Quantum neural networks — a practical approach

Piotr Gawron

AstroCeNT—Particle Astrophysics Science and Technology Centre
International Research Agenda
Nicolaus Copernicus Astronomical Center, Polish Academy of Sciences

Warsaw / Gdańsk, 22 February 2021

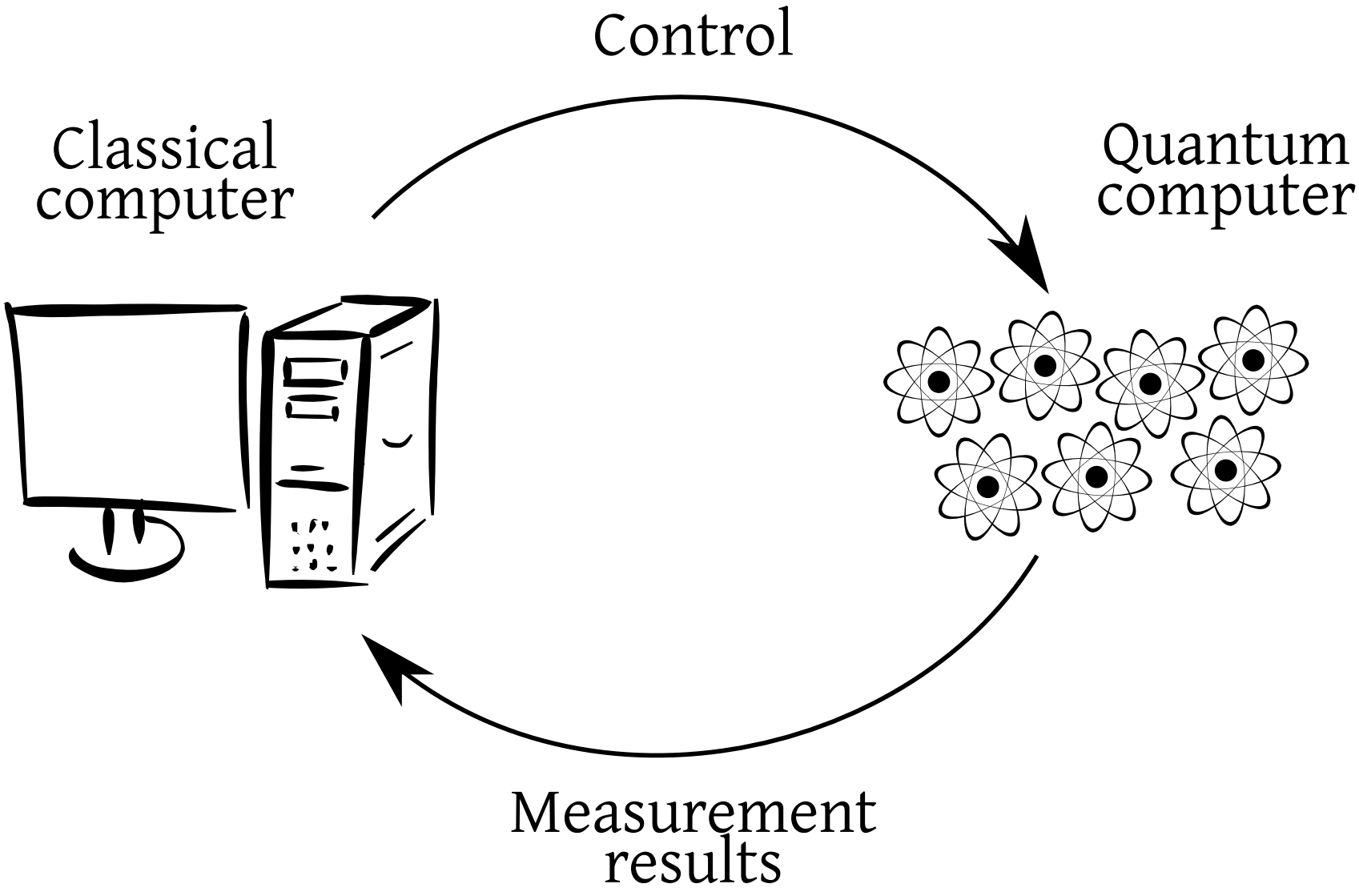
AstroCeNT
European Funds Smart Growth
Republic of Poland
FNP Foundation for Polish Science
European Union European Regional Development Fund
CAMK PAN

What we want to talk about

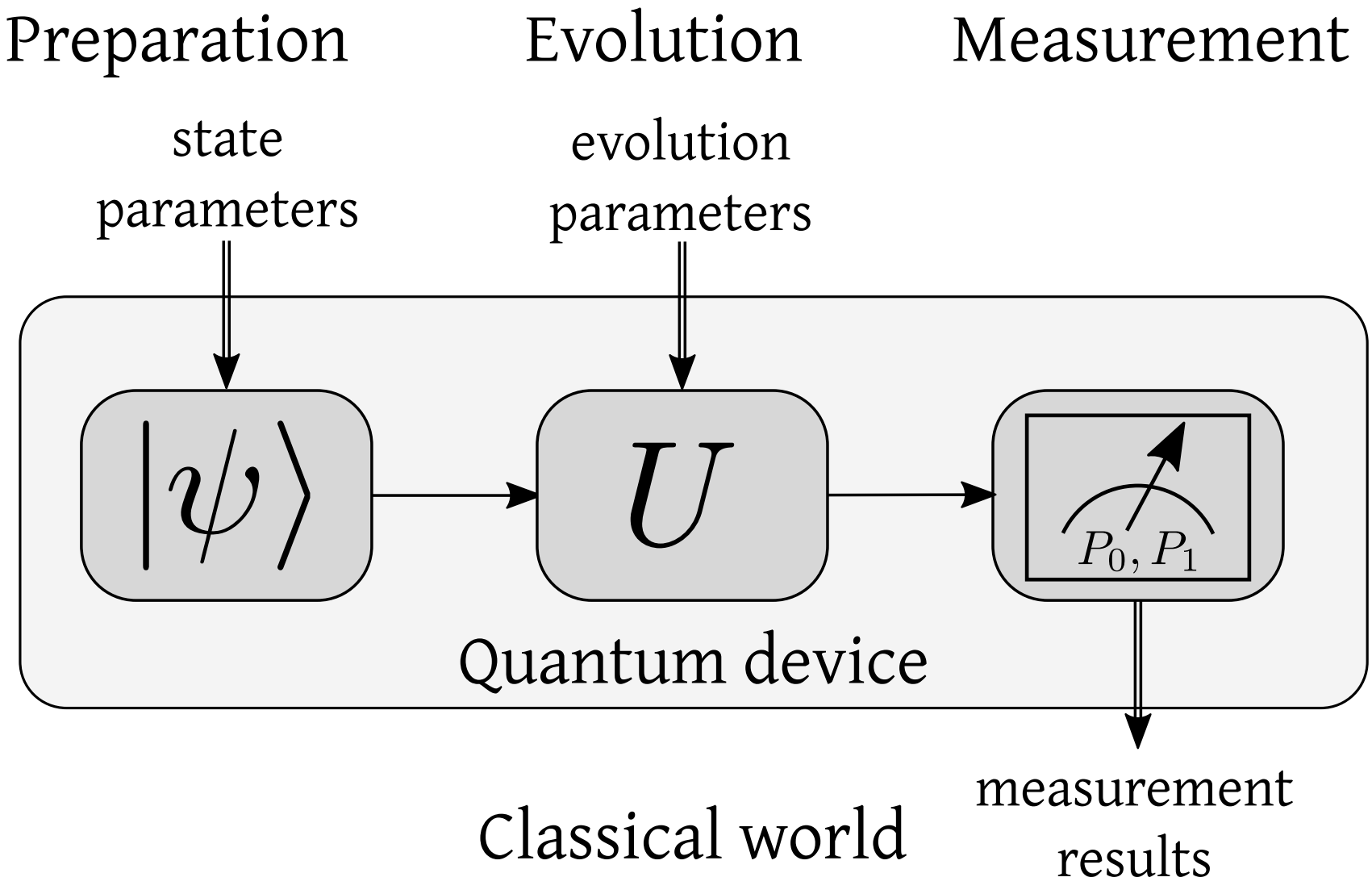
- Quantum computers – introduction
- Postulates of quantum mechanics
  - Quantum state
  - Evolution of quantum systems
  - Quantum measurement
  - Composition of quantum systems
- Quantum machine learning with quantum circuits
  - Idea of Quantum Neural Networks
  - Quantum neural networks
  - Example in python code
- Quantum APIs
- Summary
  - Conclusions
  - Bibliography

Outline for section 1

Quantum computation control loop



Computation as experiment



Outline for section 2

Quantum state I

- The **state** of quantum system:
  complex unit vector in n-dimensional Euclidean vector space $\mathbb{C}^n$.
- **Computational basis**:

$$|0\rangle = \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \; |n-1\rangle = \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Quantum state II

- '**Ket**' vector:

$$|x\rangle = x_0 \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} + \ldots + x_{n-1} \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

- '**Bra**' dual vector:

$$\langle x| = |x\rangle^\dagger = \begin{bmatrix} x_0^\star & \cdots & x_{n-1}^\star \end{bmatrix},$$

  $\square^\star$: complex conjugation.

- '**Braket**' scalar (inner) product:

$$\langle x|y\rangle .$$

- '**Ketbra**' outer product:

$$|x\rangle\langle y| .$$

Quantum state III

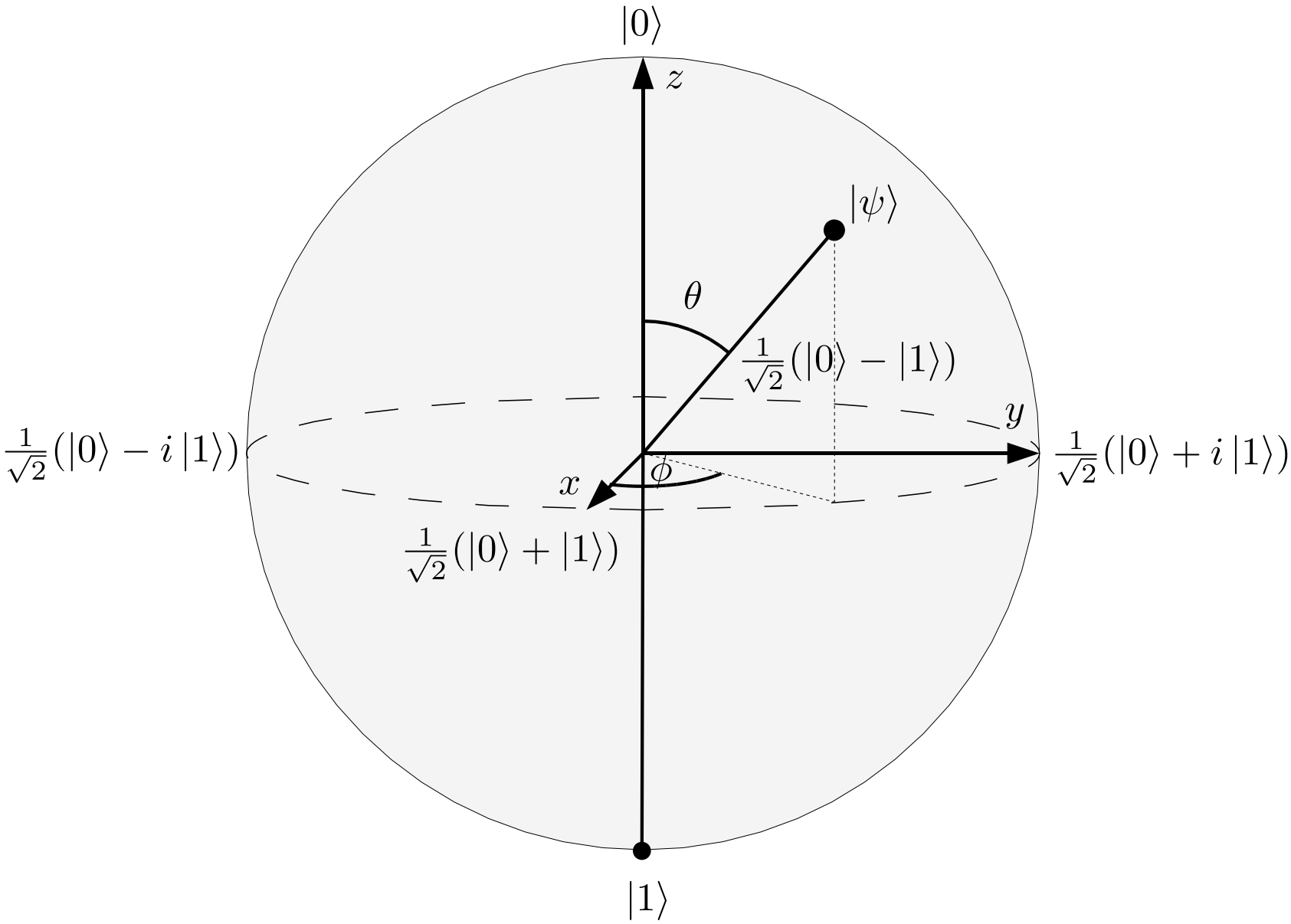
- A state $|\psi\rangle$ of a n-level quantum system is normalized **linear combination** of the basis vectors:

$$|\psi\rangle = \alpha_0 |0\rangle + \cdots + \alpha_{n-1} |n-1\rangle ,$$

where $\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{C}$ and $\sum_{i=0}^{n-1} |\alpha_i|^2 = 1$.

- Two vectors $|\psi\rangle$ and $|\phi\rangle$ **represent the same physical state** if $|\psi\rangle = e^{i\theta} |\phi\rangle$, for $\theta \in \mathbb{R}$.

Qubit, Bloch sphere



Time evolution of a quantum system I

- The evolution of quantum systems is governed by the **Schrödinger equation**

$$\frac{\mathrm{d}\,|\psi\rangle}{\mathrm{d}t} = -\mathrm{i}H\,|\psi\rangle\,,$$

where H is a hermitian operator *i.e.* $H = H^\dagger$ called **Hamiltonian** of the system.

Time evolution of a quantum system II

- Solution to Schrödinger equation:

$$|\psi_{t_1}\rangle = U(t_0, t_1)\,|\psi_{t_0}\rangle\,,$$

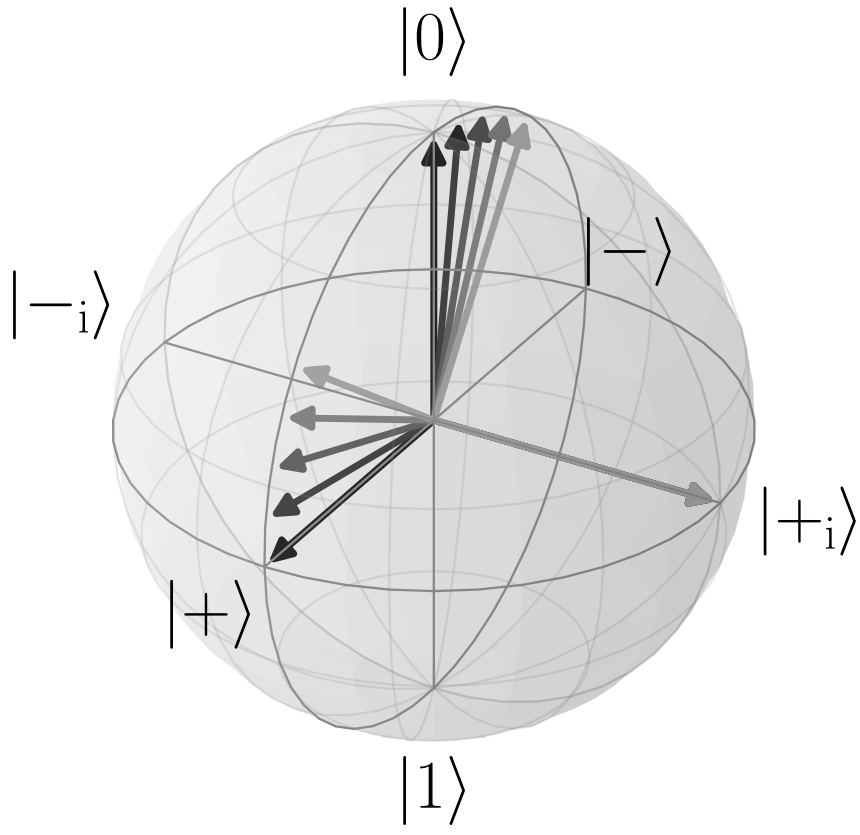
  - where $|\psi_{t_0}\rangle$ initial state of the system,
  - $|\psi_{t_1}\rangle$ final state of the system
  - $U(t_0, t_1)$ **unitary operator** driving the system from t_0 to t_1.
- An operator is unitary iff $UU^\dagger = U^\dagger U = \mathbb{1}$

Time evolution of a quantum system III

- If Hamiltonian H is constant between time t_0 and t_1:

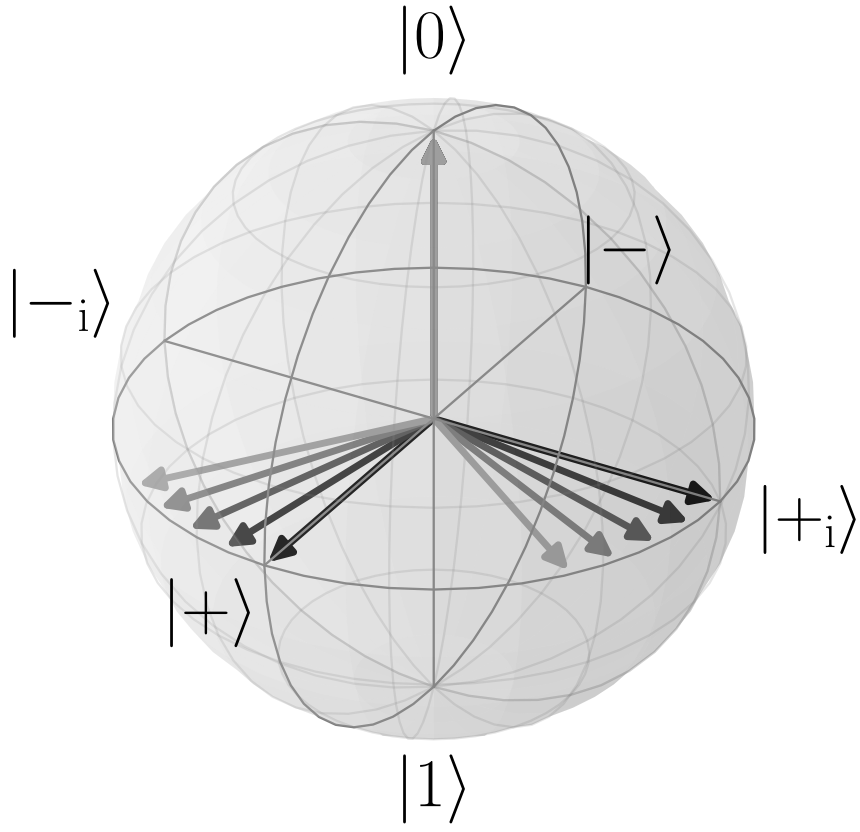
$$U(t_0, t_1) = e^{-\mathrm{i}(t_1 - t_0)H}.$$

Unitary gates, quantum evolution



Rotation around y-axis

Unitary gates, quantum evolution



Rotation around z-axis

Measurement I

- **Quantum measurement**—a function μ:
  - from finite set of measurements outcomes $A = \{a_i\}_{i=1}^n$
  - to set the of projection operators $P = \{P_i\}_{i=1}^n$
  - such that

$$\sum_{i=1}^{n} P_i = \mathbb{1} \text{ and } P_i^2 = P_i.$$

Measurement II

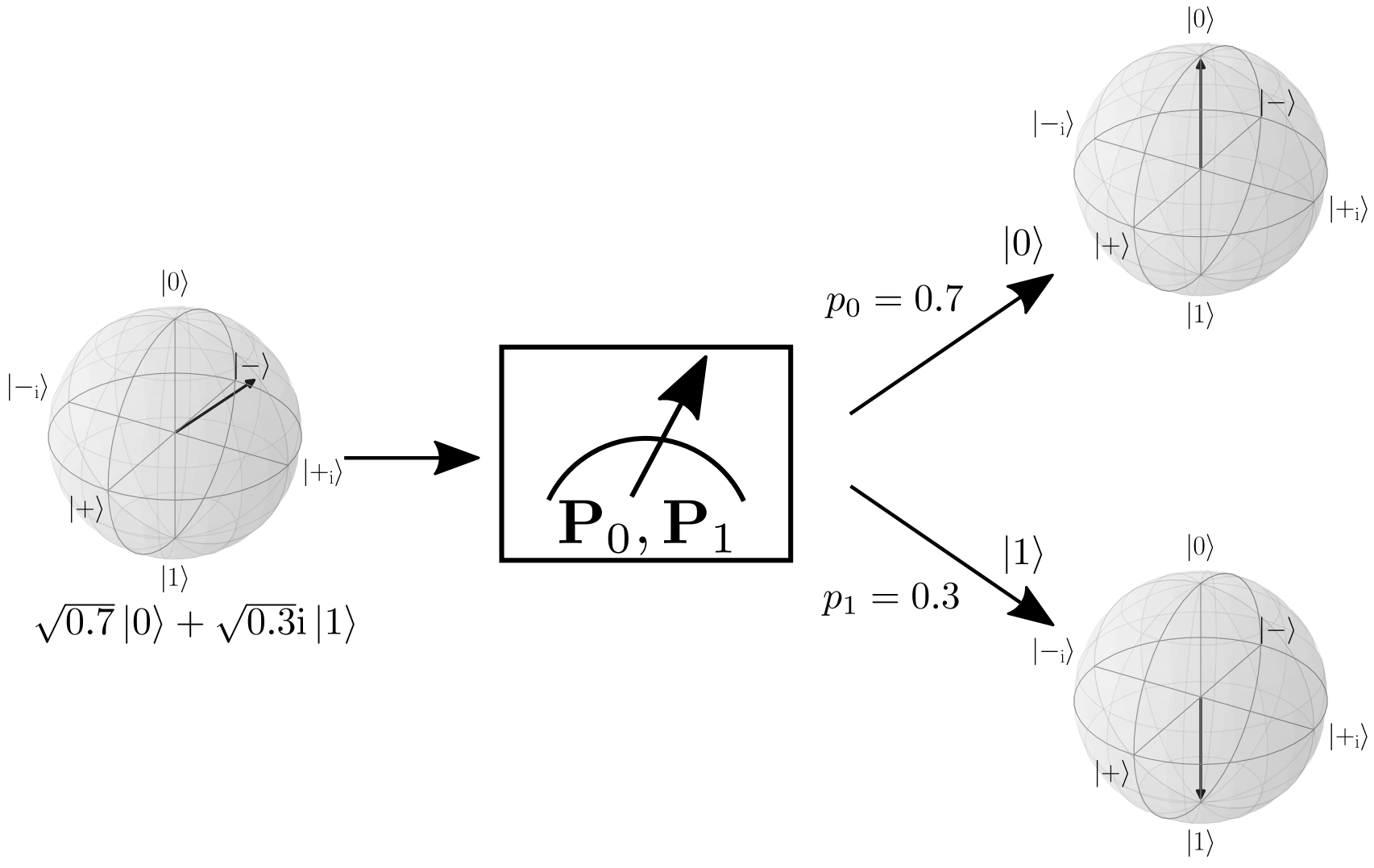
- The **probability of measuring outcome** a_i given the quantum system in in state $|\psi\rangle$ is

$$p(a_i) = \langle\psi| P_i |\psi\rangle .$$

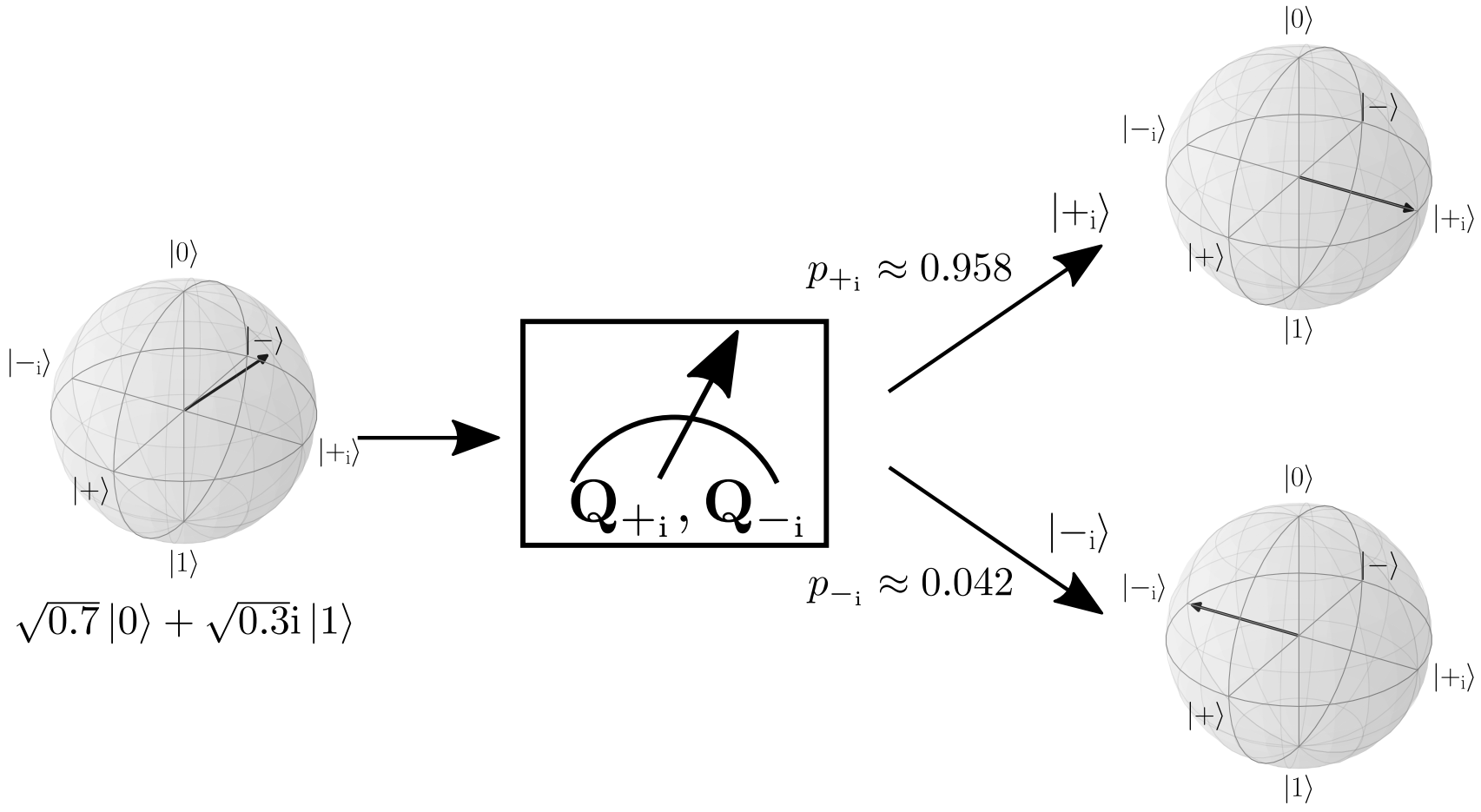
The state of the quantum system **after the measurement** outcome a_i was obtained becomes

$$|\psi\rangle_{a_i} = \frac{P_i |\psi\rangle}{\sqrt{\langle\psi| P_i |\psi\rangle}}.$$

Measurement



Measurement



Composition of quantum systems

Tensor product for **two qubit** states

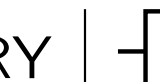
$$|\psi\rangle = \begin{bmatrix} \psi_0 \\ \psi_1 \end{bmatrix} = \psi_0 |0\rangle + \psi_1 |1\rangle ,$$

$$|\phi\rangle = \begin{bmatrix} \phi_0 \\ \phi_1 \end{bmatrix} = \phi_0 |0\rangle + \phi_1 |1\rangle , \tag{1}$$

their joint state in $\mathbb{C}^2 \otimes \mathbb{C}^2$

$$|\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} \psi_0\phi_0 \\ \psi_0\phi_1 \\ \psi_1\phi_0 \\ \psi_1\phi_1 \end{bmatrix} . \tag{2}$$

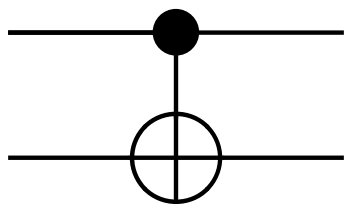
Gate model of quantum computation I

UNITARY  $U|\psi_1\rangle = |\psi_2\rangle,\ U^\dagger|\psi_2\rangle = |\psi_1\rangle$

$$U = e^{i\varphi/2}\begin{bmatrix} e^{i\psi} & 0 \\ 0 & e^{-i\psi} \end{bmatrix}\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} e^{i\Delta} & 0 \\ 0 & e^{-i\Delta} \end{bmatrix}$$

Gate model of quantum computation II

CNOT

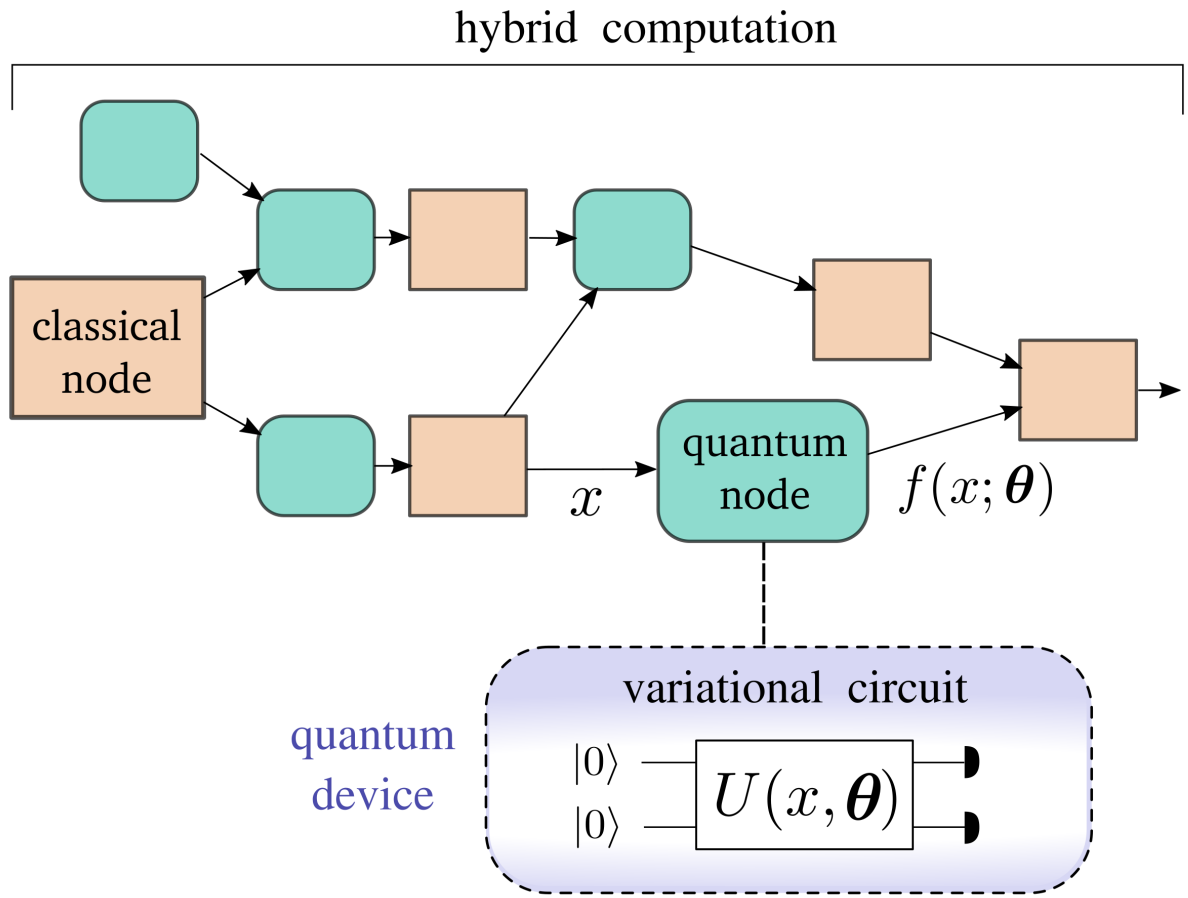


in_1	in_2	out_1	out_2
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Outline for section 3

Quantum neural networks I

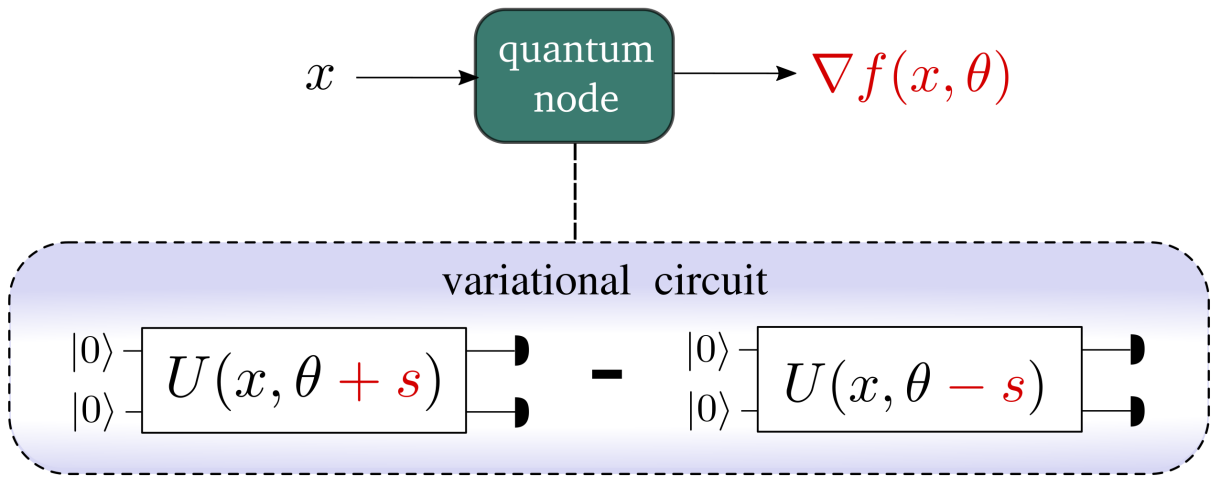
Concept of hybrid computation



Source: `https://github.com/XanaduAI/pennylane/` (Distributed under Apache Licence 2.0)

Quantum neural networks II

Gradient



Source: `https://github.com/XanaduAI/pennylane/` (Distributed under Apache Licence 2.0)

Observable, expectation value

- Let O be an **observable**—a quantum measurement whose labels are real numbers.
- Mathematically it is represented as a Hermitian operator i.e.

$$O^\dagger = O.$$

- The **expectation value** of an observable O in a state $|\psi\rangle$

$$\langle O \rangle_{|\psi\rangle} = \langle \psi | \, O \, | \psi \rangle \, .$$

- For example if we observe spin of a system, expectation value tells us about the average spin of this system in a given state.

Gradients, parameter shift rule

- A function

$$f(\theta) = \langle O \rangle_{U(\theta)|\psi\rangle}$$

if $U(\theta) = e^{-i\theta G}$, where G has two distinctive eigenvalues $\{-r, r\}$ has gradient

$$\frac{\partial}{\partial\theta} f = r\left[f(\theta + s) - f(\theta - s)\right],$$

where $s = \pi/4r$.

- For example for $R_x(\theta) = e^{-\theta i \sigma_x/2}$, and $f(\theta) = \langle O \rangle_{R_x(\theta)|\psi\rangle}$ the gradient is

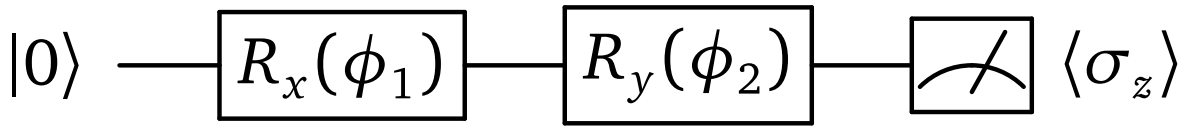
$$\frac{\partial}{\partial\theta} f(\theta) = \frac{1}{2}\left[f(\theta + \pi/2) - f(\theta - \pi/2)\right].$$

Optimizers

- Gradient descent: $\theta^{(t+1)} = \theta^{(t)} - \eta \nabla f(\theta^{(t)})$;
- Nesterov momentum: $\theta^{(t+1)} = \theta^{(t)} - a^{(t+1)}$, $a^{(t+1)} = m a^{(t)} + \eta \nabla f(\theta^{(t)} - m a^{(t)})$;
- Adam;
- Adagrad.

With η — the step size, m — the momentum.

Variational quantum algorithm I




```
1  import pennylane as qml
2  from pennylane.optimize import GradientDescentOptimizer
3  # Create device
4  dev = qml.device('default.qubit', wires=1)
5  # Quantum node
6  @qml.qnode(dev)
7  def circuit1(var):
8      qml.RX(var[0], wires=0)
9      qml.RY(var[1], wires=0)
10     return qml.expval(qml.PauliZ(0))
11 # Create optimizer
12 opt = GradientDescentOptimizer(0.25)
13 # Optimize circuit output
14 var = [0.5, 0.2]
15 for it in range(30):
16     var = opt.step(circuit1, var)
17     print("Step {}: cost: {}".format(it, circuit1(var)))
```

Source: arXiv:1811.04968

Variational quantum algorithm II

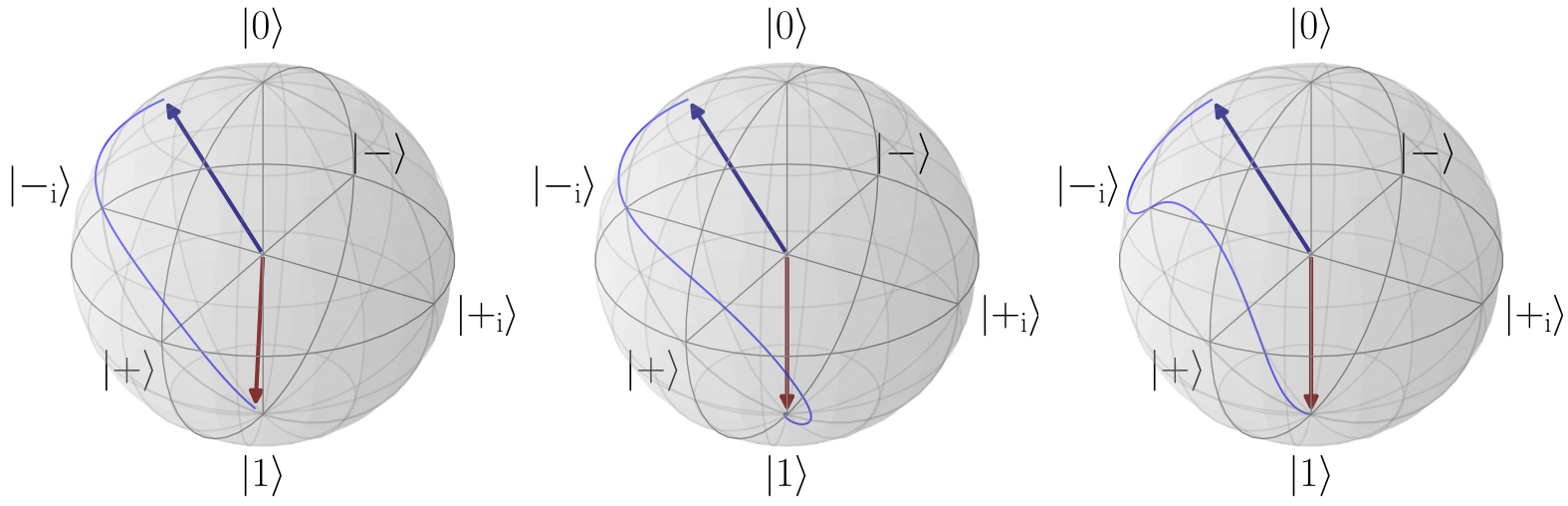


Figure: Three optimization strategies: gradient descent, Nesterov momentum optimizer, Adam optimizer ($var_{\mathrm{init}} = (0.5, 0.2)$).

An example of quantum neural network

Dataset

We use an artificial dataset (moons) consisting of two classes.

Data preprocessing

- The original feature vectors (single feature for all samples) X_{orig} are normalized

$$X_{\text{std}} = \frac{X_{\text{orig}} - \min(X_{\text{orig}})}{\max(X_{\text{orig}}) - min(X_{\text{orig}})}$$

$$X_{\text{scaled}} = X_{\text{std}}(\max - \min) + \min,$$

  with $\min = 0$, $\max = \pi$.

- In order to encode as in angle of a qubit rotation.

An example of quantum neural network

Data encoding gate

- Encoding gate $U_{\mathrm{e}}(x)$ transforms data sample $x = (x_1, x_2, \ldots, x_N) \in \mathbb{R}^N$ into data state

$$|x\rangle = U_{\mathrm{e}}(x)\,|0\rangle^{\otimes N} = R_x(x_1) \otimes R_x(x_2) \otimes \ldots \otimes R_x(x_N)\,|0\rangle^{\otimes N},$$

- where

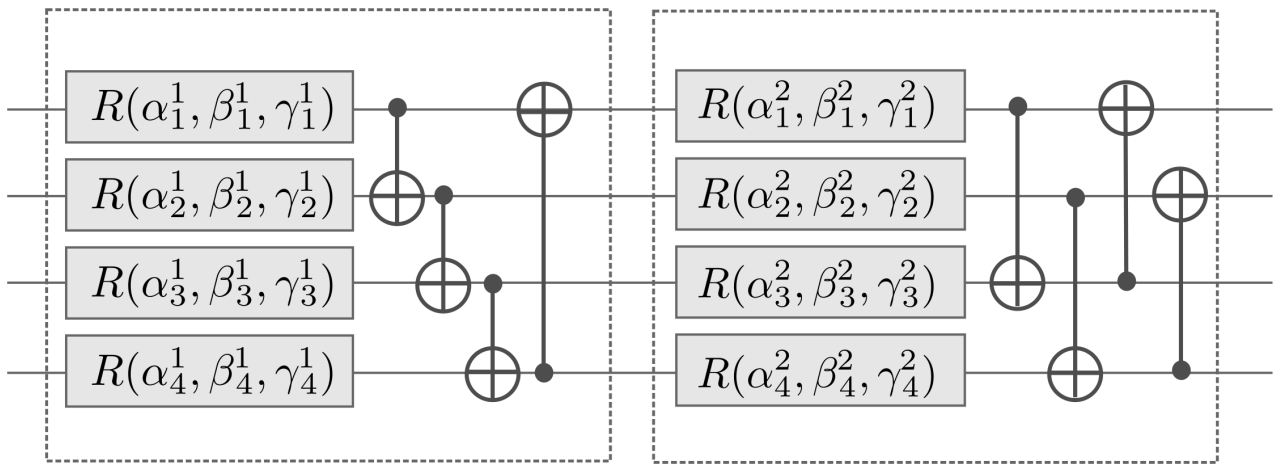
$$R_x(\phi) = e^{-i\phi\sigma_x/2} = \begin{bmatrix} \cos(\phi/2) & -i\sin(\phi/2) \\ -i\sin(\phi/2) & \cos(\phi/2) \end{bmatrix}$$

- and $x_i \in [0, \pi]$.

An example of quantum neural network

Variational gate

- Variational gate $U_{\mathrm{v}}(w)$ entangles the information and is parametrized. Angles $w \in \mathbb{R}^{N \times 3 \times L}$ are the parameters we want learn from the data and L denotes the number of quantum layers.



Source: https://github.com/XanaduAI/pennylane/ (Distributed under Apache Licence 2.0)

An example of quantum neural network

Decision function

- The decision function $f : \mathbb{R}^N \to \mathbb{R}$.
- $f(x;\theta) = \langle O \rangle_{U_{\mathrm{v}}(\theta)U_{\mathrm{e}}(x)|0\rangle}$, with

$$\langle O \rangle_{U_{\mathrm{v}}(\theta)U_{\mathrm{e}}(x)|0\rangle} = \langle 0| U_{\mathrm{e}}(x)^\dagger U_{\mathrm{v}}(\theta)^\dagger O U_{\mathrm{v}}(w) U_{\mathrm{e}}(x) |0\rangle ,$$

- Observable $O = \sigma_z \otimes \mathbb{1}^{\otimes(N-1)}$.
- $\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$, eigenpairs: $(1, |0\rangle), (-1, |1\rangle)$.
- The classification function
  $\mathrm{cls} : \mathbb{R}^N \to \{-1, 1\}$: $\mathrm{cls}(x;\theta) = \mathrm{sgn}(f(x;\theta))$.

An example of quantum neural network

Training

- The available data are divided into: train and test.
- Initial θ-s: $w \sim \mathcal{U}(0, 2\pi)$.
- Cost function

$$\text{cost}\left((y_i^{\text{train}})_{i\in\xi_t}, (f(X_{i,:}^{\text{train}};\theta))_{i\in\xi_t}\right) = \sum_{i\in\xi_t}(y_i^{\text{train}} - f(X_{i,:}^{\text{train}};\theta))^2/|\xi_t|.$$

- Batches ξ_t.

Binary classification in pennylane

Imports

```
1  from itertools import chain
2
3  from sklearn import datasets
4  from sklearn.utils import shuffle
5  from sklearn.preprocessing import minmax_scale
6  from sklearn.model_selection import train_test_split
7  import sklearn.metrics as metrics
8
9
10 import pennylane as qml
11 from pennylane import numpy as np
12 from pennylane.templates.embeddings import AngleEmbedding
13 from pennylane.templates.layers import StronglyEntanglingLayers
14 from pennylane.init import strong_ent_layers_uniform
```

Binary classification in pennylane

Data import, preprocessing and splitting

```
17  np.random.seed(42)
18
19  # create the dataset
20  X, y = datasets.make_moons(100, noise=0.1, random_state=42)
21
22  # shuffle the data
23  X, y = shuffle(X, y, random_state=42)
24
25
26  # normalize data
27  X = minmax_scale(X, feature_range=(0, np.pi))
28
29  # split data into train and test
30  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Binary classification in pennylane

Building the quantum classifier

```
32  # number of qubits is equal to the number of features
33  n_qubits = X.shape[1]
34
35  # quantum device handle
36  dev = qml.device("default.qubit", wires=n_qubits)
37
38  # quantum circuit
39  @qml.qnode(dev)
40  def circuit(weights, x=None):
41      AngleEmbedding(x, wires = range(n_qubits))
42      StronglyEntanglingLayers(weights, wires = range(n_qubits))
43      return qml.expval(qml.PauliZ(0))
44
45  def cost(theta, X, expectations):
46      e_predicted = \
47          np.array([circuit(theta, x=x) for x in X])
48
49      loss = np.mean((e_predicted - expectations)**2)
50      return loss
51
52  # helper function: returns 1 if x>0 else 0
53  def sgn(x):
54      return (x>0)*1
```

Binary classification in pennylane

Training preparation

```
56  # number of quantum layers
57  n_layers = 3
58
59  # convert classes to expectations: 0 to -1, 1 to +1
60  e_train = np.empty_like(y_train)
61  e_train[y_train == 0] = -1
62  e_train[y_train == 1] = +1
63
64  # select learning batch size
65  batch_size = 5
66
67  # calculate number of batches
68  batches = len(X_train) // batch_size
69
70  # select number of epochs
71  n_epochs = 5
72
73  # draw random quantum node weights
74  theta = strong_ent_layers_uniform(n_layers, n_qubits, seed=15)
```

Binary classification in pennylane

Training

```
76  # train the variational classifier
77
78  # start of main learning loop
79  # build the optimizer object
80  pennylane_opt = NesterovMomentumOptimizer()
81
82  log = []
83  # split training data into batches
84  X_batches = np.array_split(np.arange(len(X_train)), batches)
85  for it, batch_index in enumerate(chain(*(n_epochs * [X_batches]))):
86      # Update the weights by one optimizer step
87      batch_cost = \
88          lambda t: cost(t, X_train[batch_index], e_train[batch_index])
89      theta = pennylane_opt.step(batch_cost, theta)
90      log.append({"theta":theta})
91  # end of learning loop
```

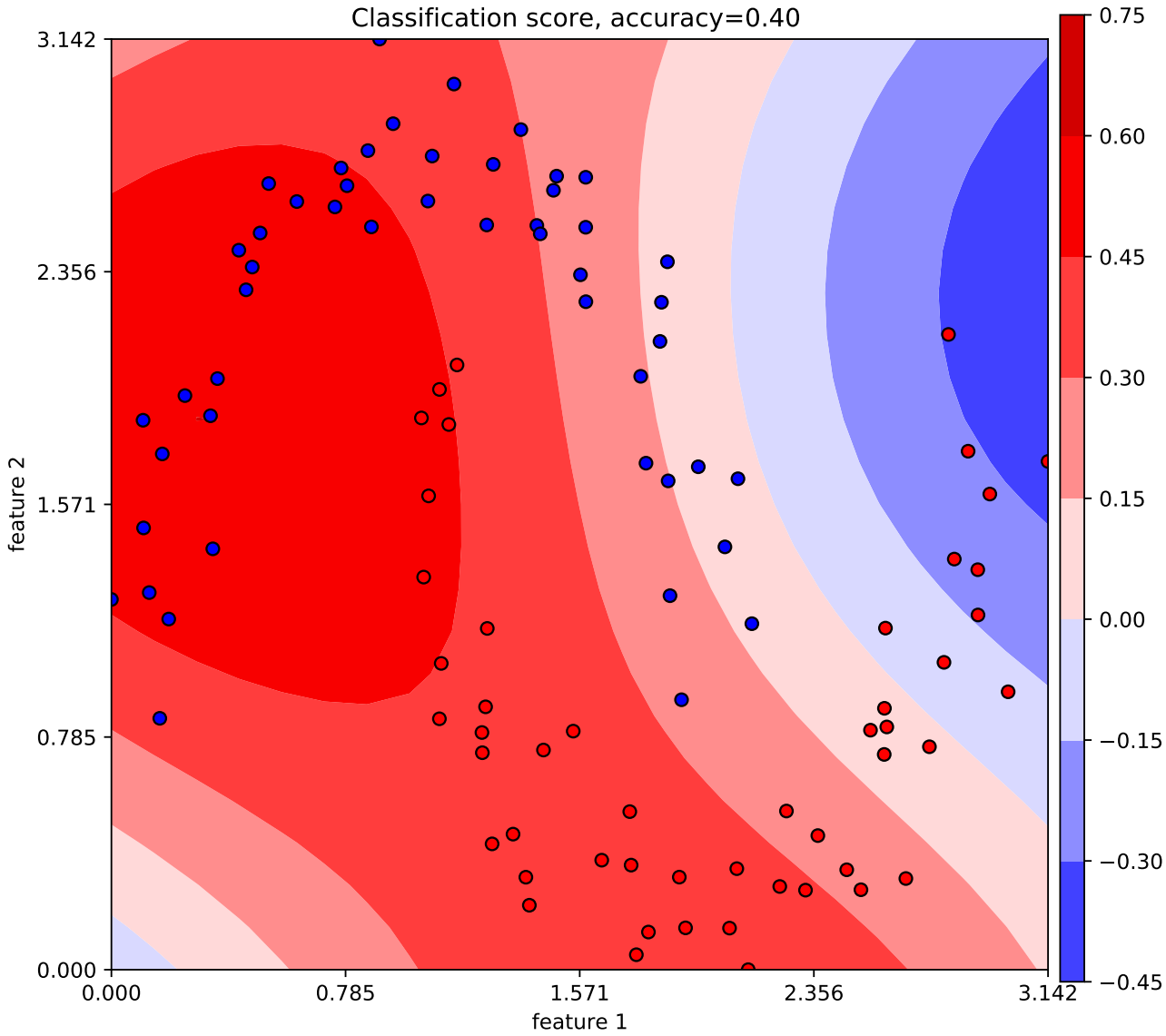
Binary classification in pennylane

Inference

```
93  # convert scores to classes
94  scores = np.array([circuit(theta, x=x) for x in X_test])
95  y_pred = sgn(scores)
96
97  print(metrics.accuracy_score(y_test, y_pred))
98  print(metrics.confusion_matrix(y_test, y_pred))
```

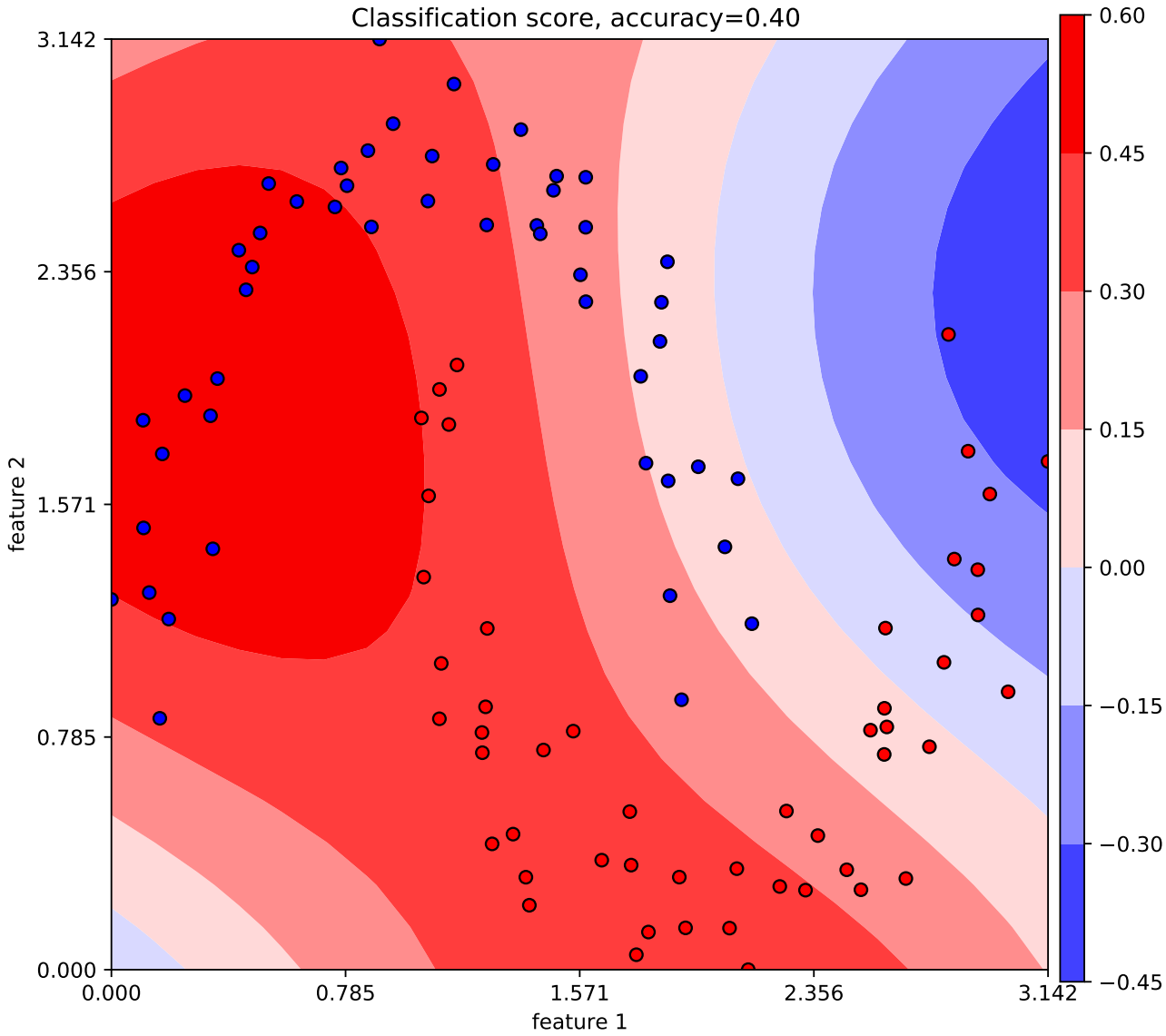
Learning process

Learning step 1



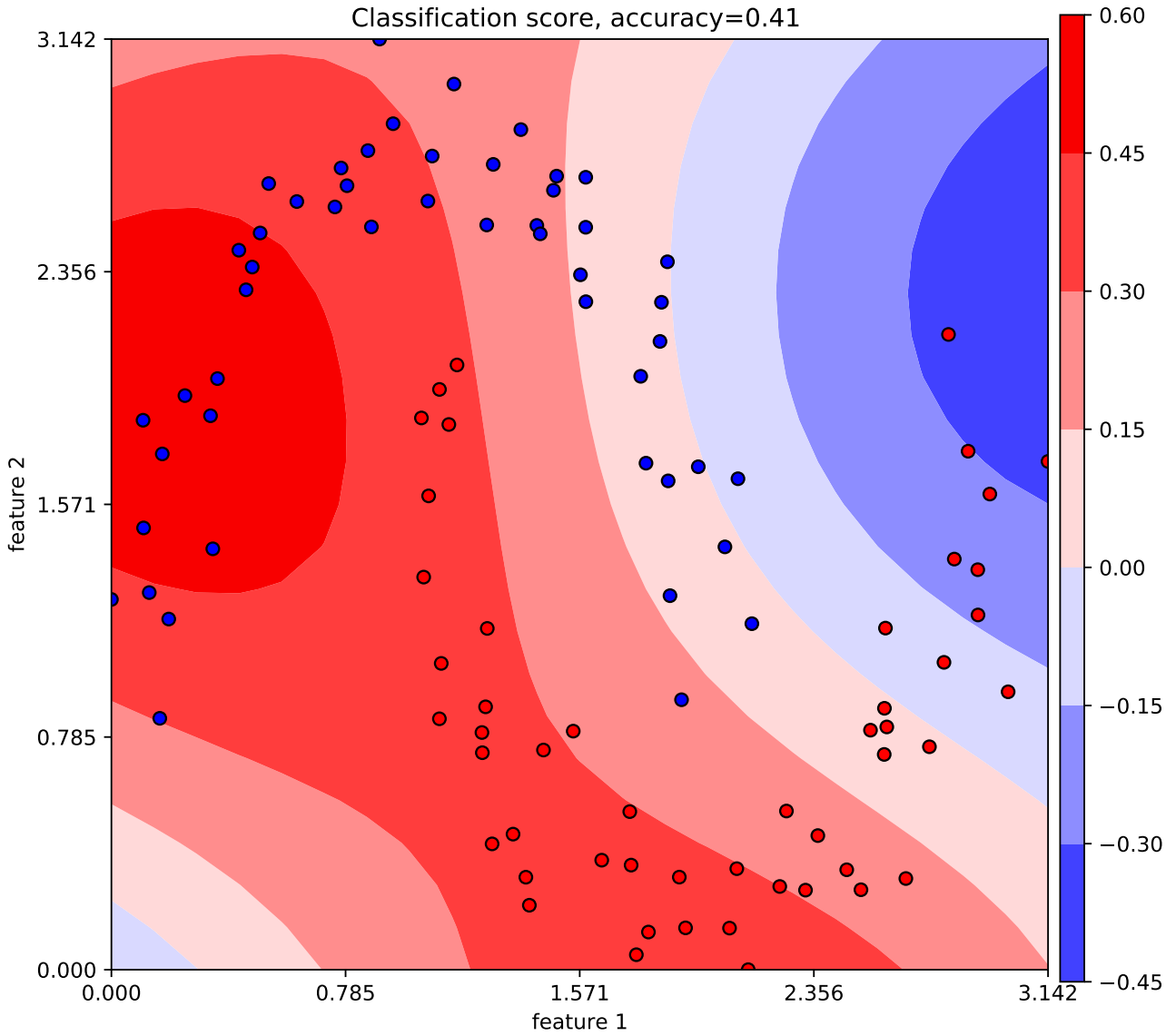
Learning process

Learning step 2



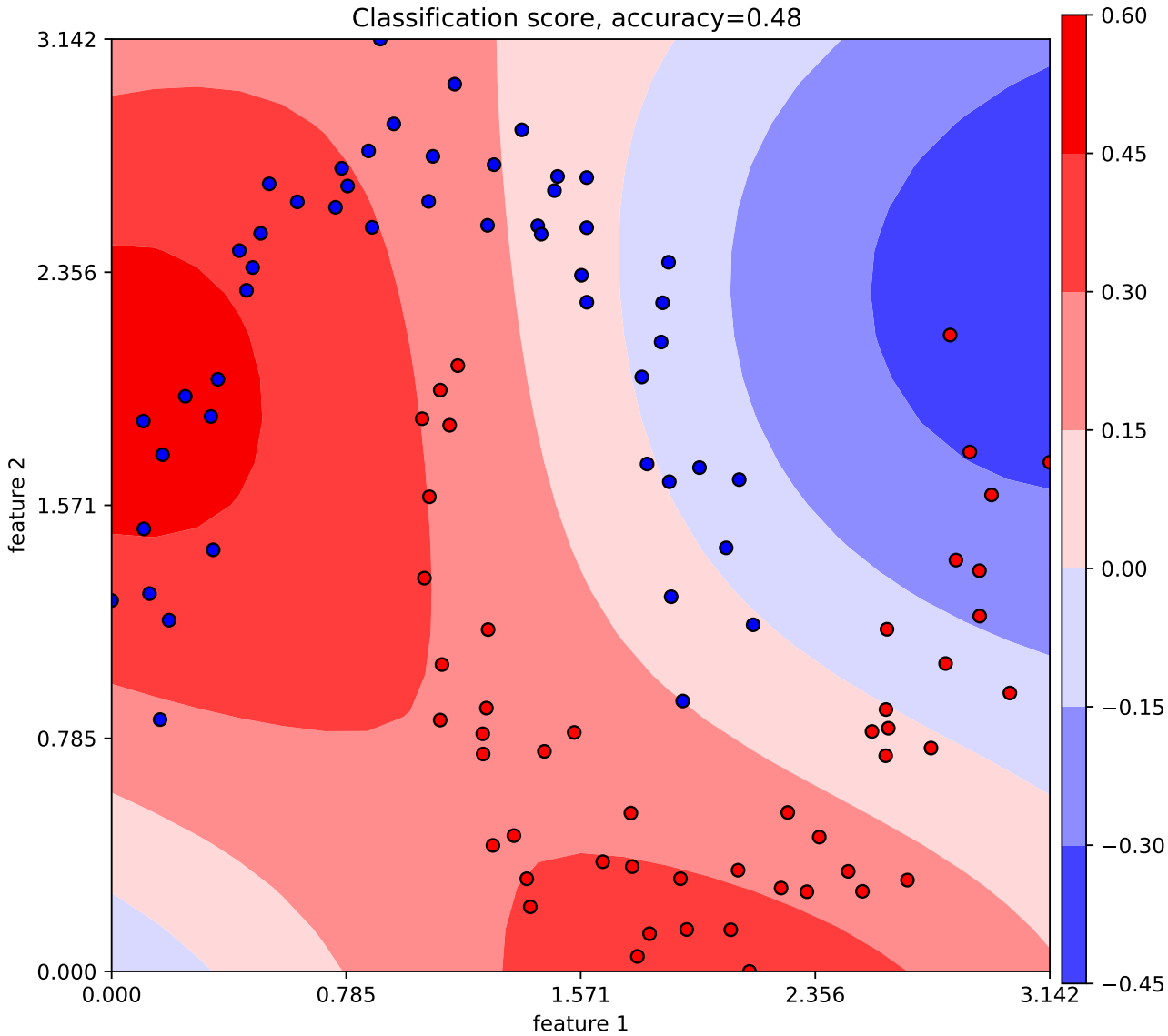
Learning process

Learning step 3



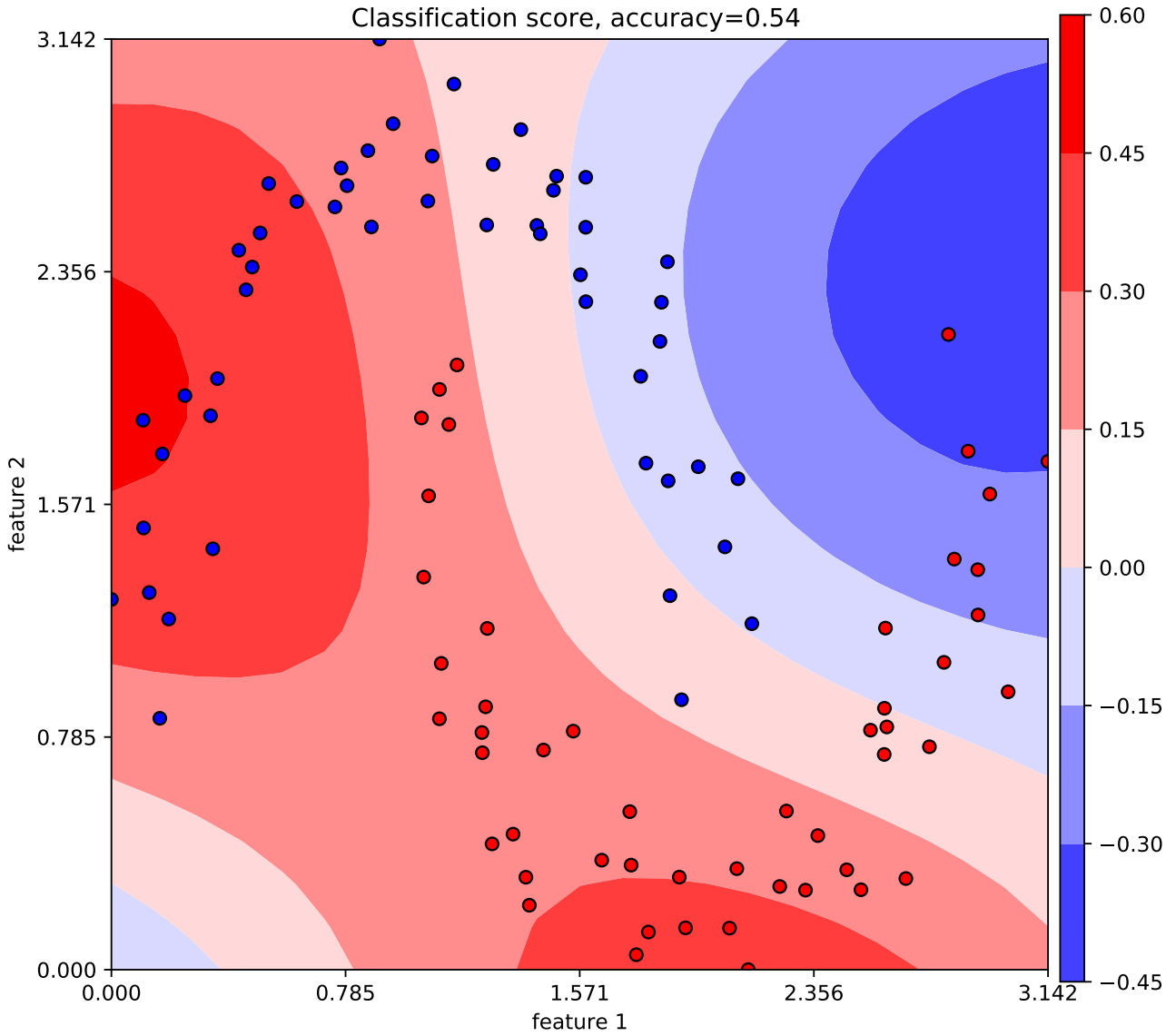
Learning process

Learning step 4



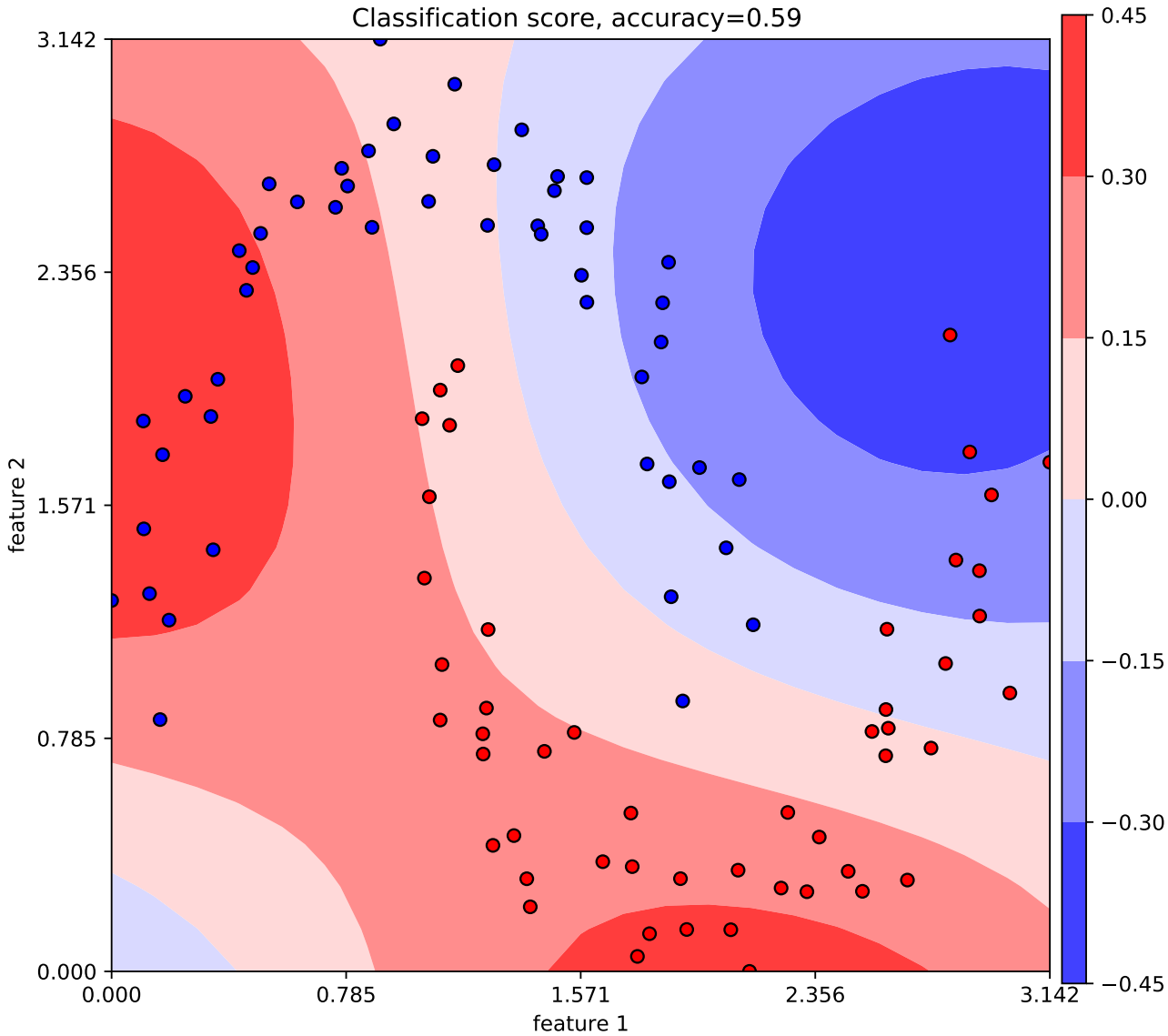
Learning process

Learning step 5



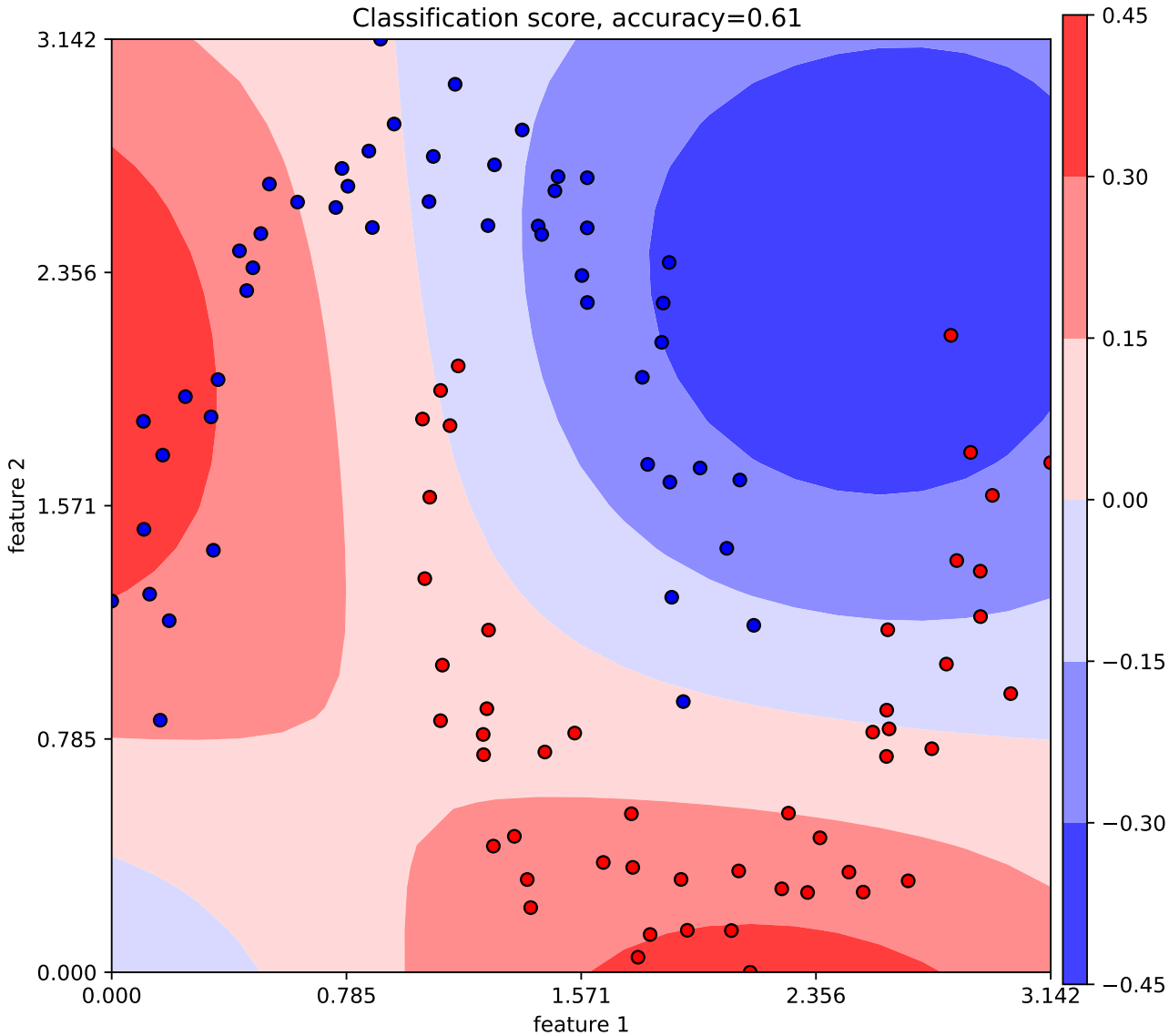
Learning process

Learning step 6



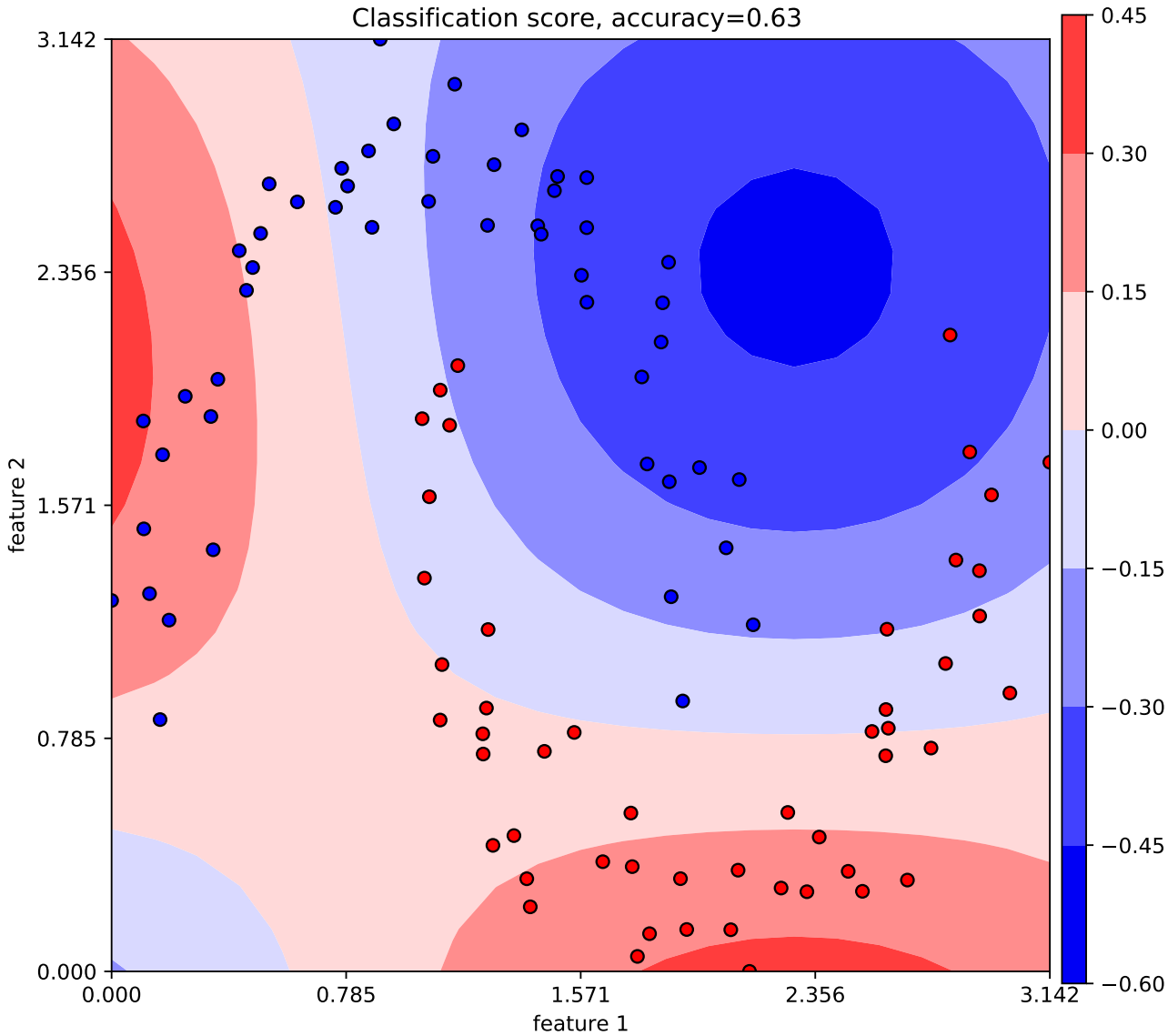
Learning process

Learning step 7



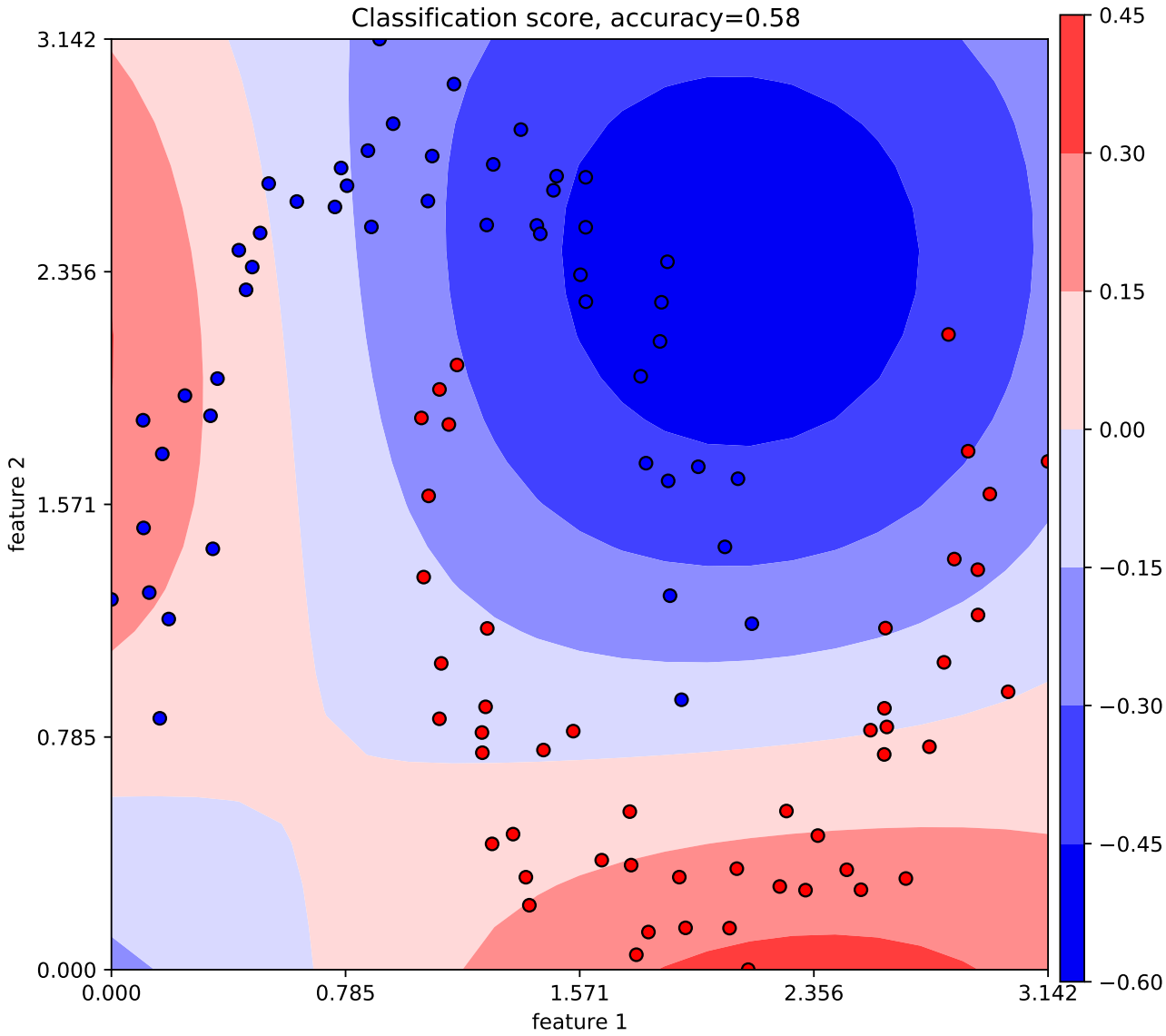
Learning process

Learning step 8



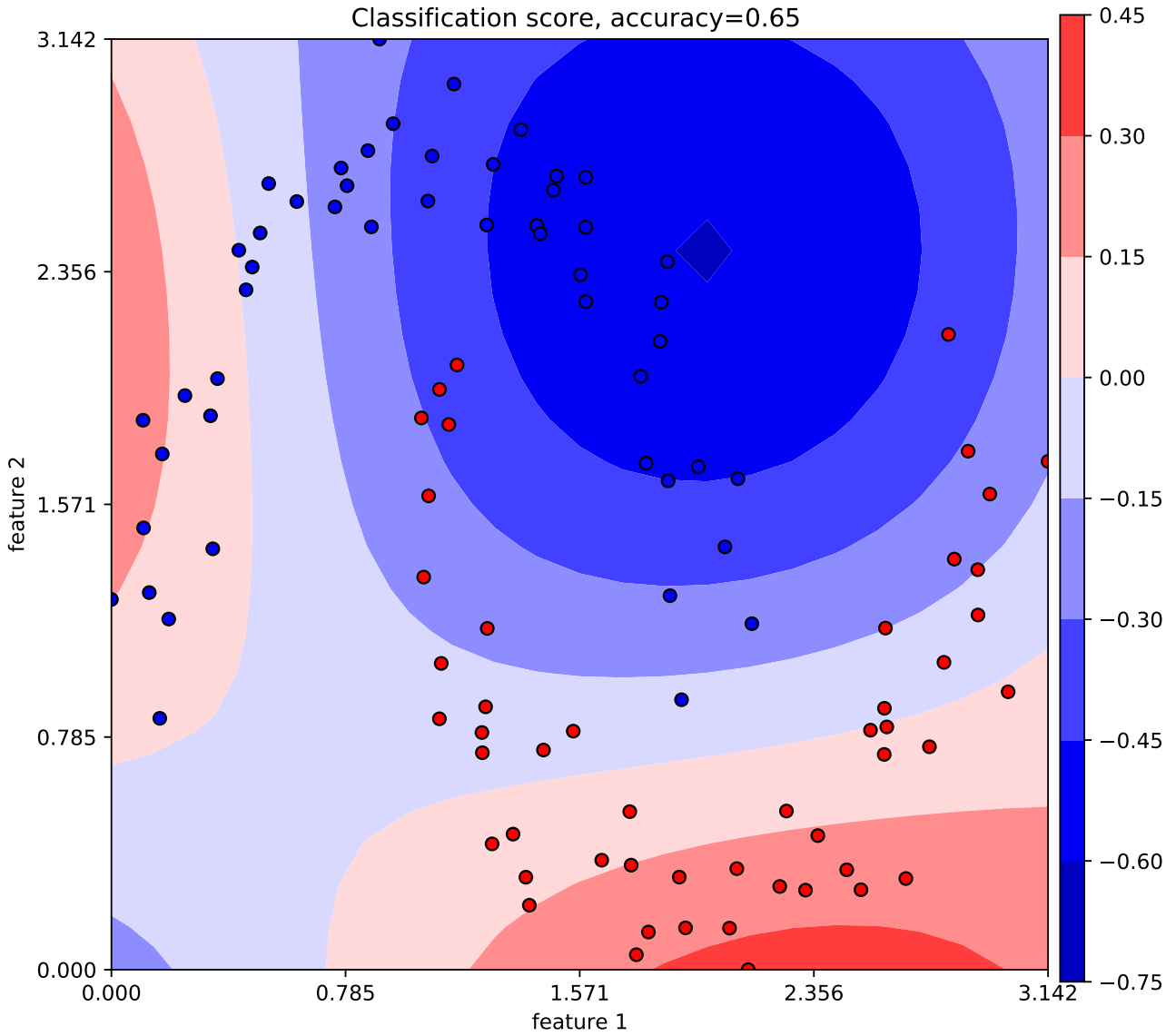
Learning process

Learning step 9



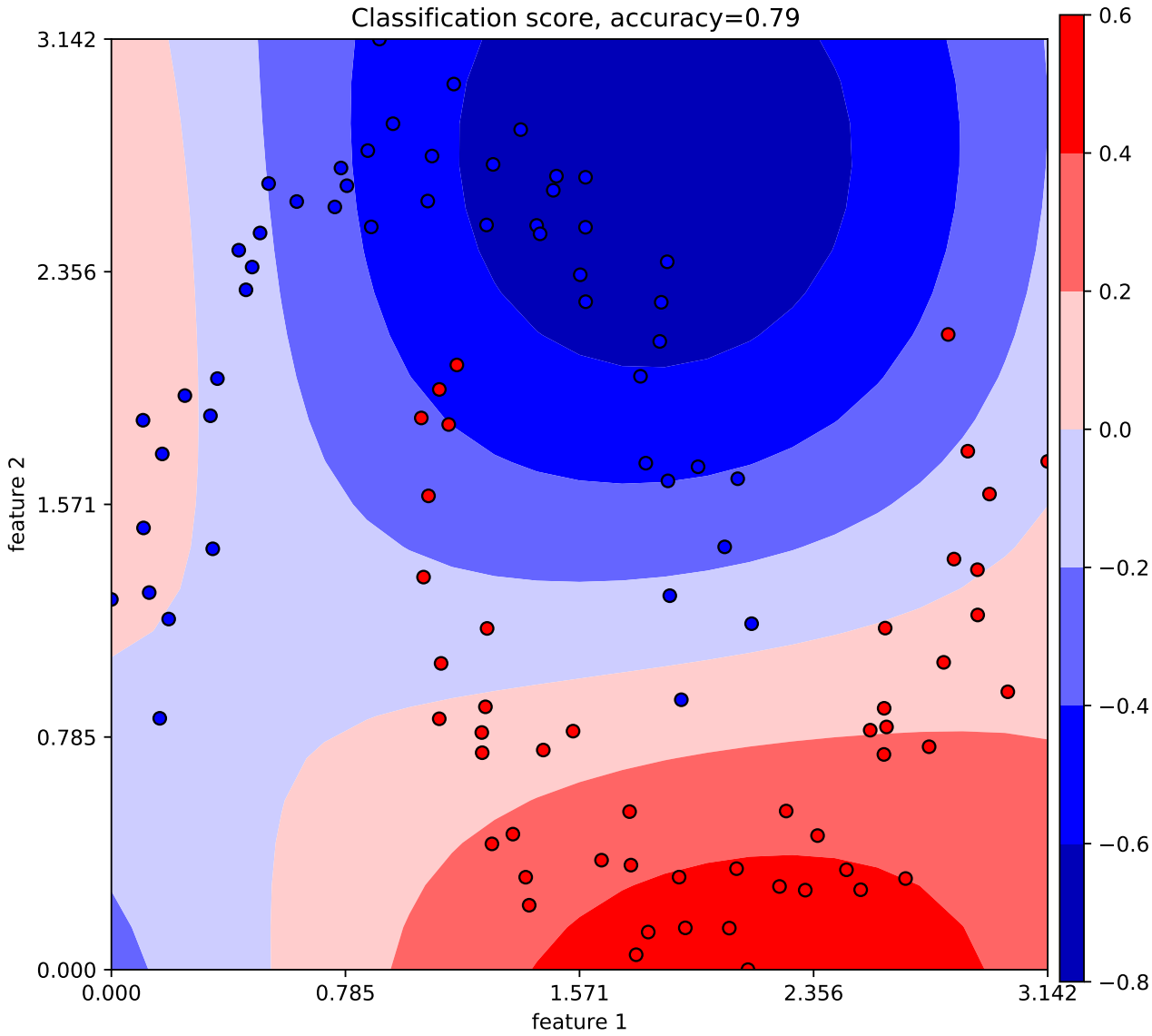
Learning process

Learning step 10



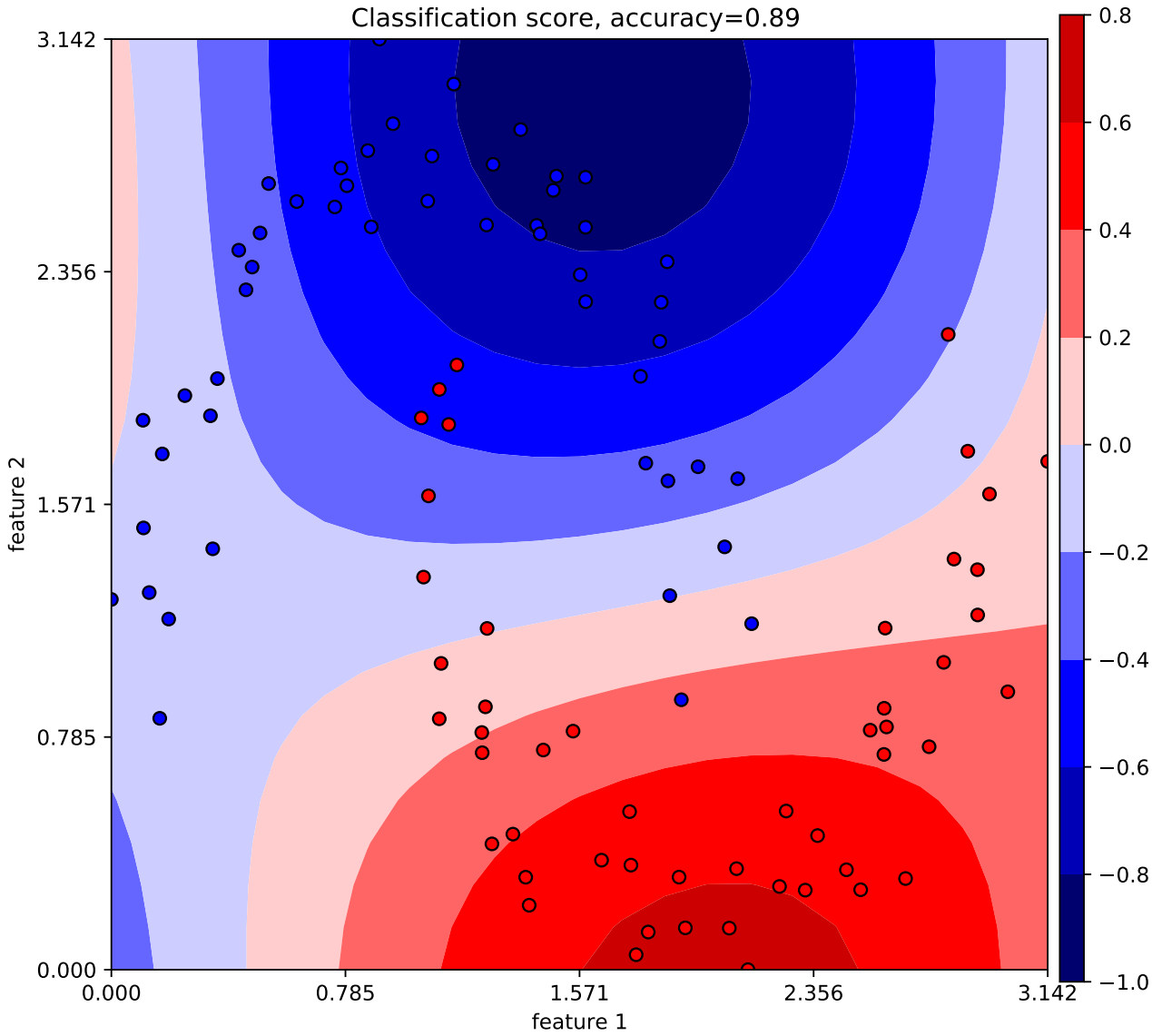
Learning process

Learning step 15



Learning process

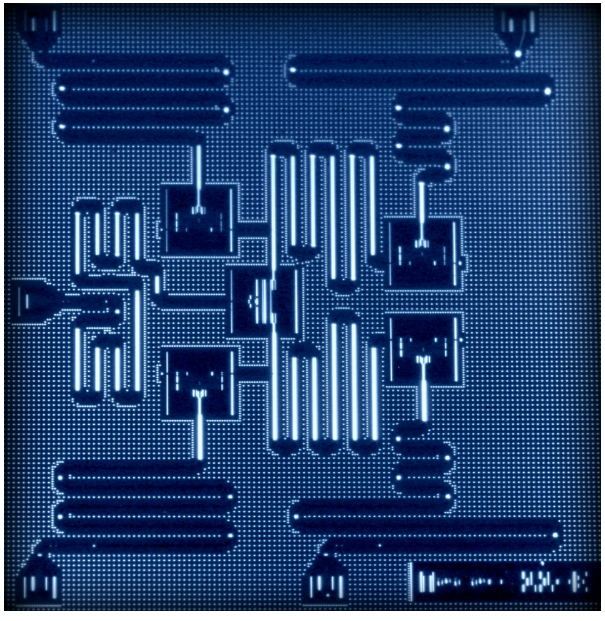
Learning step 20



Outline for section 4

IBM's Qiskit

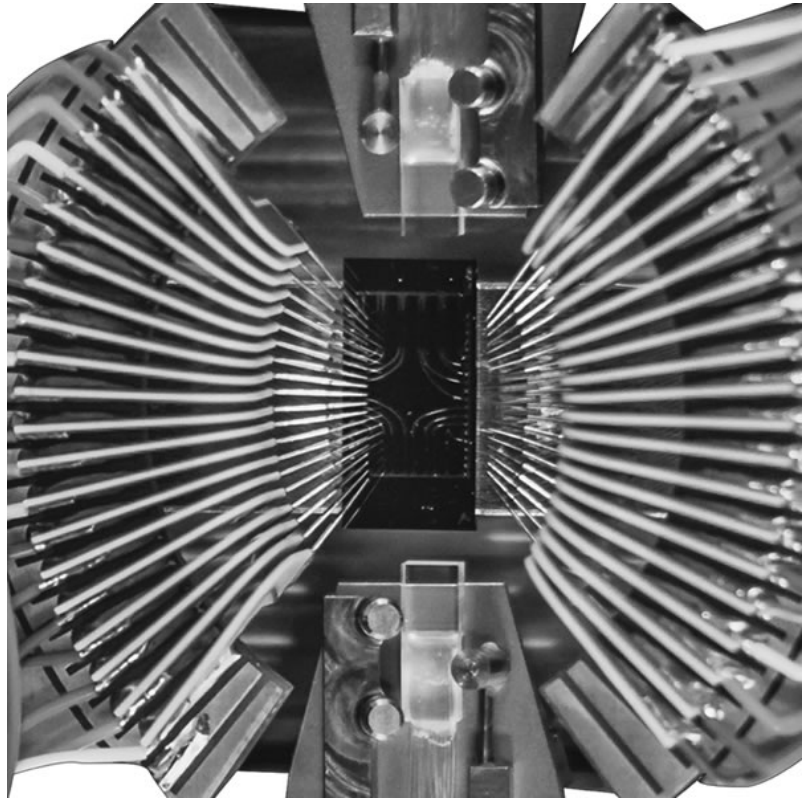
- Terra—the code foundation, for composing quantum programs at the level of circuits and pulses,
- Aqua—for building algorithms and applications,
- Ignis—for addressing noise and errors, and
- Aer—for accelerating development via simulators, emulators and debuggers.



`https://qiskit.org/`

Xanadu's Pennylane

- StrawberryFields—a toolkit for writing program for optical devices.
- Pennylane—Quantum neural networks library.



`https://xanadu.ai/`

Outline for section 5

Conclusions

What you have learned today

- What are quantum states and operations.
- How quantum variational algorithms work.
- What is a quantum neural network.
- How to program a quantum machine learning system.

What you did not learn today

- What are barren plateaus.
- What is the expressive power of QNNs models.
- What is the power of QNNs models.
- What are various QNNs architectures.

Bbibliography I


J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd.
Quantum machine learning.
Nature, 549(7671):195, 2017.

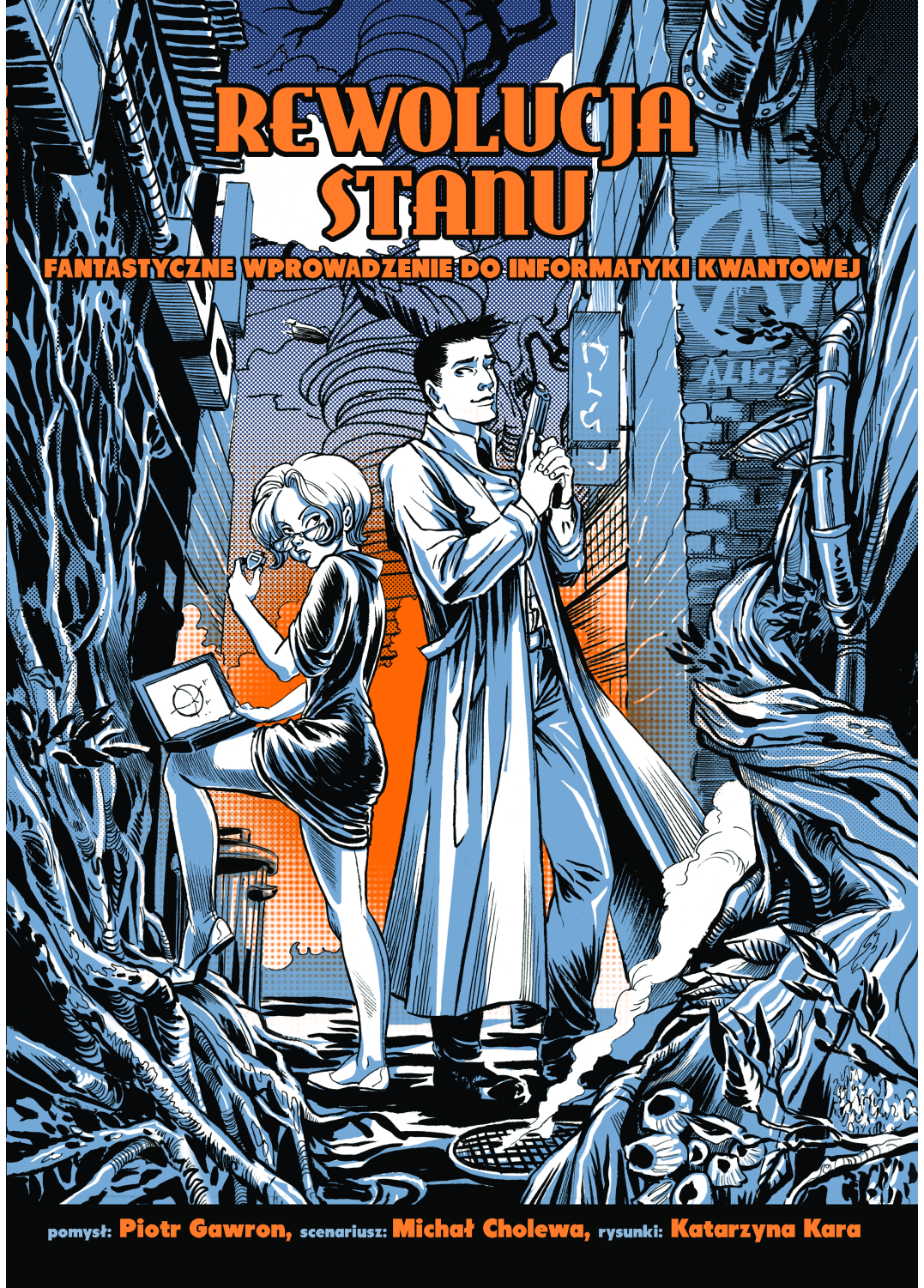

M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran.
Evaluating analytic gradients on quantum hardware.
Physical Review A, 99(3):032331, 2019.


M. Schuld, A. Bocharov, K. M. Svore, and N. Wiebe.
Circuit-centric quantum classifiers.
Physical Review A, 101:032308, Mar 2020.


M. Schuld, M. Fingerhuth, and F. Petruccione.
Implementing a distance-based classifier with a quantum interference circuit.
EPL (Europhysics Letters), 119(6):60002, 2017.

Bbibliography II


M. Schuld and F. Petruccione.
Supervised Learning with Quantum Computers (Quantum Science and Technology).
Springer, 2018.


REWOLUCJA STANU
FANTASTYCZNE WPROWADZENIE DO INFORMATYKI KWANTOWEJ
ALICE
pomysł: Piotr Gawron, scenariusz: Michał Cholewa, rysunki: Katarzyna Kara

Thank you for your attention!

Piotr Gawron

@pwgawron

gawron@camk.edu.pl

https://pgawron.github.io/

https://pl.linkedin.com/in/gawron

I am looking for post-docs!

Send me an e-mail if interested!

"Rewolucja stanu" is available at

https://depot.ceon.pl/handle/123456789/16807

AstroCeNT

European Funds Smart Growth

Republic of Poland

FNP Foundation for Polish Science

European Union European Regional Development Fund

CAMK PAN