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

Agenda

Quantum machine learning with quantum circuits

Example in python code

Multiclass classification

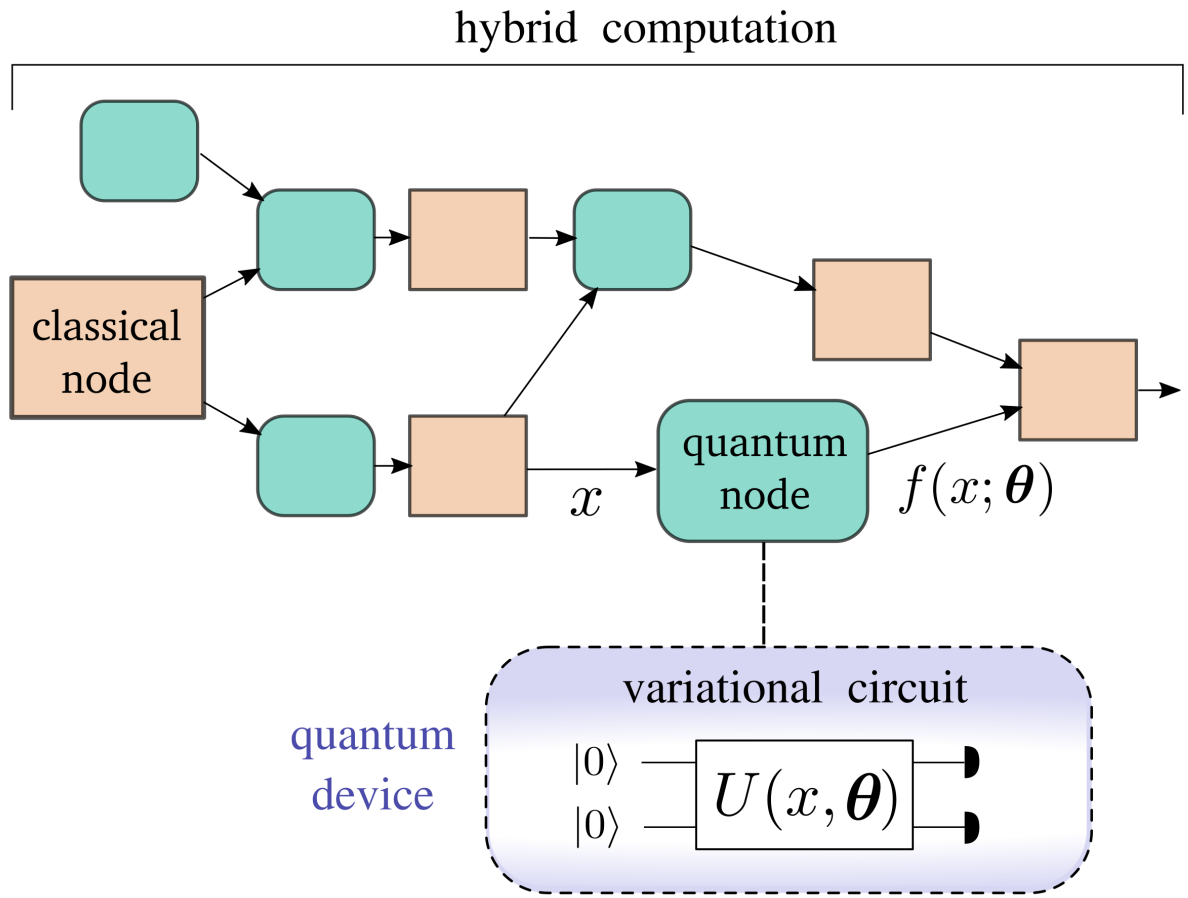
Experiment

Summary

Bibliography

Quantum neural networks I

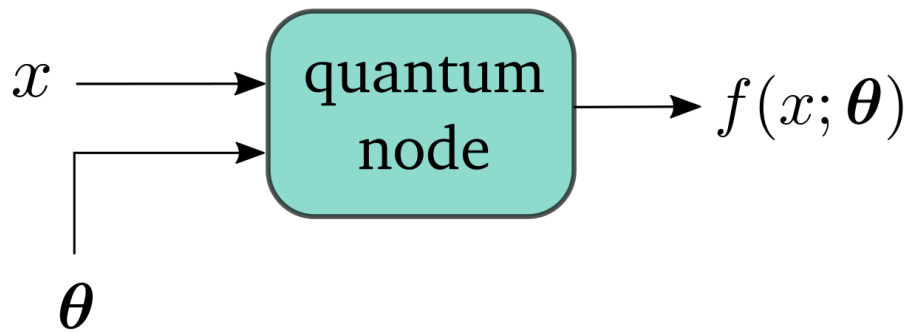
Concept of hybrid computation by xanadu.ai



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

Quantum neural networks II

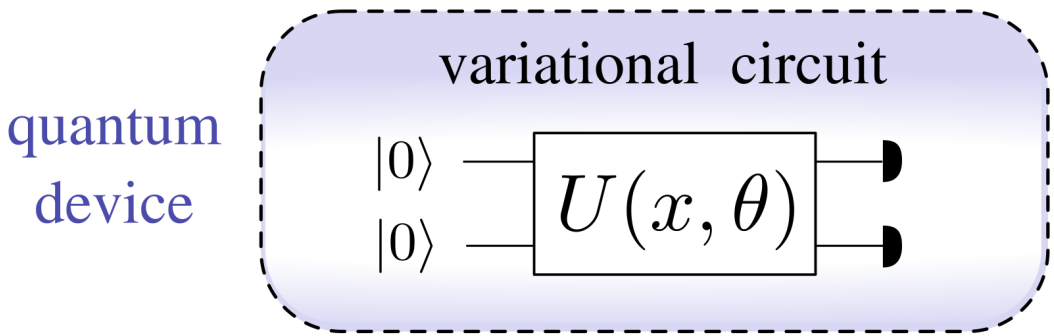
Quantum node



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

Quantum neural networks III

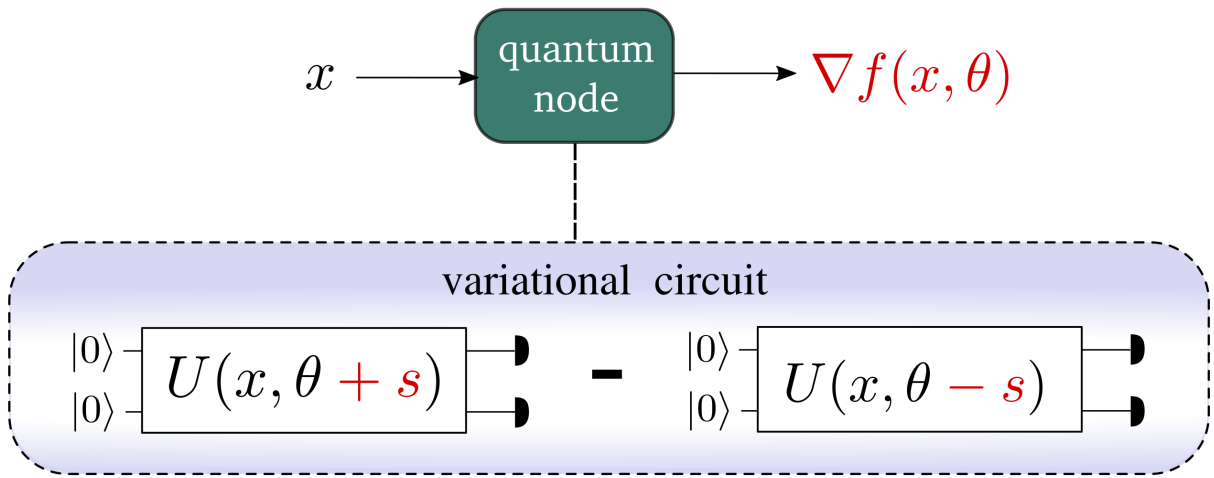
Variational circuit



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

Quantum neural networks IV

Gradient



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

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 $\min = 0$, $\max = 2\pi$.
- 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 $x = (x_0, x_1, \ldots, x_N) \in \mathbb{R}^N$ 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

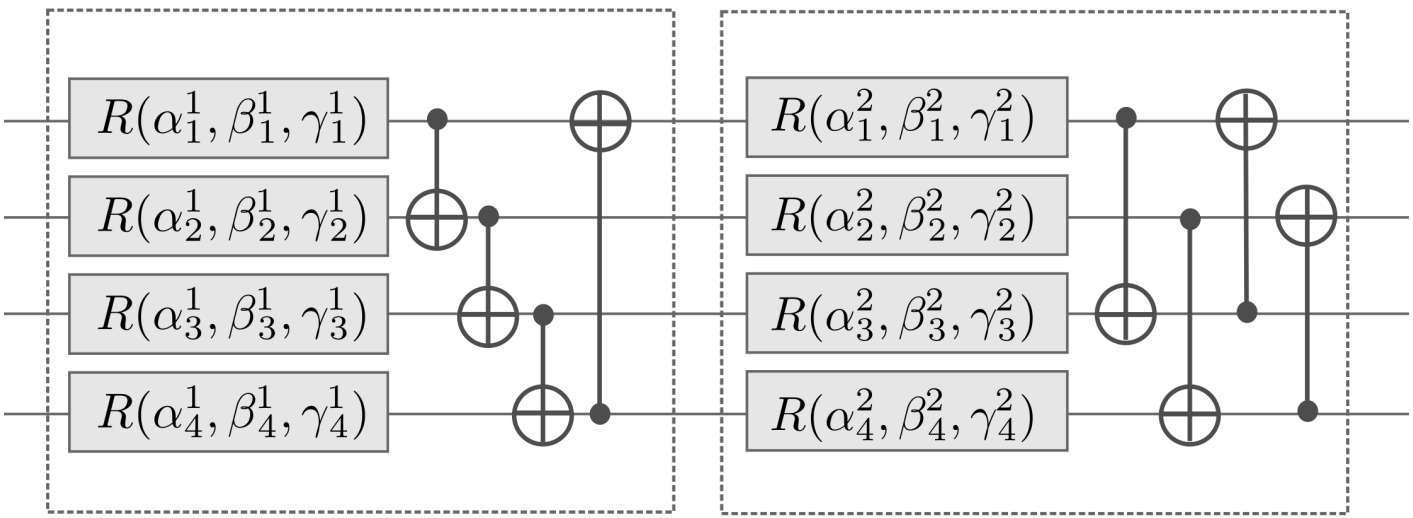
$$R_x(\phi) = e^{-i\phi\sigma_x/2} = \begin{bmatrix} \cos(\phi/2) & -i\sin(\phi/2) \\ -i\sin(\phi/2) & \cos(\phi/2) \end{bmatrix}$$

- and $x_i \in [0, 2\pi]$.

An example of quantum neural network

Variational gate

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



An example of quantum neural network

Classification function

- The classification function $f : \mathbb{R}^N \to [-1, +1]$.
- $f(x; \theta, b) = \langle O \rangle_{U_\mathrm{v}(\theta) U_\mathrm{e}(x)|0\rangle} + b$, with

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

- Observable $O = \sigma_z \otimes \mathbb{1}^{\otimes(N-1)}$.
- With gradient $\frac{\partial}{\partial \theta_k} f = c_k \left[f(\theta_k + s_k) - f(\theta_k - s_k) \right]$.
- For example:

$$\frac{d}{d\phi} f(R_x(\phi)) = \frac{1}{2} \left[f(R_x(\phi + \pi/2)) - f(R_x(\phi - \pi/2)) \right]$$

where f is an expectation value depending on $R_x(\phi)$.

An example of quantum neural network

Training

- $X, \mathbb{R}^N$: data set,
- $Y, \{-1, 1\}$: true labels,
- $\hat{Y}' = (f(x; \theta, b))_{x \in X}$: predicted "soft" labels $[-1, 1]$,
- $\mathrm{cost}(Y, \hat{Y}') = \sum_{y_i \in Y, \hat{y}'_i \in \hat{Y}'} (y_i - \hat{y}'_i)^2 / |Y|$,
- $\mathrm{accuracy}(Y, \hat{Y}) = \sum_{y_i \in Y, \hat{y}_i \in \hat{Y}} (\mathbb{1}_{y_i \neq \hat{y}_i}) / |Y|$,
- $\hat{Y} = (\mathrm{sgn}(f(x; \theta, b)))_{x \in X}$: predicted labels.

Optimizers

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

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

An example of quantum neural network

Training

- The training—fitting parameters θ and b of the function $f(x; \theta, b)$ to the training data $X \times Y$.
- Cost function minimization—gradient descent in batches (of several data-points) using an Optimizer in several epochs.
- Score function over the validation set is calculated for every batch.
- To avoid overfitting this best score determines which classifier parameters (θ, b) are used for classification.
- Initial θ-s from $\mathcal{U}(0, 1)$, $b = 0$.

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, 2*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  acc_val_best = 0.0
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
```

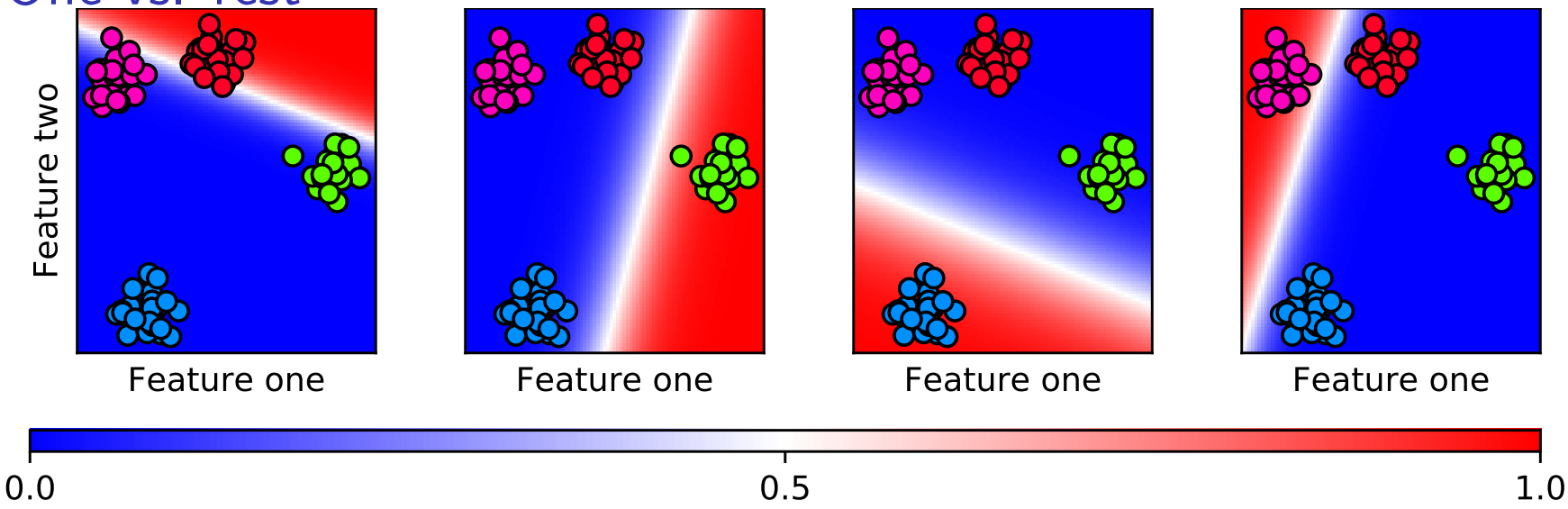
Binary classification in pennylane

Inference

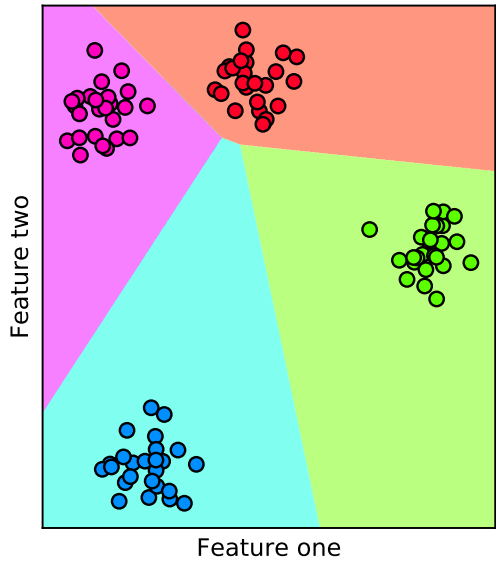
```
101  # convert expectations to classes
102  expectations = np.array([variational_classifier(theta, x=x) for x in X_test])
103  prob_class_one = (expectations + 1.0) / 2.0
104  y_pred = (prob_class_one >= 0.5)
105
106  print(metrics.accuracy_score(y_test, y_pred))
107  print(metrics.confusion_matrix(y_test,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

Multiclass classification

One vs. rest

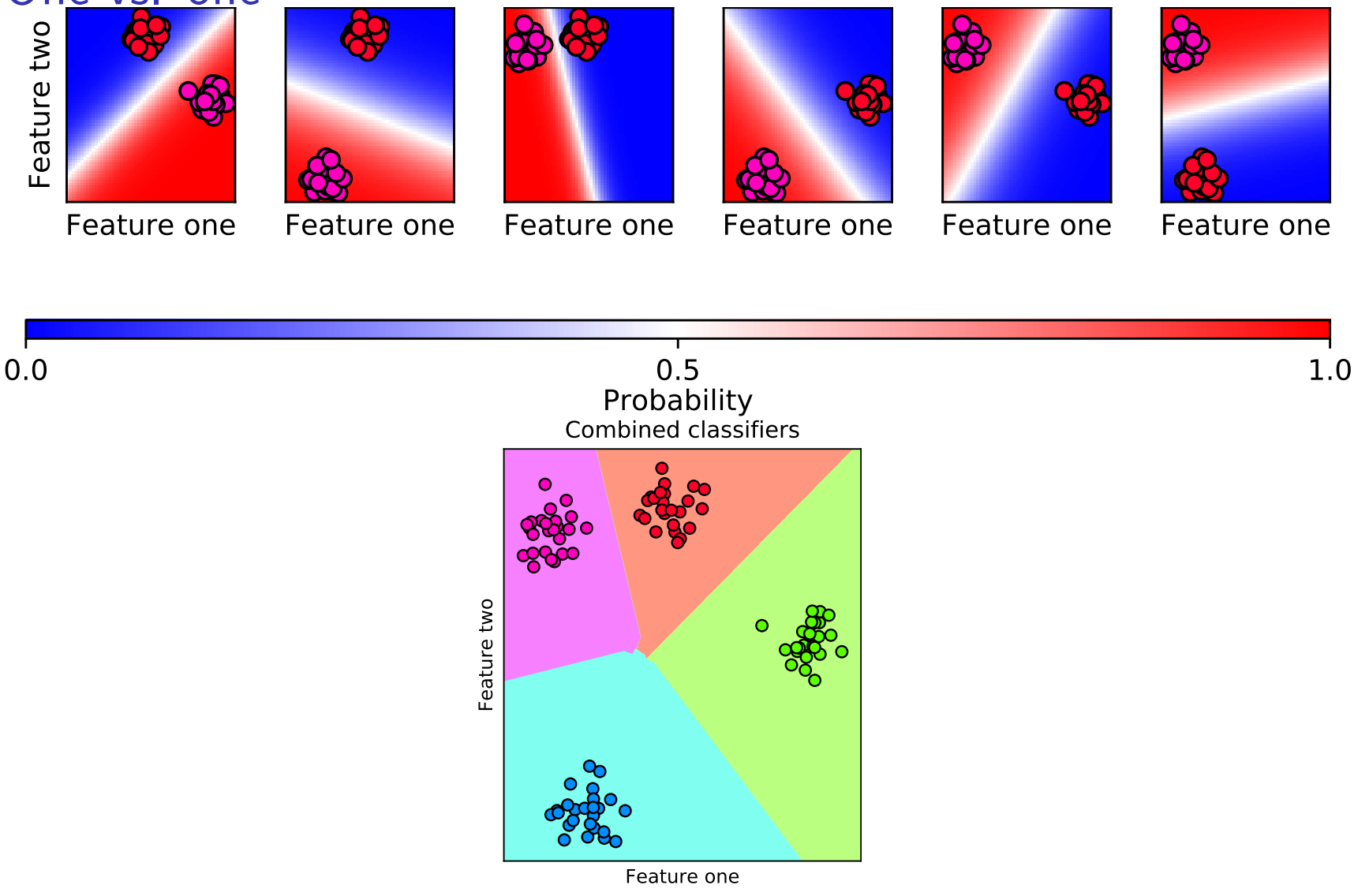


Combined classifiers



Multiclass classification

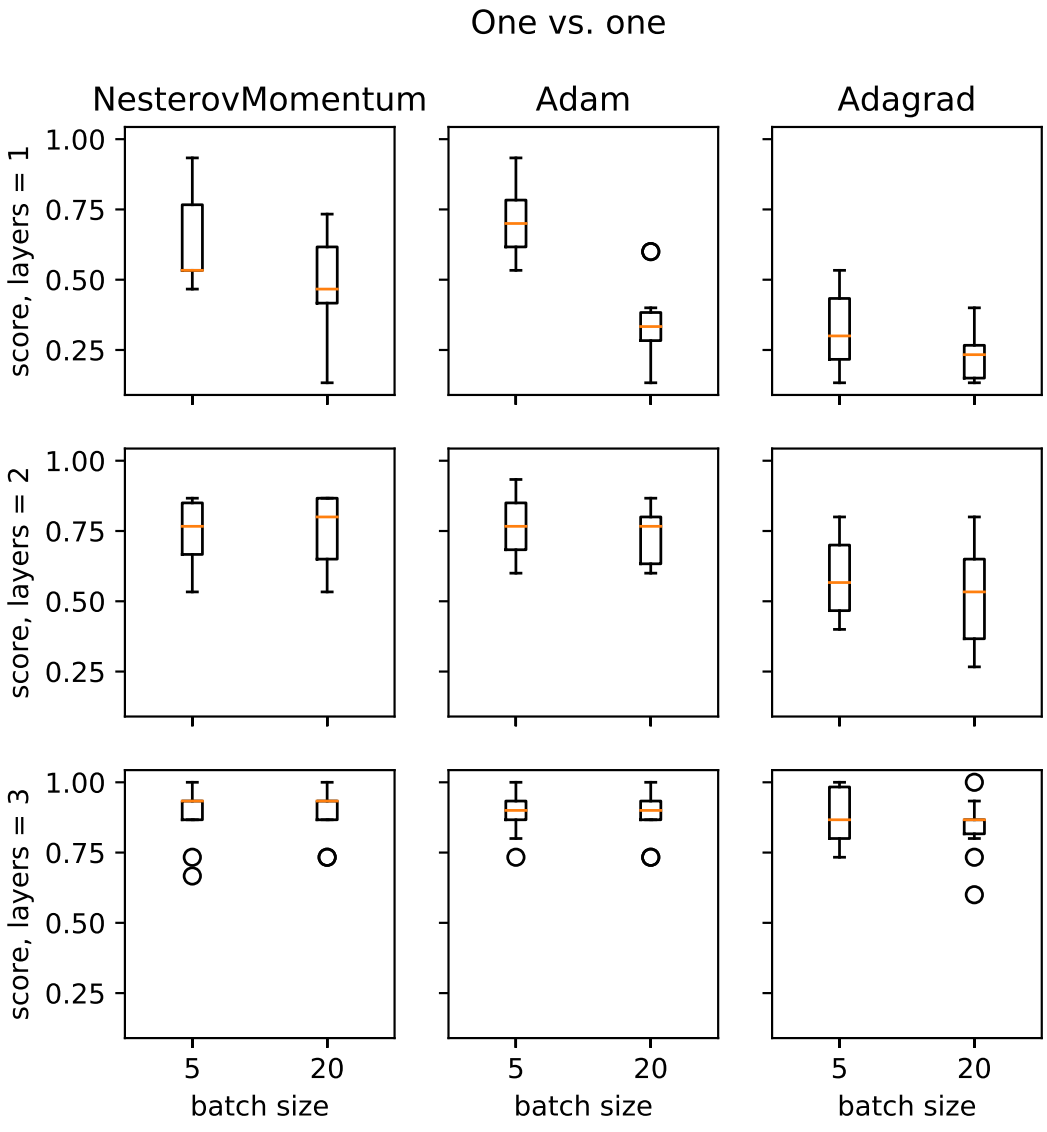
One vs. one



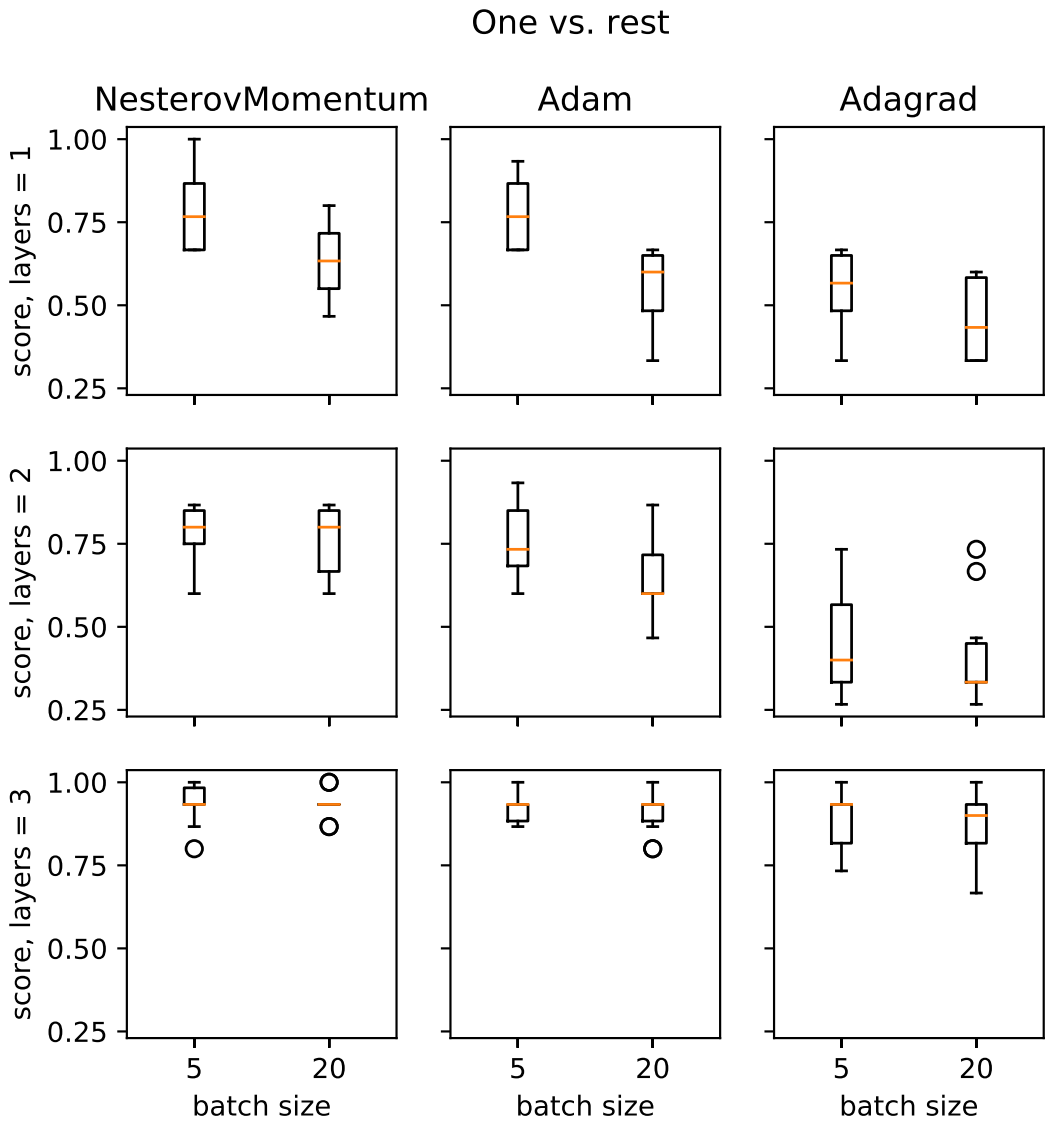
Crossvalidation parameters

- 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/`
- 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.
- Python is too slow to be useful in this kinds of experiments. W need better tools!

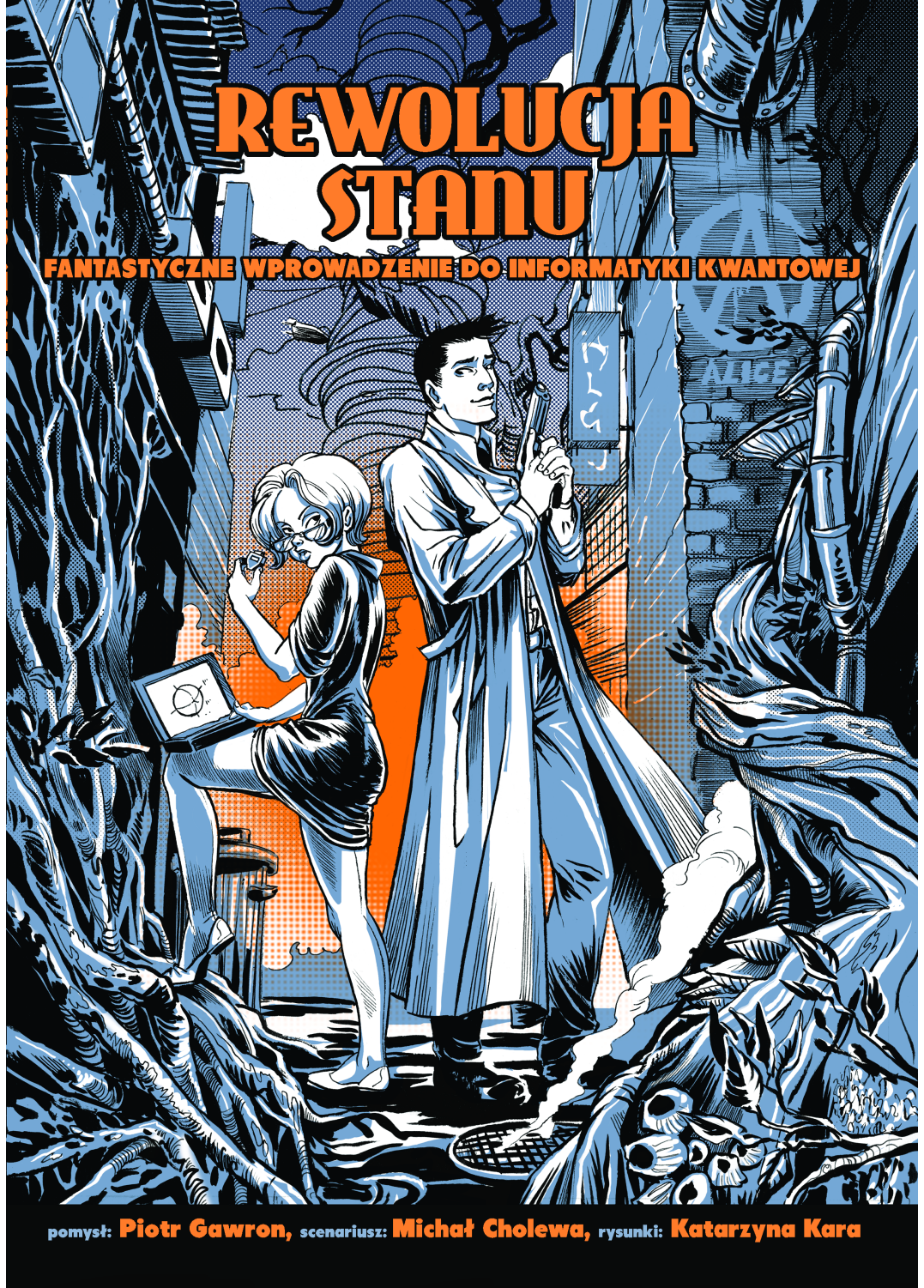
Bibliography I


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


M. Schuld, A. Bocharov, K. Svore, and N. Wiebe.
Circuit-centric quantum classifiers.
arXiv preprint arXiv:1804.00633, 2018.


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


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

p.w.gawron@gmail.com

https://pgawron.github.io/

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

"Rewolucja stanu" can be bought or downloaded

http://tinyurl.com/rewolucjastanu

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