Machine Learning : Neural Networks in AI and Quantum computing

Leonidas Bakopoulos
Alexandra Tsipouraki

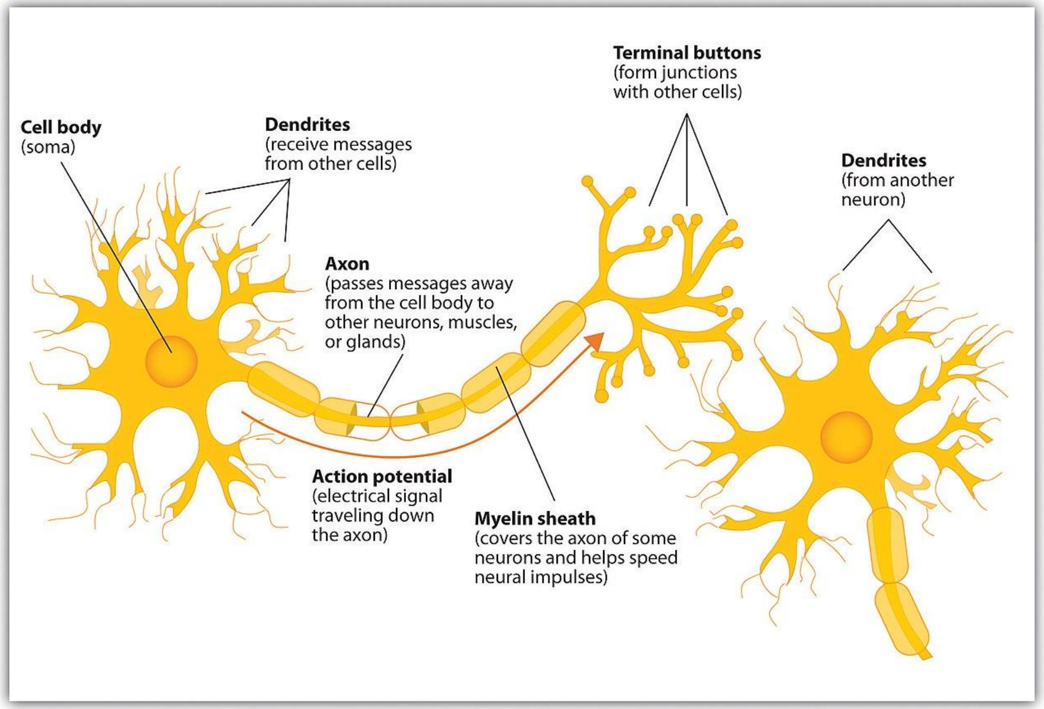
Contents

General Information

- ➔ General information on neural and quantum neural networks beginning with machine learning and following with quantum mechanics
- ➔ Show where it all begun : What is a neuron ?
- ➔ What sparked the need for neural networks?
- ➔ How do we implement a neural network in machine learning and how can it be trained to work properly ?
- ➔ Applications , Convolution & Qiskit code
- ➔ Quantum neural networks : implementation and algorithm
- ➔ Quanvolution and Qiskit code
- ➔ Other applications
- ➔ Conclusions : Comparison between quantum and classical neural networks

Neurons

Cells responsible for receiving sensory input from the external world, sending motor commands to the muscles and for transforming and relaying the electrical signals at every step in between.



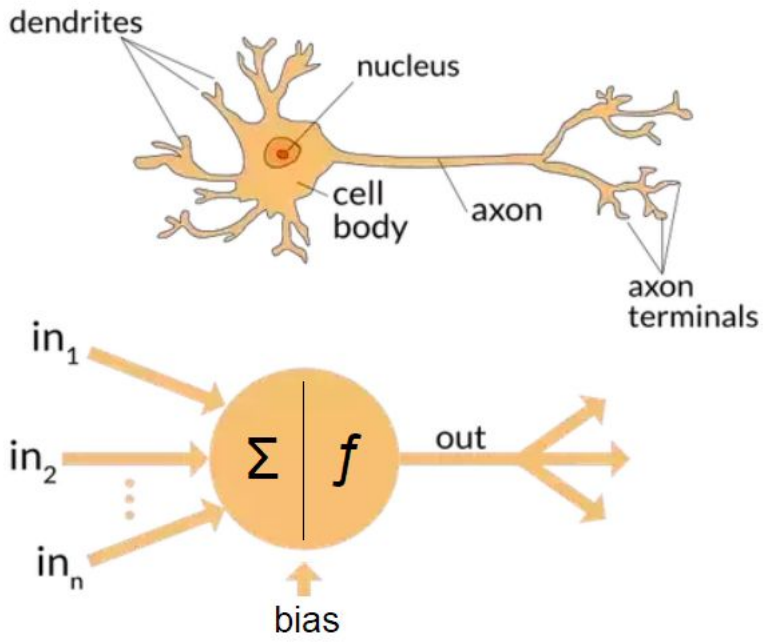
Terminal buttons
(form junctions with other cells)
Cell body
(soma)
Dendrites
(receive messages from other cells)
Dendrites
(from another neuron)
Axon
(passes messages away from the cell body to other neurons, muscles, or glands)
Action potential
(electrical signal traveling down the axon)
Myelin sheath
(covers the axon of some neurons and helps speed neural impulses)

Neural Networks in classical Machine Learning

Instead of neurons activation cells are used

The connection between the cells is attained between two layers, and dendrites are replaced by "weighted wires"

"Weighted wires" are virtual connections between nodes of two successive layers



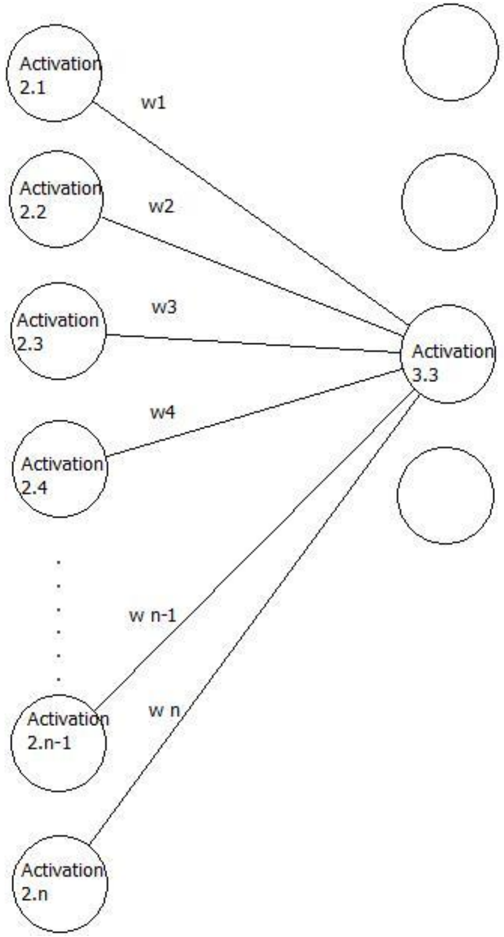
A biological and an artificial neuron (via https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network)

Neural Networks in classical Machine Learning

Neurons themselves are replaced by nodes that hold a specific number per application called "Activation"

in order to enter correct inputs as weights a specific machine learning technique was developed, known as "network training"

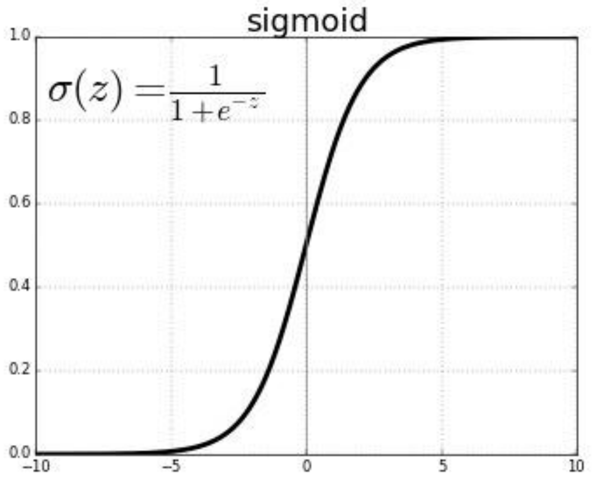
$$f\left(\sum_i Activation_{2,i} * w_i - bias\right)$$



Network Training

randomly set all values to numbers within the range [0 ,1] and afterwards proceeds with testing their correctness by testing them with several inputs and comparing the outputs of the network to the ones that are expected and marked as correct.

in most cases the function that is used for a neural network training is a sigmoid function, due to its continuous flow of “input “ values



Gradient descent

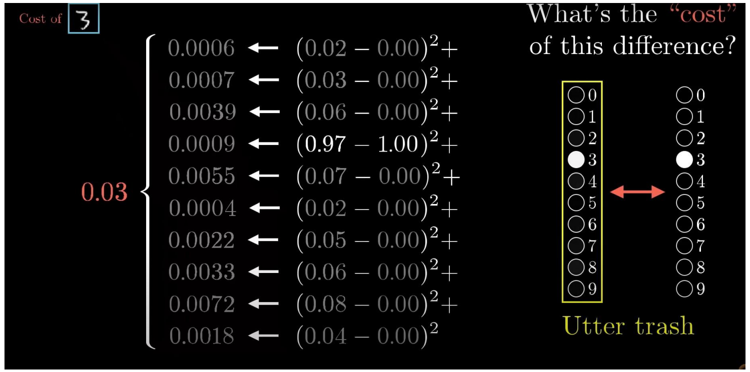
a cost function is generated
(C(x, θ) = (untrainedOutputs(x, θ)-correctOutput(x)) ^2)

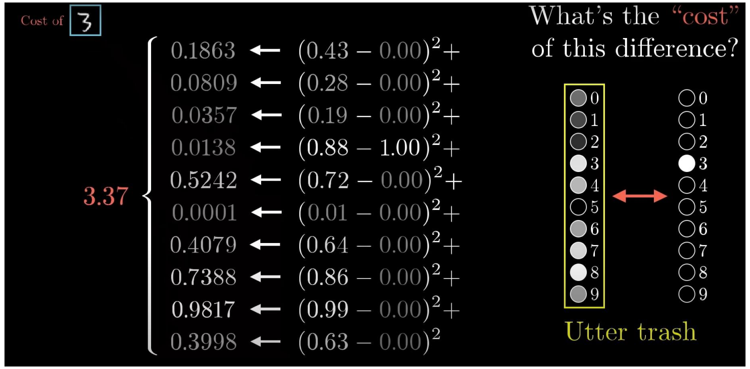
The bigger the C(x, Θ), the worst the neural network performs

To achieve the best performance we assign to θ, the values that achieve the minimum cost : min(C(x,θ).

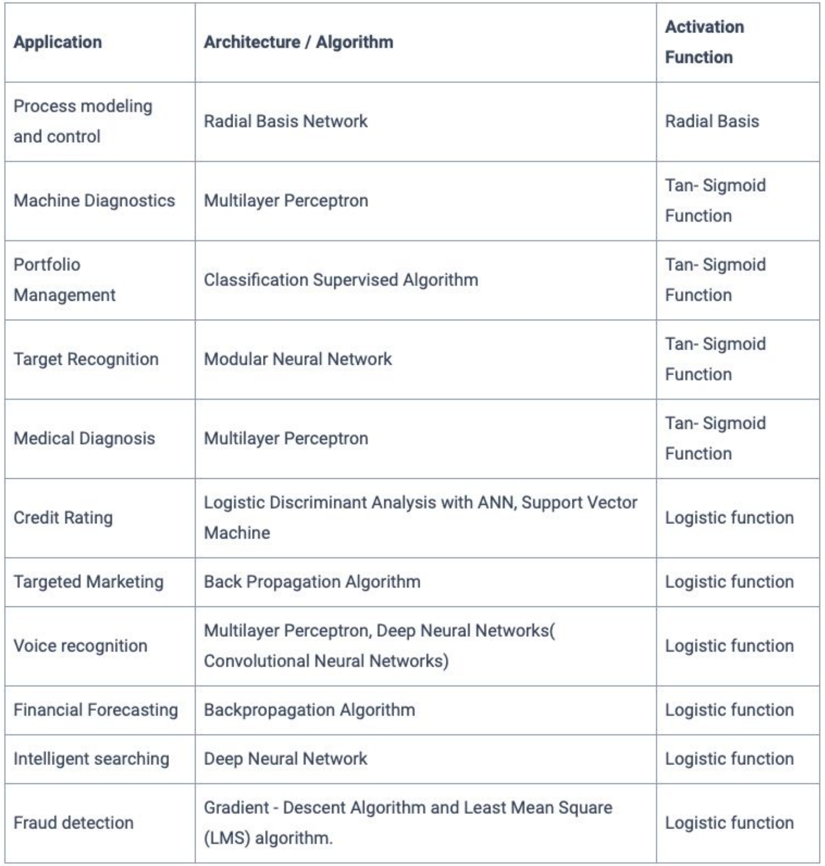
∇C(x, θ)=∇C(θ)=0.

By training the neural network, the value of θ is changed in order to achieve minC() or less mathematically to reduce all possible losses or misses and increase the performance of the network.





Applications of Neural Networks in machine learning and Convolutional Neural Networks



Application	Architecture / Algorithm	Activation Function
Process modeling and control	Radial Basis Network	Radial Basis
Machine Diagnostics	Multilayer Perceptron	Tan- Sigmoid Function
Portfolio Management	Classification Supervised Algorithm	Tan- Sigmoid Function
Target Recognition	Modular Neural Network	Tan- Sigmoid Function
Medical Diagnosis	Multilayer Perceptron	Tan- Sigmoid Function
Credit Rating	Logistic Discriminant Analysis with ANN, Support Vector Machine	Logistic function
Targeted Marketing	Back Propagation Algorithm	Logistic function
Voice recognition	Multilayer Perceptron, Deep Neural Networks(Convolutional Neural Networks)	Logistic function
Financial Forecasting	Backpropagation Algorithm	Logistic function
Intelligent searching	Deep Neural Network	Logistic function
Fraud detection	Gradient - Descent Algorithm and Least Mean Square (LMS) algorithm.	Logistic function

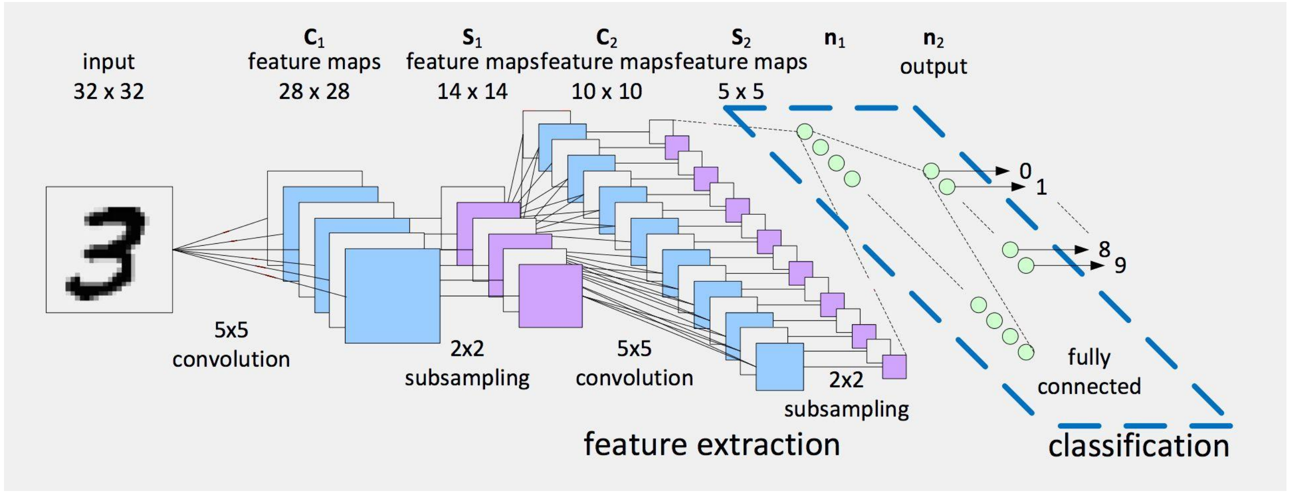
Applications of Neural Networks in machine learning and Convolutional Neural Networks

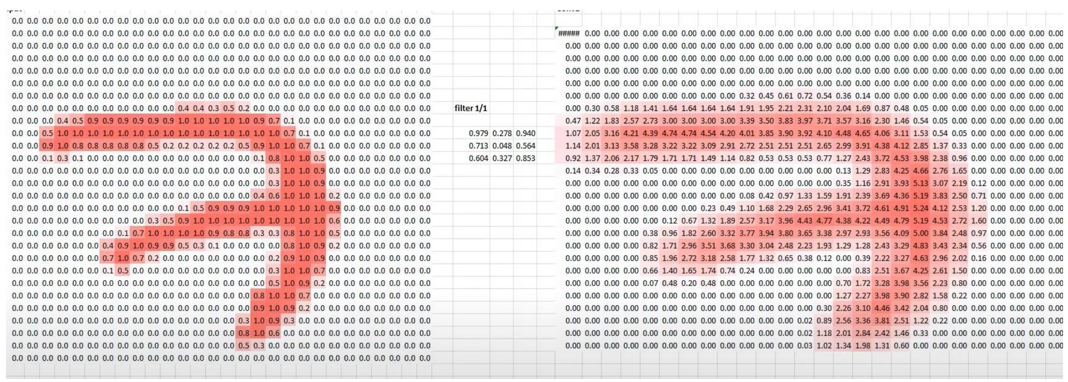
Convolutional networks are a specialized type of neural networks that use matrix convolution in place of weight linear combination in at least one of their layers

They are applied for pattern recognition within images , allowing the insertion of image-specific features into the architecture of the network, making it more suited for image-focused tasks while simultaneously reducing the complexity of setting up the model . The fact that CNNs have simpler models, meaning much fewer connections and parameters , also makes them easier to train without causing severe losses on their performance.

Convolutional Neural Networks : implementation overview

In CNNs a whole image is used as an input instead of an array of pixels
Matrices aka kernels are used instead of nodes
Filters are used instead of weights
Two hidden layers :
convolutional layer, for pattern detections and the
pooling layer for combining all fitting patterns



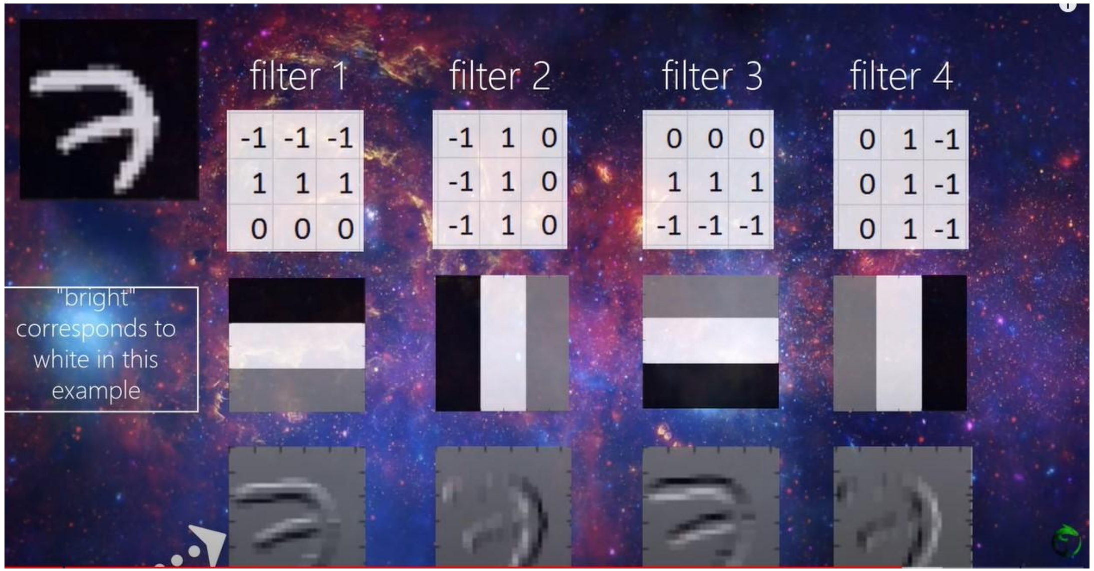


0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.4	0.3	0.5	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.4	0.5	0.9	0.9	0.9	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.9	1.0	0.8	0.8	0.8	0.8	0.8	0.5	0.2	0.2	0.2	0.2	0.2	0.5	0.9	1.0	1.0	0.7	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.1	0.3	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.8	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.6	1.0	1.0	1.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	0.9	0.9	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.5	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.7	1.0	1.0	1.0	1.0	0.9	0.8	0.8	0.3	0.3	0.8	1.0	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.4	0.9	1.0	0.9	0.9	0.5	0.3	0.1	0.0	0.0	0.0	0.0	0.8	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.7	1.0	0.7	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	1.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	1.0	0.9	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	1.0	0.9	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

filter 1/1

0.979	0.278	0.940
0.713	0.048	0.564
0.604	0.327	0.853

######	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.32	0.45	0.61	0.72	0.54	0.36	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.30	0.58	1.18	1.41	1.64	1.64	1.64	1.64	1.91	1.95	2.21	2.31	2.10	2.04	1.69	0.87	0.48	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.47	1.22	1.83	2.57	2.73	3.00	3.00	3.00	3.00	3.39	3.50	3.83	3.97	3.71	3.57	3.16	2.30	1.46	0.54	0.05	0.00	0.00	0.00	0.00	0.00	0.00
1.07	2.05	3.16	4.21	4.39	4.74	4.74	4.54	4.20	4.01	3.85	3.90	3.92	4.10	4.48	4.65	4.06	3.11	1.53	0.54	0.05	0.00	0.00	0.00	0.00	0.00
1.14	2.01	3.13	3.58	3.28	3.22	3.22	3.09	2.91	2.72	2.51	2.51	2.51	2.65	2.99	3.91	4.38	4.12	2.85	1.37	0.33	0.00	0.00	0.00	0.00	0.00
0.92	1.37	2.06	2.17	1.79	1.71	1.71	1.49	1.14	0.82	0.53	0.53	0.53	0.77	1.27	2.43	3.72	4.53	3.98	2.38	0.96	0.00	0.00	0.00	0.00	0.00
0.14	0.34	0.28	0.33	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13	1.29	2.83	4.25	4.66	2.76	1.65	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.35	1.16	2.91	3.93	5.13	3.07	2.19	0.12	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.42	0.97	1.33	1.59	1.91	2.39	3.69	4.36	5.19	3.83	2.50	0.71	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.23	0.49	1.10	1.68	2.29	2.65	2.96	3.41	3.72	4.61	4.91	5.24	4.12	2.53	1.20	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.12	0.67	1.32	1.89	2.57	3.17	3.96	4.43	4.77	4.38	4.22	4.49	4.79	5.19	4.53	2.72	1.60	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.38	0.96	1.82	2.60	3.32	3.77	3.94	3.80	3.65	3.38	2.97	2.93	3.56	4.09	5.00	3.84	2.48	0.97	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.82	1.71	2.96	3.51	3.68	3.30	3.04	2.48	2.23	1.93	1.29	1.28	2.43	3.29	4.83	3.43	2.34	0.56	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.85	1.96	2.72	3.18	2.58	1.77	1.32	0.65	0.38	0.12	0.00	0.39	2.22	3.27	4.63	2.96	2.02	0.16	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.66	1.40	1.65	1.74	0.74	0.24	0.00	0.00	0.00	0.00	0.00	0.83	2.51	3.67	4.25	2.61	1.50	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.07	0.48	0.20	0.48	0.00	0.00	0.00	0.00	0.00	0.00	0.70	1.72	3.28	3.98	3.56	2.23	0.80	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.27	2.27	3.98	3.90	2.82	1.58	0.22	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.30	2.26	3.10	4.46	3.42	2.04	0.80	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.89	2.56	3.36	3.81	2.51	1.22	0.22	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	1.18	2.01	2.84	2.42	1.46	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	1.02	1.34	1.98	1.31	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00



filter 1

-1	-1	-1
1	1	1
0	0	0

filter 2

-1	1	0
-1	1	0
-1	1	0

filter 3

0	0	0
1	1	1
-1	-1	-1

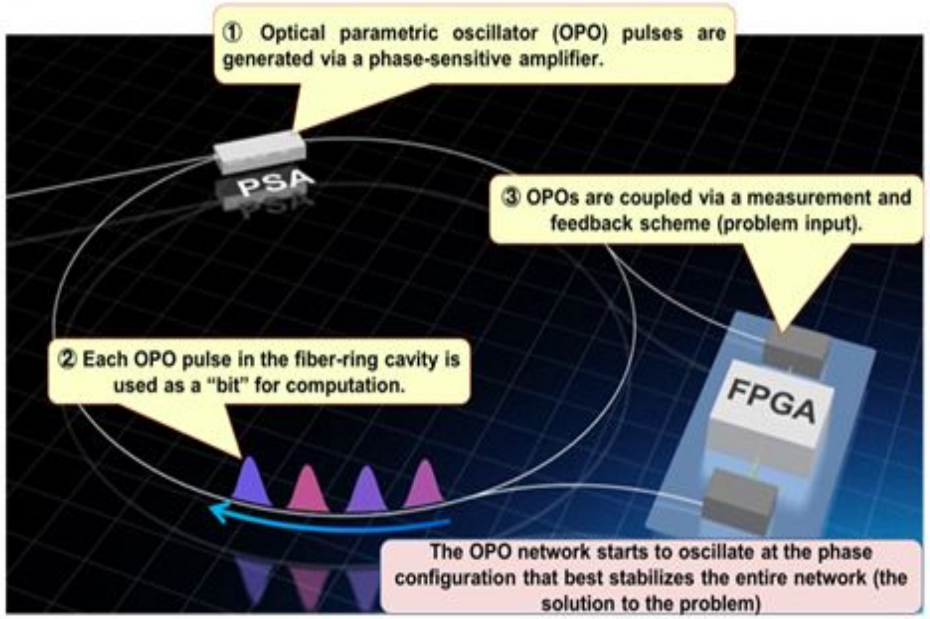
filter 4

0	1	-1
0	1	-1
0	1	-1

"bright" corresponds to white in this example

Quantum Neural Networks : a short introduction

There are three main components : an optical fiber loop, a special optical amplifier called PSA and an electronic circuit called FPGA


① Optical parametric oscillator (OPO) pulses are generated via a phase-sensitive amplifier.
PSA
③ OPOs are coupled via a measurement and feedback scheme (problem input).
② Each OPO pulse in the fiber-ring cavity is used as a "bit" for computation.
FPGA
The OPO network starts to oscillate at the phase configuration that best stabilizes the entire network (the solution to the problem)

How quantum neural networks work - Algorithm analysis

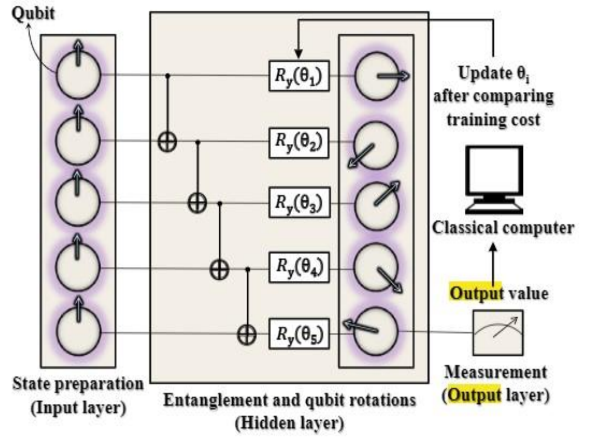
The input of a QNN is encoded with a certain number of qubits

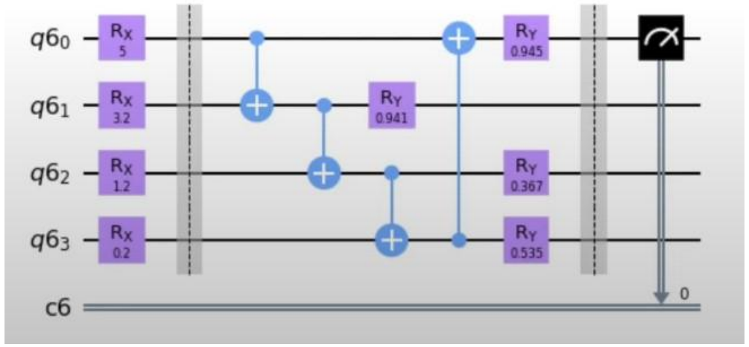
Each qubit is rotated on its axe x by certain degrees (feature mapping)

connecting every qubit i with the one following it (i+1) , if 0<i<N-1 or by connecting qubit 0 with (N-1). This connection, is formed by applying the CNOT gate on the previous qubit connections, allowing the information to flow from qubit to qubit and generating the final result , represented by the output of qubit 0.(Entanglement)

The last step is the rotation of the qubits in y axes with the “trainable angles“ that were first randomly selected in the beginning of the process

Avinash Chalumuri et al. / Procedia Computer Science 171 (2020) 568–575


Qubit
$R_y(\theta_1)$
$R_y(\theta_2)$
$R_y(\theta_3)$
$R_y(\theta_4)$
$R_y(\theta_5)$
Update θ_i after comparing training cost
Classical computer
Output value
State preparation (Input layer)
Entanglement and qubit rotations (Hidden layer)
Measurement (Output layer)



Mathematical approach

The goal is to change the parameter set in order to minimize the difference between the network predictions and input labels, using a loss function.

all data inputs will be labeled in a binary form as either 0 or 1

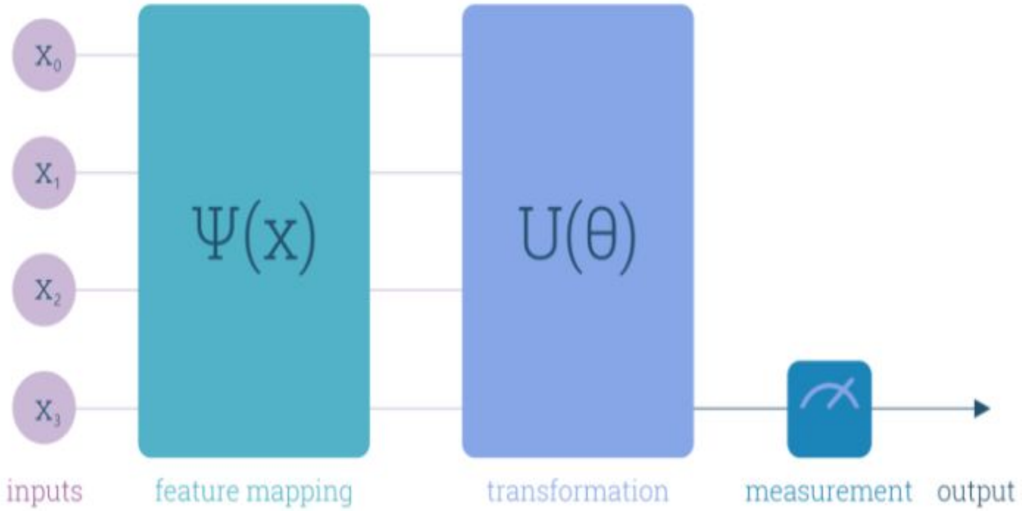
Encode input as a quantum state
x -> |ψ(x) > (with feature map function)

$$\pi(x,\theta) = \frac{1}{2}\,(1 + \langle \psi(x) \,|\, U(\theta)^\dagger Z_{n-1} U(\theta) | \psi(x) \rangle\,)\,) + b$$

$$C(X,\theta) = \sum_{x \in X} l(x,\theta)$$

and the prediction of the output is given by the function

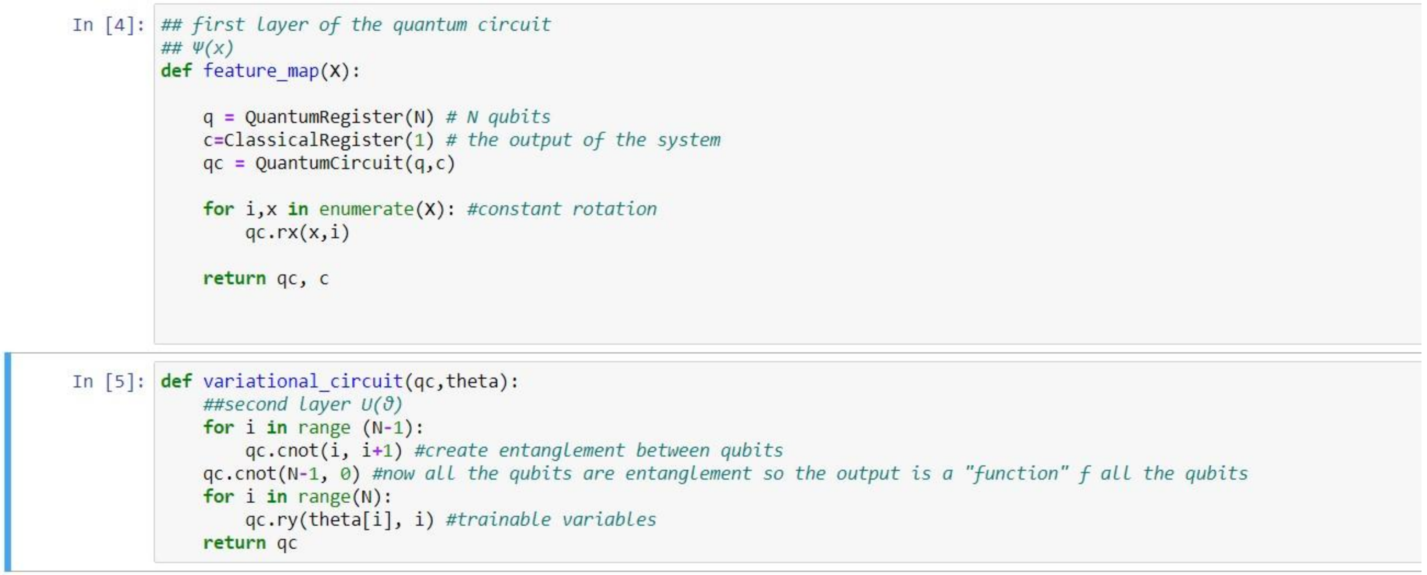
$$p = \begin{cases} 0 & \pi(x,\theta) < 0.5 \\ 1 & \pi(x,\theta) \geq 0.5 \end{cases}$$



Quantum Neural Network implementation with Qiskit

The next few slides contain the Qiskit implementation of a flower classification example.The inputs of the QNN are the Sepal Length, Sepal Width, Petal Length and Petal Width (float numbers) and the output is the kind of the iris that has characteristic as the input values.

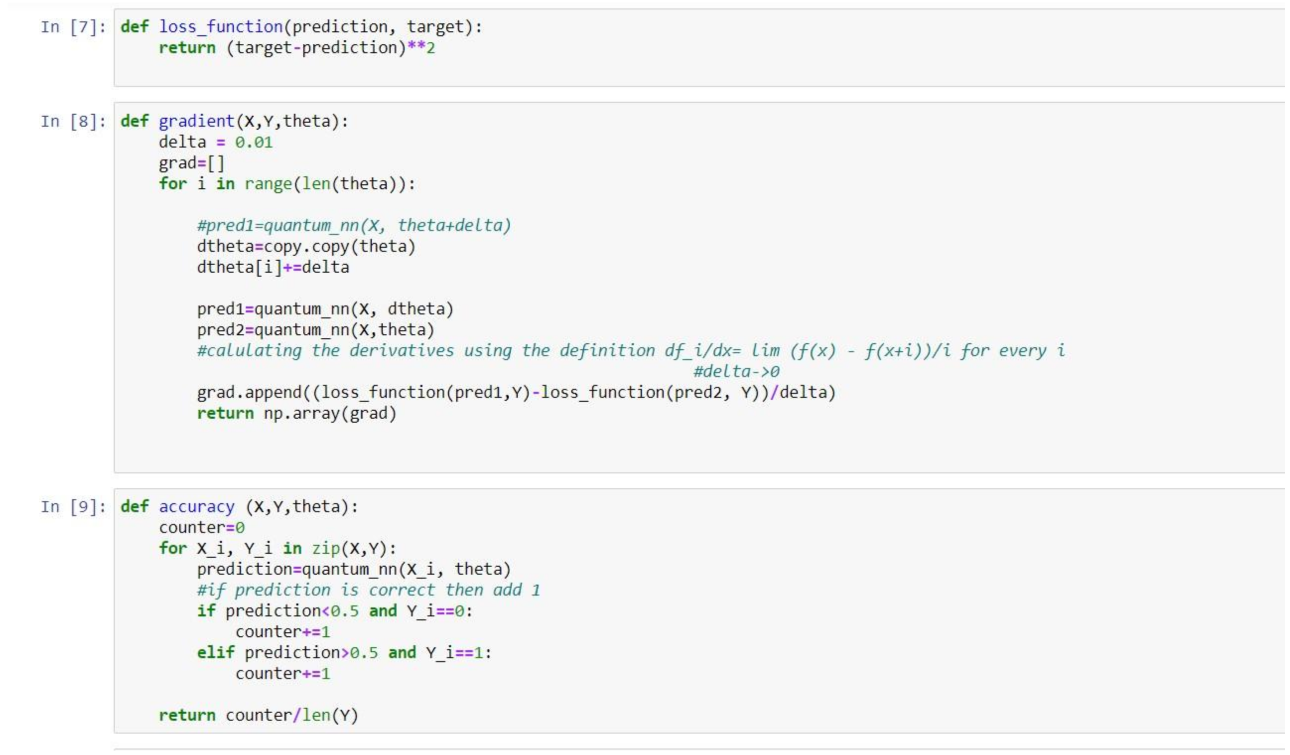
Quantum Neural Network implementation with Qiskit



```
In [4]: ## first layer of the quantum circuit
        ## Ψ(x)
        def feature_map(X):

            q = QuantumRegister(N) # N qubits
            c=ClassicalRegister(1) # the output of the system
            qc = QuantumCircuit(q,c)

            for i,x in enumerate(X): #constant rotation
                qc.rx(x,i)

            return qc, c
```

```
In [5]: def variational_circuit(qc,theta):
            ##second layer U(ϑ)
            for i in range (N-1):
                qc.cnot(i, i+1) #create entanglement between qubits
            qc.cnot(N-1, 0) #now all the qubits are entanglement so the output is a "function" f all the qubits
            for i in range(N):
                qc.ry(theta[i], i) #trainable variables
            return qc
```

Quantum Neural Network implementation with Qiskit

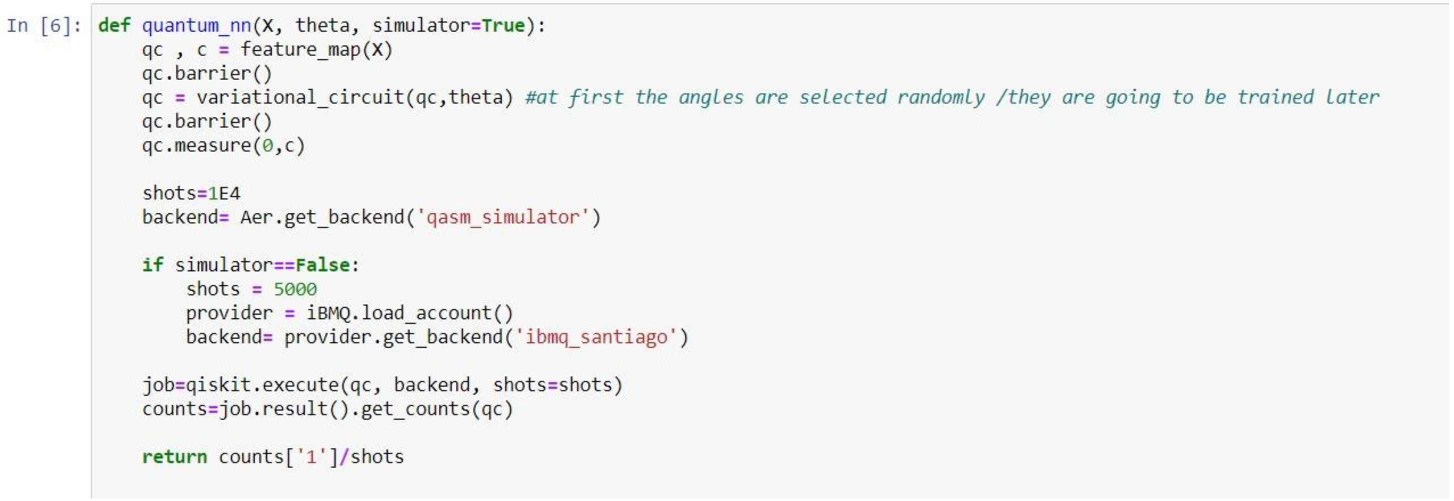


```
In [7]: def loss_function(prediction, target):
            return (target-prediction)**2
```

```
In [8]: def gradient(X,Y,theta):
            delta = 0.01
            grad=[]
            for i in range(len(theta)):

                #pred1=quantum_nn(X, theta+delta)
                dtheta=copy.copy(theta)
                dtheta[i]+=delta

                pred1=quantum_nn(X, dtheta)
                pred2=quantum_nn(X,theta)
                #calulating the derivatives using the definition df_i/dx= lim (f(x) - f(x+i))/i for every i
                                                                     #delta->0
                grad.append((loss_function(pred1,Y)-loss_function(pred2, Y))/delta)
                return np.array(grad)
```

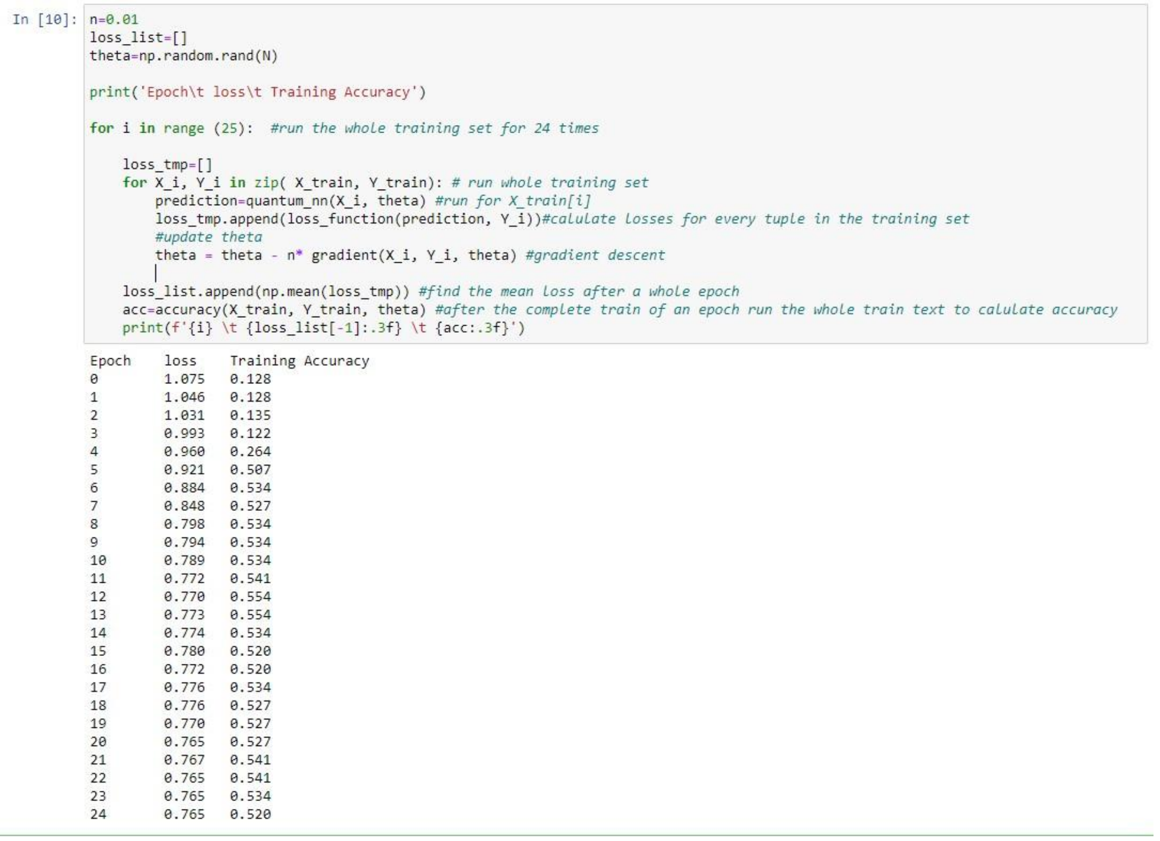
```
In [9]: def accuracy (X,Y,theta):
            counter=0
            for X_i, Y_i in zip(X,Y):
                prediction=quantum_nn(X_i, theta)
                #if prediction is correct then add 1
                if prediction<0.5 and Y_i==0:
                    counter+=1
                elif prediction>0.5 and Y_i==1:
                    counter+=1

            return counter/len(Y)
```

Quantum Neural Network implementation with Qiskit



```
In [6]: def quantum_nn(X, theta, simulator=True):
            qc , c = feature_map(X)
            qc.barrier()
            qc = variational_circuit(qc,theta) #at first the angles are selected randomly /they are going to be trained later
            qc.barrier()
            qc.measure(0,c)

            shots=1E4
            backend= Aer.get_backend('qasm_simulator')

            if simulator==False:
                shots = 5000
                provider = iBMQ.load_account()
                backend= provider.get_backend('ibmq_santiago')

            job=qiskit.execute(qc, backend, shots=shots)
            counts=job.result().get_counts(qc)

            return counts['1']/shots
```

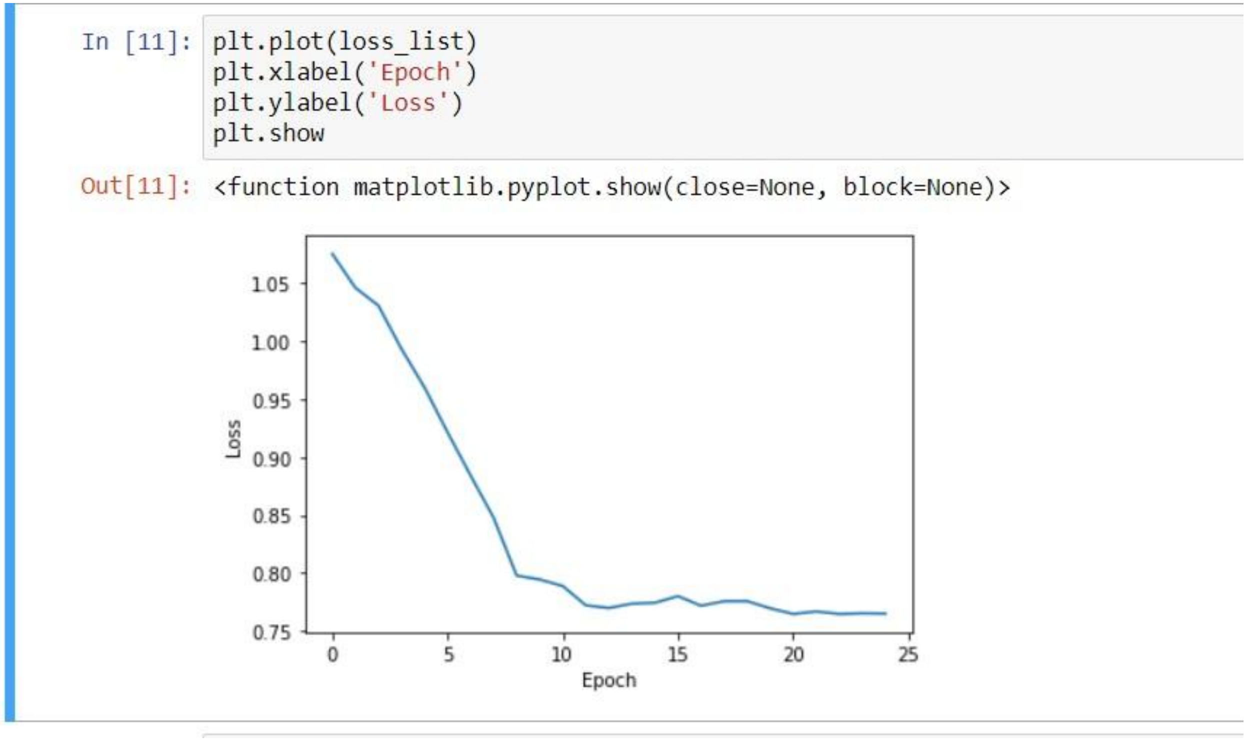
Quantum Neural Network implementation with Qiskit



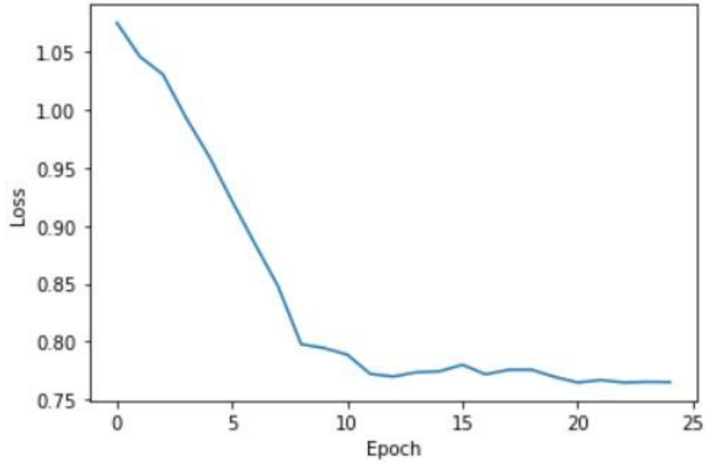
```
In [10]: n=0.01
         loss_list=[]
         theta=np.random.rand(N)

         print('Epoch\t loss\t Training Accuracy')

         for i in range (25):  #run the whole training set for 24 times

             loss_tmp=[]
             for X_i, Y_i in zip( X_train, Y_train): # run whole training set
                 prediction=quantum_nn(X_i, theta) #run for X_train[i]
                 loss_tmp.append(loss_function(prediction, Y_i))#calulate losses for every tuple in the training set
                 #update theta
                 theta = theta - n* gradient(X_i, Y_i, theta) #gradient descent

             loss_list.append(np.mean(loss_tmp)) #find the mean loss after a whole epoch
             acc=accuracy(X_train, Y_train, theta) #after the complete train of an epoch run the whole train text to calulate accuracy
             print(f'{i} \t {loss_list[-1]:.3f} \t {acc:.3f}')
```

```
Epoch    loss    Training Accuracy
0        1.075   0.128
1        1.046   0.128
2        1.031   0.135
3        0.993   0.122
4        0.960   0.264
5        0.921   0.507
6        0.884   0.534
7        0.848   0.527
8        0.798   0.534
9        0.794   0.534
10       0.789   0.534
11       0.772   0.541
12       0.770   0.554
13       0.773   0.554
14       0.774   0.534
15       0.780   0.520
16       0.772   0.520
17       0.776   0.534
18       0.776   0.527
19       0.770   0.527
20       0.765   0.527
21       0.767   0.541
22       0.765   0.541
23       0.765   0.534
24       0.765   0.520
```

Quantum Neural Network implementation with Qiskit



```
In [11]: plt.plot(loss_list)
         plt.xlabel('Epoch')
         plt.ylabel('Loss')
         plt.show

Out[11]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Quanvolutional neural networks

Quanvolutional layers operate on input data by locally transforming it by using random quantum circuits, in a way similar to the transformations performed by random convolutional filter layers

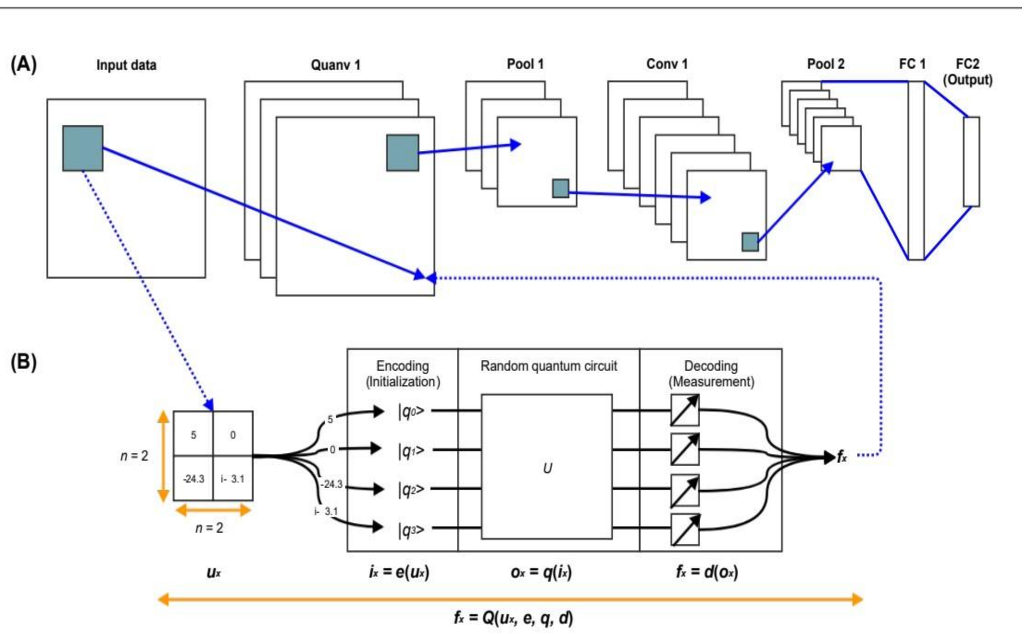
Quite useful algorithm for near term quantum computing, because it requires small quantum circuits with little to no error correction

QNN models have both higher test set accuracy as well as faster training compared to the purely classical CNNs

Quanvolutional neural networks

The main idea behind quanvolution is the replacement of at least one convolutional layer with a quanvolutional one.

The basic functionality remains the same since the final output is identical and the same pulling layers can be utilised by both networks , meaning that QuanvolutionNNs are simply an upgrade of CNNs.

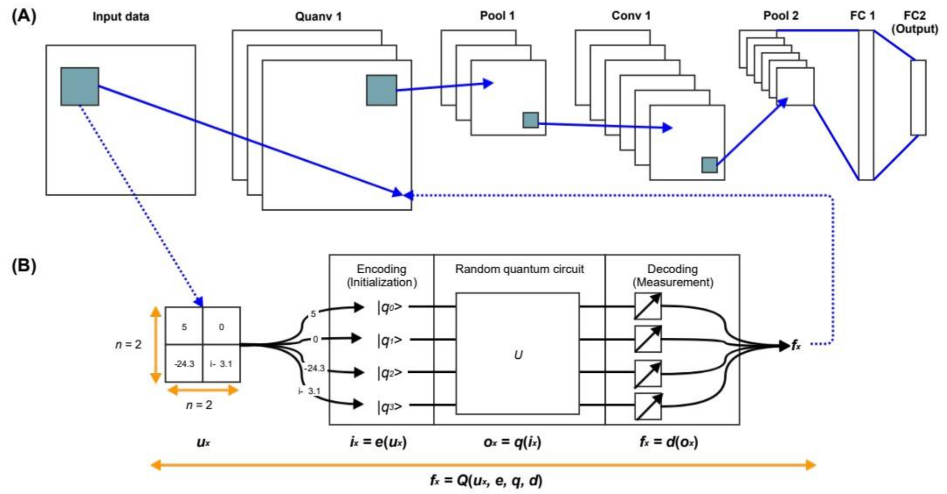


Quanvolutional neural networks

In this picture, network A depicts a QuanvolutionNN with one quanvolutional layer consisting of three quanvolutional filters , one convolution layer with six convolution filters and several pooling and fully connected layers.

Picture B represents a general form of a quanvolution filter . Quanvolution filters are filters that take as an input an n x n matrix and transform the input data using a quantum circuit. This quantum circuit can be structured or random.

With the filtering procedure, the quantum layer (layer that consists of one or more quanvolutional filters) produces a feature map when applied to an input tensor.



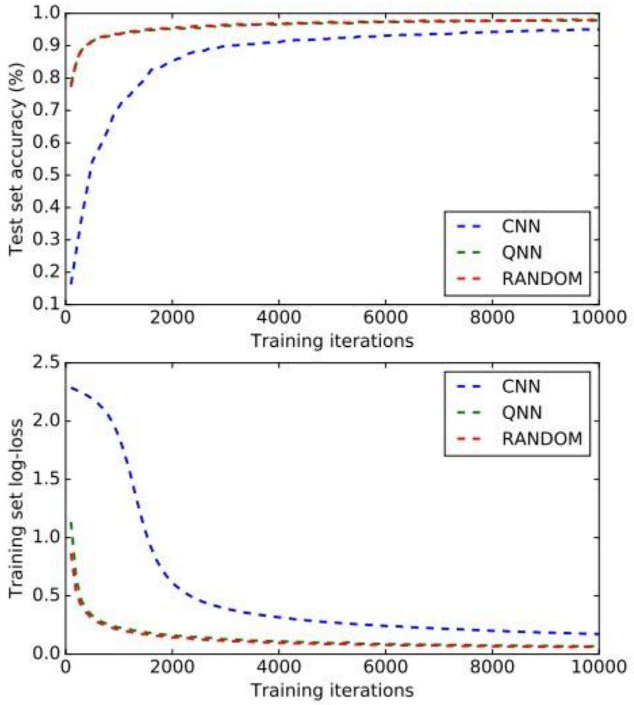
Quanvolutional neural networks : advantages

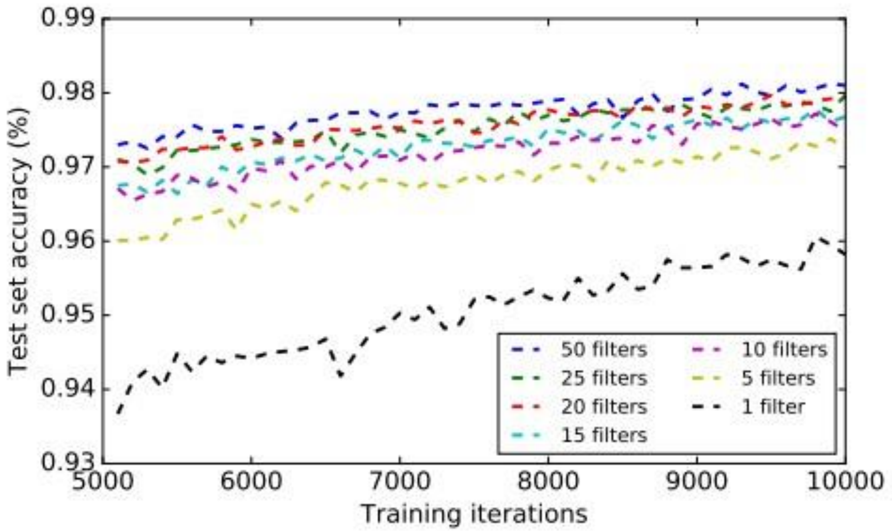
Advantages of QuanvolutionalNNs over CNNs are that features produced by random quanvolutional circuits could increase the accuracy of machine learning models for classification purposes.
QNNs could prove a powerful application for near term quantum computers, which are often called noisy (not error corrected), intermediate-scale (50 - 100 qubits) quantum computers.

The two main reasons causing this are :
(1) quanvolutional filters are **only applied to subsections of input data**, so they can operate using a **small number of qubits** with shallow gate depths and
(2) **quanvolutions are resistant to error**; as long as the error model in the quantum circuit is consistent, it can essentially be thought of as yet another component in the random quantum circuit.

Finally, since determining output of random quantum circuits is not possible to simulate classically at scale, if the features produced by quanvolutional layers provide the modeling advantage that they require quantum devices for efficient computation.

Quanvolutional neural networks : advantages





The addition of numerous filters in a quanvolutional neural network increases its efficiency

Other applications of Quantum Neural Networks

- image compression,
- pattern recognition,
- function approximation ,
- time series prediction,
- electronic remaining useful life (RUL) prediction ,
- vehicle classification
- curve fitting
- temperature control and many more.

It can be easily stated that quantum neural networks can have life-altering applications in multiple developing science sectors including biomedics, as they are already being used for breast cancer prediction.

Concluding remarks

Neural networks are based on a very simple idea generating extraordinary results . Artificial neural networks and all of their subcategories have almost arrived at their maximum capabilities since to improve them further would require a greater number of layers and as of that much higher computing power . Most of the problems resolved by classic neural networks are NP complete .
The solution to this stationary situation can be given by quantum neural networks, running at much faster rate than classical ones. The biggest constraint of QNNs is still their limitations of use , since quantum computers are not yet responding to the numeric qubit needs in order to compute algorithms more efficient than the ones in ClassicalNNs.
The scientific community is split in two with some trying to evolve quantum computing and others trying to fit quantum elements and subcircuits into classical NN algorithms.
Such an effort is the Quanvolutional Neural Network research , first presented in 2018 .

Thank you for your attention !