量子コンピュータの回帰問題への応用

name

1　背景

ニューラルネットワーク (以下、NN) を用いた機械学習が近年注目されている．NN の欠点の一つとして、計算コストが高いことが挙げられる．NN は行列積の計算を多用するため、現在では GPU を用いた並列計算により高速化が図られているが、それでも大規模な計算機資源での長期間の計算時間を要することが多い．

一方で、量子コンピュータはサイズの大きい行列積を高速に計算できることが期待されている。この特徴を生かし、量子コンピュータを用いた NN の高速化を目指しているのが QCL(Quantum Curcuit Learning) である．

2　原理

2.1　Nerual Network

NN の概要について、図 1 に示す。

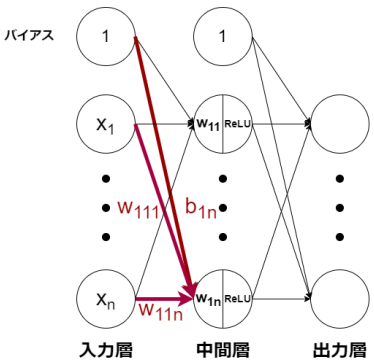

図 1: NN の概要図

NN は入力層と出力層の間に複数の中間層が存在する。第 j 層目の k 番目のノードへの入力を i_{jk}, 出力を o_{jk} とすると、それらは式 (1) で表される。

$$\begin{cases} i_{jk} = \sum_{l=1}^{n} w_{jkl} \cdot o_{j-1,l} + b_{jk} \\ o_{jk} = \max(0, i_{jk}) (ReLU \text{ 関数}) \end{cases} \tag{1}$$

式 (1) のように、NN のノードへの入力は、前層のノードによる出力と定数項の線形結合で表される。また、出力は入力に何らかの非線形関数を適用したものとなり、ReLU 関数 (Restified Linear Unit) が用いられることが多い。非線形関数の適用により、層を積み重ねることによって、NN の表現力が向上する。そのため、NN の層を深くした深層学習が注目を集めている。

式 (1) の入力は、式 (2) の定義を用いることで、行列積を用いる式 (3) へと書き換えることができる。

$$\begin{aligned} \boldsymbol{w_{jk}^t} &= \begin{pmatrix} b_{jk} & w_{jk1} & w_{jk2} & \cdots & w_{jkn} \end{pmatrix} \\ \boldsymbol{o_j^t} &= \begin{pmatrix} 1 & o_{j1} & o_{j2} & \cdots & o_{jn} \end{pmatrix} \\ \boldsymbol{i_j^t} &= \begin{pmatrix} i_{j1} & i_{j2} & \cdots & i_{jn} \end{pmatrix} \\ M &= \begin{pmatrix} \boldsymbol{w_{j1}} & \boldsymbol{w_{j2}} & \cdots & \boldsymbol{w_{jn}} \end{pmatrix} \end{aligned} \tag{2}$$

$$\boldsymbol{i_{j+1}} = M\boldsymbol{o_j} \tag{3}$$

大きさが n の行列積の計算は、単純な方法で $O(n^3)$ となる。そのため、背景でも触れた通り、NN で用いられるサイズの大きい行列積の計算は、計算コストが高い。

教師あり学習では、以下の手順で NN の学習を行う。

1. ランダムな初期値で NN の重みを初期化する。
2. 入力データ x を入力し、NN の出力 y を計算する。
3. 出力 y と教師データ t の損失関数で計算する。
4. 誤差を元に、NN の重みを更新する。

5. 2-4 を繰り返す。

教師あり学習は、重みを引数とした損失関数の最小化問題として定式化できる。一般的に、重みの自由度は非常に大きくなることが大きいため、確率的勾配降下法 (SGD) などの最適化手法が用いられる。その過程で、非常に多くの回数の損失関数の計算、つまり、行列積の計算が必要となる。

2.2 回帰問題

回帰問題とは、教師あり学習の一つであり、図 2 のように、ノイズを含んだ離散的な教師データ $t(x)$(青点) より、連続的な関数 $y(x)$(オレンジ線) を推定する問題である。

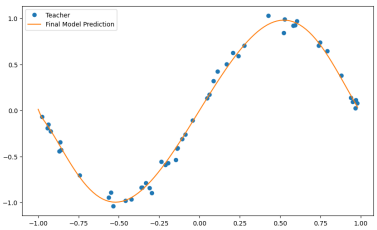

図 2: 回帰問題

近似の精度 L は、式 (4) で表される損失関数 (2 乗誤差) を用いて評価される。なお N は教師データ数である。

$$L = \frac{1}{N}\sum_{i=1}^{N}(y(x_i) - t(x_i))^2 \tag{4}$$

2.3 量子コンピュータ

古典的なコンピュータでは、電圧値で表現される単一ビット値は 2 値である。これに対して、量子ビットには光子・イオンなどの粒子の状態が対応しており、古典ビットにはなない特徴がいくつかある。

1 つ目の特徴は、量子ビットが式 (5) のベクトル $|\psi\rangle$ で表現されることである。古典ビットは 0/1 の 2 値論理であったが、量子ビットを記述するには複素数の重みを持つ 2 次元ベクトルが必要となる。$|0\rangle, |1\rangle$ はビット値が 0/1 の状態に対応する。

$$|\psi\rangle = \sum_{i=0}^{1} c_i |i\rangle = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} \tag{5}$$

NOT ゲートなどの論理ゲートは、図 3 のように、行列 U で記述され状態ベクトル $|\psi\rangle$ に作用後の状態は $U|\psi\rangle$ で記述される。

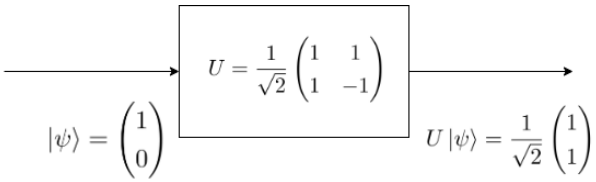

図 3: 1 入力論理ゲート

2 つ目の特徴は、複数の量子ビットの記述である。式 (6) より、量子ビットの自由度は 2 であるため、単純に考えれば、N 個の量子ビットの状態は $2N$ 次元のベクトルで記述できる。しかしながら、量子もつれという性質により、N 量子ビットの状態は 2^N 次元のベクトルで表現される。量子もつれとは、各量子ビット $|\psi_0\rangle, |\psi_1\rangle, \ldots, |\psi_{N-1}\rangle$ の合成で、N 量子ビット全体の状態ベクトル $|\psi\rangle$ が表せないという性質である。

$$|\psi\rangle = \sum_{i=1}^{2^N} c_i |i\rangle \tag{6}$$

そのため、量子ゲートは $2^N \times 2^N$ の行列で表現され、量子ビット数が大きくなると、その行列のサイズは指数関数的に大きくなる。

これらの特徴より、量子コンピュータを古典コンピュータでシミュレーションするには、大きなサイズの行列積を計算する必要があることが分かる。そのため、古典コンピュータでの計算量が大きい問題に対して、実物の量子コンピュータでの実験で求めることができることが期待されている。

3 つ目の特徴は、量子ビットの観測である。量子ビット $|\psi\rangle$ は観測を行うことによって、$|0\rangle$ あるいは $|1\rangle$ が観測される。どちらかが得られる確率は、式 (7) で表される。

$$|\psi\rangle = c_0|0\rangle + c_1|1\rangle \text{ において} \begin{cases} p_0 = |c_0|^2 \\ p_1 = |c_1|^2 \end{cases} \tag{7}$$

1 回の測定では、$|0\rangle$ / $|1\rangle$ のどちらかが得られるだけである。$|0\rangle$ / $|1\rangle$ を 1/-1 として、十分に多くの回数の測定を行うことで、測定結果の平均値を求める。理論的には測定の平均値は式 (8) で記述される。なお、Z は測定の種類を表す行列である。

$$\mu = \mathrm{tr}(Z\,|\psi\rangle\,\langle\psi|) \tag{8}$$

式 (8) の意味として、測定の平均値は実験的に求めることができるため、行列積の値を実験的に求めることができることができるという意味である。

2.4 QCL

QCL(Quantum Curcuit Learning) は、量子コンピュータ上で NN の実装を行う手法である。前述の通り、量子コンピュータおよび NN は、行列積の計算という共通点を持っており、量子回路と同様の行列計算を行う NN を考えることで、量子コンピュータ上で NN を実装することができる。

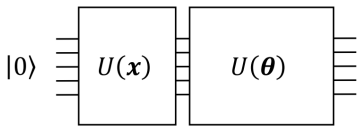

図 4: QCL の概要図

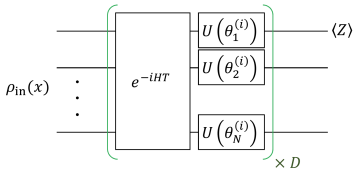

図 5: $U_{in}(\boldsymbol{\theta})$ の実装

量子ビット数が N のとき、量子ビットは 2^N 次元ベクトル、量子ゲートは $2^N \times 2^N$ の行列で表現される。図 4,5 にある $U(\boldsymbol{x}), U(\boldsymbol{\theta})$ は、パラメータとして θ を持つ量子ゲートを用いることで、観測結果をパラメータ θ の関数として表現することができる。回路と同様の計算を行う NN を図 6 に示す。注意点として、通常の NN の信号値が実数であるのに対し、今回は複素数であることある。

$$\begin{cases} f(x) = \dfrac{1}{2}\sqrt{1+\sqrt{1-x^2}}\,(\sqrt{1-x^2}+i\sqrt{1+x^2}) \\ g(x) = \dfrac{x}{2\,|x|}\sqrt{1-\sqrt{1-x^2}}\,(\sqrt{1-x^2}-i\sqrt{1+x^2}) \end{cases} \tag{9}$$

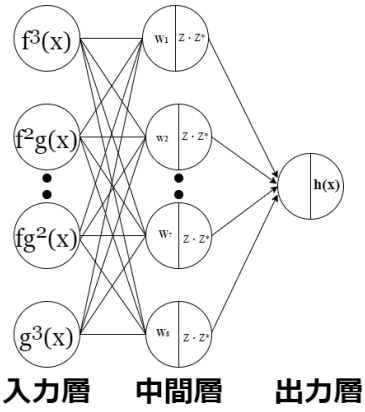

図 6: 量子回路と等価な NN(N=3)

図 6 では入力層が $U(x)$, 中間層への入力が $U(\boldsymbol{\theta})$、中間層の非線形関数と出力層が観測に対応している。各層のノード数は、量子回路を変更することで調整することができる。

NN の出力式 $o(x)$ は、式 (10) の形で表され、$N = 1$ のときは、式 (11) のようになる。ただし、a_k は $U(\theta)$ の行列表現に依存する係数である。

$$o(x) = \sum_{k=0}^{N} a_k \left| f(x)^k g(x)^{N-k} \right| \tag{10}$$

$$o(x) = A\sqrt{1-x^2} + Bx^3 + C\sqrt{1-x^4} \tag{11}$$

3 数値計算

QCL の実装として、回帰問題を解くための量子回路し、数値計算を行った。量子回路の数値計算は、行列の計算が行えればよいため、Python のライブラリである Numpy を用いれば実装可能であるが、既にゲートなどが実装されている Qulacs ライブラリを用いた。

3.1 問題設定

今回は、$\sin \pi x$ を学習する回帰問題を考える。教師データは図 7 のような (-1,1) の範囲で一様にランダムに生成した 50 個のデータ点に、ノイズを含んだ値を用いた。

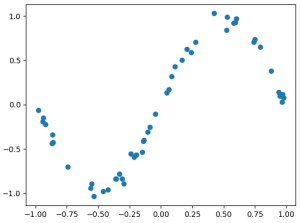

図 7: 教師データ

3.2 学習

学習は、教師データを用いて、損失関数の最小化を行うことで行う。損失関数は、式 (4) の 2 乗誤差を用いた。また、$\boldsymbol{\theta}$ の更新には、BFGS 法を用いた. なお、BFGS 法で用いる勾配は、量子回路を用いて計算できる。

3.3 結果

学習の結果を図 8 に示す。グラフより、適切に学習が行われていることが分かる。

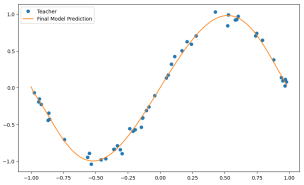

図 8: 回帰問題

式 (10) より、量子ビット数 N を増やすことで、出力式の表現力が向上することが期待されるため、$N = 1, 2, 3$ の場合について、学習を行った図 9 を示す。図 9 より、量子ビット数が増えるにつれ、学習の対象である関数 $\sin \pi x$ に近い関数が得られていることが分かる。

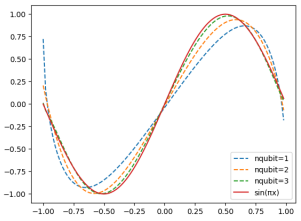

図 9: 量子ビット数の変化