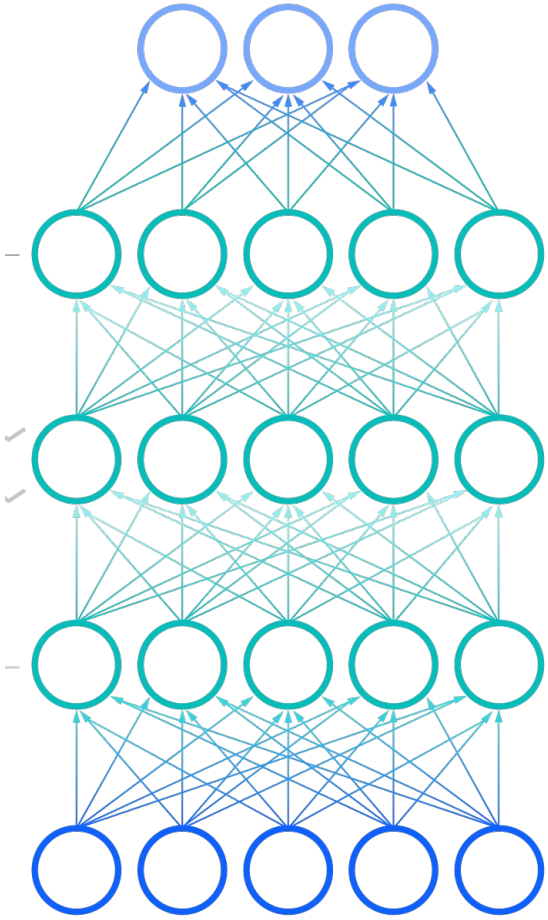


Lecture 12:
Quantum Deep Learning

COMP3366
Quantum algorithms & computing architecture

Instructor: Yuxiang Yang

Department of Computer Science, HKU

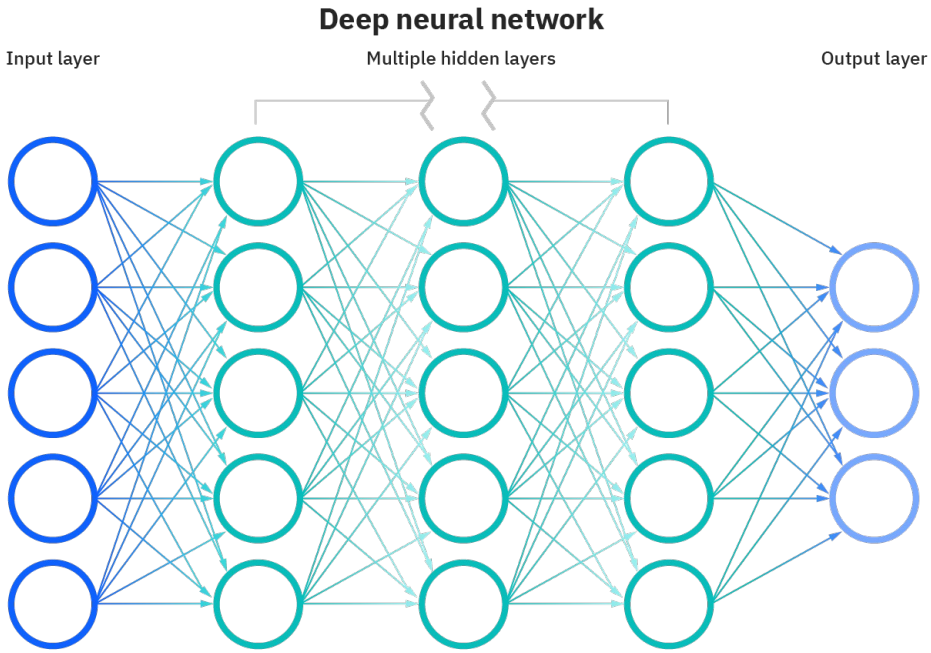
Objectives:

- **[O1] Concepts:** (Quantum) deep learning, quantum neural networks, support vector machine, kernel, (quantum) feature maps, (quantum) autoencoder, encoder and decoder, barren plateau.
- **[O2] Problem solving:** Working principle of quantum data classification and quantum autoencoder.
- **[O3] Algorithm design:** Implement a quantum data classifier with Qiskit.

Section I:
Quantum deep learning

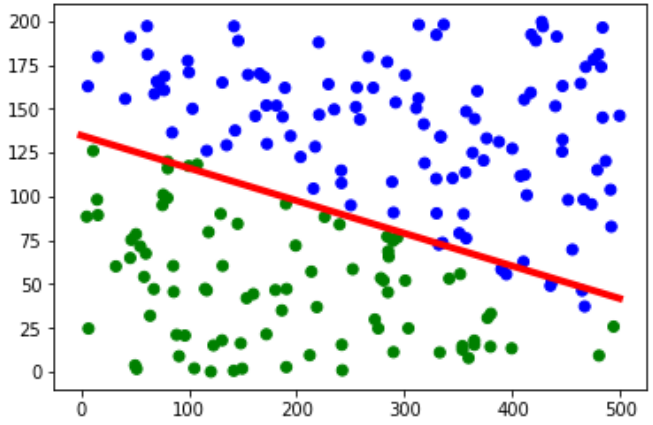
Deep learning

- Deep learning is the branch of machine learning involving the study of deep neural networks.



Deep learning workflow

1. Build a neural network model with variational parameters:
   e.g., $f\left(x, \vec{\theta}\right) = \sigma(Wx + b) \quad \vec{\theta} = \{W, b\}$
2. Define a cost function, e.g., $C\left(\vec{\theta}\right) = \sum_{(x,y)\in \text{data}} \left|f\left(x, \vec{\theta}\right) - y\right|^2$
3. Gradient descent to update the parameters: $\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \eta \nabla_{\vec{\theta}} C$ until convergence.
4. Test the trained model for classification, prediction, etc.

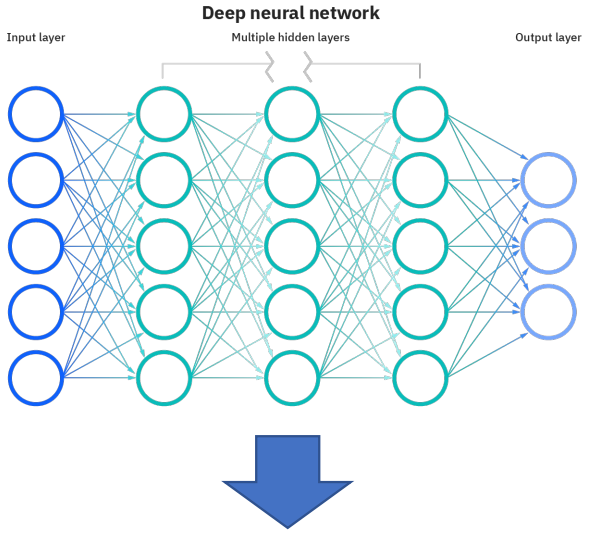


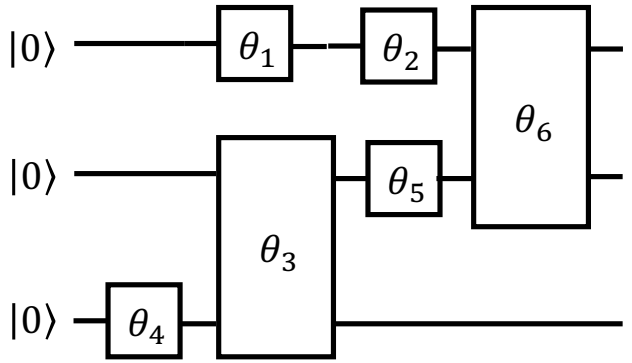
Quantizing deep learning

- Quantum deep learning replaces the classical deep neural network by a variational circuit, which is also referred to as a quantum neural network.

- The cost function should be measurable from the circuit with the help of the Born rule:
  e.g., $C(\theta) = \sum_i c_i \langle \psi(\vec{\theta}) | H_i | \psi(\vec{\theta}) \rangle$.

- Quantum neural networks are classical-quantum hybrid algorithms, where the optimizing part (gradient descent etc.) is taken care of by a classical computer.

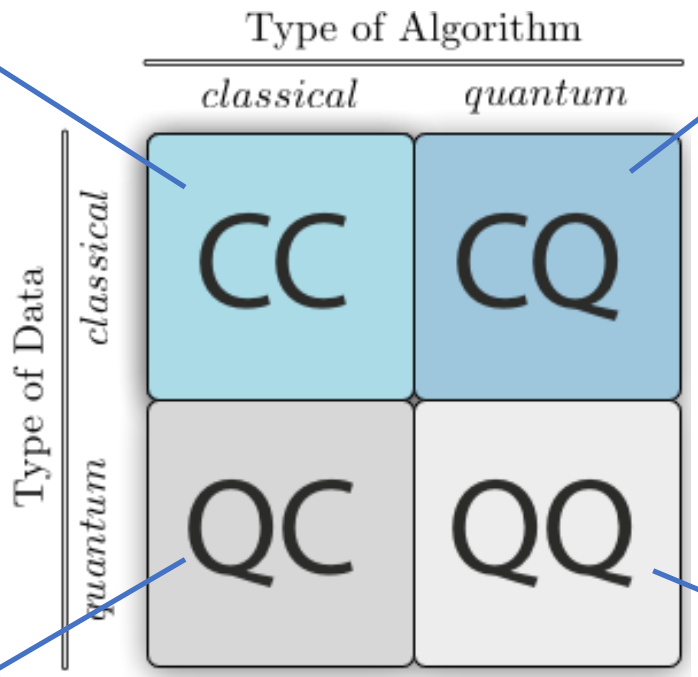




Scope of QML

Classical ML

Example: quantum classifier/support vector machine
(this lecture; Lecture 5)



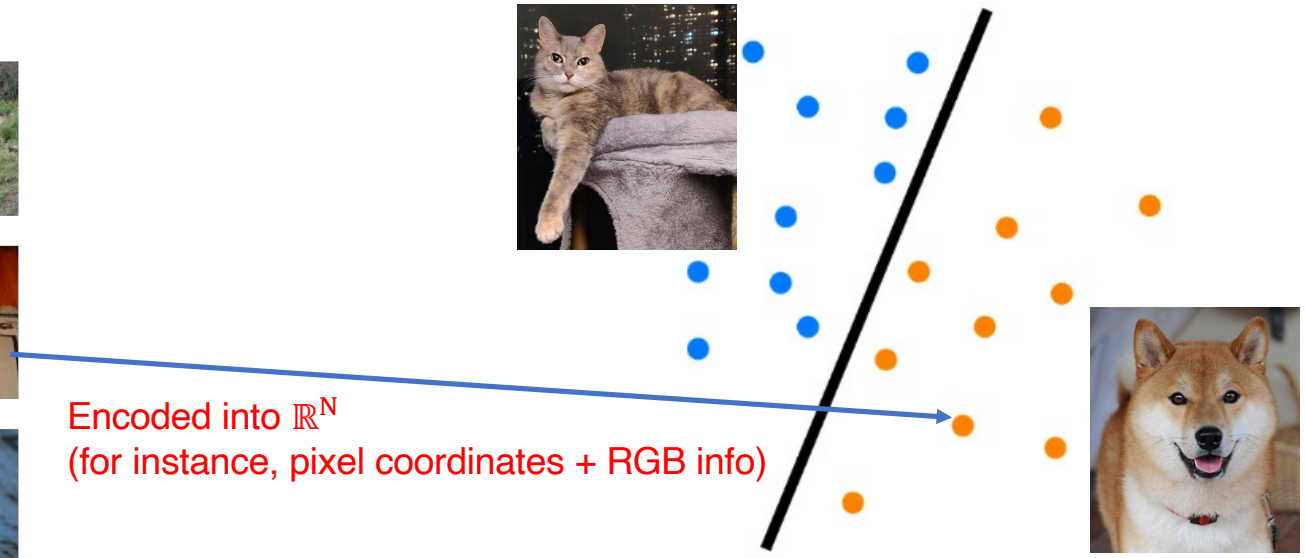
Simulated quantum computing
(Lecture 11);
Learning quantum states & gates

Example: quantum autoencoder
(this lecture)

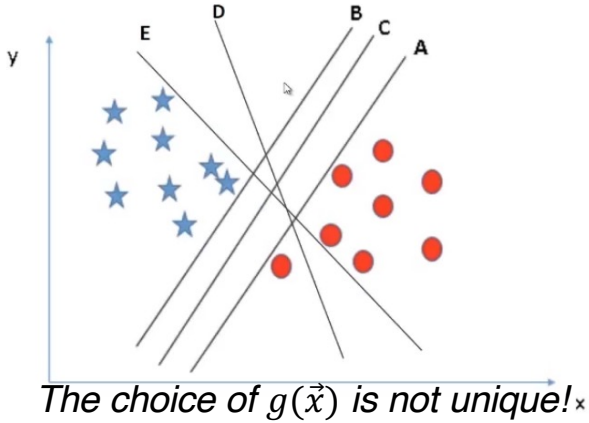
Part II:
Data classification

Classification of data

- Classification:
  Given M training data of dimension N as well as their labels, the goal is to construct a classifier f
  = separate different types of data with separating hyperplane(s).



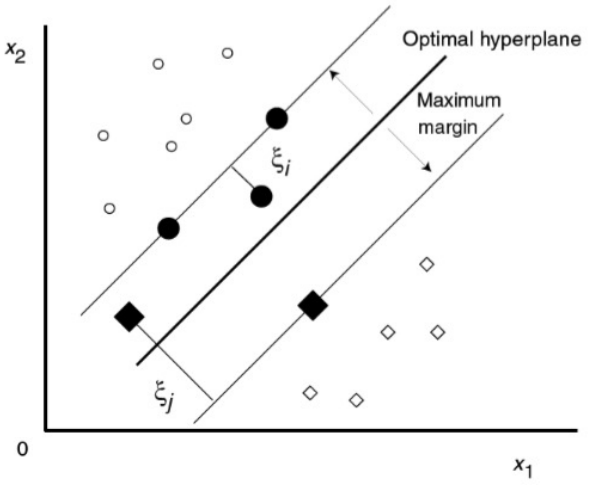
Support vector machines (SVMs)



- Goal:
  Create classifier $g(\vec{x}) = \vec{w} \cdot \vec{x} + b$ so that
  $$g(\vec{x}) \cdot y > 0$$
  for any testing datum $\vec{x}$.

Equivalent to $g(\vec{x}) > 0 \Rightarrow y = 1$ and $g(\vec{x}) < 0 \Rightarrow y = -1$!

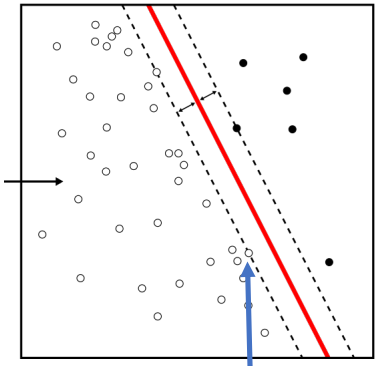
- The choice of $g(\vec{x})$ is not unique.
  How should we choose the classifier so that it behaves well for future data?
- Solution: maximize the margin!
  Margin = *minimal distance between any datum and the separating hyperplane.*
- Equivalent to minimizing $|\vec{w}|$, subject to the constraint: $g(\vec{x}) \cdot y \geq 1$ for any testing datum $\vec{x}$.



Support vector machines (SVMs)

- (Hard-margin) SVM optimization problem:

$$\min \frac{1}{2}\vec{w}\cdot\vec{w}$$
$$s.t. \quad y_k(\vec{w}\cdot\overrightarrow{x_k}+b)\geq 1, k=1,\dots,M$$



- Introducing the multipliers $\{\alpha_k\}$ and

$$Q=\frac{1}{2}\sum_{i=1}^{N} w_i^2-\sum_{k=1}^{M}\alpha_k((\vec{w}\cdot\overrightarrow{x_k}+b)-y_k)$$

- KKT condition:

$$\vec{w}-\sum_{k=1}^{M}\alpha_k\overrightarrow{x_k}=0 \qquad \sum_{k}^{M}\alpha_k=0$$

$$\vec{w}\cdot\overrightarrow{x_k}+b=y_k \text{ or } \alpha_k=0$$

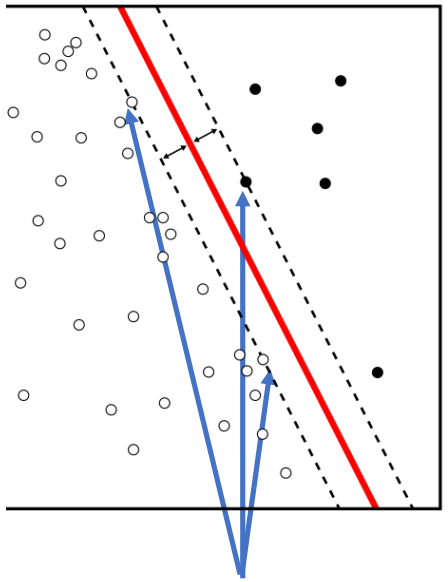
$(\overrightarrow{x_k}, y_k)$ in the support vector set S!

Support vectors

- In SVMs, $\vec{w}$ is spanned by the set S of support vectors, since $\vec{w} - \sum_{k=1}^{M} \alpha_k \overrightarrow{x_k} = 0$ and $\alpha_k = 0$ when $(\overrightarrow{x_k}, y_k) \notin S$:

$$\vec{w} = \sum_{(\overrightarrow{x_k}, y_k) \in S} \alpha_k \overrightarrow{x_k}.$$

- SVM as a compression method:
  Once we found S. It is enough to store the data in S, rather than all data in the training set, to compute the classifier $g(\vec{x})$.



The support vector set S is much smaller than the data set!

Kernel

- The classifier can be expressed as $g(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x} + b)$ with $\vec{w} = \sum_{(\overrightarrow{x_k}, y_k) \in S} \alpha_k \overrightarrow{x_k}$.
- We can rewrite it as

Inner product of the two vectors

$$g(x) = \text{sgn}\left(\sum_k \alpha_k (\overrightarrow{x_k} \cdot \vec{x}) + b\right)$$

- … or:

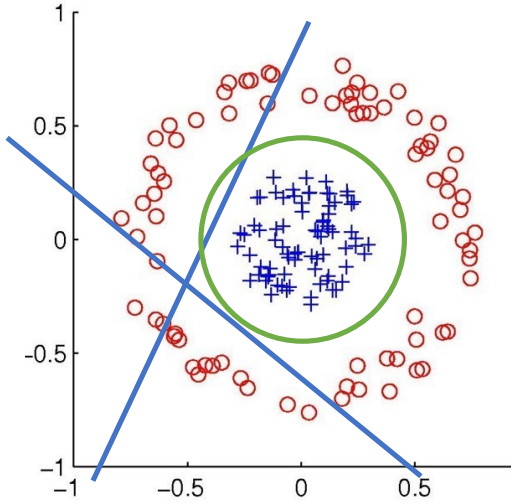
$$g(x) = \text{sgn}\left(\sum_k \alpha_k K(\overrightarrow{x_k}, \vec{x}) + b\right)$$

where $K(\vec{x}', \vec{x})$ is called the kernel (function).

- The kernel can be adjusted to make the classifier more powerful!

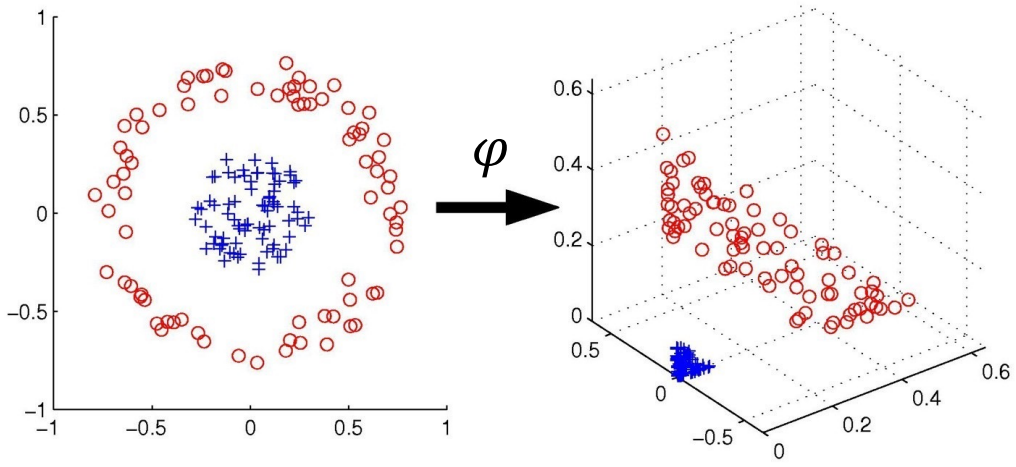
Non-linear data separation

- Suppose we have the following data. How can we construct a classifier?
- Linear classifier would not work, and we need non-linear data separation!
- Attempt: We define the classifier $g(x) = \text{sgn}[r^2 - x_1^2 - x_2^2]$ and r can be learned from the data.
- Problem: This is over-fitting! We should not choose the model based on the observed data.



Feature map

- We first enlarge the data space by introducing a 3rd dimension.
  The original data is mapped into the larger space via a feature map
  $\varphi(x_1, x_2) \in \mathbb{R}^3$.

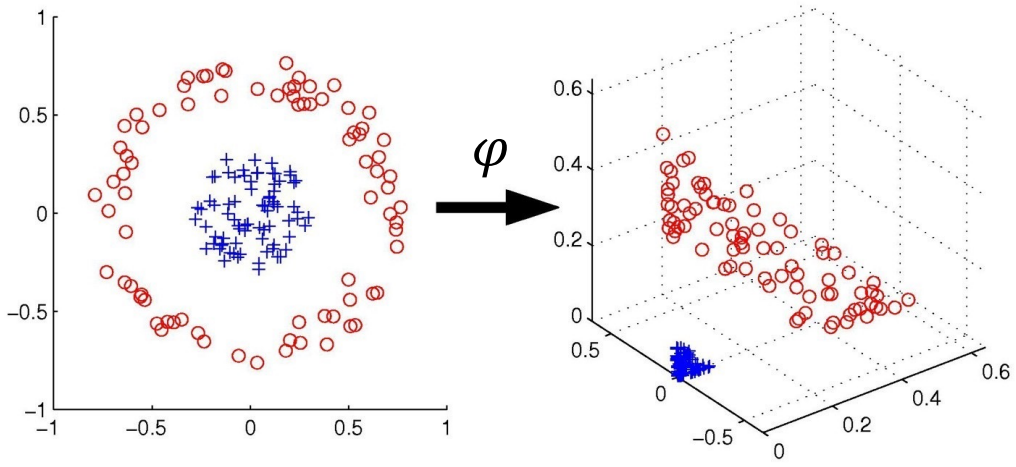


How to choose the feature map?

- For example, we can choose the feature map:

$$\varphi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2).$$

- In the enlarged space, the data can be easily separated with a linear classifier (i.e., a 2D plane).



Kernel trick

- With a feature map φ the classifier can be expressed as

$$g(x) = \operatorname{sgn}\left(\sum_k \alpha_k K(\overrightarrow{x_k}, \vec{x}) + b\right)$$

Inner product of the two vectors in the new feature space

- Kernel matrix:

$$K(\vec{x}', \vec{x}) \coloneqq \varphi(\overrightarrow{x_k}) \cdot \varphi(\vec{x})$$

- Difference choices of the feature map could yield different kernels.
- We could also directly choose a kernel to be a Hermitian matrix with non-negative eigenvalues that is not of inner product form.

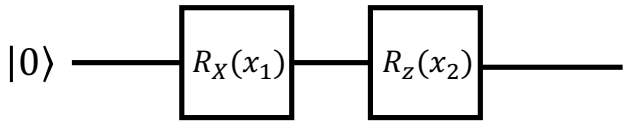
Kernel trick

- Common choices of the kernel:
  - Linear (trivial) kernel: $K(x, x') = x \cdot x$.
  - Polynomial kernel: $K(x, x') = (1 + x \cdot x')^p$.
  - Gaussian kernel: $K(x, x') = \exp(-\frac{\left||x-x'|\right|^2}{2\sigma^2})$.
  - Sigmoid kernel: $K(x, x') = \tanh(\beta_0 x \cdot x' + \beta_1)$.
- Most of these kernels contain "variational" parameters (e.g., σ in the Gaussian kernel).
- We can try to use a quantum computer to construct "quantum" kernels!

Part III:
Quantum feature maps and classifiers

Quantum feature map

- Question:
  Suppose we have a two-dimensional datum $x = (x_1, x_2)^T$.
  How to encode it into a variational circuit
- There are (in fact, infinitely) many possibilities:
- Option 1 (QRAM): $|0\rangle \to |x\rangle = \frac{1}{\sqrt{x_1^2+x_2^2}}(x_1|0\rangle + x_2|1\rangle)$.
- Option 2: Use following variational circuit:

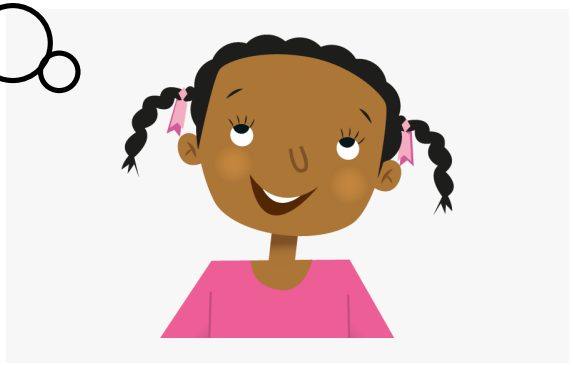


- The procedure of loading classical data x onto a quantum computer defines naturally a quantum feature map:

  $$x \to U(x)$$

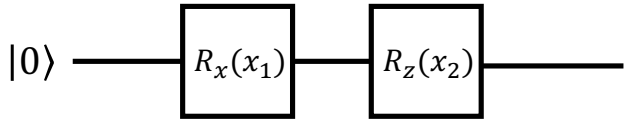
  and the encoded state is $|x\rangle = U(x)|0\rangle^{\otimes n}$.

Can you propose some other quantum feature maps?
What are their pros & cons?

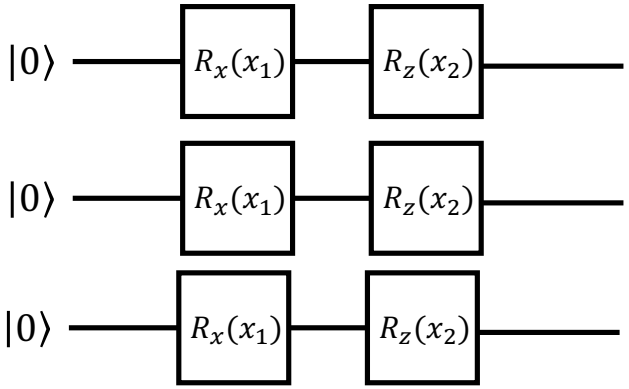


How to choose the quantum feature map

- Consider Option 2:



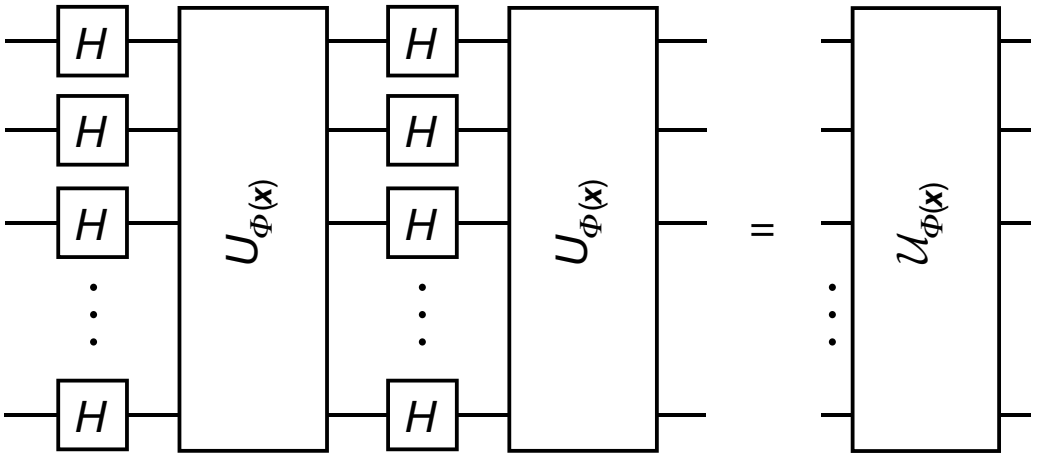
- Question: Can we use n copies of Option 2 when our quantum computer has n qubits?



- Yes. But this will be a waste of the quantum computer's power, because the feature map can be easily simulated by a classical computer!

Quantum feature map

- A common choice of $U(x)$ is *(Havlicek et al. Nature'19):*



$$U_{\Phi(\boldsymbol{x})} = \exp\left(i \sum_{S \subseteq [n]} \phi_S(\boldsymbol{x}) \prod_{i \in S} Z_i\right)$$

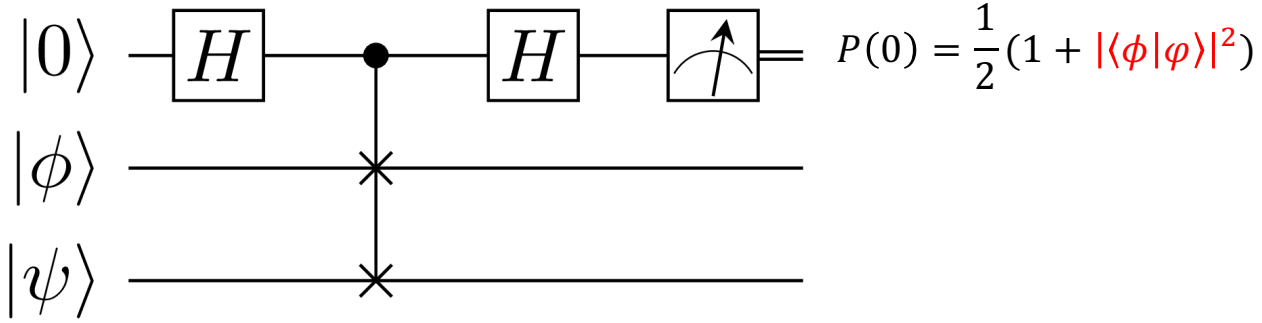
- We can choose nonzero $\phi_S(x) = x$ for $S = \{i, i+1\}$ for $i = 1, \ldots, n-1$.
- Non-linearity: the feature map is nice because it is non-linear in x.

Computing the kernel

- A natural choice of the kernel is $K(x_1, x_2) = |\langle x_1 | x_2 \rangle|^2$
- Question: How do we compute $K(x_1, x_2)$
- When the state is single-qubit, recall that $|x\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right) e^{i\varphi}|1\rangle$ for some θ, φ.
- A straightforward idea is to separately measure both $|x_1\rangle$ and $|x_2\rangle$ in the Z basis and the X basis.
  This yields $\theta_1, \varphi_1, \theta_2, \varphi_2$
- Then $K(x_1, x_2) = \left|\cos\left(\frac{\theta_1}{2}\right)\cos\left(\frac{\theta_2}{2}\right) + \sin\left(\frac{\theta_1}{2}\right)\sin\left(\frac{\theta_2}{2}\right) e^{i(\varphi_1 - \varphi_2)}\right|^2$ can be computed classically.
- Is there a quantum way?

Computing kernel via the SWAP test

- (SWAP test) the following circuit outputs an estimate of $|\langle\phi|\varphi\rangle|^2$ for any $|\varphi\rangle, |\phi\rangle$:



- Using the swap test, we can measure the kernel function much more efficiently!
- The swap test, how ever has $O(n)$ complexity when the inputs are n – qubit.
- Question: Is there an even more efficient method?

Computing kernel by reversal

- Observe that:
  1. $|x_i\rangle = U(x_i)|0\rangle^{\otimes n}$ with $U(x_i)$ being the variational circuit.
  2. The inner product $\langle x_i|x_j\rangle = \langle 0|^{\otimes n} U(x_i)^{\dagger} U(x_j)|0\rangle^{\otimes n}$.
- We can measure $K(x_i, x_j)$ directly in the following way:
  1. Prepare $|x_j\rangle$ with the trained variational circuit.
  2. Perform the inverse of the preparation of $|x_i\rangle$, i.e., $U(x_i)^{\dagger}$.
  3. Measure in the computational basis.
     The kernel is given by the probability of obtaining $00 \ldots 0$.
- Benefit: No swap required.

See this paper for more details

Supervised learning with quantum-enhanced feature spaces

Vojtěch Havlíček[1,2], Antonio D. Córcoles[1]*, Kristan Temme[1]*, Aram W. Harrow[3], Abhinav Kandala[1], Jerry M. Chow[1] & Jay M. Gambetta[1]

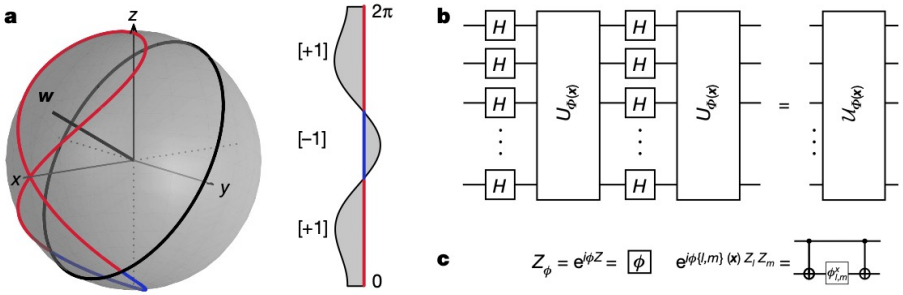


Fig. 1 | Quantum kernel functions. a, Feature map representation for a single qubit. A classical dataset in the interval $\Omega = (0, 2\pi]$ with binary labels (**a**, right) can be mapped onto the Bloch sphere (red and blue lines) by using the non-linear feature map described in **b**. For a single qubit $U_{\Phi(x)} = Z_x$ is a phase-gate of angle $x \in \Omega$. The mapped data can be separated by the hyperplane given by normal $\boldsymbol{w}$. States with a positive expectation value of $\boldsymbol{w}$ receive a [+1] (red) label, while negative values are labelled [−1] (blue). **b**, For the general circuit $U_{\Phi(x)}$ is formed by products of single- and two-qubit unitaries that are diagonal in the computational basis. In our experiments, both the training and testing data are artificially generated to be perfectly classifiable using the feature map. The circuit family depends non-linearly on the data through the coefficients $\phi_S(\boldsymbol{x})$ with $|S| \leq 2$. **c**, Experimental implementation of the parameterized diagonal single- and two-qubit operations using CNOTs and Z-gates.

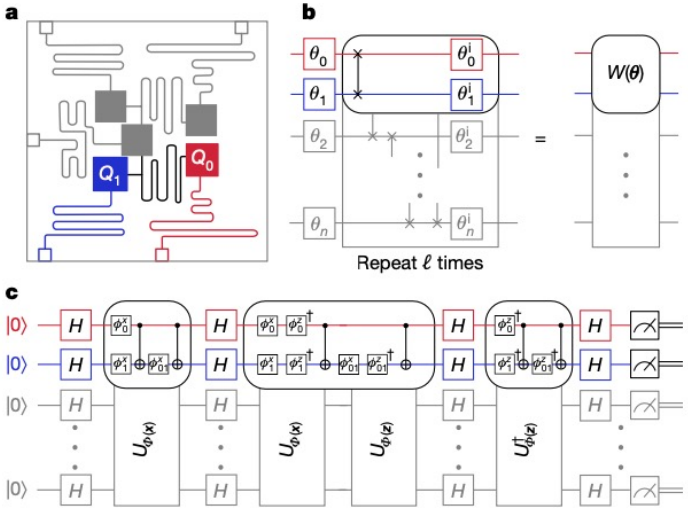


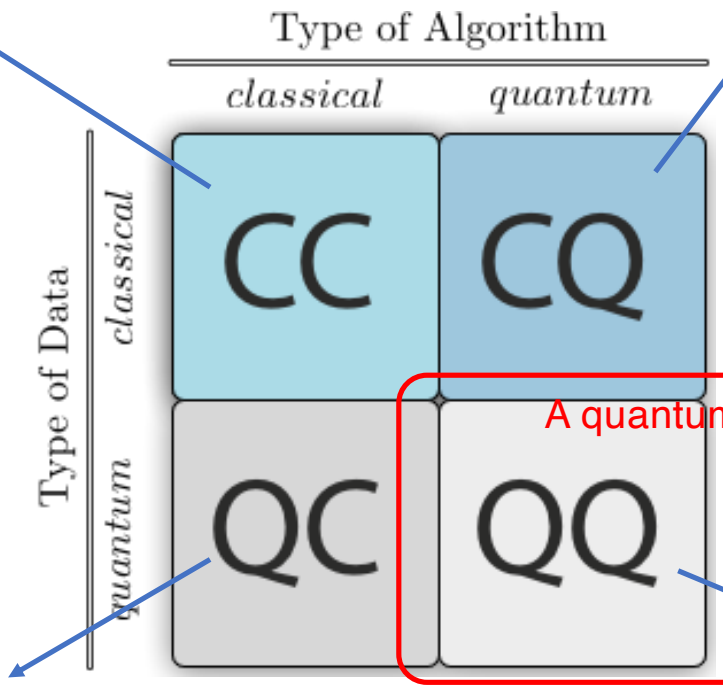
Fig. 2 | Experimental implementations. a, Schematic of the five-qubit quantum processor. The experiment was performed on qubits Q_0 and Q_1, highlighted in the image. **b**, Variational circuit used for our optimization method. The two top qubits depict the circuit implemented. We choose a common ansatz for the variational unitary $W(\boldsymbol{\theta}) = U_{\text{loc}}^{(l)}(\theta_l)\ U_{\text{ent}} \ldots U_{\text{loc}}^{(2)}(\theta_2)\ U_{\text{ent}}\ U_{\text{loc}}^{(1)}(\theta_1)$[16,17]. We alternate layers of entangling gates $U_{\text{ent}} = \prod_{(i,j)\in E} \text{CZ}(i,j)$ with full layers of single-qubit rotations $U_{\text{loc}}^{(t)}(\theta_t) = \otimes_{i=1}^{n} U(\theta_{i,t})$ with $U(\theta_{i,t}) \in \text{SU}(2)$. For the entangling step we use controlled-Z phase gates $\text{CZ}(i,j)$ along the edge (0, 1) of the interaction graph E of the superconducting chip. The grey background illustrates the scaling to a larger number of qubits. **c**, Circuit to directly estimate the fidelity between a pair of feature vectors for data $\boldsymbol{x}$ and $\boldsymbol{z}$ as used for our second method. The circuit on Q_0, Q_1 depicts the circuit implemented, while the grey background illustrates the general structure of the circuit. The circuit is comprised of Hadamard gates H interleaved with the diagonal unitary $U_{\phi(x)}$ parameterized by $\phi_S(\boldsymbol{x})$ to directly estimate the fidelity between a pair of feature vectors for data $\boldsymbol{x}$ and $\boldsymbol{z}$ as used for our second method.

Part IV:
Quantum Autoencoder

Scope of QML

Classical ML

Example: quantum support vector machine
(done just now)



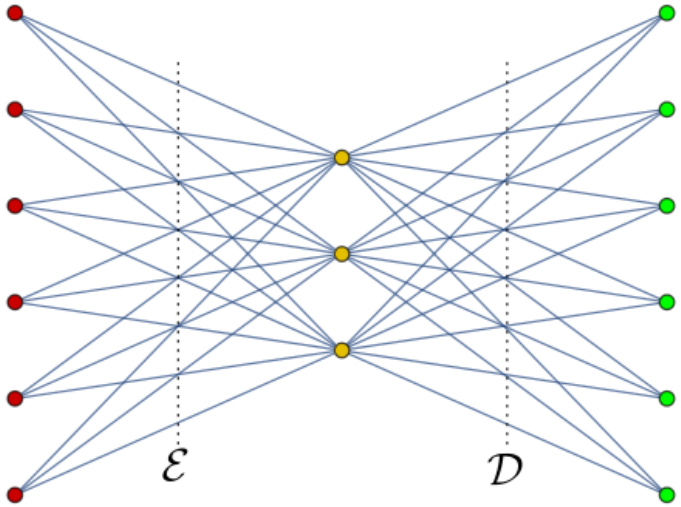
A quantum algorithm that works on quantum data.

Simulated quantum computing
(done in Lecture 11);
Learning quantum states & gates

Example: quantum autoencoder

Autoencoder in the classical world

- An autoencoder is a type of neural network used to learn efficient codings of unlabeled data.
- The cost function is the difference between the input data and the output data.
- Automatic compression is achieved by reducing the size of the bottleneck layer, while keeping the cost function low.
- The 1st half of the final network is an encoder $\mathcal{E}$, and the 2nd half is the corresponding decoder $\mathcal{D}$.



Compression of quantum data

- Consider the output of VQE when applied to an n-qubit Hamiltonian: $H = \sum_i c_i H_i$, for different values of c_i the solutions are different ground states $\{|\psi_1\rangle, |\psi_2\rangle, \ldots, |\psi_m\rangle\}$ of n qubits.

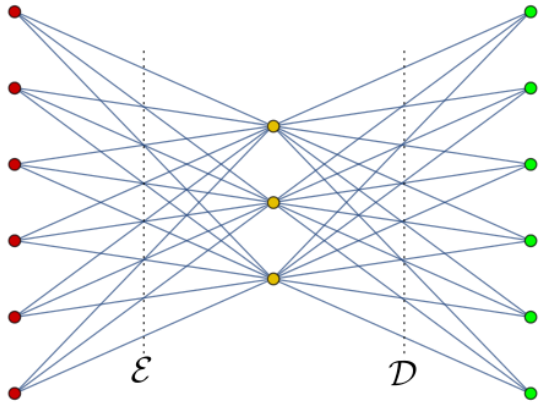
- For practical problems, n might be large, and it is costly to store them in quantum memory.

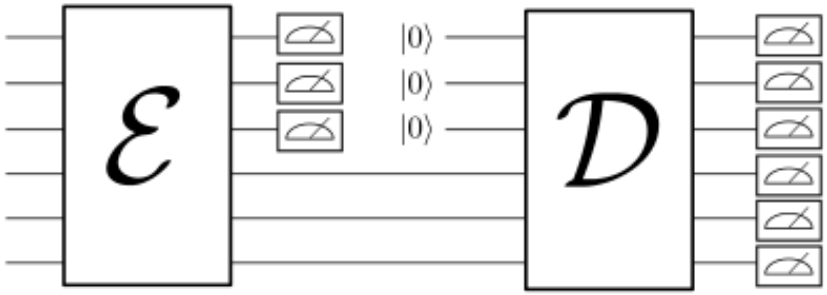
- Can we compress them before storage?
- Example: In Lecture 11, VQE for H_2 molecule yields a 4-qubit state $|\Psi(\theta)\rangle = \cos\frac{\theta}{2}|1100\rangle - \sin\frac{\theta}{2}|0011\rangle$ for some unknown θ. We can compress it to a 1-qubit state without knowing θ.

Exercise:
Design a θ-independent circuit for compression.

Quantum autoencoder

- Any universal approach? Yes, the quantum autoencoder!
- Encoder $\mathcal{E}$:
  It consists in a variational circuit, whose input is taken to be the data to be compressed, and the action of discarding part of the output.
- The remaining part (k – qubit bottleneck) is sent to the decoder.
- Decoder $\mathcal{D}$:
  It consists in preparing ancillary qubits initialized in $|0\rangle$ and applying another variational circuit on these qubits and the output of $\mathcal{E}$.





Cost function of autoencoder

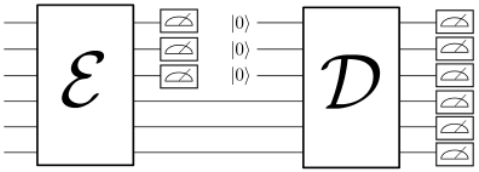
- Question: How to choose the cost function?
- (Faithfulness)
  Our goal is to make sure that the decoded state ≈ the initial state.
- We can use the average fidelity as the cost function (when the variational circuits have parameters $\vec{\theta}$):

$$C(\vec{\theta}) = \sum_i \frac{1}{m} f_i$$

  with $f_i =$ being the fidelity between the decoded state when the input is $|\psi_i\rangle$, and $|\psi_i\rangle$ itself.
- (Measurability) The cost function can be efficiently measured via a swap test gadget.

Procedure of autoencoder (preliminary version)

- **Input:** a set of m $n-$ qubit states $\{|\psi_i\rangle\}$.
  Error threshold ϵ. Desired compressed state size k (with $k < n$).
- **Output:** A pair of an encoder $\mathcal{E}$ and a decoder $\mathcal{D}$ such that $\mathcal{E}|\psi_i\rangle$ is $k-$ qubit, and $\mathcal{D}\mathcal{E}|\psi_i\rangle \approx |\psi_i\rangle$ for every $|\psi_i\rangle$.



- **Autoencoder routine**:
  1. Randomly initialize two $n-$ qubit variational circuits for $\mathcal{E}$ and $\mathcal{D}$ (with the autoencoder structure as shown in the figure) with parameters $\vec{\theta}$.
  2. For $\epsilon' > \epsilon$, do:
     - Pass each $|\psi_i\rangle$ through the encode and the decoder.
     - Evaluate the fidelity between the output and $|\psi_i\rangle$ via swap tests, and then calculate $C(\vec{\theta})$. Do gradient descend wrt. $C(\vec{\theta})$ and update the parameters to $\vec{\theta}'$.
     - Evaluate the cost function $C(\vec{\theta}') =: 1 - \epsilon'$.
  3. Output the encoder and the decoder.
- **Remark:** We can keep reducing k until the algorithm takes too long time to converge. The final k would be the minimum size of these states.

Is there any way of simplifying the training, especially the quantum part?

Cost function revisited

- Question: What is a good (encoder $\mathcal{E}$ of) autoencoder?
- For any $i = 1, \ldots, m$, a good autoencoder should be a circuit $U(\vec{\theta})$ such that there exists a $(n-k)$-qubit state $|a\rangle$ satisfying:

$$U(\vec{\theta})|\psi_i\rangle \approx |\tilde{\psi}_i\rangle \otimes |a\rangle$$

n-qubit | k-qubit | Independent of i

Up to a small error

- Taking the inverse of $U(\vec{\theta})$ we have:

$$|\psi_i\rangle \approx U(\vec{\theta})^{\dagger}\left(|\tilde{\psi}_i\rangle \otimes |a\rangle\right).$$

- The point is: We don't need to train the decoder!

Cost function revisited

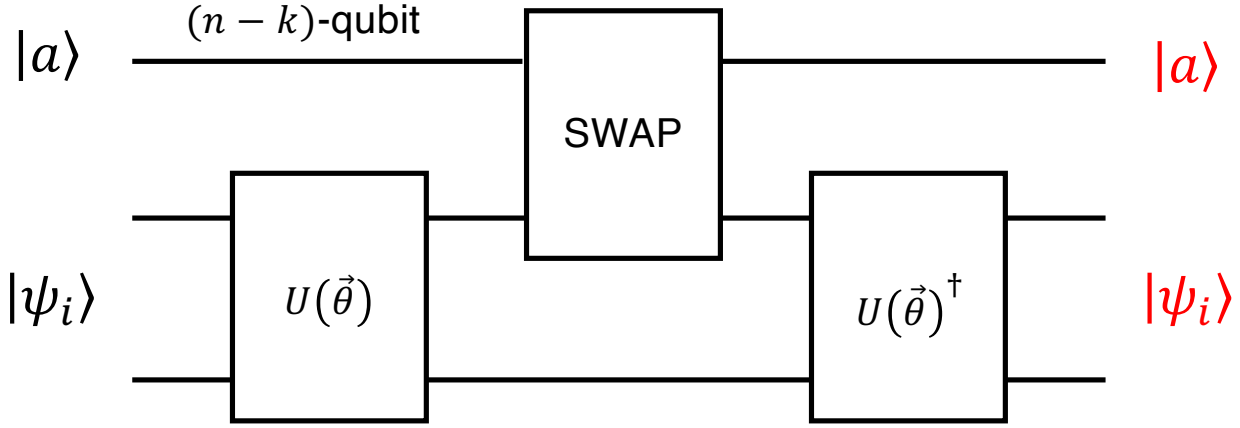
- For any $i = 1, \ldots, m$, a good autoencoder should be a circuit $U(\vec{\theta})$ such that

$$U(\vec{\theta})|\psi_i\rangle \approx |\tilde{\psi}_i\rangle \otimes |a\rangle$$

$$\Rightarrow |\psi_i\rangle \approx U(\vec{\theta})^{\dagger}\left(|\tilde{\psi}_i\rangle \otimes |a\rangle\right).$$

- Question: What should be the output, if the autoencoder has no error?

Answer:

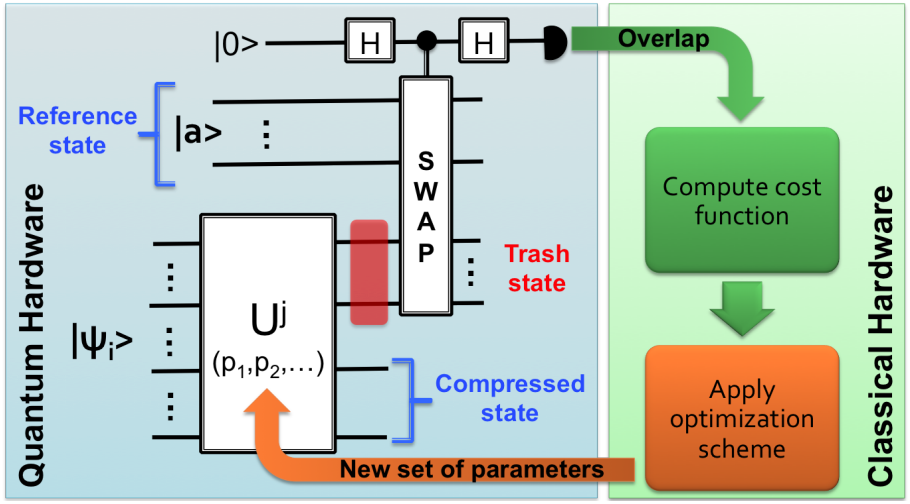


Cost function of the alternative autoencoder

- We know that the discarded state must be some (constant state) $|a\rangle$.
- We have 2 options for the cost function:

A. Same as before $C_A(\vec{\theta}) = \sum_i \frac{1}{m} f_i$ (average output fidelity).

B. Average "trashed" fidelity: $C_B(\vec{\theta}) = \sum_i \frac{1}{m} g_i$ with $g_i :=$ the fidelity between the discarded state and $|a\rangle$ when the input is $|\psi_i\rangle$.



Choosing the discarded state

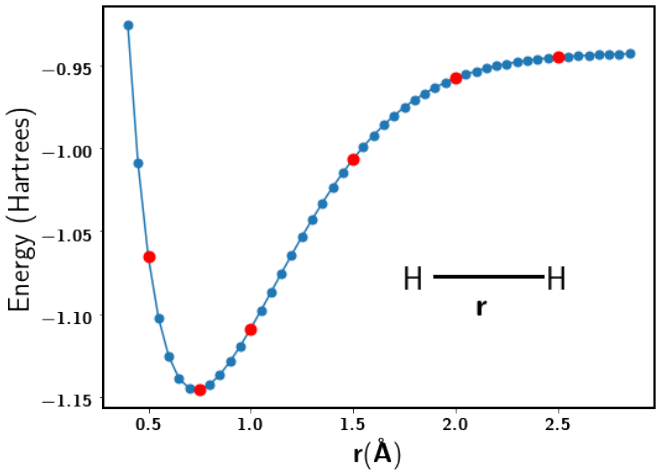
- We can choose $C_B(\vec{\theta}) = \sum_i \frac{1}{m} g_i$ with
  $g_i :=$ the fidelity between the discarded state and $|a\rangle$ when the input is $|\psi_i\rangle$.
- Even better, we can choose $|a\rangle$ to be a low-complexity state;
  for example, we can fix $|a\rangle = |0 \ldots 0\rangle$.
- Why?
  Suppose $|a\rangle = U_a|0 \ldots 0\rangle$ for some circuit U_a.
  We can merge U_a into the variational circuit $U(\vec{\theta})$ and construct a new circuit $U'(\vec{\theta}) = U(\vec{\theta})U_a$, and we have $U'(\vec{\theta})|\psi_i\rangle \approx |\tilde{\psi}_i\rangle \otimes |0 \ldots 0\rangle$.
- If the variational model $U(\vec{\theta})$ has enough representation power, there should be a configuration $\vec{\theta}'$ such that $U'(\vec{\theta}) \approx U(\vec{\theta}')$.
  By training wrt. C_B, we can find $\vec{\theta}'$ such that $U(\vec{\theta}')|\psi_i\rangle \approx |\tilde{\psi}_i\rangle \otimes |0 \ldots 0\rangle$.

Procedure of autoencoder (final version)

- Input: a set of m $n-$ qubit states $\{|\psi_i\rangle\}$.
  Error threshold ϵ. Desired compressed state size k (with $k < n$).
- Output: A pair of an encoder $\mathcal{E}$ and a decoder $\mathcal{D}$ such that
  $\mathcal{E}|\psi_i\rangle$ is $k-$ qubit, and $\mathcal{D}\mathcal{E}|\psi_i\rangle \approx |\psi_i\rangle$ for every $|\psi_i\rangle$.
- **Autoencoder routine:**
  1. Randomly initialize one $n-$ qubit variational circuit $U(\vec{\theta})$ for $\mathcal{E}$.
  2. For $\epsilon' > \epsilon$, do:
     - Pass each $|\psi_i\rangle$ through $U(\vec{\theta})$.
     - Measure the first $(n-k)$ output qubits in the computational basis.
     Evaluate $g_i :=$ probability of getting the outcome $00\ldots0$.
     - Compute $C(\vec{\theta}) := \frac{1}{m}\sum_i g_i$.
     - Do gradient descend and update $\vec{\theta} \to \vec{\theta}'$. Compute $C(\vec{\theta}') =: 1-\epsilon'$.
  3. Output the encoder as $U(\vec{\theta})$ and the decoder as $U^\dagger(\vec{\theta})$.
- **Remark:** We only need to train the encoder!

Demonstration in a quantum chemistry problem

- The autoencoder has been applied to compress the ground state of H_2 (obtained via VQE) with different values of inter-nuclear distance r.
- The original states are 4 qubits and get compressed to 2 qubits and 1 qubit.



Training vs Testing data

Circuit	Final size (# qubits)	Set	$-\log_{10}(1-\mathcal{F})$ MAE	-$\log_{10}$ Energy MAE (Hartrees)
Model A	2	Training	6.96(6.82-7.17)	6.64(6.27-7.06)
	2	Testing	6.99(6.81-7.21)	6.76(6.18-7.10)
	1	Training	6.92(6.80-7.07)	6.60(6.23-7.05)
	1	Testing	6.96(6.77-7.08)	6.72(6.15-7.05)
Model B	2	Training	6.11(5.94-6.21)	6.00(5.78-6.21)
	2	Testing	6.07(5.91-6.21)	6.03(5.70-6.21)
	1	Training	3.95(3.53-5.24)	3.74(3.38-4.57)
	1	Testing	3.81(3.50-5.38)	3.62(3.35-4.65)

* MAE: Mean Absolute Error. Log chemical accuracy in Hartrees ≈-2.80

Part V:
Barren Plateaus in Quantum Deep Learning

Recall: requirements on VQA

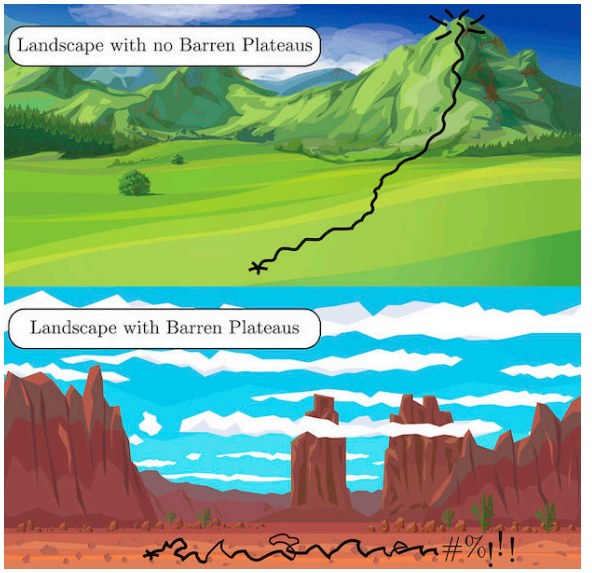
- A VQA should have a cost function that satisfies the following properties:

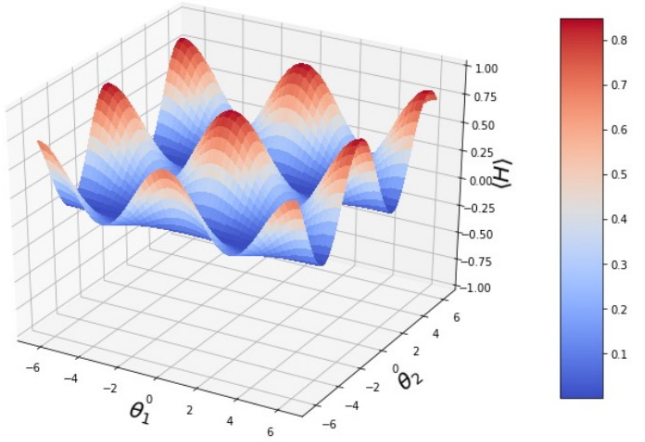
1. **Trainability:**
   It should be possible to efficiently optimize the parameters θ.
2. **Faithfulness:**
   The minimum of $C(\theta)$ should correspond to the solution of the problem.
3. **Measurability:**
   It should be efficiently estimable by performing measurements on a quantum computer and post-processing the classical data.

- The trainability may become a series issue in many tasks.

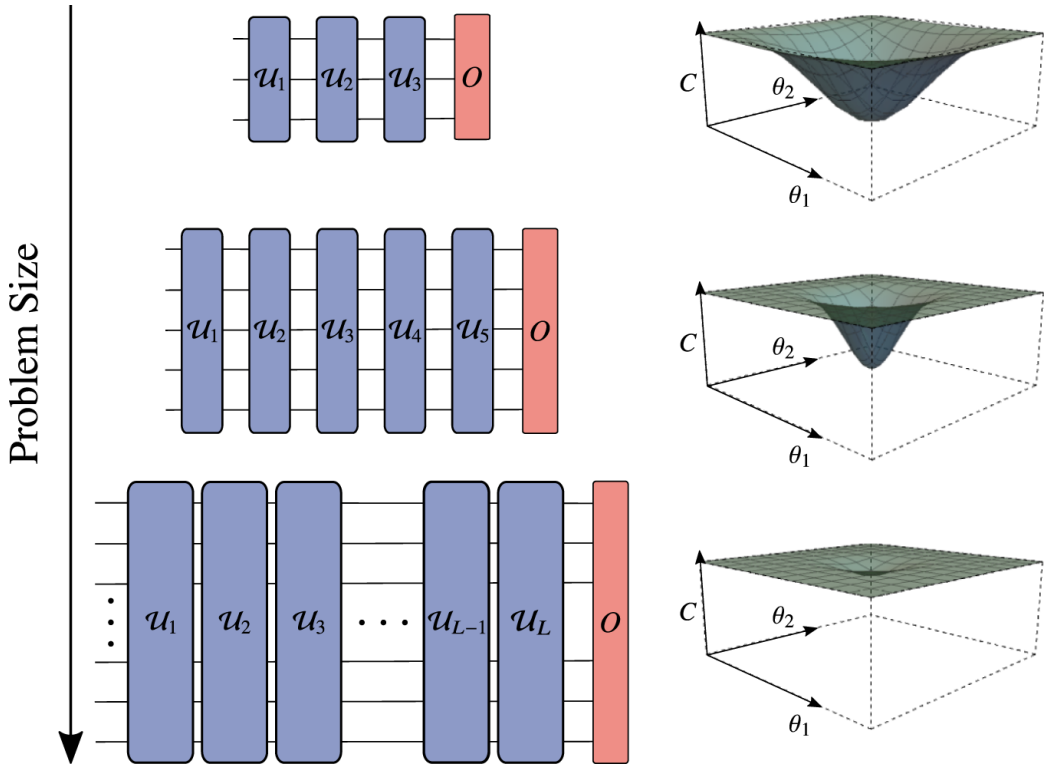
Barren plateau

- QNN training relies on the gradient descent method which, however, might suffer from the barren plateau issue.
- Barren plateau:
  Areas in the landscape where all gradients are very close to zero, i.e., areas that are very “flat”.
- When a barren plateau is encountered, the algorithm does not know where to go, since all gradients vanish!





Landscape: namely the cost function plotted as a function of the variable parameters $\vec{\theta}$.



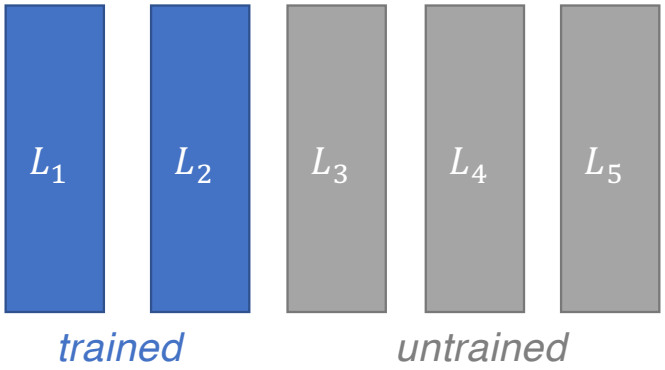
The parametric space becomes flatter, as the problem becomes harder and the required variational circuit grows deeper. [Nat. Comm. 12, 6961 (2021)]

Barren plateau in "deep" QNNs

- Barren plateau can dramatically slow down the training of a QNN, making it non-beneficial.
- It has been shown that the chance of seeing a Barren plateau increases as the variational circuit grows deeper.
- For n-qubit circuits, when depth is $\mathrm{poly}(n)$ and parameters are initialized completely randomly, $\langle \partial_k C \rangle = 0, \mathrm{Var}[\partial_k C] \sim 2^{-n}$ for every k.
- **Conclusion:** Trainability in QNNs, especially deeper QNNs, is an important issue that requires more study.

Mitigating barren plateaus

- Layer-wise learning [Skolik et al. Quantum Machine Intelligence Vol. 3, No. 5 (2021)]:
  - Start from a very shallow circuit and train it well.
  - Add more layers to the pretrained circuit and train only the new layers …



- Correlated initialization:
  - a major reason for barren plateaus is a fully-random initialization of parameters
  - instead, we could enforce certain correlations in the initialization.
    For instance, we divide the variational circuit into two parts: the first part admits a fully random initialization, and the second part is initialized to be the inverse of the first part.

Homework

- Review the lecture slides; you may find the review questions in the next slides helpful.
- Optional: Read the review paper on quantum machine learning (*doi:10.1038/nature23474*).

Review questions

- What is the advantage of kernels?
- After encoding the data into a quantum state, how could we compute the kernel function?
- The quantum autoencoder consists of two QNNs (variational circuits); what are they?
- What is the cost function of the quantum autoencoder?
- In the quantum autoencoder, do we have to train the decoder separately? Why?
- Can you come up with a few ways to remedy the barren plateau phenomenon?