Quantum Computing and Quantum Machine Learning

FYS5419/9419, Quantum computing and quantum machine learning, University of Oslo, Norway

Spring semester 2024, deaadline June 7

Possible paths for project 2

We discuss here three paths for the second project. Tentative deadline June 7.

The report should also be styled as a scientific report. The guidelines we have established at `https://github.com/CompPhysics/AdvancedMachineLearning/tree/main/doc/Projects/EvaluationGrading` could be useful in structuring your report. We have also added a lecture set by Anne Ruimy (director of EDP journals) on how to write effective titles and abstracts. See `https://github.com/CompPhysics/AdvancedMachineLearning/tree/main/doc/Projects/WritingAbstracts` for these lectures. Finally, at `https://github.com/CompPhysics/AdvancedMachineLearning/tree/main/doc/Projects/2023/ProjectExamples` you can find different examples of previous reports. See also the literature suggestions below.

We would like to suggest three different paths (**select only one of these**). They are

1. Implementing quantum Fourier Transforms (QFTs), the phase estimation algorithm and if time allows (optional Shor's algorithm).
2. Quantum Machine Learning project
3. Implementing Quantum Boltzmann machines

These projects are described here.

Alternative one, QFTs

This variant of the second project follows closely Robert Hundt's text, sections 6.1-6.6

Project 2 a): Basic mathematics of Quantum Fourier Transform. The Quantum Fourier Transform (QFT) has mathematically the same form as a discrete Fourier transform but the notation is generally different. We generally compute the quantum Fourier transform on a set of orthonormal basis state vectors $|0\rangle, |1\rangle, ..., |N-1\rangle$. The linear operator defining the transform is given by the action on basis states

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle.$$

This can be written on arbitrary states,

$$\sum_{j=0}^{N-1} x_j |j\rangle \mapsto \sum_{k=0}^{N-1} y_k |k\rangle$$

where each amplitude y_k is the discrete Fourier transform of x_j.

Show that the first equation is defined in terms of a unitary transformation. Find the matrix elements for such a unitary transformation for one-, two- and three-qubit.

Explicitely compute the Fourier transform for N qubits with the initial state $|00\ldots0\rangle$.

Project 2 b): More basic mathematics of Quantum Fourier Transform. The Quantum Fourier transform is normally written out in terms of various controlled-rotational gates R_k. Write these in terms of a decomposition into single qubit and CNOT gates. Section 6.2.4 of Hundt for the two-qubit case gives an example.

Write thereafter a quantum circuit to perform the inverse QFT. Discuss also the pros and cons of performing QFTs compared with for example a classical discrete Fourier transform. Hint: here you can think in terms for the number of floating point operations. Furthermore, keep in mind that due to the binary representation, one can achieve only a given precision.

Project 2 c): Implementing the Quantum Fourier Transform for one, two and three qubits. Here we follow Hundt's section 6.2. Write a code (without using Qiskit or pennylane or similar software) which implements the QFT for one, two and three qubits. Make sure that the inverse equation (unitarity through equation 6.2 in Hundt) is obeyed. Compare your code with for example the QFT functionality of Qiskit. Compare also your QFT code with the results of the explicit matrix-vector multiplications for two- and three-qubits.

Project 2 d): Generalize the code to an arbitrary number of qubits. Now we generalize our code to an arbitrary number of qubits. Test your code against the Fourier transform from 2a) for N qubits with the initial state $|00\ldots0\rangle$. Test also the code for unitarity.

Project 2 e): Implement the phase estimation algorithm for one and two qubits. Write thereafter your own code for the phase estimation algorithm discussed in section 6.4 of Hundt. Derive first the phase estimation equations for the one- and two-qubit cases and find the phase λ. Write thereafter a code which implements the phase estimation algorithm for an arbitrary number of qubits in the register. Extract the phase and discuss the accuracy of the phase estimation algorithm, see for example Nielsen and Chuang, **Quantum Computation and Quantum Information**, Oxford, 2000, section 5.2.

Project 2 f): Bringing back the simple Hamiltonians from project 1. We will now compute the eigenvalues of the simpler matrices from project 1 using the phase estimation algorithm. In project 1 we defined a symmetric matrix $H \in \mathbb{R}^{2\times 2}$

$$H = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix},$$

We let $H = H_0 + H_I$, where

$$H_0 = \begin{bmatrix} E_1 & 0 \\ 0 & E_2 \end{bmatrix},$$

is a diagonal matrix. Similarly,

$$H_I = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix},$$

where V_{ij} represent various interaction matrix elements. We can view H_0 as the non-interacting solution

$$H_0|0\rangle = E_1|0\rangle, \tag{1}$$

and

$$H_0|1\rangle = E_2|1\rangle, \tag{2}$$

where we have defined the orthogonal computational one-qubit basis states $|0\rangle$ and $|1\rangle$.

The two-qubit Hamiltonian was defined via a 4×4 Hamiltonian matrix. That system could be thought of as composed of two subsystems A and B. Each subsystem has computational basis states

$$|0\rangle_{\mathrm{A,B}} = \begin{bmatrix} 1 & 0 \end{bmatrix}^T \qquad |1\rangle_{\mathrm{A,B}} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T.$$

The subsystems could represent single particles or composite many-particle systems of a given symmetry. This leads to the many-body computational basis states

$$|00\rangle = |0\rangle_{\mathrm{A}} \otimes |0\rangle_{\mathrm{B}} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T,$$

and

$$|01\rangle = |0\rangle_{\mathrm{A}} \otimes |1\rangle_{\mathrm{B}} = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T,$$

and

$$|10\rangle = |1\rangle_{\mathrm{A}} \otimes |0\rangle_{\mathrm{B}} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T,$$

and finally

$$|11\rangle = |1\rangle_{\mathrm{A}} \otimes |1\rangle_{\mathrm{B}} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T.$$

These computational basis states define also the eigenstates of the non-interacting Hamiltonian

$$H_0|00\rangle = \epsilon_{00}|00\rangle,$$

$$H_0|10\rangle = \epsilon_{10}|10\rangle,$$

$$H_0|01\rangle = \epsilon_{01}|01\rangle,$$

and

$$H_0|11\rangle = \epsilon_{11}|11\rangle.$$

The interacting part of the Hamiltonian H_{I} is given by the tensor product of two σ_x and σ_z matrices, respectively, that is

$$H_{\mathrm{I}} = H_x \sigma_x \otimes \sigma_x + H_z \sigma_z \otimes \sigma_z,$$

where H_x and H_z are interaction strength parameters. Our final Hamiltonian matrix is given by

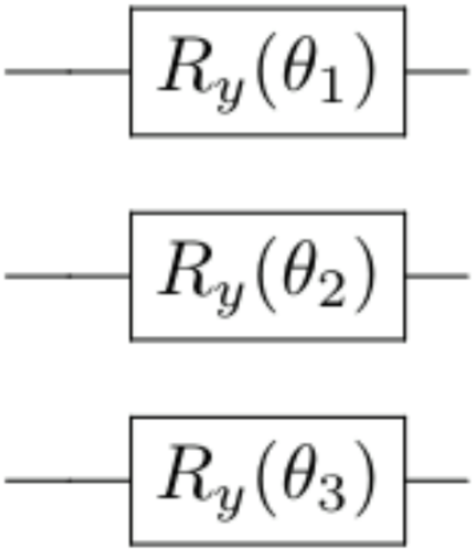
$$\boldsymbol{H} = \begin{bmatrix} \epsilon_{00} + H_z & 0 & 0 & H_x \\ 0 & \epsilon_{10} - H_z & H_x & 0 \\ 0 & H_x & \epsilon_{01} - H_z & 0 \\ H_x & 0 & 0 & \epsilon_{11} + H_z \end{bmatrix}.$$

Compute the eigenvalues of these two matrices using the phase estimation algorithm. Compare the results with those from the Variational Quantum Eigensolver from project 1. Discuss the results with pros and cons of the two methods.

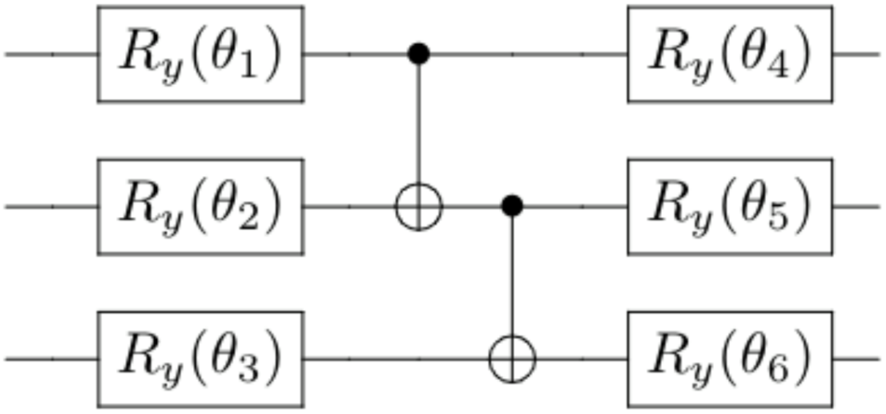
Project 2 g): These part is optional. If you have time and would to explore Shor's algorithm for factoring, sections 6.5 and 6.6 of Hundt contain an in depth discussion on how to implement this famous algorithm.

Alternative two, Quantum Machine Learning

Why do people care about "Quantum Machine Learning", or "Quantum Computing" in general? A common observation that motivates quantum computing, although it does not tell the whole story, is the striking fact than quantum information of a quantum system is in a sense larger than the classical information of a comparable classical system. In the case of bits and qubits, n classical bits allows you to store the value of n boolean variables $a \in [0, 1]$, whereas you need 2^n amplitudes $\alpha \in \mathbb{C}$ to describe the state of n quantum bits, or qubits. In other words, the (Hilbert)space in which the qubits live in exponentially bigger than that of the classical counterpart. In the context of parameterized quantum circuits, one may prepare a state in the exponentially large Hilbert space by applying some simple quantum operations to a set qubits that are all initialized in the zero-state.



By varying $\theta_i \in [0, 2\pi]$, one can attain different states. A crucial thing to note is that we are not able to prepare an arbitrary state using this ansatz as it much too constrained and simple: The exponentially large space is not available to us. One might try to remedy this by introducing a more complex ansatz:



We now have an ansatz whose number of operations scales polynomially in the number of qubits(in this case quadratic), and the added complexity enables us to reach a greater part of Hilbert space. Still, it is only a polynomially large part of it, which is vanishingly small compared to its full size. To have access to the whole space, one would in fact need to perform exponentially many operations, which is practically intractable to do (Nielsen, 4.5.4).

Does this mean that the supposed power of quantum computation is practically inaccessible to us? No. Even though we only reach a very small part of Hilbert space using "cheap" ansatzes, the states we do reach might be very useful for solving a particular problem. Moreover, these states might be classically intractable to compute (source to come), meaning the information they represent is not practical to determine using classical computers. They are however efficiently prepared using quantum computers, as the number of quantum operations needed to be applied is by construction only polynomial.

How can one leverage this in a machine learning setting? A common approach is to use variants of the previous ansatzes to encode features to qubits by performing rotations, thus embedding the data in a high dimensional quantum state. Subsequent parameterized operations on the qubits then applies powerful transformations to the embedded data in a high dimensional space. Such methods are often described as quantum kernel methods, because of their similarity to classical kernel methods in machine learning.

For this project, you will perform quantum machine learning on the two first targets of Scikit learn iris data set. The data can be obtained the following way

```
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
x = iris.data
y = iris.target
idx = np.where(y < 2) # we only take the first two targets.

x = x[idx,:]
y = y[idx]
```

x is the feature matrix and y are the targets.

Project 2 a): Encoding the Data Into a Quantum State. For this task you will consider a simple way of encoding a randomly generated data set sample into a quantum state:

```
import qiskit as qk
import numpy as np
np.random.seed(42)

p = 2 #number of features
```

```
data_register = qk.QuantumRegister(p)
classical_register = qk.ClassicalRegister(1)

circuit = qk.QuantumCircuit(data_register, classical_register)

sample = np.random.uniform(size=p)
target = np.random.uniform(size=1)

for feature_idx in range(p):
    circuit.h(data_register[feature_idx])
    circuit.rz(2*np.pi*sample[feature_idx],data_register[feature_idx])

print(circuit)
```

The above code shows how a randomly generated data sample of $p = 2$ features are encoded into a quantum state on two qubits utilizing Qiskit. Each feature is encoded into a respective qubit utilizing a $R_y(\theta)$ gate. The features are scaled with 2π to represent rotation angles (the $R_y(\theta)$ gate performs a rotation). The classical register will be used later for storing the measured value of the circuit. print(circuit) can be utilized at any point to see what the circuit looks like.

Your task is to get familiar with the functionality utilized in the above example and implement your own function to encode the features of the iris data set to a quantum state.

Project 2 b): Processing the Encoded Data with Parameterized Gates. After the quantum state has been encoded with the information of a data set sample, one needs extend the circuit with operations that process the state in a way that allows us to infer the target data. This can be done by introducing quantum gates that are dependant on learnable parameters $\boldsymbol{\theta}$. We will do this in a similar fashion as for the encoding of the features:

```
n_params = 4
theta = 2*np.pi*np.random.uniform(size=n_params)

circuit.ry(theta[0], data_register[0])
circuit.ry(theta[1], data_register[1])

circuit.cx(data_register[0], data_register[1])

circuit.ry(theta[2], data_register[0])
circuit.ry(theta[3], data_register[1])

circuit.cx(data_register[0], data_register[1])
```

```
print(circuit)
```

The above parameterization of the quantum state is what we will refer to as the 'ansatz'. Your task is again to familiarize yourself with the functionality utilized in the above example and implement your own ansatz to be utilized together with the features of the iris data set. The number of learnable parameters 'theta' should be arbitrary.

Project 2c):Measuring the Quantum State and Making Inference. The next step is to generate a prediction from our quantum machine learning model. This is done by performing a measurement on the quantum state:

```
circuit.measure(data_register[-1],classical_register[0])
shots=1000

job = qk.execute(circuit,
                backend=qk.Aer.get_backend('qasm_simulator'),
                shots=shots,
                seed_simulator=42
                )
results = job.result()
results = results.get_counts(circuit)

prediction = 0
for key,value in results.items():
    if key == '1':
        prediction += value
prediction/=shots
print('Prediction:',prediction,'Target:',target[0])
```

```
    Prediction: 0.285 Target: 0.7319939418114051
```

In the above example, we are first applying a measurement operation on the final qubit in the circuit, and we are interpreting our prediction as the probability that this qubit is in the $|1\rangle$ state. Make sure all the steps in the example are understood.

Implement your own function that generates a prediction by measuring one of the qubits.

Project 2d): Putting it all together. Now it is time to put together all of the above steps. Ideally, you should make a class or a function that given a feature matrix of n samples and an arbitrary number of model parameters, returns a vector of n outputs. For example:

```
n = 100 #number of samples
p = 4 #number of features
theta = np.random.uniform(size=20) #array of model parameters
X = np.random.uniform(size=(n,p)) #design matrix
y_pred = model(X,theta) #prediction, shape (n)
```

We will now deal with how to train the model:

Project 2e): Parameter Shift-Rule and Calculating the Analytical Gradient. Since the model with random initial parameters is no good for inference, we need to optimize the parameters in order to yield good results, as is the usual with machine learning.

Since we are dealing with classification, we will use cross-entropy as the loss function

$$L = -\sum_{i=1}^{n} y_i \ln f(x_i; \boldsymbol{\theta}),$$

where y_i are the target labels, and $f(\boldsymbol{x}_i; \boldsymbol{\theta})$ is the output of our model for a given sample $\boldsymbol{x}_i$ and parameterization $\boldsymbol{\theta}$). We calculate the gradiant by taking the derivative of the loss with respect to the parameters

$$\frac{\partial}{\partial \boldsymbol{\theta}_k} L = \sum_{i=1}^{n} \frac{f_i - y_i}{f_i(1-f_i)} \frac{\partial}{\partial \boldsymbol{\theta}_k} f_i,$$

where $f_i = f(x_i; \boldsymbol{\theta})$ for clarity. The only term we do not know how to calculate is $\frac{\partial}{\partial \boldsymbol{\theta}_k} f(x_i; \boldsymbol{\theta})$, but it turns out there is a simple trick to do this, the so-called parameter shift-rule https://pennylane.ai/qml/glossary/parameter_shift/. To calculate the derivative of the model output, we need to evaluate the model twice with the respective parameter shifted by a value $\frac{\pi}{2}$ up and down. The two resulting outputs are then put together to yield the derivative

$$\frac{\partial f(x_i; \theta_1, \theta_2, \dots, \theta_k)}{\partial \theta_j} = \frac{f(x_i; \theta_1, \theta_2, \dots, \theta_j + \pi/2, \dots, \theta_k) - f(x_i; \theta_1, \theta_2, \dots, \theta_j - \pi/2, \dots, \theta_k)}{2}$$

Train your model by utilizing the Parameter Shift-Rule and some gradient descent algorithm. Compare your results with for example logistic regression.

Regular gradient descent does the job, but it is often outperformed by momentum based optimizers like Adam.

Project 2f): Adding Variations on the Data Encoding and Ansatz. Change/add more gates to the encoder and the parameterized ansatz to produce a more complex model (See for example figure 4 and figure 5 in https://arxiv.org/abs/2011.00027 for inspiration). Train these new models on the iris data set. How do they compare? As a more challenging problem, you may train on the first few features of the Breast Cancer Data as well to see how the more powerful model performs.

```
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
x = data.data #features
y = data.target #targets
```

Alternative three, Quantum Boltzmann Machines

This alternative is a more challenging one and deals with the implementation and study of what are called Variational Quantum Boltzmann machines. It requires familiarity with machine learning and in particular generative models like Boltzmann machines. It is suited to those you who follow either FYS5429/9429 and/or FYS4411/9411. In this case you can combine the second project from one of these courses with the present report for project 2. You can then hand in one single (and identical report) for both courses.

The aim here is to implement the variational Quantum Boltzmann machine (VQBM) for studies of the Ising model. The article we base this is on is

1. C. Zoufal, A. Lucchi, and S. Woerner, **Variational quantum Boltzmann machines**, Quantum Machine Intelligence, Volume 3, article number 7, (2021), see `https://link.springer.com/article/10.1007/s42484-020-00033-7` for full text.

For an implementation, see `https://helda.helsinki.fi/server/api/core/bitstreams/eeadc874-f41c-4795-9149-2cd9a3bfe10d/content`. You should restrict the studies here to some few qubits.

Literature

The following articles and books (with codes) are relevant here:

1. Robert Hundt, Quantum Computing for Programmers, Cambridge 2022, in particular chapter 6

2. Michael A. Nielsen and Isaac L. Chuang, Quantum Computation and Quantum Information, Cambridge 2000, in particular chapter 5.

Introduction to numerical projects

Here follows a brief recipe and recommendation on how to write a report for each project.

- Give a short description of the nature of the problem and the eventual numerical methods you have used.

- Describe the algorithm you have used and/or developed. Here you may find it convenient to use pseudocoding. In many cases you can describe the algorithm in the program itself.
- Include the source code of your program. Comment your program properly.
- If possible, try to find analytic solutions, or known limits in order to test your program when developing the code.
- Include your results either in figure form or in a table. Remember to label your results. All tables and figures should have relevant captions and labels on the axes.
- Try to evaluate the reliabilty and numerical stability/precision of your results. If possible, include a qualitative and/or quantitative discussion of the numerical stability, eventual loss of precision etc.
- Try to give an interpretation of you results in your answers to the problems.
- Critique: if possible include your comments and reflections about the exercise, whether you felt you learnt something, ideas for improvements and other thoughts you've made when solving the exercise. We wish to keep this course at the interactive level and your comments can help us improve it.
- Try to establish a practice where you log your work at the computerlab. You may find such a logbook very handy at later stages in your work, especially when you don't properly remember what a previous test version of your program did. Here you could also record the time spent on solving the exercise, various algorithms you may have tested or other topics which you feel worthy of mentioning.

Format for electronic delivery of report and programs

The preferred format for the report is a PDF file. You can also use DOC or postscript formats or as an ipython notebook file. As programming language we prefer that you choose between C/C++, Fortran2008 or Python. The following prescription should be followed when preparing the report:

- Send us an email in order to hand in your projects with a link to your GitHub/Gitlab repository.
- In your GitHub/GitLab or similar repository, please include a folder which contains selected results. These can be in the form of output from your code for a selected set of runs and input parameters.

Finally, we encourage you to collaborate. Optimal working groups consist of 2-3 students. You can then hand in a common report.