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 / Durham, 30 July 2020

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

Agenda

Quantum computers — an introduction

Quantum machine learning with quantum circuits

Example in python code

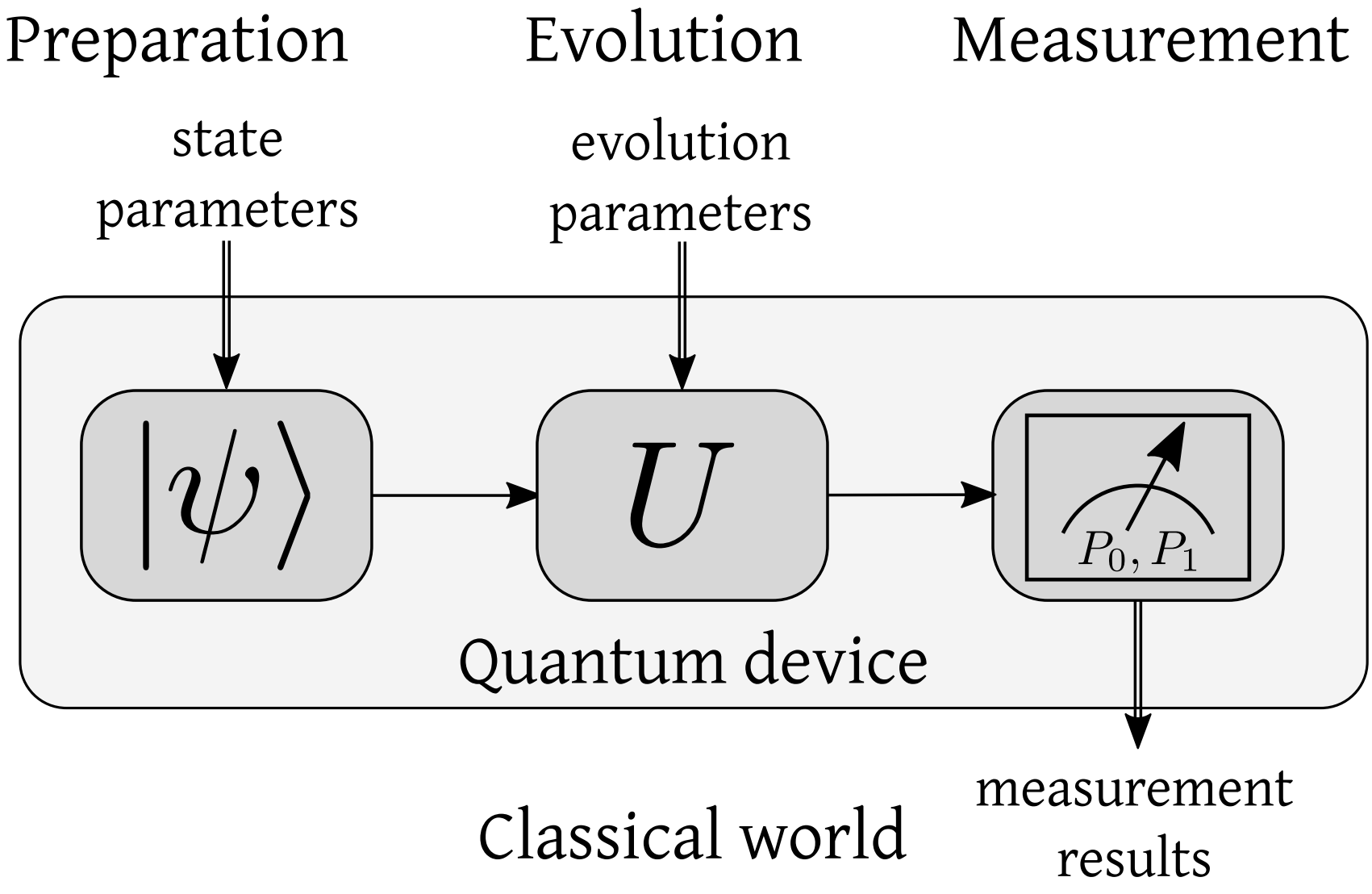
Multiclass classification

Experiment

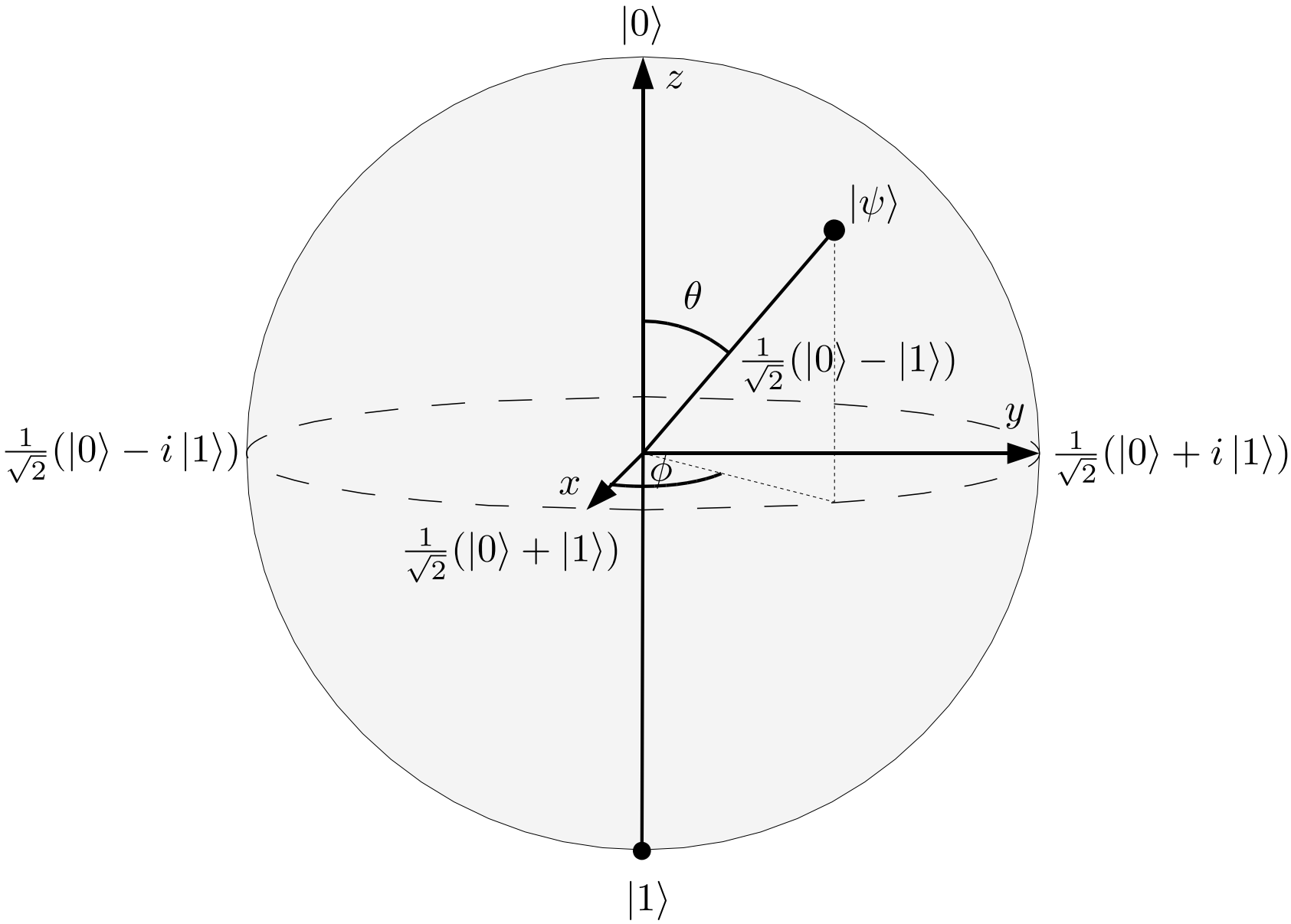
Summary

Bibliography

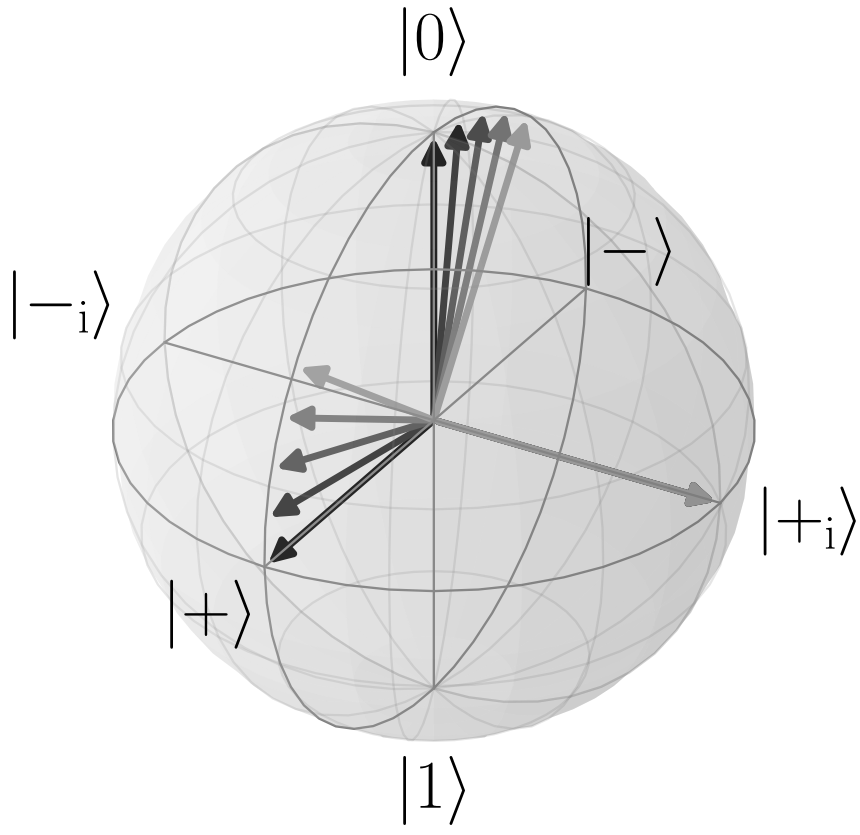
Computation as experiment



Qubit, Bloch sphere

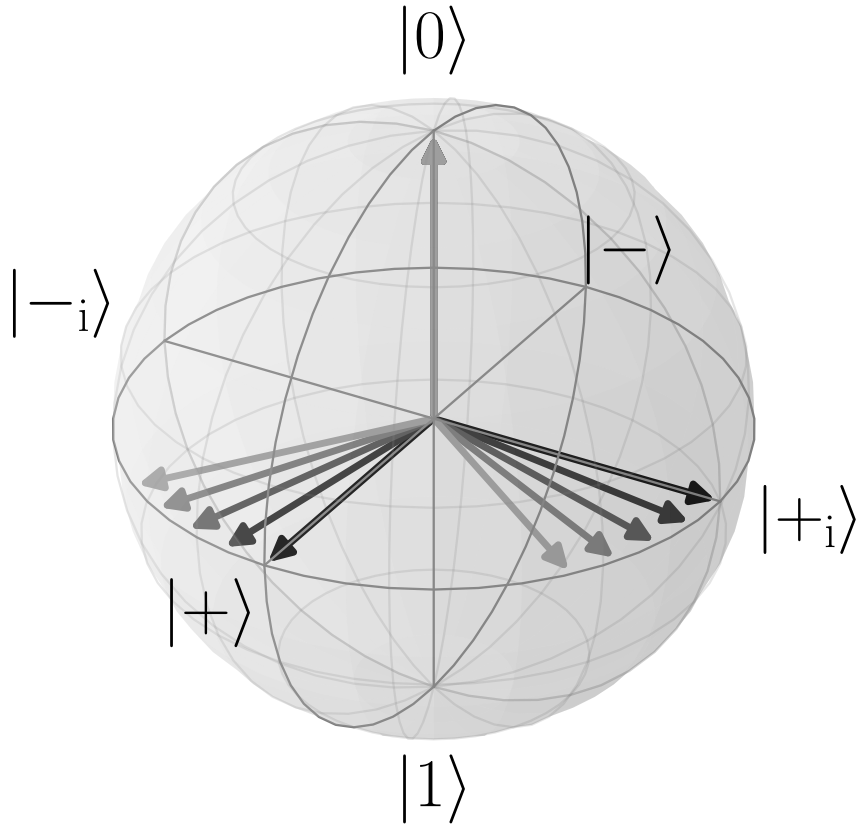


Unitary gates, quantum evolution



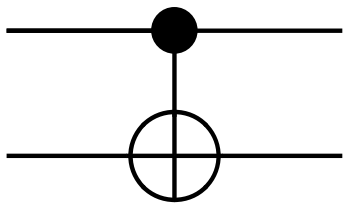
Rotation around y-axis

Unitary gates, quantum evolution



Rotation around z-axis

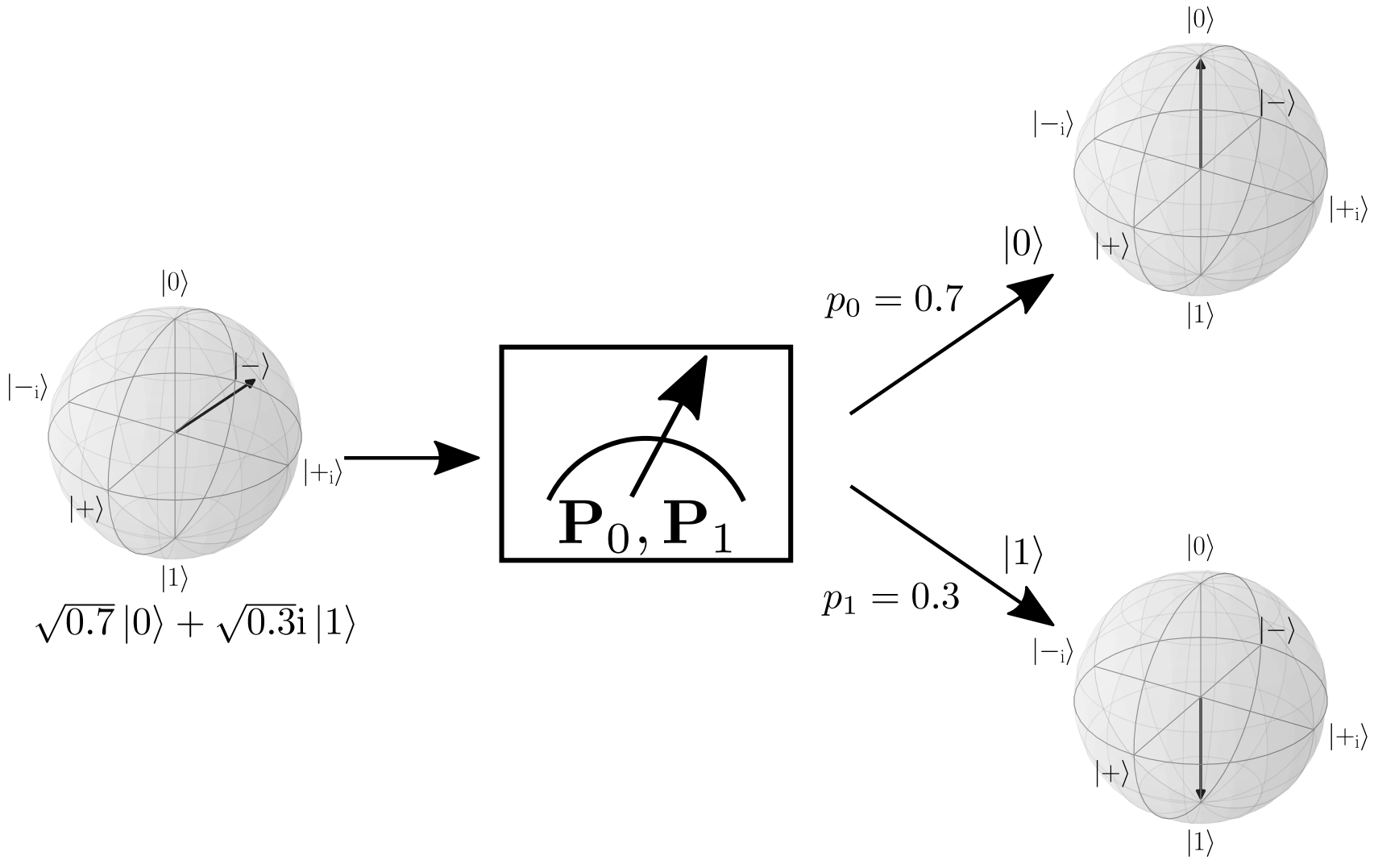
Unitary gates, quantum evolution



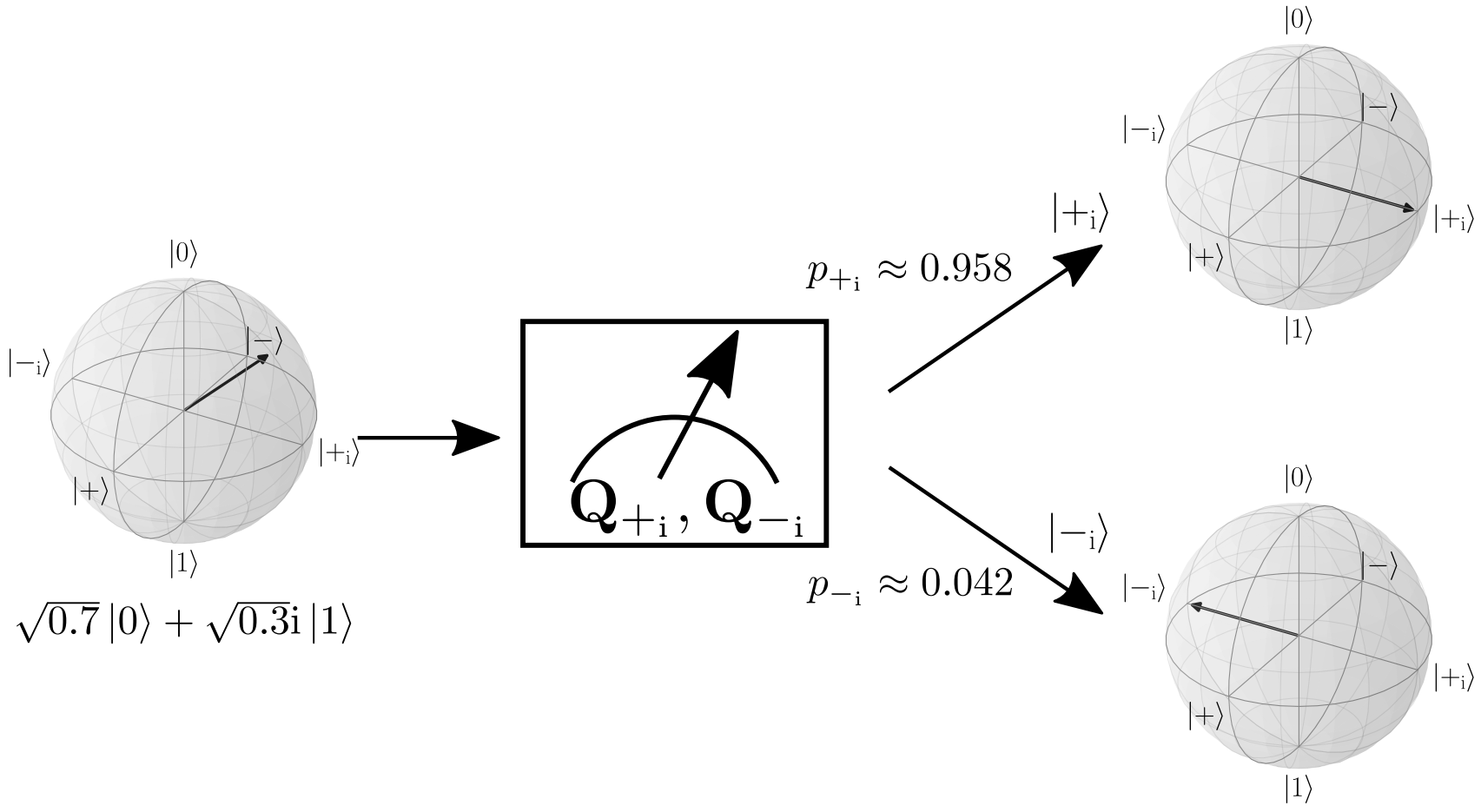
in_1	in_2	out_1	out_2
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Controlled-Not gate

Measurement



Measurement



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.

$$O^\dagger = O.$$

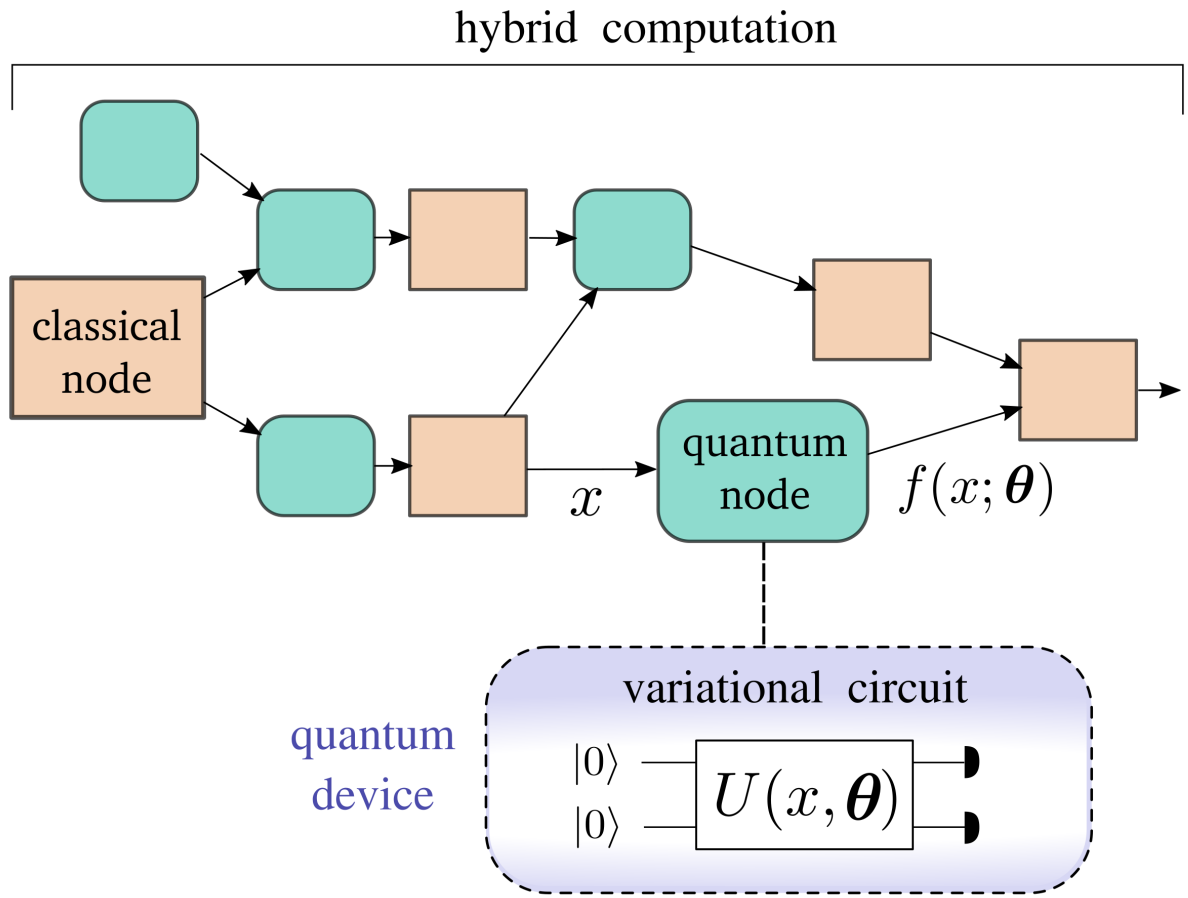
- The expectation value of an observable O in a state $|\psi\rangle$

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

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

Quantum neural networks I

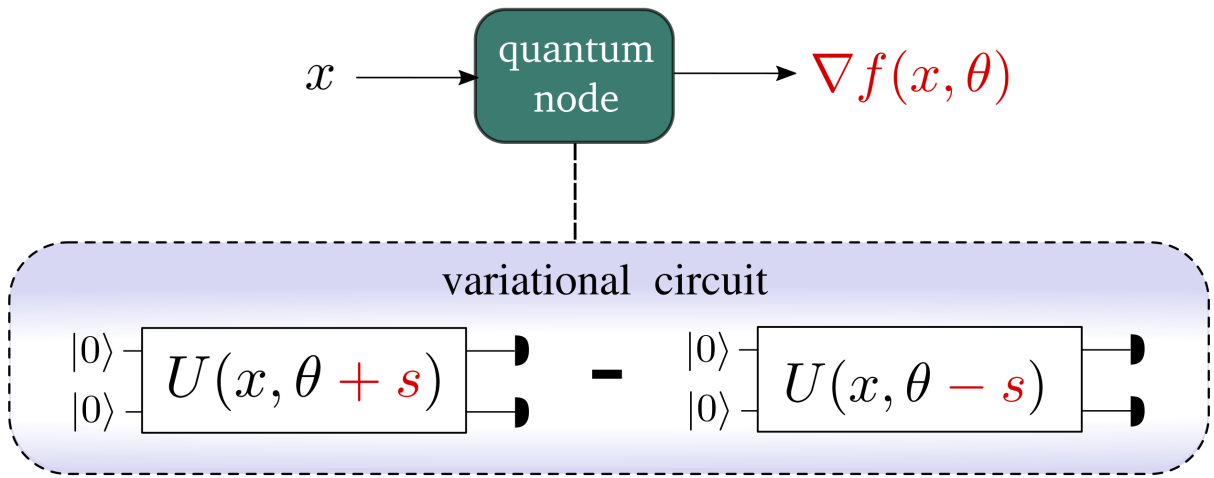
Concept of hybrid computation by xanadu.ai



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)

Gradients, paramter 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[1]

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

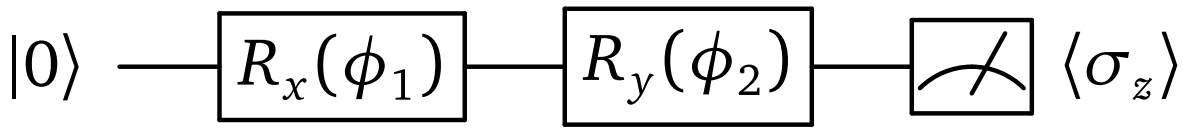
where $s = \pi/4r$.

- 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

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

[1] M. Schuld et al. "Evaluating analytic gradients on quantum hardware". In: *Physical Review A* 99.3 (2019), p. 032331.

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

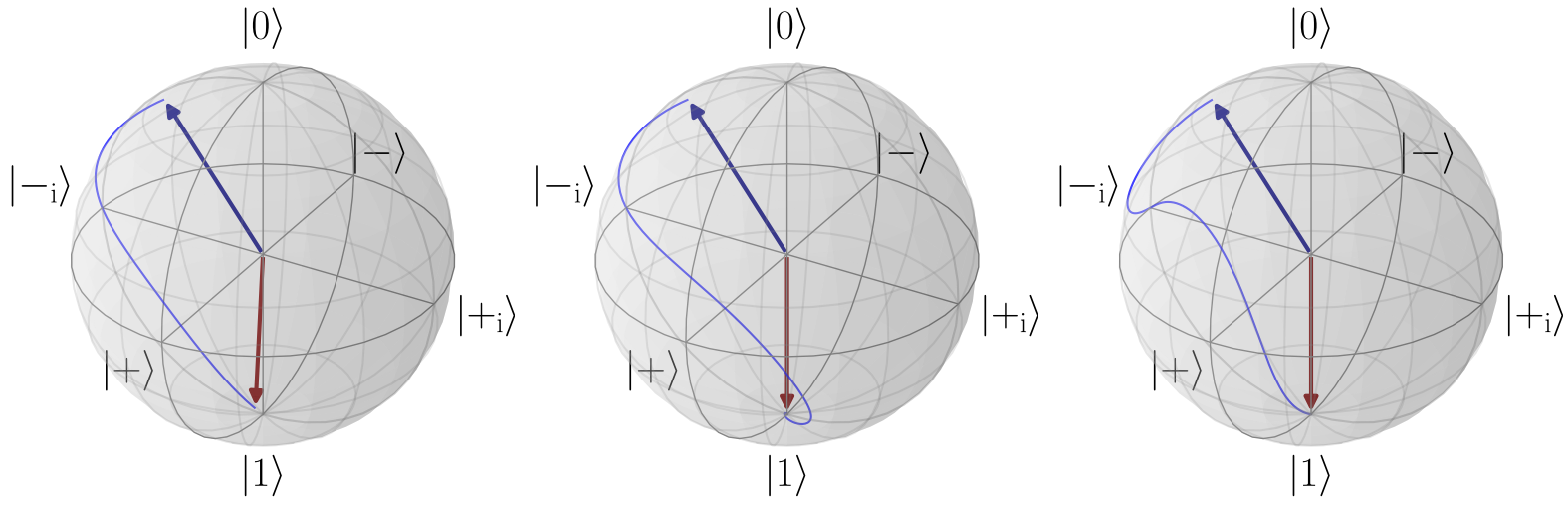


Figure: Three optimization strategies: gradient descent, Nesterov momentum optimizer, Adam optimizer ($var_{\mathrm{init}} = (0.5, 0.2)$).

Iris

(a) Setosa

(b) Virginica

(c) Versicolor

Figure: Three species of iris

Source: Wikimedia Commons / Radomil | Dlanglois | Frank Mayfield

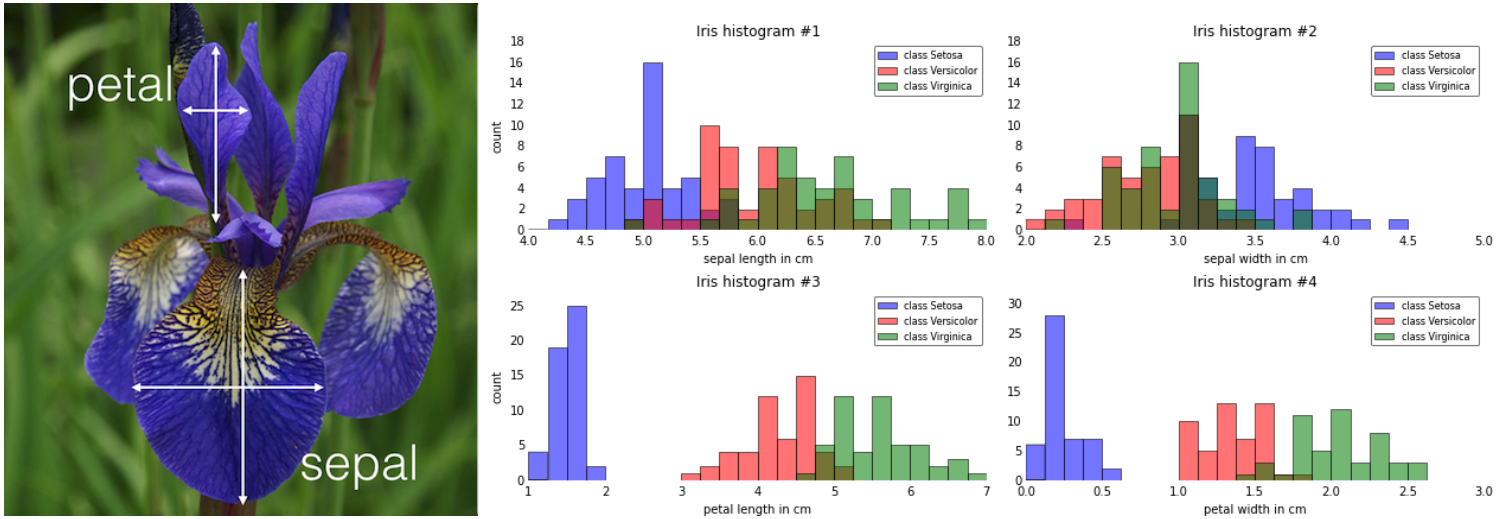


Figure: Anderson dataset — histogram
Sepal, petal

Source: `https://sebastianraschka.com/Articles/2014_intro_supervised_learning.html`

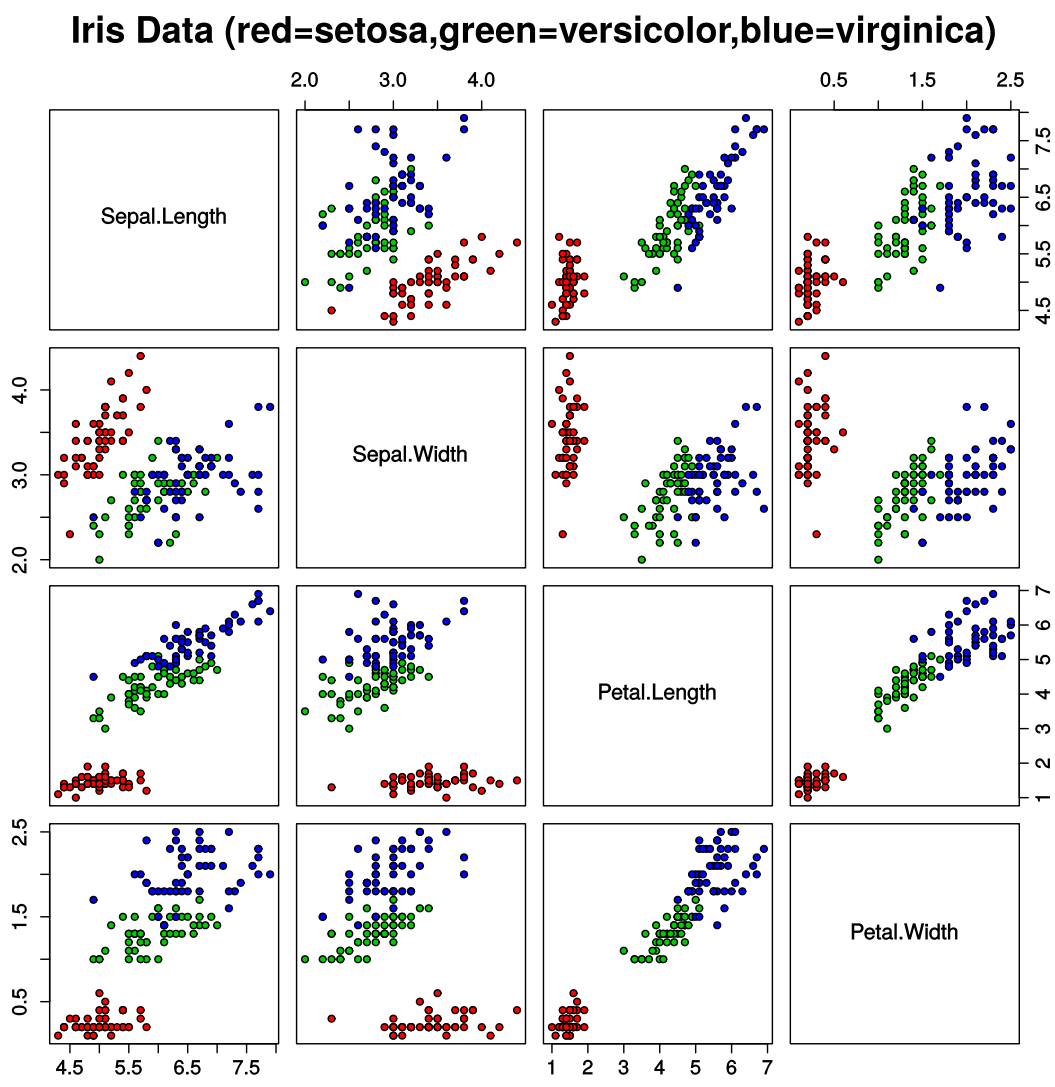


Figure: Anderson dataset

Source: Wikimedia Commons / Nicoguaro — scatterplot

An example of quantum neural network

Dataset

We use a subset of Iris dataset—only 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

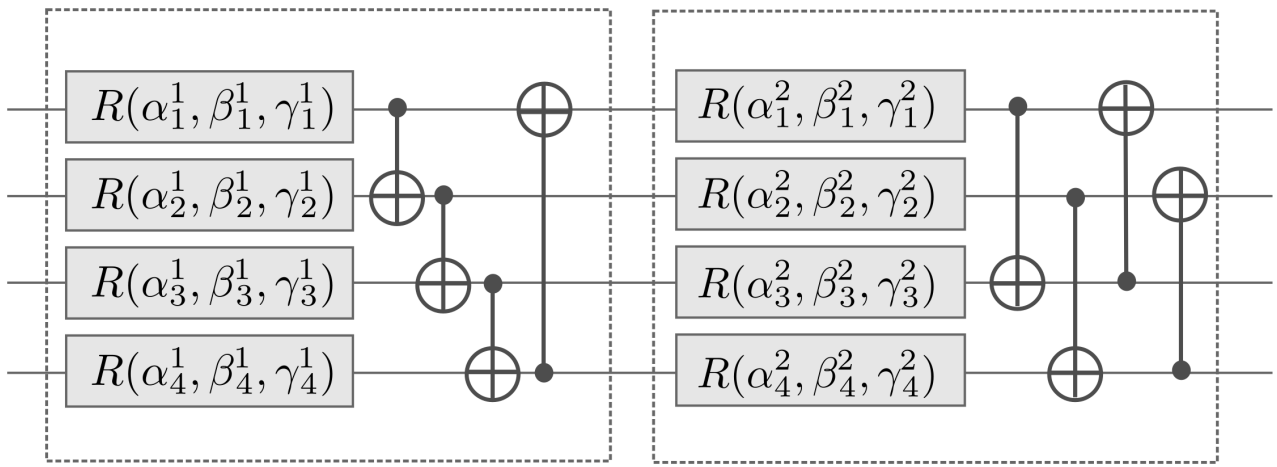
$$R_x(\phi) = e^{-i\phi\sigma_x/2} = \begin{bmatrix} \cos(\phi/2) & -i\sin(\phi/2) \\ -i\sin(\phi/2) & \cos(\phi/2) \end{bmatrix}$$

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

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

  with $\theta = (w, b)$
- Observable $O = \sigma_z \otimes \mathbb{1}^{\otimes(N-1)}$.
- 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, validation, test.
- Initial θ-s: $w \sim \mathcal{U}(0,1)$, $b = 0$.
- Cost function

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

- Batches ξ_t.
- In each step use $\theta^{(t)}$ to calculate

$$\text{accuracy}\left((y_i^{\text{validation}})_i, (\text{cls}(X_{i,:}^{\text{validation}};\theta^{(t)}))_i\right).$$

- To avoid over-fitting choose $\theta^{(t)}$ that maximizes accuracy for validation set.

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.

Entropica labs demo

https://qml.entropicalabs.io/

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  import pennylane as qml
10  from pennylane import numpy as np
11  from pennylane.templates.embeddings import AngleEmbedding
12  from pennylane.templates.layers import StronglyEntanglingLayers
13  from pennylane.init import strong_ent_layers_uniform
14  from pennylane.optimize import GradientDescentOptimizer
```

Binary classification in pennylane

Data import, preprocessing and splitting

```
16  # load the dataset
17  iris = datasets.load_iris()
18
19  # shuffle the data
20  X, y = shuffle(iris.data, iris.target, random_state=0)
21
22  # select only 2 first classes from the data
23  X = X[y<=1]
24  y = y[y<=1]
25
26  # normalize data
27  X = minmax_scale(X, feature_range=(0, np.pi))
28
29  # split data into train+validation and test
30  X_train_val, X_test, y_train_val, 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  # variational quantum classifier
46  def variational_classifier(theta, x=None):
47      weights = theta[0]
48      bias = theta[1]
49      return circuit(weights, x=x) + bias
50
51  def cost(theta, X, expectations):
52      e_predicted = \
53          np.array([variational_classifier(theta, x=x) for x in X])
54      loss = np.mean((e_predicted - expectations)**2)
55      return loss
```

Binary classification in pennylane

Training preparation

```
57  # number of quantum layers
58  n_layers = 3
59
60  # split into train and validation
61  X_train, X_validation, y_train, y_validation = \
62      train_test_split(X_train_val, y_train_val, test_size=0.20)
63
64  # convert classes to expectations: 0 to -1, 1 to +1
65  e_train = np.empty_like(y_train)
66  e_train[y_train == 0] = -1
67  e_train[y_train == 1] = +1
68
69  # select learning batch size
70  batch_size = 5
71
72  # calculate numbe of batches
73  batches = len(X_train) // batch_size
74
75  # select number of epochs
76  n_epochs = 5
```

Binary classification in pennylane

Training

```
78  # draw random quantum node weights
79  theta_weights = strong_ent_layers_uniform(n_layers, n_qubits, seed=42)
80  theta_bias = 0.0
81  theta_init = (theta_weights, theta_bias) # initial weights
82
83  # train the variational classifier
84  theta = theta_init
85
86
87  # start of main learning loop
88  # build the optimizer object
89  pennylane_opt = GradientDescentOptimizer()
90
91  # split training data into batches
92  X_batches = np.array_split(np.arange(len(X_train)), batches)
93  for it, batch_index in enumerate(chain(*(n_epochs * [X_batches]))):
94      # Update the weights by one optimizer step
95      batch_cost = \
96          lambda theta: cost(theta, X_train[batch_index], e_train[batch_index])
97      theta = pennylane_opt.step(batch_cost, theta)
98      # use X_validation and y_validation to decide whether to stop
99  # end of learning loop
```

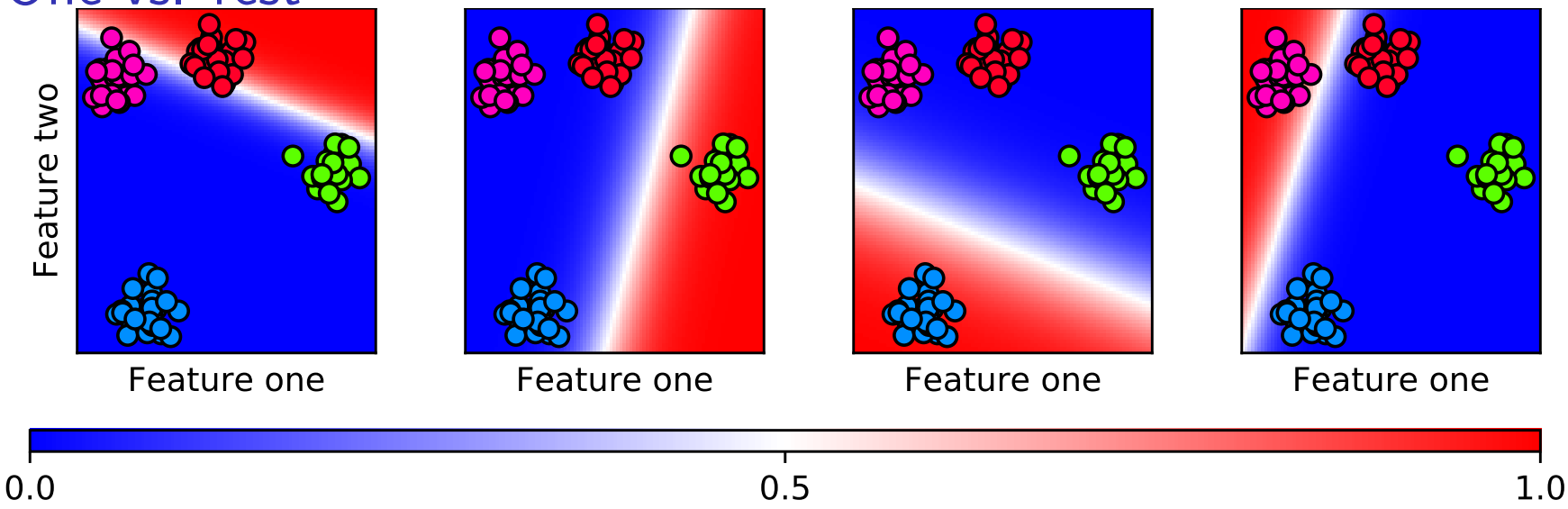
Binary classification in pennylane

Inference

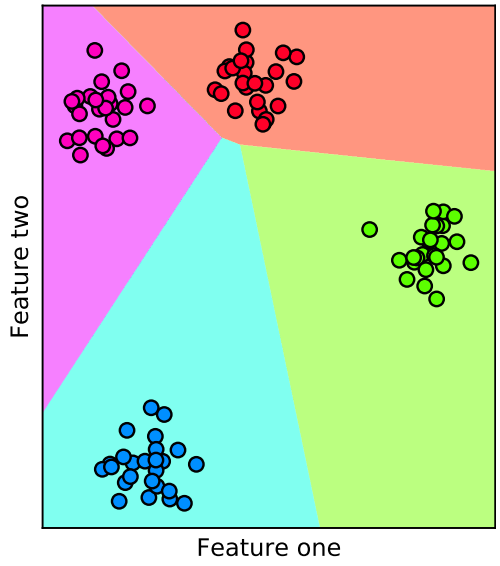
```
101  # convert expectations to classes
102  expectations = np.array([variational_classifier(theta, x=x) for x in X_test])
103  prob_class_one = (expectations + 1.0) / 2.0
104  y_pred = (prob_class_one >= 0.5)
105
106  print(metrics.accuracy_score(y_test, y_pred))
107  print(metrics.confusion_matrix(y_test, y_pred))
```

Multiclass classification

One vs. rest

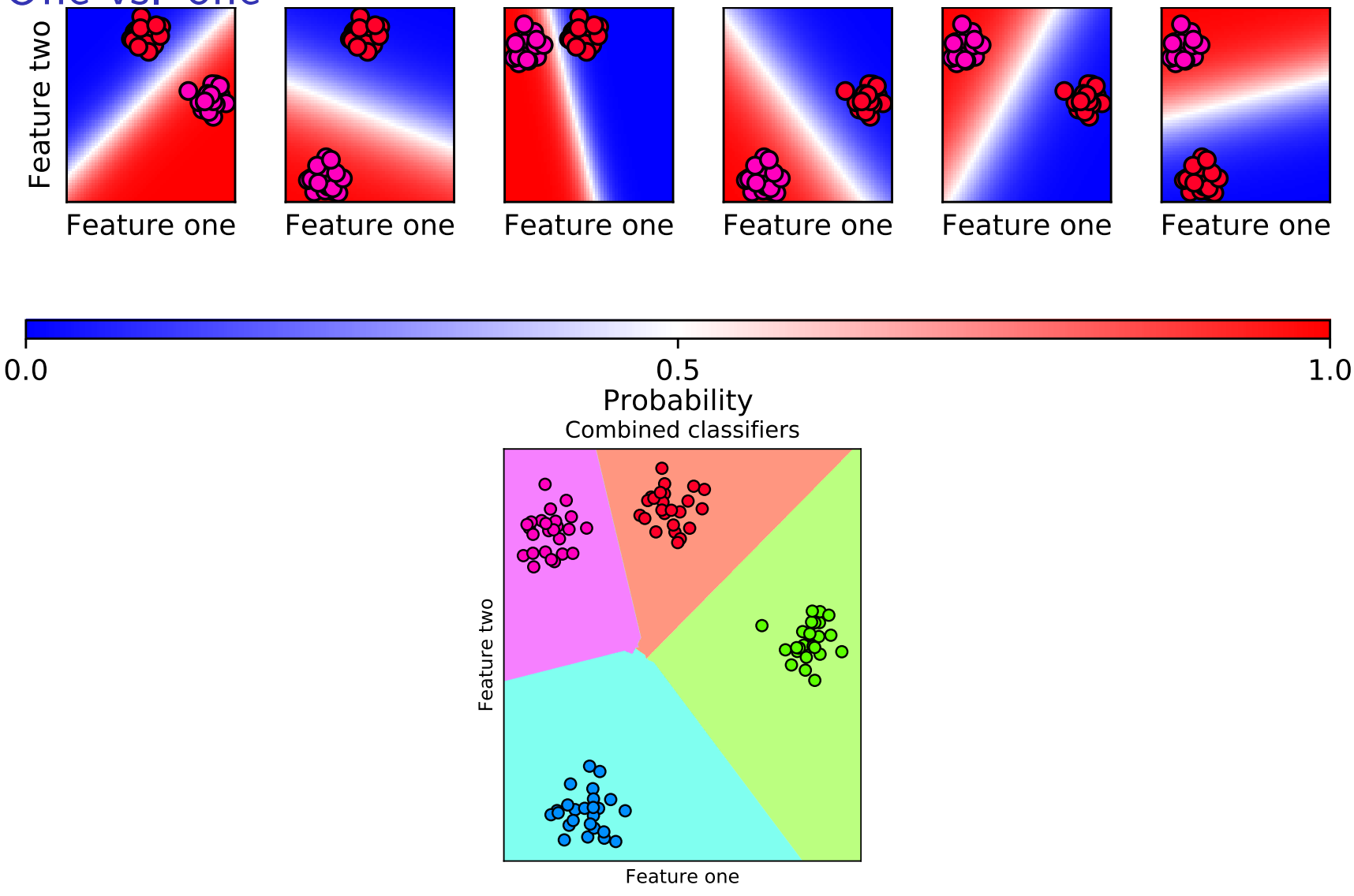


Combined classifiers



Multiclass classification

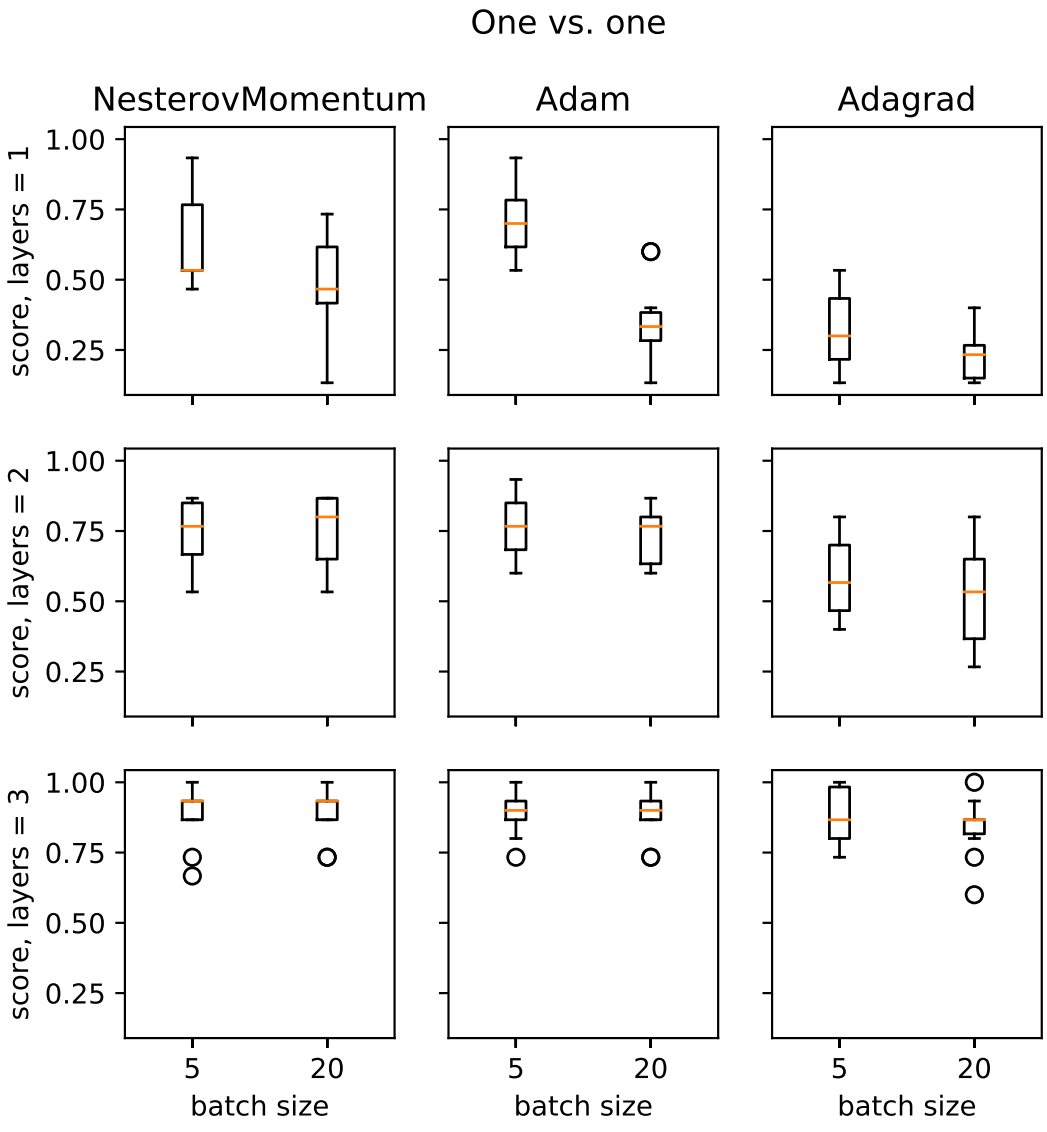
One vs. one



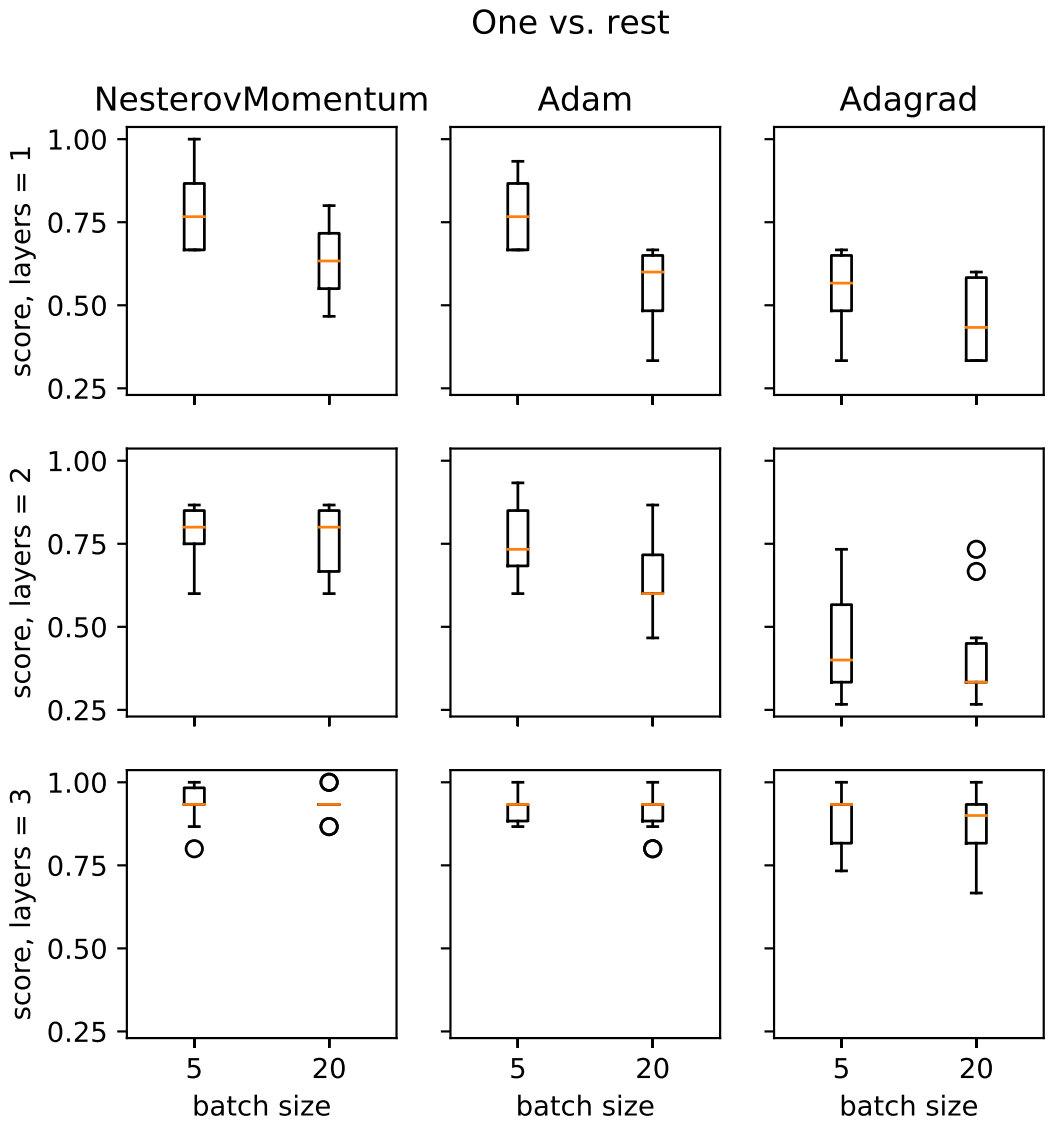
Crossvalidation parameters

- Maximal number of epochs: 20 (fixed).
- Number of layers: $\{1, 2, 3\}$.
- Batch sizes: $\{5, 20\}$.
- Optimizer: NesterovMomentumOptimizer, AdamOptimizer, AdagradOptimizer.

Crossvalidation results



Crossvalidation results



Crossvalidation best results

One vs. one

- Number of layers: 3.
- Batch sizes: 5.
- Optimizer: AdamOptimizer.
- Score: 0.89 ± 0.08.

One vs. rest

- Number of layers: 3.
- Batch sizes: 5.
- Optimizer: NesterovMomentumOptimizer.
- Score 0.93 ± 0.06.

Free software for Quantum Differentiable Computation

- Xanadu's Pennylane (Python) `https://pennylane.ai/`
- Tensorflow Quantum (Python) `https://www.tensorflow.org/quantum`
- QuantumFlow (Python) `https://quantumflow.readthedocs.io/`
- Yao (Julia) `http://yaoquantum.org/`

Conclusions

- Machine learning might be an useful application of quantum computers in their infant state.
- Quantum variational classifiers are an interesting area of research.
- I hope that this talk has encouraged you to play with quantum neural networks.

Bibliography

- Biamonte, J. et al. “Quantum machine learning”. In: *Nature* 549.7671 (2017), p. 195.
- Schuld, M., M. Fingerhuth, and F. Petruccione. “Implementing a distance-based classifier with a quantum interference circuit”. In: *EPL (Europhysics Letters)* 119.6 (2017), p. 60002.
- Schuld, M. and F. Petruccione. *Supervised Learning with Quantum Computers (Quantum Science and Technology)*. Springer, 2018. ISBN: 978-3-319-96424-9.
- Schuld, M. et al. “Circuit-centric quantum classifiers”. In: *arXiv preprint arXiv:1804.00633* (2018).
- Schuld, M. et al. “Evaluating analytic gradients on quantum hardware”. In: *Physical Review A* 99.3 (2019), p. 032331.

Thank you for your attention!

Piotr Gawron

@pwgawron

p.w.gawron@gmail.com

https://pgawron.github.io/

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

ASTROCENT

European Funds Smart Growth

Republic of Poland

FNP Foundation for Polish Science

European Union European Regional Development Fund

CAMK PAN