Machine learning using a hybrid quantum-classical algorithm

Maximilian Spiegel[1], Sebnem Gül-Ficici[1], Ulrich Göhner[1]

[1]University of Applied Sciences Kempten, Bahnhofstraße 61, 87435 Kempten, Germany

1 Abstract

The industrial sector uses artificial intelligence (AI) in many ways. E.g. anomaly detection to identify and examine abnormal behavior of machines, such as voltage and current fluctuation. To develop self driving cars AI is used to perform segmentation of the environment to navigate the vehicle and make decisions, preferably in real-time.
Quantum computers are already being used for special machine learning processes, achieving, in some instances, better results than a regular machine learning algorithm. This paper will elaborate on the upsides of a machine learning model consisting of a hybrid between a quantum machine learning (QML) algorithm and a classical machine learning algorithm.

KEYWORDS: Artificial Intelligence, Hybrid Quantum Machine Learning

2 Project Objectives

For a quick start in artificial intelligence, the following reference can be consulted for more details [1]. For an introduction on quantum mechanics take a look at [2] and for quantum computing refer to [3].
Machine learning, especially deep learning, requires an extensive data record in order to be trained appropriately. Processing large amounts of data, however, is expensive, due to its highly time consuming and computing-intensive nature. The added value of this trained model lies in its ability to lower cost through an early detection of errors. Efforts required for both, training and evaluation, could possibly be diminished notably by using a quantum computer.
Data measured by sensors are usually formatted in a way that allows regular computers to interpret them. Even if the data was an analog measurement it will still be stored as a digital bit string. However, quantum computers are unable to interpret bit strings, therefore necessitating a data encoder before handing over the data to a quantum algorithm. Such a hybrid quantum-classical algorithm is often referred to as a variational quantum algorithm [4]. In this paper the Python libraries Qiskit by IBM and Pytorch have been used to build a hybrid QML.
The standard machine learning network pre-processes the input data. Its output is then transmitted to the QML network.
As mentioned the data is encoded as a binary string and needs to be converted. To complete the data encoding, the machine learning's output will be passed on to a quantum feature map. This quantum feature map then converts a binary string into quantum states that the quantum algorithm is able to work with. After the data has been converted into quantum states, they are forwarded to the "Ansatz" which represents the weights of our "artificial quantum neurons". After the algorithm reached a conclusion a different problem arises; the data presentation now has to be reversed to compare the conclusion and adjust the weights within the neural network (see *[Fig. 1]*).

© 2023 Copyright by DYNAmore GmbH, an Ansys Company

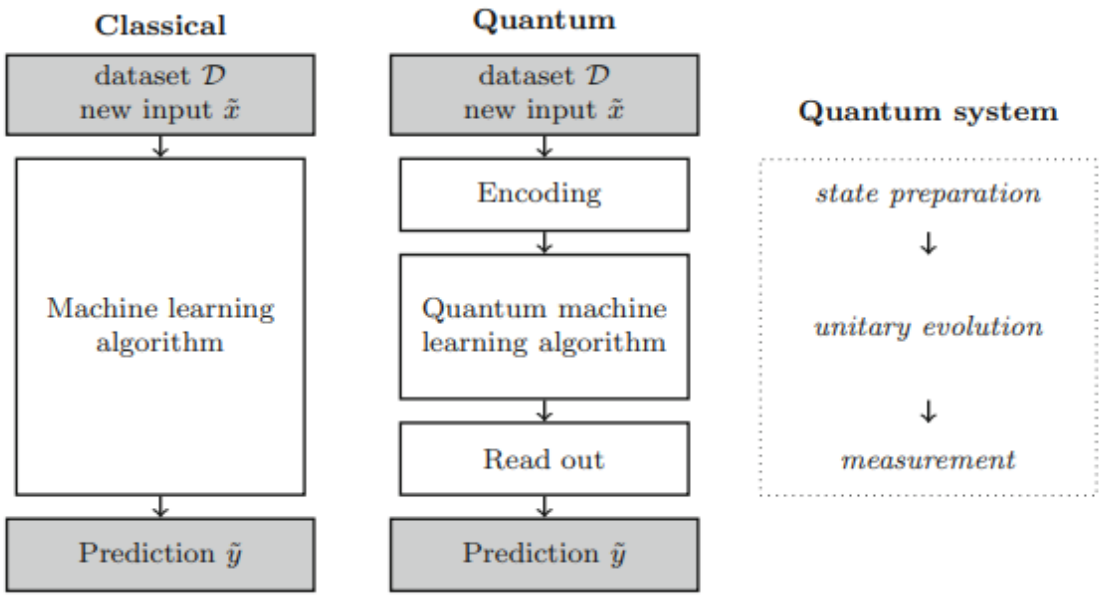

Fig.1: Comparison between a classical machine learning algorithm (left) and a quantum machine learning algorithm (middle). In our case, the prediction $\vec{y}$ of the classical algorithm is used as the input $\vec{x}$ of the quantum algorithm. Figure source: [3]

3 Data Preparation

For the machine learning process, we're going to use two different types of data.

The MNIST digit dataset is being used to evaluate the different neural networks on the classification technique between two digits. This is a comparison on a supervised machine learning method. From the dataset we're only using the digits "0" and "6".
The training dataset will be as big as 100, 200 and 800 samples. The test and evaluation dataset will be just as big as the training dataset. All samples will be chosen randomly from the entirety of the MNIST dataset.

For an unsupervised machine learning comparison, we have time-series measurements of the current taken from a machine. This scenario will show the comparison in the case of an anomaly detector.
To get better results with the anomaly detection, the training dataset will be much bigger than the classification job is using. Also we aren't able to pick out random datapoints, as this is a time-series dataset. So we are using 5,000 connected datapoints from a given start point. For the training and the evaluation, we will use 50,000 datapoints to draw a chart that's giving us more information about the performance of the different neural networks.

4 Classical Neural Network

For the classification we implement a simple "Convolutional Neural Network" (CNN). *[Fig.2]* shows the structure of the network. The training includes ten epochs. We set the learning rate to 0.001. The parameters are set for the pure classical network as well as for the hybrid network.

© 2023 Copyright by DYNAmore GmbH, an Ansys Company

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 2, 24, 24]              52
            Conv2d-2           [-1, 16, 8, 8]              816
         Dropout2d-3           [-1, 16, 4, 4]                0
            Linear-4                 [-1, 64]           16,448
            Linear-5                  [-1, 2]              130
            Linear-6                  [-1, 1]                3
================================================================
Total params: 17,449
Trainable params: 17,449
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.07
Estimated Total Size (MB): 0.09
----------------------------------------------------------------
```

Fig.2: The implemented classical CNN without QNN

For the anomaly detection we chose a “Long Short-Term Memory” (LSTM). We set the lookback to 128, the learning rate to 0.0003 and trained the network for over 200 epochs.

5 Quantum Neural Network

For our QNN we use the “ZZFeatureMap” method as data encoding. The “RealAmplitudes” is going to be our “Ansatz”. We will use two qubits. See *[Fig.3]*.

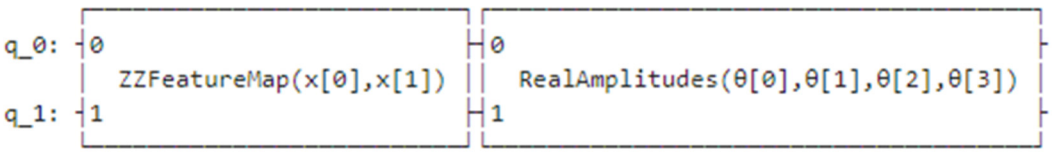

Fig.3: The implemented QNN with two qubits, “q_0” and “q_1”.

To build a hybrid algorithm we combine the classical networks with this QNN. The classical part is placed before the QNN. This means that the output of the classical model is used as input for the quantum model.

© 2023 Copyright by DYNAmore GmbH, an Ansys Company

6 Hybrid Neural Network

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 2, 24, 24]              52
            Conv2d-2           [-1, 16, 8, 8]             816
         Dropout2d-3           [-1, 16, 4, 4]               0
            Linear-4                [-1, 64]          16,448
            Linear-5                 [-1, 2]             130
    TorchConnector-6                 [-1, 1]               4
================================================================
Total params: 17,450
Trainable params: 17,450
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 0.07
Estimated Total Size (MB): 0.09
----------------------------------------------------------------
```

Fig.4: The implemented classical CNN connected to a QNN (hybrid neural network).

Comparing *[Fig.2]* and *[Fig.4]* we can see that the two networks are almost identical. The only difference between the two networks is that the hybrid neural network has one more parameter in the last layer. That is because the output of “Linear-5” isn’t densely connected to the output layer directly. Instead of having one neuron acting as a direct output (with one bias, see *[Fig. 5]*), we have two qubits that both get an independent bias (see *[Fig.3]* and *[Fig.6]*). After the calculation the output of the QNN is then forwarded to a classical neuron and can be interpreted by the classic computer.

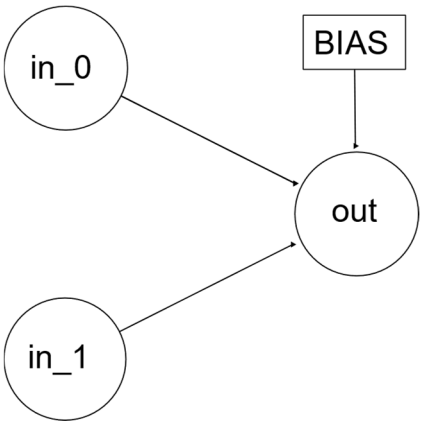

Fig.5: The implemented classical CNN’s output layer with one BIAS.

© 2023 Copyright by DYNAmore GmbH, an Ansys Company

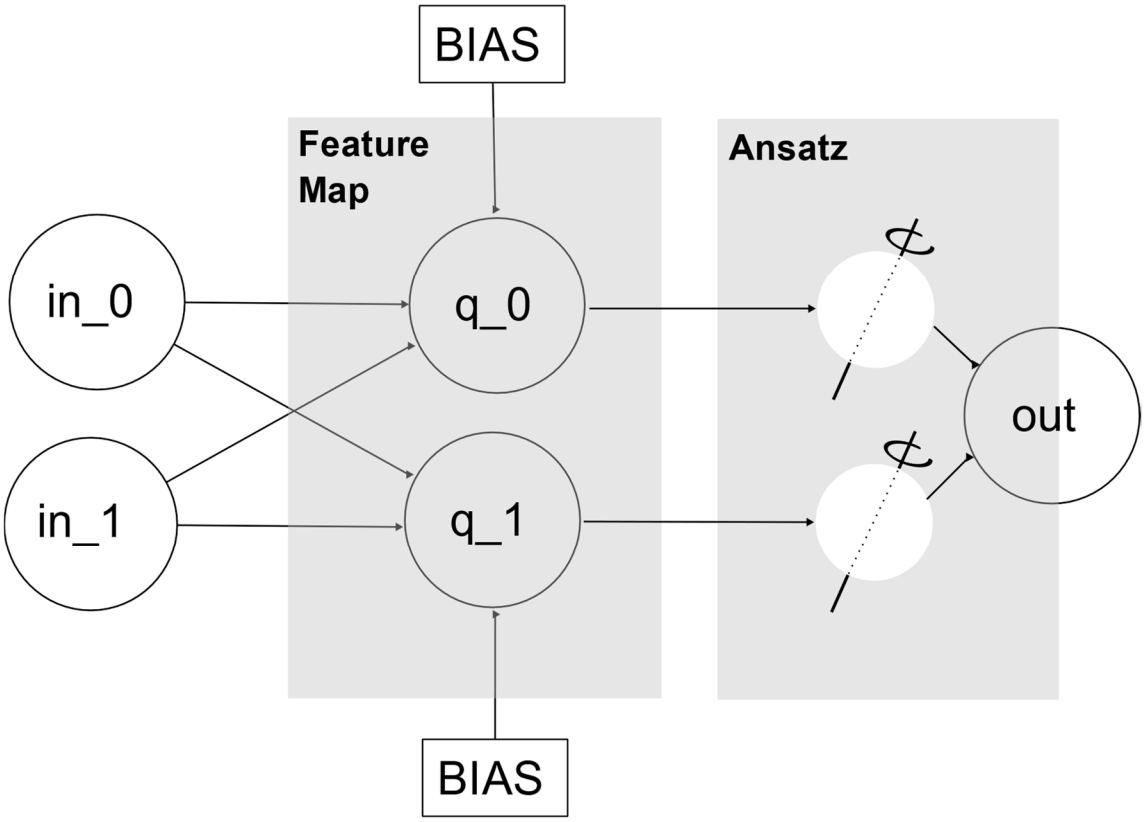

Fig.6: The implemented hybrid NN's output layer with two BIASes."in_0" and "in_1" is the output from the layer "Linear_5" of [Fig.4]. "out" is the measurement we need to do, in order to get the current state of the quantum system and turn it back into a classical bit string.

7 Training results

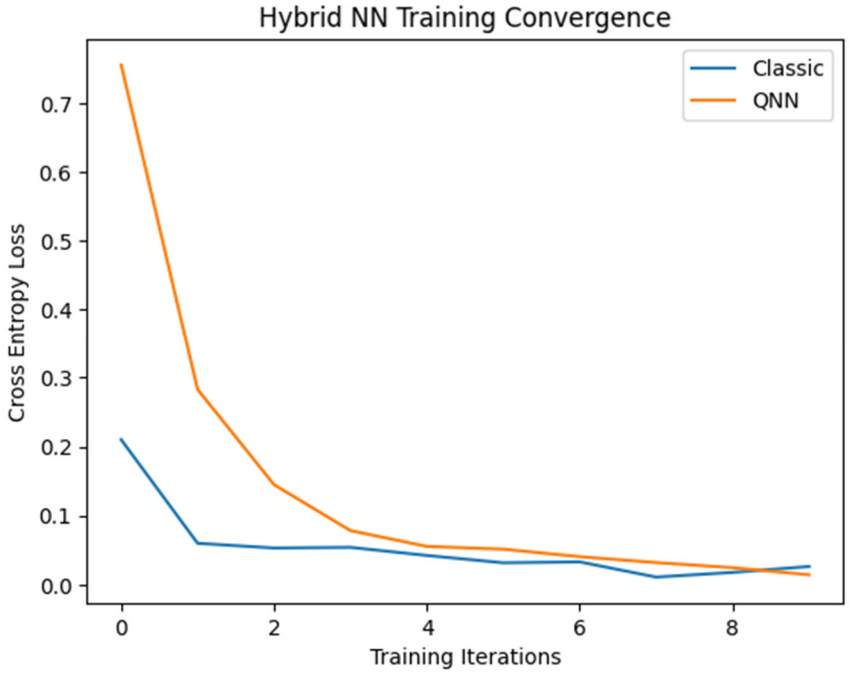

Fig.7: Training curves of both, the classical CNN and the hybrid QNN.

© 2023 Copyright by DYNAmore GmbH, an Ansys Company

[Fig.7] shows that, contrary to expectations, the purely classical network converges faster than the hybrid one. However, from the third epoch onwards, the loss values of the two are almost identical. This will not change for the remaining training of a total of ten epochs (compare *[Fig.8]* and *[Fig.9]*).

```
Training [10%]  Loss: 0.2102 Time needed: 0.86 seconds
Training [20%]  Loss: 0.0600 Time needed: 0.85 seconds
Training [30%]  Loss: 0.0532 Time needed: 0.82 seconds
Training [40%]  Loss: 0.0542 Time needed: 0.88 seconds
Training [50%]  Loss: 0.0424 Time needed: 0.86 seconds
Training [60%]  Loss: 0.0318 Time needed: 0.85 seconds
Training [70%]  Loss: 0.0331 Time needed: 0.96 seconds
Training [80%]  Loss: 0.0111 Time needed: 0.85 seconds
Training [90%]  Loss: 0.0178 Time needed: 0.87 seconds
Training [100%] Loss: 0.0264 Time needed: 0.85 seconds
```

Fig.8: Training on the pure classical CNN with the required times.

```
Training [10%]  Loss: 0.7554 Time needed: 25.52 seconds
Training [20%]  Loss: 0.2830 Time needed: 25.68 seconds
Training [30%]  Loss: 0.1450 Time needed: 25.00 seconds
Training [40%]  Loss: 0.0784 Time needed: 26.03 seconds
Training [50%]  Loss: 0.0555 Time needed: 25.02 seconds
Training [60%]  Loss: 0.0515 Time needed: 24.45 seconds
Training [70%]  Loss: 0.0406 Time needed: 24.86 seconds
Training [80%]  Loss: 0.0321 Time needed: 24.50 seconds
Training [90%]  Loss: 0.0247 Time needed: 24.67 seconds
Training [100%] Loss: 0.0144 Time needed: 24.31 seconds
```

Fig.9: Training on the hybrid QNN with the required times.

8 Results on evaluation datasets

Results on the accuracy of the classification task were almost identical (see *[Fig.10]*) however the time needed by the hybrid QNN was much higher. We have almost identical results on the anomaly detection task (compare *[Fig.11]* and *[Fig.12]*).

```
Performance on Classic:
        Loss: 0.0439
        Accuracy: 98.5%
Performance on QNN:
        Loss: 0.1047
        Accuracy: 99.1%
```

Fig.10: Classification results on the evaluation dataset.

© 2023 Copyright by DYNAmore GmbH, an Ansys Company

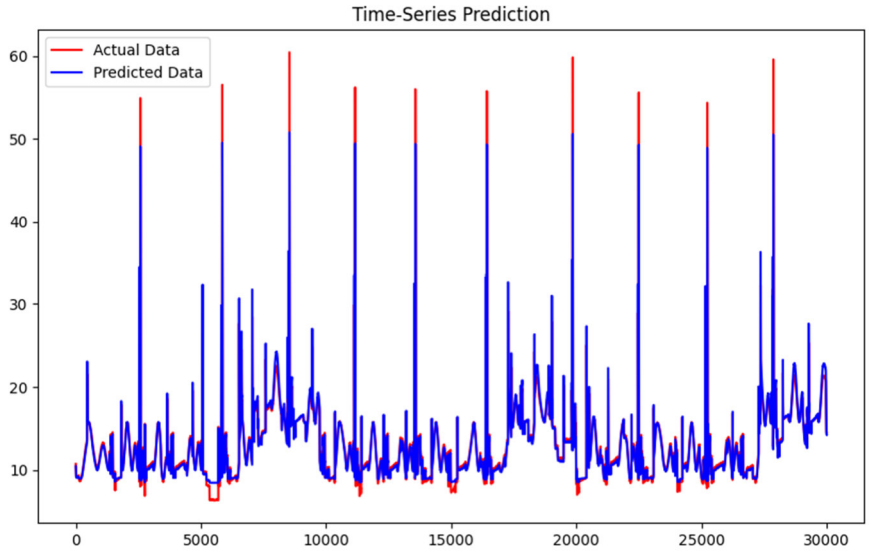

Fig.11: Anomaly detection results of the classical ML on the evaluation dataset.

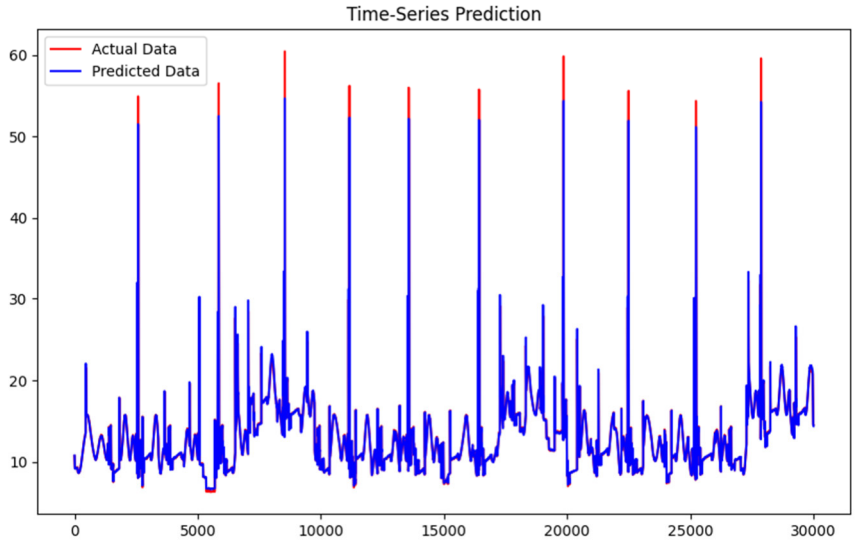

Fig.12: Anomaly detection results of the hybrid QML on the evaluation dataset.

The performance of the complete algorithm is being discussed and the compute time of the data encoding process is being evaluated for different numbers of qubits. In the evaluation process, not only the results are considered, but also the formula for data encoding is taken into account. Using the qiskit library we encode the original data using a “ZZFeatureMap” which is based on a “PauliFeatureMap”. It’s a so called “angle encoding” method. The following section is citated from [5] to visualize the mathematics behind the data encoding.
The “PauliFeatureMap” is a Pauli Expansion circuit that will transform our data, $\vec{\mathrm{x}} \in \mathrm{R}^{\mathrm{n}}$, given by the classical machine learning network, where n is the “feature-dimension“ (that is equal to the number of qubits used), into the quantum state

$$U_{\Phi(\vec{x})} = \exp\left(i \sum_{S \in \mathcal{I}} \phi_S(\vec{x}) \prod_{i \in S} P_i\right)$$

S is a set of qubit indices that describes the connections in the feature map, I is a set containing all these index sets, and $P_i \in \{I, X, Y, Z\}$. Per default the data-mapping ϕ_S is

© 2023 Copyright by DYNAmore GmbH, an Ansys Company

$$\phi_S(\vec{x}) = \{ \prod_{j \in S} \begin{matrix} x_i \text{ if } S = i \\ (\pi - x_j) \text{ if } |S| > 1 \end{matrix}$$

As we tested this on the simulator, the calculation of the quantum states is very complex. That's why the algorithm is slow like shown in *[Fig.9].*
We also ran tests on a physical quantum computer by IBM using the processor "Falcon r4T" with five qubits. The "ZZFeatureMap" we used, is fully entangling the qubits. By this, the quantum computer can encode the different inputs at the same time, which our simulator on the classical computer can't. So the quantum computer should be able to encode the data pretty fast. However, what we observed was that on the quantum computer, the time needed to fulfil a task, was also higher than the pure classical algorithm. We think that the problem here is the overhead. Our data needs to be read out of the GPU, stored in the RAM or a CPU register, and then be sent over through the internet to the quantum computer. After that we'll receive the conclusion and need to get it back into the GPU.
Another problem is that on a real quantum computer we get quantum errors leading to worse results in accuracy. Newest researches show that encoding logical qubits within many physical qubits enhances the protection against quantum errors [7].

9 Conclusion

In our experiments we only used problems that aren't highly complex. To truly outperform a classical machine learning algorithm, we need problems that are more complex. A problem with higher complexity often comes with more features. The problem of the data encoding we used is that, with an angle encoding method you need one qubit per feature. The "free to use" IBM quantum computers only have five qubits. So it's just about enough to train on the "Iris" dataset which contains data on three different species of iris.
To tackle classification tasks with more features you either need more qubits, or you use another type of encoding. A method to encode more features with the same amount of qubits could be a so-called "amplitude encoding" process, with which you are able to store 2^n features, where n is the number of qubits used [8].

10 Literature

[1] Russell, S.J.: "Artificial intelligence a modern approach", third edition, 2010.
[2] Griffiths & Schroeter, D. J. & D. F.: "Introduction to quantum mechanics", third edition, 2018
[3] Schuld & Petruccione, M. & F.: "Supervised learning with quantum computers", Vol. 17, 2018
[4] Caro & others, M. C..: "Encoding-dependent generalization bounds for parametrized quantum circuits", Vol. 5, 2021, 7 - 8
[5] IBM: "Qiskit Pauli Feature Map", https://qiskit.org/documentation/stubs/qiskit.circuit.library.PauliFeatureMap.html#qiskit.circuit.library.PauliFeatureMap, 2023, last accessed on 2023-06-05
[6] IBM: "Quantum Computing", https://quantum-computing.ibm.com/, 2023, last accessed on 2023-08-25
[7] Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 2023, 614. Jg., Nr. 7949, S. 676-681.
[8] HAVLÍČEK, Vojtěch, et al. Supervised learning with quantum-enhanced feature spaces. *Nature*, 2019, 567. Jg., Nr. 7747, S. 209-212.

© 2023 Copyright by DYNAmore GmbH, an Ansys Company