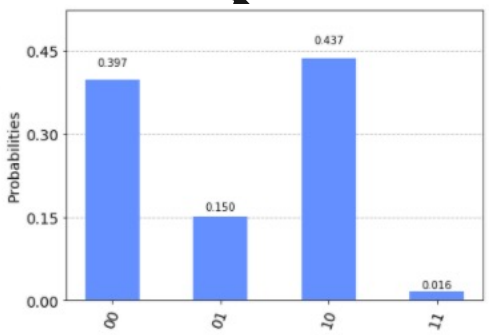



**Table 1.** Benchmarking the qGAN training.

| Data | Initialization | $k$ | $\mu_{KS}$ | $\sigma_{KS}$ | $n_{\leq b}$ | $\mu_{RE}$ | $\sigma_{RE}$ |
|---|---|---|---|---|---|---|---|
| Log-normal | Uniform | 1 | 0.0522 | 0.0214 | 9 | 0.0454 | 0.0856 |
| | | 2 | 0.0699 | 0.0204 | 7 | 0.0739 | 0.0510 |
| | | 3 | 0.0576 | 0.0206 | 9 | 0.0309 | 0.0206 |
| | Normal | 1 | 0.1301 | 0.1016 | 5 | 0.1379 | 0.1449 |
| | | 2 | 0.1380 | 0.0347 | 1 | 0.1283 | 0.0716 |
| | | 3 | 0.0810 | 0.0491 | 7 | 0.0435 | 0.0560 |
| | Random | 1 | 0.0821 | 0.0466 | 7 | 0.0916 | 0.0678 |
| | | 2 | 0.0780 | 0.0337 | 6 | 0.0639 | 0.0463 |
| | | 3 | 0.0541 | 0.0174 | 10 | 0.0436 | 0.0456 |
| Triangular | Uniform | 1 | 0.0880 | 0.0632 | 6 | 0.0624 | 0.0535 |
| | | 2 | 0.0336 | 0.0174 | 10 | 0.0091 | 0.0042 |
| | | 3 | 0.0695 | 0.1028 | 9 | 0.0760 | 0.1929 |
| | Normal | 1 | 0.0288 | 0.0106 | 10 | 0.0038 | 0.0048 |
| | | 2 | 0.0484 | 0.0424 | 9 | 0.0210 | 0.0315 |
| | | 3 | 0.0251 | 0.0067 | 10 | 0.0033 | 0.0038 |
| | Random | 1 | 0.0843 | 0.0635 | 7 | 0.1050 | 0.1387 |
| | | 2 | 0.0538 | 0.0294 | 9 | 0.0387 | 0.0486 |
| | | 3 | 0.0438 | 0.0163 | 10 | 0.0201 | 0.0194 |
| Bimodal | Uniform | 1 | 0.1288 | 0.0259 | 0 | 0.3254 | 0.0146 |
| | | 2 | 0.0358 | 0.0206 | 10 | 0.0192 | 0.0252 |
| | | 3 | 0.0278 | 0.0172 | 10 | 0.0127 | 0.0040 |
| | Normal | 1 | 0.0509 | 0.0162 | 9 | 0.3417 | 0.0031 |
| | | 2 | 0.0406 | 0.0135 | 10 | 0.0114 | 0.0094 |
| | | 3 | 0.0374 | 0.0067 | 10 | 0.0018 | 0.0041 |
| | Random | 1 | 0.2432 | 0.0537 | 0 | 0.5813 | 0.2541 |
| | | 2 | 0.0279 | 0.0078 | 10 | 0.0088 | 0.0060 |
| | | 3 | 0.0318 | 0.0133 | 10 | 0.0070 | 0.0069 |

# QGANの変分量子回路

対数正規分布を再現するための初期分布を一様分布/正規分布/ランダムを比較

20,000 samples
2000 epochs
AMSGRAD optimizer
learning late $10^{-4}$

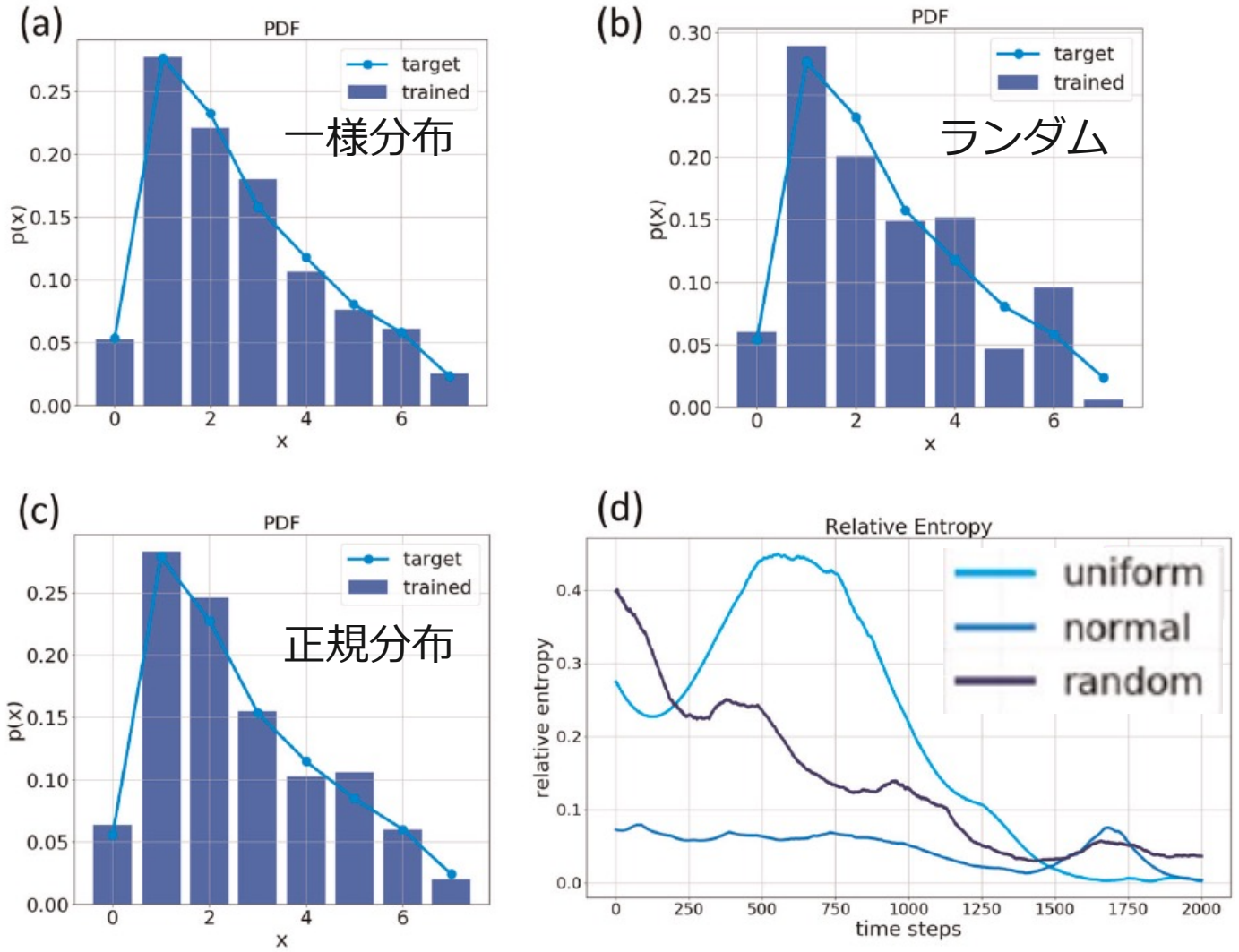



**Fig. 4** Simulation training. The figure illustrates the PDFs corresponding to $|g_{\theta}\rangle$ trained on samples from a log-normal distribution using a uniformly (**a**), randomly (**b**), and normally (**c**) initialized quantum generator. Furthermore, the convergence of the relative entropy for the various initializations over 2000 training epochs is presented (**d**).

# QGANの変分量子回路

正規分布を初期分布として対数正規分布/三角分布/双峰分布を再現

対数正規分布

三角分布

双峰分布

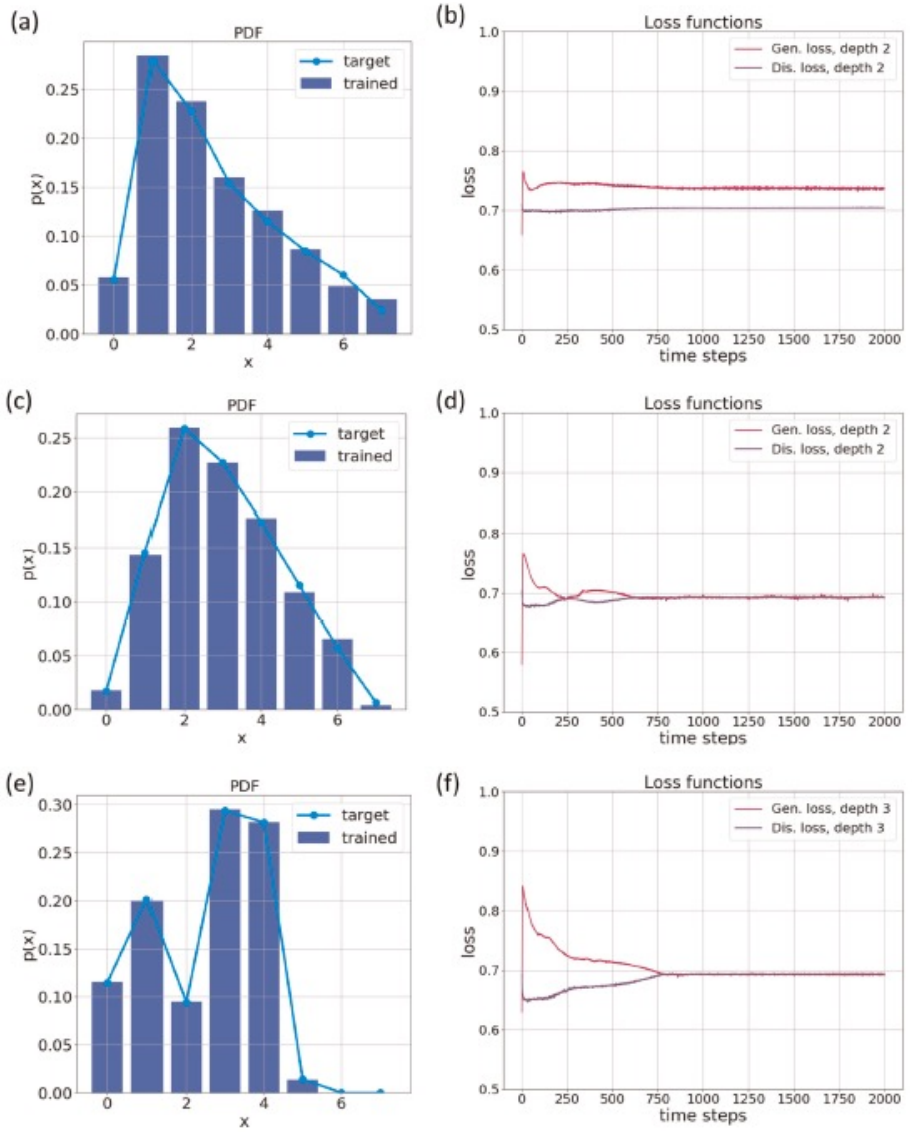



**Fig. 3** Benchmarking results for qGAN training. Log-normal target distribution with normal initialization and a depth 2 generator (**a**, **b**), triangular target distribution with random initialization and a depth 2 generator (**c**, **d**), and bimodal target distribution with uniform initialization and a depth 3 generator (**e**, **f**). The presented probability density functions correspond to the trained $|g_\theta\rangle$ (**a**, **c**, **e**) and the loss function progress is illustrated for the generator as well as for the discriminator (**b**, **d**, **f**).

# QGAN まとめ

・qGANを使うと価格の確率分布を再現でき、チュートリアルでは対数正規分布を再現
・qGANだと系統的に確率分布を作る場合に比べ少ないゲートで作成可能 $\left[O(2^n) \rightarrow O\left(poly(n)\right)\right]$
・量子部分のGeneratorと古典部分のDiscriminatorを競合させながら学習をする
競合させながら学習することで高度なダミーの確率分布を生成
・qGANのGeneratorの最適化関数$L_G(\phi,\theta)$とDiscriminatorの最適化関数$L_D\left(D_\phi, G_\theta\right)$は以下で定義

$$L_G(\boldsymbol{\phi},\boldsymbol{\theta}) = -\frac{1}{m}\sum_{l=1}^{m}\left[\log\left(D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}\left(z^l\right)\right)\right)\right]$$

$$L_D\left(D_{\boldsymbol{\phi}}, G_{\boldsymbol{\theta}}\right) = \frac{1}{m}\sum_{l=1}^{m}\left[\log D_{\boldsymbol{\phi}}\left(\boldsymbol{x}^l\right)+\log\left(1 - D_{\boldsymbol{\phi}}\left(G_{\boldsymbol{\theta}}\left(\boldsymbol{z}^l\right)\right)\right)\right]$$

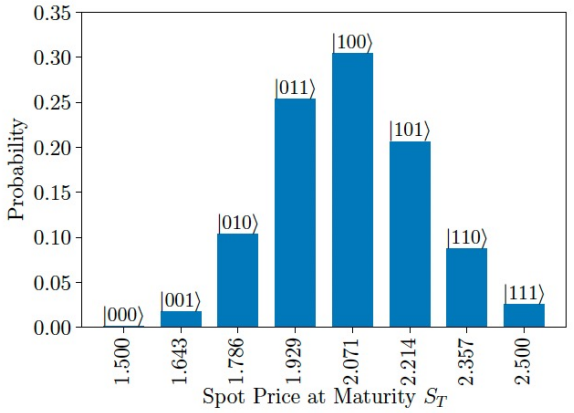
・再現度合いは相互エントロピーで評価
・初期分布の選択が重要で、正規分布の使用が推奨

# Option Pricing計算フロー

(1) 確率分布$p_i$を量子状態にエンコード

$$|\psi_1\rangle = \sum_{i=0}^{2^n-1} \sqrt{p_i}|i\rangle_n$$

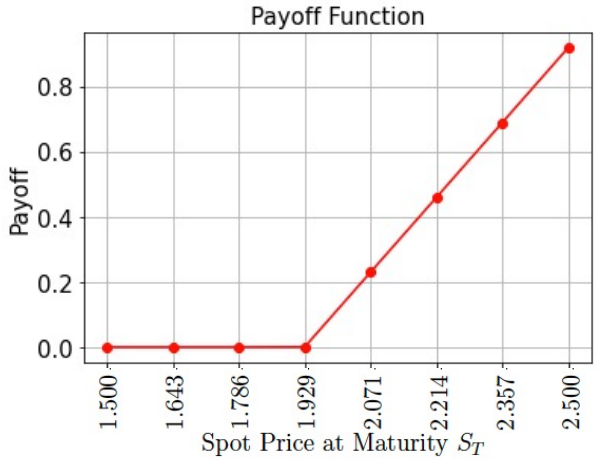
本日のメイン
Quantum GAN
(Machine Learning)




(2) Payoff関数 $f_i$を量子状態にエンコード

$$|\psi_2\rangle = \sum_{i=0}^{2^n-1} \sqrt{1-f_i}\,\sqrt{p_i}|i\rangle_n|0\rangle + \sum_{i=0}^{2^n-1} \sqrt{f_i}\sqrt{p_i}|i\rangle_n\,|1\rangle$$

Controlled-Ry




(3) Payoffの期待値として振幅$\sum_{i=0}^{2^n-1}\left(\sqrt{f_i}\sqrt{p_i}\right)^2$ を計算

Amplitude Estimation
(詳しくは田中さんが後日発表)

# Payoff関数のエンコード

$$\sum_{i=0}^{2^n-1}\sqrt{1-f(S_i)}\sqrt{p_i}\,|S_i\rangle\,|0\rangle+\sum_{i=0}^{2^n-1}\sqrt{f(S_i)}\sqrt{p_i}\,|S_i\rangle\,|1\rangle\,.$$

$f(S_i)$ : シナリオ$i$でのスポット価格$S_i$ のPayoff
$p_i$ : シナリオ$i$が起きる確率

## $f(S_i)$ をどうエンコードするか?

まずは線型関数だけ考えればよい$f(i)=f_1 i+f_0$
i.e. $|i\rangle_3=|i_2 i_1 i_0\rangle, i=4i_2+2i_1+i_0\in\{0,\dots,7\}$
$f(i)=4f_1 i_2+2f_1 i_1+f_1 i_0+f_0$
operator is,
$|i\rangle_n|0\rangle\rightarrow|i\rangle_n(\cos[f(i)]\,|0\rangle+\sin[f(i)]\,|1\rangle)$

(1) $p_i$のエンコード(qGANが実現)

$|\psi_1\rangle=\sum_{i=0}^{2^n-1}\sqrt{p_i}\,|i\rangle_n|0\rangle$

(2) $f(S_i)$のエンコード(Ryゲートを使用)

$|\psi_2\rangle=\sum_{i=0}^{2^n-1}\sqrt{p_i}\,|i\rangle_n(\cos[f(i)]\,|0\rangle+\sin[f(i)]\,|1\rangle)$

(3) 定数cを使って$f$ を $\tilde{f}$でスケール化

$|\psi_3\rangle=\sum_{i=0}^{2^n-1}\sqrt{p_i}\,|i\rangle_n\left(\cos\left[c\tilde{f}(i)+\frac{\pi}{4}\right]|0\rangle+\sin\left[c\tilde{f}(i)+\frac{\pi}{4}\right]|1\rangle\right)$ where $\tilde{f}(i)=2\frac{f(i)-f_{min}}{f_{max}-f_{min}}-1$

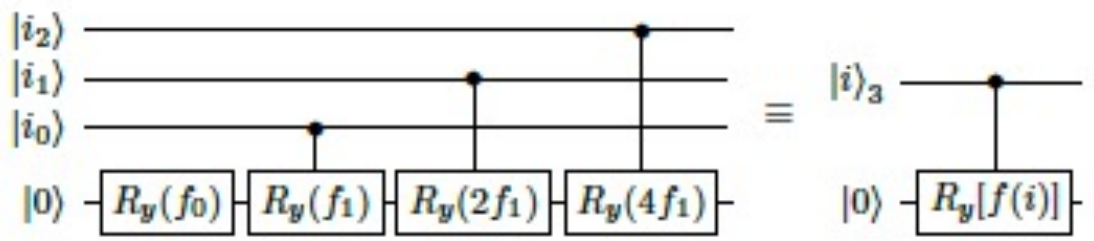



Figure 2: Quantum circuit that creates the state in Eq. (9). Here, the independent variable $i=4i_2+2i_1+i_0\in\{0,\dots,7\}$ is encoded by three qubits in the state $|i\rangle_3=|i_2 i_1 i_0\rangle$ with $i_k\in\{0,1\}$. Therefore, the linear function $f(i)=f_1 i+f_0$ is given by $4f_1 i_2+2f_1 i_1+f_1 i_0+f_0$. After applying this circuit the quantum state is $|i\rangle_3\,[\cos(f_1 i+f_0)\,|0\rangle+\sin(f_1 i+f_0)\,|1\rangle]$. The circuit on the right shows an abbreviated notation.

# Payoff関数のエンコード

$$\sum_{i=0}^{2^n-1}\sqrt{1-f(S_i)}\sqrt{p_i}\,|S_i\rangle\,|0\rangle+\sum_{i=0}^{2^n-1}\sqrt{f(S_i)}\sqrt{p_i}\,|S_i\rangle\,|1\rangle\,.$$

$f(S_i)$ : シナリオ$i$でのスポット価格$S_i$ のPayoff
$p_i$ : シナリオ$i$が起きる確率

## $f(S_i)$ をどうエンコードするか?

まずは線型関数だけ考えればよい$f(i)=f_1 i+f_0$
i.e. $|i\rangle_3=|i_2 i_1 i_0\rangle, i=4i_2+2i_1+i_0\in\{0,\dots,7\}$
$f(i)=4f_1 i_2+2f_1 i_1+f_1 i_0+f_0$
operator is,
$|i\rangle_n|0\rangle\rightarrow|i\rangle_n(\cos[f(i)]\,|0\rangle+\sin[f(i)]\,|1\rangle)$

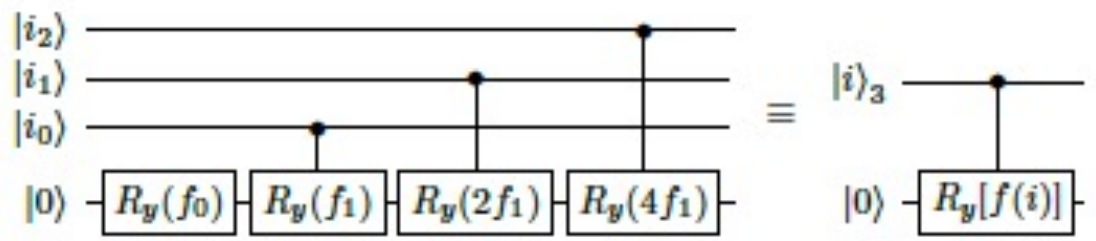
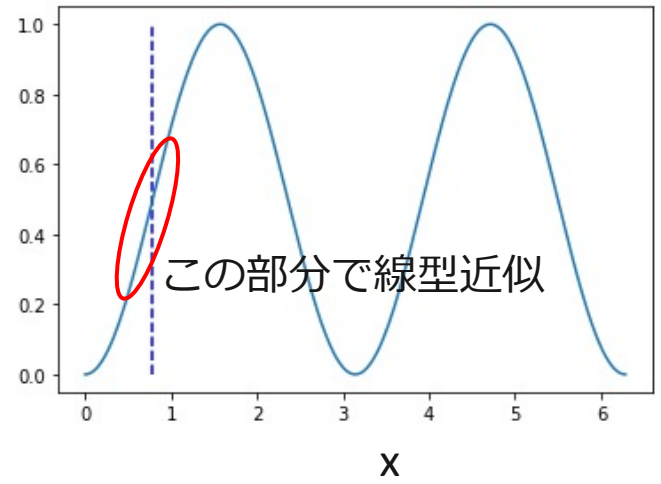



Figure 2: Quantum circuit that creates the state in Eq. (9). Here, the independent variable $i=4i_2+2i_1+i_0\in\{0,\dots,7\}$ is encoded by three qubits in the state $|i\rangle_3=|i_2 i_1 i_0\rangle$ with $i_k\in\{0,1\}$. Therefore, the linear function $f(i)=f_1 i+f_0$ is given by $4f_1 i_2+2f_1 i_1+f_1 i_0+f_0$. After applying this circuit the quantum state is $|i\rangle_3\,[\cos(f_1 i+f_0)\,|0\rangle+\sin(f_1 i+f_0)\,|1\rangle]$. The circuit on the right shows an abbreviated notation.

小さい$\tilde{f}(i)$ に対して、補助量子ビットが|1>を観測する確率

$$|\langle 1|\psi_3\rangle|^2=\sum_{i=0}^{2^n-1}p_i\sin^2\left[c\tilde{f}(i)+\frac{\pi}{4}\right]\approx\sum_{i=0}^{2^n-1}p_i\left[c\tilde{f}(i)+\frac{\pi}{4}\right]\approx \mathrm{c}\frac{2\mathrm{E}[\mathrm{f(X)}]-f_{min}}{f_{max}-f_{min}}-\mathrm{c}+\frac{1}{2}$$

$\sin^2(x)$

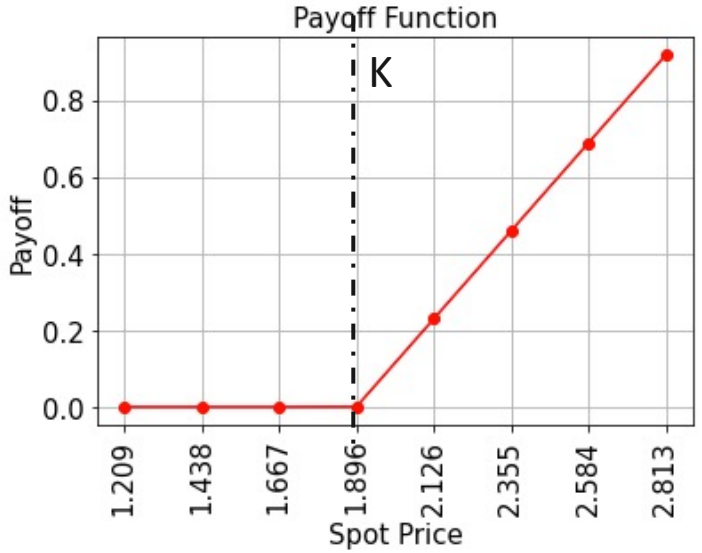


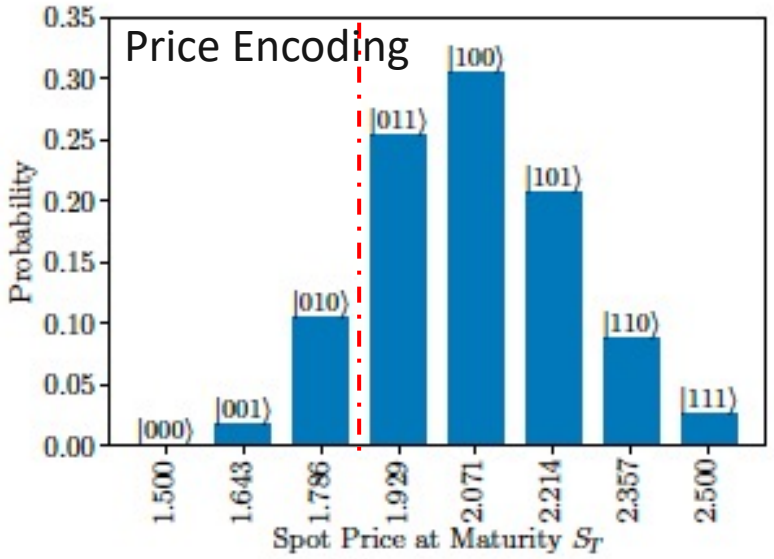

cをうまく選べば$O(M^{-2/3})$ で収束
高次の近似で$O(M^{-1})$も達成可能

$m$ (s.t. $M=2^m$) は量子ビット数

# Payoff関数のエンコード –Min/Maxのエンコード

真偽値テーブルを使って0部分と線型部分を分離可能



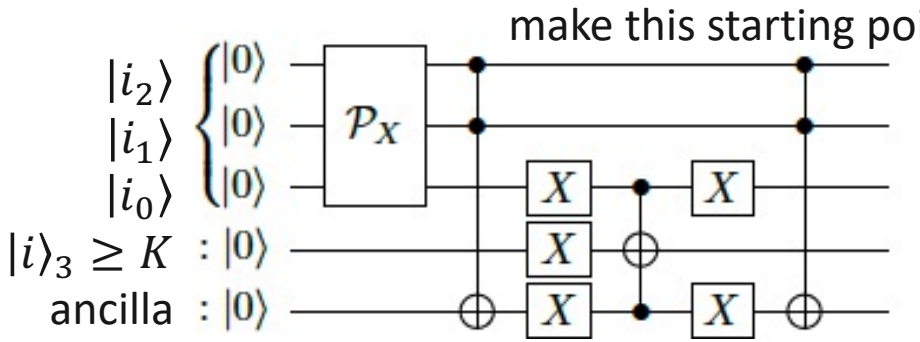

$$|\psi\rangle_n |0\rangle \rightarrow |\phi_1\rangle = \sum_{i<K} \sqrt{p_i} |i\rangle_n |0\rangle + \sum_{i\geq K} \sqrt{p_i} |i\rangle_n |1\rangle .$$

Implement Following Truth Table

$|i\rangle_3 = |i_2 i_1 i_0\rangle$, K=1.9

| Representing Price | $i_2$ | $i_1$ | $i_0$ | $\|i\rangle_3 \geq K$ |
|---|---|---|---|---|
| 1.500 | 0 | 0 | 0 | False |
| 1.643 | 0 | 0 | 1 | False |
| 1.786 | 0 | 1 | 0 | False |
| 1.929 | 0 | 1 | 1 | True |
| 2.071 | 1 | 0 | 0 | True |
| 2.214 | 1 | 0 | 1 | True |
| 2.357 | 1 | 1 | 0 | True |
| 2.500 | 1 | 1 | 1 | True |

make this starting point

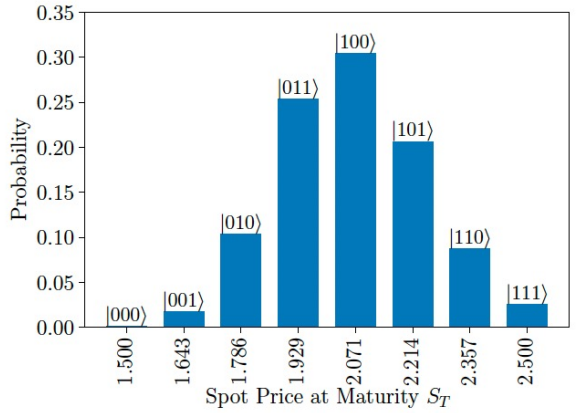


# Option Pricing計算フロー

(1) 確率分布$p_i$を量子状態にエンコード

本日のメイン
Quantum GAN
(Machine Learning)

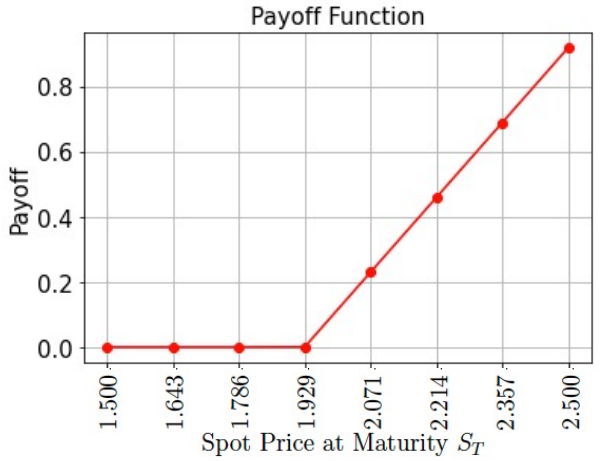
$$|\psi_1\rangle = \sum_{i=0}^{2^n-1} \sqrt{p_i}|i\rangle_n$$




(2) Payoff関数 $f_i$を量子状態にエンコード

Controlled-Ry

$$|\psi_2\rangle = \sum_{i=0}^{2^n-1} \sqrt{1-f_i}\sqrt{p_i}|i\rangle_n|0\rangle + \sum_{i=0}^{2^n-1} \sqrt{f_i}\sqrt{p_i}|i\rangle_n|1\rangle$$




(3) Payoffの期待値として振幅$\sum_{i=0}^{2^n-1}\left(\sqrt{f_i}\sqrt{p_i}\right)^2$ を計算

$$|\psi_2\rangle = \sum_{i=0}^{2^n-1} \sqrt{1-f_i}\sqrt{p_i}|i\rangle_n|0\rangle + \sum_{i=0}^{2^n-1} \sqrt{f_i}\sqrt{p_i}|i\rangle_n|1\rangle$$

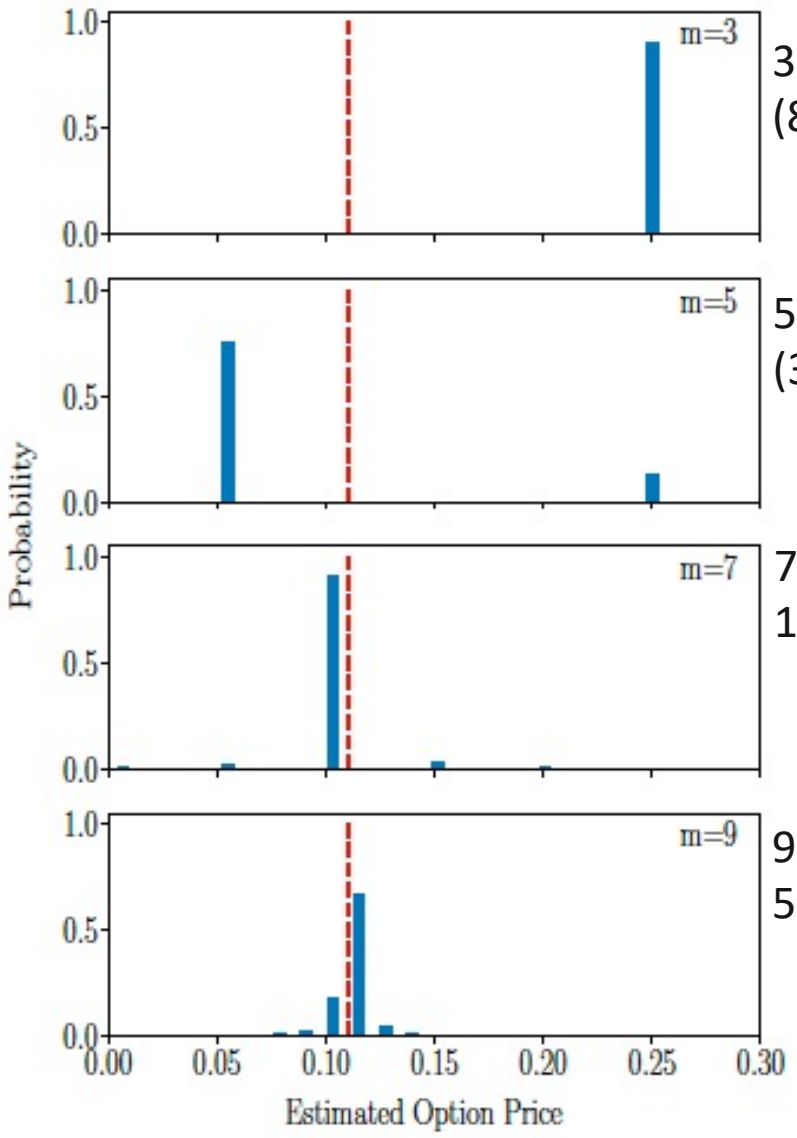
Amplitude Estimation
(詳しくは田中さんが後日発表)

box link:
https://ibm.ent.box.com/file/690593259852?s=bbbn6df18h7d1rcoiz4ut6pctau4nyhl

|1>の部分を測定しても振幅が得られるが精度/効率が△

# Option Pricingの精度

Amplitude Estimation(振幅推定)は量子ビットの数に応じて精度が向上




3量子ビット
(8サンプリングに対応)

5量子ビット
(32サンプリングに対応)

7量子ビット
128サンプリングに対応

9量子ビット
512サンプリングに対応

| # | Single-qubit | CX | CCX | Depth |
|---|---|---|---|---|
| $m = 3$ | 2,091 | 2,056 | 90 | 3,927 |
| $m = 5$ | 12,768 | 9,078 | 378 | 17,332 |
| $m = 7$ | 52,275 | 37,132 | 1,530 | 70,916 |
| $m = 9$ | 210,144 | 149,290 | 6,138 | 285,204 |

Table 2: Single-qubit, CNOT, Toffoli gate counts and overall circuit depth required for the full amplitude estimation circuits for each instance in Fig. 8, as a function of the number of sampling qubits $m$. These figures assume all-to-all connectivity across qubits.

qGAN部分の使用ゲート数は削減されたが、
Payoff関数のエンコード, 振幅推定部分の
部分で必要ゲート数が指数的な気が。。。
→改善の余地あり!?

# Option Pricingまとめ

・qGANを使って価格の確率分布を量子状態としてエンコード

・Payoff関数をCos関数の線型近似&ロジック回路でエンコード

・欲しい部分の振幅をAmplitude Estimationで計算