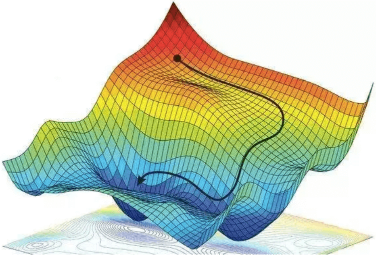
Generative Modeling using Matchgate Circuits

Peter Luo, Hong Ye Hu, Susanne F. Yelin

October 4, 2024

How Neural Networks Learn

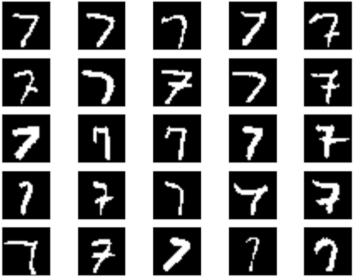
- Define a loss function $L(\boldsymbol{\theta})$ which captures how close the network's output is to the desired output



- We can use optimization algorithms to adjust $\boldsymbol{\theta}$ to get closer to the minimum of L
- Automatic differentiation is what allows modern ML models to learn

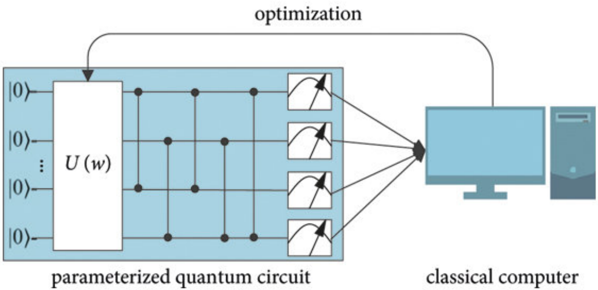
Generative Modeling

- Rather than doing a classification or prediction task, generative models strive to generate more instances of their training data
- Examples: ChatGPT, DALL-E, Synthetic Data Generation



Parametrized Quantum Circuits

- Quantum gates can be parametrized
- By replacing the neural network with a parametrized quantum circuit and then measuring its output, we obtain a hybrid quantum-classical system compatible with machine learning procedures



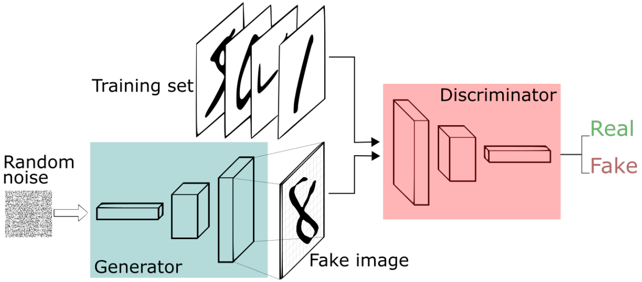
Quantum Circuits as Generative Models

- Parametrized quantum circuits (PQC) can be used as generative models
- Not a lot is known about how PQCs perform on such tasks, especially at large scale
- Solution: Matchgate circuits
- Continuously parametrized, efficiently simulatable

- Goal: To run a Matchgate circuit on generative tasks involving 49+ qubits
- Assess the generative capability of Matchgate circuits
- Test theoretical results pertaining to the large-qubit or overparametrized regime

Generative Adversarial Networks (GAN)

- Consists of two networks $G_{\boldsymbol{\theta}}$ and $D_{\boldsymbol{\phi}}$ with parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ and their own loss functions $L_G(\boldsymbol{\theta}, \boldsymbol{\phi})$ and $L_D(\boldsymbol{\theta}, \boldsymbol{\phi})$
- $G_{\boldsymbol{\theta}} : \{0,1\}^{28\times 28} \to \{0,1\}^{28\times 28}$ and $D_{\boldsymbol{\phi}} : \{0,1\}^{28\times 28} \to [0,1]$



- Real data $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ comes from an underlying probability distribution p_{real}
- The output of $G_{\boldsymbol{\theta}}$ are samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ from $p_{\boldsymbol{\theta}}$
- The loss functions are

$$L_G(\boldsymbol{\theta}, \boldsymbol{\phi}) = - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\boldsymbol{\theta}}} [\log(D_{\boldsymbol{\phi}}(\boldsymbol{x}))]$$

$$L_D(\boldsymbol{\theta}, \boldsymbol{\phi}) = - \left(\mathop{\mathbb{E}}_{\boldsymbol{y} \sim p_{\text{real}}} [\log(D_{\boldsymbol{\phi}}(\boldsymbol{y}))] + \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\boldsymbol{\theta}}} [\log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x}))] \right).$$

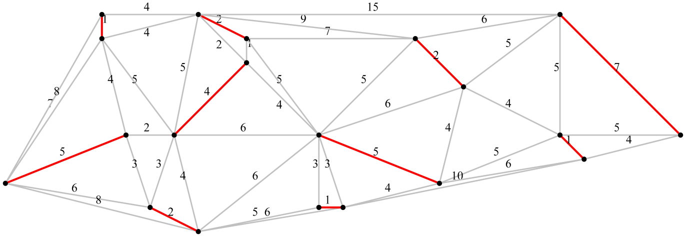
- Approximate with sample means in practice

Valiant's Definition

- Matchgates were first introduced by Valiant (2001)
- For a given undirected weighted graph G where $V(G) = [n]$, define the Pfaffian to be

$$\mathrm{Pf}(B) = \sum_{\substack{\text{perfect matchings} \\ M}} \mathrm{sgn}(M) \prod_{(ij)\in M} w_{ij},$$

where B is the adjacency matrix of G



- Define the Pfaffian sum to be

$$\mathrm{PfS}(B) = \sum_{A \subseteq [n]} \left(\prod_{i \in A} x_i \right) \mathrm{Pf}(B[A])$$

- Pfaffian sums are just Pfaffians:

$$\mathrm{PfS}(B) = \begin{cases} \mathrm{Pf}(B + \Lambda_n) & n \text{ even} \\ \mathrm{Pf}(B^+ + \Lambda_{n+1}) & n \text{ odd} \end{cases}$$

- Theorem (Cayley): $\mathrm{Pf}(B)^2 = \mathrm{Det}(B)$
- Pfaffians are computable in polynomial time
- Matchgates are Pfaffian sums where we set some x_i to 1 and all others to 0

Matchgate Speedup

- For a Matchgate circuit C,

$$|\langle \underline{y}| C |\underline{x}\rangle|^2 = |\mathrm{PfS}(M_{\underline{x},\underline{y}})|^2.$$

- Valiant: Computing Pfaffians and Pfaffian Sums are fast
- Terhal and DiVincenzo (2001): There is a correspondence to noninteracting fermions, and computing determinants is fast

Majorana Basis

- We can also understand Matchgate circuits in the context of free fermions.
- Given n fermionic modes with creation operators $c_1^\dagger, \ldots, c_n^\dagger$, the Majorana operators $\gamma_1, \ldots, \gamma_{2n}$ are

$$\gamma_{2j-1} = c_j + c_j^\dagger, \qquad \gamma_{2j} = -i(c_j - c_j^\dagger)$$

for $j = 1, \ldots, n$ and satisfy the anticommutation relation

$$\{\gamma_p, \gamma_q\} = 2\delta_{pq} I.$$

Jordan-Wigner Transformation

- Spin Hamiltonians (i.e. a collection of n qubits) are the same as fermionic Hamiltonians
- We use

$$\gamma_{2i-1} = \prod_{j=1}^{i-1}(-Z_j)X_i, \qquad \gamma_{2i} = -\prod_{j=1}^{i-1}(-Z_j)Y_i$$

- Products of two Majoranas look like

$$I \otimes \cdots \otimes I \otimes U_1 \otimes Z \otimes \cdots \otimes Z \otimes U_2 \otimes I \otimes \cdots \otimes I$$

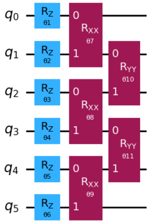
where $U_1, U_2 \in \{X, Y\}$

Defining Matchgates

- Matchgates are a class of parametrized quantum gates, generated by

$$e^{-iX\otimes X\theta/2},\ e^{-iY\otimes Y\theta/2},\ e^{-iX\otimes Y\theta/2},\ e^{-iY\otimes X\theta/2},\ e^{-iIZ\theta/2},\ e^{-iZI\theta/2}$$

- All of the exponents are products of Majoranas.
- Matchgates are differentiable with respect to θ and can be simulated in polynomial time



Computing Gradients

- To get to the minimum of a loss function, we need to compute its gradient.
- Parameter-Shift Rule: If $L(\boldsymbol{\theta}) = \langle 0| U_\theta^\dagger A U_\theta |0\rangle$ where $U_\theta = e^{-ia\theta G}$ and $G^2 = I$, then

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = r\left(L\left(\boldsymbol{\theta} + \frac{\pi}{4r}\vec{\mathbf{e}}_i\right) + L\left(\boldsymbol{\theta} - \frac{\pi}{4r}\vec{\mathbf{e}}_i\right)\right)$$

- To find the gradient of $L_G(\boldsymbol{\theta}, \boldsymbol{\phi})$, just need to compute the value of L_G at various points.

Parameter Shift in Practice

- Denote $\boldsymbol{\theta}^+ = \boldsymbol{\theta} + \frac{\pi}{4r}\vec{\mathbf{e}}_i$ and $\boldsymbol{\theta}^- = \boldsymbol{\theta} + \frac{\pi}{4r}\vec{\mathbf{e}}_i$. We have

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = r\left(L\left(\boldsymbol{\theta} + \frac{\pi}{4r}\vec{\mathbf{e}}_i\right) + L\left(\boldsymbol{\theta} - \frac{\pi}{4r}\vec{\mathbf{e}}_i\right)\right)$$
$$\approx r\left(-\frac{1}{N}\sum_{\ell=1}^{N}\log(D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}^+}(z^{(\ell)}))) - \frac{1}{N}\sum_{\ell=1}^{N}\log(D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}^-}(z^{(\ell)})))\right)$$

- For each parameter θ_i, we need to change the parameters of the quantum circuit, then sample $2N$ times

Gradient of L_D

$$\nabla_{\phi} L_D(\boldsymbol{\theta}, \boldsymbol{\phi}) = \nabla_{\boldsymbol{\phi}} \left(- \left(\mathop{\mathbb{E}}_{\boldsymbol{y} \sim p_{\text{real}}} [\log(D_{\boldsymbol{\phi}}(\boldsymbol{y}))] + \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\boldsymbol{\theta}}} [\log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x}))] \right) \right)$$

$$= - \left(\mathop{\mathbb{E}}_{\boldsymbol{y} \sim p_{\text{real}}} [\nabla_{\boldsymbol{\phi}} \log(D_{\boldsymbol{\phi}}(\boldsymbol{y}))] + \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\boldsymbol{\theta}}} [\nabla_{\boldsymbol{\phi}} \log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x}))] \right)$$

- Mathematically, we can interchange the gradient and the summation/integral
- Most ML packages (e.g. Flux.jl) can do autodifferentiation

Gradient of L_G

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} L_G(\boldsymbol{\theta}, \boldsymbol{\phi}) &= \nabla_{\boldsymbol{\theta}} \left(- \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\boldsymbol{\theta}}} \left[\log(D_{\boldsymbol{\phi}}(\boldsymbol{x})) \right] \right) \\
&= -\nabla_{\boldsymbol{\theta}} \sum_{x \in \{0,1\}^n} \log(D_{\boldsymbol{\phi}}(x)) p_{\boldsymbol{\theta}}(x) \\
&= - \sum_{x \in \{0,1\}^n} \log(D_{\boldsymbol{\phi}}(x)) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(x)
\end{aligned}
$$

- It's not as easy to differentiate $p_{\boldsymbol{\theta}}(x)$

REINFORCE Trick

- Rewrite $\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(x) = p_{\boldsymbol{\theta}}(x) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(x)$

$$
\begin{aligned}
-\sum_{x \in \{0,1\}^n} \log(D_{\boldsymbol{\phi}}(x)) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(x) &= -\sum_{x \in \{0,1\}^n} \log(D_{\boldsymbol{\phi}}(x)) p_{\boldsymbol{\theta}}(x) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(x) \\
&= -\mathop{\mathbb{E}}_{x \sim p_{\boldsymbol{\theta}}} [\log(D_{\boldsymbol{\phi}}(x)) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(x)] \\
&\approx -\frac{1}{N} \sum_{\ell=1}^{N} \log(D_{\boldsymbol{\phi}}(x^{(\ell)})) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(x^{(\ell)})
\end{aligned}
$$

Analytical Log Gradient

- Analytical algorithm to compute $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(x)$ for a given $x = x_1 x_2 \ldots x_n \in \{0, 1\}^n$
- Rather than sampling all bits at one time, we are sampling bit-by-bit
- Mathematically, $\log p_{\boldsymbol{\theta}}(x) = \log p_{\boldsymbol{\theta}}(x_1) + \sum_{j=2}^{n} \log p_{\boldsymbol{\theta}}(x_j | x_1 \ldots x_{j-1})$
- $U_{\boldsymbol{\theta}} |0\rangle$ is a Gaussian state and is thus uniquely defined by a covariance matrix $M(\boldsymbol{\theta})$ with entries $M_{pq}(\boldsymbol{\theta}) = \frac{i}{2} \langle 0| U_{\boldsymbol{\theta}}^{\dagger} |[\gamma_p, \gamma_q]| U_{\boldsymbol{\theta}} |0\rangle$
- $p_{\boldsymbol{\theta}}(x_i) = \frac{1}{2}(1 + (-1)^{x_i} M_{2i-1,2i}(\boldsymbol{\theta}))$

Updating the Covariance Matrix

- Wick's Theorem: Upon measuring the ith qubit and obtaining a measurement outcome x_i, the covariance matrix transforms as

$$M_{pq}^{x_i}(\boldsymbol{\theta}) = M_{pq}(\boldsymbol{\theta}) + \frac{(-1)^{x_i}}{2p_{\boldsymbol{\theta}}(x_i)}(M_{2i-1,q}(\boldsymbol{\theta})M_{2i,p}(\boldsymbol{\theta}) - M_{2i-1,p}(\boldsymbol{\theta})M_{2i,q}(\boldsymbol{\theta}))$$

- Subsequently, $p_{\boldsymbol{\theta}}(x_j|x_i) = \frac{1}{2}(1 + (-1)^{x_j} M_{2j-1,2j}^{x_i}(\boldsymbol{\theta}))$
- All relevant information are stored in the covariance matrix
- We can compute $\nabla_{\boldsymbol{\theta}} M_{pq}(\boldsymbol{\theta})$ efficiently

Computing the Log Gradient

- Only need to keep track of two $2n \times 2n$ matrices

$$\begin{bmatrix} \quad M_{pq}(\boldsymbol{\theta}) \quad \end{bmatrix} \qquad \begin{bmatrix} \quad \nabla_{\boldsymbol{\theta}} M_{pq}(\boldsymbol{\theta}) \quad \end{bmatrix} \longrightarrow \begin{matrix} p_{\boldsymbol{\theta}}(x_1) \\ \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(x_1) \end{matrix}$$

$$\begin{bmatrix} \quad M_{pq}^{x_1}(\boldsymbol{\theta}) \quad \end{bmatrix} \qquad \begin{bmatrix} \quad \nabla_{\boldsymbol{\theta}} M_{pq}^{x_1}(\boldsymbol{\theta}) \quad \end{bmatrix} \longrightarrow \begin{matrix} p_{\boldsymbol{\theta}}(x_2|x_1) \\ \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(x_2|x_1) \end{matrix}$$

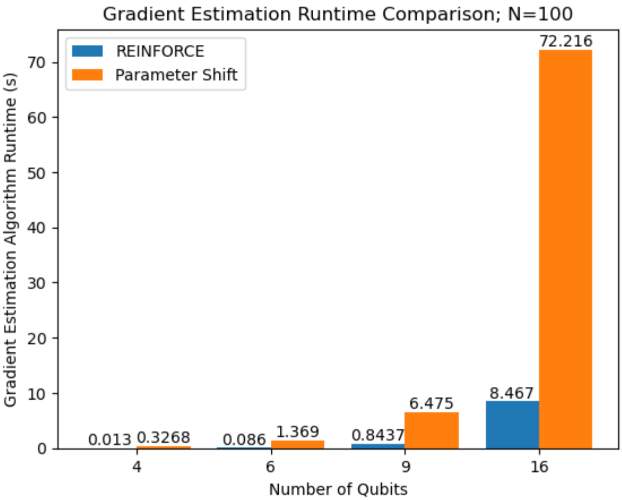
$$\vdots \qquad\qquad \vdots$$

$$\begin{bmatrix} \quad M_{pq}^{x_1,x_2,\ldots,x_{n-1}}(\boldsymbol{\theta}) \quad \end{bmatrix} \qquad \begin{bmatrix} \quad \nabla_{\boldsymbol{\theta}} M_{pq}^{x_1,x_2,\ldots,x_{n-1}}(\boldsymbol{\theta}) \quad \end{bmatrix} \longrightarrow \begin{matrix} p_{\boldsymbol{\theta}}(x_n|x_1,\ldots,x_{n-1}) \\ \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(x_n|x_1,\ldots,x_{n-1}) \end{matrix}$$

Code

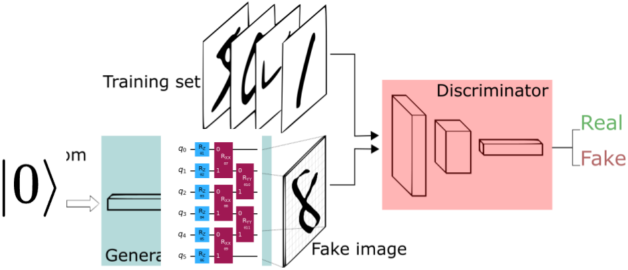
- Written in Julia, using Yao and FLOYao packages
- Utilized FLOYao package to harness Matchgate optimizations

Runtime Comparison



Setup

1. Use a parametrized Matchgate circuit for G
2. Use a classical neural network for D
3. Use the GAN loss functions and train



Acknowledgements

- Hong-Ye Hu and Susanne Yelin
- Yelin group undergrads: Jorge, Fiona, Victoria, and Kevin

Questions?

- Thanks for listening!

Pfaffians

- Define the Pfaffian sum to be

$$\mathrm{PfS}(B) = \sum_{A\subseteq[n]} \left(\prod_{i\in A} x_i\right) \mathrm{Pf}(B[A])$$

- Pfaffian sums are just Pfaffians:

$$\mathrm{PfS}(B) = \begin{cases} \mathrm{Pf}(B+\Lambda_n) & n \text{ even} \\ \mathrm{Pf}(B^+ + \Lambda_{n+1}) & n \text{ odd} \end{cases}$$

where

$$\Lambda_n = \begin{pmatrix} 0 & x_1x_2 & -x_1x_3 & \cdots \\ -x_1x_2 & 0 & x_2x_3 & \cdots \\ x_1x_3 & -x_2x_3 & \ddots & \\ \vdots & \vdots & & \end{pmatrix}$$

Matchgate Definition

- A matchgate Γ is (G, X, Y, T) where G is a graph and X, Y, T partition the vertices. For $Z \subseteq X \cup Y$, define

$$\chi(\Gamma, Z) = \mu(\Gamma, Z)\mathrm{PfS}(G - Z)$$

where $\mu(\Gamma, Z) \in \{-1, 1\}$ and $x_i = 1$ if $i \in T,\ x_i = 0$ otherwise.

- The character matrix $\chi(\Gamma)$ is the $2^{|X|} \times 2^{|Y|}$ matrix with

$$\chi(\Gamma)_{i,j} = \chi(\Gamma, X_i \cup Y_j).$$

- Defined in this way, not all matchgates are unitary
- Take the ones that are unitary and interpret them as quantum gates

REINFORCE Full Derivation

$$
\begin{aligned}
\nabla_\theta L_G(\theta,\phi) &= -\nabla_\theta \mathop{\mathbb{E}}_{x\sim p_\theta}\left[\log(D_\phi(x))\right] \\
&= -\nabla_\theta \sum_{x\in\{0,1\}^n} \log(D_\phi(x))p_\theta(x) \\
&= -\sum_{x\in\{0,1\}^n} \log(D_\phi(x))\nabla_\theta p_\theta(x) \\
&= -\sum_{x\in\{0,1\}^n} \log(D_\phi(x))p_\theta(x)\nabla_\theta \log p_\theta(x) \\
&= -\mathop{\mathbb{E}}_{x\sim p_\theta}\left[\log(D_\phi(x))\nabla_\theta \log p_\theta(x)\right] \\
&\approx -\frac{1}{N}\sum_{\ell=1}^{N} \log(D_\phi(x^{(\ell)}))\nabla_\theta \log p_\theta(x^{(\ell)})
\end{aligned}
$$

Probability Derivation

Recall that for fermionic creation and annihilation operators, we have

- The number operator: $n_j = c_j^\dagger c_j$
- Squaring to zero: $c_j c_j = c_j^\dagger c_j^\dagger = 0$
- Anticommutation relation $\{c_j^\dagger, c_k\} = \delta_{jk}$
- Definition of Majorana: $\gamma_{2j-1} = c_j + c_j^\dagger,\ \gamma_{2j} = -i(c_j - c_j^\dagger)$
- Claim: $\langle n_j \rangle = \frac{1}{2}(1 - \frac{i}{2}\langle[\gamma_{2j-1}, \gamma_{2j}]\rangle)$