Variational quantum architectures for linear algebra applications

Carlos Bravo-Prieto

IPAM Quantum Numerical Linear Algebra
27th January, 2022
• Quantum singular value decomposer: to produce singular value decomposition of bipartite pure states

• Variational quantum linear solver: for solving linear systems of equations

• Quantum generative models via adversarial learning: to learn underlying distribution functions.
Noisy intermediate-scale quantum (NISQ) era

NISQ era:
- Low number of qubits (50 qubits to a few hundreds)
- Low coherence times (~1000 operations)
- No error correction

Not yet capable of large-scale quantum computations
Variational quantum architectures

Candidates for near term advantage
- No high requisites in the number of qubits
- Shallow quantum circuits and hardware efficient
- Slightly noise resilience

Encode the problem into some cost function

Use a classical/quantum hybrid computation to minimize this cost function

$$\text{minimize}_\phi \langle 0 | U(\phi)^\dagger H_c U(\phi) | 0 \rangle$$
Quantum singular value decomposer

Quantum Singular Value Decomposer

\[ |\psi\rangle_{AB} = \sum_{i=1}^{X} \lambda_i |u_i\rangle_A |v_i\rangle_B \]

\[ |\psi\rangle_{AB} \xrightarrow{QSV D} U_A(\tilde{\Theta}) \otimes V_B(\tilde{\Omega}) |\psi\rangle_{AB} = \sum_{i=1}^{X} \lambda_i e^{i\alpha_i} |e_i\rangle_A |e_i\rangle_B \]

Variational training to correlations

Only one measurement setting
Once trained:
- Read out entropy spectrum

\[
|\psi\rangle_{AB} = \sum_{i=1}^{X} \lambda_i |u_i\rangle_A |v_i\rangle_B
\]

\[
|\psi\rangle_{AB} \xrightarrow{QSV} U_A(\Theta) \otimes V_B(\Omega) |\psi\rangle_{AB}
\]

\[
= \sum_{i=1}^{X} \lambda_i e^{i\alpha_i} |e_i\rangle_A |e_i\rangle_B
\]
Once trained:
- Read out entropy spectrum
- Recover eigenvectors with inverted unitaries
Once trained:
- Read out entropy spectrum
- Recover eigenvectors with inverted unitaries
- Autoencoder and SWAP
Variational quantum linear solver
Variational Quantum Linear Solver

\[ Ax = b \], where \( A \) is an \( N \times N \) matrix

- Machine learning
- Partial differential equations
- Polynomial curve fitting
- Analyzing electrical circuits
- …

**Classical algorithms:** polynomial scaling in \( N \)

Variational Quantum Linear Solver

\[ A x = b \], where \( A \) is an \( N \times N \) matrix

Quantum algorithm: Harrow-Hassidim-Lloyd (HHL)

- Prepare \( |x> \), such that \( |x> \sim x \)
- Log \( N \) scaling
- Further improvements: reduced complexity in \( \kappa \) and \( \epsilon \)
- Requires deep circuits

Variational quantum linear solver: geared towards NISQ
Variational Quantum Linear Solver

- Define cost function
  \[ C = 0 \quad \text{You solved the linear system!} \]

- Operational meaning of C (e.g. solution guarantees)

- Find a circuit that computes C
  - Efficient quantumly
  - Hard classically

Variational Quantum Linear Solver

\[ A = \sum_{l=1}^{L} c_l A_l \quad \text{and} \quad U \text{ such that } U|0\rangle = |b\rangle \]

Input

Output

\[ |x\rangle = \frac{x}{||x||_2} \]
VQLS: input

- Specify linear problem: \( A|x\rangle \propto |b\rangle \)

- Efficient circuit \( U \): \( U|0\rangle = |b\rangle \)

- \( A \) is given by a linear combination of unitaries

\[
A = \sum_{l=1}^{L} c_l A_l \quad \text{and} \quad U \text{ such that } U|0\rangle = |b\rangle
\]

\[
\|A\| \leq 1 \quad \kappa < \infty
\]
**VQLS: optimization**

- **Goal:** prepare $|x\rangle$ such that $A|x\rangle \propto |b\rangle$

- **Ansatz for $|x\rangle$:** $|x(\alpha)\rangle = V(\alpha)|0\rangle$

---

VQLS: output

- Optimal parameters $\alpha = \alpha_{opt}$
- Prepare $|x(\alpha_{opt})\rangle = V(\alpha_{opt})|0\rangle$

$$V(\alpha_{opt}) =$$

VQLS: Cost functions

- Global cost function

\[ \begin{align*}
C_G &= \langle x | H_G | x \rangle \\
H_G &= A^\dagger (1 - |b\rangle\langle b|) A
\end{align*} \]

- Local cost function

\[ \begin{align*}
C_L &= \langle x | H_L | x \rangle \\
H_L &= A^\dagger U \left( 1 - \frac{1}{n} \sum_{j=1}^{n} |0_j\rangle\langle 0_j| \otimes 1_j \right) U^\dagger A
\end{align*} \]

- \( C_L \leq C_G \leq nC_L \)

\[ C_L = 0 \iff C_G = 0 \iff A|x\rangle \sim |b\rangle \]
Barren plateaus: global vs local

- family of $\{A^{(n)}\}$, $A^{(n)}$ - condition number $\kappa = 20$
Operational meaning

- one can show that

\[ C_G \geq \frac{\epsilon^2}{\kappa^2} \quad \text{and} \quad C_L \geq \frac{1}{n} \frac{\epsilon^2}{\kappa^2} \]

\[ \epsilon = \frac{1}{2} \text{Tr} \left| |x_0\rangle\langle x_0| - |x(\alpha_{opt})\rangle\langle x(\alpha_{opt})| \right| \]

\[ |x_0\rangle - \text{exact solution} \]

\[ C_L = \gamma \]

\[ \epsilon \leq \kappa \sqrt{n\gamma} \]
Example: simulations

- Ising-type

\[ A = \frac{1}{\zeta} \left( \sum_{j=1}^{n} \sigma_j^X + J \sum_{j=1}^{n-1} \sigma_j^Z \sigma_{j+1}^Z + \eta 1 \right) \]

\[ |b\rangle = H^\otimes n |0\rangle \]

- \( \zeta, \eta \) such that \( A \) has condition number \( \kappa \)
Example: scaling

- time-to-solution: number of iterations needed to guarantee precision $\epsilon$
- sub-linear in $\kappa$
- logarithmic in $1/\epsilon$
Example: scaling

- time-to-solution: number of iterations needed to guarantee precision $\epsilon$
- linear in $n$ (logarithmic in $N$)
Example: simulations

- random matrix

\[ A = \frac{1}{\zeta} \left( \sum_j \sum_{k \neq j} p a_{j,k} \sigma_j^\alpha \sigma_k^\beta + \eta 1 \right) \]

\[ |b\rangle = H^{\otimes n} |0\rangle \]

- \( \zeta, \eta \) such that \( A \) has condition number \( \kappa \)

- random:
  - \( p \in \{0, 1\} \)
  - \( a_{j,k} \in (-1, 1) \)
  - \( \alpha, \beta \in \{X, Y, Z\} \)
Example: scaling

- time-to-solution: number of iterations needed to guarantee precision $\epsilon$
- slightly sub-linear in $\kappa$
- logarithmic in $1/\epsilon$
Example: scaling

- time-to-solution: number of iterations needed to guarantee precision $\epsilon$
- polylogarithmic in $N$
Example: Rigetti’s quantum computer

- **Ising-type**

\[
A = \frac{1}{\zeta} \left( \sum_{j=1}^{n} \sigma_j^X + J \sum_{j=1}^{n-1} \sigma_j^Z \sigma_{j+1}^Z + \eta \mathbf{1} \right)
\]

\[
|b\rangle = H^\otimes n |0\rangle
\]

- \(\zeta, \eta\) such that \(A\) has condition number \(\kappa\)
Example: Rigetti’s quantum computer

- largest implementation on real hardware: $n = 10$ qubits, $1024 \times 1024$
- noise resilience: correct parameters $\alpha_{\text{opt}}$ despite cost $C > 0$
Style-based quantum generative adversarial networks for Monte Carlo events
Context: Hadronic collisions at the LHC

LHC produces $O(10^9)$ proton collisions per second: huge complex environment

Simulation of the events are very intensive and requires lots of computing power.
Machine learning approach to event generation

Since 2018, many papers have approached event generation with machine learning

Main idea: train with a small dataset, use machine learning networks to learn the underlying distribution and generate for free a much larger dataset
What is a generative adversarial network (GAN)?

Two networks competing: generator produces fake data, discriminator distinguishes between real (training) input data and fake (produced by the generator) data.

*Adversarial game where the generator learns to map some input noise to the underlying (reference) distribution*

**Art forger analogy**

**Generator (art forger):** Try creating fake paintings that look authentic.

**Discriminator (art historian):** Check paintings and try to catch the forgery.

**Training:** “Catch me if you can” game between the art forger and the art historian.

**Success:** Painted forgeries are so good that the art historian has at most a 50% guess ratio. The forger creates new work.
Training procedure

Training: Adapt alternatively the generator $G(\phi_g, z)$ and the discriminator $D(\phi_d, x)$

Mathematical tool: binary cross-entropy for the loss functions

- Generator loss function:
  \[ \mathcal{L}_G(\phi_g, \phi_d) = -\mathbb{E}_{z \sim p_{prior}(z)}[\log D(\phi_d, G(\phi_g, z))] \]

- Discriminator loss function:
  \[ \mathcal{L}_D(\phi_g, \phi_d) = \mathbb{E}_{x \sim p_{real}(x)}[\log D(\phi_d, x)] + \mathbb{E}_{z \sim p_{prior}(z)}[\log(1 - D(\phi_d, G(\phi_g, z)))] \]

Game theory: min-max two-player game to reach Nash equilibrium
Hybrid approach for a qGAN

Classical setup:

`Only the generator becomes quantum`

Hybrid quantum-classical setup:
Style-based quantum generator

Quantum generator: a series of quantum layers with rotation gates and entanglement operators

\[
U_{\text{ent}} \text{ set of controlled rotations for entanglement}
\]

Style-based approach

Novelty of our network:
the noise is inserted in every gate and not only in the initial quantum state

Validation: 1D Gamma distribution

Assessing the validity of the approach: train and test on known distribution
With one qubit, one layer, using 100 bins: 1D Gamma function
\[ p(x, \alpha, \beta) = x^{\alpha-1} \frac{e^{-x/\beta}}{\beta^\alpha \Gamma(\alpha)}, \alpha = \beta = 1 \]

- Pre-processing of the data to fit samples in [-1,1]
- Train on $10^4$ samples until convergence is reached, perform hyperparameter optimization
- Use generator to generate $10^4$ and $10^5$ samples to demonstrate reproducibility and data augmentation
Simulation with actual LHC data

Testing the styled qGAN with real data: test-case with leading-order production $pp \rightarrow t\bar{t}$

Training and reference samples generated with MadGraph5_aMC@NLO [Alwall et al., JHEP 07 (2014) 079]

LHC at 13 TeV set-up, training set of 10^4 samples, Mandelstam variables $(s, t)$ and rapidity $y$
Simulation with actual LHC data

After training, we assess the performance with simulations: 3 qubits, 2 layers, 100 bins
Simulation with actual LHC data

After training, we assess the performance with simulations: 3 qubits, 2 layers, 100 bins

Correlations are well captured!
Simulation with actual LHC data

After training, we assess the performance with simulations: 3 qubits, 2 layers, 100 bins

Remarkable low KL divergences with data augmentation! Are these results maintained on real hardware?
Results on IBM Q Hardware

Access to IBM quantum hardware via IBM Q cloud service
Technology of superconducting qubit

- Run on ibmq_santiago 5-qubit machine

Still good results with relatively low KL divergence!
Results on IBM Q Hardware

Access to IBM quantum hardware via IBM Q cloud service
Technology of superconducting qubit

- Run on `ibmq_santiago` 5-qubit machine

Still good results with relatively low KL divergence!
Results on IBM Q Hardware

Access to IBM quantum hardware via IBM Q cloud service
Technology of superconducting qubit

• Run on ibmq_santiago 5-qubit machine

Still good results with relatively low KL divergence!
Testing different architectures

Superconducting transmon qubits: *ibmq_santiago* with 2-neighbouring site connectivity

Access via IBM Q cloud service

Trapped ion technology: ionQ with all-to-all connectivity

Access via Amazon Web Services
Testing different architectures: results

- Access constraints to ionQ: test limited to 1k samples only

**Very similar results:**

implementation largely hardware-independent

**ionQ samples:**

**IBM Q samples:**
Thanks for your attention

Any questions?