Quantum Machine Learning

Mahn-Soo Choi (Korea University)

양자 기계 학습은 양자 알고리즘을 기계 학습 프로그램 내에 통합하는 것을 말한다. 가장 일반적인 형태는 양자 향상된 기계 학습으로, 고전 데이터 분석을 위한 기계 학습 알고리즘을 양자 컴퓨터에서 실행하게 된다. 기계 학습 알고리즘은 흔히 방대한 양의 데이터를 계산하는 데 사용되지만, 양자 기계 학습은 계산 속도와 데이터 저장 용량을 개선하기 위해 큐비트 및 양자 연산 또는 특수한 양자 시스템을 활용한다. 이는 고전 및 양자 처리를 모두 포함하는 하이브리드 방법을 포함한다. 이 발표에서는 울프램 언어를 이용하여 손쉽게 양자 기계 학습을 시뮬레이션하면서, 실제 양자 컴퓨터에서 이루어지는 양자 기계 학습의 원리를 공부하는 방법을 소개한다. 특히, 양자 시뮬레이션 프레임워크 Q3를 이용하여 몇 가지 기본적인 양자 기계 학습 예를 소개할 것이다.

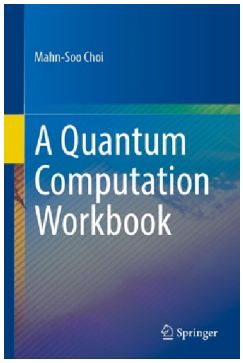
Advertisements

Q3: Symbolic Quantum Simulation

Q3 is a symbolic quantum simulation framework written in the Wolfram Language to help study quantum information systems, quantum many–body systems, and quantum spin systems; hence the name Q3. It provides various tools and utilities for symbolic and numerical calculations on these representative quantum systems.

- Q3 is free. You can download it from the GitHub repository: https://github.com/quantum-mob/Q3 .

A Quantum Computation Workbook



Chapter 1. Postulates of Quantum Mechanics
Chapter 2. Quantum Computation: Overview
Chapter 3. Virtual Realization of Quantum Computers
Chapter 4. Quantum Algorithms
Chapter 5. Quantum Decoherence
Chapter 6. Quantum Error Correction Codes
Chapter 7. Quantum Information Theory

- You can buy this book from the Springer site, https://link.springer.com/book/9783030912130.

School of Quantum

고려대학교(주관) 양자대학원은 고려대학교 등 9개 대학이 연합하여 양자기술 분야 박사급 전문인력 양성을 하기 위한 융합형 공유 대학원입니다. 본 양자대학원은 참여 대학, 연구소, 기업의 역량을 결집하여, 다양한 배경을 가진 인재들이 양자기술 분야에 성공적으로 진입하고, 전문적인 경력을 충실히 쌓으며 성장할 수 있게 함으로써, 국내 양자기술 연구 및 산업의 발전, 국가 경쟁력 확보, 미래 양자산업 발전을 이끌 수 있는 환경을 조성하여 제공하고자 합니다.

■ 더 자세한 사항은 홈페이지(https://QuantumWorkforce.kr)를 참조하세요.

Quantum Circuits

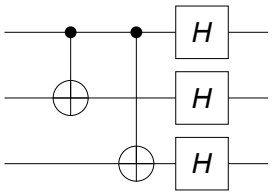
Select a symbol to refer to a quantum register of qubits.

In[•]:=
```
Let[Qubit, S]
```

Here is a typical quantum circuit including the CNOT and Hadamard gates as quantum circuit elements.

In[•]:=
```
gate = QuantumCircuit[
  CNOT[S[1], S[2]],
  CNOT[S[1], S[3]],
  S[{1, 2, 3}, 6]
 ]
```

Out[•]=



Evaluate the above quantum circuit in terms of Pauli operators.

In[•]:=
```
op = Elaborate[gate];
PauliForm[op]
```

Out[•]=

$$-\frac{I\otimes I\otimes I}{4\sqrt{2}}-\frac{i\,I\otimes I\otimes Y}{4\sqrt{2}}+\frac{I\otimes X\otimes X}{4\sqrt{2}}+\frac{I\otimes X\otimes Z}{4\sqrt{2}}-\frac{i\,I\otimes Y\otimes I}{4\sqrt{2}}+\frac{I\otimes Y\otimes Y}{4\sqrt{2}}+\frac{I\otimes Z\otimes X}{4\sqrt{2}}+\frac{I\otimes Z\otimes Z}{4\sqrt{2}}+$$
$$\frac{X\otimes I\otimes I}{4\sqrt{2}}+\frac{i\,X\otimes I\otimes Y}{4\sqrt{2}}+\frac{X\otimes X\otimes X}{4\sqrt{2}}+\frac{X\otimes X\otimes Z}{4\sqrt{2}}+\frac{i\,X\otimes Y\otimes I}{4\sqrt{2}}-\frac{X\otimes Y\otimes Y}{4\sqrt{2}}+\frac{X\otimes Z\otimes X}{4\sqrt{2}}+\frac{X\otimes Z\otimes Z}{4\sqrt{2}}+$$
$$\frac{i\,Y\otimes I\otimes I}{4\sqrt{2}}-\frac{Y\otimes I\otimes Y}{4\sqrt{2}}-\frac{i\,Y\otimes X\otimes X}{4\sqrt{2}}-\frac{i\,Y\otimes X\otimes Z}{4\sqrt{2}}-\frac{Y\otimes Y\otimes I}{4\sqrt{2}}-\frac{i\,Y\otimes Y\otimes Y}{4\sqrt{2}}-\frac{i\,Y\otimes Z\otimes X}{4\sqrt{2}}-\frac{i\,Y\otimes Z\otimes Z}{4\sqrt{2}}+$$
$$\frac{Z\otimes I\otimes I}{4\sqrt{2}}+\frac{i\,Z\otimes I\otimes Y}{4\sqrt{2}}+\frac{Z\otimes X\otimes X}{4\sqrt{2}}+\frac{Z\otimes X\otimes Z}{4\sqrt{2}}+\frac{i\,Z\otimes Y\otimes I}{4\sqrt{2}}-\frac{Z\otimes Y\otimes Y}{4\sqrt{2}}+\frac{Z\otimes Z\otimes X}{4\sqrt{2}}+\frac{Z\otimes Z\otimes Z}{4\sqrt{2}}$$

Evaluate the matrix representation of the above quantum circuit.

In[ ]:=
```
Matrix[gate] // MatrixForm
```

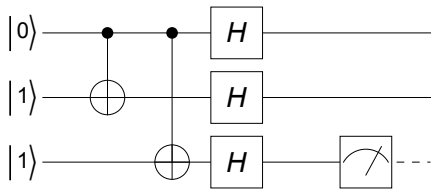
Out[ ]//MatrixForm=

$$\begin{pmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & -\frac{1}{2\sqrt{2}} \end{pmatrix}$$

In the above example, the quantum circuit involves only gate operations. More complete quantum circuit has input state and measurement as follows.

In[ ]:=
```
qc = QuantumCircuit[
  Ket[S@{1, 2, 3} → {0, 1, 1}],
  gate, "Spacer",
  Measurement[S[3, 3]]
 ]
```

Out[ ]=



Now, the evaluating the quantum circuit leads to the output state. Note that because of the measurement, the output state changes randomly at every run of the quantum circuit.

In[ ]:=
```
Elaborate[qc]
```

Out[ ]=

$$-\frac{1}{2}\left|0_{S_1}0_{S_2}1_{S_3}\right\rangle + \frac{1}{2}\left|0_{S_1}1_{S_2}1_{S_3}\right\rangle - \frac{1}{2}\left|1_{S_1}0_{S_2}1_{S_3}\right\rangle + \frac{1}{2}\left|1_{S_1}1_{S_2}1_{S_3}\right\rangle$$

Take the measurement comes running the quantum circuit many times.

In[ ]:=
```
data = Table[Elaborate[qc]; Readout[S[3, 3]], 20]
```

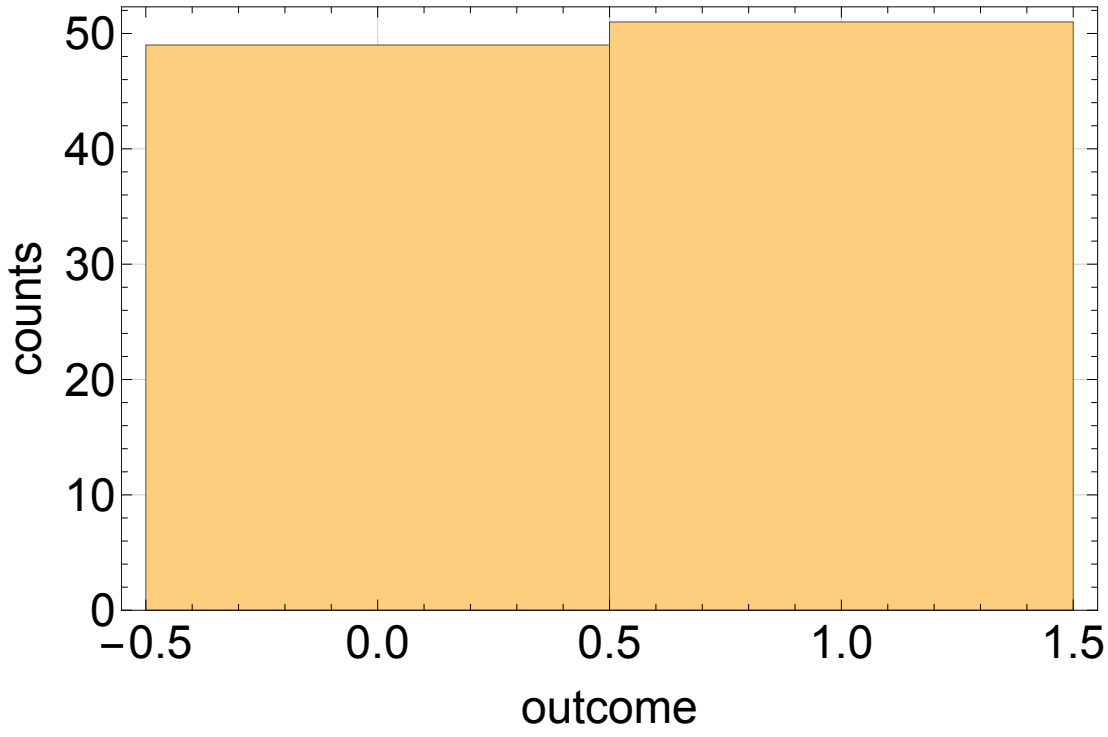
Out[ ]=

```
{1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1}
```

In[ ]:=
```
EchoTiming[data = Table[Elaborate[qc]; Readout[S[3, 3]], 100];]
Histogram[data, FrameLabel → {"outcome", "counts"}]
```

2.14127

Out[ ]=



Machine Learning

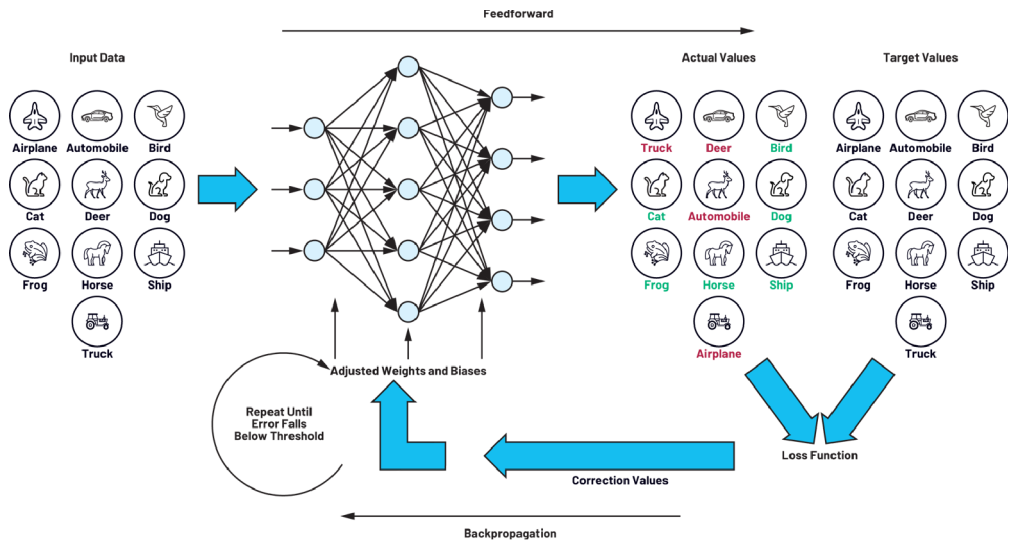


Figure 1. A training loop consisting of feedforward and backpropagation. Courtesy of O. Dreessen, Analogue Dialogue (March 2023).

- Wolfram Language supports various tools for machine learning. See the Machine Learning guide in Mathematica.

Quantum Machine Learning

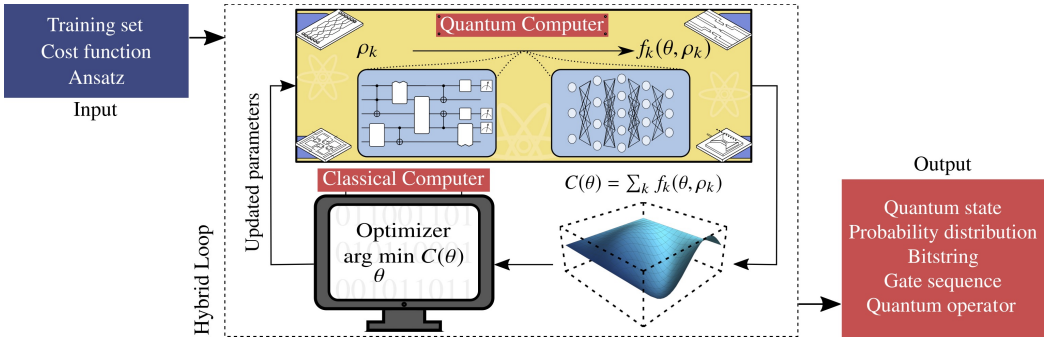


Figure 2. Typical workflow of the quantum-classical hybrid method for quantum machine learning. Courtesy of Cerezo *et al*., Nature Review Physics (2021).

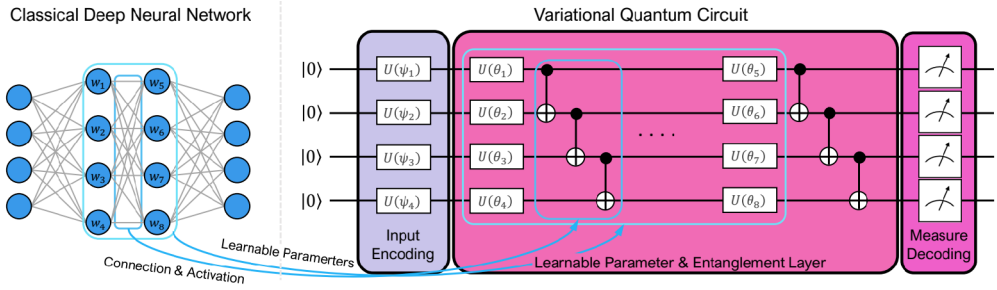


Figure 3. Comparison of classical deep neural network and variational quantum circuit. Courtesy of Y. Kwak *et al*., arXiv: 2202.11200.

Variational Quantum Classifier: Parity

Usually, Q3 is loaded automatically when Mathematica starts, but it does not harm to load it manually.

In[•]:=
```
<< Q3`
```

Setup

The qubits to be modeled are `S[i,$]` for $i = 1, \ldots, n$.

In[•]:=
```
Let[Qubit, S]
```

In[•]:=
```
$n = 4;
$N = Power[2, $n];
kk = Range[$n];
SS = S[kk, $]
```

Out[•]=
```
{S1, S2, S3, S4}
```

Data

Load and preprocess the data.

In[•]:=
```
bits = Tuples[{0, 1}, $n];
parity = 2 * Mod[Count[#, 1] & /@ bits, 2] - 1;
data = AssociationThread[bits → parity];
Normal[data] // TableForm
```

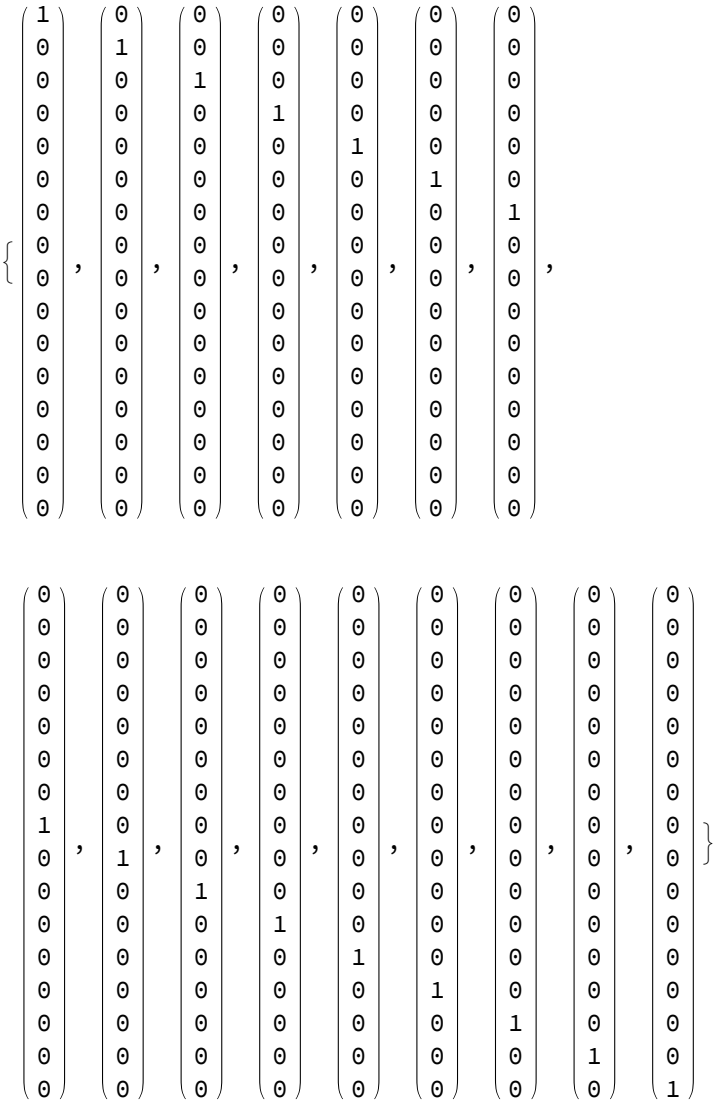
Out[•]//TableForm=

{0, 0, 0, 0}	→ −1
{0, 0, 0, 1}	→ 1
{0, 0, 1, 0}	→ 1
{0, 0, 1, 1}	→ −1
{0, 1, 0, 0}	→ 1
{0, 1, 0, 1}	→ −1
{0, 1, 1, 0}	→ −1
{0, 1, 1, 1}	→ 1
{1, 0, 0, 0}	→ 1
{1, 0, 0, 1}	→ −1
{1, 0, 1, 0}	→ −1
{1, 0, 1, 1}	→ 1
{1, 1, 0, 0}	→ −1
{1, 1, 0, 1}	→ 1
{1, 1, 1, 0}	→ 1
{1, 1, 1, 1}	→ −1

Embed the input data to quantum states.

```
In[ ]:= $in = BasisEmbedding[SS] /@ Keys[data];
        $in = Matrix /@ $in;
        MatrixForm /@ $in
```

Out[ ]=



Quantum Circuit Layer

The parameters to be optimized are `a[layer, i, j]` for $i = 1, \ldots, n$ and $j = 1, \ldots, 3$.

```
In[ ]:= Let[Real, a]
```

Define a quantum circuit layer.

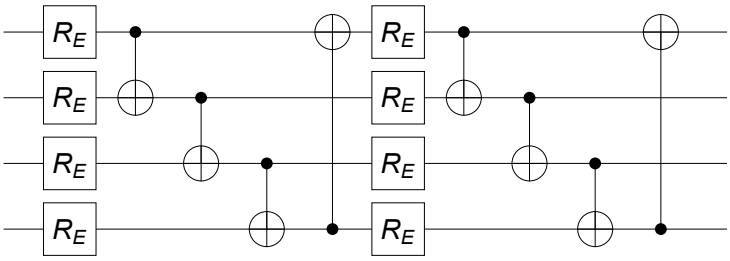
```
In[ ]:= layer[aa_?MatrixQ, ss : {__?QubitQ}] := QuantumCircuit[
          Table[EulerRotation[aa[[k]], ss[[k]]], {k, kk}],
          Apply[Sequence, CNOT @@@ Transpose@{ss, RotateLeft@ss}]
        ] /; Length[aa] == Length[ss]
        layer[aa_?MatrixQ] := layer[aa, SS]
        layer[n_Integer] := QuantumCircuit @@ Table[layer[Array[a[k], {$n, 3}]], {k, n}]
```

For example, the following quantum circuit contains 24 real parameters, with three parameters

$a[k, i, 1]$, $a[k, i, 2]$, $a[k, i, 3]$ associated with each qubit i on layer k.

In[ ]:=
```
ll = layer[2]
```

Out[ ]=



Optimization

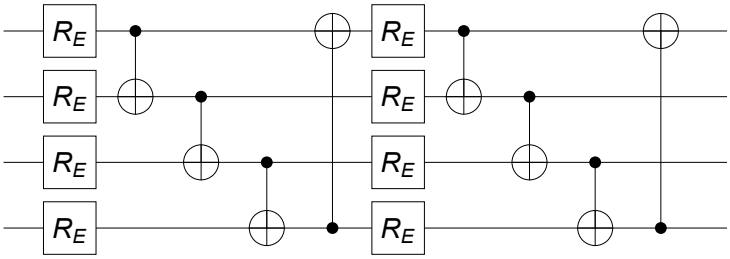
Use two layers.

In[ ]:=
```
$L = 2;
```

Pick a quantum circuit layer to apply on the input states.

In[ ]:=
```
op = layer[$L]
```

Out[ ]=



Check which parameters are actually involved in the above quantum circuit.

In[ ]:=
```
param = Cases[op, _a, Infinity]
Length[param]
```

Out[ ]=

$$\{a_{1,1,1}, a_{1,1,2}, a_{1,1,3}, a_{1,2,1}, a_{1,2,2}, a_{1,2,3}, a_{1,3,1}, a_{1,3,2}, a_{1,3,3}, a_{1,4,1}, a_{1,4,2}, a_{1,4,3}, a_{2,1,1}, a_{2,1,2}, a_{2,1,3}, a_{2,2,1}, a_{2,2,2}, a_{2,2,3}, a_{2,3,1}, a_{2,3,2}, a_{2,3,3}, a_{2,4,1}, a_{2,4,2}, a_{2,4,3}\}$$

Out[ ]=

```
24
```

In[ ]:=
```
mat = Normal@N@Matrix[op];
Dimensions[mat]
```

Out[ ]=

```
{16, 16}
```

In[ ]:=
```
Z1 = Matrix[S[1, 3], SS];
Dimensions[Z1]
```

Out[ ]=

```
{16, 16}
```

```
In[•]:= w0 = Flatten@RandomReal[{0, 1} Pi, {$L, $n, 3}]
        Length[w0]
```

```
Out[•]=
{2.97744, 1.71982, 1.04206, 0.681147, 3.13834, 0.270793, 2.80735, 1.66632,
 1.11766, 0.456491, 1.28482, 1.12747, 2.10388, 1.43562, 2.30304, 1.2068,
 0.548426, 0.914974, 2.94455, 2.40454, 2.04586, 2.49155, 0.401025, 1.17637}
```

```
Out[•]=
24
```

The set of parameters.

```
In[•]:= aa = a[Range@$L, Range@$n, {1, 2, 3}];
```

We prepare the natural constraints, but in this example, we will not impose them.

```
In[•]:= constraints = Thread[0 ≤ aa < Pi];
```

```
In[•]:= Clear[optimizer];
        optimizer[ww_] := Module[
          {ii = RandomChoice[Range@Length@data, 4],
           yy, vv, avg, sol, cost},
          yy = Part[Values@data, ii];
          vv = Transpose@SparseArray@Part[$in, ii];
          vv = Transpose[mat.vv];
          avg = Map[Conjugate[#].Z1.# &, vv];
          cost = Mean@Abs[avg - yy]^2;
          EchoTiming[
           {cost, sol} = FindMinimum[{cost, constraints}, Transpose@{aa, ww},
              StepMonitor :> PrintTemporary["Cost: ", cost],
              Method → "IPOPT"];
          ];
          Return[aa /. sol]
         ]
```

Test the optimizer.

```
In[•]:= optimizer[w0]
```

```
1.53254
```

```
Out[•]=
{2.92296, 3.12997, 1.20893, 0.934637, 3.13229, 0.637248, 3.12997, 1.5708,
 1.26152, 1.57079, 1.57078, 1.26831, 1.93571, 1.47921, 2.07668, 1.32303,
 0.00930808, 1.26962, 2.56323, 1.57078, 1.5708, 2.21392, 0.0116269, 1.15372}
```

Find the optimal values of the parameters iteratively.

```
In[•]:= $R = 3;(* the number of iterations *)
        ww = Nest[optimizer, w0, $R]
```

1.28871

0.997079

1.25245

Out[•]=

```
{1.57079, 3.12997, 1.57079, 1.56833, 3.12997, 1.57079, 3.12818, 1.5708,
 1.57079, 1.5708, 1.5708, 1.57079, 1.5708, 1.57081, 1.5708, 1.5708,
 0.0116213, 1.5687, 1.5708, 1.5708, 1.56409, 1.57079, 0.0116214, 1.57079}
```

From the optimized parameters, make predictions.

In[•]:=

```
mm = mat /. Thread[aa → ww];
vv = Transpose[mm.Transpose[$in]];
predicted = Map[Sign@Chop[Conjugate[#].Z1.#] &, vv]
```

Out[•]=

```
{-1, 1, 1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, 1, 1, -1}
```

In[•]:=

```
predicted - Values[data]
```

Out[•]=

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

Verdict

- Two layers is sufficient to properly predict the parity values.
- Using the `Method->"IPOPT"`, FindMinimum is significantly faster than the default method or any other documented methods (IPOTP seems to refer to the non-linear interior-point method from the IPOPT library).
- Otherwise, FindMinimum is rather slow to handle 24 parameters.

Variational Quantum Classifier: Iris Dataset

Usually, Q3 is loaded automatically when Mathematica starts, but it does not harm to load it manually.

In[•]:=

```
<< Q3`
```

Amplitude Embedding: Method

We want to embed classical data into a quantum state on quantum computer. One method is to use the amplitude embedding method.

In[•]:=

```
Let[Qubit, S]
```

In[•]:=

```
$n = 2;
$N = Power[2, $n];
kk = Range[$n];
SS = S[kk, $]
```

Out[•]=

$\{S_1, S_2\}$

Get the data.

```
In[•]:= aa = Normalize@RandomReal[{0, 1}, $N]
Out[•]=
{0.393255, 0.0491694, 0.789209, 0.469129}
```

Embed the data into a quantum state.

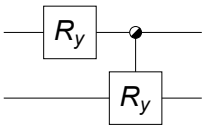
```
In[•]:= in = AmplitudeEmbedding[aa, SS]
Out[•]=
0.393255 |0_S1 0_S2⟩ + 0.0491694 |0_S1 1_S2⟩ + 0.789209 |1_S1 0_S2⟩ + 0.469129 |1_S1 1_S2⟩
```

Embedding the data into a quantum state as above on actual quantum computer is not trivial. One implementation based on quantum circuit model may be achieved by the following function.

```
In[•]:= op = AmplitudeEmbeddingGate[aa, SS]
Out[•]=
AmplitudeEmbeddingGate[{0.393255, 0.0491694, 0.789209, 0.469129}, {S1, S2}]
```

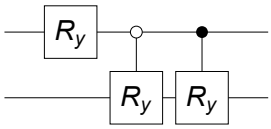
The above function may be represented by the following quantum circuit.

```
In[•]:= QuantumCircuit[Expand@op]
Out[•]=
```



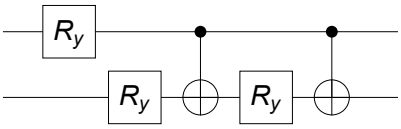
More explicitly, the above quantum circuit is equivalent to the following.

```
In[•]:= QuantumCircuit[ExpandAll@op]
Out[•]=
```



The above quantum circuit may be further simplified.

```
In[•]:= qc = QuantumCircuit[GateFactor /@ Expand@op]
Out[•]=
```



Verify the above quantum circuit.

```
In[•]:= new = qc ** Ket[SS] // Chop
Out[•]=
0.393255 |0_S1 0_S2⟩ + 0.0491694 |0_S1 1_S2⟩ + 0.789209 |1_S1 0_S2⟩ + 0.469129 |1_S1 1_S2⟩
```

```
In[•]:= Fidelity[new, in]
Out[•]=
1.
```

Data loading and embedding

Import the iris date and check the first three entries.

```
In[•]:= raw = Import[PacletObject["QuantumPlaybook"]["AssetLocation", "Iris"], "Table"];
raw = Map[Most[#] → Round[Last@#] &, raw];
raw[[ ;; 3]]
```

```
Out[•]=
{{0.399999999999999911, 0.750000000000000000,
    0.199999999999999956, 0.0500000000000000278} → -1,
 {0.300000000000000267, 0.500000000000000000, 0.199999999999999956,
    0.0500000000000000278} → -1, {0.200000000000000178,
    0.600000000000000089, 0.150000000000000222, 0.0500000000000000278} → -1}
```

Renormalize the data.

```
In[•]:= data =
  Thread[Map[Normalize@Join[Take[#, 2], 0.3 {1, 0}] &, Keys@raw] → Values[raw]];
data[[1]]
```

```
Out[•]=
{0.44376, 0.83205, 0.33282, 0.} → -1
```

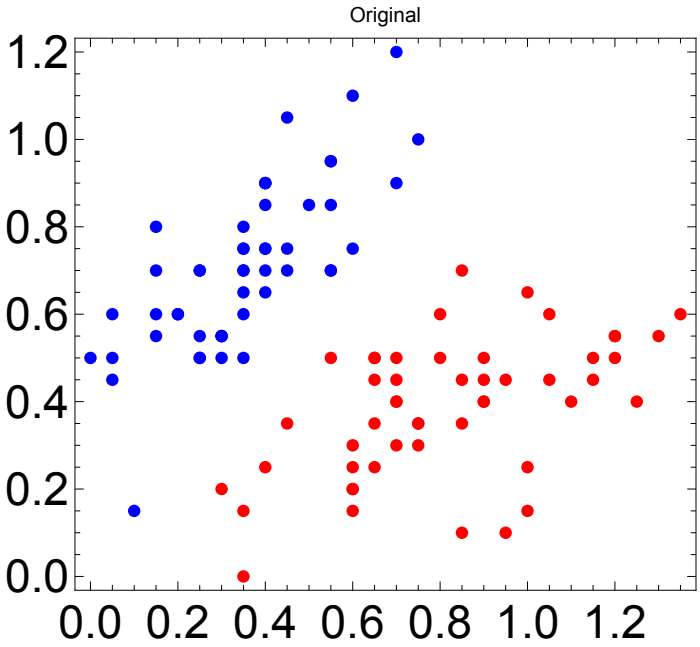
Here, vector representation is nothing but the input data itself since we just simulate the amplitude embedding.

```
In[•]:= $in = Keys[data];
```

Features

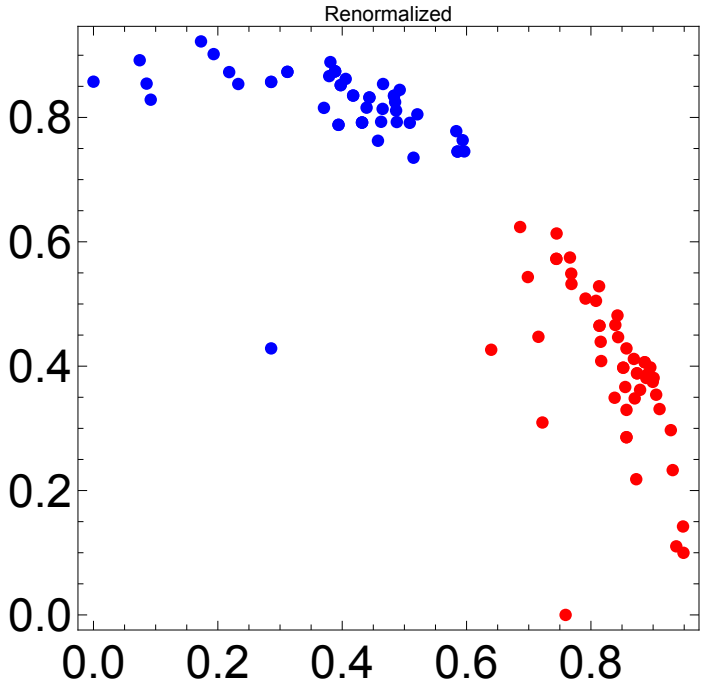
In[ ]:=
```
gg = Map[Take[#, 2] &, GroupBy[raw, Last, Map[First]], {2}];
Graphics[{PointSize[0.02], Red, Point /@ gg[1], Blue, Point /@ gg[-1]},
 PlotLabel → "Original",
 ImageSize → Medium,
 Frame → True]
```

Out[ ]=



In[ ]:=
```
gg = Map[Take[#, 2] &, GroupBy[data, Last, Map[First]], {2}];
Graphics[{PointSize[0.02], Red, Point /@ gg[1], Blue, Point /@ gg[-1]},
 PlotLabel → "Renormalized",
 ImageSize → Medium,
 Frame → True]
```

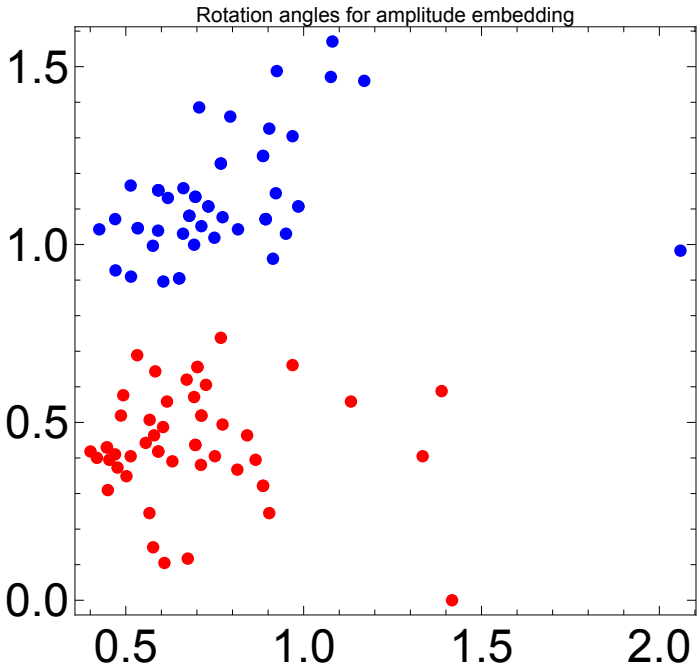
Out[ ]=



In[ ]:=
```
RR = AmplitudeEmbeddingGate[#, SS] & /@ Keys[data][[ ;; , ;; -2]];
RR = List @@@ Map[GateFactor, Expand /@ RR, {2}] /. {CNOT → Nothing};
angles = Map[RotationAngle, RR, {2}];
```

```
In[•]:= tf = Positive[Values@data] // Flatten;
        aa = Pick[angles[[ ;; , ;; 2]], tf];
        bb = Pick[angles[[ ;; , ;; 2]], Not /@ tf];
        Graphics[{PointSize[0.02], Red, Point /@ aa, Blue, Point /@ bb},
         PlotLabel → "Rotation angles for amplitude embedding",
         ImageSize → Medium,
         Frame → True]
```

Out[•]=



Quantum Circuit Layer

Construct the basic quantum circuit layer to use.

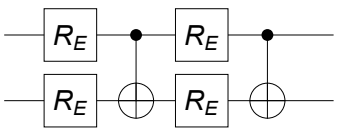
```
In[•]:= Let[Real, w]
```

```
In[•]:= layer[lyr_Integer] := QuantumCircuit[
          Table[EulerRotation[w[lyr, k, {1, 2, 3}], S[k, $]], {k, $n}],
          CNOT[S[1], S[2]]
         ]
        layer[ll : {__Integer}] := Apply[QuantumCircuit, layer /@ ll]
```

Check it by showing two layers.

```
In[•]:= layer[{1, 2}]
```

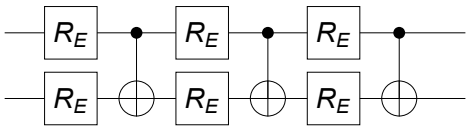
Out[•]=



Optimization using three layers

In[ ]:=
```
$L = 3;
$layer = layer[Range@$L]
$mat = Normal@Matrix[$layer, SS];
```

Out[ ]=



In[ ]:=
```
ww = w[Range@$L, Range@$n, {1, 2, 3}];
constraints = Thread[0 ≤ ww < Pi];
```

In[ ]:=
```
w0 = Flatten@RandomReal[{0, 1} * Pi, {$L, $n, 3}];
```

In[ ]:=
```
Z1 = Matrix[S[1, 3], SS];
```

In[ ]:=
```
Clear[optimizer]
optimizer[w0_] := Module[
  {ii = RandomChoice[Range@Length@data, 4],
   yy, vv, avg, sol, cost},
  yy = Part[Values@data, ii];
  vv = Transpose@Part[$in, ii];
  vv = Transpose[$mat.vv];
  avg = Map[Conjugate[#].Z1.# &, vv];
  cost = Mean@Abs[avg - yy]^2;
  EchoTiming[
   {cost, sol} = FindMinimum[{cost, constraints}, Transpose@{ww, w0},
      StepMonitor :> PrintTemporary["Cost: ", cost],
      Method → "IPOPT"];
  ];
  Return[ww /. sol]
 ]
```

Test the optimizer.

In[ ]:=
```
optimizer[w0]
```

0.654063

Out[ ]=
```
{2.8429, 1.60827, 2.16125, 2.03242, 2.09649, 2.09551, 2.69053, 0.00336912, 2.88112,
 2.52719, 1.90748, 0.327043, 1.18974, 1.61878, 2.52621, 1.32437, 1.4352, 2.58756}
```

Find the optimal values of the parameters iteratively.

In[ ]:=
```
$R = 5;(* the number of iterations *)
new = Nest[optimizer, w0, $R]
```

```
0.952526
1.9356
0.545481
0.691872
0.809051
```

Out[ ]=

```
{3.07765, 1.49956, 1.8516, 1.55063, 1.35282, 0.265915, 3.12278, 1.29472, 3.08055,
 1.77529, 2.38729, 0.419731, 1.45549, 1.53254, 3.12278, 1.50135, 1.52264, 1.82473}
```

From the optimized parameters, make predictions.

In[ ]:=
```
mm = $mat /. Thread[ww → new];
```

In[ ]:=
```
vv = Transpose[mm.Transpose[$in]];
predicted = Map[Sign@Chop[Conjugate[#].Z1.#] &, vv]
```

Out[ ]=

```
{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1,
 -1, -1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
```

In[ ]:=
```
predicted - Values[data]
```

Out[ ]=

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

Distance-Based Quantum Classifier: Iris Dataset

M. Schuld, M. Fingerhuth, and F. Petruccione, EPL 119, 60002 (2017).

We are given a training data set

$$\mathcal{D} = \{x^1 \to y^1, \boldsymbol{x}^2 \to y^2, \ldots, \boldsymbol{x}^M \to y^M\}$$

of inputs $\boldsymbol{x}^k \in \mathbb{R}^N$ ($N = 2^n$) with their respective target labels $y^k \in \{0, 1\}$.

We want to determine the label of a new input $\boldsymbol{x}$ using the threshold function

$$y_{\text{eval}}(\boldsymbol{x}) = \text{sign}\left(\sum_{m=1}^{M} \left\{1 - \frac{1}{4M} \mid \boldsymbol{x} - \boldsymbol{x}^m \mid^2\right\} y^m\right).$$

In[ ]:=
```
Let[Qubit, S]
```

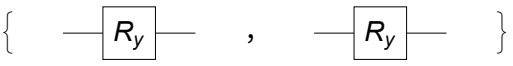
Here, only two features of two samples are considered for the purpose of demonstration. This data is after *standardization* and *normalization*.

```
In[•]:= training = Association[
          {0, 1} → 0, (*Iris sample 33*)
          {0.789, 0.615} → 1 (*Iris sample 85*)
        ];
      test = Association[
          {-0.549, 0.836} → 0, (*Iris sample 28*)
          {0.053, 0.999} → 0 (*Iris sample 36*)
        ];
```

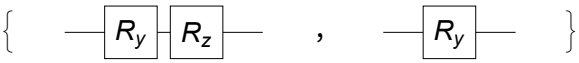
To use a quantum algorithm, each data point $\boldsymbol{x}$ is encoded to n qubits as amplitude (amplitude embedding),

```
In[•]:= trainingEmbed = Expand /@ (AmplitudeEmbeddingGate[#, {S[3]}] &) /@ Keys[training]
        testEmbed = Expand /@ (AmplitudeEmbeddingGate[#, {S[3]}] &) /@ Keys[test]
```

Out[•]=



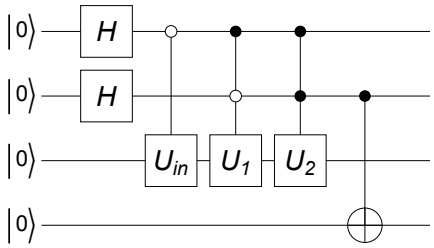
Out[•]=



To use the threshold function discussed above, we prepare a quantum state involving *ancillary qubit*, *index register*, *data register*, and *class register* (from the top to bottom).

```
In[•]:= state = QuantumCircuit[
          Ket[S@{1, 2, 3, 4}], S[{1, 2}, 6],
          ControlledGate[S[1] → 0, testEmbed⟦1⟧, "Label" → "U_in"],
          ControlledGate[S@{1, 2} → {1, 0}, trainingEmbed⟦1⟧, "Label" → "U_1"],
          ControlledGate[S@{1, 2} → {1, 1}, trainingEmbed⟦2⟧, "Label" → "U_2"],
          CNOT[S[2], S[4]]
        ]
```

Out[•]=



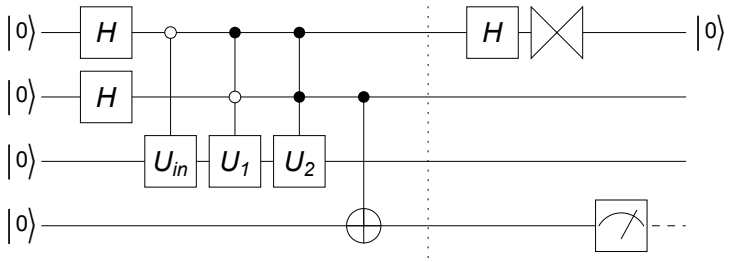
The resulting quantum state is given by

$$\left|\Psi\right\rangle = \frac{1}{\sqrt{2M}} \sum_{m=1}^{M} \left|m\right\rangle \otimes \left(\left|0\right\rangle \otimes \left|\psi_{\boldsymbol{x}}\right\rangle + \left|1\right\rangle \otimes \left|\psi_{\boldsymbol{x}^m}\right\rangle\right) \otimes \left|y^m\right\rangle$$

Now, it is time to infer the label of the input $\boldsymbol{x}$.

```
In[•]:= qc = QuantumCircuit[state, "Separator",
          S[1, 6], Projector[Ket[S[1] → 0]],
          Measurement[S[4, 3]],
          Ket[S[1] → 0]
         ]
```

Out[•]=



Running the above quantum circuit, one gets one instance of output state.

```
In[•]:= Elaborate[qc] // KetChop
```

Out[•]=

$$(0. + 0.388144\,i)\,\left|0_{S_1}0_{S_2}0_{S_3}0_{S_4}\right\rangle + (0.707107 - 0.591054\,i)\,\left|0_{S_1}0_{S_2}1_{S_3}0_{S_4}\right\rangle$$

Run the above quantum circuit many times to get the probability distribution of the measurement outcome.

```
In[•]:= EchoTiming[data = Table[Elaborate[qc]; Readout[S[4, 3]], 10]]
        0.497241
Out[•]=
        {1, 1, 1, 0, 0, 0, 0, 1, 1, 0}
```

```
In[•]:= EchoTiming[data = Table[Elaborate[qc]; Readout[S[4, 3]], 100];]
        Counts[data]
        4.8547
Out[•]=
        ‹| 0 → 55, 1 → 45 |›
```

One determines the more frequent outcome as the label of the input ***x***.

References

There are so many materials on quantum machine learning. This notebook is mainly based on the following two references.

- M. Schuld and F. Petruccione (2018), Supervised Learning with Quantum Computers (Springer).
- J. Biamonte *et al*., Nature 549, 195 (2017), “Quantum Machine Learning.”

For experts in machine learning, this review article may be interesting.

- K. A. Tychola *et al*., Electronics 12, 2379 (2023), “Quantum Machine Learning--An Overview.”