
Generant Imatges amb un Ordinador Quàntic i Intel·ligència Artificial

TREBALL DE RECERCA DE BATXILLERAT

Autor:

NoSéUnPseudònim

2n Batxillerat

9 de gener de 2022
Barcelona, Barcelona

Índex

I	Introducció	5
II	Marc Teòric	8
1	Àlgebra lineal	9
1.1	Vectors i espais vectorials	9
1.2	Aplicacions lineals	12
1.2.1	Tipus d'aplicacions lineals	13
1.3	Producte interior i producte exterior	15
1.3.1	Producte interior	15
1.3.1.1	Propietats del producte interior	16
1.3.2	Vectors ortonormals i ortogonals	17
1.3.3	Producte exterior	18
1.4	Producte tensorial	19
1.4.1	Propietats del producte tensorial	21
1.5	Traça	22
2	Computació quàntica	24
2.1	Estats quàntics i superposicions	24
2.2	Qubits i operacions quàntiques	28
2.2.1	Representació geomètrica d'un qubit	30
2.2.2	Operacions per a només un qubit	31
2.2.3	Circuit quàntics	34
2.2.4	Operacions per a múltiples qubits	35

2.2.4.1	Entrellaçament quàntic	36
2.2.4.2	Operacions controlades	36
2.3	Mesurament quàntic	38
2.4	Matriu de densitat	40
2.4.1	Matriu de densitat reduïda	41
2.4.1.1	Mesurament parcial	43
2.5	Ordinadors quàntics	44
3	Intel·ligència artificial	45
3.1	Xarxes neuronals	46
3.2	Descens del gradient	49
3.2.1	Backpropagation	52
3.3	Generative adversarial networks	55
4	Generació d'imatges amb un ordinador quàntic	58
4.1	Descens del gradient quàntic	59
4.2	Circuits quàntics per xarxes neuronals	60
III	Marc Experimental	63
5	Plantejament de l'hipòtesi	64
6	Desenvolupament del model	65
6.1	Discriminador	66
6.2	Generador	68
6.3	Creació del model	71
6.4	Execució del model	72
7	Realització del experiment	75
7.1	Anàlisi dels resultats	76

IV	Conclusions	79
V	Annexos	86
A	Que podria fer en un futur?	87
B	Més àlgebra lineal	89
	B.1 Curs ràpid de la notació de Dirac	89
C	Computació Quàntica vs Mecànica Quàntica	90
	C.1 Normalitzar	91
D	Complexitat i algoritmes quàntics	93
	D.1 Algoritme de Grover	94
	D.2 Algoritme de Shor	95
	D.3 Ordinadors quàntics actuals	97
E	Codi	99
	E.1 Part I	101
	E.1.1 Capítol 3	101
	E.1.1.0.1 Regressió lineal	101
	E.2 Part II	102
	E.2.1 Capítol 6	102
	E.2.1.0.1 Codi original per la xarxa neuronal clàssica	102
	E.2.1.0.2 Codi final per el discriminador	105
	E.2.1.0.3 Codi per el generador	109
	E.2.1.0.4 Definició del model	114
	E.2.1.0.5 Execució del model	121
	E.2.1.0.6 Funcions extres	122
	E.2.2 Capítol 7	126
	E.2.2.0.1 Multiprocessament	126

Abstract

Quantum Machine Learning is one of the most promising applications of Quantum Computing. In this work, I present a Quantum Generative Adversarial Network (qGAN) for generating gray-scale bar images. Through the Fréchet Distance score, I evaluate the effectiveness of a partial measurement on the simulated quantum circuit that generates the images. This score shows that the measurement improves qGAN performance by avoiding an oscillation on the resemblance between the generated and the real images, that is, an alternation of good and poor quality generated images that occurs through optimization of the qGAN.

El Aprendizaje Automático Cuántico es una de las aplicaciones más prometedoras de la Computación Cuántica. En este trabajo, presento una Red Generacional Adversaria Cuántica (qGAN en inglés) para generar imágenes de barras en una escala de grises. A través de la puntuación de la Distance de Fréchet, evalúo la efectividad de una medida parcial en el circuito cuántico que genera las imágenes. Esta puntuación muestra que esta medida mejora el rendimiento de la qGAN al evitar una oscilación en el parecido entre las imágenes generadas y las reales, es decir, una alternancia entre imágenes generadas de buena y mala calidad que se da a cabo a través de toda la optimización de la qGAN.

Part I

Introducció

Aquest treball de recerca té l'objectiu de desenvolupar un model de intel·ligència artificial quàntica. Concretament és tracta d'una xarxa neuronal generativa adversària, que té l'objectiu de generar imatges [1, 2]. En aquest model un circuit quàntic és l'encarregat de generar les imatges.

Com a pregunta a investigar m'he proposat verificar la utilitat d'una funció no lineal que implementen els autors en els circuits quàntics. Segons els autors aquesta funció millora el rendiment del model, és a dir, que les imatges són generades amb una major eficiència [1].

Des de fa més de dos anys, m'he dedicat a estudiar computació quàntica durant el meu temps lliure. Buscava investigar un camp relacionat amb la mecànica quàntica, però sense que sigui molt complicat; un camp que pugui entendre a un nivell teòric i que m'entusiasmi.

La Computació Quàntica encaixa perfectament amb aquests criteris. És més senzilla que la mecànica quàntica pel fet que no està basada en càlcul o equacions diferencials. En canvi, es basa en l'àlgebra lineal; utilitzant valors discrets, vectors i matrius. A més a més, si es treballa a un nivell teòric senzill, no s'han de tenir en consideració les interpretacions físiques, aquest factor simplifica molt el procés.

La meva part favorita d'aquest camp és el *Quantum Machine Learning* (QML) que consisteix a dissenyar i aplicar conceptes de *Machine Learning* als ordinadors quàntics, com per exemple implementar en circuits quàntics les famoses xarxes neuronals, que estan darrere de la majoria d'intel·ligències artificials que veiem avui dia [3].

QML és un camp de recerca jove i en creixement, ja que els algoritmes d'aquest camp són ideals per a implementar-los amb els ordinadors quàntics actuals, els quals no són molt potents.

D'entre tots els tipus d'algoritmes que existeixen dintre de QML m'he centrat en les xarxes neuronals quàntiques, anàlogues quàntiques de les xarxes neuronals tan utilitzades avui dia per a fer una gran varietat de tasques. M'he interessat particularment en elles pel fet que tenia experiència en el passat amb les xarxes

neuronals clàssiques, i havia vist que existeixen *frameworks* de programació per a treballar amb elles com *TensorFlow Quantum* [4] que em podien ajudar.

Per a endinsar-me en el camp de QML, vaig haver d'adquirir coneixements en àlgebra lineal, càlcul i física. Un cop aplegats aquests conceptes em vaig dedicar a llegir papers que em cridaven l'atenció. Fins i tot, en un parell d'ocasions vaig intentar implementar alguns algoritmes en *Python*. Tenia molt clar que faria el treball de recerca sobre aquest camp. Durant aquest temps vaig llegir l'article sobre xarxes generatives adversàries (GAN en anglès) quàntiques que em va convèncer per fer un experiment sobre aquests algoritmes [1].

Part II

Marc Teòric

Capítol 1

Àlgebra lineal

Quan vaig començar a buscar informació sobre computació quàntica, vaig ràpidament vaig adonar-me'n que necessitava més coneixements matemàtics, com que no entenia gairebé res dels llibres sobre computació quàntica. Durant aquell temps em van captar l'atenció una sèrie de vídeos sobre àlgebra lineal, que és justament la branca de les matemàtiques sobre la qual es basa la computació quàntica. Aquests vídeos són les lliçons que dona el professor Gilbert Strang a l'Institut Tecnològic de Massachusetts (MIT en anglès) [5, 6]. Una vegada havia vist gairebé tots els vídeos, ja tenia bastants conceptes apresos.

Aquelles lliçons em van ajudar a entendre les matemàtiques de *Quantum Computation and Quantum Information* [7] i *Quantum Computing: A Gentle Introduction* [8]. A poc a poc, vaig anar aprenent els fonaments matemàtics de la computació quàntica i mecànica quàntica.

En aquesta secció explicaré els conceptes bàsics de l'àlgebra lineal amb la finalitat de situar els antecedents teòrics d'aquesta recerca.

1.1 Vectors i espais vectorials

Els objectes fonamentals de l'àlgebra lineal són els espais vectorials. Un espai vectorial és el conjunt de tots els vectors amb les mateixes dimensions i amb

certes propietats en comú. Per exemple \mathbb{R}^3 seria l'espai vectorial de tots els vectors de 3 dimensions els quals normalment s'utilitzen per representar punts en un espai tridimensional. En computació quàntica es fan servir uns espais vectorials anomenats espais de Hilbert, que són aquells en els que hi ha definit un producte interior [9]. En els espais de Hilbert es defineixen un conjunt d'operacions amb certes propietats, les quals explicaré a continuació. S'ha de tenir en compte que els espais de Hilbert són molt més complicats que el que es representa en aquest treball. En endavant, quan mencionï espai vectorial, em referiré a un espai de Hilbert.

Els espais vectorials estan definits per les seves bases, i.e. un conjunt de vectors independents $B = \{|v_1\rangle, \dots, |v_n\rangle\}$. Posat d'una altra manera: el conjunt B és una base pel l'espai V , si cada vector $|v\rangle$ en l'espai es pot escriure com $|v\rangle = \sum_i a_i |v_i\rangle$ per $|v_i\rangle \in B$.

La notació estàndard per l'àlgebra lineal en mecànica quàntica és la notació de Dirac, en la qual es representa un vector com $|\psi\rangle$ (on ψ és la etiqueta del vector). Un vector $|\psi\rangle$ amb n dimensions també pot ser representat com una matriu columna que té la forma:

$$|\psi\rangle = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{n-1} \\ z_n \end{bmatrix}$$

Pels seus elements $\{z_1, z_2, \dots, z_{n-1}, z_n\} \in \mathbb{C}$. Els vectors escrits com $|\psi\rangle$ s'anomenen *ket*.

En els espais de Hilbert es defineix una addició de vectors¹:

$$|\psi\rangle + |\varphi\rangle = \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_n \end{bmatrix} + \begin{bmatrix} \varphi_1 \\ \vdots \\ \varphi_n \end{bmatrix} = \begin{bmatrix} \varphi_1 + \psi_1 \\ \vdots \\ \varphi_n + \psi_n \end{bmatrix}$$

I una multiplicació escalar:

$$z|\psi\rangle = z \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_n \end{bmatrix} = \begin{bmatrix} z\psi_1 \\ \vdots \\ z\psi_n \end{bmatrix}$$

Per a un escalar z i dos vectors $|\psi\rangle$ i $|\varphi\rangle$.

A més a més, hi ha definit un conjugat complex: Per $z = a + bi$, el seu conjugat z^* és igual a $a - bi$.

Aquesta noció es pot ampliar també a les matrius:

$$|\psi\rangle^* = \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_n \end{bmatrix}^* = \begin{bmatrix} \psi_1^* \\ \vdots \\ \psi_n^* \end{bmatrix}$$

$$A^* = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}^* = \begin{bmatrix} a_{11}^* & \cdots & a_{1n}^* \\ \vdots & \ddots & \vdots \\ a_{m1}^* & \cdots & a_{mn}^* \end{bmatrix}$$

Amb $|\psi\rangle$ sent un vector de dimensions n , i A sent una matriu de dimensions $m \times n$.

Un altre concepte important és la transposada, representada pel superíndex T que «rota» un vector o una matriu. Un vector columna amb una dimensió $n \times 1$

¹ Els vectors d'aquesta definició tenen els seus elements representats per la seva etiqueta i un subíndex e.g. el vector $|\psi\rangle$ té un element qualsevol ψ_i i el seu primer element es ψ_1 . Aquesta notació es seguirà utilitzant al llarg del treball.

es transforma a un vector fila amb una dimensió $1 \times n$ ²:

$$|\psi\rangle^T = \begin{bmatrix} \psi_1 \\ \vdots \\ \psi_n \end{bmatrix}^T = [\psi_1 \quad \dots \quad \psi_n]$$

El mateix és cert per les matrius, una matriu $m \times n$ transposada es converteix en una matriu $n \times m$:

$$A^T = \begin{bmatrix} 2 & 3 \\ 6 & 4 \\ 2 & 5 \end{bmatrix}^T = \begin{bmatrix} 2 & 6 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

La composició d'un conjugat complex i la transposada s'anomena el conjugat Hermitià, la seva notació és una \dagger superindexada. Per un vector $|\psi\rangle$ el seu conjugat Hermitià $|\psi\rangle^\dagger$ és:

$$|\psi\rangle^\dagger = (|\psi\rangle^*)^T = [\psi_1^* \quad \dots \quad \psi_n^*] = \langle\psi|$$

El conjugat Hermitià compleix que $|\psi\rangle^\dagger = \langle\psi|$ i $\langle\psi|^\dagger = |\psi\rangle$.

El conjugat Hermitià d'un vector columna $|\psi\rangle$ s'anomena *bra* o vector dual. En la notació de Dirac un vector dual s'escriu com $\langle\psi|$.

1.2 Aplicacions lineals

Per poder operar amb vectors i fer operacions amb ells, s'utilitzen les matrius, que també son anomenades aplicacions lineal o transformacions lineals; noms que descriuen millor com funcionen aquests objectes. La definició formal d'una aplicació lineal pot ser bastant complicada, per aquesta raó, faré servir termes més informals en aquesta secció.

²En realitat els vectors columna son matrius amb dimensió $n, 1$, però he estat ometent el 1. Quan em refereixo a les dimensions d'un vector qualsevol, només diré un nombre, no obstant això, especificaré si és un vector columna o un vector fila.

Bàsicament, una aplicació lineal transforma un vector en un altre vector, que poden o no ser d'espais diferents [10]. Millor dit: per un vector $|v\rangle$ en un espai V i un vector $|w\rangle$ en un espai W , una aplicació lineal A entre els vectors, fa l'acció:

$$A|v\rangle = |w\rangle$$

En altres paraules, aquesta aplicació transforma un element del espai vectorial V en un element de l'espai vectorial W . Les aplicacions lineals han de complir les següents propietats:

1. Addició de vectors:

Donats els vectors $|\psi\rangle$ i $|\varphi\rangle$ en un mateix espai vectorial, i una aplicació lineal A :

$$A(|\psi\rangle + |\varphi\rangle) = A|\psi\rangle + A|\varphi\rangle$$

2. Producte escalar:

Donats els vectors $|\psi\rangle$, l'escalar z i la transformació lineal A , és cert que:

$$A(z|\psi\rangle) = zA|\psi\rangle$$

Aquestes afirmacions han de ser certes per tots els vectors i tots els escalars en els espais on les aplicacions actuen. Cal notar que una aplicació lineal no té perquè ser una matriu necessàriament, per exemple, les derivades i les integrals són aplicacions lineals. Això es pot provar fàcilment al veure que compleixen els criteris especificats posteriorment. Tanmateix, les derivades i les integrals usualment no s'apliquen a vectors, sinó a les funcions, però és possible aplicar-les a vectors.

Les matrius serien la representació matricial de les aplicacions lineals [11].

1.2.1 Tipus d'aplicacions lineals

En la secció actual, exposaré els tipus bàsics d'aplicacions lineals que són indispensables en la teoria presentada en aquest capítol i la resta del treball.

1. Operador zero

Qualsevol espai vectorial té un vector zero expressat en notació de Dirac com a 0 , pel fet que $|0\rangle$ és un altre concepte totalment diferent en CQ i IQ³. També es pot escriure com a 0 . És aquell vector que per qualsevol vector $|\psi\rangle$ i qualsevol escalar z , es compleix que:

- (a) És l'element neutre: $|\psi\rangle + 0 = |\psi\rangle$
- (b) I l'element nul pel producte escalar: $z0 = 0$.

2. Matriu inversa

Una matriu quadrada⁴ A és invertible si existeix una matriu A^{-1} de manera que $AA^{-1} = A^{-1}A = I$, on A^{-1} seria la matriu inversa de A . La manera més ràpida de saber si una matriu és invertible és verificant si el seu determinant no és zero.

3. Matriu Identitat

Per a qualsevol espai vectorial V existeix una matriu I que és definida a partir de $I|\psi\rangle = |\psi\rangle$. Aquesta matriu no fa cap canvi als vectors als quals opera, ni tampoc a les matrius amb les quals es multiplica.

4. Matrius Hermitianes

Una matriu Hermitiana compleix que:

$$\langle\psi|(A|\varphi\rangle) = (A^\dagger\langle\psi|)|\varphi\rangle$$

On $|\psi\rangle$ i $|\varphi\rangle$, són dos vectors de l'espai en el qual actua A . Aquestes matrius o operadors en mecànica quàntica són molt importants, ja que són els observables d'un sistema quàntic. Per exemple l'operador del moviment lineal d'un electró és un operador hermitià.

5. Matriu Unitària

Una matriu unitària és qualsevol matriu que no altera la norma⁵ o longi-

³Computació Quàntica i Informació Quàntica.

⁴Una matriu quadrada és una matriu amb dimensions $n \times n$, on $n \in \mathbb{N}$.

⁵El concepte generalitzat de la longitud d'un vector. Usualment, parlaré de la norma ℓ_2 , esmentada com $\|\cdot\|_2$, i definida com a $\| |\psi\rangle \| = \sqrt{\psi_1^2 + \dots + \psi_n^2}$.

tud dels vectors als quals és aplicada, per tant, una matriu és unitària si $A^\dagger A = I$. Per convertir qualsevol matriu en unitària, es divideixen els seus components entre la norma de la mateixa matriu.

1.3 Producte interior i producte exterior

1.3.1 Producte interior

Un vector dual $\langle a|$ i un vector $|b\rangle$ combinats formen el producte interior $\langle a|b\rangle$, el qual efectua una operació que agafa els dos vectors com a input i produeix un nombre complex com a output [12]:

$$\langle a|b\rangle = a_1 b_1 + a_2 b_2 + \cdots + a_{n-1} b_{n-1} + a_n b_n = z$$

Amb $z, a_i, b_i \in \mathbb{C}$. Cal notar que els vectors $|a\rangle$ i $|b\rangle$ han de tenir la mateixa mida.

Aquest producte per dos vectors en \mathbb{R}^2 també es pot escriure com a:

$$\langle a|b\rangle = \| |a\rangle \|_2 \| |b\rangle \|_2 \cos \theta \quad (1.1)$$

Amb $\|\cdot\|_2$ sent la norma ℓ^2 i θ sent l'angle entre els vectors $|a\rangle$ i $|b\rangle$. Com he dit l'equació (1.1) és equivalent al producte interior. No obstant això, segons el que he vist aquesta equació no es usada àmpliament, ja que interpretar θ com un angle entre vectors de dimensions altes no té molt de sentit.

Usualment, aquest producte es presentat en la seva interpretació geomètrica⁶ com el producte entre un vector fila i un vector columna:

$$\langle a|b\rangle = \begin{bmatrix} a_1^* & \cdots & a_n^* \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

⁶Els detalls exactes de la interpretació geomètrica es troben fora del domini d'aquest treball, malgrat que m'agradaria molt parlar sobre el tema.

Ja he definit la norma ℓ_2 com l'arrel quadrada de la suma dels elements d'un vector al quadrat:

$$\| |a\rangle \|_2 = \sqrt{\sum_i |a_i|^2}$$

Tanmateix, la definició més comuna es basa en el producte interior. Com es pot veure el producte interior d'un vector per ell mateix és la suma dels seus components al quadrat:

$$\langle a|a\rangle = a_1 a_1 + \cdots + a_n a_n = a_1^2 + \cdots + a_n^2 = \sum_i |a_i|^2$$

Per tant, la norma pot ser definida com l'arrel quadrada del producte interior d'un vector [13]:

$$\| |a\rangle \|_2 = \sqrt{\langle a|a\rangle} \quad (1.2)$$

Quan la norma és aplicada a un vector bidimensional, es pot veure que és el mateix que la longitud Euclidiana d'aquell vector, això és perquè realment són els mateixos conceptes. Tot i això, la norma és el concepte de longitud però generalitzat a vectors de dimensions altes.

Segons el que he vist, algunes propietats de la longitud d'un vector bidimensional no es mantenen amb la norma d'un vector que té més de 2 dimensions. En altres paraules, la norma es comporta en certes maneres com la distància des de l'origen (que és la definició de la longitud), però són exactament el mateix. A més d'això, hi ha diferents tipus de normes que s'utilitzen en diversos escenaris. Aquesta és la raó per la qual em refereixo a la norma; com la norma ℓ^2 . A aquesta norma també se li diu norma Euclidiana [14].

1.3.1.1 Propietats del producte interior

Les propietats bàsiques del producte interior són les següents [15]:

1. És lineal en el segon argument $\langle a| (z_1 |b\rangle + z_2 |c\rangle) = z_1 \langle a|b\rangle + z_2 \langle a|c\rangle$
2. $\langle a|b\rangle = (\langle b|a\rangle)^*$
3. $\langle a|a\rangle$ és no negatiu i real, excepte en el cas de $\langle a|a\rangle = 0 \iff |a\rangle = 0$

1.3.2 Vectors ortonormals i ortogonals

A partir del concepte de norma sorgeixen els conceptes d'un parell de vectors ortonormals i un parell de vectors ortogonals. Mirant l'equació (1.2), podem veure que si el producte interior del vector és 1, la norma del mateix vector també és 1. Un vector que té norma 1, és un vector unitari.

Dos vectors diferents de zero són ortogonals si el seu producte interior és zero. Si aquests vectors són bidimensionals, a partir de l'equació (1.1) podem veure que el cosinus de l'angle que fan entre ells és zero, són perpendiculars entre ells:

Per $|a\rangle$ i $|b\rangle \neq 0$:

$$\text{Si } \langle a|b\rangle = 0 \text{ tenim que: } \| |a\rangle \|_2 \cdot \| |b\rangle \|_2 \cos \theta = 0$$

Perquè $|a\rangle$ i $|b\rangle$ no són zero, les seves normes tampoc ho són.

Per tant el terme que falta $\cos \theta$ ha de ser igual a zero.

D'aquesta manera, l'angle θ ha de ser $\frac{\pi}{2}$.

No obstant això, pensar que la perpendicularitat i l'ortogonalitat són els mateixos conceptes és un error, això només és veritat pels vectors bidimensionals. Perquè com en el cas de la norma i la longitud, l'ortogonalitat és el concepte de perpendicularitat però generalitzat [13].

Quan barregem els conceptes de vector unitari i vectors ortogonals arribem a l'ortonormalitat [9]. Un parell de vectors que no són zero, són ortonormals quan els dos vectors són unitaris i també són ortogonals entre ells:

$$|a\rangle \text{ i } |b\rangle \text{ són ortonormals si: } \begin{cases} \langle a|b\rangle = 0 \\ \langle a|a\rangle = 1 \\ \langle b|b\rangle = 1 \end{cases}$$

Els vectors ortonormals són importants. Són àmpliament utilitzats tant en computació quàntica com en mecànica quàntica perquè serveixen per crear bases vectorials que resulten molt útils.

Una altra cosa que remarcar és que al dir un parell de vectors, aquest parell també pot ser un conjunt de vectors. Si un conjunt està compost de vectors unitaris, i aquests són ortogonals entre ells, és un conjunt de vectors ortonormals. El conjunt de vectors $B = \{|\beta_1\rangle, |\beta_2\rangle, \dots, |\beta_{n-1}\rangle, |\beta_n\rangle\}$ és ortonormal si $\langle\beta_i|\beta_j\rangle = \delta_{ij} \forall i, j$ [9] on δ_{ij} és el Kronecker delta, definit com:

$$\delta_{ij} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases}$$

1.3.3 Producte exterior

El producte exterior és una funció (expressada com $|a\rangle\langle b|$, amb $|a\rangle$ i $|b\rangle$ sent vectors) que agafa dos vectors i produeix un operador lineal com output. Al contrari que el producte interior, no hi ha un producte equiparable en les matemàtiques ensenyades a l'institut, i és una mica difícil d'entendre perquè agafa dos vectors que poden ser d'un espai diferent com input. És definit com:

Pels vectors $|v\rangle$ i $|v'\rangle$ amb dimensions m i el vector $|w\rangle$ de dimensions n . El producte exterior és l'aplicació lineal A de dimensions $m \times n$ en l'espai $\text{Mat}_{m \times n}$:

$$|v\rangle\langle w| = A \text{ amb } A \in \text{Mat}_{m \times n}.$$

Amb la seva acció definida per [12]:

$$(|w\rangle\langle v|)|v'\rangle \equiv |w\rangle\langle v|v'\rangle = \langle v|v'\rangle |w\rangle \quad (1.3)$$

A partir de l'equació (1.3) la utilitat i significat del producte són bastant complicats d'entendre, per tant, exposaré la manera de computar-lo per clarificar com

funciona. Per dos vectors $|a\rangle$ i $|b\rangle$ de dimensions m i n respectivament, el seu producte interior es computa multiplicant cada element de $|a\rangle$ per cada element de $|b\rangle$ formant una matriu amb mida $m \times n$:

$$|a\rangle \langle b| = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_n \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_m b_1 & a_m b_2 & \cdots & a_m b_n \end{bmatrix}$$

La utilitat d'aquest producte es mostrarà més endavant.

1.4 Producte tensorial

L'últim producte que mencionaré és el tensorial, representat pel símbol \otimes . Aquest producte s'utilitza per crear espais vectorials més grans combinant espais vectorials més petits. La definició formal és bastant complicada, per tant, em centraré a explicar la manera amb la qual es computa, fent ús de la representació matricial d'aquest producte, anomenada el producte de Kronecker.

Per una matriu $m \times n$, A , i una $p \times q$ matriu B , el seu producte de Kronecker [16] és la matriu de mida $pm \times qn$:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix}$$

Cal tenir en compte que $a_{ij}B$ és una multiplicació escalar per una matriu, amb a_{ij} sent l'escalar i B sent la matriu.

Aquí hi ha un exemple més il·lustratiu amb dues matrius de mida 2×2 , es pot veure que cada element de la primera matriu és multiplicat per la segona matriu:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 1 \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 2 \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \\ 3 \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 4 \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 1 \times 0 & 1 \times 5 & 2 \times 0 & 2 \times 5 \\ 1 \times 6 & 1 \times 7 & 2 \times 6 & 2 \times 7 \\ 3 \times 0 & 3 \times 5 & 4 \times 0 & 4 \times 5 \\ 3 \times 6 & 3 \times 7 & 4 \times 6 & 4 \times 7 \end{bmatrix} = \begin{bmatrix} 0 & 5 & 0 & 10 \\ 6 & 7 & 12 & 14 \\ 0 & 15 & 0 & 20 \\ 18 & 21 & 24 & 28 \end{bmatrix}$$

El producte de Kronecker també funciona de la mateixa manera amb vectors:

Pels vectors $|\psi\rangle$ i $|\varphi\rangle$ de dimensions n i m respectivament:

$$|\psi\rangle \otimes |\varphi\rangle = \begin{bmatrix} \psi_1 |\varphi\rangle \\ \psi_2 |\varphi\rangle \\ \vdots \\ \psi_n |\varphi\rangle \end{bmatrix} = \begin{bmatrix} \psi_1 \varphi_1 \\ \psi_1 \varphi_2 \\ \vdots \\ \psi_1 \varphi_m \\ \vdots \\ \vdots \\ \psi_n \varphi_1 \\ \psi_n \varphi_2 \\ \vdots \\ \psi_n \varphi_m \end{bmatrix}$$

S'ha de tenir en compte que el producte de Kronecker també es pot fer entre un vector i una matriu, o viceversa, no obstant això, no és molt comú fer-ho.

Si a una matriu A s'he li treu el producte tensorial de si mateixa n -vegades es pot escriure com a $A^{\otimes n}$, per exemple: $|\psi\rangle^{\otimes 2} = |\psi\rangle \otimes |\psi\rangle$.

1.4.1 Propietats del producte tensorial

Les propietats bàsiques del producte tensorial són les següents⁷ [17, 18]:

1. Amb un escalar z i els vectors $|v\rangle \in V$ i $|w\rangle \in W$:

$$z(|v\rangle \otimes |w\rangle) = (z|v\rangle) \otimes |w\rangle = |v\rangle \otimes (z|w\rangle)$$

2. Per els vectors $|v_1\rangle, |v_2\rangle \in V$ i $|w\rangle \in W$:

$$(|v_1\rangle |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle$$

⁷Una cosa a notar és que $A \otimes B$ i $B \otimes A$ són permutativament equivalents: $\exists P, Q \Rightarrow A \otimes B = P(B \otimes A)Q$ on P i Q són matrius permutatives.

3. Per els vectors $|v\rangle \in W$ i $|w_1\rangle, |w_2\rangle \in W$:

$$|v\rangle \otimes (|w_1\rangle \otimes |w_2\rangle) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle$$

Aleshores podem definir una aplicació lineal que actua en $V \otimes W$:

$$A \otimes B : V \otimes W \mapsto V' \otimes W'$$

On les matrius A i B són:

$$A : V \mapsto V'$$

$$B : W \mapsto W'$$

Finalment l'aplicació $A \otimes B$ és definida com:

$$(A \otimes B)(|v\rangle \otimes |w\rangle) \equiv A|v\rangle \otimes B|w\rangle$$

Essent $|v\rangle$ un vector en V i $|w\rangle$ en W .

A més a més, és pot definir un producte interior en l'espai $V \otimes W$. Del qual no parlaré perquè és pot ús del producte de Kronecker per computar-lo considero que no és necessari.

1.5 Traça

La traça d'una matriu és tal sols la suma dels seus elements en la diagonal principal, la diagonal que va de dalt a baix i d'esquerra a dreta. Cal notar que la matriu ha de ser quadrada.

Aquí hi ha una matriu A amb la seva diagonal principal marcada:

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

I la seva traça, representada per $\text{Tr}[A]$ és:

$$\text{Tr}[A] = 1 + 1 + 1 = 3$$

Més formalment, la traça d'una matriu quadrada n -dimensional és:

$$\text{Tr}[A] = \sum_{i=1}^n a_{ii} = a_{11} + a_{22} + \cdots + a_{nn}$$

La traça d'una matriu té les propietats següents [19]:

1. És una aplicació lineal:

Degut a que la traça és un aplicació lineal, es compleix que:

$\text{Tr}[A + B] = \text{Tr}[A] + \text{Tr}[B]$ i $\text{Tr}[zA] = z \text{Tr}[A]$, per a totes les matrius quadrades A i B , i tots els escalars z .

2. Traça d'un producte tensorial:

$$\text{Tr}[A \otimes B] = \text{Tr}[A] \text{Tr}[B]$$

3. La transposada té la mateixa traça:

$$\text{Tr}[A] = \text{Tr}[A^T]$$

4. La traça d'un producte és cíclica:

Per una matriu A amb mida $m \times n$ i una matriu B de la mateixa mida:

$$\text{Tr}[AB] = \text{Tr}[BA]$$

Una altra manera molt útil d'avaluar la traça d'un operador és a través del procediment de Gram-Schmidt i el producte exterior. Utilitzant Gram-Schmidt per representar el vector unitat $|\psi\rangle$ en una base ortonormal $|i\rangle$ que inclou $|\psi\rangle$ com el seu primer element, es compleix que [19]:

$$\text{Tr}[A |\psi\rangle \langle \psi|] = \sum_i \langle i| A |\psi\rangle \langle \psi|i\rangle = \langle \psi| A |\psi\rangle$$

Capítol 2

Computació quàntica

Després de la teoria matemàtica, ha arribat el moment de parlar sobre mecànica quàntica. En aquest capítol introduiré alguns conceptes bàsics sobre Informació Quàntica i Computació Quàntica (IQ i CQ).

La Mecànica Quàntica (QM) és un marc matemàtic o conjunt de teories utilitzades per poder explicar les propietats físiques dels àtoms, molècules i partícules subatòmiques. És un marc que engloba tota la física quàntica, incloent-hi la teoria d'informació quàntica. La forma ideal de treballar la CQ es a través dels postulats de la mecànica quàntica [20]. Malauradament això requereix d'una profunditat que en aquest treball de recerca no es pot abastar. Per això he decidit explicar alguns conceptes de MQ a través d'una analogia.

2.1 Estats quàntics i superposicions

Per descriure com evolucionen els sistemes físics a través del temps, es necessita representar els sistemes d'alguna manera. En mecànica quàntica els sistemes es representen per estats quàntics, els quals són un tipus de distribucions de probabilitat que representen els possibles resultats d'una mesura en un sistema quàntic [21].

Imagina't que tens un bolígraf, però que no saps de quin color és, tanmateix, saps que pot ser vermell o blau. Per esbrinar el color, pots provar d'escriure per

veure el color de la tinta, o en altres paraules, fer una mesura del sistema. Saps que hi ha un 50% de probabilitat que sigui vermell i un 50% que sigui blau. En aquesta situació hipotètica tindries el teu sistema quàntic (el bolígraf), una manera de mesurar-lo (escriure) i una llista amb els possibles resultats (50% vermell, 50% blau), només et falta una manera de representar-lo tot matemàticament, l'estat quàntic. Per tant, per què no intentem guardar la informació que sabem del bolígraf en un vector?

Si posem cada probabilitat de treure un resultat en un vector, tenim que:

$$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

On la primera entrada és la possibilitat que el bolígraf sigui vermell i la segona entrada que sigui blau, per fer-ho més senzill aquí està en colors:

$$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

Cal remarcar que aquest vector està normalitzat amb la norma ℓ_1 , definida com la suma dels components d'un vector¹, en altres paraules la norma ℓ_1 d'aquest vector és 1.

Ara hem d'escollir una operació matemàtica per poder extreure la informació del vector, com que l'output ha de ser un nombre, podem provar d'utilitzar un producte interior. Però primer s'ha de representar el vector com una combinació lineal de les seves bases:

$$0.5 |0\rangle + 0.5 |1\rangle = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

On $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ i $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Llavors, per trobar la probabilitat, s'agafa el producte

¹Amb $|a\rangle$ sent un vector, la norma ℓ_1 , denotada per $\|\cdot\|_1$ és $\| |a\rangle \|_1 = \sum_i a_i$.

interior del vector amb la base corresponent a la probabilitat, com a continuació:

$$\langle 0|v\rangle = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 0.5$$

$$\langle 1|v\rangle = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 0.5 \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 0.5$$

Aquest procediment és bastant senzill com es pot veure. Per tindre un exemple una mica més complicat, per representar un bolígraf amb 6 colors possibles amb una possibilitat aleatòria d'escriure amb un color dels 6 possibles, l'estat d'aquest bolígraf podria ser²:

$$|w\rangle = \begin{bmatrix} 0.25 \\ 0.3 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.05 \end{bmatrix}$$

Ara per poder veure la probabilitat de per exemple que el bolígraf sigui de color verd, podem utilitzar la tercera base del vector, la qual correspon al color verd:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.25 \\ 0.3 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.05 \end{bmatrix} = 0.1$$

Deixant els bolígrafs a una banda, els podem substituir per un sistema físic quàntic, com per exemple un fotó. Els fotons tenen certes propietats que poden ser mesurades, com la seva polarització³. Al mirar als fotons com ones que oscil·len

²L'he posat colors als elements per claredat.

³Concretament, són una propietat que les ones transversals tenen, el tipus d'ona de les ones electromagnètiques, que són els fotons en realitat.

en els camps electromagnètics, la polarització és l'orientació geomètrica de l'ona. La polarització pot ser interpretada com un angle respecte a la direcció de propagació.

Al definir les bases de l'estat de polarització com vertical i horitzontal, denotades pels vectors $|\rightarrow\rangle$ i $|\uparrow\rangle$, respectivament. Podem definir un estat de superposició entre les bases, representat com $|\nearrow\rangle$ [22]:

$$|\nearrow\rangle = \alpha |\rightarrow\rangle + \beta |\uparrow\rangle \quad (2.1)$$

On α i β són números complexos. Un estat en superposició és simplement un estat on l'angle de polarització no és 0 ni $\frac{\pi}{2}$. No ens hem de preocupar de la descripció matemàtica exacta de la polarització dels fotons al parlar de computació quàntica perquè el que importa és la informació que porten aquests estats, no la física dels sistemes que representen.

Aquesta informació s'ha d'agafar mitjançant mesures, com amb els bolígrafs. Aquestes mesures en el cas de la polarització dels fotons seria passar els fotons per diversos filtres de polarització, els quals deixen passar el fotó o l'absorbeixen, tot això d'una manera probabilística, és a dir, depenent de quin sigui l'estat tenen certa possibilitat de ser absorbits o no.

Per clarificar, el filtre el que fa és col·lapsar el fotó en els dos possibles estats de polarització, l'estat en el qual el filtre és orientat o l'estat perpendicular a aquest. Si col·lapsa en l'estat del filtre el fotó passa pel filtre, en cas de col·lapsar en l'altre estat, el fotó és absorbit. La manera en la qual els fotons col·lapsen és probabilística, si agafem un filtre que està orientat horitzontalment respecte als fotons que li arriben, un fotó orientat horitzontalment té un 100% de possibilitats de poder passar, mentre que un fotó polaritzat verticalment té un 0% de probabilitat de poder passar. I si un fotó està polaritzat en un angle just entre vertical i horitzontal, és a dir a $\frac{\pi}{4}$, tindrà un 50% de possibilitats de passar i un 50% de no poder-hi.

Aquesta és la manera amb la qual els sistemes quàntics es comporten, a través de la probabilitat. Els estats que els representen simplement tenen la informació sobre quines són aquestes possibilitats. No obstant això, la polarització

dels fotons són un sistema físic concret, quan es parla de computació quàntica és millor treballar amb conceptes més generals per poder expressar tants algoritmes com sigui possible i poder implementar aquests algoritmes en tants ordinadors quàntics com sigui possible. Per aquesta raó, en la branca de la informació quàntica i la computació quàntica es treballa amb qubits, en comptes de diversos sistemes físics.

2.2 Qubits i operacions quàntiques

Els ordinadors moderns representen la informació a través de cadenes de zeros i uns, anomenades bits; des d'imatges a lletres o instruccions electròniques. Per exemple, la lletra t és representada per la cadena de bits 01110100, codificada a través de codi binari. Tot el que fas en un ordinador es codifica i representa en codi binari.

Gràcies al fet que estem molt acostumats al codi binari, en el camp de la computació quàntica també s'utilitza, no obstant això, en comptes de bits s'utilitzen qubits. Un qubit és l'anàleg d'un bit, en altres paraules, és la unitat mínima d'informació utilitzada pels ordinadors quàntics. En els qubits podem aplicar propietats quàntiques com la superposició o l'entrellaçament⁴. Si un bit pot estar en l'estat 0 o en l'estat 1, un qubit pot estar en una combinació d'aquests estats, en un estat enmig del 1 i del 0. Un qubit és una combinació lineal dels vectors que representen aquests estats [23], $|0\rangle$ i $|1\rangle$:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

On α i β són nombres complexos i $|\psi\rangle$ és un vector en un espai de Hilbert bidimensional.

⁴Ja he parlat de la primera amb la polarització dels fotons, de l'entrellaçament parlaré més endavant.

Els vectors $|0\rangle$ i $|1\rangle$ són anomenats els vectors de la base computacionals, representats com:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Per tant el vector $|\psi\rangle$ és:

$$|\psi\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Aquest vector és un estat quàntic vàlid per representar un qubit, anomenat *statevector* o vector d'estat. No obstant això, hi ha un important factor a tenir en compte. El vector ha d'estar normalitzat amb la norma ℓ_2 , per tant, els nombres α i β no poden ser nombres qualsevols, necessiten ser els coeficients que formin un vector amb una norma de 1:

$$\| |\psi\rangle \| = 1$$

Llavors:

$$\begin{aligned} \| |\psi\rangle \| &= \sqrt{|\alpha|^2 + |\beta|^2} = 1 \\ \Rightarrow |\alpha|^2 + |\beta|^2 &= 1 \end{aligned}$$

Parlar d'un qubit com una «combinació lineal dels estats fonamentals» no és de molta ajuda, per tant, elaboraré més aquesta qüestió:

Una cadena de n -bits només pot representar una única combinació d'uns i zeros, mentre que una cadena de n -qubits representa una combinació concreta de totes les possibles combinacions. En el cas d'un qubit, aquest és una combinació dels possibles estats $|0\rangle$ i $|1\rangle$. Considera un qubit com una barreja dels estats possibles, amb cada coeficient de la combinació lineal sent el nombre que indica quant d'un estat forma part de la barreja.

Una cosa molt interessant passa quan s'augmenta el nombre qubits, la «quantitat d'informació» creix exponencialment. Per una cadena de n -qubits la

«quantitat d'informació» que té, en altres paraules la quantitat de nombres que representa és 2^n , on aquests nombres són els coeficients que corresponen a les combinacions lineals. Això és perquè quan afegeixes un qubit el nombre de combinacions possibles creix exponencialment, per tant, es necessiten més coeficients per representar aquestes combinacions noves en la combinació lineal. El qubits necessiten molta més informació per poder representar-los⁵, no com els bits són una única combinació. No passa res si això no s'entén perfectament, el que és important és saber que es necessiten 2^n nombres complexos⁶ per representar n -qubits i que es necessiten n números binaris per representar n -bits [24].

Cal remarcar que els vectors de la base computacional serien les columnes d'una matriu identitat amb dimensions $2^n \times 2^n$, on n és el nombre de qubits. Per il·lustrar tot això, 2 qubits es representen amb el *statevector*:

$$\begin{aligned} |\psi\rangle &= \alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_3 |10\rangle + \alpha_4 |11\rangle \\ &= \alpha_1 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \alpha_4 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} \end{aligned}$$

Per poder representar informació amb qubits simplement es codifica aquesta informació en els qubits, això es pot fer per exemple mitjançant codi binari: Els números 0, 1, 2 i 3 són els bits 00, 01, 10 i 11 respectivament, per tant, es poden representar amb dos qubits en els estats $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$, respectivament.

2.2.1 Representació geomètrica d'un qubit

Un aspecte que sempre m'ha agradat del qubits és la seva representació geomètrica, l'esfera de Bloch 2.1 [25]. Si agafem una esfera unitària⁷ que té els seus

⁵Informació que es manifesta amb els coeficients de la combinació lineal

⁶Són complexos, ja que els coeficients de la combinació lineal són nombres complexos.

⁷Una esfera que té radi 1 i que qualsevol punt en la seva superfície correspon a un vector en \mathbb{R}^3 unitari.

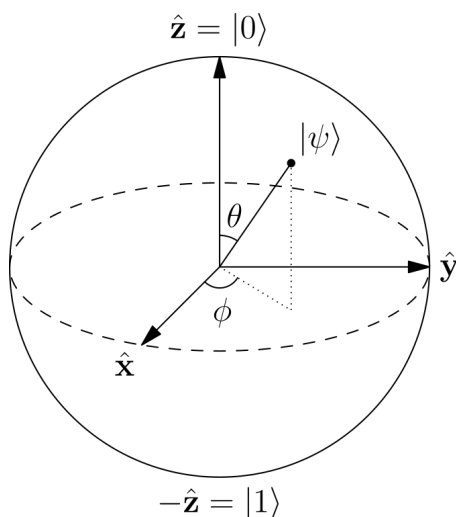


FIGURA 2.1: Esfera de Bloch, on es representa un estat arbitrari $|\psi\rangle$ amb els vectors $\hat{x}, \hat{y}, \hat{z}$ representat els eixos ortonormals de la esfera de radi 1.

pols nord i sur definits pels vectors $|0\rangle$ i $|1\rangle$, respectivament. Cada punt de la seva superfície és un estat quàntic vàlid on les seves bases computacionals són $|0\rangle$ i $|1\rangle$.

2.2.2 Operacions per a només un qubit

Una vegada la informació és representada amb els qubits, estaria bé poder operar amb aquesta informació. Operar amb la informació és justament el que fa que els ordinadors siguin ordinadors, poder operar amb la informació. En computació quàntica els qubits al poder ser representats amb vectors són operats per les anomenades portes lògiques quàntiques, que són matrius. Per exemple, si es vol passar de tindre un qubit en l'estat $|0\rangle$ a l'estat $|1\rangle$, s'utilitza la porta lògica X representada a continuació:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Podem veure com es fa aquesta l'acció al multiplicar la matriu pel vector:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Que en notació de Dirac s'expressaria com:

$$X |0\rangle = |1\rangle$$

D'una manera més general:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 0 \times \alpha + 1 \times \beta \\ 1 \times \alpha + 0 \times \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

Es pot veure que aquesta matriu el que fa és donar la volta als coeficients d'un vector, per tant:

$$X |0\rangle = |1\rangle \text{ i } X |1\rangle = |0\rangle$$

Aquesta porta lògica forma part d'un grup de matrius molt important, les matrius de Pauli, compost per 3 matrius: la X, la Y i la Z, usualment representades per X , Y , Z o per σ_x , σ_y , σ_z . Aquestes matrius són les següents:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Aquestes matrius són molt importants en la mecànica quàntica⁸ i són utilitzades àmpliament per descompondre portes lògiques quàntiques [26].

A partir d'elles podem elaborar matrius que facin una rotació d'un angle θ [26] qualsevol en un dels eixos de la representació geomètrica d'un qubit 2.1:

$$R_x(\theta) = e^{-i\theta X/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} X = \begin{bmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix} \quad (2.2)$$

$$R_y(\theta) = e^{-i\theta Y/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Y = \begin{bmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix} \quad (2.3)$$

$$R_z(\theta) = e^{-i\theta Z/2} = \cos \frac{\theta}{2} I - i \sin \frac{\theta}{2} Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \quad (2.4)$$

⁸Al ser Hermitianes són uns observables, concretament són els observables que corresponen al spin d'una partícula amb spin $\frac{1}{2}$ bàsicament estan relacionades amb els operadors del moment angular.

Per exemple la matriu $R_y(\cdot)$ (Eq.2.3) correspon a una rotació en l'eix \hat{y} de l'esfera de la figura 2.1.

Aquestes operacions poden resultar en superposicions si es fan rotacions amb certs angles. Però hi ha una porta lògica especial per poder fer una rotació que resulta en una superposició uniforme. És a dir una superposició que tingui les mateixes probabilitats de resultar en $|0\rangle$ o $|1\rangle$. Aquesta és la porta de Hadamard, denotada per H :

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (2.5)$$

Podem comprovar que és una superposició uniforme al aplicar-la a l'estat $|0\rangle$:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

L'estat resultant és un estat especial que s'escriu com $|+\rangle$ ⁹. La probabilitat que un estat col·lapsi en una determinada base és el coeficient de la seva base elevat al quadrat. Com que l'estat és:

$$|+\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$$

Al elevar al quadrat qualsevol dels coeficients es pot veure que dona $\frac{1}{2}$:

$$\left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$$

Llavors tenim que la probabilitat per obtenir els dos estats és la mateixa, és a dir que si mesurem l'estat $|+\rangle$ hi ha la mateixa probabilitat que surti $|0\rangle$ o $|1\rangle$. La porta lògica de Hadamard és molt important, ja que s'utilitza per crear distribucions uniformes, sigui en un qubit o en diversos¹⁰.

⁹Un altre estat similar és $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$, quan la matriu H s'aplica a l'estat $|1\rangle$.

¹⁰S'aplica aquesta operació a cada qubit del sistema.

Altres operacions importants de només un qubit són les portes S i T :

$$S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

2.2.3 Circuit quàntics

Aquestes operacions usualment es representen a través de circuits quàntics. Són representacions gràfiques que indiquen quines operacions s'apliquen a quins qubits i en quin ordre.

La forma de representar una porta H aplicada a un qubit és amb el circuit quàntic:

$$|0\rangle \text{ --- } \boxed{H} \text{ ---}$$

El qubit és representat per la línia que comença amb $|0\rangle$, amb $|0\rangle$ sent el seu estat inicial. Aquests diagrames es llegeixen d'esquerra a dreta, la mateixa forma en la qual s'apliquen les operacions.

Un qubit en el qual se li aplica una porta H i després una porta X , és representat com:

$$|0\rangle \text{ --- } \boxed{H} \text{ --- } \boxed{X} \text{ ---}$$

Múltiples qubits són simplement més línies, aquí estan representant dos qubits amb portes X aplicades a cada un:

$$|0\rangle \text{ --- } \boxed{X} \text{ ---}$$

$$|0\rangle \text{ --- } \boxed{X} \text{ ---}$$

Amb els circuits quàntics també s'expressa l'ordre en el qual s'apliquen les portes, de esquerra a dreta. Un circuit de dos qubits, en el qual s'aplica una porta H en l'eix x en el segon qubit, seria:

$$|0\rangle \text{ --- } \boxed{H} \text{ ---}$$

$$|0\rangle \text{ --- } \boxed{R_x(\theta)} \text{ ---}$$

Es pot veure que hi ha un ordre específic per aplicar aquestes portes, primer s'aplica la H i després la rotació.

2.2.4 Operacions per a múltiples qubits

El realment interessant és l'aplicació d'una porta a diversos qubits, perquè d'aquesta manera es pot arribar a tindre qubits entrellaçats. La porta més útil per entrellaçar qubits és la CNOT o *Controlled NOT*:

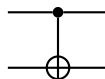
$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.6)$$

Aquesta porta s'aplica a dos qubits, que anomenaré c i t . És bàsicament una porta X que està controlada pel qubit c : si c és $|1\rangle$, s'aplica la porta X al qubit t . El cas contrari, amb el qubit c sent l'estat $|0\rangle$, en el qubit t no s'aplicaria cap porta.

Però si el qubit c està en superposició, aquesta superposició passa també al qubit t , i al mesurar-lo, la probabilitat de què s'hagi aplicat la porta X al qubit t és la mateixa probabilitat de què c estigui en l'estat $|1\rangle$. Els qubits d'alguna manera han passat a compartir la superposició. Es considera que aquests qubits estan entrellaçats, una mesura a un d'ells afecta a la mesura de l'altre.

El qubit al qual se li aplica la porta X es diu *target* (que en l'explicació anterior seria el qubit t) i el qubit sobre el qual depèn el *target* és el *control* (que seria el qubit c).

Aquesta porta representada en un circuit s'escriu com¹¹:



On el qubit *control* és el primer i el segon és el *target*, el qubit que té el símbol \oplus .

¹¹A vegades escriuré els circuits quàntics sense especificar l'estat inicial, ja que no és necessari.

2.2.4.1 Entrellaçament quàntic

L'exemple més senzill d'un entrellaçament quàntic en la computació quàntica són els parells de Bell. Es creen al aplicar a dos qubits una porta H al primer i després una porta CNOT als dos creant l'estat [27]:

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Es pot veure que només hi ha dos estats possibles $|00\rangle$ i $|11\rangle$ que tenen la mateixa probabilitat associada¹². S'afecten l'un a l'altre en el sentit que quan es mesura només un dels qubits i dona per exemple $|1\rangle$, al mesurar l'altre també donaria $|1\rangle$. D'aquesta manera acabant amb l'estat $|11\rangle$. En altres paraules, la mesura d'una part del sistema determina el resultat d'una mesura en una altra part del sistema.

Matemàticament, un sistema quàntic, i.e. un conjunt de qubits, està entrellaçant quan aquest sistema no es pot descriure amb un producte tensorial de les parts. Per exemple estat $|00\rangle$ es pot escriure com $|0\rangle \otimes |0\rangle$, mentre que l'estat $\frac{|00\rangle + |11\rangle}{\sqrt{2}}$, no. Per tant, el primer no és sistema amb qubits entrellaçats i el segon sí ho és.

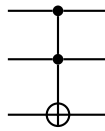
A partir de l'entrellaçament i la superposició és com els ordinadors quàntics arriben a tenir avantatges en complexitat sobre els ordinadors clàssics, per a més informació sobre els avantatges que presenten els algorismes quàntics en certes tasques i el concepte de complexitat algorítmica veure l'annex D.

2.2.4.2 Operacions controlades

A part de la porta CNOT existeixen diverses portes quàntiques controlades. Realment es pot controlar qualsevol porta, en altres paraules, si l'estat del qubit *control* és $|0\rangle$, s'aplica qualsevol porta al qubit *target*. Fins i tot poden haver-hi diversos qubits *control* i *target* [28].

Per exemple existeix la porta Toffoli:

¹²Això es pot veure a l'elevat al quadrat els coeficients del dos, que donen $\frac{1}{2}$.



La seva forma matricial és:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Aquesta porta aplica una porta X a l'últim qubit en cas que els dos primers siguin $|0\rangle$.

Tornant a dos qubits, al veure la matriu per la porta CNOT, es pot apreciar que està composta per una matriu identitat i una porta X ¹³:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

També al veure la porta Z controlada (CZ), es pot apreciar el mateix patró:

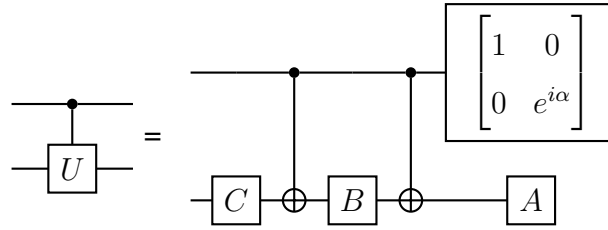
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

¹³La matriu identitat es troba a la cantonada superior esquerra, mentre que la porta X es troba a l'altre extrem.

Amb la matriu de la porta Z a la cantonada inferior dreta. Aquesta porta en un circuit quàntic es representa de la següent manera:



Una operació controlada de qualsevol matriu unitària U es pot formar a través del següent circuit [28]:



On U, α, A, B i C son tals que $U = e^{i\alpha} A X B X C$ i $ABC = I$.

2.3 Mesurament quàntic

Com ja s'ha esmentat a la secció 2.2.2 a l'elevat al quadrat el coeficient d'un estat base que forma part d'un estat, s'obté la probabilitat d'obtenir aquest estat base quan es mesura.

Aquesta és la forma més simple de poder predir el mesurament d'un estat quàntic. Però hi ha altres mètodes que a vegades resulten més útils.

Els mesuraments quàntics també es poden pensar com un conjunt d'operadors de mesura M_m , on la probabilitat d'obtenir l'estat m associat a un operador M_m al mesurar un $|\psi\rangle$ és [29]:

$$\text{prob}(m) = \langle \psi | M_m^\dagger M_m | \psi \rangle \quad (2.7)$$

On l'estat després de la mesura és:

$$\frac{M_m |\psi\rangle}{\sqrt{\langle \psi | M_m^\dagger M_m | \psi \rangle}}$$

Els operadors M_m han de complir que la suma de les seves probabilitats sigui 1:

$$1 = \sum_m \text{prob}(m) = \sum_m \langle \psi | M_m^\dagger M_m | \psi \rangle$$

La diferència entre aquesta manera de fer les mesures i elevar els coeficients al quadrat, és que l'equació 2.7 és una forma més general, on en comptes de mesurar en la base computacional, es pot mesurar en qualsevol base.

Per mesurar en la base computacional d'un *statevector* s'utilitzen operadors de mesura derivats de la base computacional fets a partir de productes exteriors. Per crear l'operador de mesura M_i associat a la base computacional $|i\rangle$ s'agafa el producte exterior de la base:

$$M_i = |i\rangle \langle i|$$

Per tant la probabilitat que la mesura de l'estat $|\psi\rangle$ resulti en $|0\rangle$, és:

$$\begin{aligned} \text{prob}(|0\rangle) &= \langle \psi | M_{|0\rangle}^\dagger M_{|0\rangle} | \psi \rangle \\ &= \langle \psi | (|0\rangle \langle 0|)^\dagger |0\rangle \langle 0| \psi \rangle \end{aligned}$$

Per $|\psi\rangle = |0\rangle$ tenim que¹⁴:

$$\begin{aligned} \text{prob}(|0\rangle) &= \langle 0|0\rangle \langle 0|^\dagger |0\rangle \langle 0|0\rangle \\ &= \langle 0|0\rangle \langle 0|0\rangle \langle 0|0\rangle \\ &= \langle 0|0\rangle \langle 0|0\rangle \\ &= 1 \end{aligned}$$

El resultat té sentit, ja que si mesurem $|0\rangle$ en la base $|0\rangle$, la probabilitat de trobar l'estat hauria de ser 1.

¹⁴Cal notar que $(|0\rangle \langle 0|)^\dagger = |0\rangle \langle 0|$ i que $|0\rangle$ és un vector unitari.

2.4 Matriu de densitat

Una sèrie de qubits es pot representar tant per un vector com per una matriu, anomenats vector d'estat i matriu de densitat, respectivament [7]. En aquesta secció explicaré ràpidament sobre el concepte de matriu de densitat i com les operacions que s'apliquen a un vector d'estat poden ser aplicades a aquesta matriu. Després parlaré dels mesuraments parcials d'un sistema, un concepte que és important per la part experimental del treball.

Una matriu de densitat és la representació matemàtica d'un estat quàntic a partir d'una matriu. Aquest concepte en anglès s'anomena *density operator/matrix*. Aquesta representació serveix per descriure sistemes quàntics que no són completament coneguts. Concretament, aquestes matrius són conjunts d'estats quàntics: Per un sistema que es descriu com un conjunt d'estats $\{|\psi_i\rangle\}$ amb cada element del conjunt té una probabilitat associada de p_i . La matriu densitat ρ del d'aquest sistema és [30]:

$$\rho = \sum_i p_i |\psi_i\rangle \langle \psi_i|$$

Aquesta matriu pot ser simplement $|\psi\rangle \langle \psi|$ per un estat qualsevol $|\psi\rangle$, per exemple la matriu que representa $|0\rangle$ és:

$$\rho = |0\rangle \langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

L'evolució d'un operador de densitat que descriu un sistema quàntic, igual que amb els vectors, s'efectua a partir de matrius unitàries. Un estat ρ evoluciona a partir d'una matriu unitària U de la següent manera:

$$\sum_i p_i U |\psi_i\rangle \langle \psi_i| U^\dagger = U \rho U^\dagger$$

Igual que es fan mesures en els *statevectors*, les podem fer en els operadors de densitat. Per un operador de mesura M_m , tenim que la probabilitat de

tindre l'estat m al mesurar un vector $|\psi\rangle$ o una matriu de densitat ρ , és:

$$\begin{aligned}\text{prob}(m) &= \langle\psi| M_m^\dagger M_m |\psi\rangle = \text{Tr}(M_m^\dagger M_m |\psi\rangle \langle\psi|) \\ &= \text{Tr}(M_m^\dagger M_m \rho)\end{aligned}$$

També tenim que l'estat $|\psi\rangle$ després de la mesura m és [30]:

$$|\psi_m\rangle = \frac{M_m |\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}} = \frac{M_m |\psi\rangle}{\sqrt{\text{Tr}(M_m^\dagger M_m \rho)}} \quad (2.8)$$

L'equació 2.8 es pot reescriure en termes d'una matriu de densitat després d'una mesura m :

$$\begin{aligned}\rho_m &= \sum_i p_i \frac{M_m |\psi\rangle \langle\psi| M_m^\dagger}{\text{Tr}(M_m^\dagger M_m \rho)} \\ &= \frac{M_m \rho M_m^\dagger}{\text{Tr}(M_m^\dagger M_m \rho)}\end{aligned}$$

Perquè això sigui cert els operadors de mesura M_m , han de satisfer:

$$\sum_m M_m^\dagger M_m = I$$

Per a tots els estats possibles m .

Per últim s'ha de recordar que les matrius densitat igual que els vectors d'estat han de tenir certes característiques:

1. La traça de ρ ha d'equivaldre a 1.
2. ρ ha de ser un operador positiu¹⁵.

2.4.1 Matriu de densitat reduïda

Una aplicació important dels operadors de densitat és la descripció d'estats parcials amb l'operador de densitat reduït, i per tant, la descripció dels mesuraments parcials.

¹⁵Un operador positiu A es aquell que $\langle\psi| A |\psi\rangle \geq 0, \forall |\psi\rangle$.

Per un sistema físic compost per dos sistemes A i B que es descriu per una matriu de densitat ρ^{AB} , la matriu de densitat reduïda del sistema A és:

$$\rho^A = \text{Tr}_B(\rho^{AB}) \quad (2.9)$$

On Tr_B és la traça parcial sobre el sistema B , que es defineix per l'equació següent [31]:

$$\text{Tr}_B(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|) = |a_1\rangle\langle a_2| \text{Tr}(|b_1\rangle\langle b_2|)$$

Amb $|a_1\rangle$ i $\langle a_2|$ sent estats vàlids pel sistema A , i $|b_1\rangle$ i $\langle b_2|$ sent-ho per B . El terme $\text{Tr}(|b_1\rangle\langle b_2|)$ s'omet quan els vectors del producte exterior són iguals i formen un operador de densitat vàlid, el qual té una traça de 1.

No obstant aquesta definició no es pot utilitzar quan no saps com representar la matriu de densitat ρ com a un producte tensorial, en el qual un dels termes és l'estat que es traça a fora. En altres paraules, en l'equació 2.9 si no es coneix ρ^A i ρ^B per $\rho^{AB} = \rho^A \otimes \rho^B$, aquesta equació no serveix de res.

A causa d'això en el llibre de text *Quantum Computing: A Gentle Introduction* [8] els autors defineixen la traça parcial d'una altra manera més general, on només s'ha de saber les bases dels sistemes A i B i un operador vàlid pel sistema AB . Per una matriu de densitat ρ^{AB} que representa un estat del sistema $A \otimes B$, la traça parcial de ρ^{AB} sobre B és:

$$\text{Tr}_B \rho^{AB} = \sum_i \langle \beta_i | \rho^{AB} | \beta_i \rangle$$

On el conjunt $\{|\beta_i\rangle\}$ són les bases del sistema B . Les entrades de la matriu $\text{Tr}_B \rho^{AB}$ representades en termes de les bases $|\alpha_i\rangle$ i $|\beta_j\rangle$ dels sistemes A i B respectivament, són:

$$(\text{Tr} \rho^{AB})_{ij} = \sum_{k=0}^{M-1} \langle \alpha_i | \langle \beta_k | \rho^{AB} | \alpha_j \rangle | \beta_k \rangle$$

Amb la matriu sent:

$$\text{Tr } \rho^{AB} = \sum_{i,j=0}^{N-1} \left(\sum_{k=0}^{M-1} \langle a_i | \langle \beta_k | \rho^{AB} | \alpha_j \rangle | \beta_k \rangle \right) | \alpha_i \rangle \langle \alpha_j |$$

On N és la dimensió del sistema A i M és la dimensió del sistema B .

Es pot comprovar que aquestes definicions són correctes ja que en els dos llibres utilitzen les seves definicions per tractar el mateix cas i obtenen el mateix resultat [32, 33].

2.4.1.1 Mesurament parcial

Es pot arribar a treure una mesura parcial¹⁶ sobre un sistema de qubits amb la traça parcial i operadors de mesura. En el paper fet per Huang et.al. [1] es descriu un estat ρ després d'un mesurament parcial Π_A sobre un sistema A que pertany a l'estat $|\psi\rangle$ com:

$$\rho = \frac{\text{Tr}_A(\Pi_A |\psi\rangle \langle \psi|)}{\text{Tr}(\Pi_A \otimes I_{2^N - 2^{N_A}} |\psi\rangle \langle \psi|)}$$

Essent Π_A un mesurament parcial definit com¹⁷ a $\Pi_A = (|0\rangle \langle 0|)^{\otimes N_A}$, on N_A és el nombre de qubits sobre els quals es treu el mesurament parcial.

El sistema A al estar composts N_A qubits, per tant, la resta de l'estat $|\psi\rangle$ té $N - N_A$, on N és el nombre total de qubits del estat $|\psi\rangle$. D'aquesta manera $\Pi_A \otimes I_{2^N - 2^{N_A}}$ té $2^N \times 2^N$ dimensions i pot ser multiplicat per la matriu $|\psi\rangle \langle \psi|$ que té les mateixes dimensions. No obstant això, sorgeix un problema amb el numerador de l'equació perquè Π_A no té les mateixes dimensions¹⁸ que $|\psi\rangle \langle \psi|$, encara no he pogut utilitzar aquesta equació adequadament. No sé com computar-la. Aquest problema l'adreçaré en la part experimental del treball.

¹⁶És a dir, una mesura a només una part dels qubits que conformen un sistema.

¹⁷No estic completament segur d'això ja que els autors no defineixen aquest operador per aquesta equació en concret, però en un altre equació si que ho fan.

¹⁸En aquesta afirmació no tinc cap dubte, ja que no és necessita la definició de Π_A per poder afirmar-ho.

En el mateix article es planteja una altra equació, per l'estat post-mesura expressat en forma de vector d'estat, molt semblant a l'equació 2.8. L'única diferència és que en l'equació de l'article no s'expressa l'operador de mesura en la forma $M_m^\dagger M_m$, en canvi, els autors ho expressen tan sols com $I_{2^N-2^{N_A}} \otimes \Pi_A$. Potser la forma plantejada pels autors té en compte el conjugat hermitià, però no estic segur. L'equació esmentada en l'article és la següent:

$$|\psi_m\rangle = \frac{I_{2^N-2^{N_A}} \otimes \Pi_A |\psi\rangle}{\sqrt{\text{Tr}(I_{2^N-2^{N_A}} \otimes \Pi_A |\psi\rangle \langle\psi|)}}$$

Al igual que amb l'altra equació, no sé com computar-la¹⁹.

2.5 Ordinadors quàntics

Tota aquesta teoria es veu plasmada a la realitat als qubits físics dels ordinadors quàntics. Hi ha diversos tipus d'ordinadors quàntics, a causa del fet que els qubits poden ser diversos sistemes. Poden ser fotons, xips de silici superconductors o ions atrapats per imants. No elaboraré més sobre aquest tema, ja que no és el tema central d'aquest treball, m'he centrat molt més en la teoria.

Però si vull generar imatges amb un ordinador quàntic, no en necessito un? No necessàriament, perquè puc simular l'evolució dels estats quàntics amb un ordinador, al cap i a la fi, és només àlgebra lineal, són operacions que es poden fer perfectament en un ordinador. No obstant això, quan s'intenta simular un sistema quàntic de molt qubits,²⁰ un ordinador de sobretaula tardaria molt de temps i realment no és viable fer-ho.

¹⁹Un cop entregat el treball me pogut veure que si que la puc computar, no obstant això, no és pertinent fer canvis en la part experimental o les conclusions del treball amb aquesta nova informació.

²⁰Més de 50 per exemple.

Capítol 3

Intel·ligència artificial

Segurament has sentit parlar de les intel·ligències artificials o de les xarxes neuronals, són conceptes que semblen abstractes, però jo penso que són bastant intuïtius i en aquest capítol els intentaré explicar de la millor manera possible.

Intel·ligència artificial és un mot una mica ambigu, que es refereix a qualsevol algoritme que entra dintre del camp del *machine learning* o aprenentatge automàtic¹. Aquests algorismes simplement s'alteren a ells mateixos per fer millor la tasca que s'els ha designat. No importa quin és l'objectiu o com s'ha aconseguit, el que importa és si aprèn automàticament. Cal notar que els canvis que s'efectuen els algorismes sobre si mateixos no han de ser predeterminats, si l'algoritme té una llista de les instruccions que va executant segon la situació no seria una intel·ligència artificial o un algoritme d'aprenentatge automàtic. [34].

La manera que tenen aquests algorismes d'alterar-se a si mateixos és canviant els paràmetres de les operacions dels quals estan compostos. Això ho fan a partir de veure patrons en un conjunt de dades. Per exemple, en una regressió lineal, s'actualitzen els paràmetres de la recta que representa la tendència de les dades, com es pot veure a la figura 3.1.

Per ajustar els paràmetres, el més comú és ajustar-los segons la derivada d'una funció anomenada funció de pèrdua o *loss function*, usualment representada per la lletra \mathcal{L} . Aquesta funció representa els objectius del programa i pot ser

¹Tanmateix, col·loquialment s'utilitza per denominar a qualsevol algoritme o robot que és intel·ligent o sembla que és intel·ligent.

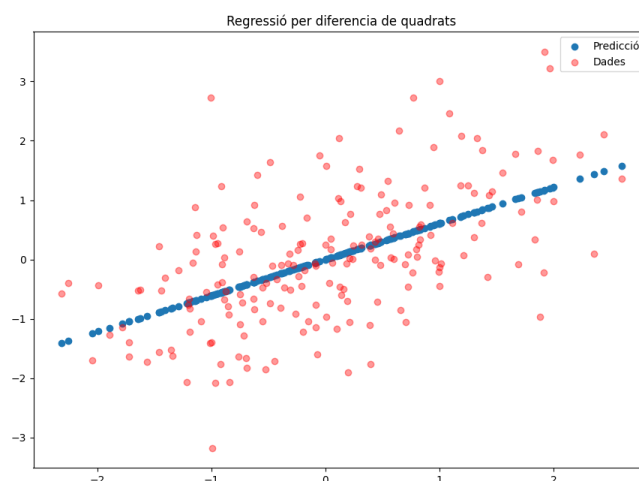


FIGURA 3.1: Exemple d'una regressió lineal de dades generades a l'atzar. Veure el codi a [E.1.1.0.1](#).

minimitzada o maximitzada. Per exemple, en una regressió lineal es vol reduir la distància entre els punts de dades i la línia que prediu la tendència.

A causa de que de la gran varietat de funcions de pèrdua que existeixen, els programes de *machine learning* són extremadament versàtils. La màxima expressió d'això es pot veure en les xarxes neuronals o *neural networks*, on les diverses arquitectures de xarxes neuronals que existeixen també contribueixen a la seva versatilitat [34].

Aquests algorismes són els més potents, complexos i polivalents, dintre de l'aprenentatge automàtic. Precisament utilitzo un d'aquests algorismes d'aquests per generar les imatges. Es fan servir, entre altres coses, pel reconeixement d'imatges, la traducció i sintetització de textos, la conducció automàtica, els algorismes de recomanació, i és clar, la generació d'imatges.

3.1 Xarxes neuronals

Aquests tipus d'algorismes tenen un nom que fa recordar a les xarxes de neurones dels nostres cervells; no és casualitat, ja que estan directament inspirades en els nostres cervells. Són uns programes que consisteixen en la connexió de

diverses operacions anomenades neurones, que conjuntament formen una xarxa, la qual s'organitza a partir de capes. Segons la variació del tipus de neurona i l'estructura que aquestes formen, podem tindre algorismes destinats a fer diferents tasques. Això juntament amb els diversos tipus de funció de pèrdua contribueix a la versatilitat de les xarxes neuronals. Aquests models d'intel·ligència artificial constitueixen el camp del *deep learning* o aprenentatge profund. S'anomenen d'aquesta forma per referenciar la gran profunditat d'aquests algorismes, és a dir el gran nombre de capes que tenen [35].

Una neurona consisteix simplement en una suma ponderada, a la qual se li suma un altre número, i finalment una funció no lineal que s'aplica al resultat. Les neurones tenen vectors com input i output. Per tant, una neurona es pot definir com:

$$\sigma \left(\sum_{i=1}^n w_i x_i + b \right) = \sigma (w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b)$$

Essent σ sent una funció no lineal i n la mida del vector. Després estan els paràmetres, w_i i b , anomenats *weights* i *bias*.

Una neurona pot tindre diversos inputs que venen de diverses neurones; el mateix passa amb els outputs. Depenent de com es connectin entre si les neurones, aquestes passen a formar diversos tipus de capes. A partir dels tipus de capes i el nombre d'aquestes és com s'especifica l'arquitectura d'una xarxa neuronal.

Una vegada especificada la interconnectivitat de les neurones puc parlar de la intuïció darrere dels paràmetres. Un *weight* especifica com és de forta la relació entre una neurona en una capa i una altra neurona en una capa veïna. I un *bias* especifica com és d'important una neurona, ja que aquest número afecta el resultat a la suma, fent que l'activació de la neurona² sigui més alta.

Tornant a l'arquitectura, usualment aquesta es divideix en tres parts la capa d'input, les capes ocultes i la capa d'output. La quantitat de neurones que hi ha a la capa d'inputs és la que defineix la mida del vector que es dona com input a la xarxa, ja que cada element del vector es dona a cada neurona amb la capa.

²Així és com s'anomena el seu resultat.

El mateix passa amb la capa d'outputs, cada output de cada neurona de la capa acaba sent un element en el vector que surt de la xarxa. Per tant, el nombre de neurones que té cadascuna d'aquestes dues capes, especifica la mida dels vectors d'input i d'output de la xarxa respectivament. Per exemple, si es vol donar com input a una xarxa una imatge de 16 per 16 píxels en blanc i negre calen 256 neurones en la capa d'inputs, una per cada píxel.

En canvi, les *hidden layers*, és a dir les capes ocultes, no tenen una mida determinada, el mateix passa amb el nombre d'aquestes que té la xarxa. Depenen de cada cas la quantitat de neurones que tenen aquestes capes i també el nombre d'aquestes, varia. Això, juntament amb els diversos tipus de capes és el que dona la versatilitat d'aquests algorismes [35].

Entre els diferents tipus de capes que poden tindre les xarxes neuronals, la més comuna i simple d'aquestes és la *fully connected layer*, o una capa completament connectada, com les que es poden veure en la figura 3.2. Les neurones que formen aquesta capa estan connectades a totes les neurones de la capa anterior i així mateix, a totes les neurones de la capa següent. L'única forma que aquestes capes tenen de variar és a partir de canviar el nombre de neurones, ja que no es pot alterar el funcionament de les neurones.

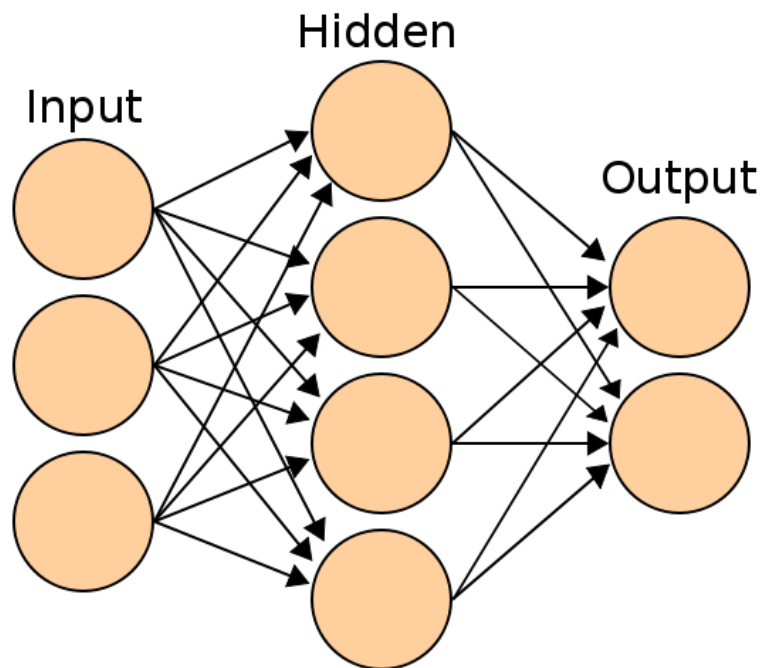


FIGURA 3.2: Usualment, les xarxes neuronals es representen d'aquesta forma, amb fletxes i boles. Les boles representarien cada neurona i les fletxes mostren com estan connectades. La *hidden layer* d'aquesta representació es pot veure que és una *fully connected layer* a causa de com està connectada a les altres capes, rebent cada neurona l'output de cada neurona anterior.

No obstant això, es pot variar la funció d'activació que tenen, que ha de ser una funció no lineal. La més utilitzada és la sigmoide:

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

3.2 Descens del gradient

Una vegada he parlat de les xarxes neuronals, he de comentar com aquestes evolucionen al llarg del temps, és a dir com se'n van actualitzen a si mateixes per complir el seu objectiu. Ja he comentat que existeix una funció anomenada *loss function*, la qual es deriva per actualitzar els paràmetres de la xarxa. El mecanisme pel qual s'actualitzen els paràmetres s'anomena el descens del gradient.

Aquest procés comença amb la funció de pèrdua, que esmenta els objectius de la xarxa. Els punts mínims d'aquesta funció representen els punts òptims de la xarxa, als quals es vol arribar.

Per explicar-lo utilitzaré un exemple pràctic, primer de tot parlaré sobre la funció de pèrdua que s'utilitza i a continuació de l'actualització dels paràmetres.

Parlaré de la classificació d'imatges, si es volen classificar imatges de gats i gossos, s'assignen dues etiquetes a aquestes, per exemple, 1 als gats i 0 als gossos. A continuació s'escull una funció de pèrdua³ com *binary cross entropy* o *log loss*. La funció *binary cross entropy* és la següent [36]:

$$\mathcal{L} = -t \log(y) - (1 - t) \log(1 - y) \quad (3.1)$$

Amb t sent l'etiqueta que ha de tenir la imatge, anomenada etiqueta real, i y sent l'etiqueta que el model assigna a la imatge. Per tant, si l'etiqueta real és 1, l'equació acaba sent [36]:

$$\mathcal{L}_1 = -\log(y)$$

I el cas contrari per $t = 0$ seria:

$$\mathcal{L}_0 = -\log(1 - y)$$

Si donen com input la imatge d'un gat i el programa es dona com output 0.92 tenim que la pèrdua és de:

$$\mathcal{L} = -t \log(y) - (1 - t) \log(1 - y) = -\log(0.92) \simeq 0.0834$$

En canvi, si el programa es dona un output de 0.15 la pèrdua, seria:

$$\mathcal{L} = -t \log(y) - (1 - t) \log(1 - y) = -\log(0.15) \simeq 1.897$$

Es pot veure que si l'etiqueta que posa el model s'allunya més de l'etiqueta real la pèrdua acaba sent més gran. El mateix es pot veure per les imatges dels gossos,

³Una que funcioni per la classificació d'imatges, és clar.

és a dir, les imatges que haurien de tenir etiqueta zero:

$$\mathcal{L} = -0 \cdot \log(0.89) - (1 - 0) \cdot \log(1 - 0.89) = -\log(1 - 0.89) \simeq 2.207$$

$$\mathcal{L} = -0 \cdot \log(0.08) - (1 - 0) \cdot \log(1 - 0.08) = -\log(1 - 0.08) \simeq 0.0834$$

Per tant, com més pròximes estiguin les prediccions (els outputs del model) a les etiquetes ideals, menor serà la pèrdua. Llavors al minimitzar la funció es trobarà el punt òptim on totes les prediccions seran iguals a les etiquetes [37].

A aquests punts òptims s'arriben actualitzant els paràmetres a partir de la derivada de la funció, concretament a través del seu gradient. El gradient d'una funció és un vector on els seus elements són la derivada parcial respecte a cada paràmetre (per aclarir, una entrada per paràmetre). El gradient d'una funció $f(\theta)$ es representa com $\nabla f(\theta)$, i es pot escriure en forma de vector com:

$$\nabla f(\theta) = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \\ \vdots \\ \frac{\partial f}{\partial \theta_{n-1}} \\ \frac{\partial f}{\partial \theta_n} \end{bmatrix}$$

No fa falta fixar-se en el que és exactament un gradient, només s'ha de saber que cada paràmetre de la xarxa neuronal⁴ s'ha d'actualitzar d'acord amb la derivada respecte al paràmetre. Això es pot entendre com canviar lleugerament un paràmetre d'acord amb la direcció de la derivada respecte a ell. L'objecte que s'encarrega d'actualitzar-los s'anomena optimitzador. L'optimitzador més senzill que hi ha és el següent [38]:

$$\theta_i^t = \theta_i^{t-1} \pm \eta \frac{\partial \mathcal{L}(\theta)}{\partial \theta_i^{t-1}} \quad (3.2)$$

Es pot veure com s'actualitza el paràmetre θ_i . En l'equació he posat el superíndex t per expressar aquesta actualització, passant d'un temps $t-1$ a t . També, utilitzo

⁴Cada *weight* i cada *bias*.

el símbol θ per expressar tots els paràmetres del model, que es troben en la forma d'un vector. En els optimitzadors s'afegeix un nombre η anomenat *learning rate*. Usualment, és un nombre petit⁵, aquest nombre especifica com de ràpid canvien els paràmetres. El *learning rate* es pot anar ajustant depenen de la situació o del model en el qual s'implementi. Alteracions en aquest poden causar diversos fenòmens tant negatius com positius, i aquesta és la principal tasca que tenen els optimitzadors, alterar el *learning rate*. Existeixen molts optimitzadors que donen a terme aquesta tasca de diverses maneres, un dels més rellevants és ADAM [39].

Cal notar que en l'optimitzador he utilitzat el signe \pm perquè si és positiu significa que s'està ascendint pel gradient, i si és negatiu s'està descendint. No obstant això, usualment es fa servir el signe negatiu per convenció, ja que la majoria de funcions de pèrdua estan dissenyades per ser minimitzades.

Al fer la derivada de la funció de pèrdua es pot veure immediatament que s'ha d'aplicar la regla de la cadena, ja que l'estructura de la xarxa neuronal està composta per una sèrie de funcions compostes entre si. En la secció següent parlaré del procediment que s'utilitza per avaluar el gradient, anomenat *backpropagation* o propagació inversa.

3.2.1 Backpropagation

A l'aplicar la regla de la cadena «de fora cap a dins» s'ha de començar a derivar per l'última capa i acabar per la primera, d'aquí ve el nom *backpropagation* perquè es propaga l'error en direcció contrària. Mentre que quan es dona un input a la xarxa, s'anomena forward propagation o *forward pass*.

No entraré en profunditat sobre l'intuïció de la *backpropagation* en aquesta secció, només em limitaré a esmentar la manera en la qual es calcula.

L'activació⁶ d'una neurona j en l'última capa L de la xarxa es defineix com:

$$a_j^L = \sigma \left(\sum_{k=0}^{n_{L-1}} w_{jk}^L a_k^{L-1} + b_j^L \right)$$

⁵Entre 0.1 i 0.001 per exemple.

⁶S'anomena així al resultat d'una neurona.

On a_k^{L-1} és l'activació d'una neurona k en la capa anterior $L - 1$, i on el *weight* w_{jk}^L és el paràmetre que expressa en què mesura es connecta la neurona j a la neurona k . Com ja he dit, expressa com és de forta aquesta connexió. La suma es fa al llarg de n_{L-1} que és el nombre de neurones que té la capa $L - 1$. Utilitzant aquesta definició ja es poden efectuar les derivades.

No obstant això, convé definir un nou terme, escrit com z_j^L , per procedir a fer les derivades. Simplement, és l'equació d'una neurona però sense la funció d'activació:

$$z_j^L = \sum_{k=0}^{n_{L-1}} w_{jk}^L a_k^{L-1} + b_j^L \quad (3.3)$$

L'objectiu és obtenir la derivada de la *loss function* respecte a un *weight* qualsevol i un *bias* qualsevol. Per tant, s'han de obtenir les següents derivades parcials:

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^L} \text{ i } \frac{\partial \mathcal{L}}{\partial b_j^L}$$

Començaré amb la derivada del *weight*, al aplicar la regla de cadena obtenim que:

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^L} = \frac{\partial z_k^L}{\partial w_{jk}^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial \mathcal{L}}{\partial a_j^L} \quad (3.4)$$

La primera derivada, és la derivada de z_j^L respecte al *weight*, que és l'activació d'una neurona en la capa $L - 1$:

$$\frac{\partial z_j^L}{\partial w_{jk}^L} = a_k^{L-1}$$

La següent és la derivada del resultat de la neurona a_j^L respecte a z_j^L , que resulta ser la derivada de la funció d'activació de la neurona:

$$\frac{\partial a_j^L}{\partial z_j^L} = \sigma'(z_j^L)$$

Finalment, està la derivada de la funció de pèrdua respecte a l'output de la xarxa, és a dir, l'activació d'una de les últimes neurones. La qual és simplement la derivada de la funció de pèrdua, per tant, varia en cada cas.

Ja sabent totes les derivades, es pot reescriure l'equació 3.4 es pot escriure com⁷:

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^L} = \frac{\partial z_k^L}{\partial w_{jk}^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial \mathcal{L}}{\partial a_j^L} = a_k^{L-1} \sigma'(z_j^L) \frac{\partial \mathcal{L}}{\partial a_j^L}$$

No obstant això, falta un detall: per efectuar la derivada de l'activació d'una neurona respecte a la funció de pèrdua s'ha d'afegir un sumatori:

$$\frac{\partial \mathcal{L}}{\partial a_j^{L-1}} = \sum_{k=0}^{n_{L-1}} \frac{\partial z_k^L}{\partial a_j^{L-1}} \frac{\partial a_k^L}{\partial z_k^L} \frac{\partial \mathcal{L}}{\partial a_k^L}$$

La suma s'efectua en totes les neurones de la capa.

Finalment, he d'esmentar la derivada de la funció de pèrdua respecte a un *bias* qualsevol b_k^L . A l'aplicar la regla de la cadena es pot veure que aquesta derivada és:

$$\frac{\partial \mathcal{L}}{\partial b_k^L} = \frac{\partial z_k^L}{\partial b_k^L} \frac{\partial a_k^L}{\partial z_k^L} \frac{\partial \mathcal{L}}{\partial a_k^L}$$

Al veure l'equació 3.3 es pot veure que la derivada $\frac{\partial z_k^L}{\partial b_k^L}$ serà 1. Les altres derivades de l'equació ja les he esmentat anteriorment. Per tant, finalment tenim que [38]:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{jk}^L} &= a_k^{L-1} \sigma'(z_j^L) \frac{\partial \mathcal{L}}{\partial a_j^L} \\ \frac{\partial \mathcal{L}}{\partial b_k^L} &= \sigma'(z_k^L) \frac{\partial \mathcal{L}}{\partial a_k^L} \end{aligned}$$

Amb aquestes derivades ja es pot desenvolupar el vector gradient, que com ja he dit està compost per les derivades de cada paràmetre de la xarxa, concretament està compost la mitjana d'aquestes, perquè es vol actualitzar els paràmetres per poder minimitzar la pèrdua respecte a diverses dades, d'aquesta manera el model s'entrena amb el nombre més gran de dades possibles.

Tota aquesta teoria es veurà implementada en la part pràctica en forma de codi, ja que m'he vist amb la necessitat de tenir una xarxa neuronal programada

⁷Deixo l'última derivada $\frac{\partial \mathcal{L}}{\partial a_j^L}$ sense reescriure perquè aquest terme pot variar depenent de la funció de pèrdua que s'utilitza.

des de zero. No obstant això, usualment s'utilitzen plataformes com *TensorFlow* [40] o *PyTorch* [41] per desenvolupar xarxes neuronals, pel fet que aquests *frameworks* faciliten moltíssim el treball.

3.3 Generative adversarial networks

Com ja he dit hi ha molts tipus de xarxes neuronals, tanmateix, en aquest treball només em centraré en un tipus en específic, les xarxes generatives adversàries o *generative adversarial networks (GAN)* en anglès.

Aquestes xarxes, com el seu nom diu, s'utilitzen per generar dades, usualment imatges. Es troben al darrere de projectes com *This person does not exist* [42, 43], una pàgina web que et genera una cara d'una persona que no existeix, ja que és una cara generada artificialment a partir d'aquest tipus de models.

Van ser introduïts per primera vegada en 2014 per Ian Goodfellow [2], des de llavors s'han convertit en un dels models de *deep learning* més sòlids i utilitzats.

Aquests algorismes consisteixen en dos models (xarxes) diferents, un generador i un discriminador, amb objectius oposats. Per aquesta raó s'inclou paraula adversària en el nom. El generador i discriminador es poden entendre com un falsificador de bitllets i uns policies que el vol atrapar. On el generador és el falsificador de bitllets i el discriminador és el policia.

L'analogia és la següent: Una vegada el falsificador comença el seu entramat, el policia comença a detectar quins són els bitllets falsificats, i amb el temps es torna millor al seu treball, podent distingir entre els bitllets falsificats i els reals a la perfecció. El falsificador respon a això millorant les seves tècniques, per tant, el policia han de millorar encara més. És un cicle en el qual aquestes forces antagonistes es fan millorar l'una a l'altra.

El mateix passa amb el generador i el discriminador. El discriminador aprèn a distingir entre les imatges reals i les imatges falses que fabrica el generador, mentre que el generador aprèn a enganyar al discriminador.

Si s'especifiquen bé els objectius de cada model, arriben a un *zero sum game*⁸. La manera en la qual es soluciona aquest tipus de joc és arribant a un equilibri de Nash [1, 2], on el discriminador no sap diferenciar entre les imatges reals i les falses⁹.

En l'article original [2], s'esmenta un pseudocodi per aquests models, que he representat en l'algoritme 1 (adaptat). En aquest es parla d'un *minibatch* que és simplement un grup d'imatges o de dades. I del soroll, que són dades generades aleatòriament i que es donen com input al generador perquè aquest no generi exactament les mateixes imatges cada vegada. El soroll afegeix variació als outputs del generador, però sempre s'intenta que sigui en una petita quantitat.

Algorithm 1 Pseudocodi per una xarxa generativa adversària

for número de interaccions **do**
 for k pasos **do**
 Crear minibatch de m mostres de soroll $\{z_i, \dots, z_m\}$ de la distribució de soroll $p_g(z)$
 Crear minibatch de m mostres d'exemples $\{x_i, \dots, x_m\}$ de la distribució d'exemples $p_{\text{data}}(x)$
 Actualitzar el discriminador ascendint el seu gradient:

$$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m [\log D(x_i) + \log(1 - D(G(z_i)))]$$

end for
 Crear minibatch de m mostres de soroll $\{z_i, \dots, z_m\}$ de la distribució de soroll $p_g(z)$
 Actualitzar el generador descendent el seu gradient:

$$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z_i)))$$

end for

En el pseudocodi es pot veure que primer s'actualitza el discriminador k vegades i després el generador una sola vegada. Això és perquè interessa que el

⁸Un *zero sum game*, és simplement un joc entre dos jugadors que per guanyar un, l'altre ha de perdre e.g. joc d'estirar la corda entre dos equips.

⁹Que el discriminador no sàpiga diferenciar implica arribar a un equilibri de Nash, aquest concepte és definit d'una altra forma que està fora de l'àmbit d'aquest treball [2].

discriminador sàpiga distingir les imatges ràpidament, per poder indicar al generador com generar les imatges. Si el discriminador esmenta que unes imatges de gats són imatges de gossos, el generador fabricarà imatges de gossos pensant que ho són de gats, perquè aquest aprèn a partir del que l'indica el discriminador.

També és pot veure com és la funció de pèrdua, que en l'article els autors anomenen *Minimax loss function*. La qual en la pràctica és la mateixa que la *Binary Cross Entropy (BCE)*. No entraré en detall en aquesta qüestió però resumidament, la BCE és la generalització de la *Minimax*. Com ja he mencionat anteriorment la BCE s'emptra en la classificació d'imatges, que realment és el mateix problema que la generació d'imatges.

Capítol 4

Generació d'imatges amb un ordinador quàntic

Investigadors en informació quàntica al veure el potencial que tenen els ordinadors quàntics i la intel·ligència artificial, no es van poder resistir a crear un nou camp d'investigació, el *Quantum Machine Learning (QML)*, o aprenentatge automàtic quàntic. Igual que les xarxes neuronals són les estrelles dintre del *machine learning*, les xarxes neuronals quàntiques també ho són dintre del *quantum machine learning*.

Des de que es van començar a investigar aquests algoritmes s'han arribat a implementar diversos tipus de xarxes neuronals en ordinadors quàntics. Principalment pel que fa a la generació i classificació d'imatges i dades.

No obstant això, aquests algoritmes no són completament quàntics, usualment consisteixen a actualitzar els paràmetres d'un circuit quàntic perquè aquest generi les dades. On les actualitzacions dels paràmetres es calculen amb un ordinador clàssic. Això es veurà molt clar quan expliqui la part pràctica del treball.

Abans de continuar amb la secció he de dir que existeixen diversos algoritmes dintre del *quantum machine learning* (no tot en la vida són xarxes neuronals). Per exemple, es pot donar a terme classificació de dades mitjançant *support vector machines*¹ [44, 45] o amb un anàleg de la regressió lineal [46]. Malgrat això, en aquest capítol em centraré exclusivament en les xarxes neuronals quàntiques.

¹Classificar dades de dimensions petites en espais vectorials molt grans.

4.1 Descens del gradient quàntic

De moment tindrè en consideració que una xarxa neuronal quàntica és un circuit quàntic qualsevol però parametritzat, és a dir, que té portes quàntiques parametritzades. Això juntament amb una funció de pèrdua i un *dataset* és suficient per explicar com s'actualitzen els paràmetres.

L'objectiu principal és avaluar la derivada respecte a un paràmetre. Sorprenentment, és dona a terme d'una manera més senzilla que amb una xarxa neuronal clàssica. Tan sols s'utilitza la definició de la derivada per poder avaluar-la: S'altera lleugerament un sol paràmetre i es treu la diferència entre dos outputs del circuit quàntic que tenen el paràmetre alterat. Aquest mètode per avaluar la derivada s'anomena *parameter shift* [4, 47].

Si un circuit quàntic té un vector de paràmetres θ , per un paràmetre θ_i , es defineix un vector de pertubació Δ_i ple de zeros, d'igual mida que θ , però que en la posició de θ_i té un 1. He vist que per un vector de pertubació $|\Delta_i\rangle$, donat un vector de paràmetres $|\theta\rangle$ es compleix que²:

$$\exists |\Delta_i\rangle \iff \langle \Delta_i | \theta \rangle = \theta_i$$

A partir d'aquest vector es pot definir la derivada de la funció de pèrdua $\mathcal{L}(\cdot)$ respecte al paràmetre θ_i [4]:

$$\frac{\partial}{\partial \theta_i} \mathcal{L}(\theta) = \mathcal{L}(\theta + \frac{\pi}{4} \Delta_i) - \mathcal{L}(\theta - \frac{\pi}{4} \Delta_i)$$

Es pot veure que el vector $\theta \pm \frac{\pi}{4} \Delta_i$ és el vector θ , però amb una petita variació en la posició i que és la que correspon al paràmetre θ_i . Amb aquest mètode ja es pot desenvolupar pràcticament qualsevol actualització de paràmetres, aquesta és la part senzilla de les xarxes neuronals quàntiques, la dificultat radica en la forma dels circuits quàntics que les componen.

²Utilitzo momentàniament la notació de Dirac perquè no se escriure aquesta equació d'una altra manera.

4.2 Circuits quàntics per xarxes neuronals

L'única qualitat obligatòria que hi han de dintre aquests circuits és que han d'estar parametritzats. Usualment, estan compostos per una gran quantitat de portes rotacionals parametritzades, les quals ja he presentat en les equacions 2.2, 2.3 i 2.4.

El problema al qual s'enfronten els investigadors dedicats a les xarxes neuronals quàntiques és la manera amb la qual implementar funcions no lineals en els circuits quàntics. Aquests tipus de funcions són les responsables de la gran complexitat i profunditat de les xarxes neuronals clàssiques. A primera vista pot semblar una tasca quasi impossible a causa de la naturalesa lineal de la computació quàntica. Tanmateix, durant els anys s'han desenvolupat diverses tècniques per donar a terme aquesta fita, les quals comentaré a continuació.

A partir d'una combinació de rotacions i portes que entrellacen qubits es pot arribar a implementar una funció semblant a la tangent hiperbòlica³ en un circuit quàntic [48]. No obstant això, aquest mètode té un gran desavantatge, consisteix en un circuit que s'ha d'anar mesurant i repetint per veure si funciona correctament, els autors parlen de *repeat until success* perquè s'ha de mesurar un qubit i mirar si dona $|0\rangle$ per assegurar que aquesta funció ha sigut aplicada correctament⁴.

En un article posterior⁵, en el qual els autors generen distribucions contínues a partir d'una xarxa generativa adversària quàntica, s'especifica que les no-linearitats presents en l'algoritme no formen part del circuit quàntic, és a dir, funcions que s'implementen clàssicament als resultats dels circuits quàntics o a les dades que s'introdueixen als circuits [49]. Aquesta és la manera més simple de resoldre aquest problema.

³La tangent hiperbòlica també s'utilitza com a funció no lineal en el *deep learning*.

⁴En cas que doni $|1\rangle$, s'ha de repetir el procés.

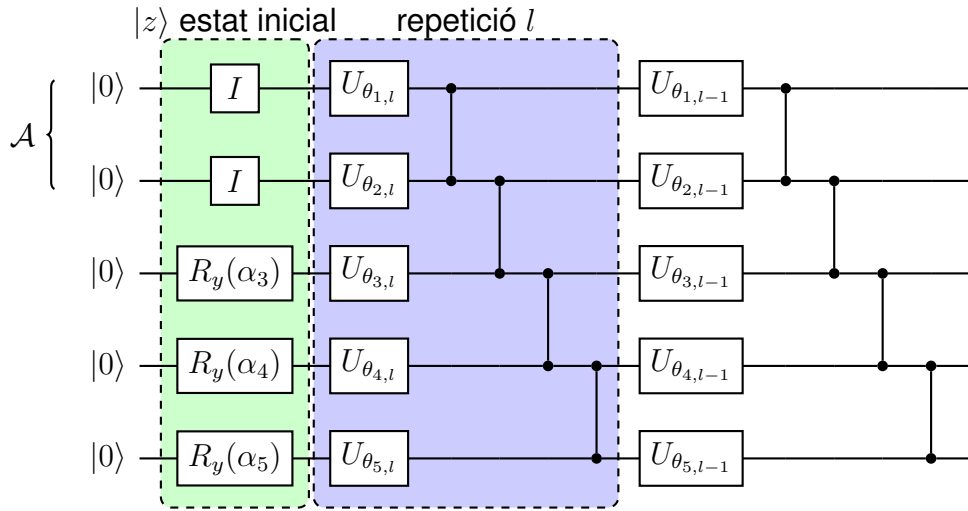
⁵Realitzat en part per un dels autors del mètode anterior.

En 2019 es va publicar un dels articles que més m'agraden⁶ Cong et. al. (2019) [50], en aquest els autors presenten una xarxa neuronal convolucional quàntica. No fa falta entrar en detall, però es diuen convolucionals perquè s'apliquen convolucions a les imatges que es volen classificar, es multiplica una part de la imatge per una matriu que s'anomena filtre [51]. Simplement, tenen un altre tipus de capes que no són les capes completament connectades. L'única afirmació dels autors que s'ha de recalcar, és que a partir de reduir els graus de llibertat (*degrees of freedom*) en els circuits quàntics, sorgeixen no-linearitats. Això és a causa dels mesuraments parcials que es donen a terme en un moment determinat de l'algoritme.

Un dels mètodes que m'ha semblat més interessant, és l'implementat en una altra xarxa convolucional quàntica, on a l'aplicar els filtres que s'utilitzen per a la convolució, s'implementa una funció no lineal [52]. Aquest mètode no el puc arribar a comprendre, les equacions que fan servir són molt complexes i no sembla que arriben a utilitzar un mesurament parcial en cap moment. Simplement, els autors presenten una equació i diuen que no és lineal, i no puc arribar a comprendre perquè no ho és.

Finalment, he de parlar de l'article que he seguit per fer aquest treball, Huang et. al. (2021) [1], en aquest, al igual que en l'article de Cong et. el. (2019) [50], s'utilitzen mesuraments quàntics per introduir no-linearitats a l'algoritme. Concretament, els autors implementen aquests mesuraments tant en el generador quàntic d'imatges, com en el discriminador. Els circuits quàntics que utilitzen per al generador d'imatges tenen la següent forma:

⁶Utilitzen imatges de gats a les figures i és un dels primers articles que vaig llegir d'aquest camp fa més de dos anys. A més a més la xarxa neuronal esmentada en l'article està completament programada en *TensorFlow Quantum* [4], i això sempre s'agraeix.



On les portes R_y amb un paràmetre α_i , marcades en verd, són utilitzades per introduir les dades al circuit. Aquestes dades són simplement soroll que crea varietat en els outputs dels circuits, igual que el soroll que s'empra en les GAN clàssiques (algoritme 1).

El circuit que genera les imatges realment consisteix en les portes U_θ i CZ, marcades en blau. Aquestes portes s'agrupen en les repeticions- l , que s'utilitzen per afegir profunditat i complexitat als circuits. En aquest cas el circuit té dues repeticions- l ⁷. Al mesurar es fa un mesurament parcial, però els autors especifiquen que es fa d'una forma concreta que explicaré a continuació.

Per un estat $|\Psi_\alpha\rangle$ que surt del circuit especificat posteriorment, l'estat $\rho(z)$ després d'una mesura parcial sobre el qubits \mathcal{A} , és:

$$\rho(z) = \frac{\text{Tr}_{\mathcal{A}}(\Pi_{\mathcal{A}} |\Psi(z)\rangle \langle \Psi(z)|)}{\text{Tr}(\Pi_{\mathcal{A}} \otimes I_{2^{N-N_{\mathcal{A}}}} |\Psi(z)\rangle \langle \Psi(z)|)}$$

Els autors afirmen que aquest estat $\rho(\alpha)$, és una funció no lineal de l'estat $|z\rangle$. Això és degut al fet que tant el denominador com el numerador de l'equació són funcions de $|z\rangle$.

Tanmateix, en altres treballs com per exemple Zoufal et.al.(2019) [53], on es defineix una GAN quàntica que genera distribucions de probabilitat, els autors no mencionen la necessitat de tenir funcions no lineals en alguna part de l'algoritme.

⁷És a dir $l = 2$.

Part III

Marc Experimental

Capítol 5

Plantejament de l'hipòtesi

Alhora de fer un experiment, podria haver anat per altres vies com la generació d'imatges amb color o la implementació d'una porta X en una part específica del circuit quàntic que genera les imatges, que al posar-la o no, el model generes dos tipus d'imatges a través del mateix circuit. Tanmateix, les dues propostes requerien desenvolupar nous conceptes, ja que són idees pròpies. No em veia amb la confiança de poder implementar-les i obtenir objectius positius.

Llavors, al ser la implementació de les funcions no lineals un assumpte lleugerament conflictiu entre els diversos models de xarxes neuronals quàntiques, com ja he comentat en la secció [4.2](#). Vaig decidir des d'un principi centrar-me en aquesta qüestió en concret.

La pregunta a investigar és la següent:

La incorporació d'una funció no lineal en el circuit quàntic generacional, a partir d'una mesura parcial, és la causa que el model arribi més ràpid al punt òptim?

En altres paraules, volia veure si la mesura parcial afectaria positivament en l'eficiència del model, fent que la generació de les imatges desitjades es donés a terme en un menor temps.

Sembla una qüestió senzilla, però la dificultat de l'experiment radica en crear la xarxa neuronal en si, amb totes les seves parts accessibles per poder fer els canvis que siguin necessaris. L'única manera de fer l'experiment seria programant el model, una tasca que tenia clar que faria des del principi.

Capítol 6

Desenvolupament del model

Posteriorment a començar a programar el model, ja tenia clar que ho havia de fer en *Python*. En el passat creat alguns algorismes i tenia experiència construint i executant circuits quàntics amb *Cirq* [54], una eina desenvolupada per Google. A més a més, sabia de l'existència de *TensorFlow Quantum* [4], una altra eina desenvolupada per Google destinada a la creació de xarxes neuronals quàntiques i algorismes de *quantum machine learning* en general. També tenia una mica d'experiència en aquest *framework*. Per tant, desenvolupar el model en *TensorFlow Quantum* va ser la primera opció.

Tenia pensat basar el meu codi en el tutorial de *TensorFlow* sobre una xarxa convolucional generativa adversària. Havia de canviar el generador per un circuit quàntic que s'optimitza a partir d'un diferenciador¹ automàtic provinent de *TensorFlow Quantum* [4].

També havia de canviar l'arquitectura del discriminador. Havia de ser una xarxa més simple amb unes poques capes totalment connectades.

El primer problema que em vaig trobar va ser la creació del *dataset* que alimenta a la xarxa discriminativa. A causa de la peculiaritat de les imatges que volia generar, havia de crear-lo manualment. Usualment les imatges que componen els *datasets* utilitzats en *deep machine learning* són extrems de bancs d'informació amb mides enormes. En el meu cas, havien de ser generades per mi, per tant havia de convertir matrius de *Numpy* en *datasets* de *TensorFlow*. Recordo que en va costar arribar a tindre la solució a aquest problema.

¹Objecte que calcula el gradient d'un circuit quàntic.

6.1 Discriminador

Una vegada ja tenia fet el *dataset* em vaig posar a fer el model. En el tutorial per una [DCGAN \(GAN convolucional\)](#) els dos models eren entrenats per la funció `train_step()`, que representa una iteració en el procés d'optimització. En aquesta es crida a la funció `tf.GradientTape` per guardar el diferenciador automàtic. El problema que vaig dintre amb aquesta funció és que directament no funcionava amb el discriminador, aquest no era optimitzat. Després d'intentar solucionar l'error pels meus propis medis, mirant la causa d'aquest i de buscar a fòrums la solució o causa, em vaig rendir. Ja havia estat uns quants mesos intentant desenvolupant el model amb *TensorFlow* i *TensorFlow Quantum*. Havia creat les capes del generador quàntic manualment, també ho havia fet amb l'optimitzador². Tenia el model gairebé acabat, però perquè no podia optimitzar el discriminador em vaig veure obligat a canviar d'estratègia.

Existeixen dos grans *frameworks* per crear i executar circuits quàntics: *Cirq* [54], desenvolupat per Google, i *Qiskit* [55], creat per IBM. Una vegada vaig decidir no continuar amb *Cirq*, havia de provar amb *Qiskit*. La veritat, havia d'haver començat amb *Qiskit* des del principi, és més útil (té moltes més característiques), i el més important, té una major comunitat, per tant, és més fàcil trobar solucions a l'error que pots tenir i és més fàcil trobar a persones disposades a ajudar-te.

Igual que *Cirq* té un *framework* per poder desenvolupar xarxes neuronals (*TensorFlow Quantum*); *Qiskit* també té el seu, anomenat *PyTorch* [41], no obstant no té una integració tan directa, ja que no estan desenvolupats pel mateix equip, ni la mateixa companyia.

Per tant, en canviar de *Cirq* a *Qiskit*, també havia de canviar de *TensorFlow* a *PyTorch*, però no tenia res d'experiència amb *PyTorch*, sabia que la transició seria complicada, i tenia raó. No vaig ni aconseguir crear el *dataset* d'imatges per alimentar al discriminador.

²No sabia ni si funcionarien adequadament, pel fet que per provar-ho, havia de tenir tot el model enllestit.

Després d'intentar-lo amb *TensorFlow/TensorFlow Quantum* i amb *PyTorch/Qiskit*, vaig decidir prescindir de *frameworks* per crear models de *machine learning*. Crearia tant el discriminador com el generador manualment.

El discriminador, al ser una xarxa tan simple, la podria crear des de zero. Llavors vaig començar a buscar codi a internet que pogués utilitzar. Volia una xarxa neuronal feta amb *Numpy*, una llibreria de *Python* per fer càlculs amb vectors i matrius amb la qual tenia bastant experiència.

Després de provar dues opcions que més o menys m'agradaven³, va aparèixer un repositori⁴ de Michael Nielsen, coautor de *Quantum Computation and Quantum Information* [7] que em va salvar.

El repositori tenia codi per xarxes neuronals que estava estructurat d'una forma que m'agradava i encara més important, que entenia. Inclús tènien diverses versions d'una mateixa xarxa neuronal, amb un nivell de complexitat diferent. Llavors, a partir del model més simple que hi havia en el repositori⁵, vaig començar a desenvolupar el discriminador.

Per verificar el correcte funcionament del discriminador, i per poder comprendre com funciona el codi, vaig utilitzar la teoria del capítol 3. No obstant això, el principal problema que em vaig trobar va ser derivar la funció de pèrdua, per exemple la *Minimax*, s'ha de derivar-la a partir de dues equacions, una per a cada etiqueta.

Com es pot veure al codi final pel discriminador a l'annex E.2.1.0.2, he fet bastants canvis, però no he canviat l'estructura del codi. La majoria dels canvis són per afegir més funcionalitat al model, com per exemple l'emmagatzematge de les dades per poder al final veure-les en un gràfic. Tanmateix, si que vaig haver-hi de canviar la funció de pèrdua utilitzada, ja que originalment, era una MSE.

³Buscava codi estructurat d'una forma en concret, que estigui dissenyat amb la filosofia de *Object Oriented Programming*, una forma d'escriure codi en el qual tot s'implementa en un sol objecte.

⁴[Enllaç del repositori de Michael Nielsen](#), la persona que més m'ha ajudat a fer aquest treball. Però no m'ha ajudat directament, ho dic perquè és el coautor de *QC i QI* [7] i autor de la xarxa neuronal que va fer que progressés amb la programació.

⁵Es pot veure el codi original a l'annex E.2.1.0.1.

El canvi més important és que inclou les dues formes d'optimitzar el model, amb dues funcions de pèrdua que funcionen de manera diferent, però que són la mateixa, la *Binary Cross Entropy* i la *Minimax*. Això en el codi està materialitzat en dues funcions⁶ diferents, `backprop_bce()` i `backprop_minimax()`. No tenen cap diferència en termes d'eficàcia o rapidesa i les vaig fer tan sols per comprovar aquest fet.

Mirant enrere, és una tasca fàcil, però en el moment vaig tindre moltes dificultats alhora d'adaptar el codi a les meves necessitats. Un aspecte que em va ajudar molt va ser comprendre la teoria darrera de les xarxes neuronals que he presentat a la secció [3.2](#).

El codi final del discriminador es pot veure a l'annex [E.2.1.0.2](#). També en el repositori d'aquest treball hi ha un altre arxiu que conté una altra versió en la qual intentava implementar diverses funcions d'activació en el model, però no ho vaig aconseguir, tanmateix, no em preocupa perquè no és una part vital del treball, no passa res per tindre el discriminador només amb la funció sigmoide com a funció d'activació. Tenia l'intenció de fer-ho perquè en l'article de Huang et. al., la funció d'activació que s'utilitza és la *ReLU* [\[1\]](#).

6.2 Generador

El desenvolupament de l'altra part del model, el generador, va ser molt diferent. Després de provar de fer-ho amb *TensorFlow Quantum*, sense obtenir bons resultats, no tenia altra opció que fer-ho tot manualment i jo mateix⁷, això no obstant, ja tenia experiència en fer petites xarxes neuronals quàntiques i això em tranquil·litzava. Sabia perfectament el que havia de fer, i com ho havia de fer: Implementar el mètode de *parameter shift* en els circuits quàntics dels quals he parlar en la secció [4.2](#).

⁶Funcions de *Python*.

⁷No em vaig ni plantejar buscar codi per internet perquè pensava que els autors dels papers que vaig llegir, les úniques persones que sabia que podien tindre el codi, no el penjarien.

Una vegada tenia la funció que creava els circuits quàntics, em vaig posar amb l'optimització dels paràmetres, de la qual parlaré a continuació.

Primer de tot cal remarcar que el model s'optimitza a partir d'un *batch*, és a dir, un grup de dades, en aquest cas, un grup d'imatges. S'agafa la mitjana dels errors de cada *batch*, i s'actualitzen els paràmetres a partir d'aquesta. La motivació per treballar en *batch* i no dades individuals és perquè s'ha de mirar l'error d'unes quantes dades a la vegada, d'aquesta manera l'optimització és més robusta [2].

Tornant al funcionament de l'optimització, ho explicaré tal i com està implementat en el codi. Per començar s'han d'agafar els paràmetres a optimitzar, que estaran en forma d'un vector θ , escollir un d'ells que anomenaré θ_i , que és el paràmetre que s'optimitzarà, i crear un vector de pertubació Δ_i .

Llavors es creen dos vectors de paràmetres nous, θ_i^+ i θ_i^- : Es multiplica $\pm \frac{\pi}{4}$ pel vector de pertubació, i es suma el resultat al vector de paràmetres original⁸.

A continuació, es creen dues imatges, cadascuna corresponent a un dels vectors de paràmetres creats, una amb θ_i^+ i l'altra amb θ_i^- .

Per últim, s'avalua la funció de pèrdua per a cada imatge generada i es treu la diferència entre elles, d'aquesta manera calculant una derivada $\frac{\partial \mathcal{L}}{\partial \theta_i}$ respecte al paràmetre θ_i .

Una vegada es té una derivada per a cada paràmetre de θ es pot construir el vector gradient ∇_{θ} . He de dir que aquest vector té la mateixa mida que el vector θ , com que cada element en el gradient correspon a la derivada d'un paràmetre de θ .

Al llarg de l'optimització es van sumant les derivades a ∇_{θ} , si això es fa per a tots els paràmetres de θ i per a totes les dades del *batch*, finalment es pot treure la derivada mitjana dividint els elements de ∇_{θ} per la quantitat de dades que hi ha en un *batch*.

Amb el gradient ∇_{θ} es poden finalment optimitzar tots els paràmetres, d'acord amb les derivades (concretament la mitjana de moltes derivades) de les quals està compost.

⁸Cada un d'aquests vectors de paràmetres té una petita variació en un paràmetre, i cada vector té una variació en un sentit.

Algorithm 2 Pseudocodi per una xarxa generativa adversària quàntica

```
 $\nabla = 0$  ▷ Creació del vector  $\nabla$  que té la mateixa mida que  $\theta$   
for soroll en batch do  
  for paràmetre  $\theta_i$  en  $\theta$  do  
     $\Delta_i = 0$  ▷ Creació del vector pertubació d'acord amb el vector  $\theta$   
     $\theta_i^+ = \theta + \frac{\pi}{4} \Delta_i$   
     $\theta_i^- = \theta - \frac{\pi}{4} \Delta_i$   
  
     $\text{Imatge}_1 = \text{generador}(\text{soroll}, \theta_i^+)$   
     $\text{Imatge}_2 = \text{generador}(\text{soroll}, \theta_i^-)$  ▷ Generació de les imatges  
  
     $\text{Predicció}_1 = \text{discriminador}(\text{Imatge}_1)$   
     $\text{Predicció}_2 = \text{discriminador}(\text{Imatge}_2)$  ▷ El discriminador posa una  
    etiqueta a cada imatge  
  
     $\text{Diferencia}_i = \mathcal{L}(\text{Predicció}_1) - \mathcal{L}(\text{Predicció}_2)$   
  
     $\nabla_\theta = \nabla_i + \text{Diferencia}_i$  ▷ On  $\mathcal{L}$  és la funció de pèrdua del generador  
  end for  
end for  
  
for  $\theta_i$  en  $\theta$  i  $\nabla_i$  en  $\nabla_\theta$  do ▷ Per a cada paràmetre i per a cada error  
   $\theta_i^{t+1} = \theta_i + \frac{\eta}{M} \nabla_i$  ▷ Actualització del paràmetre amb la mitjana de l'error que  
  correspon a aquest paràmetre  
end for
```

FIGURA 6.1: Em refereixo a l'input de la xarxa generacional com a soroll, ja que és més encertat d'aquesta manera. El vector ∇ té la mateixa mida que el vector de paràmetres θ , cal notar que també té la mateixa mida els vectors θ_i^\pm , ja que aquests vectors són θ amb una alteració al paràmetre θ_i . Les paraules *generador* i *discriminador* denoten els respectius models, per tant, Imatge_i i Predicció_i són els outputs dels models.

El pseudocodi per aquest procediment es pot trobar en la figura 6.1. En aquesta es pot veure que es crida al discriminador perquè posi etiquetes a les dues imatges generades, això és per poder avaluar aquestes etiquetes amb la funció de pèrdua. Per després poder agafar la diferència i d'aquesta manera calcular la derivada. El codi pel generador es pot trobar en l'annex, E.2.1.0.3.

6.3 Creació del model

Quan ja es tenen les dues parts del model, aquestes s'han d'ajuntar d'alguna manera. El que jo he fet és posar-ho tot en una classe de *Python* anomenada `Quantum_GAN`, que conté les funcions per definir el model, per executar-lo, per guardar les seves dades i per crear els gràfics que serveixen per avaluar l'eficiència del model. Aquesta classe és el tros de codi que junta tot, tant el discriminador i el generador. També agafa altres funcions que són necessàries, com per exemple la sigmoide. Aquestes funcions que no estan en l'arxiu del generador ni en el del discriminador, es troben en `functions.py`.

S'han de tenir en compte algunes de les funcionalitats d'aquesta classe. En particular, com circulen les dades del generador als discriminadors i com es creen les gràfiques amb les quals valoro l'eficiència dels models.

La funció que s'utilitza per entrenar el model és `Quantum_GAN.train()`. Aquesta té com input un *dataset* amb imatges reals i el soroll per poder entrenar el generador. Aquest *dataset* es divideix en grups d'imatges (els *batches*), cada *batch* conté tant imatges reals, com soroll en la mateixa quantitat.

En una iteració primer s'optimitza el generador, que substitueix el soroll del *batch* amb les imatges que genera. Llavors es passa el *batch* al discriminador que s'optimitza tant amb les imatges reals com amb les imatges falses. Aquest és el procés pel qual s'optimitzen els dos models.

Pel que fa a la creació de les gràfiques, aquesta classe selecciona una imatge real aleatòria i una de falsa per avaluar la funció de pèrdua en la iteració, també crea les etiquetes utilitzades en aquesta l'avaluació. Totes aquestes dades s'utilitzen per a la creació de diversos gràfics que mostren l'evolució d'aquestes dades a través de tota l'optimització.

6.4 Execució del model

No obstant això, en l'arxiu `qgan.py`, no es troba l'execució del model, només està la definició de la classe `Quantum_GAN`. Això és perquè el codi realment s'executa en l'arxiu `main.py`. Aquest és l'únic arxiu que té codi que realment s'executa, els altres arxius només tenen definicions. Per projectes relativament grans, com aquest, convé tenir un arxiu que fa tot el treball, el qual crida a totes les funcions definides en altres arxius i les executa d'una manera ordenada.

Com es pot veure en l'annex, [E.2.1.0.5](#), aquest arxiu té molt poc codi. En ell només es crea amb *Numpy* el dataset, es defineix el discriminador i el generador, amb els quals es crea el model amb `Quantum_GAN`.

Finalment, es crida a la funció `Quantum_GAN.train()` per optimitzar el model. En acabar l'optimització es criden les funcions `Quantum_GAN.plot()`, `Quantum_GAN.create_gif()` i `Quantum_GAN.save()`, les quals donen a terme funcions complementàries com la creació de les gràfiques, la creació d'un GIF que mostra com les imatges van evolucionar al llarg de l'optimització, i l'emmagatzematge de les dades rellevants que s'han creat durant l'optimització.

Les imatges que vaig escollir per generar són les mateixes que van generar en l'article en el qual vaig basar el treball. No entraré molt en detall sobre les distribucions concretes que formen les imatges, però per tindre una imatge general sobre com són es pot veure la figura [6.2](#). O directament es pot consultar les distribucions en l'article original [\[1\]](#) o veure com es generen les imatges en el codi [E.2.1.0.5](#).

Si executen el model, amb una mida de *batch* de 10, tant el *learning rate* del discriminador com del generador a 0.1, i un total d'iteracions de 400, hi ha una garantia d'arribar a la convergència desitjada. En altres paraules, els dos models arriben a l'equilibri de Nash i no poden continuar l'optimització. En aquest punt és quan les imatges falses i les reals són iguals.

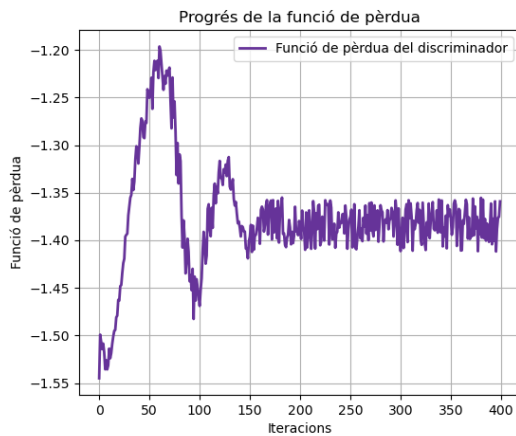


(A)

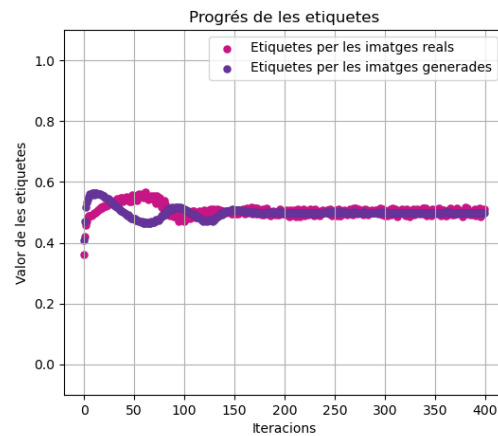


(B)

FIGURA 6.2: **A)** Aquest és un exemple d'una imatge del *dataset* amb el qual s'entrena el model que he creat. És simplement una imatge en blanc i negre de 4 píxels, en la qual hi ha dos píxels amb un valor aproximant de 0.5 i els altres dos amb un valor de 0. Les imatges es creen a partir d'una *array* de *Numpy*, com per exemple `np.array([[0.47804505, 0.], [0.47804505, 0.]])`. **B)** Aquest és un exemple d'una imatge generada que tracta de ser el més semblant possible a les imatges del *dataset*. Es pot veure que són indistingibles a simple vista, però al veure els valors es pot apreciar com els nombres no són els mateixos: `np.array([[0.504324, 0], [0.495676, 0]])`. Això és perquè només pel valor de 0.5 és possible tenir els dos píxels amb el mateix valor, sempre que els valors dels altres píxels sigui zero. Cal recordar que els valors dels píxels han de sumar 1. Ja que són valors generats per un circuit quàntic. Per tant, les imatges del *dataset* com la que he mostrat no són possibles de crear. He pensat fer-ho d'aquesta manera per veure si es generaria les imatges correctament amb aquesta limitació, el resultat ha sigut que les imatges que es generen són com la mitjana dels valors de les imatges del *dataset*, tal com es pot apreciar en la figura 7.5.



(A)



(B)

FIGURA 6.3: En aquestes figures es pot veure com per la iteració 250, el model ja s'ha estabilitzat. Això és perquè les etiquetes i el valor de la funció de pèrdua convergeixen en un valor. **A)** Funció de pèrdua per cada iteració. A partir de la iteració 175, es pot veure com el valor de la funció s'estabilitza en l'interval $(-1.35, -1.4)$, això concorda amb els valors de les etiquetes en les mateixes iteracions, ja que $\log(\frac{1}{2}) + \log(1 - \frac{1}{2}) \simeq -1.38$. **B)** Etiquetes per les imatges reals i generades per cada iteració. Es pot observar que els valors de les etiquetes per les imatges reals són més inestables que els valors de les generades. Això és perquè les imatges reals tenen una major variació en els valors dels píxels, mentre que en les generades aquest fet no és tan notable. Per tant, el discriminador assigna etiquetes amb una major variació.

En la figura, 6.3b, es pot veure com les etiquetes pels dos tipus d'imatges van oscil·lant fins a estabilitzar-se. El mateix es pot dir per la funció de pèrdua, com es pot veure en la figura 6.3a.

A causa del fet que aquests gràfics són molt semblants als seus anàlegs de les GANs clàssiques i que les distribucions reals i generades són pràcticament iguals (com es pot veure a la figura 6.4), considero que el model funciona correctament.

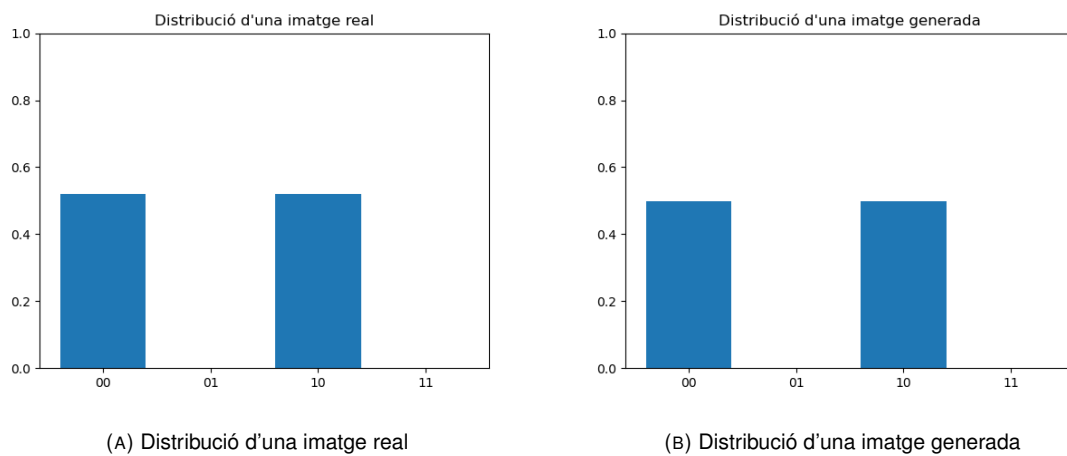


FIGURA 6.4: Comparació d'una imatge generada i una real, quan el model ha assolit la convergència. En l'eix Y es pot veure el valor d'un píxel, mentre que en l'eix X estan els píxels.

Abans he especificat que al cap de 400 iteracions podem tenir la garantia d'arribar al punt d'equilibri, no obstant això, es pot arribar a aquest punt amb menys iteracions, el que vull dir és que amb 400 de segur que s'arriba. Això és perquè, com a tots els models de *machine learning* hi ha una part de sort implicada. Si els paràmetres inicials són més semblants als paràmetres desitjats, el model assolirà la convergència més ràpidament.

Capítol 7

Realització del experiment

Una vegada havia confirmat el correcte funcionament del model, vaig començar amb el desenvolupament del experiment. Havia de crear els dos tipus de circuits quàntics que necessitava, uns amb un sistema ancilla, i uns altres sense.

Un sistema ancilla, és un grup de qubits sobre els quals es donen a terme operacions, però que no es mesuren per treure l'output del circuit. Aquests qubits es pot veure clarament en la figura [7.1](#).

Primer de tot he de mencionar que havia de fer alguns canvis al generador per acomodar aquests qubits ancilla.

Segon, cal notar que no he replicat exactament la metodologia emprada en l'article original. En ell tenen aquesta equació [\[1\]](#):

$$\rho = \frac{\text{tr}_A(\Pi_A |\psi\rangle \langle\psi|)}{\text{tr}(\Pi_A \otimes I_{2^N - 2^{N_A}} |\psi\rangle \langle\psi|)}$$

Com ja he comentat en la secció [2.4.1.1](#), no veig com aquesta equació pot tenir sentit, per tant la vaig ometre del meu experiment.

L'alternativa que he fet servir són els mesuraments de *Qiskit*, al definir el circuit específic quins són els qubits que no vull mesurar, que són els qubits que formen part del sistema ancilla. No obstant això, no sé exactament quin és

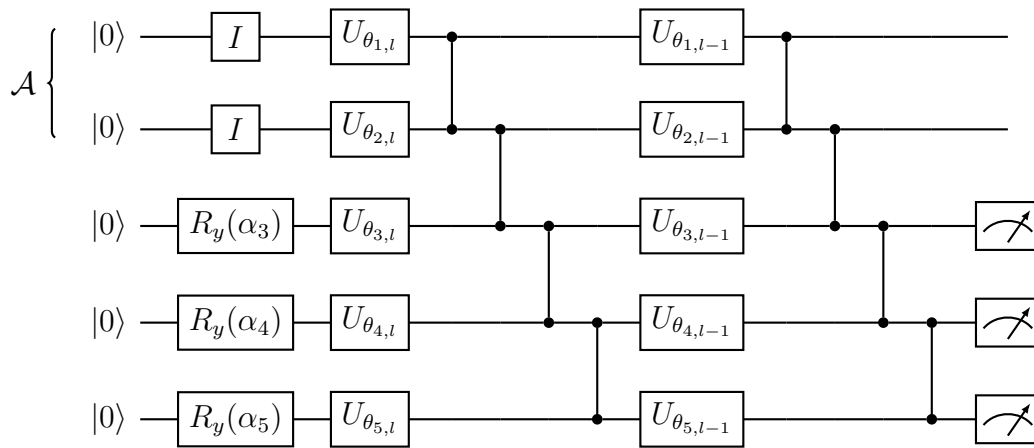


FIGURA 7.1: Aquí he marcat els qubits ancilla amb \mathcal{A} , són els dos primers. També al final del circuit he afegit mesures als qubits que s'han de mesurar.

el funcionament d'aquest mètode. Però si empro aquest procediment en altres circuits, dels quals sé el resultat, aquest mètode fa el que m'espero¹.

L'arxiu que utilitzo per fer els experiments és `experiment.py`. En ell es pot veure com defineixo dos discriminadors² i dos generadors, que es diferencien pels circuits que fan servir, un amb qubits ancilla i l'altre sense. Aquests models s'agrupen en parelles per definir dues qGANs.

Cal notar que els generadors i els discriminadors³ comencen amb els mateixos paràmetres, per tant, tenen les mateixes condicions, menys els circuits quàntics és clar. També s'utilitza el mateix dataset.

7.1 Anàlisi dels resultats

Si comparem les gràfiques que mostren les etiquetes es pot veure una clara diferència: Els models que tenen els circuits amb els qubits ancilla són més inestables que el que no els tenen, tal i com es pot veure en la figura 7.2. Tanmateix, això no vol dir necessàriament que siguin més eficients.

¹Parlo dels parells de Bell, els circuits quàntics amb entrellaçament més simples que es poden fer.

²Que tenen les mateixes característiques.

³Al principi pensava no tenir els mateixos paràmetres inicials pels discriminadors, però al veure les gràfiques de les etiquetes, l'efecte que tenen aquests és molt notable. Es pot veure com a vegades les etiquetes comencen en uns valors de 1 i en altres de 0. [enllaç per veure una imatge amb totes les gràfiques](#)

També es pot observar una clara diferència en les imatges generades en la primera iteració. En les primeres imatges d'un generador amb la funció no lineal es pot veure un píxel amb un valor de 1 mentre que els altres estan al 0. Mentre que en l'altre tipus de generador es pot veure que els píxels tenen més o menys el mateix valor, d'aquesta manera formant una distribució uniforme.

Aquest fet probablement és causat per l'estructura del circuit que té els qubits ancilla. Tanmateix, perquè exactament passa això està fora dels meus coneixements sobre la matèria⁴.

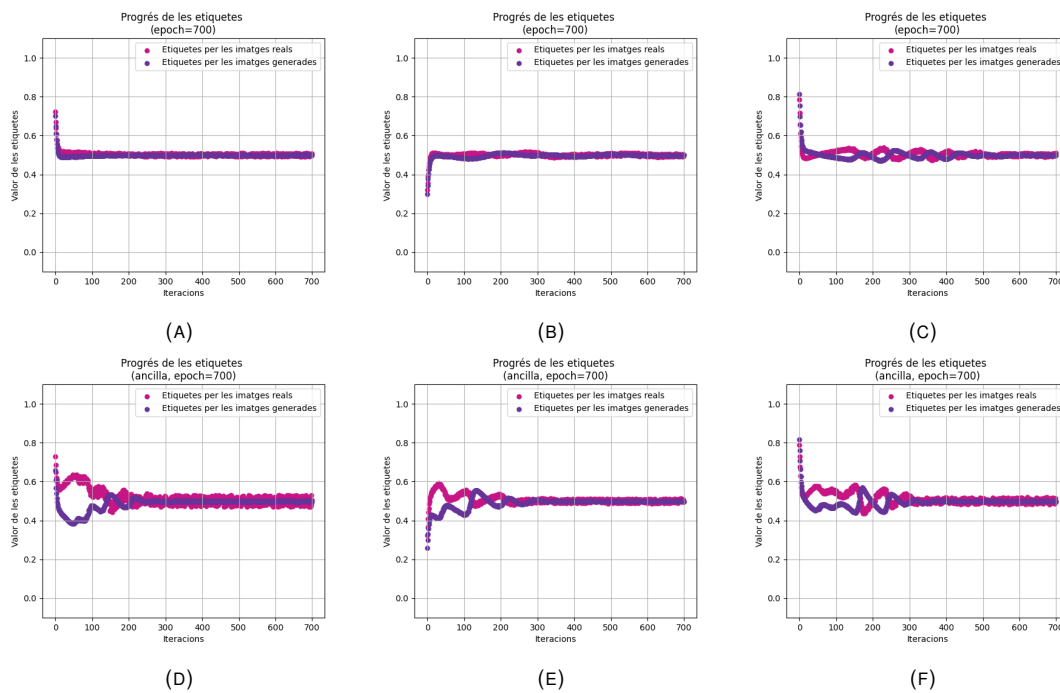


FIGURA 7.2: Es pot apreciar com els models sense la funció no lineal (A, C, B) tenen una major estabilitat en les etiquetes comparats amb els models que sí tenen la funció no lineal (D, E i F).

La part que realment m'intriga és que durant les primeres iteracions, en els circuits amb els qubits ancilla el píxel que té el valor de 1, passa d'un a un altre. Aquest comportament succeeix cada vegada que he executat el codi. Igual que he dit abans, arribar a comprendre la causa d'aquest fet està fora de les meves capacitats.

Estic bastant segur que aquesta fluctuació dels píxels és el factor que causa la inestabilitat que es pot observar en les etiquetes.

⁴Saber com es comporten sistemes quàntics que tenen parts entrellaçades.

Després de mirar les gràfiques que generaven els models, vaig posar-me a redactar les conclusions, malgrat això, no estava satisfet amb la precisió de l'anàlisi de les dades. No podia saber amb certesa quin dels dos tipus de models era el més eficient.

Necessitava una mètrica que en digués quina és la semblança entre les imatges generades i les imatges reals. Sabia que en l'article en el qual he basat el treball els autors havien fet servir una mètrica anomenada *Férchet Score* o puntuació de Férchet [1], en la qual s'emprava la distància de Férchet. Aquesta distància serveix per comparar dues distribucions a partir de la seva mitjana i la seva covariància [56].

Llavors vaig decidir implantar aquesta mètrica, i veure com evoluciona al llarg de l'optimització.

En un article sobre l'avaluació de les imatges generades per les GAN [56], vaig trobar aquesta equació per calcular la distància de Férchet:

$$FD(r, g) = |\mu_r - \mu_g|^2 + \text{tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{\frac{1}{2}}\right)$$

On μ és la mitjana⁵ i Σ és la covariància d'una distribució. Aquesta equació l'he implementat en *Python* mitjançant matrius, és a dir les imatges. La covariància és calculada amb una funció de *Numpy*. A la qual se li dona com input una matriu on cada fila d'aquesta representa una variable d'una distribució. No puc estar 100% segur de què he implementat aquesta mètrica correctament, ja que no sé el funcionament exacte de la covariància, ni el de la funció de *Numpy*. No obstant això, sembla que funciona correctament.

Cal notar que si les dues imatges són exactament iguals, la distancia dona pràcticament zero⁶. Per considerar que dues imatges són «acceptablement semblants», han de tenir una distància entre elles de 10^{-3} aproximadament.

⁵He emprat la mitjana aritmètica.

⁶Python diu que és d'un ordre de magnitud de 10^{-16} aproximadament.

Part IV

Conclusions

Una vegada havia implementat la distància de Férchet per poder avaluar el rendiment dels models, vaig arribar a una clara conclusió:

Els models que tenen implementada la funció no lineal són més eficients.

Afirmo això perquè els models sense la funció no arriben a estabilitzar-se, és a dir, no arriben al punt d'equilibri. En canvi, els models que tenen la funció sí ho fan. Arribar al punt d'equilibri garanteix que les imatges generades siguin com les reals. Per molt més que el model segueixi optimitzant-se, sempre donarà les millors imatges generades.

Els models que no presenten la funció no assoleixen aquest punt a causa que la semblança de les imatges oscil·la, és a dir, el model pot arribar a generar imatges correctament, però no ho fa indefinidament, al cap d'unes interaccions la semblança de les imatges generades amb les reals ja no és significativa. Simplement, la qualitat de les imatges generades va oscil·lant entre bona i dolenta. Tot i que hi ha excepcions, a vegades els models sense la funció no experimenten aquesta oscil·lació. Tanmateix cal remarcar que les vegades que passa són notablement més altes que les que no passa. Com es pot veure amb les dades de la taula 7.1.

Aquest problema no és causat pel discriminador, això és perquè la variable independent de l'experiment és el circuit quàntic del generador, per tant, és el factor que té més possibilitat de ser el causant d'aquest comportament. No sé molt bé la causa exacta, però està clar que el mesurament parcial té un impacte.

	Presenta oscil·lació	No Presenta oscil·lació
Amb Funció No Lineal	0	6
Sense Funció No Lineal	5	1

TAULA 7.1: Les dades provenen d'un total de 6 models, 3 d'ells amb un total de 700 iteracions i els altres 3 amb un total de 550. El nombre d'interaccions no hauria d'afectar de cada manera les dades. Degut si hi ha una oscil·lació, es pot veure clarament a partir de les 400 iteracions. Amb les dades es pot veure que és més probable que un model sense la funció lineal presenti una oscil·lació. Cal notar que cap model amb la funció ha tingut una oscil·lació. Les gràfiques que corresponen a cada model es poden veure en les figures 7.4 i 7.3.

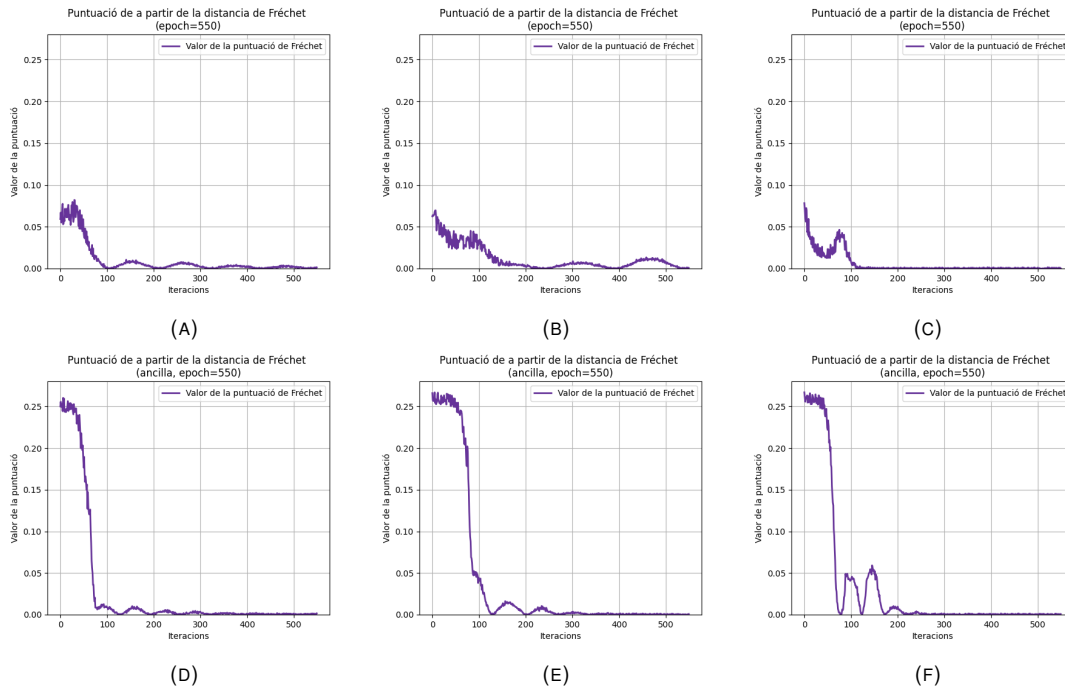


FIGURA 7.3: Totes les gràfiques corresponen a models que s'ha executat al llarg de 550 iteracions. Les figures **A**, **B** i **C**, corresponen a models sense la funció lineal. L'únic d'ells que no presenta una oscil·lació és el **C**. Les gràfiques sense la funció es poden comparar a les d'abaix, les quals representen models amb la funció implementada. Els models han estat creats per parelles, les quals estan organitzades verticalment. És a dir, les gràfiques **A** i **D** representen models que tenen els mateixos paràmetres inicials. El mateix passa amb **B** i **E** i amb **C** i **F**.

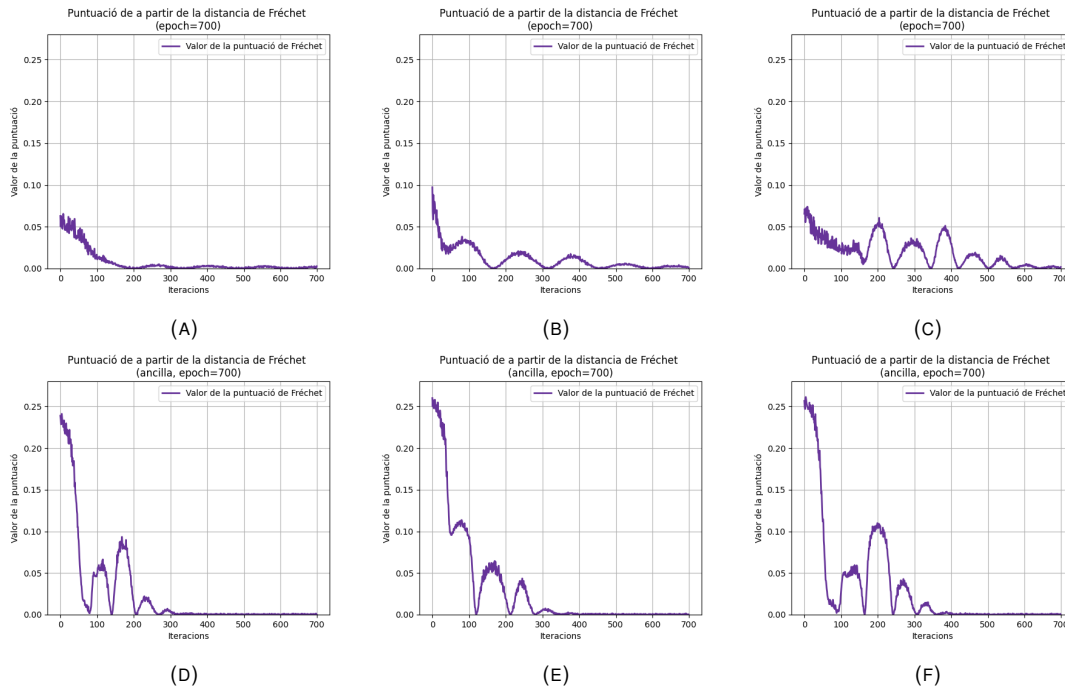


FIGURA 7.4: Aquestes gràfiques corresponen a models que s'han executat al llarg de 700 iteracions. Estan organitzades igual que les gràfiques de la figura 7.3. En aquests casos, com es pot observar tots els models sense la funció no lineal presenten les oscil·lacions. No obstant això, en la gràfica **A**, aquesta es molt feble. Per veure com afecten les oscil·lacions es pot veure la figura 7.5, on estan representades les últimes imatges que han generat els models que corresponen les gràfiques d'aquesta figura.

En les gràfiques 7.3 i 7.4 es pot veure perfectament l'oscil·lació en la Distància de Fréchet. Es pot interpretar aquesta mètrica com la semblança entre les imatges generades i les reals. Si aquesta mètrica convergeix a zero, es pot dir que el model ha arribat al seu punt d'equilibri. Es pot veure com tots els models que tenen el mesurament parcial assoleixen aquest punt. D'aquesta manera demostrant l'eficàcia de la funció no lineal.

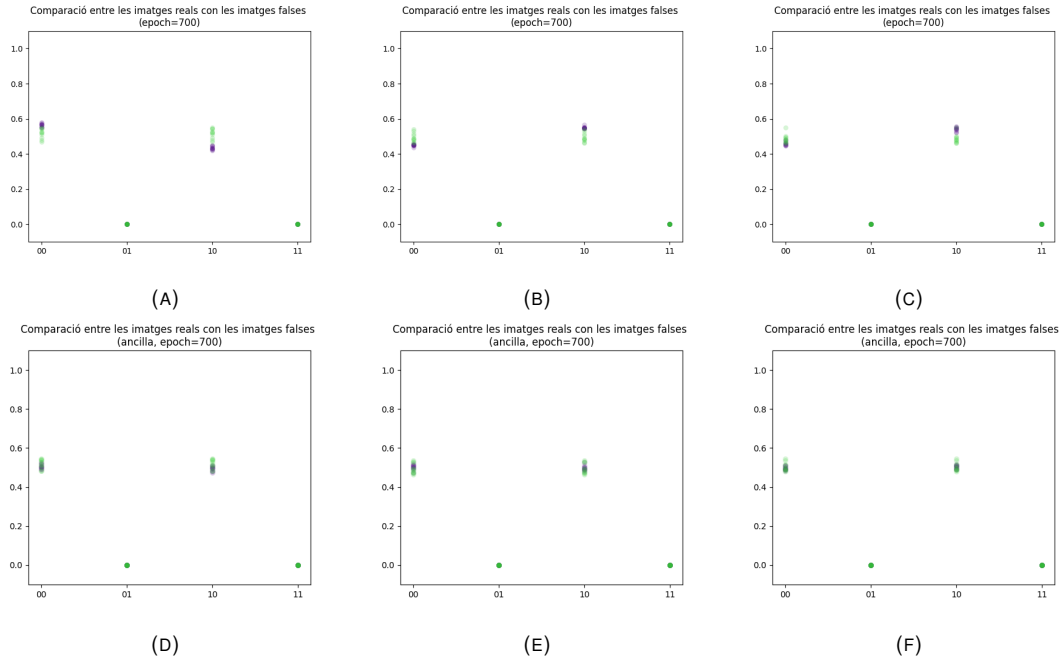


FIGURA 7.5: Aquestes gràfiques corresponen als mateixos models que els de la figura 7.4. Les posicions de les gràfiques són les mateixes, per tant, si estan en la mateixa posició que les de l'altra figura, corresponen al mateix model. Les imatges generades pels models sense la funció no lineal (A, C, B) no s'assemblen a les reals. Es pot veure com els punts de les imatges generades (color violeta) no estan als mateixos valors. Mentre que en les gràfiques D, E i F, sí que ho estan. Els punts violetes sembla que representen la mitjana dels punts verds. També es pot observar que les imatges generades tendeixen a ser més variades que les reals.

La diferència entre les imatges es pot veure la figura 7.5, en la qual es mostren les 10 últimes imatges generades pels models optimitzats al llarg de 700 iteracions.

A partir de la Distància de Fréchet es pot apreciar l'efecte que té el mesurament parcial en la generació de les imatges, fent que aquest procés sigui més eficient i evitant una oscil·lació entre la generació d'imatges de bona i dolenta qualitat. Per tant, es pot afirmar que el mesurament té un efecte positiu en el model, corroborant la tesi exposada en Huang et. al. (2021) [1].

No obstant això, es podria desenvolupar aquest experiment en major profunditat. No he tingut en compte el temps en el qual es tarden a generar les imatges. Els models amb el mesurament parcial implementat són més eficients, però en tenir més qubits és més costós simular-los i, per tant, tarden més temps per cada interacció. He fallat en mirar exactament quina és la diferència en termes de temps, tanmateix, ja que els models sense la funció no lineal no arriben

a assolir el punt d'equilibri la gran majoria de les vegades, des d'un punt pràctic és recomanable utilitzar els models que si tenen la funció.

A més a més es podria investigar la causa de l'oscil·lació en major profunditat. Deixant de banda la investigació teòrica de per què passa, es podria variar el nombre de qubits en els circuits, tant els ancilla, com el total de qubits del circuit, per veure quin és l'efecte que té tenir més qubits en el sistema ancilla. Potser investigaré pel meu compte aquesta qüestió.

El camí que he recorregut

Vull dedicar un petit espai per poder parlar sobre que significa aquest treball per a mi, de tot l'esforç que he posat en aquest, i finalment comentar tot el que he après.

Fa més de dos anys que vaig començar a aprendre conceptes de computació quàntica, ha sigut un camí molt llarg, però que m'ha encant recórrer. Pensar en tot el que he après em dona una gran alegria. Els camps dels quals he parlat realment m'encanten (computació quàntica i intel·ligència artificial). Aprendre tot això m'ajudaria molt en cas que estudiï una carrera en física, que és el meu objectiu. A més a més, alhora de realitzar el petit experiment m'he donat compte de la paciència necessària per poder desenvolupar experiments mitjançant el mètode científic. S'ha d'anar amb molta cura per assegurar-se de que la variable dependent no fossi afectada per les variables de control. Revisant una vegada i una altra fins a estar segur del resultats obtinguts.

Deixant d'una banda els evidents beneficis d'avançar temari d'una carrera que vols fer en un futur, fent aquest treball he après altres coses extremadament útils.

Primer de tot, aquest treball està redactat en \LaTeX ⁷, una espècie de llenguatge de programació per escriure equacions matemàtiques i documents de text. És àmpliament utilitzat per escriure articles científics en el món acadèmic,

⁷[La web oficial de \$\text{\LaTeX}\$](#)

però més important, és utilitzat en l'ambient educatiu de les ciències exactes per poder entregar deures, reports del laboratori, i fer apunts. Tots els meus amics que estudien física o matemàtiques el fan servir.

A continuació està el fet de crear un gran projecte científic en *Python*, he après a programar fins al punt que crec que soc bastant bo, és allò que se'm dona millor diria. Programar en *Python* també és una habilitat que em serà útil a la universitat, perquè és el llenguatge de programació que es fa servir en la carrera de Física a la UB per realitzar experiments computacionals. Però, també vull mencionar que al llarg del procés de la creació del model van sorgir moltíssims problemes, en aquest document només he esmenat una part petita dels problemes que em vaig trobar al llarg del camí. Vaig estar més de mig any programant aquest model fins que estava tot perfecte per fer l'experiment. Definitivament, ha sigut la part més dura de tot el treball.

Per últim lloc, he après a llegir articles científics sobre computació quàntica i intel·ligència artificial. Estic familiaritzat amb la notació que s'emptra en aquest camp i també el llenguatge específic que s'utilitza en els articles, amb això em refereixo a les expressions en anglès. No obstant això, hi ha coses que no puc arribar a comprendre és clar. No tinc els mateixos coneixements d'un estudiant de postgrau de física, matemàtiques o ciències de la computació. I no tinc la intenció d'afirmar que puc valorar els articles d'alguna manera, tan sols puc comprendre'ls fins a un nivell que ha estat suficient per a la realització d'aquest treball.

Aquest treball de recerca s'ha acabat, quan en realitat, és el començament del meu camí en la física, la computació quàntica, les ciències de la computació i la programació científica. La vida universitària em donarà més eines i aprenentatges per poder, de mica en mica, aprofundir i revisar tot el que he après al llarg d'aquest camí.

Part V

Annexos

Capítol A

Que podria fer en un futur?

M'agradaria parlar sobre que és el que vull fer en el futur. Gaudeixo massa treballar en aquests camps i programar experiments, com per a deixar tot això de banda.

La primera opció seria investigar en major profunditat com afecta aquest mesurament parcial a l'eficiència dels models, i veure per què la funció de pèrdua oscil·la d'aquesta manera.

Pel que fa a la segona opció, en aquesta entraria tot el que té a veure amb generar imatges amb la qGAN que he fet. Podria posar-me a generar imatges més complexes com dígitos escrits a mà, igual que han fet els autors de l'article en el qual he basat aquest treball. També podria provar de generar imatges a color, ja m'he llegit uns quants articles que donen a terme aquesta fita i em sembla molt interessant. A més de tot això s'ha m'acudeix implementar el meu codi en un ordinador quàntic de veritat, però no crec que no faci perquè el meu model requereix moltes avaluacions de diversos circuits quàntics i no penso que *IBM Quantum Experience* em deixaria fer-les totes.

Una altra funció que he pensat fer una cosa completament diferent de la generació d'imatges. Existeixen models generatius que es basen en el que s'anomena *feature encoding*, és a dir, trobar les característiques d'un objecte i a partir d'elles sintetitzar nous objectes. Un gran exemple d'aquests models és *Ju-xebox* desenvolupat per OpenAI [57]. Aquest model serveix per a la generació

de música. Consisteix a treure les característiques de l'estil i música d'un artista, i partir d'una gravació d'una veu que canta, transformar la gravació en una cançó amb l'estil de l'artista analitzat.

Aquests tipus de models tenen una arquitectura anomenada *autoencoder*, la qual ja ha sigut implementada en models quàntics [58]. Vull veure si és possible fer alguna cosa semblant a *Jukebox*, no necessàriament per generar dades, sinó per veure si és possible fer *feature encoding* en un circuit quàntic. No penso que sigui probable fer-ho, però serà divertit intentar-ho.

Finalment, he pensat en veure com codificar imatges en color en un circuit quàntic. En una primera instància pots pensar que s'hauria de fer mitjançant una codificació en RGB. Una tasca que ja ha sigut realitzada [59]. No obstant això, he tingut l'idea d'utilitzar una codificació en YCbCr. En comptes de fer servir els vermell, blau i verd; es faria servir un valor de lluminositat, un to de color entre el blau i el groc, i un altre entre el vermell i el verd. Bàsicament és seguir utilitzant tres valors per píxel però els qual representen diferents colors que no són RGB.

Codificar les imatges en YCbCr dona la possibilitat de fer *subsampling*, una tècnica de compressió d'imatges utilitzada en formats com JPEG [60]. Per no dedicar molt de temps a aquesta qüestió, només mencionaré que realitzar *subsampling*, permetria per exemple que cada dos píxels comparteixin els mateixos valor pels color, però que cadascun tingui un valor de lluminositat propi. D'aquesta manera es requeriran emmagatzemar menys nombres per píxel.

Tanmateix, l'opció més intel·ligent seria centrar-me en les bases de la computació quàntica i aprendre les subrutines que existeixen com la transformada de Fourier quàntica i l'estimació de fase, els pilars dels algorismes quàntics. A més a més hi ha algorismes molt interessants i útils que he d'aprendre a utilitzar i encara més important, comprendre'ls; he de desenvolupar més intuïció. Aquesta última opció és que segur que faré en algun moment, possiblement en l'estiu abans d'entrar en la universitat.

Capítol B

Més àlgebra lineal

B.1 Curs ràpid de la notació de Dirac

A la taula següent hi ha un resum de conceptes matemàtics de l'àlgebra lineal importants expressats en la notació de Dirac [61]:

Notació	Descripció
z	Nombre complex
z^*	Conjugat complex d'un nombre complex z . $(a + bi)^* = (a - bi)$
$ \psi\rangle$	Vector amb una etiqueta ψ . Conegut com a <i>ket</i>
$ \psi\rangle^T$	Transposada d'un vector $ \psi\rangle$
$ \psi\rangle^\dagger$	Conjugat Hermitià d'un vector. $ \psi\rangle^\dagger = (\psi\rangle^T)^*$
$\langle\psi $	Vector dual a $ \psi\rangle$. $ \psi\rangle = \langle\psi ^\dagger$ i $\langle\psi = \psi\rangle^\dagger$. Conegut com a <i>bra</i>
$\langle\varphi \psi\rangle$	Producte interior dels vectors $\langle\varphi $ i $ \psi\rangle$
$ \varphi\rangle\langle\psi $	Producte exterior dels vectors $ \varphi\rangle$ i $ \psi\rangle$
$ \psi\rangle \otimes \varphi\rangle$	Producte tensorial dels vectors $ \varphi\rangle$ i $ \psi\rangle$
0	Vector zero i operador zero
\mathbb{I}_n	Matriu identitat de dimensions $n \times n$
\mathbb{C}_n	Espai vectorial complex de dimensió n
\mathbb{C}_1 o \mathbb{C}	Espai dels nombres complexos

La notació utilitzada per un espai vectorial complex i l'espai dels nombres complex no són de la notació de Dirac estàndard, però les poso per explicar el que signifiquen.

Capítol C

Computació Quàntica vs Mecànica Quàntica

En la introducció havia mencionat que una de les raons per les quals havia començat a aprendre i recercar sobre computació quàntica era perquè era fàcil. En aquest annex explicaré exactament a què em refereixo per això amb un exemple pràctic. Dic que és fàcil, perquè ho és si es compara amb la mecànica quàntica.

En mecànica quàntica la manera més usual de representar els estats quàntics, com els orbitals d'un àtom d'hidrogen és a través de *wavefunctions* o funcions d'ona, no se solen representar mitjançant vectors d'estat. Aquestes funcions d'ona són molt útils i poden representar casos més generals que els vectors d'estat. No obstant això, treballar amb elles és molt més complicat, ja que són funcions que depenen del temps. Això implica que s'han d'utilitzar altres operacions per normalitzar les funcions, determinar com evolucionen en el temps o per fer mesures. A continuació faré una comparació entre com es preserva la norma d'una funció d'ona i la d'un vector d'estat quan aquests objectes evolucionen en el temps. Es podrà veure perquè penso que la mecànica quàntica més bàsica em sembla més complexa que la computació quàntica més bàsica.

C.1 Normalitzar

A causa de la interpretació probabilística dels vectors d'estat i de les funcions d'ona, aquests objectes han de ser normalitzats perquè la suma de les probabilitats dels possibles estats de mesura sigui 1.

Per una funció d'ona $\Psi(x, t)$ que representa una partícula, la probabilitat de trobar aquesta partícula en un punt x és $|\Psi(x, t)|^2$. Per tant, la funció d'ona ha de ser normalitzada seguint la fórmula:

$$\int_{-\infty}^{+\infty} |\Psi(x, t)|^2 dx = 1 \quad (\text{C.1})$$

Ja que la funció d'ona evoluciona a través del temps d'acord amb l'equació de Schrödinger (veure la figura C.1) qualsevol solució d'aquesta equació ha d'estar també normalitzada. En altres paraules, aquesta equació ha de preservar la normalització de les funcions d'ona [62].

$$i\hbar \frac{\partial \Psi}{\partial t} = -\frac{\hbar^2}{2m} \frac{\partial^2 \Psi}{\partial x^2} + V\Psi$$

FIGURA C.1: **Equació de Schrödinger.** On \hbar és $h/2\pi$, i V és una funció potencial d'energia.

Podem provar que aquesta equació preserva la fórmula C.1, començant per la igualtat trivial:

$$\frac{d}{dt} \int_{-\infty}^{+\infty} |\Psi(x, t)|^2 dx = \frac{\partial}{\partial t} \int_{-\infty}^{+\infty} |\Psi(x, t)|^2 dx$$

Per la regla del producte tenim que ¹:

$$\frac{\partial}{\partial t} |\Psi|^2 = \frac{\partial}{\partial t} (\Psi\Psi^*) = \Psi^* \frac{\partial \Psi}{\partial t} + \frac{\partial \Psi^*}{\partial t} \Psi$$

Ara amb l'equació de Schrödinger:

$$\frac{\partial \Psi}{\partial t} = \frac{i\hbar}{2m} \frac{\partial^2 \Psi}{\partial x^2} - \frac{i}{\hbar} V\Psi$$

¹A partir d'ara escriuré $\Psi(x, t)$ simplement com Ψ per no fer les equacions tan enrevessades.

A continuació calculem el conjugat complex de la derivada anterior:

$$\frac{\partial \Psi^*}{\partial t} = -\frac{i\hbar}{2m} \frac{\partial^2 \Psi^*}{\partial x^2} + \frac{i}{\hbar} V \Psi^*$$

Quan s'ha d'arregla d'aquesta forma:

$$\frac{\partial}{\partial t} |\Psi|^2 = \frac{i\hbar}{2m} \left(\Psi^* \frac{\partial^2 \Psi}{\partial x^2} - \frac{\partial^2 \Psi^*}{\partial x^2} \Psi \right) = \frac{\partial}{\partial x} \left[\frac{i\hbar}{2m} \left(\Psi^* \frac{\partial \Psi}{\partial x} - \frac{\partial \Psi^*}{\partial x} \Psi \right) \right],$$

finalment podem avaluar l'integral del principi:

$$\frac{d}{dt} \int_{-\infty}^{+\infty} |\Psi(x, t)|^2 dx = \frac{i\hbar}{2m} \left(\Psi^* \frac{\partial \Psi}{\partial x} - \frac{\partial \Psi^*}{\partial x} \Psi \right) \Big|_{-\infty}^{+\infty}$$

Ja que $\Psi(x, t)$ ha de convergir a zero quan x va cap a infinit, es veritat que:

$$\frac{d}{dt} \int_{-\infty}^{+\infty} |\Psi(x, t)|^2 dx = 0$$

Es pot veure que l'integral és constant, per tant, quan Ψ és normalitzada a $t = 0$, es queda d'aquesta manera per qualsevol t (positiu és clar), i la normalització d'una funció d'ona és constant si aquesta evoluciona d'acord amb l'equació de Schrödinger.

Aquesta demostració amb un vector d'estat és molt més simple: al evolucionar a partir de matrius unitàries, les quals preserven la norma del vector a la qual s'apliquen; si el vector al qual s'aplica una porta quàntica té una norma de 1, aquesta norma es preservarà.

Capítol D

Complexitat i algoritmes quàntics

En ciències de la computació existeix el concepte de *Big-O Notation*, una forma de quantificar l'eficiència dels algoritmes, la qual s'anomena la complexitat algorítmica. Bàsicament, és una forma de classificar-los segon la rapidesa que tenen en fer la tasca que els correspon, aquesta rapidesa no és mesura en segons, ja que aquesta mètrica pot variar d'ordinador a ordinador per les diferències en hardware que aquests poden tindre. En canvi, es mesura segons el nombre d'operacions que són necessàries, sense unitats.

La *Big-O Notation* consisteix en definir el nombre màxim d'operacions que necessita un algoritme, es denota com $O(\cdot)$ on l'argument usualment depèn de n que és la mida de l'input a l'algoritme, per exemple un algoritme de cerca ha de cercar a través de n coses.

Com a exemple: un algoritme de cerca de cadenes binàries corre en un temps $O(\log_2 n)$, on n és el nombre de cadenes entre les quals ha de cercar. La notació $O(\cdot)$ es diu que és el *upper boundary*, això significa que $\log_2 n$ és la quantitat de temps més gran en la qual es trobaria la cadena. Això vol dir que és possible que a la primera comprovació aconseguir trobar la cadena que es cerca per exemple.

Simplement, aquesta notació és una manera d'avaluar com d'eficients són els algoritmes amb relació a la mida de l'input que tenen. Cal notar que les operacions específiques que es quantifiquen varien entre els tipus d'algoritmes. Cada

grup d'algoritmes que tenen la mateixa tasca, tenen una operació a quantificar diferent.

Amb aquesta notació tenim una manera de comparar l'eficiència que tenen els algoritmes quàntics amb els clàssics que tenen la mateixa funció. Per il·lustrar l'avantatge en eficiència que presenten els algoritmes quàntics, presentaré a continuació els dos algoritmes més famosos, el de Grover, i el de Shor. Que serveixen per a la cerca d'un element en una llista desordenada, i per la factorització en nombres primers; respectivament.

D.1 Algoritme de Grover

En 1996, Lov Grover va presentar un algoritme quàntic per cercar entre una sèrie de dades desordenades [63] (e.g. cercar el número de telèfon en una llista desordenada). Per aquest problema un algoritme clàssic té una complexitat de $O(N)$ cerques¹, mentre que l'algoritme de Grover té una complexitat de $O(\sqrt{N})$, sent substancialment més eficient. En les paraules de Grover [63] (adaptades): un ordinador clàssic per tindre una probabilitat de $\frac{1}{2}$ de trobar un nombre de telèfon d'una persona en una llista desordenada necessita mirar a un mínim de $\frac{N}{2}$ números, mentre que amb el seu algoritme s'obté el número de telèfon en només $O(\sqrt{N})$ passos².

L'algoritme funciona de la següent manera:

1. S'agafa un sistema de n qubits en l'estat $|0\rangle$ que resulten en una combinació de $N = 2^n$ estats. S'aplica una distribució uniforme als qubits, és a dir, aplicar portes Hadamard a tots els qubits, tenint al final un estat $|s\rangle$:

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{2^n-1} |x\rangle$$

¹Una cerca completada és quan es verifica si un element de la llista és l'element que se cerca o quan no es troba aquest nombre al comprovar tots els elements de la llista

²Per passos entenc que es refereix al nombre de vegades que es mira a l'oracle, és a dir, el nombre que de vegades que es verifica si s'ha trobat l'element que se cerca.

2. Aplicar $\approx \frac{\pi}{4}\sqrt{N}$ vegades l'operador de Grover G , que consisteix en el següent conjunt d'instruccions:

- 1: Aplicar l'*Oracle* U_ω
- 2: Aplicar portes Hadamard a tots els qubits
- 3: Aplicar el *Grover diffusion operator* $U_s = 2|s\rangle\langle s| - I$
- 4: Aplicar una altre vegada les portes Hadamard

L'*oracle* és un tipus de funció que s'utilitza en els algorismes de cerca, té la finalitat de reconèixer si un element és l'element que s'està cercant. Els algorismes de cerca es construeixen al voltant de l'*oracle*, i diversos algorismes tenen diverses formes de consultar-lo. A més a més, la quantitat de consultes a l'*oracle* serveixen per quantificar la complexitat de l'algoritme, a més consultes, més ineficient és l'algoritme.

Al veure el procediment de l'algoritme es pot veure que l'*oracle* es consulta aproximadament $\frac{\pi}{4}\sqrt{N}$, vegades, d'aquesta manera resultat en una complexitat aproximada de $O(\sqrt{N})$, on $N = 2^n$ per n qubits, com ja he esmentat anteriorment.

No entraré més en profunditat sobre aquest algoritme, perquè si no hauré d'introduir més conceptes de computació quàntica com ara la fase d'un estat, o començar a utilitzar notació més complicada que també hauré d'explicar. Em sap greu, perquè és un algoritme que funciona d'una manera molt maca de veure.

També em sembla un bon algoritme per poder explicar una de les millors característiques de la computació quàntica: és possible descartar automàticament els estats dolents dintre d'una combinació d'estats, d'aquesta manera lliurant un estat concret desitjat d'entre la combinació. En el cas de l'algoritme de Grover aquest estat desitjat és l'estat que està cercant l'*oracle*.

D.2 Algoritme de Shor

No explicaré aquest algoritme en profunditat perquè és troba fora de l'àmbit d'aquest treball³. Em limitaré a explicar quina és la seva utilitat i el compararé amb

³Hauria d'introduir nous conceptes com la fase d'un qubit i com efectuar la transformada de Fourier inversa en un circuit quàntic, a més a més hauria d'explicar-lo en una notació bastant complexa. Però, es pot entendre com funciona sense cap problema veient l'entrada de Wikipedia sobre aquest algoritme.

les seves alternatives clàssiques.

L'algoritme de Shor per la factorització en nombres primers [64], és probablement l'algoritme quàntic més famós i més rellevant que hi ha. Això és a causa de la gran diferencia en complexitat que té en comparació als seus anàlegs clàssics, i a causa del gran impacte que tindria la seva implementació a gran escala. Aquest últim punt és el més interessant i cridaner, per tant, és el que desenvoluparé en major profunditat.

Actualment, totes les dades encriptades que s'envien a través d'internet són encriptades mitjançant la família de protocols RSA⁴. Aquests protocols són una mica complicats, no els explicaré. L'únic fet que hem de tenir en compte és que per desencryptar un missatge s'ha de trobar els factors primers d'un nombre n , és a dir, dos nombres primers p i q que multiplicats donin n . Amb aquests nombres es poden saber les claus públiques i privades per encriptar i desencryptar missatges. n seria la clau pública, un nombre que tothom pot conèixer i amb el qual es poden encriptar missatges. Mentre que a partir de p i q es pot trobar la clau privada que correspon a la clau pública. Amb aquesta clau es pot desencryptar un missatge encriptat amb n , per aquesta raó es mantenen els nombres p i q secrets. Es pot entendre la clau pública com un cadenat que només s'obre amb la clau privada. Cal mencionar que a cada clau privada correspon una clau pública, són paral·leles úniques.

En encriptació es busca el que es diu *one way operations*, és a dir, una operació que sigui fàcil d'efectuar, però que la seva inversa requereixi molts recursos computacionals per donar-la a terme. L'operació $n = pq$ compleix aquest requisit, és fàcil multiplicar els nombres primers p i q i trobar n , però és molt difícil trobar els nombres p i q sabent n , amb n sent un nombre de moltes xifres. Els millors algoritmes per donar a terme aquesta tasca requereixen moltíssim temps computacional en els ordinadors clàssics. Per aquesta raó el protocol RSA s'utilitza àmpliament, és molt segur perquè és molt difícil saber la clau privada a partir de n . Si n és un nombre de 2048 bits, amb la tecnologia actual no és possible

⁴RSA són les sigles dels creadors d'aquests protocols; Ron Rivest, Adi Shamir i Leonard Adleman, que en 1977 van publicar els resultats de la seva investigació [65].

trobar els nombres primers p i q en un temps raonable. Dic raonable perquè es pot arribar a descobrir, però es tardaria milers d'anys inclús amb els ordinadors més potents del planeta.

Ara és quan l'algoritme de Shor juga la seva part, com ja he dit s'utilitza per a la factorització de nombres primers, i és extremadament eficient en comparació als seus anàlegs clàssics. En termes de complexitat, aquest algoritme pot factoritzar un nombre de N dígit en un temps polinomi de $O(\log N)$ ⁵. Mentre que el millor algoritme clàssic per factoritzar en nombres primers té una complexitat de $O(e^{c(\log N)^{\frac{1}{3}}(\log \log N)^{\frac{2}{3}}})$ per una constant c . És a dir, que clàssicament es pot factoritzar en un temps exponencial [64], o com a mínim en un temps sub-exponencial⁶. De totes maneres només cal tenir en compte que l'algoritme de Shor és molt més eficient que el millor algoritme clàssic que dona a terme la mateixa tasca.

S'estima que amb un ordinador quàntic prou avançat es podrien arribar a factoritzar nombres els quals tenen una mida que actualment utilitzem⁷, però amb la tecnologia actual aquesta fita és impossible. El record pel nombre més gran factoritzat per un ordinador quàntic en l'actualitat el posseeix un equip de la Universitat de Bristol al Regne Unit. En 2021 van aconseguir factoritzar 21 amb un ordinador quàntic que funciona a partir d'un sistema fotònic [66].

D.3 Ordinadors quàntics actuals

El només ser capaços de poder factoritzar 21 il·lustra lo nova que és la tecnologia que està al darrere dels ordinadors quàntics. És un camp que té pocs anys de vida, i en el qual és difícil fer grans avanços. Al moment només es pot utilitzar el millor algoritme que ofereix la computació quàntica per poder fer una tasca extremadament simple⁸.

⁵Concretament la seva complexitat mesurada mitjançant la quantitat de portes quàntiques seria de $O((\log n)^2(\log \log n)(\log \log \log n))$, per factoritzar un nombre n amb una longitud de $\log n$ bits.

⁶Un algoritme que s'executa en temps sub-exponencial segueix sent més ineficient que un que s'executa en temps polinomi, però significativament més eficient que un que s'executa en temps exponencial.

⁷D'uns 2048 o 1024 dígit per exemple.

⁸ $3 \times 7 = 21$

Això és a causa de la petita quantitat de qubits dels quals disposen els ordinadors quàntics actuals i als errors que aquests donen a terme a l'hora d'aplicar operacions. Quan un ordinador quàntic aplica una rotació en l'estat d'un qubit físic, aquest ho fa amb una certa imprecisió. El mateix passa a l'hora d'aplicar portes a dos qubits, com per exemple una CNOT. Els ordinadors quàntics actuals només són utilitzats per fer experiments, com provar nous algoritmes i valorar els mateixos ordinadors.

Són extremadament poques les situacions pràctiques en les quals els ordinadors quàntics actuals siguin més útils que els clàssics. Tanmateix, és una tecnologia molt prometedora, en la qual grans companyies com Google, IBM, Amazon estan invertint per investigar-la, igual que altres companyies més petites i nombrosos equips d'investigació al llarg del món. S'espera que un futur els ordinadors quàntics ens permetran dur a terme tasques que són impossibles de realitzar en ordinadors clàssics, sobretot en tenir en compte que s'espera que començarem a arribar a limitacions físiques si continuem millorant els processadors d'ordinadors actuals.

No obstant això, aquesta afirmació s'ha de agafar amb cautela. És una tecnologia que s'està desenvolupant actualment, no és pot afirmar amb total seguretat que arribarà a un nivell en concret. És completament possible que al cap d'uns anys els investigadors es trobin amb una limitació física que és impossible de solucionar i que els ordinadors quàntics quedin rellevants com una tecnologia que va prometre molt i no va deliberar. L'única cosa que es pot fer és seguir investigant.

Però deixant tot això de banda, dels continguts d'aquest annex, el més rellevant són els algoritmes. Tant el seu funcionament, com la comparació que vaig entre els quàntics i els clàssics. Valorar el nivell tecnològic actual del ordinadors quàntics no és el focus del treball, i per tant, no tinc molta confiança en les afirmacions sobre aquest tema. Tanmateix, les he inclòs perquè penso que és una qüestió que s'ha d'adreçar. Encara que no he fer una recerca que fa justícia a aquesta qüestió.

Capítol E

Codi

En l'annex actual presentaré el codi que he utilitzat al llarg del treball. Està organitzat segons el moment en el qual he referenciat el codi en el text. Lamento no poder explicar el codi profundament, ja que no era pertinent. M'hauria agradat fer-ho.

He de mencionar que és possible que hi hagin errors en els espais o la indexació del codi presentant en aquestes pàgines del treball. Per una revisió exhaustiva del codi, consulteu [el repositori de Github](#).

Aquest projecte de [Python](#) el vaig començar a desenvolupar mitjançant una [Jupyter Notebook](#), a través de [Google Colab](#). Però quan el projecte va augmentar de mida i necessitava més eines (com un *debugger*¹), vaig decidir passar-me a un IDE (*Integrated Development Environment*), un programa que pose més recursos en les mans del programador. Vaig decidir-me per un IDE expressament dissenyat per a *Python*: [PyCharm](#). Concretament per *PyCharm Community*, que és d'ús gratuït. Per poder manejar els paquets de *Python* he utilitzat [Anaconda](#). Aquestes eines o programes en van facilitar molt la feina. Aquest *workflow* potser és una mica avançat, però he arribat a un punt en el qual sé com funciona tot i no penso canviar-lo en un futur tret que hagin millors opcions². Finalment he de

¹Un *debugger* és una eina que serveix per diagnosticar i arreglar problemes en el codi, és extremadament útil.

²Com per exemple [JetBrain Fleet](#), però ho dubto, que *Pycharm* m'encanta.

mentonar que tots els programes o recursos que he utilitzat en aquest treball són gratuïts/accessibles.

Els paquets que són necessaris per executar el codi estan especificats en el [repositori del treball](#). Tota la informació corresponent a aquest treball es pot trobar en el [meu repositori de Github](#). Des dels arxius de \LaTeX o *Python*, fins als canvis que vaig anar fent al llarg del treball.

E.1 Part I

E.1.1 Capítol 3

E.1.1.0.1 Regressió lineal Codi per efectuar una regressió lineal a dades que es generen a l'atzar en el mateix arxiu, l'utilitzo per poder generar la figura que il·lustra un exemple de regressió lineal. Aquest tros de codi l'he tret de GitHub³.

```
1  import numpy as np
2  from matplotlib import pyplot as plt
3  import matplotlib
4
5  font = {'family' : 'Helvetica',
6         'size'    : 18}
7
8  matplotlib.rc('font', **font)
9
10 # generate the data
11 np.random.seed(222)
12 X = np.random.normal(0,1, (200, 1))
13 w_target = np.random.normal(0,1, (1,1))
14 # data + white noise
15 y = X@w_target + np.random.normal(0, 1, (200,1))
16
17 # least squares
18 w_estimate = np.linalg.inv(X.T@X)@X.T@y
19 y_estimate = X@w_estimate
20
21 # plot the data
22 plt.figure(figsize=(15,10))
23 plt.scatter(X.flat, y_estimate.flat, label="Predicció")
24 plt.scatter(X.flat, y.flat, color='red', alpha=0.4, label="Dades")
25 plt.tight_layout()
26 plt.title("Regressio per diferencia de quadrats")
27 plt.legend()
28 plt.savefig("least_squares.png")
29 plt.show()
```

LISTING E.1: Regressió lineal

³Gist realitzat per l'usuari *jimimvp*: [enllaç](#)

E.2 Part II

E.2.1 Capítol 6

E.2.1.0.1 Codi original per la xarxa neuronal clàssica Està extret del [repo-sitori de GitHub de Micheal Nielsen](#), concretament del arxiu `network.py`.

```
1 """
2 network.py
3 ~~~~~
4 A module to implement the stochastic gradient descent learning
5 algorithm for a feedforward neural network. Gradients are calculated
6 using backpropagation. Note that I have focused on making the code
7 simple, easily readable, and easily modifiable. It is not optimized,
8 and omits many desirable features.
9 """
10
11 ##### Libraries
12 # Standard library
13 import random
14
15 # Third-party libraries
16 import numpy as np
17
18 class Network(object):
19
20     def __init__(self, sizes):
21         """The list 'sizes' contains the number of neurons in the
22         respective layers of the network. For example, if the list
23         was [2, 3, 1] then it would be a three-layer network, with the
24         first layer containing 2 neurons, the second layer 3 neurons,
25         and the third layer 1 neuron. The biases and weights for the
26         network are initialized randomly, using a Gaussian
27         distribution with mean 0, and variance 1. Note that the first
28         layer is assumed to be an input layer, and by convention we
29         won't set any biases for those neurons, since biases are only
30         ever used in computing the outputs from later layers."""
31         self.num_layers = len(sizes)
32         self.sizes = sizes
33         self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
34         self.weights = [np.random.randn(y, x) for x, y in zip(sizes[:-1], sizes
35         [1:])]
36
37     def feedforward(self, a):
```

```

37     """Return the output of the network if 'a' is input."""
38     for b, w in zip(self.biases, self.weights):
39         a = sigmoid(np.dot(w, a)+b)
40     return a
41
42 def SGD(self, training_data, epochs, mini_batch_size, eta, test_data=None):
43     """Train the neural network using mini-batch stochastic
44     gradient descent. The 'training_data' is a list of tuples
45     '(x, y)' representing the training inputs and the desired
46     outputs. The other non-optional parameters are
47     self-explanatory. If 'test_data' is provided then the
48     network will be evaluated against the test data after each
49     epoch, and partial progress printed out. This is useful for
50     tracking progress, but slows things down substantially."""
51     if test_data: n_test = len(test_data)
52     n = len(training_data)
53     for j in xrange(epochs):
54         random.shuffle(training_data)
55         mini_batches = [training_data[k:k+mini_batch_size] for k in xrange(0, n
, mini_batch_size)]
56         for mini_batch in mini_batches:
57             self.update_mini_batch(mini_batch, eta)
58         if test_data:
59             print "Epoch {0}: {1} / {2}".format(j, self.evaluate(test_data),
n_test)
60         else:
61             print "Epoch {0} complete".format(j)
62
63 def update_mini_batch(self, mini_batch, eta):
64     """Update the network's weights and biases by applying
65     gradient descent using backpropagation to a single mini batch.
66     The 'mini_batch' is a list of tuples '(x, y)', and 'eta'
67     is the learning rate."""
68     nabla_b = [np.zeros(b.shape) for b in self.biases]
69     nabla_w = [np.zeros(w.shape) for w in self.weights]
70     for x, y in mini_batch:
71         delta_nabla_b, delta_nabla_w = self.backprop(x, y)
72         nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
73         nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
74     self.weights = [w-(eta/len(mini_batch))*nw for w, nw in zip(self.weights,
nabla_w)]
75     self.biases = [b-(eta/len(mini_batch))*nb for b, nb in zip(self.biases,
nabla_b)]
76
77 def backprop(self, x, y):
78     """Return a tuple '(nabla_b, nabla_w)' representing the
79     gradient for the cost function C_x. 'nabla_b' and

```

```

80     'nabla_w' are layer-by-layer lists of numpy arrays, similar
81     to 'self.biases' and 'self.weights'."""
82
83     nabla_b = [np.zeros(b.shape) for b in self.biases]
84     nabla_w = [np.zeros(w.shape) for w in self.weights]
85     # feedforward
86     activation = x
87     activations = [x] # list to store all the activations, layer by layer
88     zs = [] # list to store all the z vectors, layer by layer
89     for b, w in zip(self.biases, self.weights):
90         z = np.dot(w, activation)+b
91         zs.append(z)
92         activation = sigmoid(z)
93         activations.append(activation)
94
95     # backward pass
96     delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1])
97     nabla_b[-1] = delta
98     nabla_w[-1] = np.dot(delta, activations[-2].transpose())
99     # Note that the variable l in the loop below is used a little
100    # differently to the notation in Chapter 2 of the book. Here,
101    # l = 1 means the last layer of neurons, l = 2 is the
102    # second-last layer, and so on. It's a renumbering of the
103    # scheme in the book, used here to take advantage of the fact
104    # that Python can use negative indices in lists.
105    for l in xrange(2, self.num_layers):
106        z = zs[-l]
107        sp = sigmoid_prime(z)
108        delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
109        nabla_b[-l] = delta
110        nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
111    return (nabla_b, nabla_w)
112
113    def evaluate(self, test_data):
114        """Return the number of test inputs for which the neural
115        network outputs the correct result. Note that the neural
116        network's output is assumed to be the index of whichever
117        neuron in the final layer has the highest activation."""
118        test_results = [(np.argmax(self.feedforward(x)), y) for (x, y) in test_data
119    ]
120
121        return sum(int(x == y) for (x, y) in test_results)
122
123    def cost_derivative(self, output_activations, y):
124        """Return the vector of partial derivatives \partial C_x /
125        \partial a for the output activations."""
126        return (output_activations-y)

```

```

126 ##### Miscellaneous functions
127 def sigmoid(z):
128     """The sigmoid function."""
129     return 1.0/(1.0+np.exp(-z))
130
131 def sigmoid_prime(z):
132     """Derivative of the sigmoid function."""
133     return sigmoid(z)*(1-sigmoid(z))

```

LISTING E.2: Codi original per la xarxa neuronal clàssica

E.2.1.0.2 Codi final per el discriminador Aquest codi es pot trobar al [repositori del treball](#) en l'arxiu `discriminator.py`.

```

1  """DISCRIMINATOR"""
2  import json
3  from typing import Dict, List
4
5  import numpy as np
6
7  from quantumGAN.functions import BCE_derivative, minimax_derivative_fake,
8      minimax_derivative_real, sigmoid, sigmoid_prime
9
10 def load(filename):
11     f = open(filename, "r")
12     data = json.load(f)
13     f.close()
14     # cost = getattr(sys.modules[__name__], data["cost"])
15     net = ClassicalDiscriminator_that_works(data["sizes"], data["loss"])
16     net.weights = [np.array(w) for w in data["weights"]]
17     net.biases = [np.array(b) for b in data["biases"]]
18     return net
19
20
21 class ClassicalDiscriminator:
22
23     def __init__(self,
24                 sizes: List[int],
25                 type_loss: str) -> None:
26
27         self.num_layers = len(sizes)
28         self.sizes = sizes
29         self.type_loss = type_loss
30         self.data_loss = {"real": [], "fake": []}
31         self.ret: Dict[str, any] = {"loss": [],

```

```

32         "label real": [],
33         "label fake": [],
34         "label fake time": [],
35         "label real time": []}
36
37     self.biases = [np.random.randn(y, ) for y in sizes[1:]]
38     self.weights = [np.random.randn(y, x) for x, y in zip(sizes[:-1], sizes
39 [1:])]
40
41 def feedforward(self, a):
42     """Return the output of the network if 'a' is input."""
43     for b, w in zip(self.biases, self.weights):
44         a = sigmoid(np.dot(w, a) + b)
45     return a
46
47 def predict(self, x):
48     # feedforward
49     activation = x
50     zs = [] # list to store all the z vectors, layer by layer
51     for b, w in zip(self.biases, self.weights):
52         z = np.dot(w, activation) + b
53         zs.append(z)
54         activation = sigmoid(z)
55     return activation
56
57 def evaluate(self, test_data):
58     test_results = [(np.argmax(self.feedforward(x)), y)
59 for (x, y) in test_data]
60     return sum(int(x == y) for (x, y) in test_results)
61
62 def forwardprop(self, x: np.ndarray):
63     activation = x
64     activations = [x] # list to store all the activations, layer by layer
65     zs = [] # list to store all the z vectors, layer by layer
66     for b, w in zip(self.biases, self.weights):
67         z = np.dot(w, activation) + b
68         zs.append(z)
69         activation = sigmoid(z)
70         activations.append(activation)
71     return activation, activations, zs
72
73 def backprop_bce(self, image, label):
74     """Return a tuple '(nabla_b, nabla_w)' representing the
75 gradient for the cost function C_x. 'nabla_b' and
76 'nabla_w' are layer-by-layer lists of numpy arrays, similar
77 to 'self.biases' and 'self.weights'."""
78     nabla_b = [np.zeros(b.shape) for b in self.biases]

```

```

78     nabla_w = [np.zeros(w.shape) for w in self.weights]
79
80     # feedforward and back error calculation depending on type of image
81     activation, activations, zs = self.forwardprop(image)
82     delta = BCE_derivative(activations[-1], label) * sigmoid_prime(zs[-1])
83
84     # backward pass
85     nabla_b[-1] = delta
86     nabla_w[-1] = np.dot(delta, activations[-2].reshape(1, activations[-2].
shape[0]))
87
88     for l in range(2, self.num_layers):
89         z = zs[-1]
90         delta = np.dot(self.weights[-l + 1].transpose(), delta) * sigmoid_prime
(z)
91         nabla_b[-1] = delta
92         nabla_w[-1] = np.dot(delta.reshape(delta.shape[0], 1), activations[-1 -
1].reshape(1, activations[-1 - 1].shape[0]))
93         return nabla_b, nabla_w, activations[-1]
94
95 def backprop_minimax(self, real_image, fake_image, is_real):
96     """Return a tuple '(nabla_b, nabla_w)' representing the
97     gradient for the cost function C_x. 'nabla_b' and
98     'nabla_w' are layer-by-layer lists of numpy arrays, similar
99     to 'self.biases' and 'self.weights'."""
100     nabla_b = [np.zeros(b.shape) for b in self.biases]
101     nabla_w = [np.zeros(w.shape) for w in self.weights]
102
103     # feedforward and back error calculation depending on type of image
104     activation_real, activations_real, zs_real = self.forwardprop(real_image)
105     activation_fake, activations_fake, zs_fake = self.forwardprop(fake_image)
106
107     if is_real:
108         delta = minimax_derivative_real(activations_real[-1]) * sigmoid_prime(
zs_real[-1])
109         activations, zs = activations_real, zs_real
110     else:
111         delta = minimax_derivative_fake(activations_fake[-1]) * sigmoid_prime(
zs_fake[-1])
112         activations, zs = activations_fake, zs_fake
113
114     # backward pass
115     nabla_b[-1] = delta
116     nabla_w[-1] = np.dot(delta, activations[-2].reshape(1, activations[-2].
shape[0]))
117
118     for l in range(2, self.num_layers):

```

```

119         z = zs[-1]
120         delta = np.dot(self.weights[-1 + 1].transpose(), delta) * sigmoid_prime
(z)
121         nabla_b[-1] = delta
122         nabla_w[-1] = np.dot(delta.reshape(delta.shape[0], 1),
123         activations[-1 - 1].reshape(1, activations[-1 - 1].shape[0]))
124         return nabla_b, nabla_w, activations[-1]
125
126     def train_mini_batch(self, mini_batch, learning_rate):
127         global label_real, label_fake
128         nabla_b = [np.zeros(b.shape) for b in self.biases]
129         nabla_w = [np.zeros(w.shape) for w in self.weights]
130
131         if self.type_loss == "binary cross entropy":
132             for real_image, fake_image in mini_batch:
133                 delta_nabla_b, delta_nabla_w, label_real = self.backprop_bce(
real_image, np.array([1.]))
134                 nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
135                 nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
136
137                 delta_nabla_b, delta_nabla_w, label_fake = self.backprop_bce(
fake_image, np.array([0.]))
138                 nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
139                 nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
140
141         elif self.type_loss == "minimax":
142             for real_image, fake_image in mini_batch:
143                 delta_nabla_b, delta_nabla_w, label_real = self.backprop_minimax(
real_image, fake_image, True)
144                 nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
145                 nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
146
147                 delta_nabla_b, delta_nabla_w, label_fake = self.backprop_minimax(
real_image, fake_image, False)
148                 nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
149                 nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
150         else:
151             raise Exception("type of loss function not valid")
152
153         # gradient descent
154         # nabla_w and nabla_b are multiplied by the learning rate
155         # and taken the mean of (dividing by the mini batch size)
156         self.weights = [w - (learning_rate / len(mini_batch)) * nw
157         for w, nw in zip(self.weights, nabla_w)]
158         self.biases = [b - (learning_rate / len(mini_batch)) * nb

```



```
159         for b, nb in zip(self.biases, nabla_b)]
```

LISTING E.3: Codi final pel discriminador

E.2.1.0.3 Codi per el generador Aquest codi es pot trobar al [repositori del treball](#) en l'arxiu `generador.py`.

```
1  """QUANTUM GENERATOR"""
2
3  from typing import Any, Dict, Optional, cast
4
5  import numpy as np
6  import qiskit
7  from qiskit import ClassicalRegister, QuantumRegister
8  from qiskit.circuit import QuantumCircuit
9  from qiskit.circuit.library import TwoLocal
10 from qiskit.providers.aer import AerSimulator
11
12 from quantumGAN.functions import create_entangler_map, create_real_keys,
    minimax_generator
13
14
15 class QuantumGenerator:
16
17     def __init__(
18         self,
19         shots: int,
20         num_qubits: int,
21         num_qubits_ancilla: int,
22         generator_circuit: Optional[QuantumCircuit] = None,
23         snapshot_dir: Optional[str] = None
24     ) -> None:
25
26         super().__init__()
27         # passar els arguments de la classe a metodes en de classes
28         # d'aquesta manera son accessibles per qualsevol funcio dintre de la classe
29         self.num_qubits_total = num_qubits
30         self.num_qubits_ancilla = num_qubits_ancilla
31         self.generator_circuit = generator_circuit
32         self.snapshot_dir = snapshot_dir
33         self.shots = shots
34         self.discriminator = None
35         self.ret: Dict[str, Any] = {"loss": []}
36         self.simulator = AerSimulator()
37
38     def init_parameters(self):
```

```

39     """Inicia els parametres inicial i crea el circuit al qual se li posen els
    parametres"""
40     # iniciacia dels parametres inicials i dels circuits al qual posar aquests
    parametres
41     self.generator_circuit = self.construct_circuit(latent_space_noise=None,
    to_measure=False)
42     # s'ha de crear primer el circuit porque d'aquesta manera es pot saber el
    nombre de parametres que es necessiten
43     self.parameter_values = np.random.normal(np.pi / 2, .1, self.
    generator_circuit.num_parameters)
44
45     def construct_circuit(self,
46                           latent_space_noise,
47                           to_measure: bool):
48         """Crea el circuit quantic des de zero a partir de diversos registres de
    qubits"""
49         if self.num_qubits_ancilla is 0:
50             qr = QuantumRegister(self.num_qubits_total, 'q')
51             cr = ClassicalRegister(self.num_qubits_total, 'c')
52             qc = QuantumCircuit(qr, cr)
53         else:
54             qr = QuantumRegister(self.num_qubits_total - self.num_qubits_ancilla, '
    q')
55             anc = QuantumRegister(self.num_qubits_ancilla, 'ancilla')
56             cr = ClassicalRegister(self.num_qubits_total - self.num_qubits_ancilla,
    'c')
57             qc = QuantumCircuit(anc, qr, cr)
58
59         # creacia de la part del circuit que conte la implantacio dels parametres d
    'input. En cas que no es donin aquests parametres es creen automaticament
60         if latent_space_noise is None:
61             randoms = np.random.normal(-np.pi * .01, np.pi * .01, self.
    num_qubits_total)
62             init_dist = qiskit.QuantumCircuit(self.num_qubits_total)
63
64             # es col.loca una porta RY en cada qubits i amb un parametre diferent
    cadascuna
65             for index in range(self.num_qubits_total):
66                 init_dist.ry(randoms[index], index)
67         else:
68             init_dist = qiskit.QuantumCircuit(self.num_qubits_total)
69
70             for index in range(self.num_qubits_total):
71                 init_dist.ry(latent_space_noise[index], index)
72
73         # la funcio create_entagler_map crea les parelles de qubits a les qual col.
    locar les portes CZ

```

```

74     # en funcio del nombre de qubits
75     if self.num_qubits_ancilla is 0:
76         entangler_map = create_entangler_map(self.num_qubits_total)
77     else:
78         entangler_map = create_entangler_map(self.num_qubits_total - self.
num_qubits_ancilla)
79
80     # creacio final dels circuits a partir una funcio integrada a Qiskit que va
repetint les operacions
81     # que se li especifiquen
82     ansatz = TwoLocal(int(self.num_qubits_total), 'ry', 'cz', entanglement=
entangler_map, reps=1, insert_barriers=True)
83
84     # aqui s'ajunten el circuit que funciona com a input amb el circuit que
consisteix en la repeticio
85     # de les portes RY i CZ
86     qc = qc.compose(init_dist, front=True)
87     qc = qc.compose(ansatz, front=False)
88
89     if to_measure:
90         qc.measure(qr, cr)
91
92     return qc
93
94 def set_discriminator(self, discriminator) -> None:
95     self.discriminator = discriminator
96
97 def get_output(
98     self,
99     latent_space_noise,
100     parameters: Optional[np.ndarray] = None
101 ):
102     """Retorna un output del generador quan se li dona un estat d'input i
opcionalment uns parametres en especific. Els pixels estan compostos per la
probabilitat que un qubit resulti en ket_0 en cada base. Per tant, els pixels
de l'imatge estan normalitzats amb la norma l-1."""
103     real_keys_set, real_keys_list = create_real_keys(self.num_qubits_total -
self.num_qubits_ancilla)
104
105     # en cas de que no es donin parametres com a input, es treuen els
parametres de la variable
106     # self.parameter_values. Es a dir els parametres que es creen
automaticament al principi i que es van
107     # actualitzant al mateix temps que el model s'optimitza
108     if parameters is None:
109         parameters = cast(np.ndarray, self.parameter_values)
110

```

```

111     qc = self.construct_circuit(latent_space_noise, True)
112
113     parameter_binds = {parameter_id: parameter_value for parameter_id,
114                         parameter_value in zip(qc.parameters, parameters)}
115
116     # el metode bind_parametres del circuit quantic
117     qc = qc.bind_parameters(parameter_binds)
118
119     # Simulacio dels circuits mitjançant el simulador Aer de Qiskit. El nivell
120     d'optimitzacio es zero, porque al ser circuits petits i que simulen una vegada,
121     no es necessari. Al optimitzar el proces acaba sent mes lent.
122     result_ideal = qiskit.execute(experiments=qc,
123                                   backend=self.simulator,
124                                   shots=self.shots,
125                                   optimization_level=0).result()
126
127     counts = result_ideal.get_counts()
128
129     try:
130         # creacio de l'imatge resultant
131         pixels = np.array([counts[index] for index in list(real_keys_list)])
132
133     except KeyError:
134         # aquesta excepcio sorgeix quan en el diccionari dels resultats no
135         estan totes les keys pel fet que qiskit, en cas de que no hi hagi un mesurament
136         en una base, no inclou aquesta base en el diccionari
137
138         keys = counts.keys()
139         missing_keys = real_keys_set.difference(keys)
140
141         # s'utilitza un la resta entre dos sets per poder veure quina es la key
142         que falta en el diccionari
143
144         for key_missing in missing_keys:
145             counts[key_missing] = 0
146
147         # una vegada es troba les keys que faltaven es crea l'imatge resultant
148         pixels = np.array([counts[index] for index in list(real_keys_list)])
149
150     pixels = pixels / self.shots
151
152     return pixels
153
154 def get_output_pixels(
155     self,
156     latent_space_noise,
157     params: Optional[np.ndarray] = None
158 ):
159     """Retorna un output del generador quan se li dona un estat d'input i
160     opcionalment uns parametres en especific. Cada pixel es la probabilitat de que
161     un qubits resulti en l'estat ket_0, per tant, els valors cada pixel (que son
162     independents entre si) es troba en l'interval (0, 1) """

```

```

149     qc = QuantumCircuit(self.num_qubits_total)
150
151     init_dist = qiskit.QuantumCircuit(self.num_qubits_total)
152     assert latent_space_noise.shape[0] == self.num_qubits_total
153
154     for num_qubit in range(self.num_qubits_total):
155         init_dist.ry(latent_space_noise[num_qubit], num_qubit)
156
157     if params is None:
158         params = cast(np.ndarray, self.parameter_values)
159
160     qc.assign_parameters(params)
161
162     # comptes de simular els valors que donara cada qubits, es simula l'estat
    final del circuit i d'aquest s'extreuen els valors que es mesuraran per a cada
    qubit
163     state_vector = qiskit.quantum_info.Statevector.from_instruction(qc)
164     pixels = []
165     for qubit in range(self.num_qubits_total):
166         # per treure la probabilitat s'utilitza una funcio implementada en
    Qiskit
167         pixels.append(state_vector.probabilities([qubit])[0])
168
169     # creacio de l'imatge resultada a partir de la llista que conte el valor
    per a cada pixel
170     generated_samples = np.array(pixels)
171     generated_samples.flatten()
172
173     return generated_samples
174
175 def train_mini_batch(self, mini_batch, learning_rate):
176     """Optimitzacio del generador per una batch d'imatges. Retorna una batch de
    les imatges generades amb unes imatges reals que poder donar com a input al
    generador. """
177     nabla_theta = np.zeros_like(self.parameter_values.shape)
178     new_images = []
179
180     for _, noise in mini_batch:
181         for index, _ in enumerate(self.parameter_values):
182             perturbation_vector = np.zeros_like(self.parameter_values)
183             perturbation_vector[index] = 1
184
185             # Creacio dels parametres per generar les dues imatges
186             pos_params = self.parameter_values + (np.pi / 4) *
    perturbation_vector
187             neg_params = self.parameter_values - (np.pi / 4) *
    perturbation_vector

```

```

188
189         pos_result = self.get_output(noise, parameters=pos_params) #
Generacio imatges
190         neg_result = self.get_output(noise, parameters=neg_params)
191
192         pos_result = self.discriminator.predict(pos_result) # Assignacio
de les etiquetes
193         neg_result = self.discriminator.predict(neg_result)
194
195         # Diferencia entre les avaluacions de la funcio de perdua entre les
dues etiquetes
196         gradient = minimax_generator(pos_result) - minimax_generator(
neg_result)
197         nabla_theta[index] += gradient # Afegir derivada d'un parametre al
gradient
198         new_images.append(self.get_output(noise))
199
200         for index, _ in enumerate(self.parameter_values):
201             # Actualitzacio dels parametres a traves del gradient
202             self.parameter_values[index] += (learning_rate / len(mini_batch)) *
nabla_theta[index]
203
204         # Creacio de la batch d'imatges a retornar
205         mini_batch = [(datapoint[0], fake_image) for datapoint, fake_image in zip(
mini_batch, new_images)]
206         return mini_batch

```

LISTING E.4: Codi final pel generador

E.2.1.0.4 Definició del model Aquest codi es pot trobar al [repositori del treball](#) en l'arxiu `qgan.py`.

```

1 import glob
2 import os
3 import json
4 import random
5 from datetime import datetime
6 from typing import List
7
8 import imageio
9 import matplotlib.pyplot as plt
10 import numpy as np
11
12 from quantumGAN.discriminator import ClassicalDiscriminator
13 from quantumGAN.functions import fechet_distance, minimax, images_to_distribution,
images_to_scatter

```

```

14 from quantumGAN.quantum_generator import QuantumGenerator
15
16
17 class Quantum_GAN:
18
19     def __init__(self,
20                 generator: QuantumGenerator,
21                 discriminator: ClassicalDiscriminator
22                 ):
23
24         self.last_batch = None
25         now = datetime.now()
26         init_time = now.strftime("%d_%m_%Y_%H_%M_%S")
27         self.path = "data/run{}".format(init_time)
28         self.path_images = self.path + "/images"
29         self.filename = "run.txt"
30
31         if not os.path.exists(self.path):
32             os.makedirs(self.path)
33
34         if not os.path.exists(self.path_images):
35             os.makedirs(self.path_images)
36
37         with open(os.path.join(self.path, self.filename), "w") as file:
38             file.write("RUN {} \n".format(init_time))
39             file.close()
40
41         self.generator = generator
42         self.discriminator = discriminator
43         self.loss_series, self.label_real_series, self.label_fake_series, self.
44         FD_score = [], [], [], []
45
46         self.generator.init_parameters()
47         self.example_g_circuit = self.generator.construct_circuit(
48             latent_space_noise=None,
49             to_measure=False)
50         self.generator.set_discriminator(self.discriminator)
51
52     def __repr__(self):
53         return "Discriminator Architecture: {} \n Generator Example Circuit: \n{}"
54         \
55         .format(self.discriminator.sizes, self.example_g_circuit)
56
57     def store_info(self, epoch, loss, real_label, fake_label):
58         file = open(os.path.join(self.path, self.filename), "a")
59         file.write("{} epoch LOSS {} Parameters {} REAL {} FAKE {} \n"
60                 .format(epoch,

```

```

58         loss,
59         self.generator.parameter_values,
60         real_label,
61         fake_label))
62     file.close()
63
64     def plot(self):
65         # save data for plotting
66         fake_images, real_images = [], []
67         for image_batch in self.last_batch:
68             fake_images.append(image_batch[1])
69             real_images.append(image_batch[0])
70
71         keys, average_result = images_to_distribution(fake_images)
72         print(average_result)
73         if self.generator.num_qubits_ancilla != 0:
74             plt.title(f"Distribucio d'una imatge generada \n(ancilla, epoch={self.
num_epochs})")
75         else:
76             plt.title(f"Distribucio d'una imatge generada \n(epoch={self.num_epochs
})")
77         plt.ylim(0., 1.)
78         plt.bar(keys, average_result)
79         plt.savefig(self.path + "/fake_distribution.png")
80         plt.clf()
81
82         keys_real, average_result_real = images_to_distribution(real_images)
83         print(average_result_real)
84         if self.generator.num_qubits_ancilla != 0:
85             plt.title(f"Distribucio d'una imatge real \n(ancilla, epoch={self.
num_epochs})")
86         else:
87             plt.title(f"Distribucio d'una imatge real \n(epoch={self.num_epochs})")
88         plt.ylim(0., 1.)
89         plt.bar(keys_real, average_result_real)
90         plt.savefig(self.path + "/real_distribution.png")
91         plt.clf()
92
93         y_axis, x_axis = images_to_scatter(fake_images)
94         y_axis_real, x_axis_real = images_to_scatter(real_images)
95         if self.generator.num_qubits_ancilla != 0:
96             plt.title(f"Comparacio entre les imatges reals con les imatges falses \
n(ancilla, epoch={self.num_epochs})")
97         else:
98             plt.title(f"Comparacio entre les imatges reals con les imatges falses \
n(epoch={self.num_epochs})")
99         plt.ylim(0. - .1, 1. + .1)

```



```

100     plt.scatter(y_axis, x_axis, label='Valors per les imatges falses',
101                 color='indigo',
102                 linewidth=.1,
103                 alpha=.2)
104     plt.scatter(y_axis_real, x_axis_real, label='Valors per les imatges reals',
105                 color='limegreen',
106                 linewidth=.1,
107                 alpha=.2)
108     plt.savefig(self.path + "/scatter_plot.png")
109     plt.clf()
110
111     t_steps = np.arange(self.num_epochs)
112     plt.figure(figsize=(6, 5))
113     if self.generator.num_qubits_ancilla != 0:
114         plt.title(f"Progres de la funcio de perdua \n(ancilla, epoch={self.
num_epochs})")
115     else:
116         plt.title(f"Progres de la funcio de perdua \n(epoch={self.num_epochs})"
)
117     plt.plot(t_steps, self.loss_series, label='Funcio de perdua del
discriminador', color='rebeccapurple', linewidth=2)
118     plt.grid()
119     plt.legend(loc='best')
120     plt.xlabel('Iteracions')
121     plt.ylabel('Funcio de perdua')
122     plt.savefig(self.path + "/loss_plot.png")
123     plt.clf()
124
125     t_steps = np.arange(self.num_epochs)
126     plt.figure(figsize=(6, 5))
127     if self.generator.num_qubits_ancilla != 0:
128         plt.title(f"Progres de les etiquetes \n(ancilla, epoch={self.num_epochs
})")
129     else:
130         plt.title(f"Progres de les etiquetes \n(epoch={self.num_epochs})")
131     plt.scatter(t_steps, self.label_real_series, label='Etiquetes per les
imatges reals',
132                 color='mediumvioletred',
133                 linewidth=.1)
134     plt.scatter(t_steps, self.label_fake_series, label='Etiquetes per les
imatges generades',
135                 color='rebeccapurple',
136                 linewidth=.1)
137     plt.grid()
138     plt.ylim(0. - .1, 1. + .1)
139     plt.legend(loc='best')
140     plt.xlabel('Iteracions')

```

```

141     plt.ylabel('Valor de les etiquetes')
142     plt.savefig(self.path + "/labels_plot.png")
143     plt.clf()
144
145     plt.figure(figsize=(6, 5))
146     if self.generator.num_qubits_ancilla != 0:
147         plt.title(f"Puntuacio de a partir de la distancia de Frechet \n(ancilla
148 , epoch={self.num_epochs})")
149     else:
150         plt.title(f"Puntuacio de a partir de la distancia de Frechet \n(epoch={
151 self.num_epochs})")
152     plt.plot(t_steps, self.FD_score, label='Valor de la puntuacio de Frechet',
153 color='rebeccapurple', linewidth=2)
154     plt.grid()
155     plt.legend(loc='best')
156     plt.ylim(0., .28)
157     plt.xlabel('Iteracions')
158     plt.ylabel('Valor de la puntuacio')
159     plt.savefig(self.path + "/FD_score.png")
160     plt.clf()
161
162 def train(self,
163         num_epochs: int,
164         training_data: List,
165         batch_size: int,
166         generator_learning_rate: float,
167         discriminator_learning_rate: float,
168         is_save_images: bool):
169
170     self.num_epochs = num_epochs
171     self.training_data = training_data
172     self.batch_size = batch_size
173     self.batch_size = batch_size
174     self.generator_lr = generator_learning_rate
175     self.discriminator_lr = discriminator_learning_rate
176     self.is_save_images = is_save_images
177
178     noise = self.training_data[0][1]
179     time_init = datetime.now()
180     for o in range(self.num_epochs):
181         mini_batches = create_mini_batches(self.training_data, self.batch_size)
182         output_fake = self.generator.get_output(latent_space_noise=mini_batches
183 [0][0][1], parameters=None)
184
185         for mini_batch in mini_batches:
186             self.last_batch = self.generator.train_mini_batch(mini_batch, self.
187 generator_lr)

```

```

183         self.discriminator.train_mini_batch(self.last_batch, self.
discriminator_lr)
184
185         output_real = mini_batches[0][0][0]
186         if is_save_images:
187             self.save_images(self.generator.get_output(latent_space_noise=noise
, parameters=None), o)
188
189             self.FD_score.append(fechet_distance(output_real, output_fake))
190             label_real, label_fake = self.discriminator.predict(output_real), self.
discriminator.predict(output_fake)
191             loss_final = 1 / 2 * (minimax(label_real, label_fake) + minimax(
label_real, label_fake))
192
193             self.loss_series.append(loss_final)
194             self.label_real_series.append(label_real)
195             self.label_fake_series.append(label_fake)
196
197             print("Epoch {}: Loss: {}".format(o, loss_final), output_real,
output_fake)
198             print(label_real[-1], label_fake[-1])
199             self.store_info(o, loss_final, label_real, label_fake)
200
201             time_now = datetime.now()
202             print((time_now - time_init).total_seconds(), "seconds")
203
204     def save_images(self, image, epoch):
205         image_shape = int(image.shape[0] / 2)
206         image = image.reshape(image_shape, image_shape)
207
208         plt.imshow(image, cmap='gray', vmax=1., vmin=0.)
209         plt.axis('off')
210         plt.savefig(self.path_images + '/image_at_epoch_{:04d}.png'.format(epoch))
211         plt.clf()
212
213     def create_gif(self):
214         anim_file = self.path + '/dcgan.gif'
215
216         with imageio.get_writer(anim_file, mode='I') as writer:
217             filenames = glob.glob(self.path_images + '/image*.png')
218             filenames = sorted(filenames)
219
220             for filename in filenames:
221                 image = imageio.imread(filename)
222                 writer.append_data(image)
223
224             image = imageio.imread(filename)

```

```

225         writer.append_data(image)
226
227     def save(self):
228         """Save the neural network to the file 'filename'."""
229         data = {"batch_size": self.batch_size,
230                 "D_sizes": self.discriminator.sizes,
231                 "D_weights": [w.tolist() for w in self.discriminator.weights],
232                 "D_biases": [b.tolist() for b in self.discriminator.biases],
233                 "D_loss": self.discriminator.type_loss,
234                 "Q_parameters": [theta for theta in self.generator.parameter_values
235 ],
236                 "Q_shots": self.generator.shots,
237                 "Q_num_qubits": self.generator.num_qubits_total,
238                 "Q_num_qubits_ancilla": self.generator.num_qubits_ancilla,
239                 "real_labels": self.label_real_series,
240                 "fake_labels": self.label_fake_series,
241                 "loss_series": self.loss_series
242                 }
243         f = open(os.path.join(self.path, "data.txt"), "a")
244         json.dump(data, f)
245         f.close()
246
247     def create_mini_batches(training_data, mini_batch_size):
248         n = len(training_data)
249         random.shuffle(training_data)
250         mini_batches = [
251             training_data[k:k + mini_batch_size]
252             for k in range(0, n, mini_batch_size)]
253         return [mini_batches[0]]
254
255
256     def load_gan(filename):
257         f = open(filename, "r")
258         data = json.load(f)
259         f.close()
260         discriminator = ClassicalDiscriminator(data["D_sizes"], data["D_loss"])
261
262         generator = QuantumGenerator(num_qubits=data["Q_num_qubits"],
263                                     generator_circuit=None,
264                                     num_qubits_ancilla=data["Q_num_qubits_ancilla"]
265 ],
266                                     shots=data["Q_shots"])
267
268         quantum_gan = Quantum_GAN(generator, discriminator)

```

```

269     quantum_gan.discriminator.weights = [np.array(w) for w in data["D_weights"
270 ]]
271     quantum_gan.discriminator.biases = [np.array(b) for b in data["D_biases"]]
272     quantum_gan.generator.parameter_values = np.array(data["Q_parameters"])
273
274     return quantum_gan, data["batch_size"]
275

```

LISTING E.5: Codi per la definició del model

E.2.1.0.5 Execució del model Aquest codi es pot trobar al [repositori del treball](#) en l'arxiu `main.py`.

```

1  import numpy as np
2
3  from quantumGAN.discriminator import ClassicalDiscriminator
4  from quantumGAN.qgan import Quantum_GAN
5  from quantumGAN.quantum_generator import QuantumGenerator
6
7  num_qubits: int = 3
8
9  # Set number of training epochs
10 num_epochs = 150
11 # Batch size
12 batch_size = 10
13
14 train_data = []
15 for _ in range(800):
16     x2 = np.random.uniform(.55, .46, (2,))
17     fake_datapoint = np.random.uniform(-np.pi * .01, np.pi * .01, (num_qubits,))
18     real_datapoint = np.array([x2[0], 0, x2[0], 0])
19     train_data.append((real_datapoint, fake_datapoint))
20
21 discriminator = ClassicalDiscriminator(sizes=[4, 16, 8, 1],
22                                       type_loss="minimax")
23 generator = QuantumGenerator(num_qubits=num_qubits,
24                              generator_circuit=None,
25                              num_qubits_ancilla=1,
26                              shots=4096)
27
28 quantum_gan = Quantum_GAN(generator, discriminator)
29 print(quantum_gan)
30 print(num_epochs)
31 quantum_gan.generator.get_output(fake_datapoint[0], None)
32 quantum_gan.train(num_epochs, train_data, batch_size, .1, .1, True)

```

```

33
34 quantum_gan.plot()
35 quantum_gan.create_gif()
36 quantum_gan.save()

```

LISTING E.6: Codi final pel generador

E.2.1.0.6 Funcions extres Hi han funcions en el codi que utilitzo en diversos arxius, aquestes les defineixo en l'arxiu `functions.py`, que es pot trobar al [repo-sitori del treball](#). En aquest arxiu també hi ha una funció que dona a terme una traça parcial a una matriu donada, malgrat que no la he arribat a utilitzar.

```

1 import itertools
2 import math
3
4 import numpy as np
5 from matplotlib import pyplot as plt
6
7
8 # DATA PROCESSING
9 from scipy import linalg
10
11
12 def save_images_color(image, epoch):
13     plt.imshow(image.reshape(int(image.shape[0] / 3), 1, 3))
14     plt.axis('off')
15     plt.savefig('images/image_at_epoch_{:04d}.png'.format(epoch))
16
17
18 def create_real_keys(num_qubits):
19     lst = [[str(a) for a in i] for i in itertools.product([0, 1], repeat=num_qubits)]
20     new_lst = []
21     for element in lst:
22         word = str()
23         for number in element:
24             word = word + number
25         new_lst.append(word)
26     return set(new_lst), new_lst
27
28
29 def create_entangler_map(num_qubits: int):
30     lst = [list(i) for i in itertools.combinations(range(num_qubits), 2)]
31     index = 0
32     entangler_map = []

```

```

33     for i in reversed(range(num_qubits)):
34         try:
35             entangler_map.append(lst[index])
36             index += i
37         except IndexError:
38             return entangler_map
39
40
41 def images_to_distribution(batch_image):
42     num_images = len(batch_image)
43     sum_result = 0
44     for image in batch_image:
45         sum_result += image
46     average_result = sum_result / num_images
47     keys = create_real_keys(int(math.sqrt(batch_image[0].shape[0]))) [1]
48     return keys, average_result
49
50
51 def images_to_scatter(batch_image):
52     keys = create_real_keys(int(math.sqrt(batch_image[0].shape[0]))) [1]
53     x_axis = []
54     y_axis = []
55
56     for image in batch_image:
57         pixel_count = 0
58         for pixel in image:
59             x_axis.append(pixel)
60             y_axis.append(keys[pixel_count])
61             pixel_count += 1
62
63     return y_axis, x_axis
64
65 def fechet_distance(image1: np.array,
66                     image2: np.array):
67     assert image1.shape == image2.shape
68     y = np.arange(0, image1.flatten().shape[0])
69
70     matrix_a_cov = np.cov(np.stack((y, image1.flatten()), axis=0))
71     matrix_b_cov = np.cov(np.stack((y, image2.flatten()), axis=0))
72
73     to_trace = matrix_a_cov + matrix_b_cov - 2*(linalg.fractional_matrix_power(np.
74     dot(matrix_a_cov, matrix_b_cov), .5))
75     return np.abs(image1.mean() - image2.mean())**2 + np.trace(to_trace)
76
77 # ACTIVATION FUNCTIONS
78

```

```

79 def sigmoid(z):
80     """The sigmoid function."""
81     return 1.0 / (1.0 + np.exp(-z))
82
83
84 def sigmoid_prime(z):
85     """Derivative of the sigmoid function."""
86     return sigmoid(z) * (1 - sigmoid(z))
87
88
89 def relu(z):
90     return np.maximum(0, z)
91
92
93 def relu_prime(z):
94     z[z <= 0] = 0
95     z[z > 0] = 1
96     return z
97
98
99 # LOSSES
100 def MSE_derivative(prediction, y):
101     return 2 * (y - prediction)
102
103
104 def MSE(prediction, y):
105     return (y - prediction) ** 2
106
107
108 def BCE_derivative(prediction, target):
109     # return prediction - target
110     return -target / prediction + (1 - target) / (1 - prediction)
111
112
113 def BCE(predictions: np.ndarray, targets: np.ndarray) -> np.ndarray:
114     return targets * np.log(predictions) + (1 - targets) * np.log(1 - predictions).
115     mean()
116
117
118 def minimax_derivative_real(real_prediction):
119     real_prediction = np.array(real_prediction)
120     return np.nan_to_num((-1) * (1 / real_prediction))
121
122
123 def minimax_derivative_fake(fake_prediction):
124     fake_prediction = np.array(fake_prediction)
125     return np.nan_to_num(1 / (1 - fake_prediction))

```



```

125
126
127 def minimax(real_prediction, fake_prediction):
128     return np.nan_to_num(np.log(real_prediction) + np.log(1 - fake_prediction))
129
130
131 def minimax_generator(prediction_fake):
132     return (-1) * np.log(1 - prediction_fake)
133
134
135 # QUANTUM FUNCTIONS
136
137 class Partial_Trace:
138     def __init__(self, state: np.array, qubits_out: int, side: str):
139
140         self.state = state
141         self.qubits_out = qubits_out
142         self.side = side
143
144         if self.state.ndim == 1:
145             self.state = np.outer(self.state, self.state)
146
147         self.total_dim = self.state.shape[0]
148
149         self.num_qubits = int(np.log2(self.total_dim))
150         self.a_dim = 2 ** (self.num_qubits - self.qubits_out)
151         self.b_dim = 2 ** self.qubits_out
152
153         # if self.side == "bot":
154         self.basis_b = [_ for _ in np.identity(int(self.b_dim))]
155         self.basis_a = [_ for _ in np.identity(int(self.a_dim))]
156
157         # elif self.side == "top":
158         #     self.basis_a = [_ for _ in np.identity(int(self.b_dim))]
159         #     self.basis_b = [_ for _ in np.identity(int(self.a_dim))]
160         #
161         # else:
162         #     raise NameError("invalid side argument, enter \"bot\" or \"top\"")
163         print(self.basis_a, self.basis_b)
164
165     def get_entry(self, index_i, index_j):
166         sigma = 0
167
168         if self.side == "bot":
169             for k in range(self.qubits_out + 1):
170                 ab_l = np.kron(self.basis_a[index_i],
171                               self.basis_b[k])

```

```

172         ab_r = np.kron(self.basis_a[index_j],
173                        self.basis_b[k])
174
175         print(ab_r, ab_l)
176
177         right_side = np.dot(self.state, ab_r)
178         sigma += np.inner(ab_l, right_side)
179
180     if self.side == "top":
181         for k in range(self.qubits_out + 1):
182             ba_l = np.kron(self.basis_b[index_i],
183                           self.basis_a[k])
184             ba_r = np.kron(self.basis_b[index_j],
185                           self.basis_a[k])
186
187             print(ba_r, ba_l)
188
189             right_side = np.dot(self.state, ba_r)
190             sigma += np.inner(ba_l, right_side)
191
192     return sigma
193
194     def compute_matrix(self):
195         a = [_ for _ in range(self.a_dim)]
196         b = [_ for _ in range(self.a_dim)]
197
198         entries_pre = [(x, y) for x in a for y in b]
199         entries = []
200
201         for i_index, j_index in entries_pre:
202             entries.append(self.get_entry(i_index, j_index))
203
204         entries = np.array(entries)
205         return entries.reshape(self.a_dim, self.a_dim)

```

LISTING E.7: Funcions varies

E.2.2 Capítol 7

E.2.2.0.1 Multiprocessament Per poder ser més eficient a l'hora d'executar els models, vaig decidir utilitzar una característica molt útil de Python, el *multiprocessing*. Usualment, una instància de Python, és a dir; un arxiu executant-se, no més es fa servir un nucli del processador a la vegada, però amb *multiprocessing*

pots emprar múltiples nuclis amb un mateix arxiu, d'aquesta manera executant diverses línies de codi al mateix temps. Per poder executar diversos models a la mateixa vegada i d'una forma simple, vaig fer servir aquest mòdul de Python per executar els models dels quals he agafat les dades que he presentat en aquest treball. Aquest codi es pot trobar al [repositori del treball](#) en l'arxiu `test_multi.py`.

```
1  import multiprocessing
2  import time
3  import numpy as np
4
5  from quantumGAN.discriminator import ClassicalDiscriminator
6  from quantumGAN.qgan import Quantum_GAN
7  from quantumGAN.quantum_generator import QuantumGenerator
8
9  batch_size = 10
10
11  train_data = []
12  # generacio de les dades
13  for _ in range(800):
14      x2 = np.random.uniform(.55, .46, (2,))
15      fake_datapoint = np.random.uniform(-np.pi * .01, np.pi * .01, (4,))
16      real_datapoint = np.array([x2[0], 0, x2[0], 0])
17      train_data.append((real_datapoint, fake_datapoint))
18  list_epochs = [550, 550, 550]
19
20  example_generator_ancilla = QuantumGenerator(num_qubits=4,
21                                              generator_circuit=None,
22                                              num_qubits_ancilla=2,
23                                              shots=4096)
24  example_generator = QuantumGenerator(num_qubits=2,
25                                      generator_circuit=None,
26                                      num_qubits_ancilla=0,
27                                      shots=4096)
28
29  # generem els parametres inicials d'acord amb el circuit amb un mayor nombre de
30  # parametres
31  circuit_ancilla = example_generator_ancilla.construct_circuit(None, False)
32  circuit_not_ancilla = example_generator_ancilla.construct_circuit(None, False)
33
34  init_parameters_ancilla = np.random.normal(np.pi / 2, .1, circuit_ancilla.
35  num_parameters)
36
37  # canviem les dimensions dels parametres perquè coincideixin amb les dimensions
38  # necessaries pel circuit
39  # sense qubits ancilla
40  not_ancilla_list = init_parameters_ancilla.tolist()
```

```

38 circuit_ancilla.num_parameters - circuit_not_ancilla.num_parameters:]
39 init_parameters_not_ancilla = np.array(not_ancilla_list)
40
41 example_discriminator = discriminator_ancilla = \
42     ClassicalDiscriminator(sizes=[4, 16, 8, 1], type_loss="minimax")
43
44 init_biases_discriminator, init_weights_discriminator = example_discriminator.
45 init_parameters()
46
47 def to_train(num_epochs):
48     # definicio d'un discriminador pel model amb la funcio no-lineal i una
49     # altre pel model que no la te
50     discriminator_ancilla = \
51         ClassicalDiscriminator(sizes=[4, 16, 8, 1], type_loss="minimax")
52     discriminator_not_ancilla = \
53         ClassicalDiscriminator(sizes=[4, 16, 8, 1], type_loss="minimax")
54
55     # es defineix un generador que te la funcio no-lineal integrada en el
56     # circuit
57     generator_ancilla = \
58         QuantumGenerator(num_qubits=4,
59                           generator_circuit=None,
60                           num_qubits_ancilla=2, # IMPORTANT
61                           shots=4096)
62
63     # i un altre generador que no te la funcio no-lineal
64     generator_not_ancilla = \
65         QuantumGenerator(num_qubits=2,
66                           generator_circuit=None,
67                           num_qubits_ancilla=0, # IMPORTANT
68                           shots=4096)
69
70     quantum_gan_ancilla = \
71         QuantumGAN(generator_ancilla, discriminator_ancilla)
72     time.sleep(1)
73     quantum_gan_not_ancilla = \
74         QuantumGAN(generator_not_ancilla, discriminator_not_ancilla)
75
76     print(quantum_gan_not_ancilla.path)
77     print(quantum_gan_ancilla.path)
78
79     # abans de comencar a entrenar el model s'ha de canviar els parametres als
80     # definits anteriorment
81     quantum_gan_ancilla.generator.parameter_values = init_parameters_ancilla
82     quantum_gan_ancilla.discriminator.weights = init_weights_discriminator
83     quantum_gan_ancilla.discriminator.biases = init_biases_discriminator
84     quantum_gan_ancilla.train(num_epochs, train_data, batch_size, .1, .1, False
85 )

```

```

80
81     quantum_gan_not_ancilla.generator.parameter_values =
init_parameters_not_ancilla
82     quantum_gan_not_ancilla.discriminator.weights = init_weights_discriminator
83     quantum_gan_not_ancilla.discriminator.biases = init_biases_discriminator
84     quantum_gan_not_ancilla.train(num_epochs, train_data, batch_size, .1, .1,
False)
85
86     # una vegada ha acabat l'optimitzacio es creen els grafics per compara l'
eficiencia del models
87     quantum_gan_not_ancilla.plot()
88     quantum_gan_ancilla.plot()
89
90
91 def main():
92     # Crear una queue para compartir les dades entre els processos
93     # Crear i iniciar el proces de simulacio
94     jobs = []
95     for epoch in list_epochs:
96         simulate = multiprocessing.Process(None, to_train, args=(epoch,))
97         jobs.append(simulate)
98         time.sleep(2)
99         simulate.start()
100
101
102 if __name__ == '__main__':
103     main()

```

LISTING E.8: Executar els models amb multiprocessing

Bibliografia

- [1] H.-L. Huang, Y. Du, M. Gong, Y. Zhao, Y. Wu, C. Wang, S. Li, F. Liang, J. Lin, Y. Xu, and et al., “Experimental quantum generative adversarial networks for image generation,” *Physical Review Applied*, vol. 16, no. 2, 2021. [Online; accedit en 19/09/2021: [Enllaç](#)].
- [2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 2014. [Online; accedit en 19/09/2021: [Enllaç](#)].
- [3] M. Schuld, I. Sinayskiy, and F. Petruccione, “The quest for a quantum neural network,” *Quantum Information Processing*, vol. 13, p. 2567–2586, 2014. [Online; accedit en 19/09/2021: [Enllaç](#)].
- [4] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa, E. Peters, O. Lockwood, A. Skolik, S. Jerbi, V. Dunjko, M. Leib, M. Streif, D. V. Dollen, H. Chen, S. Cao, R. Wiersema, H.-Y. Huang, J. R. McClean, R. Babbush, S. Boixo, D. Bacon, A. K. Ho, H. Neven, and M. Mohseni, “TensorFlow Quantum: A software framework for quantum machine learning,” 2021. [Online; accedit en 19/09/2021: [Enllaç](#)].
- [5] G. Strang, “Linear Algebra MIT OCW,” 2011. [Online; accedit en 19/09/2021: [Enllaç](#)].
- [6] G. Strang, “Matrix methods in data analysis, signal processing, and machine learning mit ocw,” 2018. [Online; accedit en 19/09/2021: [Enllaç](#)].

- [7] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 10 ed., 2010.
- [8] E. Rieffel and W. Polak, *Quantum Computing: A Gentle Introduction*. MIT Press, 1 ed., 2011.
- [9] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, p. 66. Cambridge University Press, 10 ed., 2010.
- [10] S. Axler, *Linear Algebra Done Right*, ch. 3, pp. 37–38. Springer, 2 ed., 1997.
- [11] S. Axler, *Linear Algebra Done Right*, ch. 3, pp. 48–52. Springer, 2 ed., 1997.
- [12] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, pp. 65–68. Cambridge University Press, 10 ed., 2010.
- [13] S. Axler, *Linear Algebra Done Right*, ch. 6, pp. 102–112. Springer, 2 ed., 1997.
- [14] W. MathWorld, “L2-Norm,” 2021. [Online; accedit en 19/09/2021: [Enllaç](#)].
- [15] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 1, pp. 2–36. Cambridge University Press, 10 ed., 2010.
- [16] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, p. 74. Cambridge University Press, 10 ed., 2010.
- [17] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, p. 73. Cambridge University Press, 10 ed., 2010.
- [18] Wikipedia, “Tensor product,” 2021. [Online; accedit en 19/09/2021: [Enllaç](#)].
- [19] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, pp. 75–76. Cambridge University Press, 10 ed., 2010.
- [20] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, pp. 80–96. Cambridge University Press, 10 ed., 2010.

- [21] A. Peres, *Quantum Theory: Concepts and Methods*, ch. 2, pp. 24–29. Kluwer Academic Publishers, 2002.
- [22] E. Rieffel and W. Polak, *Quantum Computing: A Gentle Introduction*, ch. 2, pp. 12–13. MIT Press, 1 ed., 2011.
- [23] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 1, pp. 13–15. Cambridge University Press, 10 ed., 2010.
- [24] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 1, pp. 16–17. Cambridge University Press, 10 ed., 2010.
- [25] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 1, p. 15. Cambridge University Press, 10 ed., 2010.
- [26] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 4, p. 174. Cambridge University Press, 10 ed., 2010.
- [27] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, pp. 95–96. Cambridge University Press, 10 ed., 2010.
- [28] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 4, pp. 178–181. Cambridge University Press, 10 ed., 2010.
- [29] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, pp. 84–86. Cambridge University Press, 10 ed., 2010.
- [30] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, p. 99. Cambridge University Press, 10 ed., 2010.
- [31] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, p. 105. Cambridge University Press, 10 ed., 2010.
- [32] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, pp. 105–106. Cambridge University Press, 10 ed., 2010.
- [33] E. Rieffel and W. Polak, *Quantum Computing: A Gentle Introduction*, ch. 10, pp. 212–213. MIT Press, 1 ed., 2011.

- [34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
<http://www.deeplearningbook.org>.
- [35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ch. 6, pp. 168–170. MIT Press, 2016.
- [36] C. de [ml-glossary](#), “Loss fuctions.” [Online; accedit en 09/01/2022: [Enllaç](#)].
- [37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ch. 5, pp. 130–133. MIT Press, 2016.
- [38] “Backpropagation calculus | chapter 4, deep learning.” [Online; accedit en 09/01/2022: [Enllaç](#)].
- [39] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017. [Online; accedit en 26/10/2021: [Enllaç](#)].
- [40] T. Developers, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software disponible a [tensorflow.org](https://www.tensorflow.org).
- [41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.
- [42] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, “Analyzing and improving the image quality of stylegan,” 2020. [Online; accedit en 26/10/2021: [Enllaç](#)].
- [43] lucidrains, “This person does not exist,” 2021. [Online; accedit en 26/10/2021: [Enllaç](#)].
- [44] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, “Supervised learning with quantum-enhanced feature spaces,” *Nature*, vol. 567, no. 7747, p. 209–212, 2019.

- [45] M. Schuld and N. Killoran, “Quantum machine learning in feature hilbert spaces,” *Physical Review Letters*, vol. 122, no. 4, 2019.
- [46] M. Schuld, I. Sinayskiy, and F. Petruccione, “Prediction by linear regression on a quantum computer,” *Physical Review A*, vol. 94, no. 2, 2016.
- [47] A. W. Harrow and J. C. Napp, “Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms,” *Physical Review Letters*, vol. 126, Apr 2021.
- [48] Y. Cao, G. G. Guerreschi, and A. Aspuru-Guzik, “Quantum neuron: an elementary building block for machine learning on quantum computers,” 2017. [Online; accedit en 6/11/2021: [Enllaç](#)].
- [49] J. Romero and A. Aspuru-Guzik, “Variational quantum generators: Generative adversarial quantum machine learning for continuous distributions,” 2019. [Online; accedit en 6/11/2021: [Enllaç](#)].
- [50] I. Cong, S. Choi, and M. D. Lukin, “Quantum convolutional neural networks,” *Nature Physics*, vol. 15, p. 1273–1278, Aug 2019.
- [51] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.
- [52] J. Liu, K. H. Lim, K. L. Wood, W. Huang, C. Guo, and H.-L. Huang, “Hybrid quantum-classical convolutional neural networks,” *Science China Physics, Mechanics and Astronomy*, vol. 64, aug 2021.
- [53] C. Zoufal, A. Lucchi, and S. Woerner, “Quantum generative adversarial networks for learning and loading random distributions,” *npj Quantum Information*, vol. 5, no. 103, 2019. [Online; accedit en 01/11/2021: [Enllaç](#)].
- [54] C. Developers, “Cirq,” Aug. 2021. Veure la llista completa d’autors a Github: <https://github.com/quantumlib/Cirq/graphs/contributors>.

- [55] Q. Developers, “Qiskit: An open-source framework for quantum computing,” 2021.
- [56] S. Semeniuta, A. Severyn, and S. Gelly, “On accurate evaluation of gans for language generation,” 2019. [Online; accedit en 19/09/2021: [Enllaç](#)].
- [57] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, “Jukebox: A generative model for music,” 2020.
- [58] J. Romero, J. P. Olson, and A. Aspuru-Guzik, “Quantum autoencoders for efficient compression of quantum data,” *Quantum Science and Technology*, vol. 2, p. 045001, Aug 2017. [Online; accedit en 19/09/2021: [Enllaç](#)].
- [59] B. Sun, P. Q. Le, A. M. Ilyasu, F. Yan, J. A. Garcia, F. Dong, and K. Hirota, “A multi-channel representation for images on quantum computers using the rgb color space,” pp. 1–6, 2011.
- [60] Wikipedia, “Chroma subsampling,” 2022. [Online; accedit en 13/04/2022: [Enllaç](#)].
- [61] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, ch. 2, p. 62. Cambridge University Press, 10 ed., 2010.
- [62] D. J. Griffiths, *Introduction to Quantum Mechanics*, ch. 1, pp. 11–12. Prentice Hall, 1995.
- [63] L. K. Grover, “A fast quantum mechanical algorithm for database search,” *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996. [Online; accedit en 05/10/2021: [Enllaç](#)].
- [64] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM Journal on Computing*, vol. 26, p. 1484–1509, Oct 1997. [Online; accedit en 02/01/2022: [Enllaç](#)].
- [65] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, p. 120–126, 1978.

- [66] E. Martín-López, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O'Brien, "Experimental realization of shor's quantum factoring algorithm using qubit recycling," *Nature Photonics*, vol. 6, no. 11, p. 773–776, 2012. [Online; accedit en 06/01/2022: [Enllaç](#)].