CUSTOM QUANTUM DEVICES AND THEIR USE IN A NEURAL NETWORK AT A QUANTUM LEVEL

A RESEARCH PAPER

Sophia Ivanko
Quantum Software Engineer
sofia.ivanko@gmail.com

Peter Farag
Quantum Software Engineer
peterfarag12@gmail.com

October 22, 2023

ABSTRACT

Current quantum computers are expensive and require professional equipment for building it. The price for such computers depends on the amount of qubits but a 2 qubit device in average costs $10000 so the goal of this paper is to show the way of creating an optical quantum computer that is cheap and prove it utility by using the device as a quantum layer in a neural network that analyzes pictures.

The article describes a way of building two devices: with 1 qubit and with 2 qubits. The devices were built from materials that everyone can easily find in a shop.

A library called WavePlate was created for this project. It is made publicly available so everyone can use it.

Keywords

Quantum, Quantum Computer, Neural Network, Photons, Optics, Quantum Machine Learning, WavePlate

1 Introduction

To prove all our concepts we used quantum physics laws and formulas. You can read about this in detail in chapter 2 called "Physics Behind" and more from a mathematical point of view all our concepts are covered in chapter 3 - "Math and Simulations". The general idea behind the quantum device is put photons into a superposition, changing their wave lengths, thereby changing the probability of the photons hitting the right and left photo-resistors. After getting some values in photo-resistors we compute expectation values and send them to neural network as output of the quantum linear layer.

Of course, the quality of the materials we use is not the highest, but this did not prevent us from conducting quality experiments and proving the effectiveness of the device. In chapter 4 called "Hardware" you will be able to read about all the materials for building our device, and in addition, you can find a scheme of the quantum device and information about connecting it to a classical computer. We have ideas on how to improve quality and you can read about it in chapter 9, which is called "The Future of the Projects and Further Devices".

Chapter 5, entitled "Neural Network", deals with the classical levels of the neural network, as well as in detail with the quantum linear level. In addition, there are examples of pictures from the dataset we used. For this project, we wrote a simulator of our quantum device so in this chapter, you will be able to familiarize yourself with how to use the simulator and a real device with a code.

In addition to the 1-qubit device, we also collected a 2-qubit device. All aspects such as: scheme, code for neural network, training and results are discussed in chapter 6 titled "2-Qubit Quantum Device". And finally, all our conclusions are described in detail in the 7th chapter.

2 The Physics behind

This whole project relies on photons, so it is essential to study polarization and optical properties.

2.1 Polarization

Photons have a unique property known as polarization. Light can be polarized or randomly polarized (unpolarized light), which means it vibrates in every direction. However, we can obtain polarized light by using a polarizing lens, such as sunglasses. Polarized light can be vertically or horizontally polarized, or a superposition of both. To measure the polarization of light, we can use a polarizing beam-splitter. This device reflects vertically polarized light and transmits horizontally polarized light. However, this device might be a little expensive and since our goal is an affordable quantum computer we will use the Brewster angle

2.1.1 The Brewster angle

In 1815, Sir David Brewster discovered an incident angle of light on a refractive material at which the reflected light becomes fully vertically polarized while the transmitted light becomes horizontally polarized. This angle can be calculated using the formula

$$\theta = \tan^{-1}\left(\frac{n2}{n1}\right) \tag{1}$$

where n1 and n2 are the refractive indices of the two media. For glass, the Brewster angle is approximately 56.3°. This is exactly what we will use to measure the polarization of light.

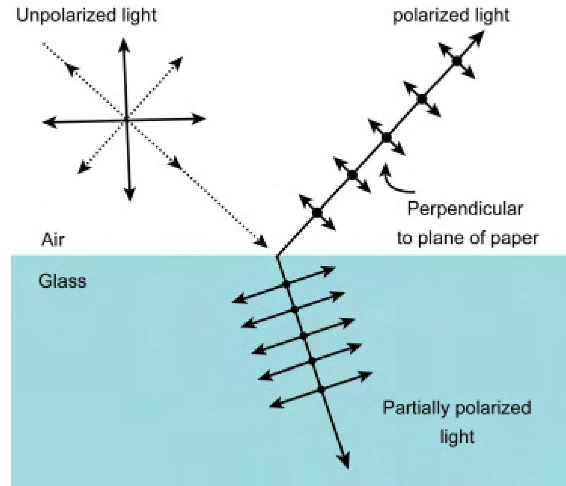


Figure 1: illustration of the brewster angle

2.2 Reflection and Refraction

Photons does not only have wave properties but particles properties as well (wave-particle duality) so we will take closer look to some particle properties such as reflection and refraction.

2.2.1 Reflection

Reflection is the throwing back by a surface without absorbing it. The best way to get light reflections is mirrors.
Given a mirror place at an angle of θ_m and light at an angle of incident θ_i, the angle of reflection can be calculated using the formula

$$\theta_r = \theta_m - \theta_i \tag{2}$$

Where θ_r is the angle of reflection
For example given a mirror placed at an angle of 56.3° and a light with an angle of incidence of 0 the angle of reflection will be 56.3°

2.2.2 Refraction

Refraction is a property that refractive materials have like glass and water where particles become deflected with a different angle through the interface between one medium and another
the angle of refraction can be calculated using Snell's law

$$n1 \cdot \sin(\theta_1) = n2 \cdot \sin(\theta_2) \tag{3}$$

Where n1 and n2 are the refractive indices for the two media and θ_1 is the angle of incident and θ_2 is the angle of refraction

2.3 The Science behind CDs as Half Wave Plates

When it comes to the interaction of light with materials, certain objects, like compact discs (CDs) have interesting optical properties. One of these properties is their ability to function as half-wave plates, which can manipulate the polarization of light. To understand this phenomenon, we can delve into the principles of wave optics.

2.3.1 Half Wave Plates

Polarization refers to the orientation of the oscillation of light waves as we mentioned above. When light interacts with a surface or material, its polarization state can change. Half-wave plates are optical devices that can modify the polarization of incident light.

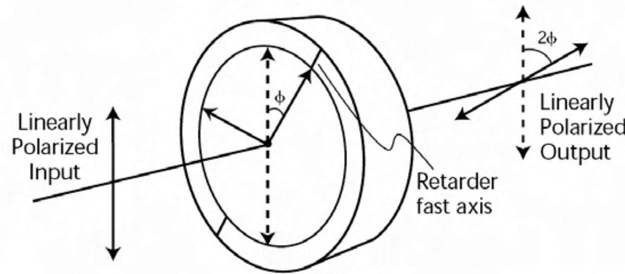


Figure 2: how a half wave plate works

When light enter a half-wave plate, it splits into two perpindicular components: one travels along the fast axis and another along the slow axis when this two components combine again,the phase difference between them will be half a wavelength and the polarization direction will rotate by angle θ_new and i can be calculated by

$$\theta_n = \theta + 2\phi \tag{4}$$

Where θ is the current polarization direction and ϕ is the angle the half-wave plate is rotated with.
But Half wave plates are so expensive which lead us to our next point

2.3.2 Birefringence

Birefringence occurs because the microscopic grooves on the CD surface create two distinct paths for the polarized components of light. One component is delayed relative to the other, resulting in a something called **phase shift**.
As a result of birefringence, a CD can effectively function as a half-wave plate. This capability allows it to convert incident linearly polarized light into light with a polarization direction corresponding to that of the CD. It's noteworthy that we can freely rotate the CD over a 360° angle.

2.3.3 Applications

The ability of CDs to act as half-wave plates has practical applications, especially in optical experiments and devices where polarization control is essential. Researchers and hobbyists can use CDs as cost-effective tools for manipulating polarized light and that's exactly what we'll be using.

2.4 Two photon interference

In order to implement 2 qubit gates we had to achieve a phenomenon called 2-photon interference.This phenomenon is an important step to enable quantum operations.We can achieve this by using the Hong-Ou-Mandel effect

2.4.1 The Hong-Ou Mandel effect

This effect is a quantum interference phenomenon that arise when 2 photons are incident upon a beam-spliter. This effect result in the photons becoming entangled in such a way that they exit the beam-splitter entangled.

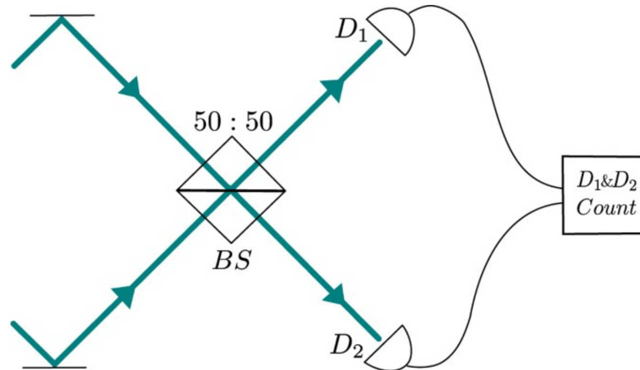


Figure 3: The Hong-Ou Mandel effect, where BS stands for beam-splitter and D1 and D2 stands for detector 1 and 2

Now we can apply quantum optical gates like a Half Wave Plate for the photons after they exit the beam-splitter by this way we can get multi-qubit gates. These gates play an important role in the manipulation and processing of quantum information so we can fully harness the potential of quantum computing

3 Maths and Simulations

Prior to constructing the physical hardware, one essential step was to keep track of the mathematics involved and do some simulations to validate the functionality of the hardware. As a result, we embarked on a comprehensive study of the mathematics and developed our Python library for simulating optical hardware.

3.1 Light as Vectors

We started by representing the light beam as 1x8 vector where each element represents both the direction and polarization of the light. The vector is structured as follows, with 'H' indicating horizontal polarization and 'V' indicating vertical polarization:

$[RightH, RightV, LeftH, LeftV, UpH, UpV, DownH, DownV]$

3.2 Optical elements as matrices

For optical elements such as Beamsplitters or half-wave plates, they are represented by an **8x8** matrix, which we will discussing further in the code. Operations are performed by multiplying the current state with the optical matrix.

3.2.1 The Research

After conducting some research in an effort to make the optical element matrices as precise as possible, we arrived at a solution utilizing trigonometric functions for amplitude and Euler's number for phase. Resulting in expressions like this: $\cos\theta * e^{*\pi}$

3.2.2 Optical elements examples

For example this is the matrix representing the half wave plate in Python since the half wave plate have a phase shift of $e^{*\pi}$ which is -1 so required values will be multiplied by -1

Listing 1: Code for simulating half-wave plate

```
def HWP(self, theta):
    theta = math.radians(theta)
    first = math.cos(theta)**2 - math.sin(theta)**2
    second = 2 * math.cos(theta) * math.sin(theta)
    third = math.sin(theta)**2 - math.cos(theta)**2
    fourth = 2 * (-math.sin(theta)) * math.cos(theta)
    self.hwpmatrix = np.array([
        [first,second,0,0,0,0,0,0],
        [second,third,0,0,0,0,0,0],
        [0,0,third,fourth,0,0,0,0],
        [0,0,fourth,third,0,0,0,0],
        [0,0,0,0,first,fourth,0,0],
        [0,0,0,0,fourth,third,0,0],
        [0,0,0,0,0,0,third,second],
        [0,0,0,0,0,0,second,first]
        ])
    result = self.multiply(self.state , self.hwpmatrix)
    return result
```

3.3 Simulations

To perform simulations, we require classical computers. We chose Python as our preferred programming language and created a class called **PhotonicCircuit()**. This class starts with an initial state of |Right H> (which can be changed). As of now, this class can simulate four optical elements: **mirrors, beam splitters, polarizing beam splitters, and half-wave plates**. We are actively exploring opportunities to expand its capabilities further. After performing simulations, you can obtain counts from the circuit. Additionally, after some careful consideration, we found out a method to measure the Z expectation value.

3.3.1 Testing

We conducted tests on simple algorithms and protocols such as the **Deutsch-Jozsa** or the **BB84 protocol using 1 and 2-bit strings**.
The simulations were successful.
You can access tutorials and documentation via the following Link

3.4 Quantum layer simulation

Back to our main mission (a quantum layer for our neural network) we used half wave plates to get trainable parameters and polarizing beam splitter to get measurements notably, we opted not to use beam splitters or mirrors, as this is a 1-qubit quantum layer, which we plan to expand upon at the end of this paper.

We used a simple gradient formula to calculate our gradients:

$$\nabla F(\theta) = \frac{F(\theta + \frac{\pi}{2}) - F(\theta - \frac{\pi}{2})}{2} \tag{5}$$

Analytic gradients Formula
and to calculate the loss we used 2 loss function for the 1 and 2 qubit devices respectively

$$L = -\sum_{i=1}^{M} y_i \log \hat{y}_i \tag{6}$$

The NLL loss where $\hat{y}$ is the probability distribution and y is the target

$$L = \sum_{i=1}^{n} |y - \hat{y}| \tag{7}$$

The L1 loss also known as Absolute Error
and the quantum parameters are updated by:

$$\theta_{\text{new}} = \theta_{\text{current}} - (\alpha \cdot L \cdot \nabla F(\theta)) \quad (8)$$

where α is the learning rate , L is the loss ,$\nabla F(\theta)$ is the gradients

3.4.1 Final simulation result

Adding the quantum layer to our classical neural network which we will mentioned below
The training process proceeded successfully. Full tutorial is here: Link [1]

4 Hardware

We started from creating creating scheme of the 1-qubit device. The device have to contain laser, polarizer, half-wave plates(HWP), mirror, beam-splitter(BS) and photoresistors.

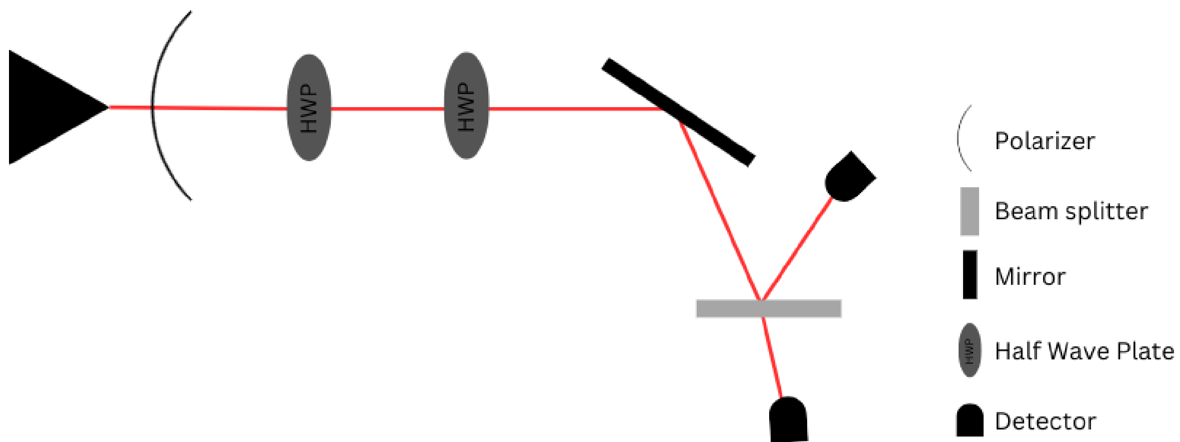


Figure 4: Scheme of 1-qubit device.Where the mirror is placed at an angle of 56.3° and the beam splitter at an angle of 0°

[1] You can download the library now by **pip install waveplate**

4.1 Parts of the Quantum Computer

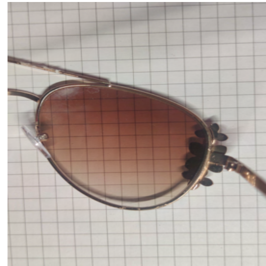
We have to specify some details about all parts of the device:

Laser. We are using a red laser (length of wave is 650nm).

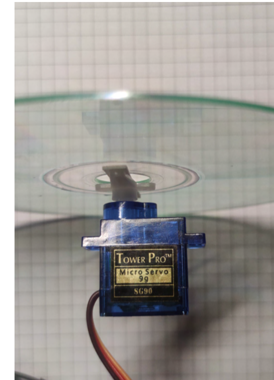
Polarizer. As a polarizer we used sunglasses.

Half-Wave Plate. To create all possible rotation of rotation gates we decided to use transparent refractive CD as HWP and micro servos to change angle of the CD and as a result change the wave length of photons to change probabilities.

(a) Laser 650nm.



(b) Polarizer.



(c) CD and Micro Servo.

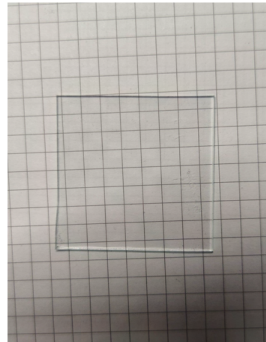
Figure 5: Laser, polarizer and HWP.

Mirror. It is important to use mirrors without glass layer (because it will make unwanted interference) so we used dental mirrors.

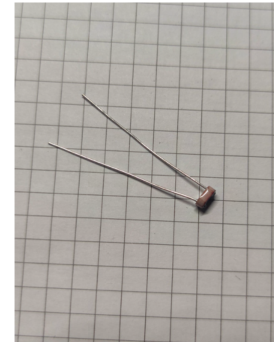
Beam Splitter. The cheapest thing that gives the closest result to non-polarizing beam split cubes is a piece of glass.

Photoresistor. The photoresistors that were used with a resistor of 10 kOhm.

(a) Dental mirror.



(b) Piece of glass.



(c) Photoresistor.

Figure 6: Mirror, beam-splitter and photoresistor.

4.2 Arudino

To control quantum computer we have build two types of connections: from classical computer to quantum (C-to-Q) and from quantum computer to classical(Q-to-C). For both connections we used Arduino.

C-to-Q

Our goal is to get data from neural network so we can interpret input numbers as angles for rotation of HWP. To make such rotation we used Micro Servo.

Q-to-C

To get output from quantum computers we used photoresistors. We are measuring values of photoresistors and then counting expectation values after making measurement several times.

Required Arduino Equipment

The required Arduino equipment is: wires(mainly male-male and male-female), 2 Micro Servos, 2 photoresistors, 2 resistors (10 kOhm) and breadboard.

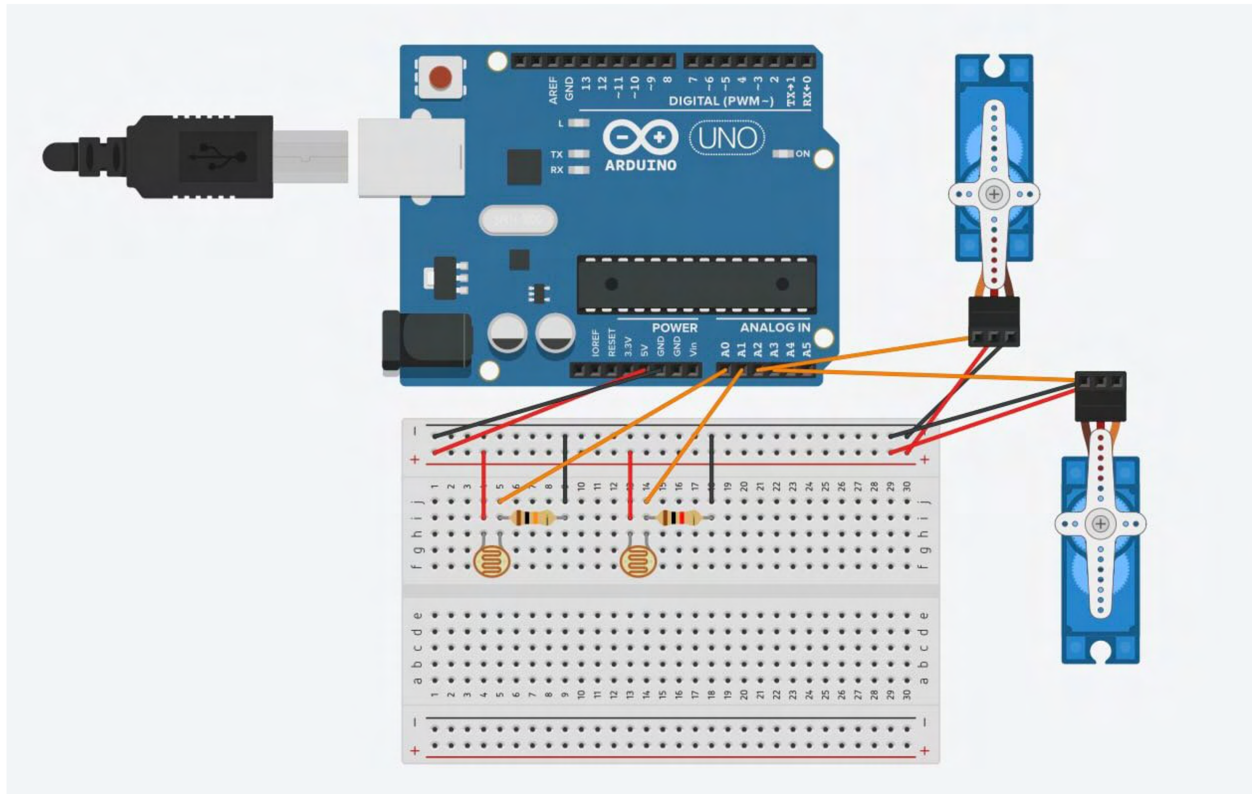


Figure 7: Scheme of all connections to Arduino chip.

4.2.1 Code for Arduino

The code for Arduino takes input angles for Micro Servos from Python NN. Then measure values from photoresistors and send output back to NN.

Listing 2: Code for Arduino with 2 microservos and 2 photoresistors.

```
#include <Servo.h>

Servo servo1;
Servo servo2;
int incoming[2];

void setup(){
    Serial.begin(9600);
    servo1.attach(9);
    servo2.attach(10);
    pinMode(A0,INPUT);
    pinMode(A1,INPUT);
}
```

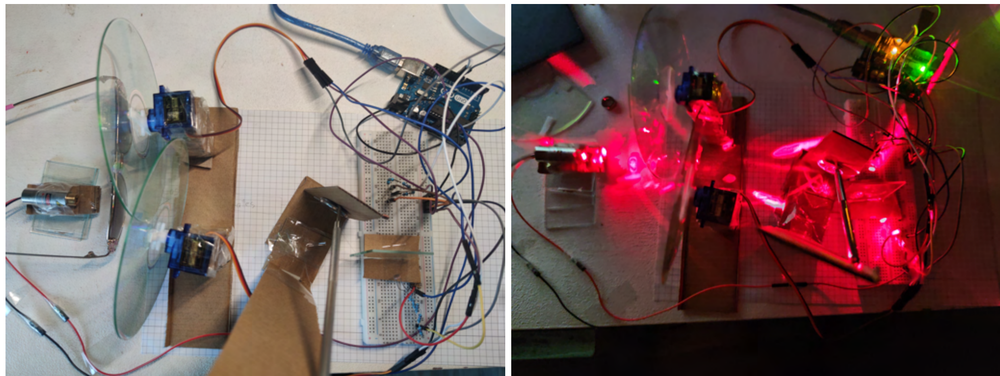
```
void loop() {
  // put your main code here, to run repeatedly:
    while(Serial.available() >= 2){

        for (int i = 0; i < 2; i++){
          incoming[i] = Serial.read();
        }

        if(incoming[1] == 0){
          servo1.write(incoming[0]);
        }
        if(incoming[1] == 1){
          servo2.write(incoming[0]);
        }
  }
        delay(1000);
        int result1 = analogRead(A0);
        int result2 = analogRead(A1);
        Serial.println(result1);
        Serial.println(result2);
}
```

4.3 Building the device

To build the quantum device we use scheme, all required parts, connect everything to Arduino and test it using the NN.



(a) Light version (b) Dark version

Figure 8: Upper view of the quantum device.

5 Neural Network

We created the neural network (NN) in Python programming language that contains classical layers and a quantum layer. The dataset that we used contains pictures of ants and bees. The goal of the NN is analyze picture and say is it an ant or a bee in a given picture.

5.1 Classical part

To create the NN we used PyTorch library. The classical part contains 6 layers: 2 convolutional and 4 linear. Also we added a dropout function to prevent the risk of the model overfitting. In the figure you can see classical layers in `__init__` (or main) function of the NN.

Listing 3: Part of the code with classical layers in the main function.

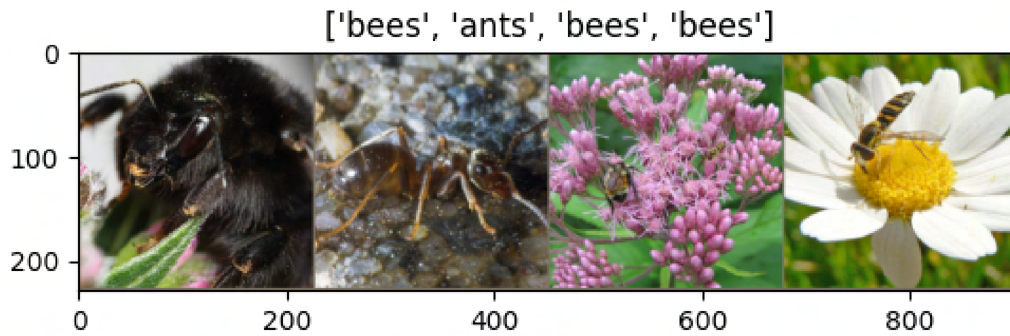


Figure 9: Examples of data.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3)
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout1 = nn.Dropout2d(0.25)
        self.conv2 = nn.Conv2d(16, 32, 3)
        self.fc1 = nn.Linear(32 * 54 * 54, 1200)
        self.fc2 = nn.Linear(1200, 120)
        self.fc3 = nn.Linear(120, 84)
        self.fc4 = nn.Linear(84, 1)
```

In forward function of NN we used ReLu as an activation function for our classical layers.

Listing 4: Part of the code with classical layers in the forward function.

```
    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.dropout1(x)
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 32 * 54 * 54)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
```

5.2 Quantum part

5.2.1 Simulation

We created a custom layer using our own library we mentioned above, with the simulation of the actual quantum device. We implemented three functions: a function for the quantum layer, a function for getting gradients, and a function for running the circuit for gradients. Here are the code respectively

Listing 5: The Quantum Layer

```
def quantum_layer(q_input_features, q_weights_flat):
    q_weights = q_weights_flat.reshape(q_depth, n_qubits)

    qc = PhtotonicCircuit()
    qc.HWP(22.5)
    for idx, angle in enumerate(q_input_features):
        qc.HWP(angle, idx)
```

```
    for layer in range(q_depth):
        for idx, angle in enumerate(q_weights[layer]):
            qc.HWP(angle, idx)

    return qc.Z_expectation(shots=100)[2]
```

Listing 6: Function for running the gradients

```
def run(q_weights):
    qc = PhtotonicCircuit()
    qc.HWP(22.5)

    for idx, angle in enumerate(q_weights):
        qc.HWP(angle, idx)

    return qc.Z_expectation(shots=100)[2]
```

Listing 7: Gradients Function

```
def apply_gradient(params):
    params = params.tolist()
    s = np.pi/2
    gradient = []
    for k in params:
        k_plus = k + s
        k_minus = k - s
        exp_plus = run([k_plus])
        exp_minus = run([k_minus])
        gr = (exp_plus - exp_minus) / 2
        gradient.append(gr)
    return torch.tensor(gradient,dtype=torch.float32)
```

5.2.2 Actual quantum device

We created a custom layer using the Serial library for sending input values to Arduino and receiving the output.
We implemented the same three functions as the simulation but with different methods to get and send data to the real quantum device

Listing 8: Functions for sending and receiving data from the Arduino

```
ser = serial.Serial('COM8', 9600)

def send_to_arduino(ser, values):
    ser.write(struct.pack('>BB',*values))

def receive():
    if ser.in_waiting > 0:
        data = ser.readline().decode('utf-8').rstrip()
        return data
```

Please note that when connecting Serial, you must ensure that the Arduino is connected to the correct port. In our case, it was **COM8**.

Listing 9: The Quantum layer with Arduino

```
def quantum_layer(q_input_features, q_weights_flat):
    value1, value2 = None, None
    q_weights = q_weights_flat.reshape(q_depth, n_qubits)

    for idx, angle in enumerate(q_input_features):
        ang = angle.item()
        while ang < 0:
```

```
            ang += math.pi/2
        an = int(((ang*2)*180/math.pi))
        send_to_arduino(ser,[an,idx])
    time.sleep(2)
    for layer in range(q_depth):
        for idx, angle in enumerate(q_weights[layer]):
                ang = angle.item()
                while ang < 0:
                  ang += math.pi/2
                an = int(((ang*2)*180/math.pi))
                send_to_arduino(ser,[an,idx])

    time.sleep(4)
    while value1 == None: value1 = receive()
    while value2 == None: value2 = receive()
    return get_exp(value1,value2)
```

Listing 10: Function for running the gradients

```
def run(input,q_weights):
    value1, value2 = None, None
    for idx, angle in enumerate(input[0]):
        ang = angle
        while ang < 0:
          ang += math.pi/2
        an = int(((ang*2)*180/math.pi))
        send_to_arduino(ser,[an,idx])
    time.sleep(2)
    for idx, angle in enumerate(q_weights):
        ang = angle
        while ang < 0:
          ang += math.pi/2
        an = int(((ang*2)*180/math.pi))
        send_to_arduino(ser,[an,idx+1])
    time.sleep(4)
    while value1 == None: value1 = receive()
    while value2 == None: value2 = receive()
    return get_exp(value1,value2)
```

Listing 11: Gradients Function with Arduino

```
def apply_gradient(input,params):
    input,params = input.tolist(), params.tolist()
    s = np.pi/2
    gradient = []
    for k in params:
        k_plus = k + s
        k_minus = k - s
        exp_plus = run(input,[k_plus])
        exp_minus = run(input,[k_minus])
        gr = (exp_plus - exp_minus) / 2
        gradient.append(gr)
    return torch.tensor(gradient,dtype=torch.float32)
```

5.3 Training

Now comes the training part with used the SGD optimizer and the cross entropy loss for our training function of course after inserting the quantum layer in the neural network forward function

5.3.1 Using the quantum layer

```
self.q_params = torch.nn.Parameter( q_delta torch.randn(q_depth,dtype=torch.float32))
```

quantum parameters in the main function in the NN where $qdelta$ and $qdepth$ are constants

Listing 12: The quantum layer in the forward function

```
q_out = None
for elem in x:
    # print(elem)
    q_out_elem = quantum_layer(elem, self.q_params)
    if q_out == None:
        q_out = torch.tensor([[q_out_elem]])
    else:
        q_out = torch.add(q_out, torch.tensor([[q_out_elem]]))

q_out = (q_out+1)/2
q_out = torch.cat((q_out, 1-q_out), -1)
q_out = q_out.requires_grad_()
return q_out
```

5.3.2 Training the NN

Listing 13: The Training loop

```
network = Net()
optimizer = optim.SGD(network.parameters(), lr=0.001, momentum=0.9)
epochs = 10
criterion = nn.CrossEntropyLoss()
loss_list = []

for epoch in range(epochs):
    total_loss = []
    target_list = []
    for data, target in dataloaders['train']:
        data = data.to(device)
        target = target.to(device)
        target_list.append(target.item())
        optimizer.zero_grad()
        output = network(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        # update quantum parameters
        gradient = apply_gradient(output,network.q_params)
        outlist = output.tolist()[0]
        out = (1-outlist[0]) if outlist[0] > outlist[1] else outlist[1]
        new_params = nn.Parameter(network.q_params - (0.001 * (out-target.item()) * gradie
        network.q_params = new_params
        total_loss.append(loss.item())
    loss_list.append(sum(total_loss) / len(total_loss))
    print('Loss␣=␣{:.2f}␣after␣epoch␣#{:2d}'.format(loss_list[-1], epoch + 1))
```

After completing the training, you will observe continuous improvement in the loss function. This concludes the discussion of the neural network.
Link for the full simulation tutorial here and Hardware tutorial here

6 Two-Qubits Quantum device

We also implemented a 2-qubits quantum device so we can use entangling gates and explore other projects as well as enhance the quality of the quantum layer of our NN

6.1 The theme of the device

We also started by drawing a scheme of what the device should look like

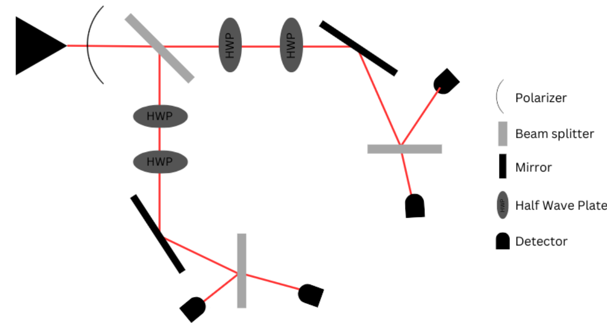


Figure 10: The scheme of the 2 qubit device, where first beam splitters are placed at an angle of 45°, 90°, 0° respectively and the mirrors are placed at an angle of 56.3°

6.2 The simulation

We implemented a code using our library to simulate the scheme we designed. In this code, we developed a quantum layer that accepts input features and weights and returns the expectation value.
Link for the 2 qubit simulation tutorials Here.

Listing 14: 2 qubit device quantum layer simulation

```
import numpy as np
VS=['|RH>','|RV>','|UH>','|UV>','|LH>','|LV>','|DH>','|DV>']

n_qubits = 2
n_layers = 1

inputs = [1,2]
weights = np.random.random([n_layers, n_qubits])

def 2qubitlayer(inputs,weights):
      qc = PhtotonicCircuit()
      qc.BS(135)
      # H layers
      for i in range(n_qubits):
         qc.HWP(22.5,i)

      # INPUT features
      for idx,i in enumerate(inputs):
         qc.HWP(math.degrees(i),idx)

      # WEIGHTS
      for _ in range(n_layers):
         for idx,i in weights:
            qc.HWP(math.degrees(i),idx)
         # -- CNOT --
         for i in range(n_qubits):
            if i % 2 == 0:
```

```
            probs = qc.measure_probs()
            try:
                if VS[ (2**i) - 1 ] in probs and VS[2**i] in probs:
                    qc.HWP(22.5, i+1)
                elif probs[VS[2**i]] >= 50:
                    qc.HWP(45, i+1)
            except: pass
    return qc.Z_expectation(shot=100)

2qubitlayer(inputs, weights)
```

You can find the full tutorial Here

6.3 The Actual Hardware

6.3.1 CNot implementation

We used conditional operations based on the polarization properties of photons to implement the CNot gate. Details about how this is achieved can be found in the code above. In essence, we measure the control qubit and, based on that measurement, adjust the orientation of the half-wave plate on the target qubit before measuring the target qubit.

6.3.2 Connections

For the 2-qubit device, we used 4 micro servo motors, 4 photo-resistors and 4 10Kohm resistors for measurements, a PCA9685 chip to power and optimize everything, a 9V battery, and of course, an Arduino Uno.

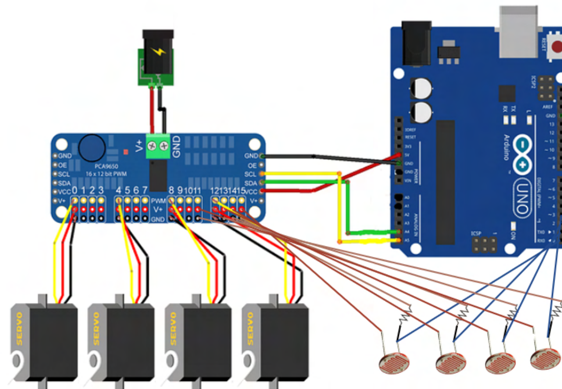


Figure 11: The Arduino connections

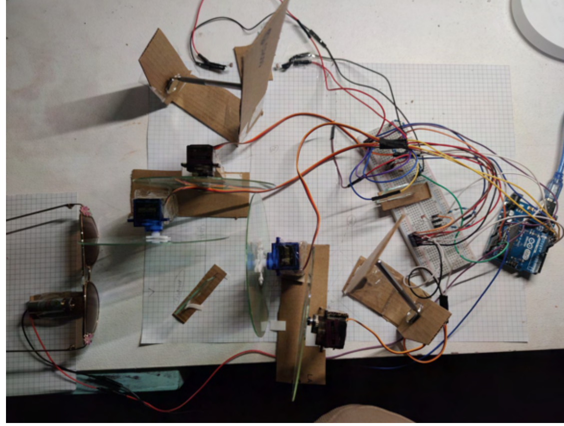


Figure 12: The 2 qubit device

6.4 Neural network with 2-qubit device

We implemented the same neural network but with a new quantum layer and expectation value function that can process the data from and to the 2-qubit device

It's noteworthy that we replace the NLL loss with the L1 loss in the 2 qubit device

6.4.1 Code

Here is the code of quantum layer with 2 qubits:

Listing 15: Python code for Quantum layer

```
def quantum_layer(q_input_features, q_weights_flat):
    value1, value2, value3,value4 = None, None,None,None
    q_weights = q_weights_flat.reshape(q_depth, n_qubits)
    for idx, angle in enumerate(q_input_features):
        ang = angle.item()
        while ang < 0:
          ang += math.pi/2
        an = int(math.degrees(ang))
        #print(an)
        send_to_arduino(ser,[an,idx])

    for layer in range(q_depth):
        for idx, angle in enumerate(q_weights[layer]):
          ang = angle.item()
          while ang < 0:
            ang += math.pi/2
          an = int(math.degrees(ang))
          #print(an)
          send_to_arduino(ser,[an,idx+2])

    #time.sleep(1)
    #print('Started layer')
    while value1 == None:
        try: value1 = int(receive())
        except: pass
    while value2 == None:
        try: value2 = int(receive())
        except: pass
    while value3 == None:
```

```
        try: value3 = int(receive())
        except: pass
    while value4 == None:
        try: value4 = int(receive())
        except: pass {value3} and {value4}')
␣␣␣␣return␣get_exp(value1, value2, value3, value4)
```

And here is the code for Arduino:

Listing 16: Code for Arduino with 4 micro servos and 4 photoresistors.

```
#include <Servo.h>

Servo servo1; // qubit 0
Servo servo2; // qubit 0
Servo servo3; // qubit 1
Servo servo4; // qubit 1
int incoming[2];

void setup() {
    Serial.begin(9600);
    servo1.attach(10);
    servo2.attach(11);
    servo3.attach(12);
    servo4.attach(13);
    pinMode(A0,INPUT); // qubit 0 state 0
    pinMode(A1,INPUT); // qubit 0 state 1
    pinMode(A2,INPUT); // qubit 1 state 0
    pinMode(A3,INPUT); // qubit 1 state 1
}

void loop() {
  while(Serial.available() >= 2){
    for(int i = 0; i < 2, i++){
      incoming[i] = Serial.read()
    }
    if(incoming[1] == 0){
      servo1.write(incoming[0]); // qubit 0 first gate
    }
    else if(incoming[1] == 1){
      servo3.write(incoming[0]); // qubit 1 first gate
    }
    else if(incoming[1] == 2){
      servo2.write(incoming[0]); // qubit 0 second gate
    }
    else if(incoming[1] == 2){
      servo4.write(incoming[0]); // qubit 1 second gate
    }
  }
  delay(1000);
  int value1 = analogRead(A0); // qubit 0 state 0
  int value2 = analogRead(A1); // qubit 0 state 1
  int value3 = analogRead(A2); // qubit 1 state 0
  int value4 = analogRead(A3); // qubit 1 state 1
  Serial.println(value1);
  Serial.println(value2);
  Serial.println(value3);
  Serial.println(value4);
}
```

You can find the full tutorial code for the 2 qubit device Here

6.4.2 Results

We trained the neural network and after 40 epochs we got such results:

```
Loss = 0.7689 after epoch # 6
Loss = 0.7390 after epoch # 7
Loss = 0.7163 after epoch # 8
Loss = 0.6986 after epoch # 9
Loss = 0.6811 after epoch #10
Loss = 0.6543 after epoch #11
Loss = 0.6456 after epoch #12
Loss = 0.6574 after epoch #13
Loss = 0.6333 after epoch #14
Loss = 0.6103 after epoch #15
Loss = 0.5573 after epoch #16
Loss = 0.5396 after epoch #17
Loss = 0.5221 after epoch #18
Loss = 0.5343 after epoch #19
Loss = 0.5008 after epoch #20
Loss = 0.4866 after epoch #21
Loss = 0.4533 after epoch #22
Loss = 0.4160 after epoch #23
Loss = 0.3992 after epoch #24
Loss = 0.3800 after epoch #25
Loss = 0.3911 after epoch #26
Loss = 0.3753 after epoch #27
Loss = 0.3452 after epoch #28
Loss = 0.3134 after epoch #29
Loss = 0.2923 after epoch #30
Loss = 0.2746 after epoch #31
Loss = 0.2986 after epoch #32
Loss = 0.2777 after epoch #33
Loss = 0.2571 after epoch #34
Loss = 0.2039 after epoch #35
Loss = 0.1963 after epoch #36
Loss = 0.1510 after epoch #37
Loss = 0.1156 after epoch #38
Loss = 0.0990 after epoch #39
Loss = 0.0670 after epoch #40
Accuracy: 148 / 153 = 96.73202614379085%
```

Figure 13: Loss for every epoch and accuracy of the neural network for the 2 qubit device.

As you can see the Neural Network started with a loss of 0.78 and ended the training with a loss of 0.067 after 40 epochs which is 30% better compared to the 1 qubit device, and using the testing dataset the NN got 148 samples correct out of 153 which gives an accuracy 96.7%

7 Conclusions

By running the neural network with the 1 qubit and 2 qubit devices we have made some conclusions:

1. The mean value of the loss for the 1 qubit device is -0.42, the highest value is -0.70 and the lowest value is -0.21
2. The mean value of the loss for the 2 qubit device is 0.39, the highest value is 0.78 and the lowest value is 0.067
3. There is a clear pattern that the loss exponentially go down while maintaining the same number of epochs by increasing the number of qubits
4. by training the same NN without the quantum layer with the same number of epochs we noticed a significant difference in the final performance
5. in order to get the best results from the photo-resistors the quantum computer had to operate at a total dark room
6. the simulation and math has shown that the device is fully capable of other quantum projects as well
7. There is an optical implementation for almost every quantum gate, which will enable the quantum computer to achieve its full potential.
8. by increasing the quality of the components and getting new ones we can achieve much better results
9. For this moment the speed of the quantum layer is not high because of C-to-Q connection.

The total cost of the 1 quit device is around 50 dollars and the 2 qubit one is around 65 dollars what is very cheap compared to other quantum computers.

8 The Future of the project and further devices

The project has many prospects.

1. One of them is find a faster way of C-to-Q and Q-to-C connections.
2. We can find a pattern in the increase of the number of qubits and their entanglement.
3. We can find ways to overcome the limitations we currently have like in gates implementation for example
4. In the near future we plan to make a device with 3-4 qubits and in the future 7 and 8 qubits.
5. We plan to try out different entanglement methods like the nonlinear sign-shift gate
6. Also we have ideas and plans to achieve and try more projects with the quantum devices by getting new components like a precise 50:50 and a polarizing beam-splitter,etc and improving the quality of the exiting ones
7. In order to get a better 50:50 beam splitter we can use a window tint on the glass that has a 50% Visible Light Transmission (VLT) so we can get more better results

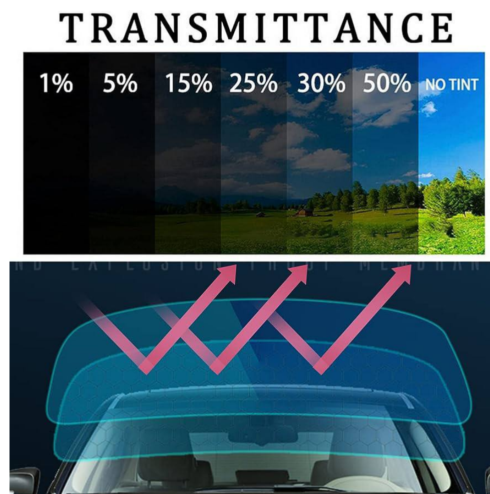


Figure 14: Different VLT window firms illustration

8. Moreover, the price of such a device will be hundreds of times cheaper than modern quantum devices, which will be perfect for learning and processing purposes.
9. Also, by receiving financial support and collaborating with universities, we can improve the quality of the the quantum computer accordingly, increase the accuracy of calculations
10. It is likely that in the future it will be possible to create a device with 15 qubits, the computing power of which, in theory, will be equal to the computing power of a classical computer.

9 References

For this paper we used information from 9 references. You can find all of them below [1, 2, 3, 5, 4, 6, 7, 8, 9].

References

[1] Agata M. Branczyk. arxiv paper on the hong-ou mandel interference. https://arxiv.org/pdf/1711.00080.pdf.

[2] CircuitBasics. Circuit basiscs tutorial on how to connect photo-resistors. https://www.circuitbasics.com/how-to-use-photoresistors-to-detect-light-on-an-arduino/.

[3] PennyLane. Pennylane tutorial for a quantum layer. https://pennylane.ai/qml/demos/tutorial_quantum_transfer_learning.

[4] PyTorch. Pytorch loss functions. https://pytorch.org/docs/stable/nn.html#loss-functions.

[5] PyTorch. Pytorch tutorial on creating custom models. https://pytorch.org/tutorials/beginner/examples_nn/two_layer_net_module.html.

[6] Qiskit. Qiskit gradients. https://qiskit.org/documentation/tutorials/operators/02_gradients_framework.html.

[7] ScienceDirect. Science direct explanation of the brewster angle. https://www.sciencedirect.com/topics/earth-and-planetary-sciences/brewster-angle.

[8] Wikipedia. Wikipedia explantion of birefringence. https://en.m.wikipedia.org/wiki/Birefringence.

[9] Wikipedia. Wikipedia explantion of specular reflection. https://en.wikipedia.org/wiki/Specular_reflection.