Generant Imatges amb un Ordinador Quàntic

TREBALL DE RECERCA DE BATXILLERAT

IES MIQUEL TARRADELL

Autor:

Tomàs Ockier Poblet
2nd Batxillerat
ockier1@gmail.com

Tutor:

Tomàs Ockier Poblet

Institut Miquel Tarradell

4 de desembre de 2021
Barcelona, Barcelona

Índex

Introducció

Desde hace más de un año, me he dedicado a estudiar computación cuántica durante mi tiempo libre. Buscaba investigar un campo relacionado con la mecánica cuántica, pero sin que sea muy complicado, que se pueda entender a un nivel teórico y que me entusiasme.

La Computación Cuántica encaja perfectamente con esos criterios. Es más sencilla que la mecánica cuántica debido a que no está basada en cálculo o ecuaciones diferenciales, se basa en la álgebra lineal, utilizando valores discretos, vectores y matrices. Además si se trabaja a un nivel teórico sencillo, no se tienen en consideración las interpretaciones físicas, lo cual simplifica mucho las cosas. Cuanto más me adentraba, más ganas tenía de seguir.

Mi parte favorita de este campo es el Quantum Machine Learning que consiste en diseñar y aplicar conceptos de Machine Learning a los ordenadores cuánticos, como por ejemplo implementar cuánticamente las famosas Redes Neuronales, que están detrás de la mayoría de inteligencias artificiales que vemos hoy en día [1].

QML es un campo de investigación joven y en crecimiento debido a que sus algoritmos son ideales para implementarlos con los ordenadores cuánticos actuales, los cuales no son muy potentes. Ejemplos de estas implementaciones serían [insertar aplicaciones aquí], etc.

De entre todos los tipos de algoritmos me he centrado en las Redes Neuronales Cuánticas, análogas cuánticas de las Redes Neuronales tan utilizadas hoy en día para hacer gran variedad de tareas. Me he interesado particularmente en ellas debido a que tenía experiencia en el pasado con las RNs clásicas y había visto que existen frameworks de software para trabajar con ellas como TensorFlow Quantum [2] que me podían ayudar.

Para adentrarme en el campo de QML, he tenido que adquirir conocimientos en álgebra lineal, cálculo y física. Dentro de QML en concreto me he dedicado a leer papers que me interesan y en un par de ocasiones intentar implementar los algoritmos detallados en esos papers. Puede parecer algo imposible en principio debido a que no tengo acceso directo a un ordenador cuántico, no obstante estos no son necesarios debido a que las operaciones cuánticas pueden ser simuladas en un ordenador corriente de escritorio (con ciertas limitaciones). Pero puedo tener acceso a ordenadores cuánticos ya que IBM permite acceder a los

suyos mediante IBM Quantum Experience [3], aunque nunca he dado uso de ello debido a que no lo veía necesario.

En este trabajo de investigación me he propuesto implementar mediante código uno de los algoritmos que he visto en un paper, una Red Adversaria Generativa Cuàntica (GAN, en inglés) [4] que genera imágenes a partir de un circuito cuántico [5]. Como objetivo tengo verificar una sugerencia que hacen los autores del paper: implementar una función no-lineal en una parte del algoritmo que podría mejorar el rendimiento de este. Mi hipótesis al igual que los autores (aunque ellos lo comentan muy brevemente) es que el algoritmo va a reducir ligeramente el número de interacciones que son necesarias para llegar a su punto óptimo. Es decir, el modelo con la función no-lineal va a necesitar menos operaciones que lo entren conseguir los mismos resultados que el modelo sin la función.

Part I

Marc Teòric

Capítol 1

Àlgebra lineal

Quan vaig començar a buscar informació sobre computació quàntica, en vaig ràpidament donar compte que necessitava molt més coneixement matemàtic, degut a que no entenia gairebé res dels llibres sobre computació quàntica. Arran aquell temps, una serie de vídeos sobre àlgebra lineal en va captar l'atenció, que es justament la branca de les matemàtiques sobre la qual es basa la computació quàntica. Els vídeos son les lliçons que dona el Professor Gilbert Strang al Institut Tecnològic de Massachusetts (MIT en anglès) [6, 7]. Una vegada havia vist gairebé tots els vídeos, ja tenia bastants conceptes apresos.

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

En aquesta secció aniré explicant els conceptes bàsics de l'àlgebra lienal, per formar els coneixements en matemàtiques necessaris per poder comprendre aquest treball.

1.1 Vectors i espais vectorials

Els objectes fonamentals de l'àlgebra lineal són els espais vectorials. Un espai vectorial es el conjunt de tots els vectors que tenen les mateixes dimensions. Per exemple $\mathbb{R}^3$ seria el espai vectorial de tots els vectors de 3 dimensions, aquests

vectors normalment s'utilitzen per representar punts en un espai tridimensional. En computació quàntica un tipus d'espais vectorials en concret són utilitzats: Els espais de Hilbert, en altres paraules, un espai vectorial amb un producte interior [9]. Els espais de Hilbert segueixen un conjunt de productes i compleixen unes certes normes, en aquest capítol presentaré una part d'aquestes normes i productes, la quantitat que és necessària. S'ha de tenir en compte que els espais de Hilbert són molt més complicats que el que es representa en aquest treball, també que d'aquí en endavant, quan mencioni espai vectorial hem referiré a un espai de Hilbert, d'ha no ser que s'especifiqui el contrari.

Els espais vectorial estan definits per les seves bases, un set de vectors $B = \{|v_1\rangle, \ldots, |v_n\rangle\}$ es una base vàlida per 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 $|v_i\rangle \in B$. Els vectors de la base B són linealment independent entre ells.

La notació estàndard pels conceptes de àlgebra lienal en mecànica quàntica es la notació de Dirac, en la qual es representa un vector com $|\psi\rangle$. On ψ es la etiqueta del vector. Un vector $|\psi\rangle$ amb n dimensions també pot ser representat com una matriu columna que te la forma:

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

On els nombres complexes $(z_1, z_2, \ldots, z_{n-1}, z_n)$ són els seus elements. Un vector escrit com a $|\psi\rangle$ també s'anomena *ket*.

La adició d'un par de vectors en un espai de Hilbert es definida per [1]:

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

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

A més a més, hi ha una multiplicació per un escalar[2] definida per:

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

On z es un escalar i $|\psi\rangle$ un vector. Cal que notar que cada element del vector es multiplicar per el escalar.

Degut a que els espais de Hilbert son complexos tenen un conjugat complex definit per escalar com a: Per un escalar complex $z = a + bi$, el seu conjugat z^* es igual a $a - bi$.

Aquesta noció pot ampliar per a vectors i matrius, agafant el conjugat de totes els seus elements:

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

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

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

Un altre concepte important es la transposada, representada per el superíndex T que 'rota' un vector o una matriu. Un vector columna amb una dimensió $n \times 1$ es transforma amb un vector fila amb una dimensió $1 \times n$[3]:

$$\left|\psi\right\rangle^T = \begin{bmatrix}\psi_1\\ \vdots \\ \psi_n\end{bmatrix}^T = \begin{bmatrix}\psi_1 & \dots & \psi_n\end{bmatrix}$$

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

[2] Un numero qualsevol en $\mathbb{R}$.

[3] En realitat els vectors columna son matrius amb dimensió $n, 1$ però he estat ometent el 1. Quan hem refereixo a les dimensions de un vector qualsevol, només diré un numero, no obstant, especificaré si és un vector columna o un vector fila.

en una matriu $n \times m$. Per exemple:

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

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

$$|\psi\rangle^\dagger = (|\psi\rangle^*)^T = \begin{bmatrix} \psi_1^* & \dots & \psi_n^* \end{bmatrix} = \langle\psi|$$

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[4], que són noms que descriuen millor com funcionen aquests objectes. La definició formal d'una aplicació lienal pot ser bastant complicada, per aquesta raó, utilitzaré termes més informals en aquesta secció.

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

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

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

[4]

1. Addició de vectors:
   Donats els vectors $|\psi\rangle$ i $|\varphi\rangle$ en un mateix espai vectorial, i un operador 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 lienal A, es certs 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 son aplicacions lineals, això es pot provar fàcilment al veure que compleixen els criteris especificats posteriorment. No obstant, les derivades i les integrals usualment no s'apliquen a vectors, sinó a les funcions, però es possible aplicar-les a vectors [5].

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

1.1 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 rest del treball.

1. **Operador zero**
   Qualsevol espai vectorial té un vector zero expressat en notació de Dirac com a 0, degut a que $|0\rangle$ es un altre concepte totalment diferent en CQ i IQ[6]. El vector zero es aquell vector que per qualsevol vector $|\psi\rangle$ i qualsevol escalar z, es compleix que: $|\psi\rangle + 0 = |\psi\rangle$ i $z0 = 0$.
   El operador zero també s'escriu com a 0 i es defineix com l'aplicació que transforma qualsevol vector al vector zero: $0|\psi\rangle = 0$.

[5] No et preocupis, que es clar que les aplicaré a vectors :D.
[6] Computació Quàntica i Informació Quàntica.

2. **Matriu inversa**
   Un matriu quadrada[7] A és invertible si existeix una matriu A^{-1} de manera que $AA^{-1} = A^{-1}A$. $A^{-1} = I$ és la matriu inversa de A. La manera més ràpida de saber si una matriu es invertible es veient si el seu determinant no és zero. En altres paraules, és l'element neutre per la suma i l'element null per la multiplicació.

3. **Matriu Identitat**
   Per a qualsevol espai vectorial V existeix un operador identitat I que es definit com $I|\psi\rangle = |\psi\rangle$, aquest operador no fa cap canvi al vectors als quals opera. Cal notar també que per qualsevol matriu A i la seva inversa és veritat que $AA^{-1} = I$.

4. **Matriu Unitari**
   Un operador unitari es qualsevol operador que no altera la norma dels vectors al quals es aplicat, per tant, una matriu es unitària si $AA^{\dagger} = I$ Per convertir qualsevol operador en unitari, es divideix les seves entrades entre la norma del operador.

5. **Matrius Hermitianes**
   Una matriu Hermitiana compleix que:

$$\langle\psi|\left(A|\varphi\rangle\right) = \left(A^{\dagger}\langle\psi|\right)|\varphi\rangle$$

   On $|\psi\rangle$ i $|\varphi\rangle$, són dos vectors del espai en el qual actua A. Si apliquem el producte interior a aquesta equació tenim que: $A = A^{\dagger}$.

1.3 Producte interior i producte exterior

1.1 Producte interior

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

$$\langle a|b\rangle = a_1b_1 + a_2b_2 + ... + a_{n-1}b_{n-1} + a_nb_n = z$$

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

Amb $z, a_i, b_i \in \mathbb{C}$. Quan hem refereixo a un producte interior, normalment diré "el producte interior de dos vectors", quan en realitat es una operació entre un vector dual i un vector.

El equivalent d'aquest producte en un espai real de dos dimensions $\mathbb{R}^2$ és el producte escalar, que s'expressat com a :

$$\langle a|b\rangle = \||a\rangle\|_2 \cdot \||b\rangle\|_2 \cos\theta \tag{1.1}$$

Amb $\|\cdot\|_2$ sent la norma ℓ^2 definida com a $\||\psi\rangle\|_2 = \sqrt{\psi_1^2 + \cdots + \psi_n^2}$ amb 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, segons el que he vist, no es usada àmpliament ja que interpretar θ com un angle entre vectors de dimensions altes no té molt de sentit. En contrast, he vist aquest producte presentat en la seva interpretació geomètrica[8] com el producte entre un vector fila i un vector columna:

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

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

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

No obstant, la definició més comú es basa en el producte interior. Com es pot veure el producte interior de un vector per ell mateix es la suma de les seves entrades 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:

$$\||a\rangle\|_2 = \sqrt{\langle a|a\rangle} \tag{1.2}$$

Quan la norma es aplicada a un vector bidimensional, es pot veure que es lo mateix que la longitud Euclidiana d'aquell vector, això es perquè realment son el

[8] Els detalls exactes de l'interpretació geomètrica estan fora del domini d'aquest treball, malgrat que m'agradaria molt parlar sobre el tema.

mateixos conceptes, tot i això, la norma es el concepte de longitud però generalitzat a vectors de dimensions altes.

Pel el que jo entenc algunes propietats de la longitud d'un vector bidimensional no es mantenen amb la norma d'un vector que te més de 2 dimensions. En altres paraules, la norma es comporta en certes maneres com la distancia des de d'origen (que es la definició de la longitud), per tant, no són exactament lo mateix. A més d'això, hi han diferents tipus de normes que s'utilitzen en diversos escenaris. Aquesta es la raó per la qual en refereixo a la norma, com la norma ℓ^2. A aquesta norma també se li diu norma Euclidiana [12].

1.1.1 Propietats del producte interior

Les propietats bàsiques del producte interior són les següents:

1. És lineal en el segon argument $(z_1 \langle a| + z_2 \langle c|) |b\rangle = z_1 \langle a|b\rangle + z_2 \langle c|b\rangle$
2. Té simetria en el conjugat $\langle a|b\rangle = (\langle b|a\rangle)^*$
3. $\langle a|a\rangle$ es no-negativa i real, excepte en el cas de $\langle a|a\rangle = 0 \Leftrightarrow |a\rangle = 0$

1.2 Vectors ortonormals i ortogonals

A partir del concepte de norma sorgeixen els conceptes de un par de vectors ortonormals i un par de vectors ortogonals [9]. 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. D'aquesta manera, si el producte interior d'un vector és 1, és un vector unitari.

De vectors que no són zero son ortogonals si el seu producte interior és zero. Si aquests vectors són de bidimensionals, a partir de l'equació (1.1) podem veure

[9] Una nota graciosa, la primera vegada que vaig trobar aquest dos conceptes els vaig confondre i pensava que eren la mateixa cosa, això hem va confondre moltíssim i no entenia gairebé res.

que l'angle que fan entre ells és zero i per tant son perpendiculars entre ells:

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

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

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

Per tant el terme que falta $\cos\theta$ és igual a zero.

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

No obstant, pensar que la perpendicularitat i la ortogonalitat són els mateixos conceptes es un error, degut a que, el que siguin només es veritat per els vectors bidimensionals. Perquè com en el cas de la norma i la longitud, la ortogonalitat és el concepte de perpendicularitat generalitzat.

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

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

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

Una altre cosa que remarcar i no he dit es que al dir un par de vectors, aquest par també pot ser un set de vectors. Si un set té tots els vectors unitaris i ortogonals entre tots ells, és un set de vectors ortonormals. El set de vectors $B = \{|\beta_1\rangle, |\beta_2\rangle, ..., |\beta_{n-1}\rangle |\beta_n\rangle\}$ 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 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 al institut, i és una mira 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 dimensió n. El producte exterior és l'aplicació lineal A de dimensions $m \times n$ en l'espai $\mathrm{Mat}_{m\times n}$:

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

Amb la seva acció definida per:

$$(|v\rangle\langle w|)\,|v'\rangle \equiv |w\rangle\langle v|v'\rangle = \langle v|v'\rangle\,|w\rangle \tag{1.3}$$

A partir de l'equació (1.3) l'utilitat i significat del producte es bastant complicada 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 elment de $|b\rangle$ formant una matriu amb mida $m \times n$:

$$|a\rangle\langle b| = \begin{bmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_n \\ a_2b_1 & a_2b_2 & \cdots & a_2b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_mb_1 & a_mb_2 & \cdots & a_mb_n \end{bmatrix}$$

L'utilitat d'aquest producte és mostrara més endavant.

1.4 Producte tensorial

L'últim producte que mencionaré és el tensorial, representat per el 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 en

centraré en explicar la manera amb la qual es computa utilitzant la representació matricial d'aquest producte, anomenada el producte de Kronecker.

Per una $m \times n$ A, i una $p \times q$ matriu B, el seu producte de Kronecker [13] 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 $a_{ij}B$ es 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 es multiplicat per la segona matriu:

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

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

Una altre notació a tenir en compte és el símbol $\bigotimes$, utilitzat per representar el equivalent de la suma (expressada amb $\sum$), però en comptes de la adició el producte de Kronecker és utilitzat. En altres paraules, $\bigotimes$ representa el producte de Kronecker de un nombre finit de termes. Per clarificar, aquí hi ha un exemple amb una matriu identitat:

Amb $\mathbb{I}$ com la matriu $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ i n com una potencia de 2:

$$\mathbb{I}_n = \bigotimes^{\log_2 n} \mathbb{I}$$

Aquí està el cas per $n = 8$:

$$\mathbb{I}_8 = \bigotimes^{\log_2 8} \mathbb{I} = \bigotimes^{3} \mathbb{I} = \mathbb{I} \otimes \mathbb{I} \otimes \mathbb{I} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

El producte de Kronecker també funciona amb vectors de la mateixa manera, però amb una multiplicació de escalar per vector.

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

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

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, no és molt comú fer-ho.

page 34 of QC-intro, inner product for a space $V \otimes W$

1.1 Propietats del producte tensorial

Les propietats bàsiques del producte tensorial són les següents [14, 15]:

1. Associativitat:
   $A \otimes (B + C) = A \otimes B + A \otimes C$
   $(zA) \otimes B = A \otimes (zB) = z(A \otimes B)$
   $(A \otimes B) \otimes C = A \otimes (B \otimes C)$
   $A \otimes 0 = 0 \otimes A = 0$
2. No-commutativitat [10]:
   $A \otimes B \neq B \otimes A$

1.5 Traça

La traça d'una matriu és tal sols la suma dels seus elements en la diagonal principal, la diagonal que va d'abaix a dalt i d'esquerra a dreta.

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

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

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

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

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

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

[10] Una cosa guay es 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.

La traça d'una matriu té les propietats següents:

1. Operador lineal:
   Degut a que la traça és un mapa lienal, es compleix que:
   $\mathrm{Tr}[A+B] = \mathrm{Tr}[A] + \mathrm{Tr}[B]$ i $\mathrm{Tr}[zA] = z\,\mathrm{Tr}[A]$, per a totes les matrius quadrades A i B, i tots els escalars z.

2. Traça d'un producte tensorial:
   $\mathrm{Tr}[A \otimes B] = \mathrm{Tr}[A]\,\mathrm{Tr}[B]$

3. La transposada té la mateixa traça:
   $\mathrm{Tr}[A] = \mathrm{Tr}\left[A^T\right]$

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:
   $\mathrm{Tr}[AB] = \mathrm{Tr}[BA]$

Una altre manera molt útil de computar la traça d'un operador és a través del procediment de Gram-Schmidt[11] i un producte exterior. Utilitzant Gram-Schmidt per representar el vector unitat $|\psi\rangle$ amb una base ortonormal $|i\rangle$ que inclou $|\psi\rangle$ com el seu primer element, es veritat que:

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

[11] Mirar A.1 per una definició del procediment de Gram-Schmidt.

Capítol 2

Computació quàntica

Després de bastant teoria matemàtica, ha arribat el temps de parlar sobre mecànica quàntica[1], en aquest capítol introduiré alguns conceptes bàsics sobre Informació Quàntica i Computació Quàntica (QI i QC).

Quantum mechanics is a mathematical framework or rather a set of theories used to describe and explain the physical properties of atoms, molecules, and subatomic particles. It is the framework of all quantum physics including quantum information science. The right way of presenting quantum computation is through the formal quantum mechanics postulates because, with them, the statements made in quantum computation do not seem to come from anywhere [16]. However, to not complicate this section more than it is, I will do my best to explain the concepts and math of quantum computing just on their own, without presenting more generalized concepts from quantum mechanics, unless it is totally necessary to do so.

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 computació quàntica és representen per estats quàntics, els quals són algun tipus de distribucions de probabilitat per els possibles resultats d'una mesura on un sistema quàntic [17].

[1] No crec que aquesta frase doni molts ànims per continuar.

Imaginat que tens un boli, però que no saps de quin color és, no obstant, saps que pot ser vermell o blau. Per esbrinar de quin color és, pots provar a escriure amb el boli per veure el color de la pinta, o en altres paraules, fer una mesura. Saps que hi ha un 50% de probabilitat de que sigui vermell i un 50% de que sigui blau. En aquesta situació hipotètica tindries el teu sistema quàntic (el boli), una manera de mesurar-lo (escriure alguna cosa) i una llista amb els possibles resultats (50% vermell, 50% blau), només et falta una manera de representar-lo tot matemàticament, el estat quàntic. Per tant, per què no intentem guardar la informació que sabem del boli 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 de que el boli sigui vermell i la segona entrada de 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 de les entrades d'un vector[2], en altres paraules la norma ℓ_1 d'aquest vector és 1.

Llavors, hem d'escollir una operació matemàtica per poder extreure la informació del vector, com que el output ha de ser un número, podem provar a 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 $|0\rangle$ és el vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, i, $|1\rangle$ és el vector $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Per tant, per trobar la probabilitat, s'agafa el producte interior del vector amb la base corresponent a la probabilitat, com a continuació:

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

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

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

Aquest procediment és bastant senzill, com a un altre exemple, per representar un boli amb 6 colors possibles amb una possibilitat aleatòria d'escriure amb un color del 6 possibles, l'estat d'aquest boli és[3]:

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

Ara veure la probabilitat de per exemple treure el color verd, utilitzant la tercera base, la que 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 bolis a una banda, ara 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 polarització[4]. Al mirar als fotons com ones en que oscil·len en el camps electromagnètic, la polarització és l'orientació geomètrica de l'ona. La polarització pot ser interpretada com un angle respecte a la direcció de propagació. posar una figura aquí

Al definir les bases del estat de polarització com vertical i horitzontal, denotades per els vectors $|\rightarrow\rangle$ i $|\uparrow\rangle$, respectivament. Podem definir un estat de superposició entre les bases, representat com $|\nearrow\rangle$ [18]:

$$|\nearrow\rangle = \alpha\,|\rightarrow\rangle + \beta\,|\uparrow\rangle \tag{2.1}$$

[3] Le posat colors a els elements per claredat.

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

On α i β són números complexos. Un estat en superposició es simplement un estat on l'angle de polarització no és 0 ni $\frac{\pi}{2}$. No ens tenim que preocupar de la descripció matemàtica exacte de la polarització dels fotons al parlar de computació quàntica perquè el que importa és l'informació que porten aquests estat, no la física dels sistemes que representen. Aquesta informació s'ha d'agafar mitjançant mesures, com amb els bolis. Aquestes mesures en el cas de la polarització dels fotons seria passar-los per diversos filtres de polarització, els quals deixen passar el fotó o l'absorbeixen, tot això d'una manera probabilística, es a dir, dependent de quin sigui l'estat tenen certa possibilitat de ser absorbits o no.

Per clarificar, el filtre lo que fa es col·lapsar el fotó en els dos possibles estats de polarització, l'estat en el quan el filtre es orientat o l'estat perpendicular a aquest. Si col·lapsa en l'estat del filtre el fotó passa pel el filtre, en cas de col·lapsar en l'altre estat, es absorbit. La manera en la que els fotons col·lapsen és probabilística, si agafen un filtre que està orientat horitzontalment respecte al fotons que li arriben, un fotó orientat en horitzontalment té un 100% 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, es a dir a 45º, 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, on els estats que els representen tenen la informació sobre quines són aquestes possibilitats. No obstant, 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 saqueta 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

Ordinadors modern representen informació a través de cadenes de zeros i uns, anomenats bits. Tot, des de imatges a lletres o instruccions electròniques. Per exemple, la lletra t és representada per la cadena de bits 01110100, codificat a través de codi binari. Tot el que fas en un ordinador es codifica i representa en codi binari.

Degut a que estem molt acostumats a codi binari, en el camp de la computació quàntica també s'utilitza, no obstant, en comptes de bits s'utilitzen qubits. Un qubit es 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'entrellaçament[5]. Si un bit pot estar en l'estat 1 o en l'estat 1, un qubit pot estar en una combinació d'aquests estats, en un estat enmig del 1 o el 0. És una combinació lienal dels vectors que representen aquests estats, $|0\rangle$ 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 bidimensional espai de Hilbert[6]. 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, 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 qualsevol nombre, necessiten ser els coeficients que formin un vector amb una norma de 1.

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

Llavors:

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

Definir un qubit com una 'combinació lienal dels estats fundamentals' no és de molta ajuda, per això, elaboraré més sobre aquesta definició: Una cadena de n-bit pot només representar una única combinació d'uns i zeros, mentres que una cadena de n-qubits representa una combinació de totes les possibles combinaci-

[5] Ja he parlat de la primera amb la polarització del fotons, del entrellaçament parlaré més endavant.

[6] Un espai vectorial amb un producte interior.

ons. En el cas d'un qubit, aquest és una combinació del possible 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.

Definir un qubit com una çombinació lienal dels estats fundamentals"no és de molta ajuda, per això, elaboraré més sobre aquesta definició: Una cadena de n-bit pot només representar una única combinació d'uns i zeros, mentres que una cadena de n-qubits representa una combinació de totes les possibles combinacions. En el cas d'un qubit, aquest és una combinació del possible 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 números que representa és 2^n, on aquests números son els coeficients de la combinació lineal. Això es 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-los[7], no com els bits que al ser només una combinació és necessita només saber quina combinació és. No passa res si això no s'entén perfectament, lo important és saber que es necessiten 2^n números complexos[8] per representar n-qubits i que es necessiten n números binaris per representar n-bits.

Per il·lustrar tot això, 2 qubits es representen amb el *statevector*:

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

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.

[7] Informació que es manifesta amb els coeficients de la combinació lineal

[8] Són complexos degut a que els coeficients de la combinació lienal són números complexos.

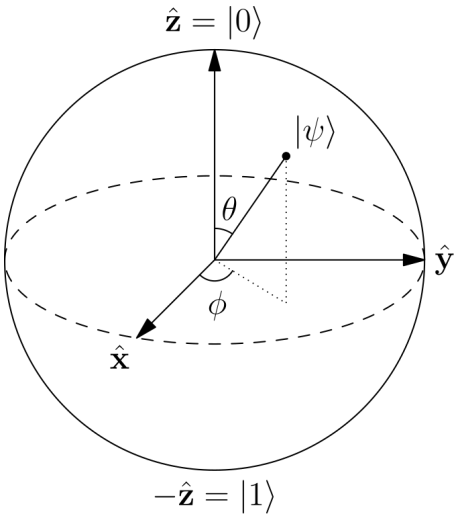


Figura 2.1: Esfera de Bloch, on es representa un estat arbitrari $|\psi\rangle$ amb els vectors $\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}$ representat els eixos ortonormals de la esfera.

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.

2.1 Representació geomètrica d'un qubit

Una aspecte que sempre m'agradat del qubits és la seva representació geomètrica, la esfera de Bloch 2.1. Si agafem una esfera unitària[9] que té els seus pol nord i sur definits per els 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 Operacions per a només un qubit

Una vegada es té la informació representada, estaria bé poder operar amb aquella informació, aquest es justament lo que fa que els ordinadors siguin ordinadors, poder operar amb la informació. En computació quàntica els qubits al poder ser

[9]Una esfera que té radi 1 i que per tant qualsevol punt en la seva superfície correspon a un vector en $\mathbb{R}^3$ unitari.

representats amb vectors que tenen els seus coeficients són operats per les anomenades portes lògiques quàntiques, que són matrius. Per exemple, si es vol passar de tindre un qubit en l'estat $|0\rangle$ al estat $|1\rangle$, s'utilitza la porta lògica X representada a continuació:

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

Podem veure com fa l'acció al multiplicar la matriu per el 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 lo 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 important, les matrius de Pauli. Hi han 3 d'aquestes la X, la Y i la Z, usualment representades per X, Y, Z o per $\sigma_x, \sigma_y, \sigma_z$. Aquestes matrius són les següents:

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

Aquestes matrius són molt important en la mecànica quàntica[10] i són utilitzades àmpliament per descompondre i com a portes lògiques quàntiques.

A partir d'elles podem elaborar matrius que facin una rotació de qualsevol

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

angle en un del eixos de la representació geomètrica d'un qubit 2.1:

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

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

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

Per exemple la matriu $R_y(\cdot)$ (Eq.2.3) correspon a una rotació en el eix $\hat{\mathbf{y}}$ de la 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. Es a dir una superposició que tingui les mateixes probabilitats de resultar en $|0\rangle$ o $|1\rangle$[11]. Aquesta és la porta de Hadamard, denotada per H:

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

Podem comprovar que és una superposició uniforme al aplicar-la al 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 $|+\rangle$[12] La probabilitat de 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 del coeficients es pot veure que dona $\frac{1}{2}$:

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

[11] Un 50% cada una.

[12] Un altre estat similar és $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$, quan la matriu H s'aplica al estat $|1\rangle$.

Llavors tenim que la probabilitat per obtindre ambos estats és la mateixa, es a dir que si mesurem l'estat $|+\rangle$ hi ha la mateixa probabilitat de que surti $|0\rangle$ o $|1\rangle$. La porta lògica de Hadamard és molt important ja que s'utilitza per crear distribucions uniformes, ja sigui en un qubit o en diversos [13].

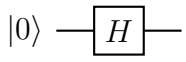
Altres operacions importants de només un qubit són les portes S i T:

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

2.3 Circuit quàntics

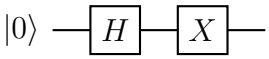
Aquestes operacions usualment es representen a través de circuits quàntics. Són representacions gràfiques que indiquen de quines operacions s'apliquen a quins qubits i en quin ordre[14].

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



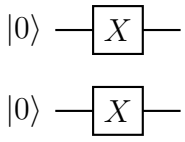
El qubit es representat per la línea que comença amb $|0\rangle$, amb $|0\rangle$ sent el seu estat inicial. Aquests diagrames es llegeixen d'esquerra a dreta, la mateixa forma en la qual s'apliquen les operacions.

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

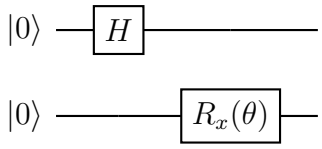


Múltiples qubits son simplement més línies, aquí estan representat dos qubits amb portes X aplicades a cada un:

[13] S'aplica aquesta operació a cada qubit del sistema.
[14] A mi em semblen semblats a les partitures musicals.



Amb múltiples qubits també hi han un ordre en el qual les portes s'han d'aplicar, un circuit en el qual es representa que s'aplica una porta Hadamard al primer, al principi, i una rotació[15] en l'eix x en el segon qubit a continuació, seria:



2.4 Operacions per a múltiples qubits

Lo realment interessant es quan s'apliquen portes a diversos qubits, perquè d'aquesta manera és pot arribar a tindre qubits entrellaçats. La porta més útil per entrellaçar qubits és la CNOT o *Controlled NOT*:

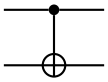
$$\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.6}$$

És bàsicament una porta X[16] que està controlada per un altre qubit, si l'altre qubit és $|1\rangle$ s'aplica la porta X al altre qubit. Però si l'altre qubit està en superposició, per exemple en $|+\rangle$, aquesta superposició es passa també al qubit controlat, i al mesurar el qubit, la probabilitat de que s'apliqui la porta X és la probabilitat de mesurar l'estat $|1\rangle$. Es considera que aquests qubits estan entrellaçats, una mesura a un d'ells afecta a la mesura del altre. El qubit al qual se li aplica la porta X es diu *target* i el qubit sobre el qual depèn el *target* és el *control*.

Aquesta porta representa en un circuit s'escriu com:

[15] D'un angle θ.

[16] També anomenada porta NOT degut al paral·lelisme que es fa amb la porta lògica del ordinadors clàssics NOT [19] que simplement inverteix els bits d'1 a 0 (i viceversa), igual que X que inverteix els qubits $|1\rangle$ a $|0\rangle$ i viceversa.



On el qubit *control* és el primer i on el segon és el *target*, el qubit que té el símbol $\oplus$[17].

2.4.1 Entrellaçament quàntic

L'exemple més senzill d'un entrellaçament quàntic en la computació quàntica son els parells de Bell, que es creen al aplicar a dos qubits una porta H al primer i després una porta CNOT als dos creant l'estat:

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

Es pot veure que només hi han dos estats possibles $|00\rangle$ i $|11\rangle$ que tenen la mateixa probabilitat associada[18]. S'afecten l'un al altre en el sentit que quan es mesura només un dels qubits i dona per exemple $|1\rangle$, al mesurar l'altre també dona $|1\rangle$, 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 altre part del sistema.

Matemàticament un sistema quàntic, e.i. 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 si ho és.

A partir del entrellaçament i la superposició és com els ordinadors quàntics arribem a tenir avantatges en complexitat sobre els ordinadors clàssics, per més informació sobre els avantatges que presenten els algoritmes quàntics en certes tasques veure l'apèndix D.

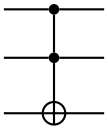
[17] A vegades escriure els circuits quàntics sense especificar l'estat inicial degut a que no és necessari.

[18] Això es pot veure al elevar al quadrat els coeficients del dos, que donen $\frac{1}{2}$.

2.4.2 Operacions controlades

A part de la porta CNOT existeixen diverses portes quàntiques controlades. Realment és 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 podem haver-hi diversos qubits *control* i *target*.

Per exemple existeix la porta Toffoli[19]:



Que en la seva forma matricial és:

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

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

Tornant a dos qubits, al veure la matriu per la porta CNOT, es pot apreciar que està comporta per una una matriu identitat i una porta X[20]:

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

[19] Inventada per Tommaso Toffoli XD

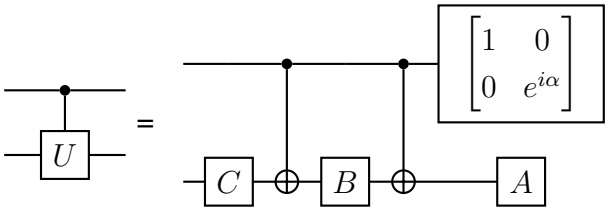
[20] La matriu identitat es troba a la cantonada superior esquerra, mentre que la porta X es troba al altre extrem.

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

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

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

On obstant, una operació controlada de qualsevol unitària U es forma a través del següent circuit:



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

2.3 Mesurament quàntic

Com ja s'ha esmentat a la secció 2.2 al elevar al quadrat el coeficient d'un estat base que forma part d'un estat, s'obté la probabilitat d'obtenir l'estat base quan es mesura.

Aquesta és la forma més simple de poder predir el mesurament d'un estat quàntic. Però hi han més coses a dir que son útils.

Els mesuraments quàntics són un conjunt d'operadors de mesura M_m, la probabilitat del estat m associat a un operador M_m, és:

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

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 las suma de les seves probabilitats sigui u:

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

La diferencia entre aquesta manera de fer les mesures i elevar els coeficients al quadrat, és que l'equació 2.3 és una forma més general, on en comptes de mesurar en la base computacional, és pot mesurar en qualsevol base. A més a més, utilitzant aquest mètode no es necessari saber la composició[21] del estat que vols mesurar.

Per mesurar en la base computacional un *statevector* s'utilitzen operadors de mesura derivats de la base computacional amb productes exteriors. Per crear l'aplicació lienal 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 del estat $|\psi\rangle$ resulti en $|0\rangle$, és:

$$\text{prob}(|0\rangle = \langle\psi|0\rangle \langle 0|^\dagger |0\rangle \langle 0|\psi\rangle$$

Per $|\psi\rangle = |0\rangle$ tenim que[22]:

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

El resultat té sentit degut a que si mesurem $|0\rangle$ en la base $|0\rangle$, esperem que el resultat sigui 1.

[21] Els coeficients del estats base que descomponen el vector.

[22] Cal 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 serie de qubits es pot representar tant per un vector com per una matriu, anomenats *statevector* i *density matrix*, respectivament [8]. En aquesta secció aniré ràpidament sobre el concepte de la matriu de densitat i com les operacions que s'apliquen a un *statevector* poden ser aplicades a una *density matrix*. Després parlaré del mesuraments parcial d'un sistema, un concepte que es important per la part experimental del treball.

Una matriu densitat és la representació matemàtica d'un estat quàntic a partir de d'un matriu, es a dir, d'un operador. Aquesta representació serveix descriure sistemes quàntics que no són completament coneguts. Concretament aquests operadors són conjunts de estats quàntics, per un sistema que es descriu amb el estats $|\psi_i\rangle$ que tenen probabilitats p_i[23] la matriu densitat del sistema ρ [20]:

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

Cal matriu densitat 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 estat ρ que descriu un sistema quàntic, al igual que amb els vectors, s'efectua a partir d'operadors unitaris[24], d'aquesta manera tenim que la evolució d'un operador densitat és:

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

Al igual que es fan mesures en *statevectors*, les podem amb les *density matrices*. Per operadors de mesura M_m, tenim que la probabilitat de tindre l'estat m és:

$$\begin{aligned} \text{prob}(m) &= \left\langle\psi_i\right| M_m^\dagger M_m \left|\psi_i\right\rangle = \text{tr}\big(M_m^\dagger M_m \left|\psi_i\right\rangle \left\langle\psi_i\right|\big) \\ &= \text{tr}\big(M_m^\dagger M_m \rho\big) \end{aligned}$$

[23] i és l'índex que relaciona un estat a un probabilitat.
[24] Recorda que han de preservar la norma del vector, i en el cas de les matrius la seva traça.

També tenim que l'estat $|\psi_i\rangle$ després de la mesura m és [20]:

$$|\psi_i\rangle = \frac{M_m |\psi_i\rangle}{\sqrt{\langle\psi_i| M_m^\dagger M_m |\psi_i\rangle}} = \frac{M_m |\psi_i\rangle}{\sqrt{\mathrm{tr}\Big(M_m^\dagger M_m \rho\Big)}} \tag{2.8}$$

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

$$\begin{aligned}\rho_m &= \sum_i p_i \frac{M_m |\psi_i\rangle \langle\psi_i| M_m^\dagger}{\mathrm{tr}\Big(M_m^\dagger M_m \rho\Big)} \\ &= \frac{M_m \rho M_m^\dagger}{\mathrm{tr}\Big(M_m^\dagger M_m \rho\Big)}\end{aligned}$$

Perquè això sigui cert els operadors de mesura M_m, ha'n 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 al igual que els vectors d'estat han de tenir certes característiques:

1. La traça de ρ ha d'equivaler 1.
2. ρ ha de ser un operador positiu[25].

2.1 Matriu de densitat reduïda

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

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

$$\rho^A = \mathrm{tr}_B(\rho^{AB}) \tag{2.9}$$

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

On tr_B és la traça parcial sobre el sistema B, que es defineix per l'equació següent [21]:

$$\mathrm{tr}_B(|a_1\rangle \langle a_2| \otimes |b_1\rangle \langle b_2|) = |a_1\rangle \langle a_2| \,\mathrm{tr}(|b_1\rangle \langle b_2|)$$

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

No obstant aquesta definició no es pot utilitzar quan no saps com representar la matriu de densitat ρ com a un producte vectorial, en el qual un dels termes és l'estat que es traça afora. En altres paraules, en la 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.

Degut a això en el llibre de text *Quantum Computing: A Gentle Introduction* [22] els autors defineixen la traça parcial d'una altre manera més general, on només s'ha de saber les bases del sistemes A i B i un operador vàlid per el sistema AB. Per una matriu de densitat ρ^{AB} que representa el sistema $A \otimes B$, la traça parcial de ρ^{AB} sobre B és:

$$\mathrm{tr}_B\, \rho^{AB} = \sum_i \langle \beta_i| \, \rho^{AB} \, |\beta_i\rangle$$

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

$$(\mathrm{tr}\, \rho^{AB})_{ij} = \sum_{k=0}^{M-1} \langle a_i| \langle \beta_k| \, \rho^{AB} \, |\alpha_j\rangle \, |\beta_k\rangle$$

Amb la matriu sent:

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

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

Es pot comprovar que aquestes definicions són correctes degut a que en els dos llibres utilitzen les seves definicions per tractar el mateix cas i obtenen el mateix resultat [23, 24].

2.1.1 Mesurament parcial

Es pot arribar a treure una mesura parcial sobre un sistema de qubits amb la traça parcial i operadors de mesura. En el paper fet per Huang et.al. [5] es descriu un estat ρ després d'un mesurament parcial Π_A sobre un sistema A del estat $|\psi\rangle$ com:

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

El sistema A està compost per N_A qubits per tant la resta del 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 sorgeix un problema amb el numerador de l'equació perquè Π_A no té les mateixes dimensions que $|\psi\rangle \langle\psi|$, encara no he pogut utilitzar aquesta equació adequadament. No sé com computar-la. Parlaré d'això més endavant en la part experimental d'aquest treball.

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

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

Al igual que l'equació per les matrius de densitat parlaré d'aquesta contradicció en la part experimental.

2.5 Ordinadors quàntics

Tota aquesta teoria és aplicada a través de qubits físics que s'ubiquen al ordenadors quàntics. Hi han diversos tipus d'ordinadors quàntics, degut a que els qubits poden ser diversos sistemes. Poden ser fotons, chips de silici superconductors o ions atrapats per imants. No elaboraré més sobre aquest tema degut a 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 necessito un? No, perquè puc simular l'evolució dels estats quàntics amb un ordinador, pensa que tot ser expressat amb àlgebra lienal. No obstant, quan s'intenta simular un sistema quàntic de molt qubits[26] un ordinador de sobretaula tardaria molt de temps i realment no és viable. Per més informació sobre algoritmes quàntics i com s'executen en ordinadors i en simuladors, es pot llegir l'apèndix D.

[26] Més de 50 per exemple.

Capítol 3

Intel·ligència artificial

Segurament has sentit parlar de l'intel·ligènica artificial o de les xarxes neuronals, són conceptes que semblen abstractes però jo penso que son bastant intuïtius, intentaré que tú et sentis de la mateixa manera al final d'aquest capítol.

Intel·ligència artificial és un mot una mica ambigu, que és refereix a qualsevol algoritme que entra dintre del camps del *machine learning* o aprenentatge automàtic[1]. Aquests algoritmes simplement s'alteren a ells mateixos per fer millor la tasca que s'ha li ha designat, no importa quin és el objectiu o com ho aconseguix, lo que importa és si aprèn automàticament. Cal notar que els canvis que s'efectuen 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 ven bé una intel·ligència o un algoritme de *machine learning*.

La manera que tenen aquests algoritmes d'alterar-se a si mateixos usualment es canviant els paràmetres de les operacions dels quals estan compostos. 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 reffig:leastsquares.

Hi han diversos mètodes per ajustar els paràmetres, el més comú es ajustarlos 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 minimitzada o maximitzada, per exemple, en una regressió

[1] No obstant, col·loquialment s'utilitza per denominar a qualsevol algoritme o robot que és intel·ligent o sembla que és intel·ligent.

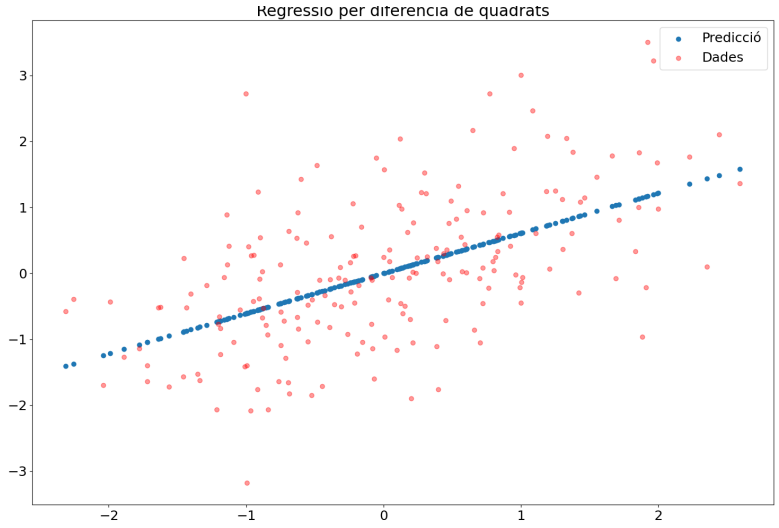


Figura 3.1: Exemple d'una regressió lineal de dades generades al atzar. Veure el codi a E.1.0.1.

lienal es vol reduir la distancia entre els *data points* o dades i la línia que prediu la tendència, Fig. 3.1.

Degut a que es poden realitzar molts tipus de funcions de pèrdua, ja sigui per la forma de la funció en si o per els paràmetres de la funció. Per conseqüència, el programes de *machine learning* són extremadament versàtils, la màxima expressió d'això es pot veure en les xarxes neuronals o *neural networks*. Aquests algoritmes són els més potents, complexos i polivalents. Precisament utilitzo un d'aquests per generar les imatges. Són àmpliament utilitzats pel reconeixement d'imatges, traducció i sintetització de textos, conducció automàtica, algoritmes de recomanació i es clar, generació d'imatges [citation].

3.1 Xarxes neuronals

Aquests tipus d'algoritmes que entren dintre de la categoria de *machine learning* no tenen un nombre que fa recordar a les xarxes de neuronals que formen part del nostre sistema nerviós central per casualitat, 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 la estructura que aquestes formen podem tindre algoritmes 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·ligencia artificial constitueixen el camp del *deep learning* o aprenentatge profund. S'anomenen d'aquesta forma per referenciar la profunditat d'aquests algoritmes, es a dir el nombre de capes que tenen.

Una neurona consisteix simplement en una suma ponderada, una altra suma i una funció no lineal que s'aplica al resultat. Les neurones tenen com input i output vectors. Per tant, una neurona es pot definit com:

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

Per σ sent una funció no lienal i n sent la mida del vector. Després estan el paràmetres, w_i i b, anomenats *weights* i *bias*. Aquests paràmetres tenen una utilitat pot clara, per això tenim que parlar en termes d'importància. En altres paraules, com afecta cada paràmetre a una part de la xarxa i també com afecta aquest al output de la xarxa. Després de deixar clara l'equació s'ha d'il·lustrar la interconnectivitat que tenen les neurones entre si.

Una neurona pot tindre diversos inputs que venen de diverses neurones, el mateix passa amb els outputs. Depèn de com es connectin entre si o de quina operació addicional fan les neurones, aquestes formen diversos tipus de capes. A partir de les tipus de capes i el nombre d'aquestes es com s'especifica l'arquitectura d'una xarxa neuronal. Tornant a l'utilitat del paràmetres, un *weight* especifica com de forta es la relació entre una neurona en una capa i una altre neurona en una capa veïna. I un *bias* especifica com d'important és una neurona, degut a que aquest número afecta al resultat de la suma de la neurona, fent que aquesta sigui més alta.

Usualment l'arquitectura es divideix en tres parts la capa d'input, les capes ocultes i la capa d'output. La quantitat de neurones que hi han a la capa d'inputs es la que defineix la mida del vector que es dona com input a la xarxa, degut a que cada element del vector es dona a cada neurona amb la capa. 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 número de neurones que té cadascuna d'aquestes dues capes, especifica la mida del 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[2] calen 256 neurones en la capa d'inputs, una per cada pixel.

[2]Una imatge en blanc i negre.

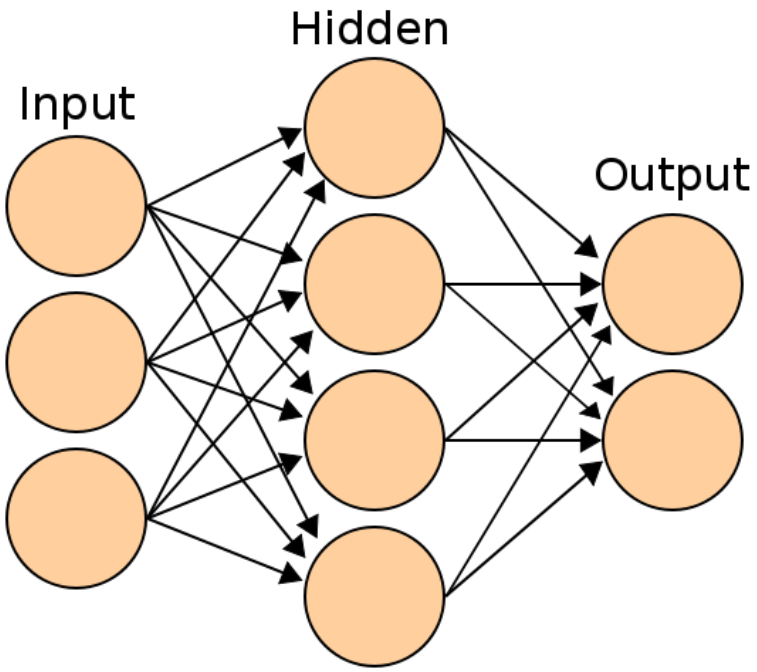


Figura 3.2: Usualment les xarxes neuronals es representen d'aquesta forma, amb fletxes i boles. Les boles representarien cada neurona i les fletxes mostren com estan connectades. La *hidden layer* d'aquesta representació es pot veure que és una *fully connected layer* degut a com està connectada a les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

En canvi, les *hidden layers*, es a dir les capes ocultes, no tenen una mida determinades, 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 el diversos tipus de capa és el que dona la versatilitat d'aquests algoritmes, com ja he comentat.

Entre els diferents tipus de capes que ponen tindre les xarxes neuronals, el més comú i simple d'aquestes és una *fully connected layer*, o una capa completament connectada, veure 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. La forma que tenen aquestes capes de variar es mitjançant la mida que tenen, es a dir, la quantitat de neurones que tenen. També es pot variar la funció d'activació que tenen, que ha de ser una funció no lineal. La més utilitzada és la sigmoide:

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

3.2 Descens del gradient

Una vegada he parlar de les xarxes neuronals, he de comentar com aquestes evolucionen al llarg del temps, es a dir com es van actualitzen a si mateixes per complir la tasca que s'ha li's ha encomanat. Ja he comentat que existeix una funció anomenada *loss function*, la qual es deriva per actualitzar els paràmetres de la xarxa. En aquesta secció parlaré més en profunditat d'aquest mecanisme que s'anomena el descens (o ascens) del gradient.

Es 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, els punts als quals es vol arribar. Per exemple, si es volen classificar imatges de gats i gossos, s'assignen dos etiquetes a les imatges, 1 als gats i 0 als gossos. Utilitzaré com a exemple la funció de pèrdua *binary cross entropy* o *log Loss* per poder il·lustrar com una funció pot representar si una xarxa neuronal està fent el seu treball o no. La funció *binary cross entropy* és la següent:

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

Per t_i sent la etiqueta real que ha de tenir l'imatge i y_i sent la etiqueta que el model dona a l'imatge. Per tant si l'etiqueta real és 1, l'equació acaba sent:

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

I el cas contrari per $t = 0$:

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

Si donen com input l'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 és més gran. El mateix es pot veure per les imatges del gossos o en

altres paraules 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 quan més pròximes estén les prediccions (els outputs del model) a les etiquetes del programa, menor serà la pèrdua, llavors al minimitzar la funció es trobarà el punt òptim on totes les prediccions seran iguals a les etiquetes.

A aquests punts òptims s'han arribat actualitzant els paràmetres a través de la derivada de la funció, concretament a través del gradient. El gradient d'una funció és un vector on els seus elements són la derivada parcial respecte a cada paràmetre (per aclarir, un element 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 lo que és exactament un gradient, l'únic que importa es que cada paràmetre de la xarxa neuronal[3] s'ha d'actualitzar acorde amb la derivada. Això es pot entendre com canviar lleugerament un paràmetre acord amb la direcció. Aquesta direcció es indica per la derivada. L'objecte que s'encarrega d'actualitzar-los, s'anomena optimitzador. L'optimitzador més senzill que hi ha és:

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

Aquí s'actualitza un paràmetre θ_i, he posat el superíndex t per expressar que s'actualitza, passant d'un temps $t-1$ a t. En aquesta equació utilitzo θ per expressar tots els paràmetres del model. En els optimitzadors s'afegeix un número η anomenat *learning rate*. Usualment és un número que petit[4], aquest paràmetre especifica com de ràpid cabien els paràmetres. El *learning rate* es pot anar ajustant depenen de la situació, del model en el qual s'implementi. Alteracions en aquest pot causar diversos fenòmens tan negatius com positius. Cal notar que en l'optimitzador he utilitzar el signe $\pm$ perquè si és positiu significa que s'està

[3] Cada *weight* i cada *bias*.
[4] Entre 0.1 i 0.001 per exemple.

ascendint pel gradient, i si és negatiu s'està descendint. No obstant, usualment s'utilitza el signe negatiu per convenció.

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 s'ha de fer la derivada del output de la xarxa neuronal degut a que depèn del paràmetre. Al ser les xarxes neuronals funcions molt complexes, s'utilitza una tècnica concreta per efectuar la derivada de la xarxa neuronal anomenada *backpropagation*.

3.1 Backpropagation

Al 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 contraria. Mentre que quan es dona un input a xarxa, s'anomena forward propagation o *forward pass*.

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

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

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

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 que mesura es connecta la neurona j a la neurona k. Com ja he dit, expressa com de forta és aquesta connexió. Utilitzant aquesta definició ja es poden fer les derivades. La suma es fa al llarg de n_{L-1} que és el nombre de neurones que té la capa $L-1$.

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

$$z_j^L = \sum_{j=0}^{n_L-1} w_{jk}^L a_k^{L-1} + b_j^L$$

L'objectiu és obtenir la derivada de la *loss function* respecte a un *weight* qualsevol i un *bias* qualsevol. Per tant, s'han 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*, a aplicar la regla de cadena obtenim que:

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^L} = \frac{\partial z_k^L}{\partial w_{jk}^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial \mathcal{L}}{\partial a_j^L} \tag{3.2}$$

L'última derivada, 'la que està més afora' és simplement la derivada de la funció de pèrdua respecte al output de la xarxa. En el cas de la funció de pèrdua *Squared Error*, és[5]:

$$\frac{\partial \mathcal{L}}{\partial a_j^L} = \frac{\partial}{\partial a_j^L} \left(a_j^L - y\right)^2 = 2(a_j^L - y)$$

On y és la predicció desitjada del model. A continuació, la derivada del resultat de la neurona a_j^L respecte a z_j^L, que és la derivada la funció d'activació.

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

Per últim tenim la derivada de z_j^L respecte al *weight*:

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

La derivada és la neurona anterior, cal recordar que el *weight* lo que fa es establir la connexió entre dues neurones, les neurones j i k.

Finalment es pot veure que l'equació 3.2 es pot escriure com[6]:

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

No obstant falta un detall, la derivada d'una neurona que passa el seu resultat a diverses neurones. Aquesta derivada és:

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

La suma representa que aquesta neurona té un output que es propaga cap a endavant i afecta a les neurones que estan més endavant.

[5] No aplico la regla de la cadena en aquesta derivació perquè ja es té en compte en l'equació 3.2

[6] Deixo l'ultima 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.

Amb aquestes derivades ja es pot desenvolupar el vector gradient, en el qual estan totes les derivades de la *loss function* respecte a tots els paràmetres. Concretament, la mitjana d'aquestes derivades, perquè es vol actualitzar els paràmetres per poder minimitzar la pèrdua en totes les dades disponibles. En altres paraules, si es vol que una xarxa reconegui imatges de gats, se li té que ensenyar moltes imatges de gats. Si només se li ensenya una, només aprendrà a reconèixer aquella imatge.

Tota aquesta teoria es veurà implementada en la part pràctica en forma de codi, degut a que m'he vist amb la necessitat de tenir una xarxa neuronal programada des de zero. Usualment s'utilitzen plataformes com *TensorFlow* [25] o *PyTorch* [26].

3.3 Generative adversial networks

Com ja he dit hi han molts tipus de xarxes neuronals, no obstant, en aquest treball només em centraré en un tipus en específic, les xarxes generatives adversatives o *generative adversial networks (GAN)* en anglès.

Aquestes xarxes, com el seu nom diu, s'utilitzen per generar dades, usualment s'apliquen a imatges. Es troben al darrera de projectes com *This person does not exist* [27, 28], una pàgina web que et genera una cara d'una persona que no existeix, degut a que es una cara generada artificialment a partir d'aquest tipus de models.

Aquests tipus de models van ser introduïts per primera vegada al 2014 per Ian Goodfellow [4], des de llavors s'han convertit en un dels models de *deep learning* més sòlids i utilitzats.

Aquests algoritmes consisteixen en dos models (xarxes) diferents, un generador i un discriminador, amb objectius oposats que s'enfrenen entres si. Per aquesta raó tenen la paraula adversatives en el nom. El generador i discriminador es poden entendre com uns falsificador de bitllets i uns policies que els volen atrapar, respectivament. Els policies es tornen millors al seu treball, podent distingir millor entre els bitllets falsificats i els reals. Els falsificadors responen a això millorant les seves tècniques de falsificació, per tant els policies han de millor encara més. Es un cicle en el qual aquestes forces antagonistes es fan

millorar l'una al altra. El mateix passa amb el generador i el discriminador. El discriminadors 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, arribem a *zero sum game*[7] de teoria de joc. La manera en la que es soluciona es al arribar a un equilibri de Nash [5], on el discriminador no sap diferenciar entre les imatges reals i les falses[8].

En el paper original [4] s'esmenta un pseudocodi per aquests models, que he representat en l'algoritme 1. En aquest és parla de *minibatch* que és simplement un grup d'imatges o de dades, i de soroll que són dades generades aleatòriament i que es poden 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 en una petita quantitat.

Algorithm 1 Pseudocodi per una xarxa generativa adversativa

for número de interaccions **do**
  for k pasos **do**
    Treure minibatch de m mostres de soroll $\{z_i, \cdots, z_m\}$ de la distribució de soroll $p_g(z)$
    Treure minibatch de m mostres d'exemples $\{x_i, \cdots, x_m\}$ 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
  Treure minibatch de m mostres de soroll $\{z_i, \cdots, z_m\}$ 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 discriminadors k vegades i després el generador una sola vegada. Això és perquè interessa que el

[7] Un *zero sum game* es simplement un joc entre dos jugadors en que per guanyar un l'altre ha de perdre e.g. joc d'estirar la corda entre dos equips.

[8] Que el discriminador no sàpiga diferenciar no implica arribar a un equilibri de Nash, aquest concepte es definit d'una altra forma.

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, perquè ell aprèn a partir del que indica el discriminador.

També és pot veure com és la funció de pèrdua, que en el paper anomenen *minimax loss function* que és la mateixa que la *Binary Cross Entropy*. Es pot veure que en la BCE, quan t és zero, que seria l'etiqueta per les imatges falses del generador dona $\log(1-y)$. En el cas contrari, per t igual a un, que expressa que les imatges són reals dona $\log(y)$.

Capítol 4

Generació d'imatges amb un ordinador quàntic

Investigadors al veure el potencial que tenen els ordinadors quàntics i l'intel·ligència artificial, no es van poder resistir a crear un nou camp d'investigació, el *Quantum Machine Learning (QML)*, o aprenentatge automàtic quàntic. Al igual que les xarxes neuronals són les estrelles dintre del *machine learning*, les xarxes neuronals quàntiques ho són dintre del *quantum machine learning*. Des de que es va començar a investigar aquests algoritmes s'han arribat a implementar diversos tipus de xarxes neuronals en algoritmes quàntics. Principalment pel que fa a la generació i classificació d'imatges i dades.

No obstant aquests algoritmes no són completament quàntics, usualment consisteixen en 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 veure molt clar quan expliqui la part pràctica del treball.

Abans de continuar amb el capítol 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] o amb un anàleg de la regressió lineal [31]. No obstant, en aquest capítol em centraré exclusivament en les xarxes neuronals quàntiques.

[1] Classificar dades de dimensions petites en espais vectorials molt grans [29, 30].

4.1 Descens del gradient quàntic

De moment tindre en consideració que una xarxa neuronal quàntica és un circuit quàntic parametritzat, és a dir que simplement té paràmetres. Això juntament amb una funció de pèrdua i dades que corresponen a aquesta es suficient per explicar com s'actualitzen els paràmetres.

L'objectiu principal es avaluar la derivada respecte amb un paràmetre. Sorprenentment és dona a terme d'una manera més senzilla que amb una xarxa neuronal clàssica. Simplement s'utilitza la definició de la derivada per avaluar-la. S'altera lleugerament un sol paràmetre i es treu la diferencia entre els dos outputs del circuit quàntic. Aquest mètode per avaluar la derivada s'anomena *parameter shift* [2, 32].

Si un circuit quàntic té paràmetres θ representats per un vector, per un paràmetre θ_i, es defineix un vector de pertubació Δ_i ple de zeros, de igual mida que θ, però que en la posició de θ_i té un 1. A partir del vector de pertubació es pot definir la derivada de la funció de pèrdua $\mathcal{L}(\cdot)$ respecte al paràmetre θ_i:

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

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 es la que correspon al paràmetre θ_i. Amb aquest mètode ja es pot desenvolupar pràcticament qualsevol actualització de paràmetres, aquesta es la part senzilla de les xarxes neuronals quàntiques, la dificultat radica en la forma dels circuits quàntics que les componen.

4.2 Circuits quàntics per xarxes neuronals

L'única certesa sobre aquests circuits es que han de 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 que esmenten en els articles sobre xarxes neuronals quàntiques es que s'han d'implementar funcions no lienals en els circuits, degut a que la complexitat de les xarxes neuronals radica en aquest tipus de funcions. A primera

vista por semblar una tasca quasi impossible a causa de la naturalesa lienal de la computació quàntica. No obstant, durant els anys s'han desenvolupat diverses tècniques per donar a terme aquesta fita.

A partir d'una combinació de rotacions i portes que entrellacen qubits es pot arribar a implementar una funció semblant a la tangent hiperbòlica[2] en un circuit quàntic [33]. No obstant, 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* degut a que s'ha de mesurar un qubit i mirar si dona $|0\rangle$ per assegurar que aquesta funció ha sigut aplicada correctament[3].

En un article posterior[4], en el qual els autors generen distribucions continues a partir d'una xarxa generativa adverbial quàntica. S'especifica que les no-linearitats presents en l'algoritme no formen part del circuit quàntic, es a dir, funcions que s'implementen clàssicament als resultats dels circuits quàntics o a les dades que s'introdueixen als circuits [34]. Aquesta és una mesura molt simple que possiblement implementaré durant la part experimental.

Al 2019 es va publicar un dels articles que més m'agraden[5] Cong at. al (2019) [35], en ell els autors presenten una xarxa neuronal convolucional quàntica. No fa falta entrar en detall sobre aquestes xarxes. Però he de dir que es diuen convolucionals perquè s'apliquen convolucions a les imatges que es volen classificar, es multiplica una part de l'imatge per una matrius que s'anomena filtre [36]. 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 es degut als mesuraments parcials que es donen a terme en un moment determinat del algoritme.

Un dels mètodes que m'ha semblat més interessant, és l'implementat en una altra xarxa convolucional quàntics[6], on al aplicar els filtres que s'utilitzen per la convolució s'implementa una funció no-lienal. Aquest mètode no el puc arribar

[2] La tangent hiperbòlica també s'utilitza com a funció no-lienal en el *deep learning*.

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

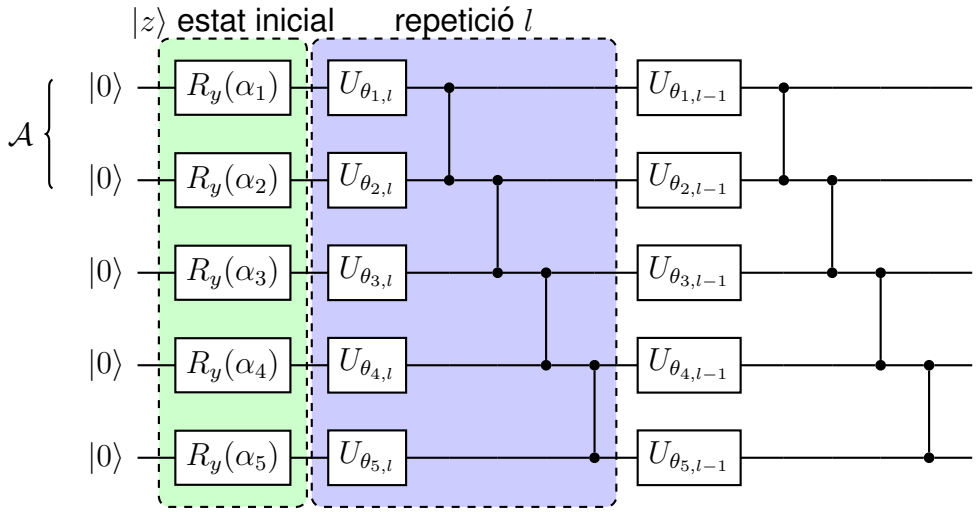
[4] Realitzat en part per un dels autors del mètode anterior

[5] Utilitzen 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 [2], i això sempre s'agraeix.

[6] En l'article també utilitzen l'imatge d'un gat com a exemple a les il·lustracions. Es que els gats i els físics tenen una llarga historia.

a comprendre, les equacions que utilitzen són molt complexes i no sembla que arriben a utilitzar un mesurament parcial en cap moment. Simplement els autors expressen una equació i diuen que no és lienal, i no puc arribar a entendre perquè ho és.

Finalment he de parlar del paper que he seguit per realitzar aquest treball, Huang et. al. (2021) [5], en el al igual que en l'article de Cong et. el. (2019) sobre xarxes convolucionals quàntiques, s'utilitzen mesuraments quàntics per introduir no-linearitats al algoritme. Concretament implementen aquests mesuraments tant en el generador quàntic d'imatges, com en el discriminador. Els circuits quàntics que utilitzen pel generador d'imatges són de la següent forma:



On les portes R_y amb un paràmetre α_i són les dades que s'introdueixen en el circuit, és simplement soroll que crea varietat en els outputs dels circuits, al igual que el soroll que s'utilitza en les GAN (algoritme 1). El circuit consisteix en aquestes portes R_y i la repetició-l amb unitàries i les portes CZ. En aquest cas el circuit té dues repeticions-l, amb $l = 2$. Al hora de mesurar és fa un mesurament parcial però els autors especifiquen que es fa d'una forma concreta.

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

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

Els autors afirmen que aquest estat $\rho(\alpha)$ és una funció no lineal del estat $|z\rangle$ que es defineix per les primeres portes R_y que s'apliquen en el circuit[7]. Això es degut a que tant el denominador com el numerador de l'equació són funcions de $|z\rangle$.

No obstant, en altres treball com per exemple Zoufal et. al. (2019) [37], on es defineix una GAN que a partir de circuits quàntics genera distribucions de probabilitat. Els autors no mencionen la necessitat de tenir funcions no lineals en alguna part del algoritme.

[7] En la figura (-), aquestes operacions estan indicades de color verd.

Part II

Part Experimental

Capítol 5

Plantejament de l'hipòtesi

Al ser l'implementació de les funcions no-lineal un assumpte lleugerament conflictiu entre els diversos models de xarxes neuronals quàntiques havia decidit des de un principi centrar-me en aquest assumpte en concret. Podria haver anat per al altres vies com la generació d'imatges amb color o l'implementació d'un d'una porta X en una part especifica del circuit quàntic que genera les imatges, que al posar-la o no, el model generes dos tipus d'imatge a través del mateix circuit. Tanmateix, les dues propostes requerien de desenvolupar nous conceptes, llavors al no tindre el coneixements necessaris per poder desenvolupar-les vaig descartar aquestes opcions.

La pregunta a investigar és la següent:
L'incorporació d'una funció no-lineal en el circuit quàntic del model, a partir d'una mesura parcial, causa que el model arribi més ràpidament al punt òptim?

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

Sembla una qüestió senzilla, però la dificultat del 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-ho seria programant el model.

Capítol 6

Programació del model

Posteriorment a començar a programar el model, jo ja sabia que ho havia de realitzar en Python, ja creat petits algoritmes abans de començar aquest treball i tenia experiència construint i executant circuits quàntics amb Cirq, una eina desenvolupada per Google. A més a més, sabia de l'existència de TensorFlow Quantum [2], una altra eina desenvolupada per Google destinada a la creació de xarxes neuronals quàntiques i algoritmes de *quantum mechine learning* en general. També tenia experiència en aquest *framework*. Llavors TensorFlow Quantum va ser la meva primera opció, tenia pensat basar el meu codi en el tutorial de TensorFlow sobre un xarxa convolucional generativa adverbial. Havia de canviar el generador per un circuit quàntic que s'obtimitza a partir d'un diferenciador automàtic provinent de TensorFlow Quantum. Els canvis que corresponien al discriminador simplement havien de ser un canvi d'arquitectura, passar d'una xarxa més complexa a una de més simple que només consistia en unes poques capes totalment connectades.

El primer problema que em vaig trobar va ser la creació del *data set* que alimentar a la xarxa discriminativa. A causa de la peculiaritat de les imatges que volia generar[1]. Recordo que en va costar arribar a tindre la solució a aquest problema.

Una vegada ja tenia fet el *dataset* em vaig posar a fer el model. En el tutorial per una DCGAN (GAN conolucional) els dos models eren entrenats per la

[1] Usualment les imatges que componen els *datasets* utilitzats en *deep machine learning* són extretes 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* per TensorFlow.

funció `train_step`. En aquesta es crida a la funció `tf.GradientTape` per guardar el diferenciador automàtic. El problema que vaig dintre amb aquesta funció es que directament no funcionava amb el discriminador, aquest no era optimitzat. Després de intentar solucionar l'error pels meus propis medis, mirant la causa d'aquest i de buscar a forums 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 el optimitzador. Tenia el model gairebé acabat, però al no poder optimitzar el discriminador em vaig veure obligat a canviar d'estrateiga.

Existeixen dos grans *frameworks* per crear i executar circuits quàntics: Cirq [38], desenvolupat per Google i Qiskit [39], creat per IBM. Una vegada vaig decidir no continuar amb Cirq, havia de provar amb Qiskit. La veritat, havia d'haber començar amb Qiskit des de el principi, és més útil (té moltes més característiques), i el més important, té una gran comunitat, per tant, es més fàcil trobar solucions als error que pots tenir i és més fàcil trobar a persones disposades a ajudar-te.

Al igual que Cirq té un *framework* per poder desenvolupar xarxes neuronals, TensorFlow Quantum, Qiskit també té el seu PyTorch [26], no obstant no té una integració tan directa, ja que no estan desenvolupats per el mateix equip, ni la mateixa companyia.

Per tant, al canviar de Cirq a Qiskit, també havia de canviar de TensorFlow a PyTorch, no obstant, no tenia res d'experiencia amb PyTorch, sabia que seria més complicat, i tenia raó. No vaig ni aconseguir crear el *dataset* que contenia les imatges que alimentar al discriminador.

Després d'intetar-lo amb TensorFlow Quantum i amb PyTorch, vaig decidir prescindir de *frameworks* per crear models de *machine learning*, el discriminador, al ser una xarxa tan simple, la podia crear des de zero. Llavors vaig començar a buscar codi, d'una xarxa neuronal que havia estat feta amb Numpy, una llibreria de Python per fer càlculs amb vectors i matrius que havia fer servir des de que vaig començar amb Python.

Després de provar dues opcions que més o menys m'agradaven[2], va aparèixer un repositori[3] de Michael Nielsen, un dels autors de *Quantum Computation and Quantum Information* [8] que em va salvar.

[2]Busca codi estructurat d'una forma en concret, que estigui dissenyat amb *Object Oriented Programming*, una forma d'escriure codi en el qual tot s'implementa en un objecte.

[3]link del respositori

Era codi que estava estructurat d'una forma que m'agradaba i encara més important, que l'entenia. Inclús tènia diverses versions d'una xarxa neuronal, amb un nivell de complexitat diferent. Llavors, a partir del model més simple que havia en el repositori, veure el codi original al apèndix E.1.0.1, vaig començar a desenvolupar el discriminador.

Com es pot veure al codi final per el discriminador, he fet bastants canvis, però ho he canviat l'estructura o el funcionament teòric, la majoria són per tindre més funcionalitat, com per exemple l'enmagatzematge de les dades per poder al final veure-les en gràfic. Però el canvi més important és que inclou les dues formes d'optimitzar el model, amb dues funcions de pèrdua que funcionen de manera diferents però que són el mateix, la *Binary Cross Entropy* i la *MinMax*. Ja he parlat d'aquestes dues funcions en la part teòrica del treball, però a mode de recordatori, amb la primera s'agafa una imatge falsa o una real, i s'altera la funció, mentre que amb la *MinMax* depenent de si es una imatge falsa o una real, s'utilitza una funció diferent. Això en el codi està materialitzat en dues funcions[4] diferents, `backprop_bce()` i `backprop_minimax()`. No fa falta entrar en detall sobre que va cadascuna de les funcions, degut a que, fan el mateix i no tenen ninguna diferencia en termes de eficàcia o rapidesa. Les vaig fer tan sols per comprovar que no hi hauria ninguna diferencia. El codi final del discriminadors es pot veure al apèndix E.1.0.2. També en el repositori del treball hi ha un altre arxiu anomenat `discriminator.py`, que conté una altre versió en la qual intentava implementar diverses funcions d'activació en el model, però no ho vaig aconseguir, tanmateix, no em preocupa perquè no es una part vital del treball, no passa res per tindre el discriminador només amb la sigmoide.

El desenvolupament de l'altre part del model, el generador, va ser molt diferent. Després de provar a fer-ho amb TensorFlow, sense resultats, no tenia altra opció que fer-lo tot manualment i jo mateix[5], no obstant, ja tenia experiència en fer pseudo-xarxes neuronals quàntics. Sabia perfectament que tenia que fer, i com ho havia de fer. Implementar el mètode de *parameter shift* en els circuits quàntics de l'article en el que vaig basar el treball [5]

[4] Funcions de Python.

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

Capítol 7

Realització del experiment

Part III

Conclusions

Part IV

Apèndix

Appendices

Apèndix A

Més àlgebra lineal

Ja he escrit bastants pàgines sobre àlgebra lineal, però aparentment no eren les suficients perquè estic content amb el treball[1], així que aquí hi ha més àlgebra lineal.

A.1 Procediment de Gram–Schmidt

El procediment de Gram-Schmidt és un mètode utilitzat per produir bases per a espais vectorials [9]. Per un espai V amb producte interior de d dimensions amb el set de vectors $|v_1\rangle, \cdots, |v_d\rangle$, podem definir una nova base de vectors ortonormals $\{|u\rangle\}$. El primer element d'aquest set és $|u_1\rangle = |v_1\rangle / \||v_1\rangle\|$, amb el següent element $|v_{k+1}\rangle$ sent:

$$|u_{k+1}\rangle = \frac{|v_{k+1}\rangle - \sum_{i=1}^{k} \langle u_i|v_{k+1}\rangle |u_i\rangle}{\left\||v_{k+1}\rangle - \sum_{i=1}^{k} \langle u_i|v_{k+1}\rangle |u_i\rangle\right\|}$$

Per k en el interval $1 \leq k \leq d-1$.

Si seguim per k en $1 \leq k \leq d-1$, obtenim el set de vectors $|u_1\rangle, \cdots, |u_d\rangle$ que es una base vàlida per l'espai ortonormal per l'estai V. Els vectors creats han de tindre el mateix span[2] que el dels vectors que originalment eren la base per V:

$$\text{span}(\{|v\rangle\}) = \text{span}(\{|v\rangle\}) = V$$

[1] Hi han moltes coses guays i interessants que vull explicar.

[2] L'span d'un set de vectors són totes les combinacions lineals possibles amb aquests vectors.

Cal notar que l'span de del set base és la definició del espai. En altres paraules, cada vector en V pot ser representat per una combinació del vectors base.

La prova de que és una base ortonormal és bastant simple: Podem veure immediatament que els elements de $\{|u\rangle\}$ són vectors unitaris perquè estan normalitzats (els vectors $|v_{k+1}\rangle - \sum_{i=1}^{k} \langle u_i|v_{k+1}\rangle |u_i\rangle$ estan dividits per la seva norma). També podem veure que són ortogonals els uns als altres mirant que el producte interior entre els doni 0:

Per $k = 1$:

$$|u_2\rangle = \frac{|v_2\rangle - \langle u_1|v_2\rangle |u_1\rangle}{\||v_2\rangle - \langle u_1|v_2\rangle |u_1\rangle\|}$$

Per tant el producte interior amb $|v_1\rangle$ és:

$$\begin{aligned}
\langle u_1|u_2\rangle &= \langle u_1| \left(\frac{|v_2\rangle - \langle u_1|v_2\rangle |u_1\rangle}{\||v_2\rangle - \langle u_1|v_2\rangle |u_1\rangle\|} \right) \\
&= \frac{\langle u_1|v_2\rangle - \langle u_1|v_2\rangle \langle u_1|u_1\rangle}{\||v_2\rangle - \langle u_1|v_2\rangle |u_1\rangle\|} \\
&= 0
\end{aligned}$$

Per inducció podem veure que per $j \leq d$, amb d sent la dimensió del espai vectorial:

$$\begin{aligned}
\langle u_j|u_{n+1}\rangle &= \langle u_j| \left(\frac{|v_{n+1}\rangle - \sum_{i=1}^{n} \langle u_i|v_{n+1}\rangle |u_i\rangle}{\||v_{n+1}\rangle - \sum_{i=1}^{n} \langle u_i|v_{n+1}\rangle |u_i\rangle\|} \right) \\
&= \frac{\langle u_j|v_{n+1}\rangle - \sum_{i=1}^{n} \langle u_i|v_{n+1}\rangle \langle u_j|u_i\rangle}{\||v_{n+1}\rangle - \sum_{i=1}^{n} \langle u_i|v_{n+1}\rangle |u_i\rangle\|} \\
&= \frac{\langle u_j|v_{n+1}\rangle - \sum_{i=1}^{n} \langle u_i|v_{n+1}\rangle \delta_{ij}}{\||v_{n+1}\rangle - \sum_{i=1}^{n} \langle u_i|v_{n+1}\rangle |u_i\rangle\|} \\
&= \frac{\langle u_j|v_{n+1}\rangle - \langle u_j|v_{n+1}\rangle}{\||v_{n+1}\rangle - \sum_{i=1}^{n} \langle u_i|v_{n+1}\rangle |u_i\rangle\|} \\
&= 0
\end{aligned}$$

Tot això no és veu molt clar al principi però cal recordar que el producte interior de dos vectors ortonormals és zero, i que el producte interior entre el mateix vector unitari és un.

A.2 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[3] [40]

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 *ket*
$\|\psi\rangle^T$	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 *bra*
$\langle\varphi\|\psi\rangle$	Producte interior dels vectors $\langle\varphi\|$ i $\|\psi\rangle$
$\|\varphi\rangle\langle\psi\|$	Producte exterior del vectors $\langle\varphi\|$ i $\|\psi\rangle$
$\|\psi\rangle \otimes \|\varphi\rangle$	Producte tensorial del 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 del nombres complexos

A.3 Més on la traça parcial

[3] 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.

Apèndix B

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 apèndix explicaré exactament a que em refereixo per això amb un exemple pràctic. Dic que és fàcil, perquè si ho és, si es compara la computació quàntica amb la mecànica quàntica. Presentaré un exemple pràctic per il·lutrar-lo.

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 es 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, treballar amb elles és molt més complicat degut a que són funcions que depenen del temps, no son vectors. 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 normalitzar una funció d'ona i un vectors d'estat.

B.1 Normalitzar

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

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

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

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 normalitza seguint la fórmula:

$$\int_{-\infty}^{+\infty} |\Psi(x,t)|^2\, dx \; = 1 \tag{B.1}$$

Degut a que la funció d'ona evoluciona a través del temps d'acord amb l'equació de Schrödinger, veure la figura B.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 [41].

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

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

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 l'equació de Schrödinger diu que

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

després calculant el complex conjugat tenim que

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

per tant

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

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

finalment podem avaluar l'integral del principi:

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

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

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

Es pot veure que l'integral es constant i per tant quan Ψ es normalitzada a $t=0$, es queda d'aquesta manera per qualsevol t (positiu es clar).

Apèndix C

Polarització d'un fotó

En l'equació 2.1 he excluit el concepte de fase, que determina el tipus de polarització que té un fotó. Hi han 3 tipus:

1. **Linear:** Un fotó té polarització lineal quan els angles de la fase α_x, α_y en els estats base $|x\rangle, |y\rangle$ són iguals:

$$\begin{aligned} |\nearrow\rangle &= \cos(\theta)e^{i\alpha_x}|x\rangle + \cos(\theta)e^{i\alpha_y}|y\rangle \\ &= [\cos(\theta)|x\rangle + \sin(\theta)|y\rangle]e^{i\alpha} \end{aligned}$$

   On $\alpha = \alpha_x = \alpha_y$.

2. **Circular:** Quan els angles α_x, α_y son separats per exactament $\frac{\pi}{2}$ i la amplitud per les dos bases és la mateixa:

$$\begin{aligned} |\nearrow\rangle &= \frac{1}{\sqrt{2}}\cos(\theta)e^{i\alpha_x}|x\rangle \pm i\frac{1}{\sqrt{2}}\sin(\theta)e^{i\alpha_y}|y\rangle \\ &= [\cos(\theta)e^{i\alpha_x}|x\rangle \pm i\sin(\theta)e^{i\alpha_y}|y\rangle]\frac{1}{\sqrt{2}} \end{aligned}$$

   On el signe $\pm$ indica la diferencia entre la diferencia entre la polarització circular cap a la dreta o la esquerra, amb $+$ i $-$, respectivament.

3. **El·líptica:** On els angles α_x, α_y son diferents per una quantitat arbitraria[1]:

$$|\nearrow\rangle = \cos(\theta)e^{i\alpha_x}|x\rangle + \sin(\theta)e^{i\alpha_y}|y\rangle$$

   Aquest és el cas més general.

[1] Però que no sigui la quantitat que dona a terme la polarització circular.

Apèndix D

Complexitat i algoritmes quàntics

En ciència de la computació existeix el concepte de *Big-O Notation*, una forma d'expressar lo eficients que són els algoritmes per fer certes tasques, en altres paraules la complexitat dels algoritmes. Bàsicament es una forma de classificar-los segon la rapidesa que tenen en ver la tasca que els correspon, aquesta rapidesa no és mesura en segons, degut a que aquesta mètrica pot variar d'ordinador a ordinador per les diferencies en hardware que aquest poden tindre. En canvi es mesure en nombre d'operacions o temps directament, però sense unitats.

La *Big-O Notation* consisteixes en definir el temps màxim que necessita un algoritme, es denota com $O(\cdot)$ on l'argument usualment depèn de n que és la mida del input al algoritme, per exemple un algoritme de cerca ha de cercar a través de n coses. Com a un exemple més concret tenim que un l'algoritme de cerca de cadenes binaries corre en un temps $O(\log_2 n)$, on n és el nombre de cadenes entre les quals ha de cercar. Recorda que la notació $O(\cdot)$ és el màxim, es a dir es *upper bounded*, això significa que $\log_2 n$ és la quantitat de temps més gran en la que es troba la cadena, també es possible que es trobi-s'hi a la primera comprovació que es va[1], llavors l'algoritme acabaria en un temps $O(1)$. Simplement és una manera de mirar lo eficients que són els algoritmes en relació a la mida del input que tenen.

Amb aquesta notació tenim una manera de comparar la eficiència que tenen els algoritmes quàntics amb la del clàssics que tenen la mateixa funció.

[1] Que la primera cadena que es cerca, és la que s'ha de trobar.

D.1 Algoritme de Grover

Al 1996, Lov Grover va presentar un algoritme quàntic per cercar en dades desordenades [42] (e.g. cercar el número de telèfon d'una persona en una llista desordenada). Per aquest problema un algoritme clàssic té una complexitat de $O(N)$ cerques[2], mentre que l'algoritme de Grover té una complexitat de $O(\sqrt{N})$, sent substancialment més eficient. En les paraules de Grover [42] (adaptades), un ordinador clàssic per tindre un probabilitat de $\frac{1}{2}$ de trobar el número de telèfon d'una persona en una llista desordenada necessita mirar a un mínim de $\frac{N}{2}$ números, mentre que amb el seu s'obté el número de telèfon en només $O(\sqrt{N})$ passos[3].

L'algoritme funciona de la següent manera:

[2] Una cerca és quan es verifica si un element de la llista és l'element que es cerca.

[3] Per passos entenc que es refereix al nombre de vegades que es mira al oracle, es a dir el nombre que de vegades que es verifica si s'ha trobar el que es cerca.

Apèndix E

Codi

En l'apèndix 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.

E.1 Part I

E.1 Capítol 3

E.1.0.1 Regressió lienal Codi per efectuar una regressió lineal a dades que es generen al atzar en el mateix arxiu, l'utilitzo per poder generar una gràfic per il·lustrar un exemple de regressió lienal. Aquest troç de codi l'he tret de GitHub[1].

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib

font = {'family' : 'Helvetica',
  'size'   : 18}

matplotlib.rc('font', **font)

# generate the data
np.random.seed(222)
X = np.random.normal(0,1, (200, 1))
```

[1] Gist realitzat per l'usuari *jimimvp*: link

```
  w_target = np.random.normal(0,1, (1,1))
  # data + white noise
  y = X@w_target + np.random.normal(0, 1, (200,1))

  # least squares
  w_estimate = np.linalg.inv(X.T@X)@X.T@y
  y_estimate = X@w_estimate

  # plot the data
  plt.figure(figsize=(15,10))
  plt.scatter(X.flat, y_estimate.flat, label="Prediccio")
  plt.scatter(X.flat, y.flat, color='red', alpha=0.4, label="Dades"
   )
  plt.tight_layout()
  plt.title("Regressio per diferencia de quadrats")
  plt.legend()
  plt.savefig("least_squares.png")
  plt.show()
```

Listing E.1: Regressió lineal

E.2 Part II

E.1 Capítol 6

E.1.0.1 Codi original per la xarxa neuronal clàssica Està extret del repositori de GitHub de Micheal Nielsen, concretament del arxiu `network.py`.

```
"""
network.py
~~~~~~~~~~
A module to implement the stochastic gradient descent learning
algorithm for a feedforward neural network.  Gradients are
   calculated
using backpropagation.  Note that I have focused on making the code
simple, easily readable, and easily modifiable.  It is not
   optimized,
and omits many desirable features.
"""

#### Libraries
# Standard library
```

```
import random

# Third-party libraries
import numpy as np

class Network(object):

  def __init__(self, sizes):
    """The list ``sizes`` contains the number of neurons in the
    respective layers of the network.  For example, if the list
    was [2, 3, 1] then it would be a three-layer network, with the
    first layer containing 2 neurons, the second layer 3 neurons,
    and the third layer 1 neuron.  The biases and weights for the
    network are initialized randomly, using a Gaussian
    distribution with mean 0, and variance 1.  Note that the first
    layer is assumed to be an input layer, and by convention we
    won't set any biases for those neurons, since biases are only
    ever used in computing the outputs from later layers."""
    self.num_layers = len(sizes)
    self.sizes = sizes
    self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
    self.weights = [np.random.randn(y, x) for x, y in zip(sizes
   [:-1], sizes[1:])]

  def feedforward(self, a):
    """Return the output of the network if ``a`` is input."""
    for b, w in zip(self.biases, self.weights):
      a = sigmoid(np.dot(w, a)+b)
    return a

  def SGD(self, training_data, epochs, mini_batch_size, eta,
   test_data=None):
    """Train the neural network using mini-batch stochastic
    gradient descent.  The ``training_data`` is a list of tuples
    ``(x, y)`` representing the training inputs and the desired
    outputs.  The other non-optional parameters are
    self-explanatory.  If ``test_data`` is provided then the
    network will be evaluated against the test data after each
    epoch, and partial progress printed out.  This is useful for
    tracking progress, but slows things down substantially."""
    if test_data: n_test = len(test_data)
    n = len(training_data)
    for j in xrange(epochs):
      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     return sum(int(x == y) for (x, y) in test_results)
120
121   def cost_derivative(self, output_activations, y):
122     """Return the vector of partial derivatives \partial C_x /
123     \partial a for the output activations."""
124     return (output_activations-y)
125
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.1.0.2 Codi final per el discriminador

Aquest codi es pot trobar al repositori del treball en l'arxiu `discriminator_functional.py`.

```
"""DISCRIMINATOR"""
import json
from typing import Dict, List

import numpy as np

from quantumGAN.functions import BCE_derivative,
    minimax_derivative_fake, minimax_derivative_real, sigmoid,
    sigmoid_prime


def load(filename):
  f = open(filename, "r")
  data = json.load(f)
  f.close()
  # cost = getattr(sys.modules[__name__], data["cost"])
  net = ClassicalDiscriminator_that_works(data["sizes"], data["loss
    "])
  net.weights = [np.array(w) for w in data["weights"]]
  net.biases = [np.array(b) for b in data["biases"]]
  return net


class ClassicalDiscriminator_that_works:

  def __init__( self,
          sizes: List[int],
          type_loss: str) -> None:

    self.num_layers = len(sizes)
    self.sizes = sizes
    self.type_loss = type_loss
    self.data_loss = {"real": [], "fake": []}
    self.ret: Dict[str, any] = {"loss": [],
      "label real": [],
      "label fake": [],
      "label fake time": [],
      "label real time": []}
    self.biases = [np.random.randn(y, ) for y in sizes[1:]]
    self.weights = [np.random.randn(y, x) for x, y in zip(sizes
    [:-1], sizes[1:])]

```

```
  def feedforward(self, a):
    """Return the output of the network if ``a`` is input."""
    for b, w in zip(self.biases, self.weights):
a = sigmoid(np.dot(w, a) + b)
return a

def predict(self, x):
# feedforward
activation = x
zs = []  # list to store all the z vectors, layer by layer
for b, w in zip(self.biases, self.weights):
z = np.dot(w, activation) + b
zs.append(z)
activation = sigmoid(z)
return activation

def evaluate(self, test_data):

test_results = [(np.argmax(self.feedforward(x)), y)
for (x, y) in test_data]
return sum(int(x == y) for (x, y) in test_results)


def forwardprop(self, x: np.ndarray):
activation = x
activations = [x]  # list to store all the activations, layer by
    layer
zs = []  # list to store all the z vectors, layer by layer
for b, w in zip(self.biases, self.weights):
z = np.dot(w, activation) + b
zs.append(z)
activation = sigmoid(z)
activations.append(activation)
return activation, activations, zs

def backprop_bce(self, image, label):
"""Return a tuple ``(nabla_b, nabla_w)`` representing the
gradient for the cost function C_x.  ``nabla_b`` and
``nabla_w`` are layer-by-layer lists of numpy arrays, similar
to ``self.biases`` and ``self.weights``."""
nabla_b = [np.zeros(b.shape) for b in self.biases]
nabla_w = [np.zeros(w.shape) for w in self.weights]

# feedforward and back error calculation depending on type of image
activation, activations, zs = self.forwardprop(image)
```

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

```
for l in range(2, self.num_layers):
z = zs[-l]
delta = np.dot(self.weights[-l + 1].transpose(), delta) *
    sigmoid_prime(z)
nabla_b[-l] = delta
nabla_w[-l] = np.dot(delta.reshape(delta.shape[0], 1),
activations[-l - 1].reshape(1, activations[-l - 1].shape[0]))
return nabla_b, nabla_w, activations[-1]

def train_mini_batch(self, mini_batch, learning_rate):
global label_real, label_fake
nabla_b = [np.zeros(b.shape) for b in self.biases]
nabla_w = [np.zeros(w.shape) for w in self.weights]

if self.type_loss == "binary cross entropy":
for real_image, fake_image in mini_batch:
delta_nabla_b, delta_nabla_w, label_real = self.backprop_bce(
    real_image, np.array([1.]))
nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]

delta_nabla_b, delta_nabla_w, label_fake = self.backprop_bce(
    fake_image, np.array([0.]))
nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]

elif self.type_loss == "minimax":
for real_image, fake_image in mini_batch:
delta_nabla_b, delta_nabla_w, label_real = self.backprop_minimax(
    real_image, fake_image, True)
nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]

delta_nabla_b, delta_nabla_w, label_fake = self.backprop_minimax(
    real_image, fake_image, False)
nabla_b = [nb + dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
nabla_w = [nw + dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
else:
raise Exception("type of loss function not valid")

# gradient descent
# nabla_w and nabla_b are multiplied by the learning rate
# and taken the mean of (dividing by the mini batch size)
self.weights = [w - (learning_rate / len(mini_batch)) * nw
for w, nw in zip(self.weights, nabla_w)]
```

```
159 self.biases = [b - (learning_rate / len(mini_batch)) * nb
160 for b, nb in zip(self.biases, nabla_b)]
```

Listing E.3: Codi final pel discriminador

Bibliografia

[1] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13:2567–2586, 2014. [Online; accedit en 19/09/2021: Link].

[2] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, Ramin Halavati, Murphy Yuezhen Niu, Alexander Zlokapa, Evan Peters, Owen Lockwood, Andrea Skolik, Sofiene Jerbi, Vedran Dunjko, Martin Leib, Michael Streif, David Von Dollen, Hongxiang Chen, Shuxiang Cao, Roeland Wiersema, Hsin-Yuan Huang, Jarrod R. McClean, Ryan Babbush, Sergio Boixo, Dave Bacon, Alan K. Ho, Hartmut Neven, and Masoud Mohseni. TensorFlow Quantum: A software framework for quantum machine learning. 2021. [Online; accedit en 19/09/2021: Link].

[3] IBM quantum, 2021. [Online; accedit en 19/09/2021: Link].

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. 2014. [Online; accedit en 19/09/2021: Link].

[5] He-Liang Huang, Yuxuan Du, Ming Gong, Youwei Zhao, Yulin Wu, Chaoyue Wang, Shaowei Li, Futian Liang, Jin Lin, Yu Xu, and et al. Experimental quantum generative adversarial networks for image generation. *Physical Review Applied*, 16(2), 2021. [Online; accedit en 19/09/2021: Link].

[6] Gilbert Strang. Linear Algebra MIT OCW, 2011. [Online; accedit en 19/09/2021: Link].

[7] Gilbert Strang. Matrix methods in data analysis, signal processing, and machine learning mit ocw, 2018. [Online; accedit en 19/09/2021: Link].

[8] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambrindge University Press, 10 edition, 2010.

[9] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 2, page 66. Cambrindge University Press, 10 edition, 2010.

[10] Sheldon Axler. *Linear Algebra Done Right*, chapter 3, pages 37–58. Springer, 2 edition, 1997.

[11] Sheldon Axler. *Linear Algebra Done Right*, chapter 3, pages 48–52. Springer, 2 edition, 1997.

[12] Wolfram MathWorld. L2-Norm, 2021. [Online; accedit en 19/09/2021: Link].

[13] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 2, page 74. Cambrindge University Press, 10 edition, 2010.

[14] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 2, page 73. Cambrindge University Press, 10 edition, 2010.

[15] Wikipedia. Tensor product, 2021. [Online; accedit en 19/09/2021: Link].

[16] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 2, pages 80–96. Cambrindge University Press, 10 edition, 2010.

[17] Asher Peres. *Quantum Theory: Concepts and Methods*, chapter 2, pages 24–29. Kluwer Academic Publishers, 2002.

[18] Eleanor Rieffel and Wolfgang Polak. *Quantum Computing: A Gentle Introduction*, chapter 2, pages 12–13. MIT Press, 1 edition, 2011.

[19] Wikipedia. Inverter (logic gate), 2021. [Online; accedit en 04/10/2021: Link].

[20] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 2, page 99. Cambrindge University Press, 10 edition, 2010.

[21] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 2, page 105. Cambrindge University Press, 10 edition, 2010.

[22] Eleanor Rieffel and Wolfgang Polak. *Quantum Computing: A Gentle Introduction*. MIT Press, 1 edition, 2011.

[23] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 2, pages 105–106. Cambrindge University Press, 10 edition, 2010.

[24] Eleanor Rieffel and Wolfgang Polak. *Quantum Computing: A Gentle Introduction*, chapter 10, pages 212–213. MIT Press, 1 edition, 2011.

[25] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software disponible a tensorflow.org.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[27] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. 2020. [Online; accedit en 26/10/2021: Link].

[28] lucidrains. This person does not exist, 2021. [Online; accedit en 26/10/2021: Link].

[29] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019.

[30] Maria Schuld and Nathan Killoran. Quantum machine learning in feature hilbert spaces. *Physical Review Letters*, 122(4), 2019.

[31] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Prediction by linear regression on a quantum computer. *Physical Review A*, 94(2), 2016.

[32] Aram W. Harrow and John C. Napp. Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms. *Physical Review Letters*, 126(14), Apr 2021.

[33] Yudong Cao, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. Quantum neuron: an elementary building block for machine learning on quantum computers. 2017. [Online; accedit en 6/11/2021: Link].

[34] Jonathan Romero and Alan Aspuru-Guzik. Variational quantum generators: Generative adversarial quantum machine learning for continuous distributions. 2019. [Online; accedit en 6/11/2021: Link].

[35] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, Aug 2019.

[36] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[37] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Quantum generative adversarial networks for learning and loading random distributions. *npj Quantum Information*, 5(103), 2019. [Online; accedit en 01/11/2021: Link].

[38] Cirq Developers. Cirq, August 2021. See full list of authors on Github: https://github .com/quantumlib/Cirq/graphs/contributors.

[39] MD SAJID ANIS, Héctor Abraham, AduOffei, Rochisha Agarwal, Gabriele Agliardi, Merav Aharoni, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Matthew Amy, Sashwat Anagolum, Eli Arbel, Abraham Asfaw, Anish Athalye, Artur Avkhadiev, Carlos Azaustre, PRATHAMESH BHOLE, Abhik Banerjee, Santanu Banerjee, Will Bang, Aman Bansal, Panagiotis Barkoutsos, Ashish Barnawal, George Barron, George S. Barron, Luciano Bello, Yael Ben-Haim, M. Chandler Bennett, Daniel Bevenius, Dhruv Bhatnagar, Arjun Bhobe, Paolo Bianchini, Lev S. Bishop, Carsten Blank, Sorin Bolos, Soham Bopardikar, Samuel Bosch, Sebastian Brandhofer, Brandon, Sergey Bravyi, Nick Bronn, Bryce-Fuller, David Bucher, Artemiy Burov,

Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adam Carriker, Ivan Carvalho, Adrian Chen, Chun-Fu Chen, Edward Chen, Jielun (Chris) Chen, Richard Chen, Franck Chevallier, Kartik Chinda, Rathish Cholarajan, Jerry M. Chow, Spencer Churchill, CisterMoke, Chri. Qiskit: An open-source framework for quantum computing, 2021.

[40] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*, chapter 2, page 62. Cambrindge University Press, 10 edition, 2010.

[41] David J. Griffiths. *Introduction to Quantum Mechanics*, chapter 1, pages 11–12. Prentice Hall, 1995.

[42] Lov K Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996. [Online; accedit en 05/10/2021: Link].