Machine Learning : Neural Networks in AI and Quantum computing

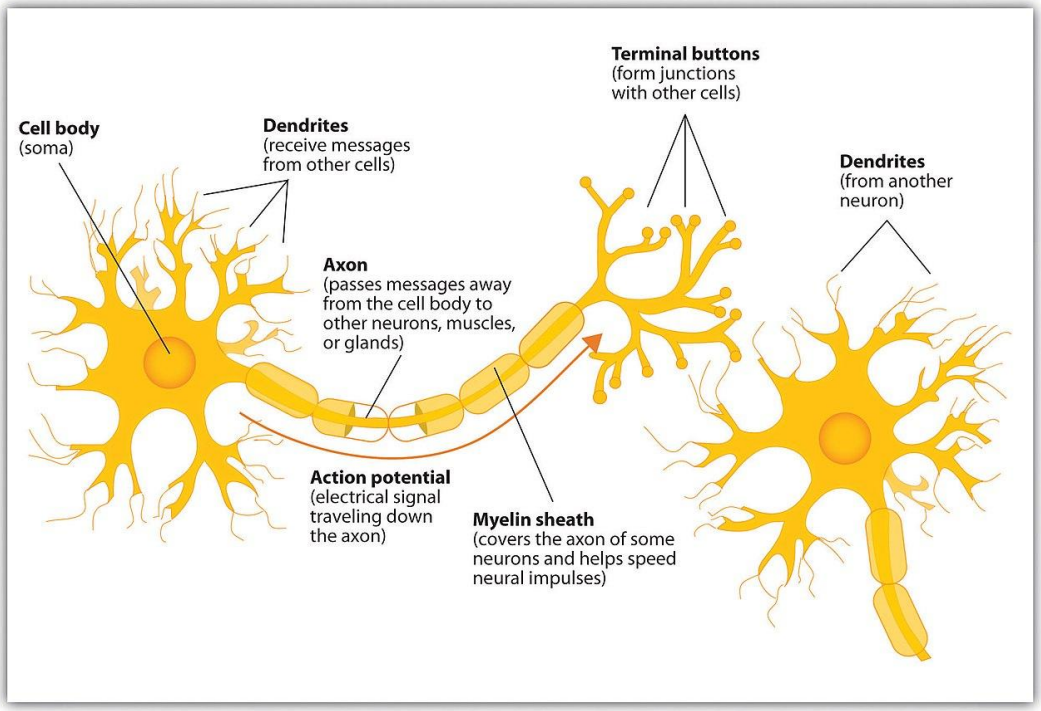
A project by Leonidas Bakopoulos and Alexandra Tsipouraki
for the class of Quantum Technology (ΦΥΣ602)
Technical University of Crete ,
July 2021

Abstract

The purpose of this paper is to highlight the multiple uses of neural networks beginning with machine learning and following with quantum mechanics. Let's first get an insight of where it all begun by answering some fundamental questions . What is a neuron ? Where is the implementation of neural networks based on and how similar is their form to the one we are more familiar with, the biological form. 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 ? After answering all these questions for classical and quantum neural networks, a detailed comparison between quantum vs classical neural networks will follow and finally two forms of an application of neural networks in classical and quantum machine learning will be addressed , the convolution and quanvolution ones.

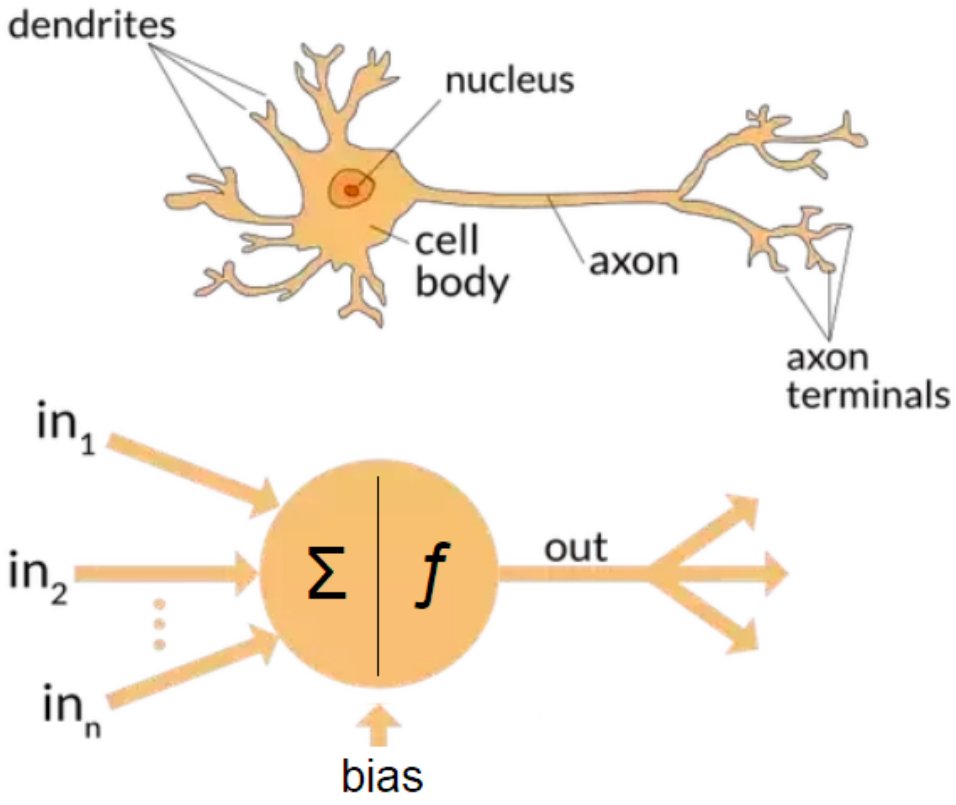
Chapter 1 : Neurons

To comprehend neural networks , it is first mandatory to understand the structure they are based on and inspired by; the human neurons.
Neurons (also called neurones or nerve cells) are the fundamental units of the brain and nervous system, the cells responsible for receiving sensory input from the external world, for sending motor commands to the muscles, and for transforming and relaying the electrical signals at every step in between. New neurons are thought to be constantly created in a person's brain via a process called neurogenesis.
What does a neuron look like?



A useful analogy is to think of a neuron as a tree. A neuron has three main parts: dendrites, an axon, and a cell body or soma , which can be represented as the branches, roots and trunk of a tree, respectively. A dendrite (tree branch) is the receiving part of the neuron , where synaptic inputs arrive from axons with the sum total of dendritic inputs determining whether the neuron will fire an action potential ; an electrical message that is sent throughout the entire axon, when a neuron wants to interact with another neuron. Dendrites branch as they move towards their tips, just like tree branches do, and they even have leaf-like structures on them called spines.The axon (tree roots) is the output structure of the neuron.

Chapter 2 : Neural networks in classic machine learning



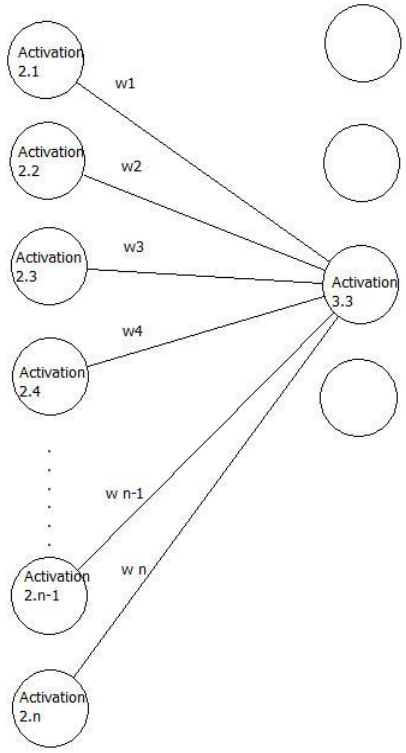
A biological and an artificial neuron (via https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network)

The human neurons are connected with the action potential, defining our every move and reaction to all external stimuli. These connections inspired the creation of neural networks in machine learning. But of course, as birds inspired the creation of airplanes, it can be easily stated that neurons were just the first idea behind neural networks.

To get into further detail, an artificial neural network exploits the basic function of neurons, as it collects multiple information and sends it to other neurons. As a result, the neurons decide the body's next move based on a collection of various, input information.

In 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" . Layers can be described as an array of M - neurons. Each layer solves a sub problem so M-layers can create M-subproblems, creating a much easier way to describe and solve it.

The connection between layers is achieved with the use of the "weighted wires" that are virtual connections between nodes of two successive layers (for example each node of layer (n-1) is connected to every node in layer n).



Specifically neurons themselves are replaced by nodes that hold a specific number per application called “Activation”, which is calculated by the following expression :

$Activation_{3.3}$ =$\sum_{i} Activation_{2.i} * w_{i}$ and more specifically $Activation_{3.3}$ =

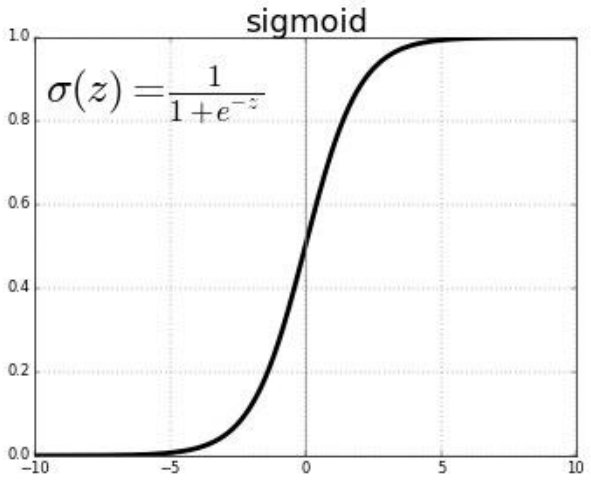
$f(\sum_{i} Activation_{2.i} * w_{i} - bias)$

where f() is a number between 0 and 1 (because activation is within the range 0 to 1) ,
and bias is a number that declares how high the weighted sum needs to be before the neuron starts getting active .
w_i is a weight (a number $\epsilon[0, 1]$ or $sometimes$ $\epsilon[-1, 1]$) responsible for the accurate function of the neural network and its calculation can be properly shown by citing a simple arithmetic example.

Assuming that a neural network consists of two layers with sizes 784 and 16 respectively, it is necessary to manually initialize 784 X 16 =12.554 values of weights , something almost impossible to be performed correctly. So in order to enter correct inputs as weights a specific machine learning technique was developed, known as “network training” . This technique , randomly sets 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. If the output is the expected, the

values are thought to be correct , in any other case , more testing is conducted and weight values are changed based on a specific function that differs from problem to problem , until the demanded output is extracted. After this procedure is concluded, the network's training can be marked as complete and the network can proceed to solve its respective problems.

Finally , 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 , as it can be seen in the graph below . The mathematics hidden behind the networks' training are going to be further analyzed in chapters 2.1 and 4, The methods for neural and quantum neural network training are practically identical.



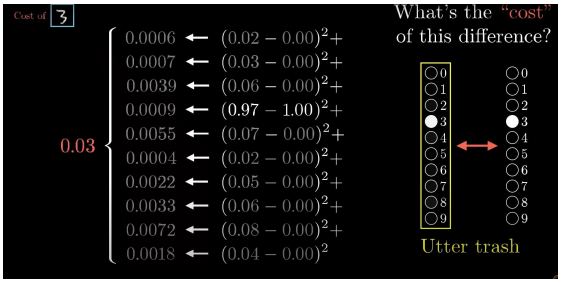
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

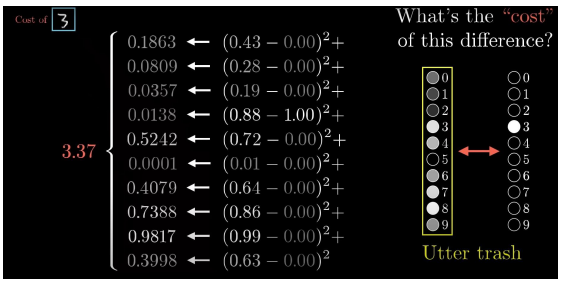
Chapter 2.1 : Training a neural network

As it has already been mentioned, the weights or angles of a neural network are initialized with random values and the network gradually develops a training function in order to calculate and assign to them the best possible values.
For this purpose , a variety of functions can be considered effective, but the most commonly used is a mathematical method called gradient descent and it has almost the same implementation in both quantum and classical neural networks.
First of all a cost function (

$C(x, \theta) = \sum(untrainedOutputs(x, \theta) - correctOutput(x))^2$) is

generated, where untrainedOutputs(x,θ) is the output of the neural network when fed data x with parameters θ and correctOutput(x), is the desired output of the neural network when data x is inserted.





Via https://www.youtube.com/watch?v=IHZwWFHWa-w

The bigger the C(x, Θ), the worst the neural network is performing, as shown in the figures above .
In order to increase the performance of the neural network a change in θ is needed, since θ, represents the weights or filter parameters or angles of the network's input data.
Using the cost function as defined, in order to achieve the best performance it is needed to assign to θ, the values that achieve the minimum cost : min(C(x,θ)).
It can be easily proved that C(x,θ)=min(C(x,θ)) (assuming that x is constant) when θ satisfies the equation : $\nabla C(x,\ \theta) = \nabla C(\theta) = 0$.
So in order to gradually achieve the desired result , θ must be set as
$\theta i_{new} = \theta i_{old} - n\frac{\delta C(x,\theta)}{\delta \theta i}$, with n constant and 0<n<0.2

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.

Chapter 3 : Applications of Neural Networks in machine learning and Convolutional Neural Networks

Neural networks have been successfully applied to a variety of fields , generating many useful applications as it is depicted in the table below :

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

Source : http://www.xenonstack.com/blog/artificial-neural-network-applications

Nevertheless , this paper is going to analyse a specific application of neural networks based on the action of convolution on a neural network. Based on the previously referred idea , in deep learning a class of deep neural network has been developed , named convolutional neural network (CNN). 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.

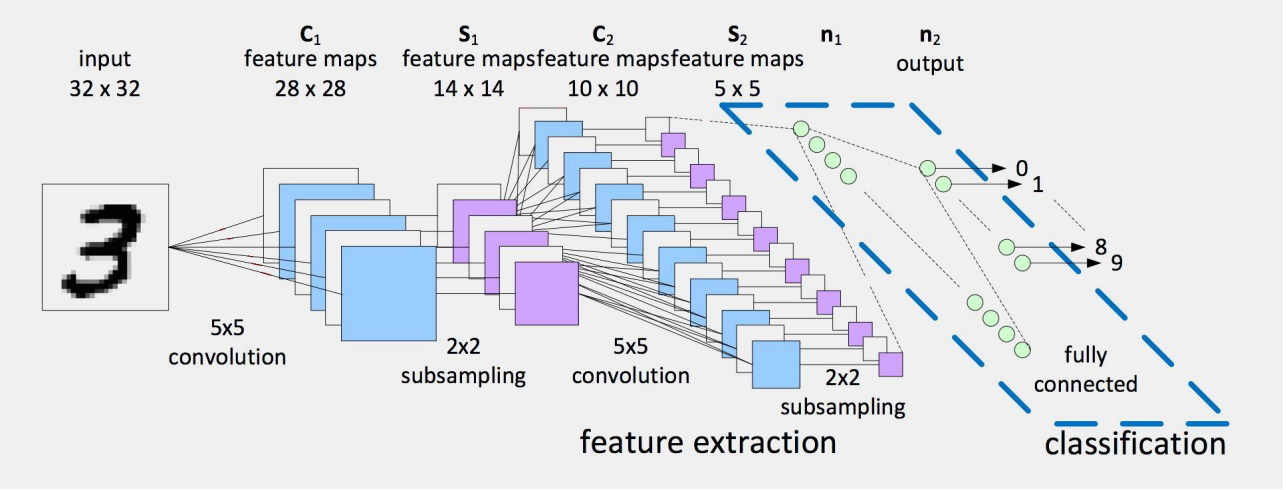
This specific form of NN is most commonly applied for pattern recognition within images , allowing the insertion of image-specific features into the architecture of the network , making it more suited for image-focused tasks while simultaneously reducing the complexity of setting up the model . The fact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

Chapter 3.1 : Convolutional Neural Networks : implementation overview

After explaining the basic idea behind CNNs , now follows a simple theoretical approach .

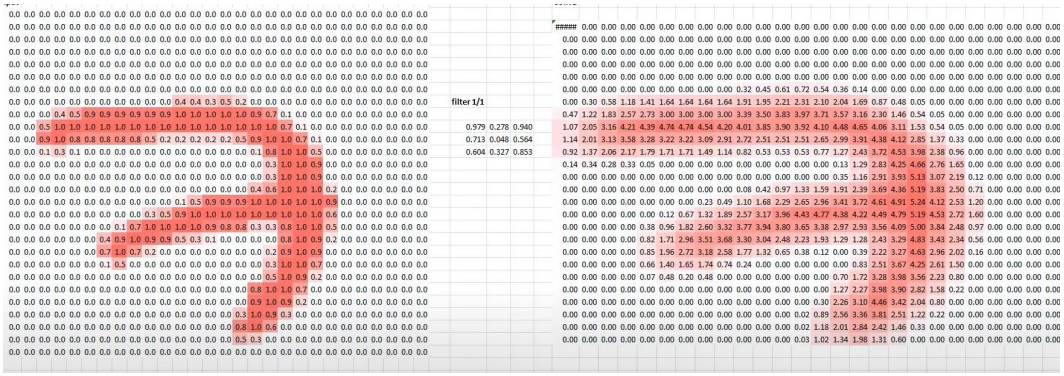
➔ Input of a CNN

In CNNs a whole image is used as an input instead of an array of pixels.That is because we need to evaluate some pixels several times for better pattern detection.



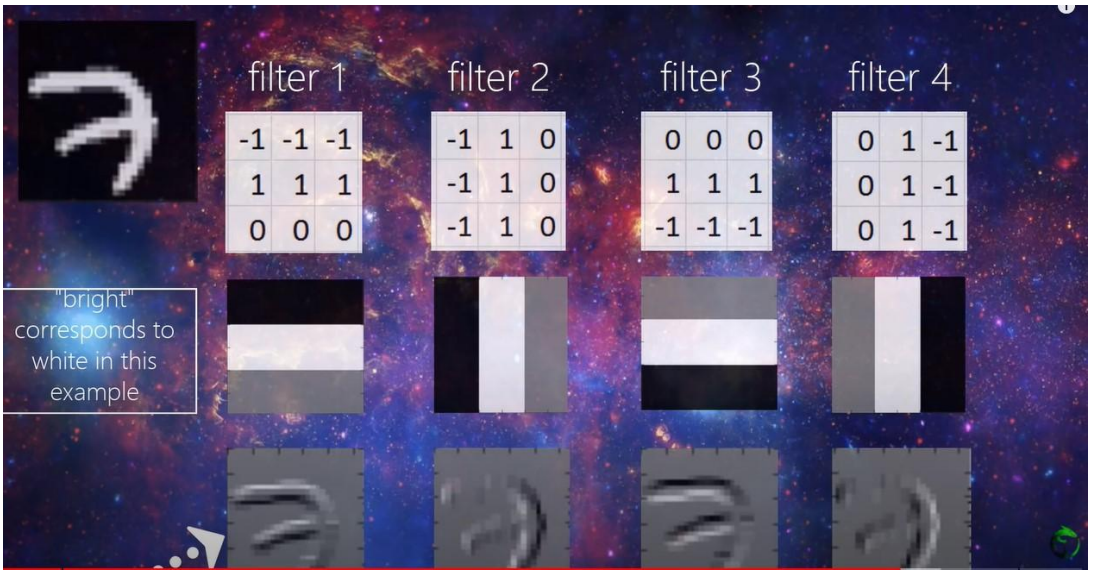
Matrices aka kernels are used instead of nodes (a couple of neighbor pixels are evaluated instead of a single pixel) and filters(aka matrices with weights) are used instead of weights as to easily detect patterns, such as lines or circles.

In this model , there are also two basic categories of hidden layers:
The convolutional layer, for pattern detections and the pooling layer for combining all fitting patterns , such as two lines to produce a bigger one (a triangle composed by three lines). Many possible repetitions of convolution and the pulling layer may be performed before arriving to the last layer, which is fully connected and whose role is to merge different types of patterns in order to produce a more complex one and finally generate the expected output.



Via https://www.youtube.com/watch?v=YRhxdVk_sIs&t=69s

The figure above depicts the input of the CNN (left image) and a layer's output(right image) when convolution is applied.



Via https://www.youtube.com/watch?v=YRhxdVk_sIs&t=69s

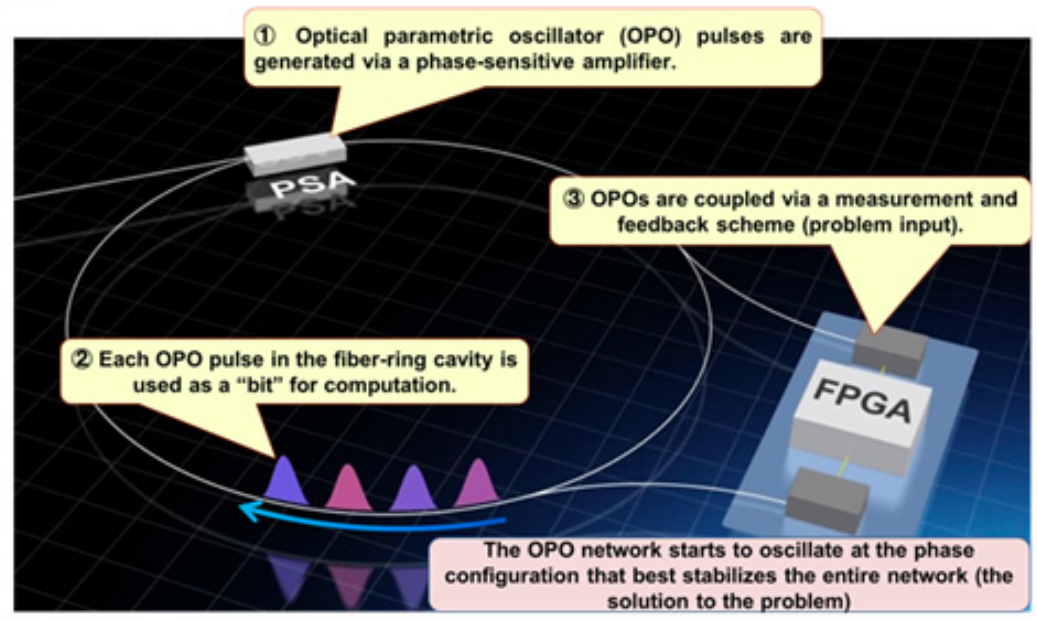
This picture shows an example of four filters utilized to detect either horizontal or vertical lines.

A small example of a simple convolution act, written in pseudocode :

```
convolutionOfMatrices{
        for int i=0; i< size i++{
                sum+=A[i]*B[i]
        }
        return sum;
```

}
}

Chapter 4 : Quantum neural networks : a short introduction



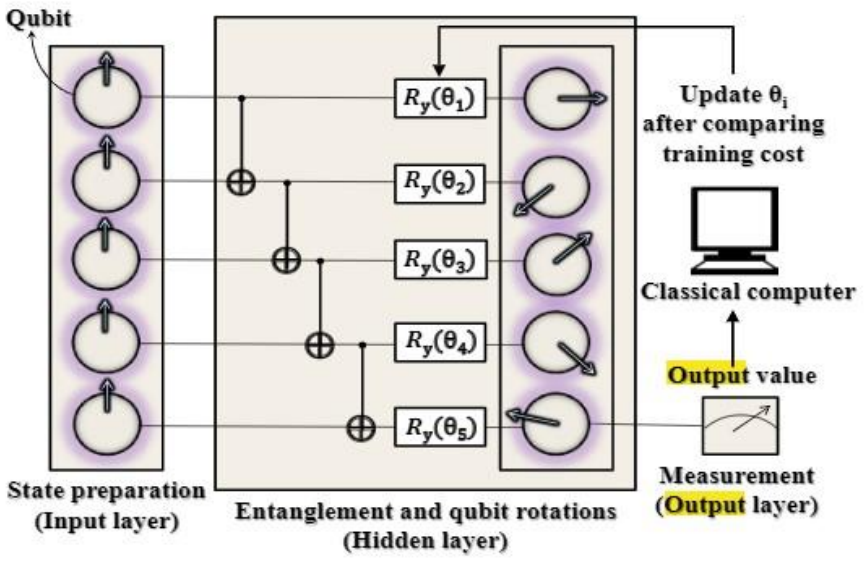
Via https://www.ntt.co.jp/news2017/1711e/171120a.html

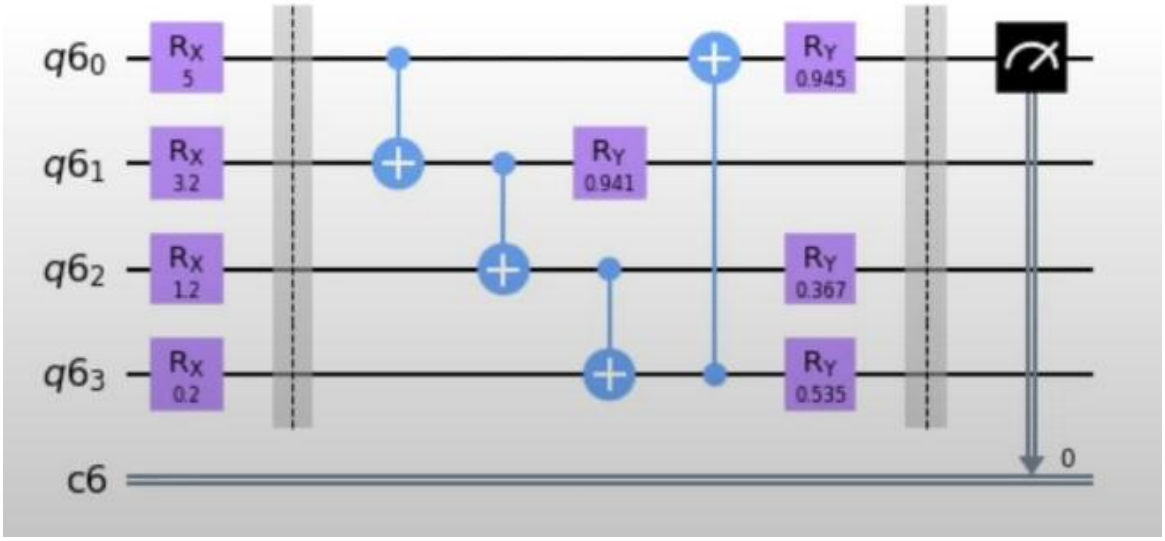
A quantum neural network is technically a computer, whose basic structure is simple. There are three main components : an optical fiber loop, a special optical amplifier called PSA and an electronic circuit called FPGA.
The PSA receives pump light and uses its energy to amplify the light whose wavelength is twice the one of the pump light. The PSA most efficiently amplifies lights with phases zero or π ,relatively to the phase of the pump light (pump phase) .
With the prior system, pulses of pump light are given as an input into the PSA . The PSA then outputs noise optical pulses varying randomly in phase from 0 to 2π .The computer is initialized intentionally with random phases. Using the measurement results of optical pulses, the FPGA creates a new pulse according to a theoretical model called the easing model and superimposes it on the original light pulse. As a result, the pulses affect each other to have either the same or the opposite phase. The FPGA repeats this process each time the light pulse circles the loop formed by the components of the computer until due to the effects of the PSA each pulse approaches either 0 or π. When it is determined to be either 0 or π, the computation result is produced. This system can help us solve a variety of problems, especially those that require binary results (yes or no, true or false etc).

Chapter 4.1 : How quantum neural networks work - Algorithm analysis

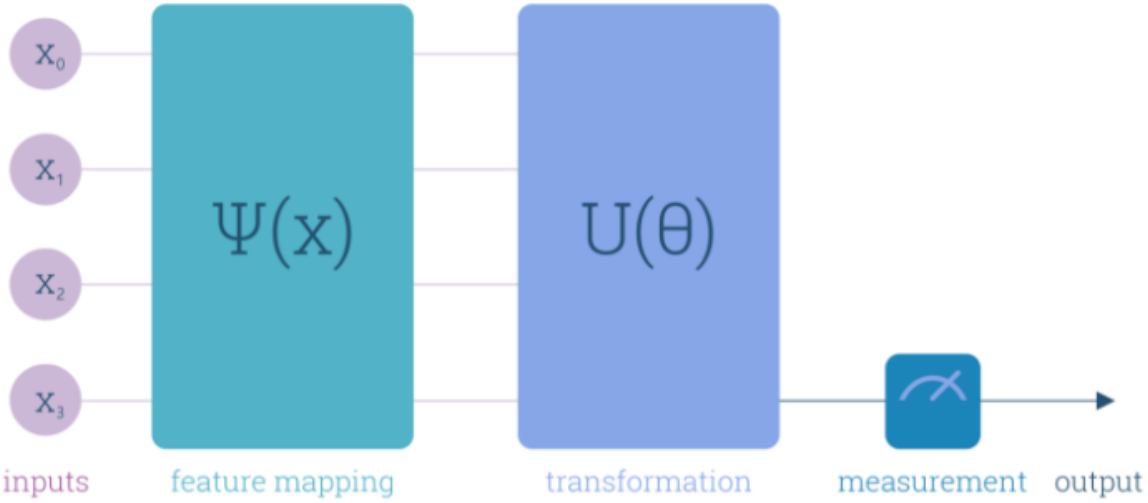
At first , the input of a QNN is encoded with a number of qubits (N) .
Each qubit is rotated on its axe x by certain degrees, specifically assigned to each qubit at the systems' input, that will remain constant and are not going to be trained by the network.This first step is called the feature mapping step. It is worth mentioning that this is just a simple way of setting an input in the quantum network without meaning the absence of many other, eventually more efficient ones, depending on the application they are used for.
The second step consists in the creation of the entanglement that is achieved by 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.
The next step consists of the rotation of the qubits in y axes with the "trainable angles" that were first randomly selected in the beginning of the process.These angles are trained in order to increase the performance of the network until the system achieves the desired result.

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





Chapter 4.2 : Mathematical approach



Via
https://towardsdatascience.com/quantum-machine-learning-learning-on-neural-networks-fdc03681aed3

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.
To simplify the analysis , all data inputs will be labeled in a binary form as either 0 or 1. Data X is used as an input to the network and transformed into the input quantum state with the help of the feature map function .

$x \rightarrow |\psi(x)\succ$

A feature map function can take almost any form ; its purpose being the rotation the axis x by input degrees, as it has already been referred.
Once x is encoded as a quantum state , a series of quantum gates are applied (U(f)| $\psi(x) >$) and the networks' output , called π(x,θ) , is the probability of the last qubit measures as the |1> state plus a bias term added classically .

$$\pi(x, \theta) = \frac{1}{2}(1 + < \psi(x) \mid U(\theta) \dagger Zn - 1U(\theta)|\psi(x) >)) + b$$

where $Zn - 1$ is a Z gate applied to the last qubit.

Finally, using the quadratic loss function

$l(x, \theta) = \frac{1}{2}||w(x, \theta) - y(x)||^2$, with y(x) being the desired output and w(x,θ) being the network's output when data x with parameters θ is input ,

The cost over all X data is

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

and the prediction of the output is given by the function

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

In order to completely understand the functionality of QNN , it is important to explain once again the computation of gradients but on a quantum computer.

$$\partial\theta i\, l\,(x, \theta) = (\pi(x, \theta) - \psi(x))\partial\theta i\, \pi(x, \theta)$$

The expansion of the last term equals :

$$\partial\theta i\, \pi(x, \theta) = \partial\theta i\, ((1 + < \psi(x) \mid U(\theta) \dagger Zn - 1U(\theta)|\psi(x) >)) + b)$$

If θi =b , the gradient is 1 so $\frac{1}{2}\partial\theta i\, (< \psi(x) \mid U(\theta) \dagger Zn - 1U(\theta)|\psi(x) >)$

Using the product rule the previous expression becomes

$$\frac{1}{2}$$

$$(< \psi(x) |\partial\theta i\ U(\theta) \dagger Zn - 1U(\theta)|\psi(x) >) + (< \psi(x) \mid U(\theta) \dagger Zn - 1\partial\theta i\, U(\theta)|\psi(x) >)$$

Now , since U(θ) consists of multiple gates controlled by many different parameters , the partial derivative of U only calls for differentiation of the gate Ui(θi) ,dependent on θi:
$\partial\theta i\ U(\theta) = Un(\theta)nUn - 1(\theta n - 1)...U_1(\theta_1)$

Then for every U_i is going to be used a G gate that consists of a random 2x2 rotate matrix . Let's say

$$G(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

then we get the derivative :

$$\delta_\theta G(\theta) = \begin{bmatrix} -\sin\theta & \cos\theta \\ -\cos\theta & -\sin\theta \end{bmatrix}$$ or

$$\delta_\theta G(\theta) = G(\theta + \frac{\pi}{2})$$

Afterwards , we apply a Hadamard gate for |0> on *Re(<y|AZn-1B|y>)*

and we get

$$\frac{1}{2}\left(|0\rangle(A + Z_{n-1}B)|\psi\rangle + |1\rangle(A - Z_{n-1}B)|\psi\rangle\right)$$

so the probability of measuring 0 is

$$p(0) = \frac{1}{2} + \frac{1}{2}Re(\langle\psi|AZ_{n-1}B|\psi\rangle)$$

then we appropriately substitute B and A with U(θ) and the derivative of it to finally get the gradient of π(χ,θ) with respect to θi.

Now that the process of training quantum neural networks has been slightly clarified, the Qiskit code for a quantum neural network is going to be apposed, classifying plants (specifically iris). Characteristics of the flower are given as an input and the network is trained for 148 input data then run 25 times.

Qiskit code for a quantum neural network

The first figure, contains the first and the second layer of the quantum circuit as previously mentioned.

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
```

The update on the trainable angles was decided with the gradient descent method.In order to implement this method, the calculation of losses' function gradient is required. The approximation of the gradient was based on the derivative definition.

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)
```

The QNN was constructed by combining the feature mapping Ψ(x) with the trainable circuit U(θ) and by measuring the output .(The QNN ran on a quantum computer in Santiago, Chile)

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
```

The remaining process is to train the neural network in order for it to react with the classification problem. For this training is used a dataset with 148 elements. In order to achieve the goal 25 epoch were needed. An epoch, is a complete training with the full dataset.

In [10]:

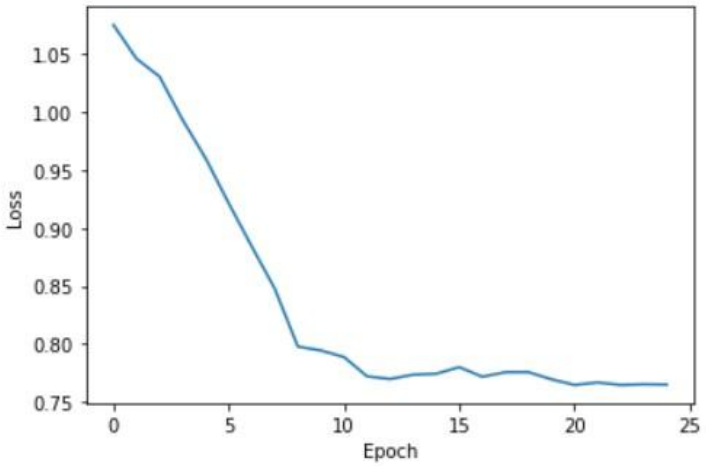
```
n=0.01
loss_list=[]
theta=np.random.rand(N)

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
```

The plot of the loss function through the iteration is similar to the gradient descent plot.

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

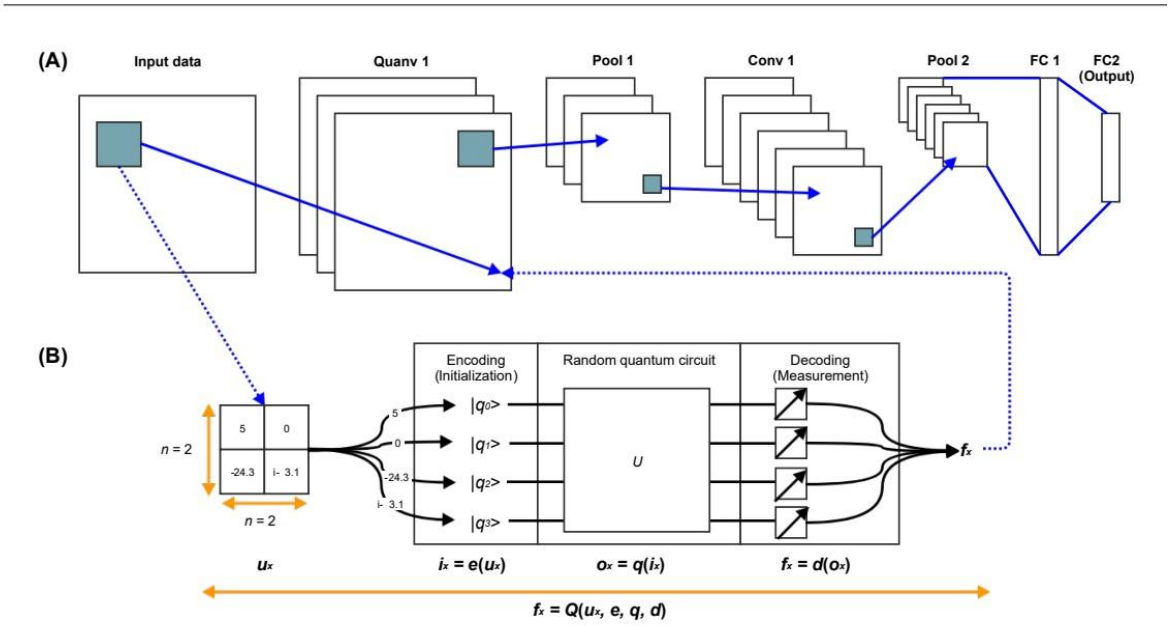
```
Out[11]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Chapter 5 : Quanvolutional neural networks and Qiskit example

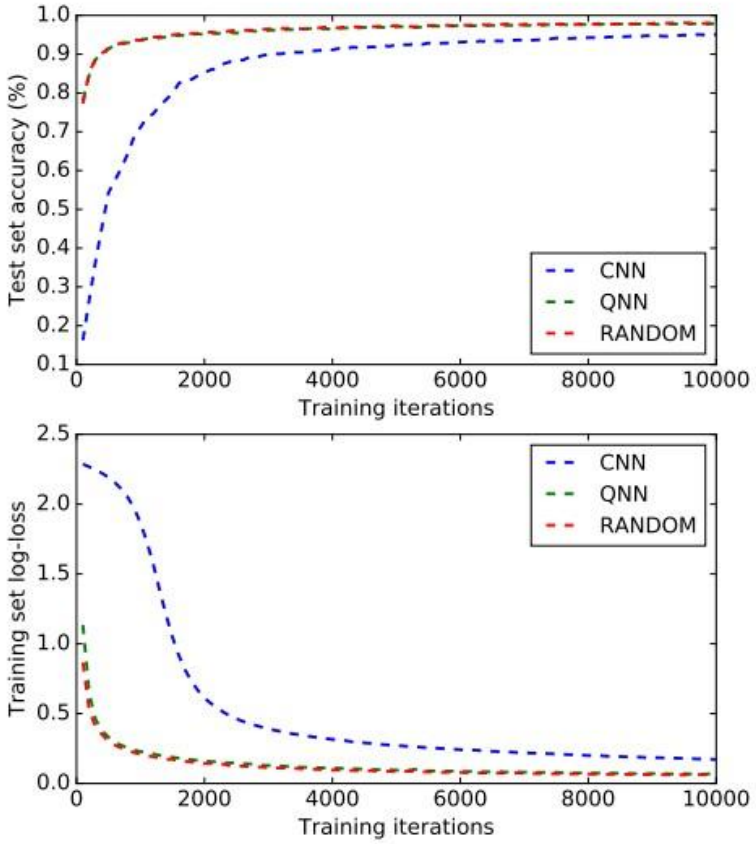
As convolutional neural networks continue gaining affidability in many machine learning applications and especially in the field of image recognition , a new form of CNs is introduced . Quanvolutional layers operate on input data by locally transforming it by using random quantum circuits, in a way similar to the transformations performed by random convolutional filter layers. These quantum transformations produce accountable features for classification purposes. This algorithm could be quite useful for near term quantum computing, because it requires small quantum circuits with little to no error correction. Research(*) has shown that the QNN models have both higher test set accuracy as well as faster training compared to the purely classical CNNs.

The main idea behind quanvolution is the replacement of at least one convolutional layer with a quanvolutional one. The basic functionality of the neural network remains the same since the final output is identical, even though the action of convolution and quanvolution on the network differs. As a result , the same pulling layers can be utilised by both networks , meaning that QuanvolutionNNs are simply an upgrade of CNNs.
In the picture below , 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.

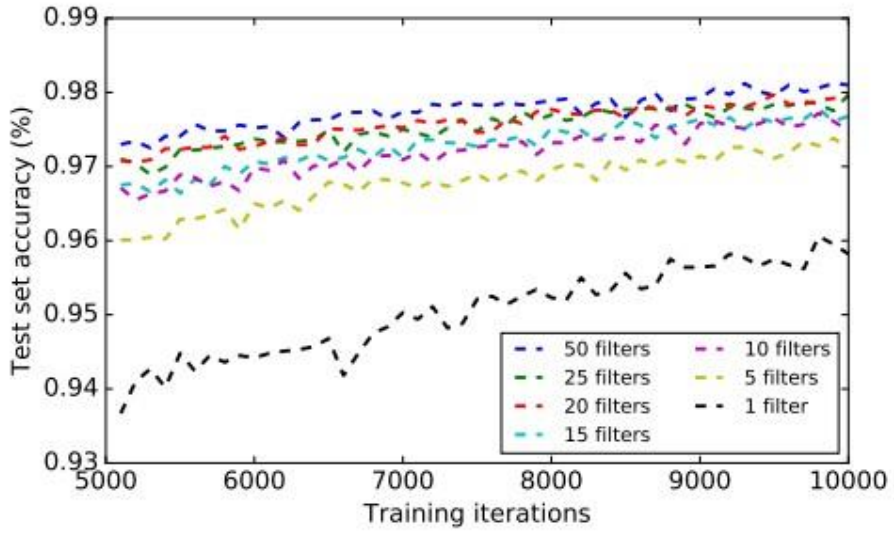


In the picture above , B represents a general form of a quanvolution filter .
Quanvolution filters are filters that take as an input an n x n matrix and transform the input data using a quantum circuit. This quantum circuit can be structured or random. With the filtering procedure, the quantum layer (layer that consists of one or more quanvolutional filters) produces a feature map when applied to an input tensor.
Advantages of QuanvolutionalNNs over CNNs are that features produced by random quanvolutional circuits could increase the accuracy of machine learning models for

classification purposes. Researchers(*) state that if this hypothesis holds true, then QNNs could prove a powerful application for near term quantum computers, which are often called noisy (not error corrected), intermediate-scale (less than 100 qubits) quantum computers. The two main reasons causing this are that quanvolutional filters are only applied to subsections of input data, so they can act by using a small number of qubits with shallow gate depths and secondly that quanvolutions are resistant to error ; as long as the error model in the quantum circuit is consistent, it can 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.



Where
CNN: Convolutional Neural Network
QNN : Quantum Neural Network



As it can be easily seen by the graph above, the addition of many filters in a quanvolutional neural network increases its efficiency.

Quanvolution coding example on Qiskit :

Github Repository : https://github.com/leoBakop/QuanvolutionNN

Chapter 6 : Other applications of Quantum Neural Networks

Quantum neural networks are already being used in numerous applications including 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.

Chapter 7 : 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 .

References :

(*) Quanvolutional Neural Networks: Powering Image Recognition with Quantum Circuits Maxwell Henderson1 ∗ , Samriddhi Shakya1 , Shashindra Pradhan1 , and Tristan Cook1 1QxBranch, Inc., 777 6th St NW, 11th Floor, Washington DC, 20001

Videos

https://www.youtube.com/watch?v=aircAruvnKk&t=17s
https://www.youtube.com/watch?v=5Kr31IFwJiI
https://www.youtube.com/watch?v=2-OI7ZB0MmU
https://www.youtube.com/watch?v=YRhxdVk_sIs&t=69s
https://course17.fast.ai/lessons/lesson4.html
https://www.youtube.com/watch?v=_M2GQAknykg&list=LL&index=2&t=908s&ab_channel=TobiasOsborne
https://www.youtube.com/watch?v=plcKVSId6ak&ab_channel=NTTSCL

Papers

https://arxiv.org/pdf/1904.04767.pdf
https://arxiv.org/pdf/1704.07543.pdf
https://towardsdatascience.com/quantum-machine-learning-learning-on-neural-networks-fdc03681aed3

quantum implementation
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6072506&fbclid=IwAR39pWqnwgDt3mvxTYmsVAaLjA0zVJGLbObHXfOkXsBEHvHJUZOFUFeI0q0
Training an Artificial Neural Network Using Qubits as Artificial Neurons: A Quantum Computing Approach
(figures)

https://towardsdatascience.com/quantum-machine-learning-learning-on-neural-networks-fdc03681aed3

https://www.sciencedirect.com/science/article/pii/S1877050920310280
(figures)

conv n n
https://arxiv.org/pdf/1511.08458.pdf
https://www.cs.toronto.edu/~kriz/imagenet_classification_with_deep_convolutional.pdf

qcnn
https://pennylane.ai/qml/demos/tutorial_quanvolution.html
https://www.nature.com/articles/s41467-020-14454-2.pdf
https://arxiv.org/pdf/1904.04767.pdf

biology
https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron

used for gradient
https://towardsdatascience.com/quantum-machine-learning-learning-on-neural-networks-fdc03681aed3