

# Software Environments for Quantum Machine Learning

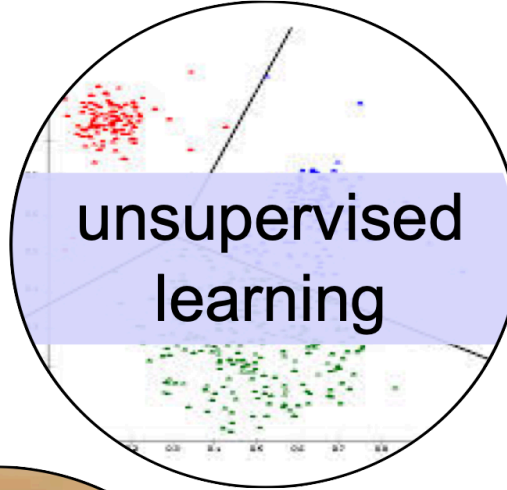
---

Dániel Nagy



# Types of machine learning

Need a lot of  
labeled data



Learning from  
unlabeled data




Learning from  
interactions  
with an  
environment

# Why quantum machine learning?

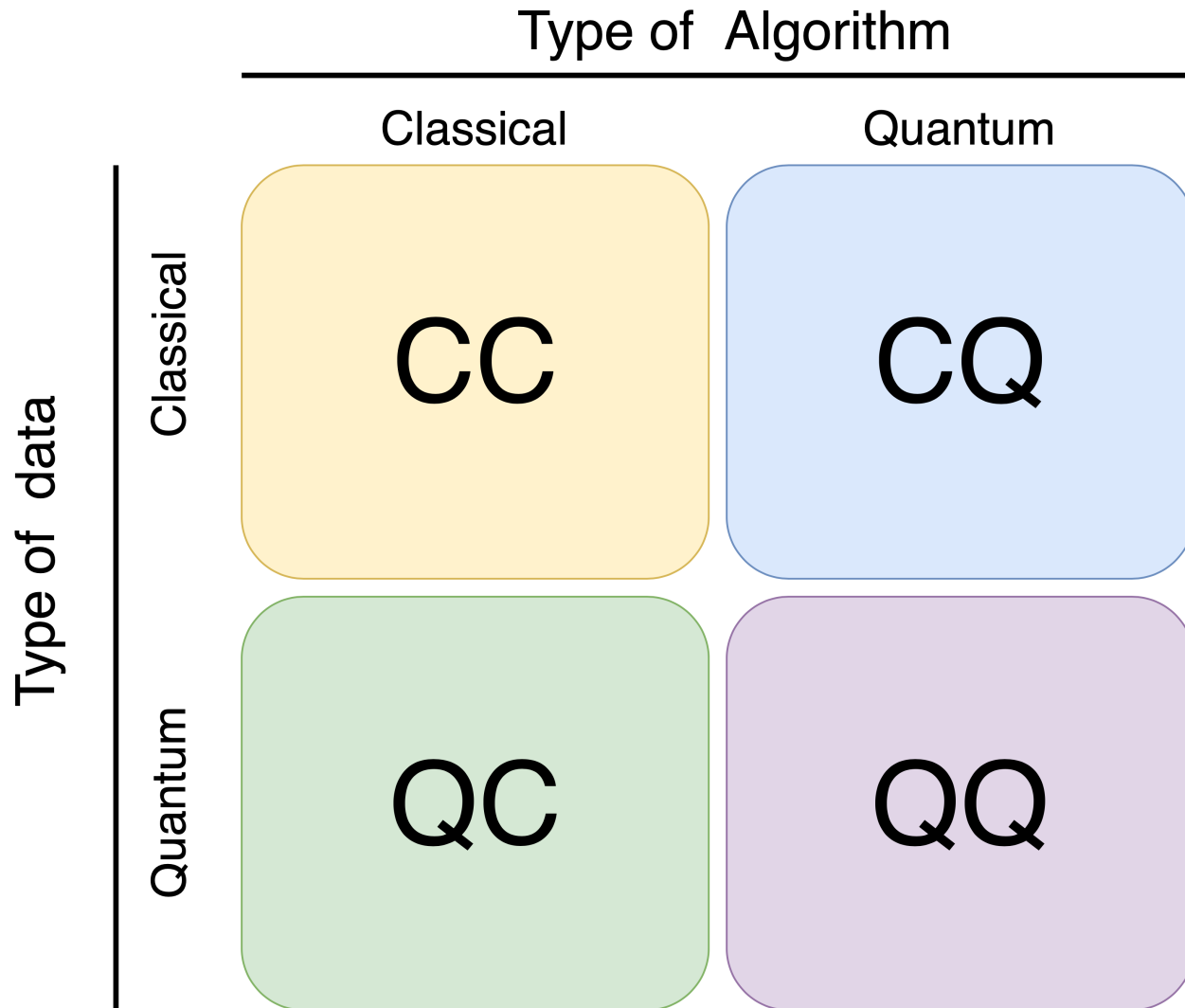
- ML is linear algebra + nonlinearities
- QM is linear algebra + measurements
- ML deals with probability distributions, which naturally appear in QM.



# Why quantum machine learning?

- The dimensionality of the Hilbert-space scales exponentially with the number of qubits.
  - Quantum circuits can generate probability distributions that are impossible to generate with classical computers.
  - They may be able to learn distributions that would be infeasible on classical computers.
- 

# Classifying QML algorithms by quantumness



- CC: every classical ML algorithm
- CQ: variational quantum regression, variational quantum classification e.g. q-SVM [[arxiv](#)]
- QC: ML assisted quantum error correction [[arxiv](#)]
- QQ: Quantum Autoencoders to Denoise Quantum Data [[PRL](#)], VQC for state tomography [[arxiv](#)]

# Supervised Learning – Gradient descent

$$x \longrightarrow y_{pred} = f(x; \theta)$$

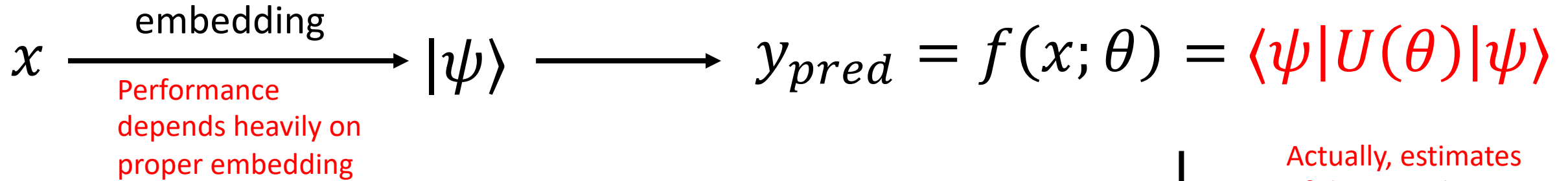
The model is represented by an almost everywhere differentiable function,  $f(x; \theta)$



$$y_{true} \longrightarrow L = \mathcal{L}(y_{pred}, y_{true})$$

$$\theta_{t+1} = \theta_t - \eta_t \frac{\partial L}{\partial \theta}$$

# Quantum Supervised Learning – Gradient descent



The model is represented by a unitary,  $U(\theta)$ .

Actually, estimates of the expval

$$y_{true} \longrightarrow L = \mathcal{L}(y_{pred}, y_{true})$$

$$\theta_{t+1} = \theta_t - \eta_t \frac{\partial L}{\partial \theta}$$

# How to calculate the gradient of a quantum node?

- Parameter-shift rule:

- If  $f(\mu)$  is a quantum node, then

$$\partial_{\mu}f(\mu) = c[f(\mu + s) - f(\mu - s)]$$

(c and s are finite parameters from a lookup table)

- Finite difference method

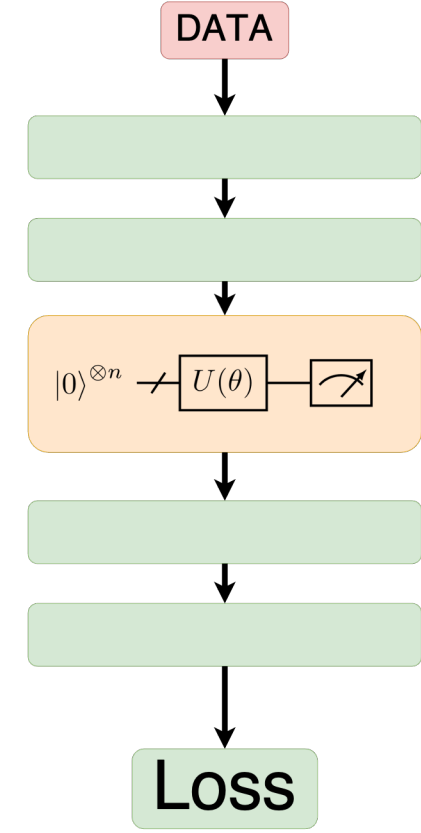
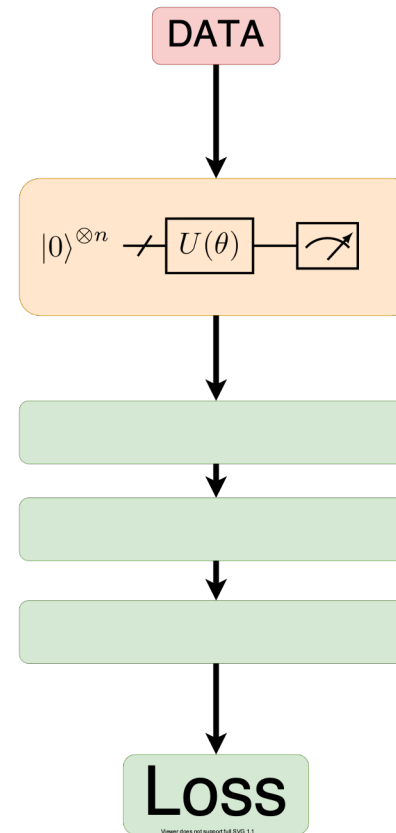
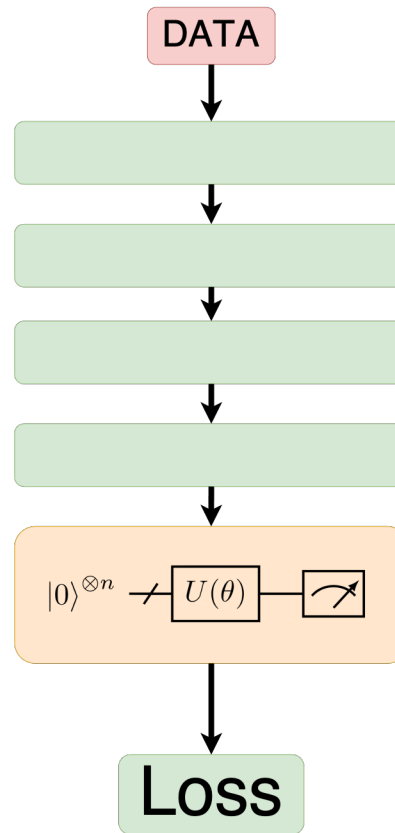
- If  $f(\mu)$  is a quantum node, then

$$\partial_{\mu}f(\mu) \approx \frac{f\left(\mu + \frac{1}{2}\Delta\mu\right) - f\left(\mu - \frac{1}{2}\Delta\mu\right)}{\Delta\mu}$$



# What do we have now? Hybrid devices

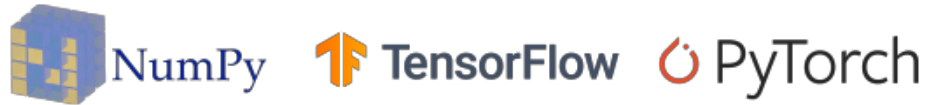
- High performance classical computers
- Small and noisy quantum devices
- => Build hybrid quantum-classical algorithms
- Compose models from both quantum and classical nodes



# Software packages for QML

---

*Machine learning interfaces*



P E N N Y L A N E



*Quantum hardware and simulators*



**TensorFlow** Quantum



# TensorFlow Quantum: A Software Framework for Quantum Machine Learning

Michael Broughton,<sup>1,9,\*</sup> Guillaume Verdon,<sup>1,2,8,10,†</sup> Trevor McCourt,<sup>1,11</sup> Antonio J. Martinez,<sup>1,8,12</sup>  
Jae Hyeon Yoo,<sup>3</sup> Sergei V. Isakov,<sup>4</sup> Philip Massey,<sup>5</sup> Murphy Yuezhen Niu,<sup>1</sup> Ramin Halavati,<sup>6</sup>  
Evan Peters,<sup>8,10,13</sup> Martin Leib,<sup>14</sup> Andrea Skolik,<sup>14,15,16,17</sup> Michael Streif,<sup>14,16,17,18</sup> David Von Dollen,<sup>19</sup>  
Jarrod R. McClean,<sup>1</sup> Sergio Boixo,<sup>1</sup> Dave Bacon,<sup>7</sup> Alan K. Ho,<sup>1</sup> Hartmut Neven,<sup>1</sup> and Masoud Mohseni<sup>1,‡</sup>

---

## PennyLane: Automatic differentiation of hybrid quantum-classical computations

Ville Bergholm,<sup>1</sup> Josh Izaac,<sup>1</sup> Maria Schuld,<sup>1</sup> Christian Gogolin,<sup>1</sup> M. Sohaib Alam,<sup>2</sup>  
Shahnawaz Ahmed,<sup>3</sup> Juan Miguel Arrazola,<sup>1</sup> Carsten Blank,<sup>4</sup> Alain Delgado,<sup>1</sup> Soran  
Jahangiri,<sup>1</sup> Keri McKiernan,<sup>2</sup> Johannes Jakob Meyer,<sup>5</sup> Zeyue Niu,<sup>1</sup> Antal Száva,<sup>1</sup> and  
Nathan Killoran<sup>1</sup>

	 <b>TensorFlow Quantum</b>	 <b>P E N N Y L A N E</b>
<b>Supported ML frameworks</b>	TensorFlow	PyTorch, TensorFlow, autograd
<b>Supported quantum backends</b>	Google*, Alpine Quantum Technologies*	Rigetti, IBM, Xanadu*, Google*
<b>Publicly available QPU</b>	✗	✓
<b>Differentiation techniques</b>	Parameter shift, forward & central difference, linear combination, stochastic generator, custom quantum gradients	Parameter shift, finite difference, classical backpropagation
<b>Available simulators</b>	Custom high-performance simulator and cirq	Custom qubit and qumode simulators and simulator plugins for Forest, cirq, Qiskit, ProjectQ, Q# and Strawberry Fields
<b>Advantages</b>	<p>Deeply integrated through new low-level C++ TensorFlow operations</p> <p>Might be better for workflows heavy on quantum data due to support for quantum datasets (i.e. quantum circuits)</p>	<p>Agnostic approach allows you to program in your ML + quantum framework of choice</p> <p>Ability to combine multiple quantum backends into one model (e.g. one layer on Rigetti and one on a CV** backend)</p>
<b>Pro tip from the maintainers</b>	Avoid the parameter shift rule and differentiate using the stochastic generator or finite difference for extra performance.	The default simulator was not optimized for performance. Try out the <i>qulacs</i> plugin or the tensor network simulator.
<b>TLDR</b>	QML framework for TensorFlow fans with a strong focus on performance	Versatile QML framework with many plugins to various hardware vendors

\*currently no publicly available QPUs \*\*continuous-variable quantum computing



[From QOS foundation](#)

# Defining a quantum node in PennyLane

```
1 import pennylane as qml
2
3 device_simulator = qml.device('default.qubit',
4 wires=2, shots=1000)
5 device_hardware = qml.device('qiskit.ibmq', wires=2,
6 backend='ibmq_16_melbourne')
7
8 @qml.qnode(device_simulator)
9 def quantum_func(x, y):
10     qml.RZ(x, wires=0)
11     qml.CNOT(wires=[0,1])
12     qml.RY(y, wires=1)
13     return qml.expval(qml.PauliZ(1))
14
15 quantum_func(0.5, 0.45) # Result: 0.9004471023526768
```

# Interfacing with Pytorch

```
1 import torch
2 import pennylane as qml
3
4 device_simulator = qml.device('default.qubit', wires=2,
5     shots=1000)
6 @qml.qnode(device_simulator, interface='torch')
7 def quantum_func(x, y, theta, phi):
8     qml.RZ(x, wires=0)
9     qml.RZ(theta, wires=1)
10    qml.RX(phi, wires=1)
11    qml.CNOT(wires=[0,1])
12    qml.RY(y, wires=1)
13    return qml.expval(qml.PauliZ(1))
14
15 theta = torch.tensor(0.99)
16 phi = torch.tensor(1.12)
17 quantum_func(0.5, 0.45, theta, phi) # Result: tensor(0.3923,
18     dtype=torch.float64)
```

# Calculating Jacobians

```
1 import numpy as np
2 import pennylane as qml
3 dev = qml.device('default.qubit', wires=2)
4
5 @qml.qnode(dev)
6 def circuit(params):
7     qml.Hadamard(wires=0)
8     qml.CNOT(wires=[0, 1])
9     qml.RX(params[0], wires=0)
10    qml.RY(params[1], wires=1)
11    qml.CNOT(wires=[0, 1])
12    return qml.expval(qml.PauliY(0)), qml.expval(qml.PauliZ(1))
13
14 # Calculate the gradient of the circuit
15 J = qml.jacobian(circuit)
16
17 params = np.array([np.pi/2, 0.2])
18
19 J(params)
20
21 # Output:
22 # array([[ 0.          , -0.98006658],
23 #        [-0.98006658,  0.          ]])
```

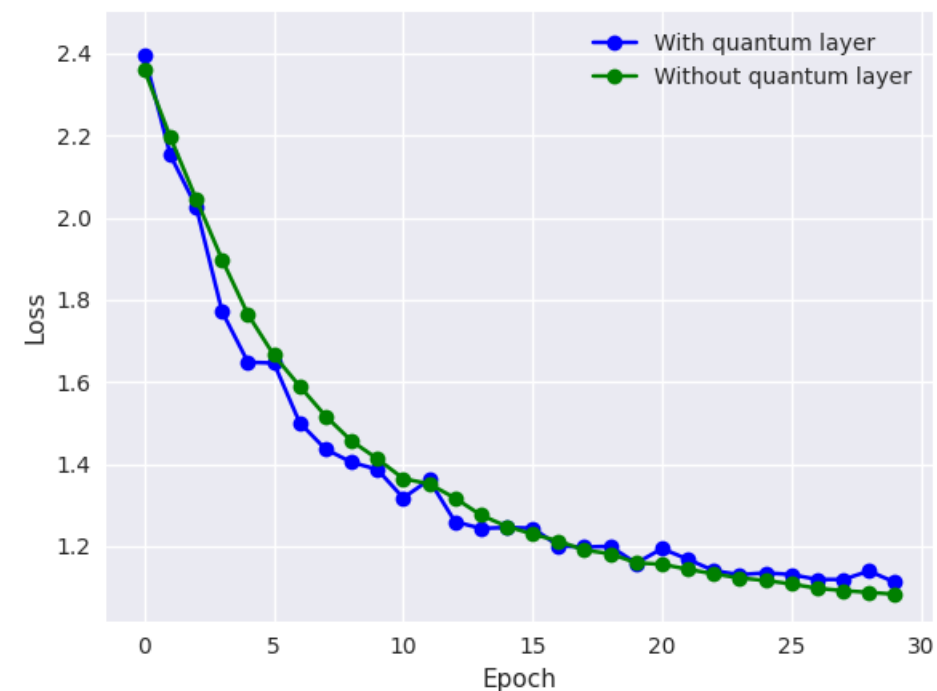
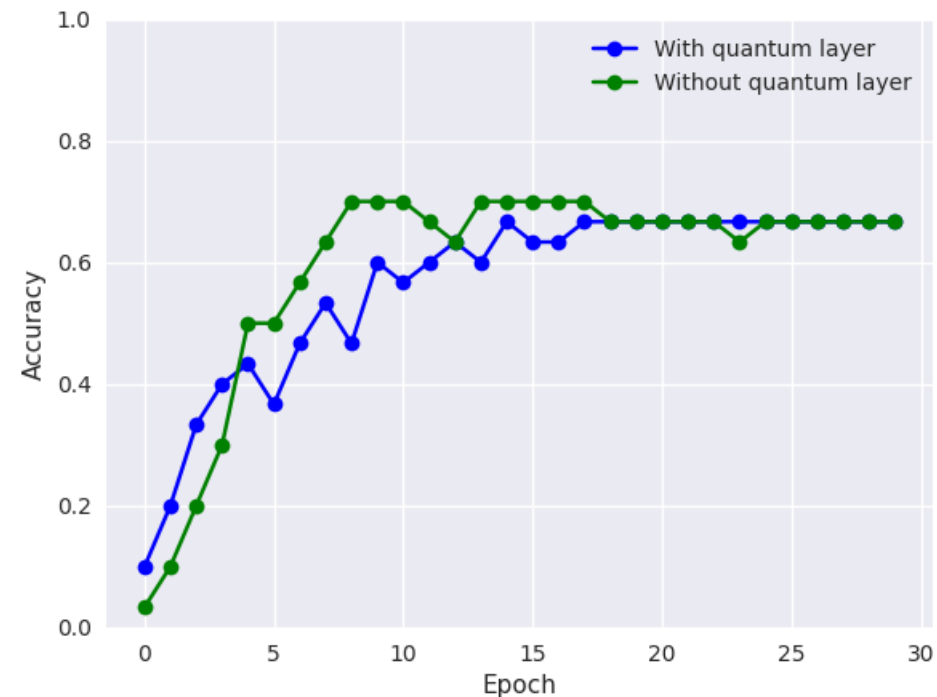
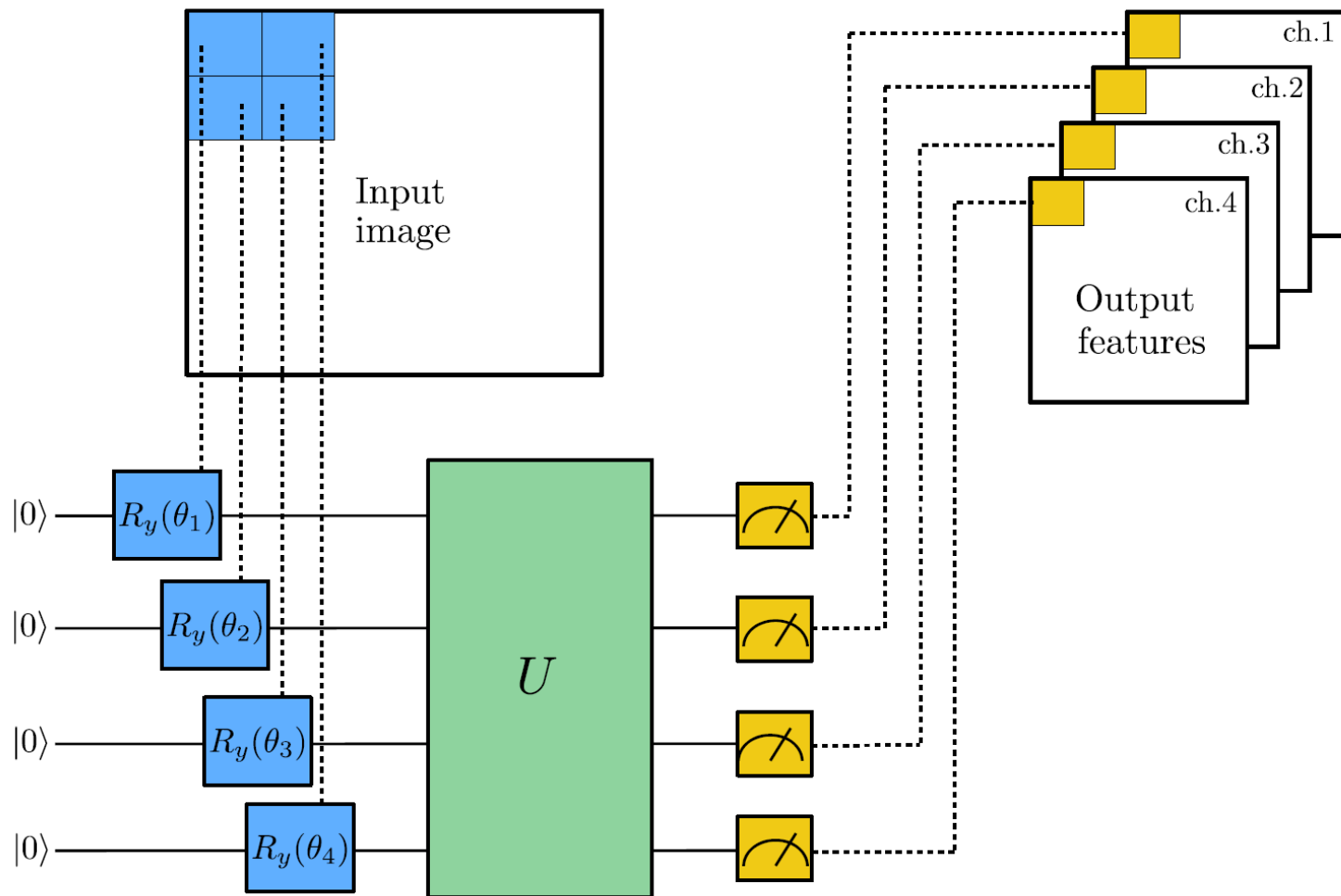


# Using quantum layers in Pytorch models

```
1 import torch
2 import pennylane as qml
3 dev = qml.device("default.qubit", wires=2)
4
5 @qml.qnode(dev)
6 def qnode(inputs, weights_0, weight_1):
7     qml.RX(inputs[0], wires=0)
8     qml.RX(inputs[1], wires=1)
9     qml.Rot(*weights_0, wires=0)
10    qml.RY(weight_1, wires=1)
11    qml.CNOT(wires=[0, 1])
12    return qml.expval(qml.PauliZ(0)), qml.expval(qml.PauliZ(1))
13
14 weight_shapes = {"weights_0": 3, "weight_1": 1}
15 qlayer = qml.qnn.TorchLayer(qnode, weight_shapes)
16
17 clayer = torch.nn.Linear(2, 2)
18 model = torch.nn.Sequential(qlayer, clayer)
19 model(torch.tensor([0.99, 1.21]))
20 # Output: tensor([-0.2455,  0.7882], grad_fn=<AddBackward0>)
```



# Example: Quantum Convolutional networks on the MNIST dataset



# Challenges and open questions

---

Available hardware devices are small and noisy, simulation is slow and computationally heavy

---

Strategies for guessing the right ansatz circuit?

---

Can we prove that QML is better than ML?

---

The power of ML relies on nonlinearities between linear layers, but quantum layers are always linear only (even if the Hilbert space is very high dimensional). Should we find a way to add nonlinearities?



Thank You for Your  
attention!

Contact: [nagy.dani@wigner.hu](mailto:nagy.dani@wigner.hu)