Advanced Quantum Kernel Construction for Quantum Machine Learning

Seong Heon Song
Nano Engineering
Electronic/Electrical Engineering

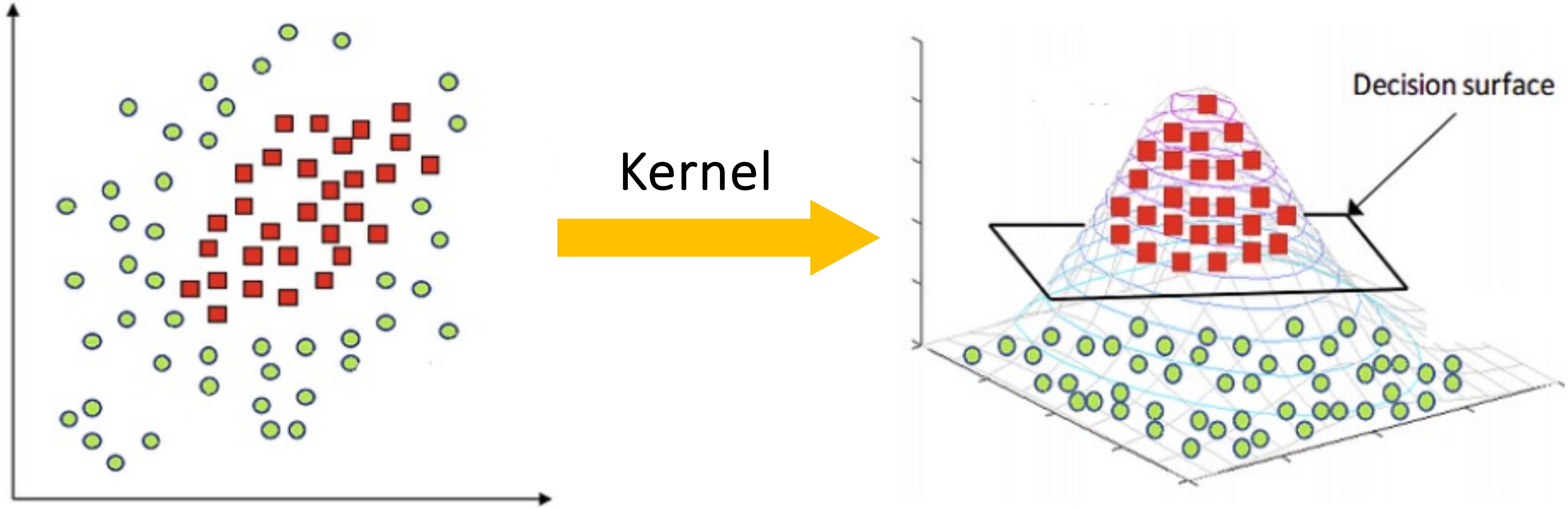
Sung Yeon Kook
Electrical Engineering

Overview

- What is Quantum Kernel?
- Suggestions for Advanced Quantum Kernel
  - Weight function
  - RY Rotation
  - Von Neumann Entropy
- Discussion

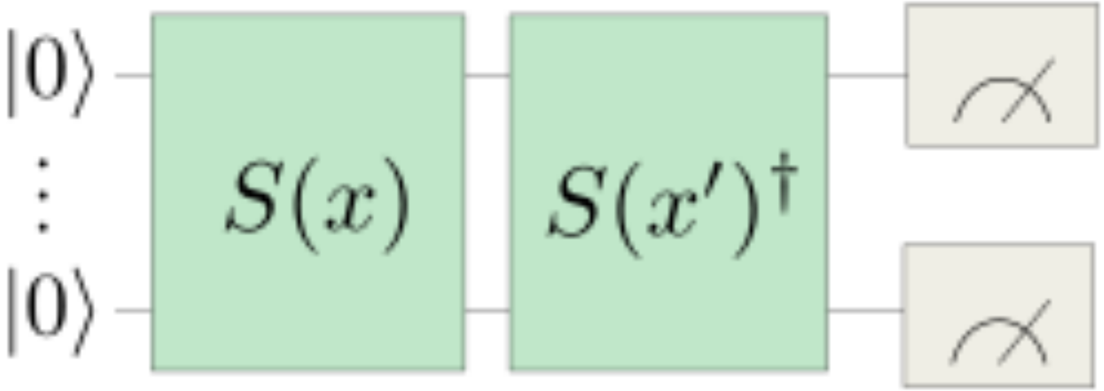
What is Quantum Kernel?

What is Quantum Kernel?



What is Quantum Kernel?

Quantum Kernel $K^{Q}_{x_i,x_j} \equiv \left|\langle \phi(x_j) | \phi(x_i) \rangle\right|^2$



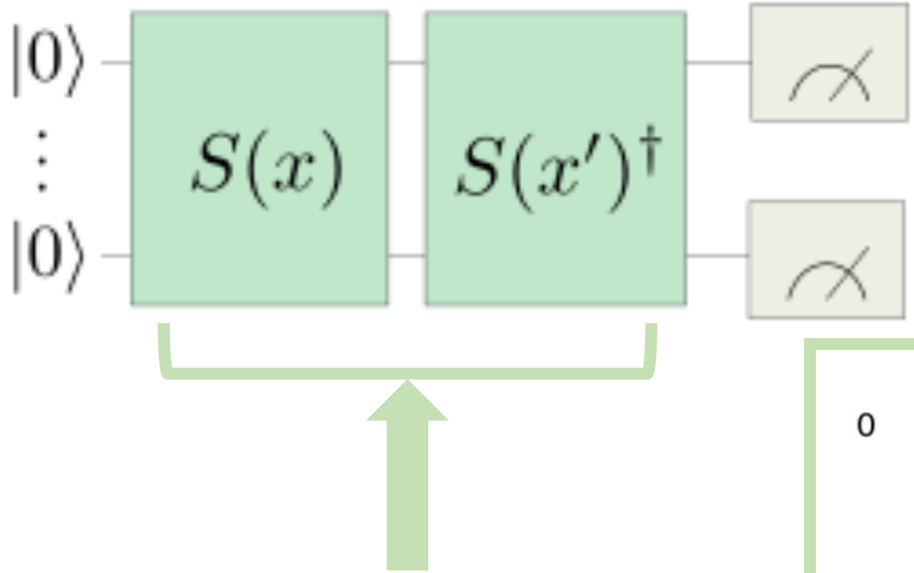
$\left\langle \begin{matrix} \textit{Projector } \mathcal{M} \\ |00\cdots0\rangle\langle00\cdots0| \end{matrix} \right\rangle$

Derive:

$$\langle \mathcal{M} \rangle = \langle 00\cdots0 | S(x_2) S^\dagger(x_1) \mathcal{M} S^\dagger(x_2) S(x_1) | 00\cdots0 \rangle$$
$$= \langle 00\cdots0 | S(x_2) S^\dagger(x_1) \, |00\cdots0\rangle\langle 00\cdots0| S^\dagger(x_2) S(x_1) | 00\cdots0 \rangle$$
$$= \{\langle 00\cdots0 | S^\dagger(x_2) S(x_1) | 00\cdots0 \rangle\}^* \langle 00\cdots0 | S^\dagger(x_2) S(x_1) | 00\cdots0 \rangle$$
$$= \left| \underbrace{\langle 00\cdots0 | S^\dagger(x_2)}_{\langle \phi(x_2)|} \underbrace{S(x_1) | 00\cdots0 \rangle}_{|\phi(x_1)\rangle} \right|^2 = |\langle \phi(x_2) | \phi(x_1) \rangle|^2 = K^{Q}_{x_1,x_2}$$

What is Quantum Kernel?

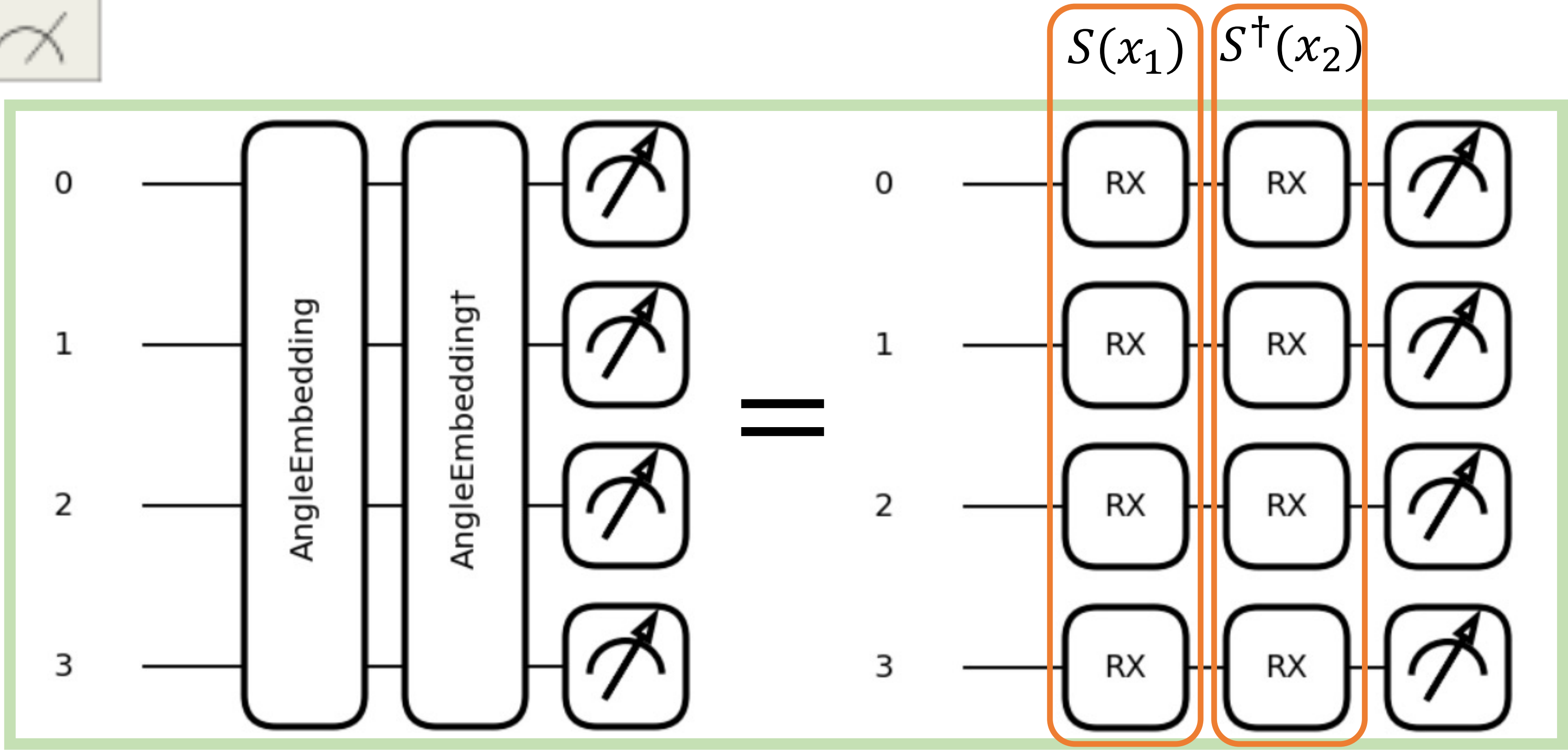
Quantum Kernel $K^{Q}_{x_i,x_j} \equiv \left|\langle \phi(x_j) | \phi(x_i) \rangle\right|^2$



$\langle \mathcal{M} \rangle$

Angle Embedding

RX Rotation ($\theta = x_1$)



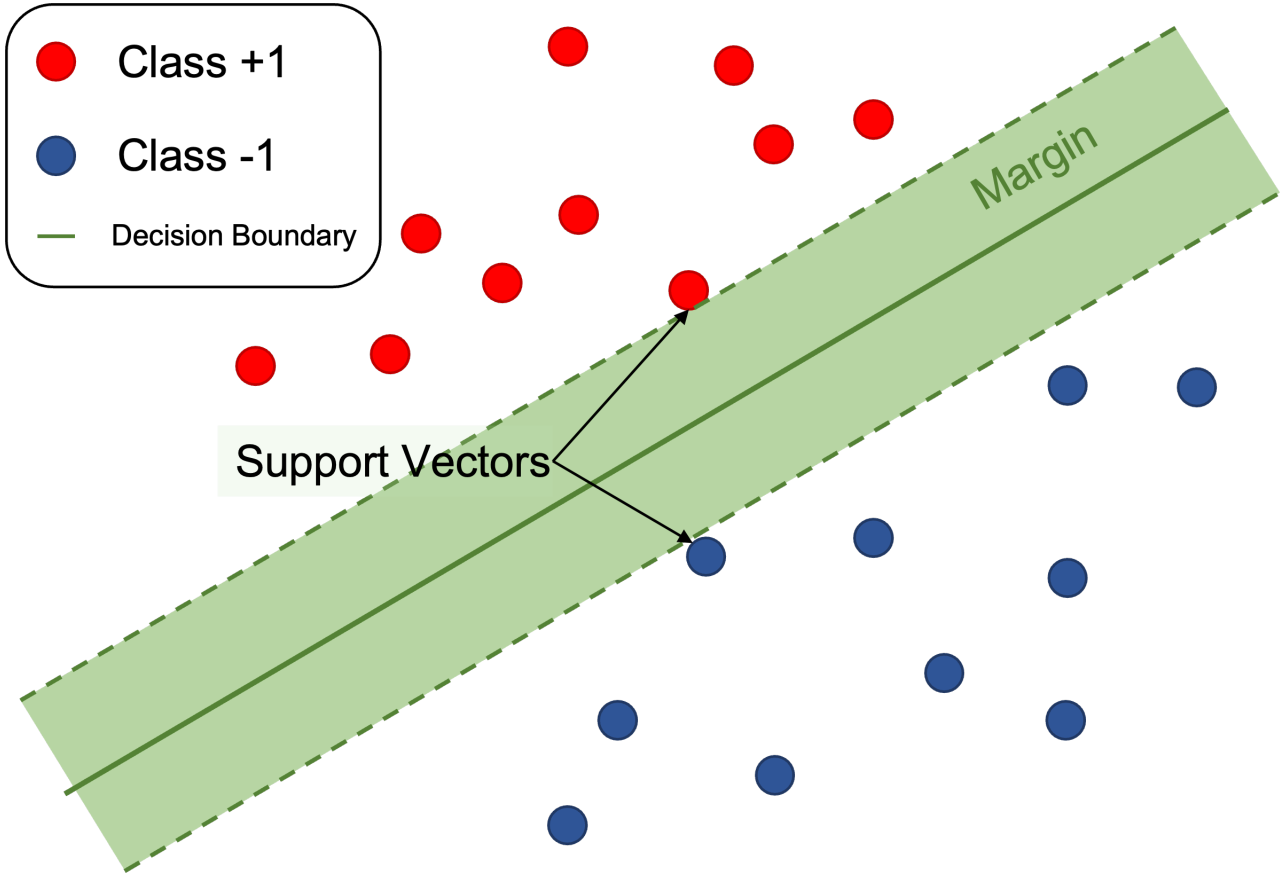
Suggestions for Advanced Quantum Kernel

- Weight function
- RY Rotation
- Von Neumann Entropy

How to measure efficiency?

- Prediction Accuracy
- Margin

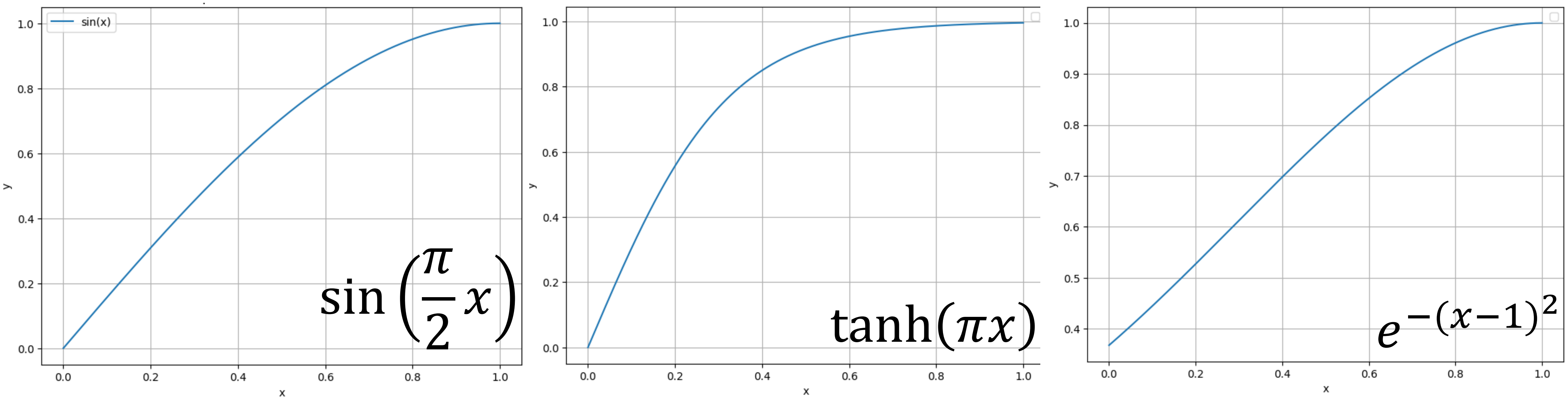
```
svm = SVC(kernel=kernel_matrix).fit(X_train, y_train)

predictions = svm.predict(X_test)
print('Accuracy: ',accuracy_score(predictions, y_test))

margin = 2 / np.linalg.norm(svm.dual_coef_)
print("Margin of the SVM:", margin)
```



Advanced Quantum Kernel ①

Weight function $K^{Q}_{x_i,x_j} \equiv Weight\left(\left|\langle\phi(x_j)|\phi(x_i)\rangle\right|^2\right)\left|\langle\phi(x_j)|\phi(x_i)\rangle\right|^2$

$$Weight(x) = \begin{cases} \sin\left(\frac{\pi}{2}x\right) \\ \tanh(\pi x) \\ e^{-(x-1)^2} \end{cases}, x \in [0,1]$$



Advanced Quantum Kernel ①

Weight function $K^{Q}_{x_i,x_j} \equiv \boldsymbol{Weight}\left(\left|\langle \phi(x_j) | \phi(x_i) \rangle\right|^2\right) \left|\langle \phi(x_j) | \phi(x_i) \rangle\right|^2$

$$\boldsymbol{Weight}(x) = \begin{cases} \sin\left(\frac{\pi}{2}x\right) \\ \tanh(\pi x) \\ e^{-(x-1)^2} \end{cases}, x \in [0,1]$$

```
dev_kernel = qml.device("lightning.qubit", wires=n_qubits)
@qml.qnode(dev_kernel, interface="autograd")
def kernel_step1(x1, x2):
    qml.AngleEmbedding(x1, wires=range(n_qubits))
    qml.adjoint(AngleEmbedding)(x2, wires=range(n_qubits))
    return qml.expval(qml.Hermitian(projector, wires=range(n_qubits)))
```

$\sin\left(\frac{\pi}{2}x\right)$

```
def kernel_step2(x1, x2):
    temp=kernel_step1(x1,x2)
    kernel_final=((np.sin((np.pi/2)*temp)))*temp
    return kernel_final
```

$\tanh(\pi x)$

```
def kernel_step2(x1, x2):
    temp=kernel_step1(x1,x2)
    kernel_final=np.tanh(np.pi*temp)*temp
    return kernel_final
```

$e^{-(x-1)^2}$

```
def kernel_step2(x1, x2):
    temp=kernel_step1(x1,x2)
    kernel_final=np.exp(-((temp-1)**2))*temp
    return kernel_final
```

Advanced Quantum Kernel ①

Weight function $K^{Q}_{x_i,x_j} \equiv \boldsymbol{Weight}\left(\left|\langle\phi(x_j)|\phi(x_i)\rangle\right|^2\right)\left|\langle\phi(x_j)|\phi(x_i)\rangle\right|^2$

$$\boldsymbol{Weight}(x) = \begin{cases} \sin\left(\frac{\pi}{2}x\right) \\ \tanh(\pi x) \\ e^{-(x-1)^2} \end{cases}, x \in [0, 1]$$

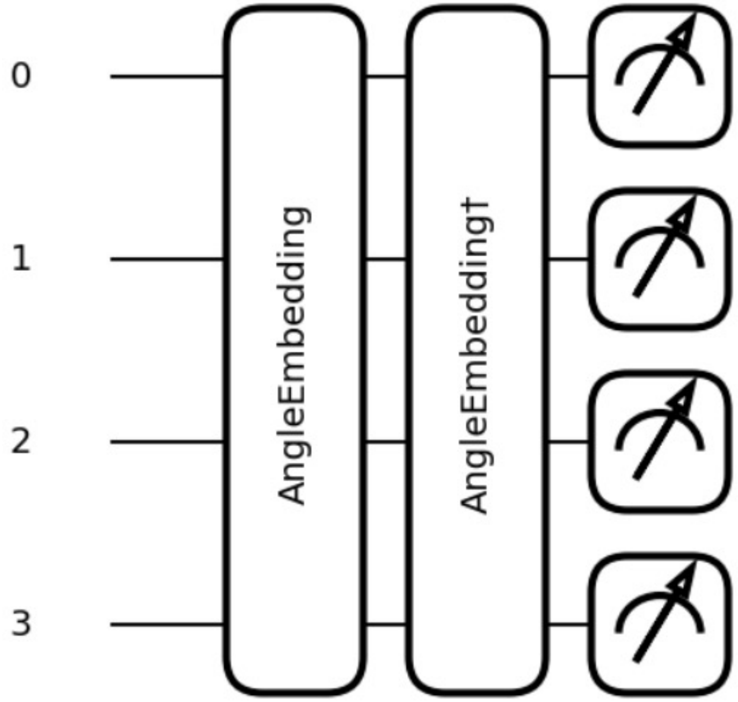
Weight	Efficiency	MAX	MIN
$\sin\left(\frac{\pi}{2}x\right)$	Accuracy: 1.0 Margin of the SVM: 0.9384953288568477	1.0000000000000009	1.6122592705159323e-14
$\tanh(\pi x)$	Accuracy: 1.0 Margin of the SVM: 0.9478802930649102	0.9962720762207509	3.224518541031769e-14
$e^{-(x-1)^2}$	Accuracy: 1.0 Margin of the SVM: 0.9456780466706409	1.0000000000000009	3.727031891869874e-08

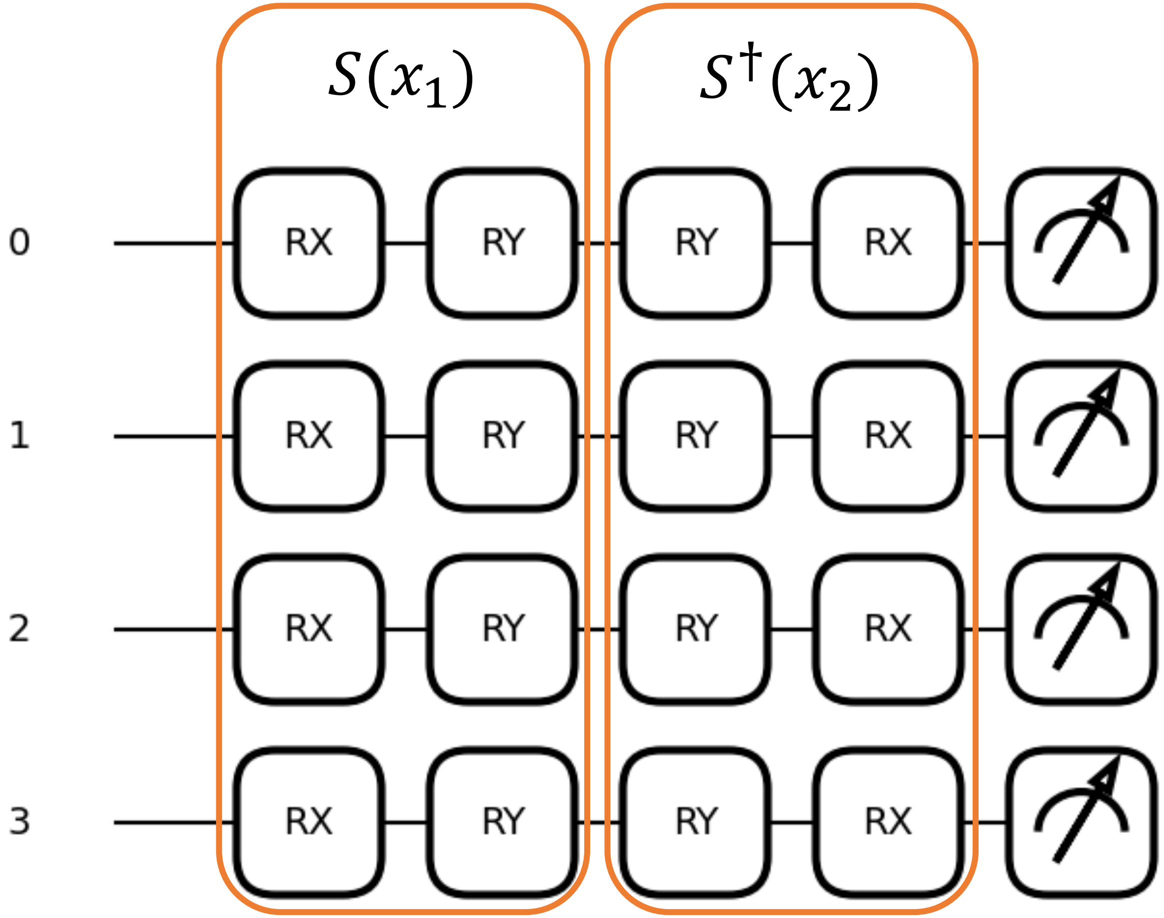
Advanced Quantum Kernel ②

RY Rotation $|\phi(x_i)\rangle = S(x_i)|00\cdots0\rangle$

$= RY(x_i)RX(x_i)|00\cdots0\rangle,\ S(x_i) = RY(x_i)RX(x_i)$

2-Step Angle Embedding





Advanced Quantum Kernel ②

RY Rotation $|\boldsymbol{\phi}(\boldsymbol{x_i})\rangle = \boldsymbol{S}(\boldsymbol{x_i})|\mathbf{00}\cdots\mathbf{0}\rangle$

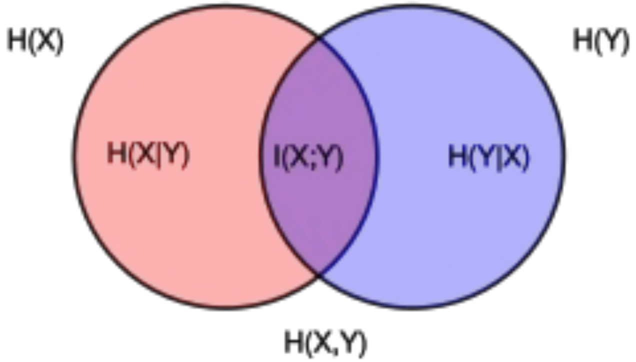
$= \boldsymbol{RY}(\boldsymbol{x_i})\boldsymbol{RX}(\boldsymbol{x_i})|\mathbf{00}\cdots\mathbf{0}\rangle,\ \boldsymbol{S}(\boldsymbol{x_i}) = \boldsymbol{RY}(\boldsymbol{x_i})\boldsymbol{RX}(\boldsymbol{x_i})$

```
@qml.qnode(dev_kernel, interface="autograd")
def kernel_step1(x1, x2):
    qml.AngleEmbedding(x1, wires=range(n_qubits))
    for i in range(n_qubits):
      qml.RY(x1[i], wires=i)
      qml.RY(-x2[i], wires=i)
    qml.adjoint(AngleEmbedding)(x2, wires=range(n_qubits))
    return qml.expval(qml.Hermitian(projector, wires=range(n_qubits)))
```

Efficiency	Accuracy: 1.0 Margin of the SVM: 1.0707388882260818
Max	1.0000000000000018
Min	8.309033480143277e-07

Von Neumann Entropy

- Entropy (Information Theory)
  : Average amount of information,
  Uncertainty under certain conditions



$$\mathrm{H}(X) := -\sum_{x\in\mathcal{X}} p(x)\log p(x) = \mathbb{E}[-\log p(X)]$$

- Mutual Information
  : Interdependence information of 2 random variables

$$I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\log\left(\frac{p(x,y)}{p(x)\,p(y)}\right) \equiv \mathrm{H}(X) + \mathrm{H}(Y) - \mathrm{H}(X,Y)$$

Von Neumann Entropy

- Von Neumann Entropy
  : Average amount of the information in a quantum system,
  Uncertainty in a quantum system

$$S = -\operatorname{tr}(\rho \ln \rho) = -\sum_j \eta_j \ln \eta_j \quad \left(\rho = \sum_j \eta_j |j\rangle \langle j| \right)$$

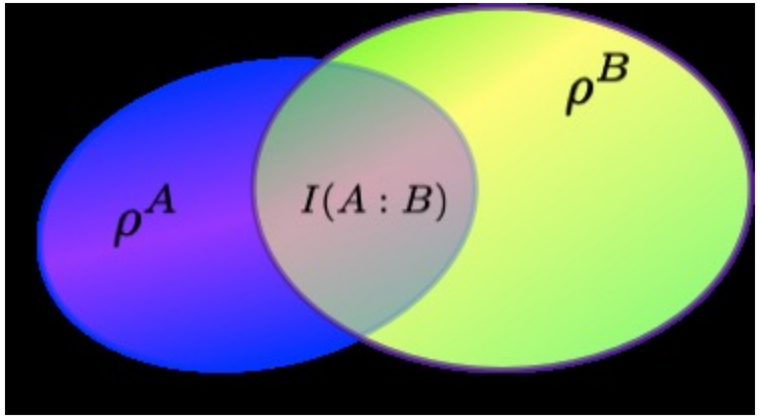
- Quantum Mutual Information
  : Interdependence information of 2 quantum systems

$$I(A:B) := S(\rho^A) + S(\rho^B) - S(\rho^{AB}) = S(\rho^{AB} \| \rho^A \otimes \rho^B)$$

Advanced Quantum Kernel ③

$$
\begin{aligned}
\boldsymbol{Mutual\ Information\ Kernel(A,B)} &= \boldsymbol{MI(A,B)} \\
&= \boldsymbol{I(A{:}B)} \\
&= \boldsymbol{S(\rho^A) + S(\rho^B) - S(\rho^{AB})} \\
&= \boldsymbol{Tr(\rho^A \log \rho^A) + Tr(\rho^B \log \rho^B) - Tr(\rho^{AB} \log \rho^{AB})}
\end{aligned}
$$

- Mutual Information Kernel
  → Correlation between 2 system



No Need to embed or project the data
As long as you have a density matrix

Advanced Quantum Kernel ③

- Implementation (Using density matrix) $S = -\mathrm{tr}(\rho \ln \rho)$

```
[63] import pennylane as qml
     import numpy as np
     from qiskit import QuantumCircuit, Aer

     # Set up the PennyLane device
     dev = qml.device("default.qubit", wires=4)
```

```
[64] # Define the quantum circuit for calculating the density matrix
     @qml.qnode(dev)
     def circuit(params):
         for i in range(4):
             qml.RY(params[i], wires=i)
         return qml.density_matrix(wires=list(range(4)))
```

```
[65] # Set the number of samples
     num_samples = 10

     # Loop to compute multiple mutual information values
     for _ in range(num_samples):
       # Generate random parameters for the quantum circuit
       np.random.seed()
       params_A = np.random.rand(4)
       params_B = np.random.rand(4)

       # Compute the density matrices for the two states using PennyLane
       rho_A = circuit(params_A)
       rho_B = circuit(params_B)
```

```
[66] # Calculate the von Neumann entropy of a density matrix
     def von_neumann_entropy(rho):
         eigvals = np.linalg.eigvalsh(rho)
         non_zero_eigvals = eigvals[eigvals > 1e-10]

         # Handle small eigenvalues by setting them to a small positive value
         non_zero_eigvals[non_zero_eigvals < 1e-10] = 1e-10

         entropy = -np.sum(non_zero_eigvals * np.log2(non_zero_eigvals))
         return entropy
```

```
[67] # Compute the von Neumann entropies of the individual states
     entropy_A = von_neumann_entropy(rho_A)
     entropy_B = von_neumann_entropy(rho_B)
```

```
[68] # Compute the von Neumann entropy of the combined state
     rho_AB = np.kron(rho_A, rho_B)
     entropy_AB = von_neumann_entropy(rho_AB)
```

```
[69] # Calculate the mutual information
     mutual_information = (entropy_A + entropy_B - entropy_AB)*10**14
```

```
[70] print("Mutual Information:", mutual_information)

     Mutual Information: 0.03203426503814917
```

Advanced Quantum Kernel ③

- Implementation (Using State Vecto

$$S = -\sum_j \eta_j \ln \eta_j \quad \left(\rho = \sum_j \eta_j |j\rangle \langle j| \right)$$

[79]
```
import pennylane as qml
from pennylane import numpy as np
from qiskit import QuantumCircuit, Aer, assemble
from qiskit.quantum_info import Statevector

# Set up the PennyLane device
dev = qml.device("default.qubit", wires=4)
```

[80]
```
# Define the quantum circuit for calculating the density matrix
@qml.qnode(dev)
def circuit(params):
    for i in range(4):
        qml.RY(params[i], wires=i)
    return qml.state()
```

[81]
```
# Set the number of samples
num_samples = 10

# Loop to compute multiple mutual information values
for _ in range(num_samples):
    # Generate random parameters for the quantum circuit
    np.random.seed()
    params_A = np.random.rand(4)
    params_B = np.random.rand(4)

    # Compute the state vectors for the two states using PennyLane
    statevector_A = circuit(params_A)
    statevector_B = circuit(params_B)
```

[82]
```
# Convert state vectors to Qiskit's Statevector objects
sv_A = Statevector(statevector_A)
sv_B = Statevector(statevector_B)
```

[83]
```
# Calculate the von Neumann entropy of a state vector
def von_neumann_entropy(statevector):
    probabilities = np.abs(statevector) ** 2
    non_zero_probabilities = probabilities[probabilities > 1e-10]

    # Handle small eigenvalues by setting them to a small positive value
    non_zero_probabilities[non_zero_probabilities < 1e-10] = 1e-10

    entropy = -np.sum(non_zero_probabilities * np.log2(non_zero_probabilities))
    return entropy
```

[84]
```
# Compute the von Neumann entropies of the individual states
entropy_A = von_neumann_entropy(statevector_A)
entropy_B = von_neumann_entropy(statevector_B)
```

[85]
```
# Combine the two states and calculate the joint state vector
statevector_AB = np.kron(statevector_A, statevector_B)
```

[86]
```
# Compute the von Neumann entropy of the joint state
entropy_AB = von_neumann_entropy(statevector_AB)
```

[87]
```
# Calculate the mutual information
mutual_information = (entropy_A + entropy_B - entropy_AB)*10**7
```

[88]
```
print("Mutual Information:", mutual_information)
```

```
Mutual Information: 0.361733558662535871
```

Discussion

Discussion

- Comparison overall results

Method	Accuracy	Margin	MAX KERNEL	MIN KERNEL
Conventional	1.0	0.923745312129494	1.0000000000000009	1.013112101296485e-07
Weight(Exponential)	1.0	0.9456780466706409	1.0000000000000009	3.727031891869874e-08
Weight(Sine)	1.0	0.9384953288568477	1.0000000000000009	1.612259270515932e-14
Weight(Tanh)	1.0	0.9478802930649102	0.9962720762207509	3.224518541031769e-14
RY Rotation	1.0	1.0707388882260818	1.0000000000000018	8.309033480143277e-07
Mutual Information (Density Matrix)	0.28	0.5403325828707691	0.17037340414085358	-0.1922055902288949
Mutual Information (State vector)	0.56	0.23570226039551587	0.5759711818598134	-2.6645352591003757e-08

Discussion

- Research Significance
  - Design a way to increase the margin of the quantum kernel function.
  - Increase the weight by multiplying the function value one more time.
  - Suggest a more quantum mechanical method by calculating the density matrix.

THANK YOU

REFERENCE

[1] Wikipedia contributors. (2023). Von Neumann entropy. *Wikipedia*. https://en.wikipedia.org/wiki/Von_Neumann_entropy
[2] Wikipedia contributors. (2023b). Entropy (information theory). *Wikipedia*.
[3] https://en.wikipedia.org/wiki/Entropy_(information_theory)
[4] Wikipedia contributors. (2023a). Mutual information. *Wikipedia*. https://en.wikipedia.org/wiki/Mutual_information
[5] Wikipedia contributors. (2023a). Quantum mutual information. *Wikipedia*.
[6] https://en.wikipedia.org/wiki/Quantum_mutual_information
[7] Nielsen, M. A., & Chuang, I. L. (2010). *Quantum computation and quantum information: 10th Anniversary Edition*. Cambridge University Press.
[8] *Parameter-shift rules — PennyLane*. (n.d.). https://pennylane.ai/qml/glossary/parameter_shift
[9] Wikipedia contributors. (2023d). Pauli matrices. *Wikipedia*. https://en.wikipedia.org/wiki/Pauli_matrices