

Project 2, Quantum Machine Learning.

Deadline June 1, Spring 2021

Computational Physics II FYS4411/FYS9411

Department of Physics, University of Oslo, Norway

Apr 13, 2021

Introduction

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)
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
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 θ . 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 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

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

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

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

where $f_i = f(x_i; \theta)$ for clarity. The only term we do not know how to calculate is $\frac{\partial}{\partial \theta_k} f(x_i; \theta)$, but it turns out there is a simple trick to do this, the so-called parameter shift-rule

`citeParameterShift`. 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
```

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 reliability 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 your 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:

- Use canvas to hand in your projects, log in at <http://canvas.uio.no> with your normal UiO username and password.
- Upload **only** the report file! For the source code file(s) you have developed please provide us with your link to your github domain. The report file should include all of your discussions and a list of the codes you have developed. The full version of the codes should be in your github repository.

- In your github 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.
- Still in your github make a folder where you place your codes.
- In this and all later projects, you should include tests (for example unit tests) of your code(s).
- Comments from us on your projects, approval or not, corrections to be made etc can be found under your Devilry domain and are only visible to you and the teachers of the course.

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