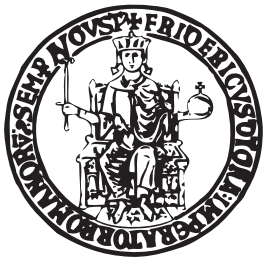
UNIVERSITÀ DEGLI STUDI DI NAPOLI
"FEDERICO II"



Scuola Politecnica e delle Scienze di Base

Area Didattica di Scienze Matematiche Fisiche e Naturali

Dipartimento di Fisica "Ettore Pancini"

Laurea Triennale in Fisica

Generalizzazione di un neurone quantistico al caso multi-classe implementato sull'IBM-Q Tenerife

Relatori:
Prof. Giovanni Acampora

Candidato:
Ferdinando D'Apice
Matr. $N85000921$

Anno Accademico 2018/2019

Indice

Capitolo 1

Machine learning

La branca del Machine Learning indaga i metodi statistici atti a migliorare progressivamente le performances di un algoritmo che ha come obiettivo finale l'identificazione di pattern nei dati. Attraverso questo approccio è possibile modellizzare alcuni fenomeni partendo da set di dati completi e quindi effettuare previsioni sulla base dei dataset iniziali, questo rende il machine learining incredibilmente versatile a seconda del caso in studio. Esempi di particolare interesse, noti nell'utilizzo delle teorie dell'apprendimento, sono: il filtraggio delle mail di spam e delle notizie all'interno dei social network e le intelligenze artificiali applicate alla guida di automobili. Inoltre è possibile notare che lo spettro di utilizzo di queste tecniche è incredibilmente vario e va oltre i casi citati.

Esistono numerose tipologie di Machine Learning ma quella utilizzata in questa trattazione sarà quella riferita al "Supervised Learning" dove alla macchina saranno forniti un dataset di training cioè dei campioni di dati a cui sono gia state assegnate correttamente le etichette da riconoscere; in questo modo l'algoritmo apprende gli schemi da riconoscere mediante il dataset noto. I classificatori binari rientrano nella categoria del Supervised Learning, in questo caso l'algoritmo riconosce, dopo una prima fase di learining, un determinato pattern. Un famoso esempio è quello del filtraggio di mail, dove dato un set di mail note, l'intelligenza acquisirà la capacità di distingurle.

Prima di discutere delle implementazioni di un neurone quantistico vediamo più nel dettaglio il suo funzionamento. Provando a capire come funzionasse il cervello umano, al fine di creare un AI, McCullock e Pitts publicarono nel 1943 la prima idea di come un neurone biologico lavori arrivando a capire che i vari neuroni sono interconessi tra loro al fine di processare e trasmettere informazioni attraverso segnali elettrici.

1.1 Neurone di Rosenblatt

McCullock e Pitts hanno modellizzato un neurone come una semplice porta logica con un output binario. Qualche anno dopo Frank Rosenblatt pubblicò il primo concetto di apprendimento da parte di neurone, con le

regole di apprendimento venne proposto un algoritmo che in autonomia fosse in grado di imparare i pesi ottimali moltiplicati ai vari input così da permettere alla cellula di attivarsi o meno [1].

In maniera più formale, è possibile schematizzare l'idea dietro un neurone come un classificatore binario, si può definire una funzione di attivazione $\phi(z)$ come combinazione lineare tra un vettore in input $\mathbf{i}$ e un vettore che rappresenti i pesi $\mathbf{w}$, dove z è

$$z = \sum w \cdot i \tag{1.1}$$

dove

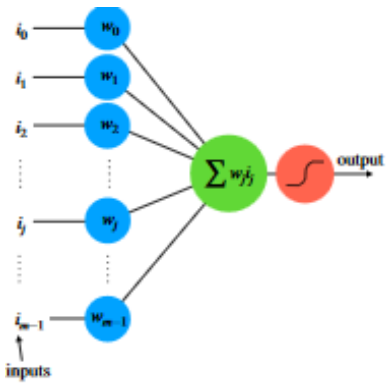
$$\mathbf{w} = \begin{bmatrix} w_1 \\ . \\ . \\ w_m \end{bmatrix} \mathbf{i} = \begin{bmatrix} i_1 \\ . \\ . \\ i_m \end{bmatrix} \tag{1.2}$$

Nel caso più semplice è possibile definire una soglia θ, in modo tale da poter definire l'output positivo o negativo rispetto a z dipendente da questo nuovo parametro; quindi la funzione $\phi(z)$ potrà essere definita come segue

$$\phi(z) = 1 \quad if \quad z \geq \theta \tag{1.3}$$

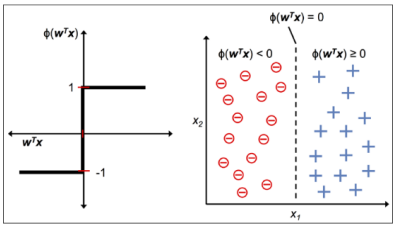
Per semplicità, so può considerare la soglia θ come limite che divide l'attivazione o no del percettrone, e potremo traslare a 0 la soglia in modo da riscrivere z in forma più compatta:

$$\mathbf{z} = w^T x \tag{1.4}$$



Nella notazione di machine learning, la soglia negativa $--\theta$ è di norma chiamata bias unit. La seguente figura mostra come nel caso di output binario (1 o -1), di una rete che rispetta la seguente legge, e come può essere utilizzato per descriminare due tipi di pattern.

$$z = w^T x \tag{1.5}$$



1.2 The perceptron learning rule

L'idea del neurone di Rosenblatt è quella di un modello di percettrone dotato di una certa soglia e che mimi l'approccio come un singolo neurone di un cervello lavori. La proposta iniziale di Rosenblatt per le regole di apprendimento so semplice e possono essere sintetizzate in pochi passi:

- Inizializzare il vettore relativo ai pesi ad un valore casuale
- Per un dato esempio di training $x^{(i)}$ bisogna computare il valore di output y^i e aggiornare il vettore relativo ai pesi.

Nel caso di un neurone classico, l'aggiornamento dei pesi **w** può essere formalmente schematizzato come :

$$w_j = w_j + \Delta w_j \tag{1.6}$$

Il valore di Δw_j usato per aggiornare i pesi e calcolato come segue :

$$\Delta w_j = \eta(y^i - y^{'i})x_j^i \tag{1.7}$$

Dove η è il learining rate, y^i è il valore vero di aspettazione dell i-esimo esempio del training, mentre $y^{'i}$ è il valore di aspettazione predetto. Quindi il percettrone riceve un x come campione e w come pesi e una combinati vengonono compilati come input. Se la combinazione supera una certa soglia, l'output sarà positivo, in caso contrario è negativo. Durante la fase di learning, l'output è usato per calcolare l'errore della predizione e aggiornare i pesi.

1.3 Approccio All vs One

Nel caso di analisi di dataset con molteplici output, come nel caso di riconoscimento di colori o determinati tipi di fiori, si può adattare il neurone in esame con un approccio **One versus All**. Questa è una tecnica che permette di estendere un classificatore binario al caso multiclasse, dove una particolare classe è trattata come esito positivo mentre tutte le restanti come esito negativo. Al fine di classificare un particolare dataset con n classi da riconoscere è possibile creare una rete tale che ogni neurone riconosca un solo pattern come positivo.

Capitolo 2

Quantum machine learning

2.1 Teoria dei qubits

Un qubit è l'equivalente quantistico di un bit classico [2]. Un bit classico è la base dell'informazione nel caso di un calcolatore classico, esso potrà assumere come valori solo 1 o 0. Un qubit analogamente è la base per un calcolatore quantistico che rappresenta uno delle possibili combinazioni di due valori. La potenza di computer quantistico è tutta nella possibilità di manipolare i qubit. Mentre un bit può rappresentare solo il bianco o il nero, un qubit può rappresentare le varie sfumature di grigio che intercorrono tra i due estremi. In questa trattazione i qubits verranno descritti come oggetti matematici, quindi ci disinteresseremo completamente di come questo possa essere prodotto materialmente. Un qubit potrà trovarsi contemporaneamente in due stati con una certa probabilità, quindi definire un qubit come una funzione d'onda:

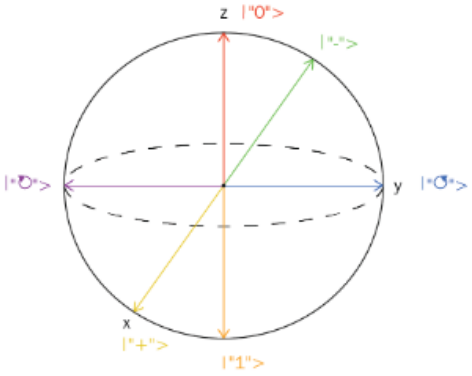
$$|\psi >= \alpha|0 > +\beta|1 > \tag{2.1}$$

Dove α e β appartengono al campo complesso. I $|\alpha|^2$ e $|\beta|^2$ rappresentano le probabilità di trovarsi in una o l'altra base. Gli stati sono normalizzati ad 1. A causa di una mancata corrispondenza biunivoca con elementi reali e qubit è di difficile intuizione il comportamento di sistemi quantistici. In questa trattazione la scelta di utilizzare $|0 >$ e $|1 >$ come basi è completamente arbitraria, si potrebbe in questo modo specificare la percentuale di sovraposizione di stati su una e sull'altra base. Ulteriori basi ,oltre quella utlizzata chiamata standard basis, possono essere ottenute mediante combinazioni lineari della base standard. Lo stato di un qubit può essere pensato come un vettore bidimensionale appertente a uno spazio vettoriale complesso. Nel caso classico è possibile eseminarne il valore senza perturbare lo stato in cui si trova il bit, mentre nel caso quantistico esistono numerose complicazioni dovute in primo luogo nell'effettuare misure sul circuito in studio dato cha la nostra funzione d'onda collasserà ad un singolo valore classico così da non permettere ulteriori operazioni postume alla misura. Un'ulteriore problematica riguarda la decoerenza quantistica, ovvero la capacità che due o più qubit possano lavorare in

maniera coerente necessitando di mantenere la stessa frequenza di elaborazione e la stessa fase; purtroppo il computer quantistico utilizzato subisce la decoerenza che non permette delle performace ideali, queste ultime sono dovute a quanto velocemente i qubits perdono energia a causa dell'interazione con l'ambiente e a causa dello sfasamento che cresce con il crescere del tempo di esecuzione del codice. Nella trattazione che segue cercheremo di ignorare questa problematica mediante simulazioni ,dove la decoerenza non è presente, oppure mediante misure ripetute così da cercare di arginare il fenomeno. Se immaginiamo un bit classico come una moneta, dove una faccia vale 0 e l'altra vale 1, avremo un 50% di probabilità di osservare uno dei due valori; l'equivalente quantistico di questo esempio è il seguente stato:

$$|\phi >= 1/\sqrt{2}|0 > +1/\sqrt{2}|1 > \tag{2.2}$$

Si può notare che qualsiasi qubit può essere ottenuto mediante combinazione lineare di due o più qubit, questo è il principio della sovrapposizione dei qubit. Al fine di visualizzare lo stato di un singolo qubit è possibile avvalersi di una rappresentazione geometrica, ovvero la sfera di Bloch. Mediante questo tecnica è possibile osservare lo stato di un qubit, infatti manipolando i coefficienti è possibile osservare come varia con continuità lo stato di un qubit. La sfera di Bloch può essere pensata come una sfera tridimensionale con raggio unitario, un qualsiasi qubit può essere visto come una linea dal centro fino ad un punto sulla superfice della stessa. La visualizzazione avviene sulle sei basi definite in precedenza, infatti l'asse z sarà rappresetato da $|0 >$ per il semiasse positivo mentre $|1 >$ per quello negativo, l'asse x da $|+ >$ per il semiasse positivo mentre $|- >$ per quello negativo, analogamente per l'asse y avremo $|Cs >$ e $|CCs >$.Il limite di questa rappresentazione è il fatto che non vi è permessa una generalizzazione a più qubit.



Il fatto che la sfera di Bloch possiede infiniti punti, cioè i coeffiecienti del qubit α e β possono assumere qualsiasi valore, indica che il sistema non è interrogato dall'esterno; in realtà nell'effettuare la misura si passerà da un'iniziale sovrapposizione di stati al collasso in un stato determinato, ovvero quello associato all'autovalore ottenuto.

2.2 Quantum states e register

Un Quantum state è un gruppo di uno o più qubit i quali sono fisicamente collegati tra loro, in ogni stato composto da almeno due qubit è possibile che essi sperimentino il fenomeno dell'entanglment. Un registro classico potrà conservare n bit mentre un quantum register potrà mantere in memoria la sovrapposizione di n-qubit; per meglio dire nel caso classico si può conservare con n bit uno dei possibili 2^n numeri, in quello quantistico con n qubit è possibile conservare in memoria ogni combinazione di 2^n numeri. In generale, un sistema di n qubits può essere sovrapposizione di 2^n stati e pertanto possiede una potenza computazionale che cresce come un esponenziale in base 2. Per esempio, volendo rapresentare il numero 19 in base binaria, ovvero 10011 su un registro classico, occorerrebbero 5 bit. Nel caso di n qubit per ricreare l'analogo stato, si andrà a preparare lo stato $|10011>= |1> \otimes |0> \otimes |0> \otimes |1> \otimes |1>$. Dove $\otimes$ rappresenta il prodotto di kronecker. Dato uno stato, è possibile provare a ridurlo nei singoli qubit che lo costituiscono; nel caso in cui sia effettivamente possibile ridurlo lo stato in questione verrà chiamato "Separable state", nel caso invece di uno stato formato da due qubit , ad esempio uno stato con a e b, non è possibile risalire ai singoli qubit e per questo viene detto "Nonseparable state". Uno stato non separabile è detto anche "Entagled state".
$|entangled_state>= 1/\sqrt{2}(|00> +|11>)$
L'entanglement verrà successivamente trattato come un processo atto alla creazione di un nonseparable state, una volta effettuato non potremo più considerare ogni qubit costituente del registro in maniera separata l'una dall'altra. Al fine di rendere più chiara la nostra trattazione verrà utilizzata come base quella costituita dai qubit $|1>$ e $|0>$.

2.3 Quantum Gates

Analogamente ad un calcolatore classico, anche quelli quantistici presentano gates che permettono di manipolare l'informazione effettuando operazioni quantistiche elementari. In un computer quantistico, un gate quantistico opera su un quantum register al fine di evolvere i suoi stati. Dato che un quantum state è rappresentato da una matrice, l'evoluzione di uno stato può essere visto come un operatore che agisce sullo stato iniziale; in particolare l'operatore deve produrre uno stato che deve conservare la norma e non può produrre una sovrapposizione di stati con una probabilità maggiore di uno. Al fine di rispettare le condizioni precedenti, gli operatori dovranno essere unitari, cioè tale che $U^TU = UU^T = I$, dove I è la mtrice identità. Nella computazione quantistica, un quantum gate corrisponde una matrice unitaria, ogni matrice unitaria è invertibile, quindi per ogni gate ne esiste uno ulteriore che permette di invertire l'azione su uno stato. Alcuni quantum gate operano solo su qubit singoli mentre altri, invece, operano su due o più qubit; è possibile una genera-

lizzazione dei gate che agiscono su due qubit alla volta dove si passa a trattare casi a n qubit mediante un analisi degli operatori che costituiscono i gate. Mediante la sfera di Bloch è possibile vedere come operano i single-gate su i singoli qubit. Nel caso quantistico bisogna pensare alla sovrpposizione di stati e non al singolo valore ottenuto dalla misura, inoltre i vari operatori devono agire in modo lineare sul sistema in studio.
Il gate H è noto come **Hadamard gate** è definito come segue :

$$H = 1/\sqrt{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.3}$$

Ovvero , ad esempio, dato uno stato iniziale $|0>$ ci restiuisce lo stato $|+>$, in altre parole :

$$|+>= H|0>= 1/\sqrt{2}(|0>+|1>) \tag{2.4}$$

Per quanto riguarda la porta logica NOT, nella computazione quantistica ha l'analogo nelle matrici di Pauli, cioè, considerando ad esempio nel caso di $|0>$ e $|1>$, è possibile passare da base all'altra, ovvero scambiare i coefficienti utilizzando il gate X; quindi ruotando di 180 gradi rispetto ad un asse lo stato rappresentato mediante la sfera di Bloch. I cosidetti gate di Pauli sono:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.5}$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{2.6}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.7}$$

Il gate I è conosciuto come gate identità e restuisce in output la stessa funzione in ingresso, è definito come segue:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.8}$$

Il gate S conosciuto come gate di phase, e il corrispettivo aggiunto $s^\dagger$, ruotano in senso antiorario (o orario a seconda del gate) lo stato sul piano xy della sfera di Bloch; sono definiti in questo modo :

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \tag{2.9}$$

$$S^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix} \tag{2.10}$$

Il gate T e il corrispettivo aggiunto, chiamato $\pi/8$, ruota lo stato di $\pi/4$ sul piano xy della sfera di Bloch, sono definiti come segue:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \tag{2.11}$$

$$T^{\dagger} = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix} \tag{2.12}$$

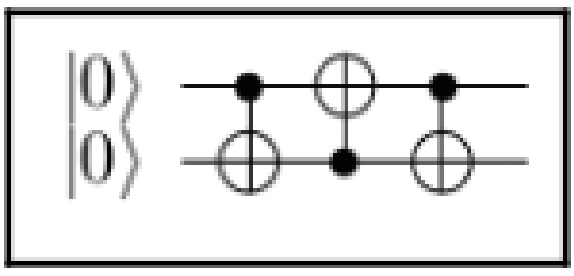
Il gate CNOT è conosciuto come **quantum NOT** ed è un gate di partcolare interesse poichè accetta due qubit in input, questo gate è chiamato anche $2 - qubitcontrollednotgate$. Il primo input è visto come qubit di controllo mentre il secondo è considerato come target, lo stato del primo qubit rimane invariato invece il target qubit varierà il suo valore come se su di esso venisse appliccato un gate X, questo se e solo se il qubit di controllo è $|1>$. L'applicazione del gate CNOT genera un entanglement tra i due qubit, è possibile notare che la combinazione tra un CNOT e un ulteriore gate restituisce ancora un gate, che essendo ancora rappresentato da una matrice unitaria e invertibile, risulta essere valido. Il gate CNOT è definito come segue:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.13}$$

Può essere rilevante notare che non tutte le porte classiche hanno un corrispettivo quantistico, infatti, ad esempio la porta NAND non essendo invertibile non è possibile rappresentarla mediante una matrice unitaria e invertibile.

2.4 Quantum circuit

Un quantum circuit è semplicemente una sequenza di quantum gate applicati ad un quantum register. Lo stato $|+>= H|0>$, ottenuto mediante un Hadamard gate è un primordiale circuito atto alla creazione di un nuovo stato. Un quantum circuit diagram è un metodo per visuallizzare un quantum circuit, tipicamente viene letto da sinistra verso destra e lo stato iniziale del registro sono n qubit nello stato $|0>$. Ogni linea è uscente da un qubit ma non necessariamente il filo è corrispondente ad una connessione elettrica. I circuiti quantistici possono essere trattati come moduli in modo da comporre circuiti via via sempre più complessi. Vediamo l'esempio di un semplice circuito che scambia lo stato di due qubit:

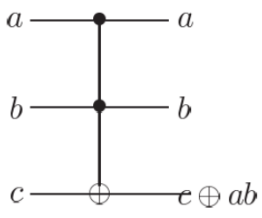


Tale circuito, costruito mediante tre porte CNOT dove si alternano i compiti di controller e target volta per volta, ci permette infatti di scambiare lo stato di due qubit.

$$|b, a >= SWAP|a, b > \tag{2.14}$$

Un ulteriore circuito di particolare interesse è la porta Toffoli, che è una versione quantistica della porta AND tuttavia è basata su sole porte che operano in maniera reversibile, essa possiede tre input, dove due qubit fungono da controller mentre il terzo da target. Se i qubit di controllo non sono settati nello stato $|1 >$ segue che il qubit target rimane invariato, in caso contrario il terzo qubit invertirà il suo valore.

Inputs			Outputs		
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0



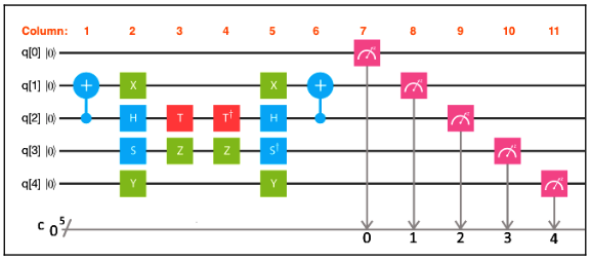
Attraverso questa porta è possibile realizzare anche il corrispettivo quantistico delle porte OR e NAND, replicando queste componenti è possibile simulare qualunque altro elemento di un circuito classico.

Ci sono alcune caratteristiche dei circuiti classici che non posso essere riprodotte nel caso quantistico, ad esempio i loop non sono realizzabili. Nei circuiti quantistici inoltre non è possibile effettuare copie di stati: questo principio può essere dimostrato mediante il principio di indeterminazione di Heisenberg. A causa di questo limite, anche riproducendo il circuito innumerevoli volte si possono ottenere risultati diversi a causa di un rumore che non può essere rimosso, dovuto a molteplici fattori.

2.5 QASM

Nonostante non sia possibile replicare direttamente tutti i circuiti logici classici mediante gli strumenti della computazione quantistica, a causa dei limiti visti precendentemente, è comunque possibile ricrearli medianti circuiti costruiti ad hoc. A causa della principale differenza tra computazione quantistica e computazione classica, tale discrepanza risiede nel fatto che la prima sia non deterministica a differenza della seconda, è permesso in quantistica lavorare con valori casuali e non mediante una funzione che restituisce un numero pseudocasuale.
Al fine di lavorare con un computer quantistico ci avvaliamo, in questa trattazione, del quantum composer mediante un SDK chiamato OpenQUASM che ci permette di visualizzare e modificare in maniera chiara il circuito in costruzione. La sintassi di OpenQUASM è molto similare

a quella della sintassi in C, infatti non conta l'identazione e ogni riga di comando dovra terminare con ";". Ecco di seguito un circuito specchio che mostra la corrispondenza tra codice e rappresentazione.



2.6 Quantum Error

Con un computer quantistico, a causa dell'interazione con l'ambiente, si sperimenta il fenomeno della decoerenza durante l'esecuzione del codice, questo causa una degenerazione dell'informazione da elaborare. L'hardware IBM-QX possiede dei tempi di decoerenza T1 e T2 ancora abbastanza bassi tali che buona parte dei circuiti implementati contiene degli errori che si manifestano come rumore durante il processo di misura, questi possono presentarsi come differenza di fase rispetto al valore di aspettazione del caso teorico. Segue quindi che un semplice algoritmo non può essere esente da errori e al fine di ridurli al minimo è possibile implementare particolari operatori, quindi costruire circuiti inusuali. Questo significa che il numero di gate e/o qubit necessari, a seconda dei casi, sarà maggiore rispetto a quelli previsti nel caso teorico. Poichè la correzione degli errori richiede dei qubit supplementari al fine di correggere il codice dell'algoritmo originale. Ciò significa che un sistema di calcolo quantistico necessità di un architettura molto più articolata di quanto dovrebbe se la correzione degli errori non fosse necessaria [5].

Capitolo 3

Quantum circuit modeling of a classical perceptron

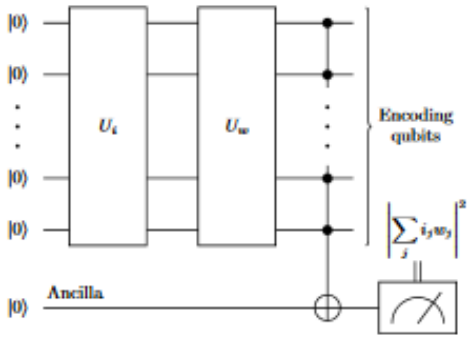
Al fine di introdurre un modello di percettrone di Rosenblatt su un computer quantistico è necessario creare un modello per i vettori in input. Per fare ciò possiamo codificare un vettore m-dimensionale, nel caso di hardware quantistico, attraverso l'uso di N qubit dove $m = 2^N$. Sarà mostrato in seguito il processo per codificare i vettore nel caso con quattro qubit grazie al processore messo a disposizione dall'IBM tuttavia a causa di determinate limitazioni è necessario ricorrere ad alcune modifiche costruite appositamente per arginare il problema. Sorprendentemente sarà mostrato come il modello di percettrone quantistico può essere usato per risolvere determinati pattern semplici.

3.1 Quantum algorithm proposed

I vettori di input e dei pesi sono limitati a valori binari $i_j, w_j \in -1, 1$, come nel modello di neurone di McCulloch-Pitts [4]. Un vettore m-dimensionale è codificato mediante m coefficienti necessari per definire la funzione d'onda $|\phi>$ formata da N qubit. In pratica non sarà fatto altro che creare i vettori $i = (i_0, i_1, ..., 1_{m-1})$ e $w = (w_0, w_1, ..., w_{m-1})$ con $i_j, w_j \in -1, 1$, per farlo è necessario definire due stati

$$|\psi_i >= 1/\sqrt{m} \sum_{j=0}^{m-1} i_j |j >; |\psi_w >= 1/\sqrt{m} \sum_{j=0}^{m-1} w_j |j > \tag{3.1}$$

Gli stati $|j>$ non sono altro che i vettori che compongono la base associata al processore quantistico, cioè la base nello spazio di Hilbert formata da N qubit; questi qubit corrispondo a tutti gli stati ottenuti mediante combinazione degli stati che formano i singoli qubit. Di norma questi stati potranno essere indicati medianti un "etichetta", ovvero un enumerazione, $j \in 0, .., m-1$. Possiamo usare i fattori $+1$ e -1 per codificare i vettori classici m-dimensionali in stati rappresentabili attraverso la base $|j>$.



Il primo passo è preparare lo stato $|\psi_i>$ attraverso la codifica degli input i. Assumendo che il registro sia inizializzato a $|00..0>$, è possibile applicare una trasformazione unitaria U_i tale che

$$U_i|0>^{\otimes N} = |\psi_i> . \tag{3.2}$$

In linea di principio, qualsiasi matrice quadrata unitaria di ordine m avente elementi solo nella prima colonna può essere utilizzata per questo obiettivo. Il secondo passo da operare è quello di calcolare il prodotto scalare tra w e i utilizzando il registro quantistico. Per questo compito è necessario avvalersi di una trasformazione unitaria U_w, che ruoterà il vettore relativo ai pesi come segue

$$U_w|\psi_w> = |1>^{\otimes N} = |m-1> . \tag{3.3}$$

Qualsiasi matrice unitaria quadrata di ordine m avente elementi solo nell'ultima riga soddisfa le condizioni per questo caso. Applicando U_w dopo U_i, lo stato che si ottenuto è

$$U_w|\psi_i> = \sum_{j=0}^{m-1} c_j|j> . \tag{3.4}$$

Il prodotto scalare tra i due stati è

$$<\psi_w|\psi_i> = <\psi_w|U_w^\dagger U_i|\psi_i> = c_{m-1} \tag{3.5}$$

Da questo risultato è facile notare che il prodotto scalare tra i e w restituisce $m < \psi_w|\psi_i >$. Il risultato desiderato, a meno di una costante di normalizzazione, è il coefficiente c_{m-1} dello stato finale. Al termine del prodotto scalare, è proposto un metodo per estrarre informazioni dal circuito in maniera chiara e tale da minimizzare il rumore, ci si propone di utilizzare un qubit ausiliare, inizializzato nello stato $|0>$,costruendo un multi-controlled NOT gate tra i vari qubit utilizzati per la codifica dei vettori.
La misura effettuata sul qubit ausiliario nella base scelta produrrà come output uno stato $|1>$ con una probabilità di $|c_{m-1}|^2$, tale probabilità è indice dell'attivazione del percettrone. Il risultato del prodotto è memorizzato in un terzo qubit al fine di ridurre gli errori. É rilevante notare

che nel caso in cui i vettori i e w siano paralleli o antiparalleli segue l'attivazione del percettrone, nel caso di vettori ortogonali il risultato della misura effettuata sul qubit ausiliario restituisce $|0>_a$. Utilizzando il percettrone come classificatore di pattern, se ne ottiene l'attivazione, oltre che con il pattern corretto, anche con il corrispettivo negativo, questo avviene a causa dell'invarianza della codifica di $|\psi_i>$ e $|\psi_w>$ di un fattore -1.

3.2 Implementation of encoding by Unitary Trasformation

Nelle applicazioni riguardanti il machine learning, l'ottimizzazione dell'algoritmo può mostrare il divario che si intrepone tra computazione quantistica e classica. L'implememntazione delle matrici unitarie, atto a creare lo stato di input $|\psi_i>$ e la trasformazione U_w, risulta un processo basato sulla generazione di particolari stati. L'algortimo più semplice che potremmo pensare di costruire utilizzerà l'applicazione dei sign-flip. Il primo passo sarà definire i sign-flip $SF_{N,j}$ come una trasformazione unitaria che agisce sulla base di computazione. Per o Per ogni N, $m = 2^N$, opererà un Controlled-Z tra N qubit ($C^N - Z$) tale che sia $SF_{N,m-1}$, nel caso di un signolo qubit lo Z gate agirà come $SF_{1,1}$. Possiamo quindi implementare la struttura per costruire blocchi sign-flip per N qubits usando $C^N - Z$ gate in combinazione con gate NOT :

$$SF_{N,j} = O_j(C^N Z)O_j; \tag{3.6}$$

dove O_j :

$$O_j = \bigotimes (NOT_l)^{1-j_l} \tag{3.7}$$

Nella precedente espressione, il NOT_l significa che il bit glip è applicato al l-esimo qubit e $j_l = 0$ se l-essimo qubit è nello stato $|0>$ nella base definita da $|j>$. Alternativamente questo identico risultato può essere ottenuto mediante un multi-CNOT gate cosi che i bit flip siano i controllori dei CNOT. Ricordando che una condizione necessaria di una trasfromazione unitaria è che sia reversibile, infatti $SF_{N,j}$ è l'inverso di se stesso. La sequenza per implementare U_i può essere riepilogata come segue: partendo da un registro inizializzato a $|0>^N$ e mediante $H^{\otimes N}$ possiamo creare una sovrapposizione di N stati di tutti gli elementi della base. Quindi i blocchi $SF_{N,j}$ sono applicati uno per uno dove c'è un fattore -1 prima di $|j>$, nella rappresentazione dell'obbiettivo $|\Psi_i>$. Notiamo come ogni $SF_{N,j}$ agisce solo su un singolo elemento della base mentre lascia tutti gli altri invariati. Tutti i blocchi $SF_{N,j}$ commutano tra di loro, quindi posso essere applicati in qualsiasi ordine. Ci potranno essere al massimo $\frac{m}{2} = 2^{N-1}$ fattori -1 indipedenti e al massimo $2?N - 1$ sign flip. Una strategia simile può essere utilizzata per creare l'operatore U_w nella seconda parte dell'algoritmo del percettrone quantistico. Infatti

applicando prima i blocchi $SF_{N,j}$ che sarebbero necessari per "capovolgere" di segn0 -1 alla base associata a $|\psi_w>$, porta ad una sovrapposizione di stati bilanciata $|\psi_w> \longrightarrow |\psi_0>$. Questo stato può essere portato in quello desiderato mediante l'applicazione infine di porte hadamard e CNOT.

$$|\psi_0\rangle \xrightarrow{H^{\otimes N}} |0\rangle^{\otimes N} \xrightarrow{NOT^{\otimes N}} |1\rangle^{\otimes N}$$

Evidentemente questa strategia è esponenzialmente dispendiosa in termini di profondità di circuito a causa della moltitudine di qubit necessari a creare il modello proposto. Notiamo che il ruolo di U_w in questo algoritmo, possiede essenzialmente il ruolo di cancellare la prima trasformazione unitaria dovuta a U_i per preparare lo stato $|\psi_i>$. Possiamo notae che per via della codifica, la natura dei vettori i e w rimane sconosciuta durante il tempo di esecuzione.

3.3 Implementation of quantum code

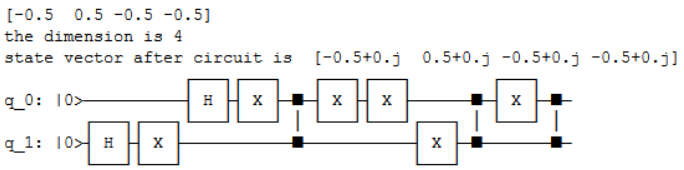
L'algoritmo indicato in precedenza sarà applicato al caso di un percettrone implementato mediante 4 qubit. Potremo lavorare in python mediante un backend in modo da rendere il codice meno ostico. Il primo passo è definire i 16 stati rappresentabili mediante i nostri qubit [3]. Ricordiamo che il guadagno è esponziale quindi con 4 qubit avremo a disposizione 2^N stati.

```
k= 0   [1. 1. 1. 1.]
k= 1   [ 1.  1.  1. -1.]
k= 2   [ 1.  1. -1.  1.]
k= 3   [ 1.  1. -1. -1.]
k= 4   [ 1. -1.  1.  1.]
k= 5   [ 1. -1.  1. -1.]
k= 6   [ 1. -1. -1.  1.]
k= 7   [ 1. -1. -1. -1.]
k= 8   [-1.  1.  1.  1.]
k= 9   [-1.  1.  1. -1.]
k= 10   [-1.  1. -1.  1.]
k= 11   [-1.  1. -1. -1.]
k= 12   [-1. -1.  1.  1.]
k= 13   [-1. -1.  1. -1.]
k= 14   [-1. -1. -1.  1.]
k= 15   [-1. -1. -1. -1.]
```

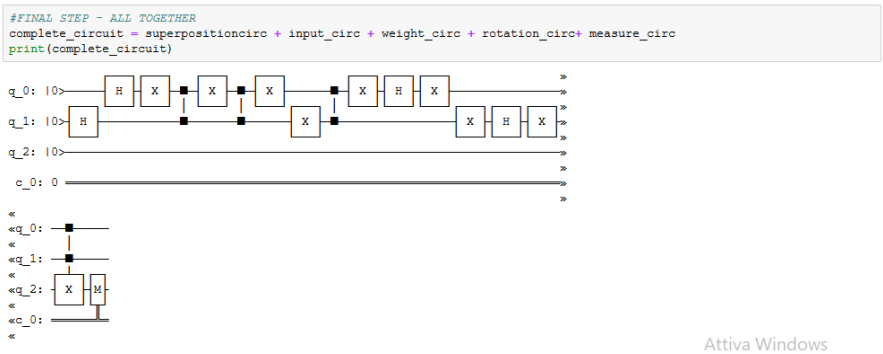
Gli stati rappresentati sono esenti da normalizzazione. Essendo che il quantum code , mediante Qiskit, è possibile maneggiarlo in moduli e solo nella parte finale comporlo il primo passo è quello di creare il primo modulo formato da $H^{\otimes N}$ e $CNOT$ applicati in maniera tale da creare lo stato desiderato $|\psi_i>$.

```
def sign_flip_encoder(i):
"""
Parameters:
i a classical binary valued input vector
Returns:
circ a quantum circuit that properly encodes the input vector
Descripton:
- convert vector to binary string
- Apply Sign-flip algorithm whenever there is a negative -1 in i
- Use -1 position to identify corresponding computatuion basis.
"""
i = np.array(i)
d = len(i) #dimension of input vector

print("the dimension is", d)
#controll the input
#if the rest isn't identical to zero
#validate input
assert math.log(d,2) %1 == 0, "Invalid vector length. Must be 2^N"
N = int(math.log(d, 2)) # number of qubits
#Generate computational basis (standard basis)
basis = [("{:0%db}"%N).format(k) for k in range(d)]

#Create quantum register and circuit
q = QuantumRegister(N, 'q')
circ = QuantumCircuit(q)
#Start in a superposition
circ.h(q[0])
circ.h(q[1])

# Find all components with a -1 factor in i
indice = np.where(i == -1)[0]
if indice.size > 0:
#index variable
for idx in indice:
#Need to switch qubits in the 0 state so CZ will take effect
for i,b in enumerate(basis[idx]):
if b == '0' :
circ.x(q[(N-1)-i])
#(N-1)-i is to match the qubit ordered inverse
circ.cz(q[0],q[1]) #adapted for two qubits
#Add switch the flipped qubits back
for i,b in enumerate(basis[idx]):
if b == '0':
circ.x(q[(N-1)-i])
return circ
```

La prima sezione è dedicata a creare una base dipendente dal numero di qubit a disposizione, successivamente partendo da un registo composto da tutti stati equiprobabili. Successivamente l'algoritmo in base allo stato da dover creare applicherà nella giusta sequenza i sign flip e i phase flip. Questo è un esempio del circuito atto a creare lo stato relativo al numero 11 :



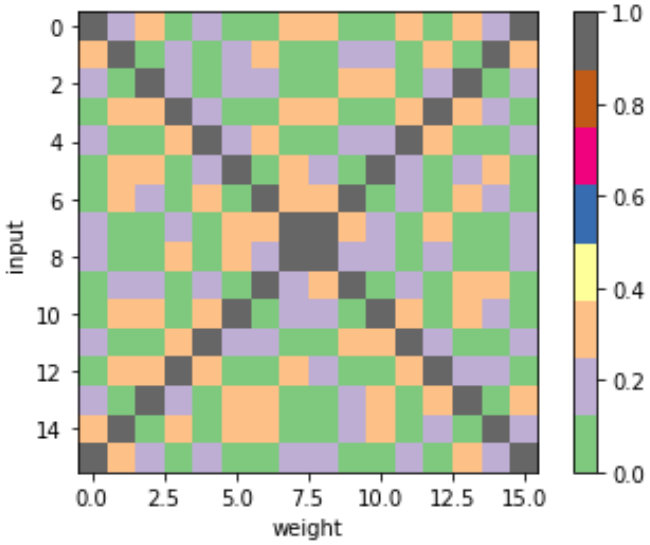
Una volta definiti $|\psi_i>$ e $|\psi_w>$ andremo a definire i moduli relativi alla rotazione dei vettori in modo tale da minimizzare il rumore e avere una lettura relativa alla base $|0>^N$, infine andremo a posizionare una porta Toffoli tra tre qubit utilizzando il terzo qubit come ausiliario. La composizione del codice, in un caso esempio risulterà un circuito del genere :



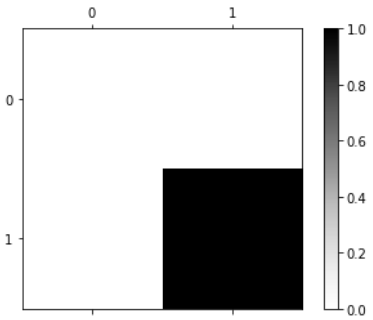
3.4 Numerical simulation

Una volta implementato l'algoritmo per un singolo percettrone quantistico a due qubit, possiamo notare che qusto modello può essere eseguito sia su un simulatore esente da rumore, sia su uno che permette di simulare il rumore, sia sull'IBMQX-4, di particolare interesse poiche saremo limitati da determinati vincoli come la decoerenza e la topologia, in termini di connettività tra i differenti qubits, siamo limitati al caso reale di due qubit. Nonostante un numero cosi piccolo di qubit, possiamo mostrare tutte le caratteristiche riguardo il funzionamento, come la crescita esponenziale di possibilità di conservare dati al fine di risolvere problemi, come il riconiscemento di determinati pattern. In generale, è possibile riconoscere un pattern di 2^N dimensionale avendo a disposizione N qubit. Tutti i gli input binari e i pesi possono essere facilmente convertiti in un pattern bianco-nero, questo provvederà ad un intrepetrazione grafica dell'attività

del percettrone. Nel nostro caso con 2 qubit, quindi un pattern a 4 pixel binari, dove un pixel potrà assumere solo il colore bianco o nero. La conversione tra i ew e una griglia di pixel $2x2$ può essere compiuta come segue. Possiamo eticchettare ogni immagine partendo dal pixel in alto a sinistra fino ad arrivare in basso a destra, assegnando a valore $n_j = 0$ per un pixel bianco (o nero). Il corrispondente vettore input o peso è quindi costruito impostando i_j. Possiamo anche assegnare univocamente un etichetta intera k_i (o k_w) per ogni pattern da convertire. Lo spazio di Hilbert relativo a 2 qubit, è relativamente piccolo, con un totale di 16 possibili valori per i e w. Quindi il modello di percettrone quantistico può essere testato sul computer quantistico dell'IBM per tutte le possibilità di i e w. Questi risultati, sono mostrati nella seguente figura :



La prima cosa che si può notare è che l'attivazione del percettrone accade anche nel caso di "pattern al negativo" a causa dell'invarianza per simmetria. In particolare, per una data combinazione di i e w, il percettrone è in grado di attivarsi (e la sua negativa), con un coefficente $|c_{m-1}|^2$ pari ad 1 , negli altri casi noteremo che gli output sono sempre minori di 0.5. Se gli input sono tradotti come una griglia $2x2$ di pixel bianca e nera, non sarà affatto difficile capire quando ci sarà una corrispondenza tra i due. Ad esempio, qui è mostrato la codifica del vettore relativo a k= 1 :



Attualmente questo esperimento può essere riprodotto direttamente mediante la shell interattiva dell'IBMQ-X Tenerife. I risultati mostrati, riguardano l'approccio ottimizzato mediante i flip sign e flip phase. Nel caso elementare proposto con 2 qubit la precisione del percettrone è ottimale al problema in studio ma al crescere della complessità del circuito, quindi con l'aumentare dei qubit e dei possibili stati rappresentabili accade che la precisione cala portando qualche volta l'attivazione del percettrone in casi errati, non previsti dal caso teorico. L'unico modo di vedere la generalizzazione a più qubit è per via simulativa, infatti non essendo supportata una topologia tale da permettere queste operazioni ci possiamo avvalere di un simulatore che permetterà di analizzare una struttura a 3 +2 qubit al fine di riconoscere un pattern di immagini a $3x3$ pixel bianco-nero. In questo caso noteremo come attraverso la simulazione che utilizza algebra lineare e le funzioni "oracle" ci ritroveremo con dei risultati che imitano in maniera soddisfacente la computazione su hardware quantistico.

3.5 Training

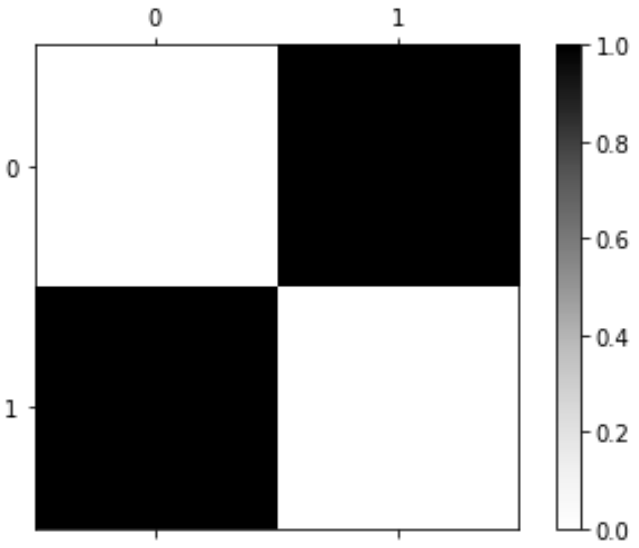
Una volta definito il nostro percettrone, la nostra codifica per i 16 stati rappresentabili mediante griglie di pixel $2x2$. Rappresentiamo le 16 griglie come matrici per comodità, e ad ognuno di queste assegnamo un etichetta. Il primo passo al fine di implementare il training, procedurra che permetterà al nostro percettrone di "imparare" un determinato pattern utilizzandolo poi successivamente per l'attivazione della cellula artificiale, è quello di ricreare la stessa struttura spiegata in precedenza con la sostanziale differenza che il vettore w_o è inizializzato casualmente, questo sarà aggiornato ad ogni ciclo. Quando il pattern da riconoscere è mandato in input e il percettrone si attiva senza problemi lasciando w invariato, ciò significa che w è adatto a riconoscere quel dato set di dati; in caso contrario interviene l'algoritmo di correzione di w. In quest ultimo caso la misura restituirà un coefficiente minore di 0.7 ,soglia di attivazione definita in maniera classica, quindi con una collaborazione tra il computer utilizzato in locale e quello quantistico, il pattern è riconosciuto come diverso, in questo caso il vettore peso e il vettore in input vengono messi a confronto e nel caso di discrepanze il vettore w viene aggiornato in maniera pseudocasuale, infatti il vettore relativo ai pesi è modificato con un fattore -1 in una posizione casuale, inoltre avendo in memoria la precedente versione di se stesso cosi da permettere al nuovo vettore di essere differente dal precedente. Dobbiamo inoltre notare che il nostro percettrone non differenzia tra input positivo e negativo quindi questo processo deve essere effettuato sia nel caso il vettore input viene riconosciuto come positivo sia come negativo, questo sistema è realizzabile attraverso due learning rate, uno positivo e uno negativo, mentre nel caso classico il learining rate è unico. Notiamo che nel caso nettamente quantistico questa necessità di dividere i rating dato che data la simme-

tria il nostro percettrone si attiverà sia in caso di pettern corretti sia con pattern negativi. Notiamo inoltre come il vettore w_t è sconosciuto durante questa fase.

```
def training(label):
threshold = 0.5
iteration = 2
lp=0.5
ln=0.5
#Starting from the initial w0
w0 = np.array(random.choices([-1,1], k=4))
print(w0)
epochs = 1
while True:
N = 0
#whenever the outcome for a given i in the training set and the current w is
#correctly above or below the threshold fixed at the beginning w is left
#unchanged, while it is updated when the input is wrongly classified
for ki in range(16):
#i must be ki but in this case we use the training only for a only label
i = 0
counts = circuit(i,w0) #simulate all cases
if len(counts) == 1:
present_counts = int([*counts][0])
else:
present_counts = float([*counts][1])/1000

#threshold function
if present_counts >= threshold:
output = 1
else:
output = -1
print(output)
#Check the sign, divide the set in a positive and negative
if np.sign(output) != np.sign(labels[i]):
N += 1
#first case :  when a positive output should be negative
if np.sign(output) > 0 :
for p in range (4):
a = conv_km(p,4)
if w0[p] != a[p] :
casual = random.randint(p,3)
w0[casual] *= -1
print(w0)
print(a)
#second case : negative output shoul be positive
if np.sign(output) < 0:
```

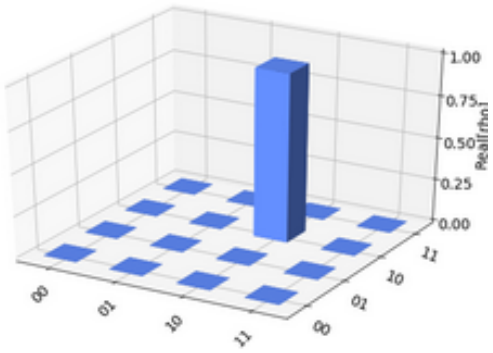
```
for t in range (4):
b = conv_km(t,4)
if w0[t] != b[t] :
dcasual = random.randint(t,3)
w0[dcasual] *= -1
print(w0)
print(b)
if N == 0:
print('converged')
break
elif epochs == iteration:
print("Not converging")
print("w:", w0)
break
else:
epochs += 1

return w0
```

Andiamo a vedere il caso in cui il nostro percettrone dovrà imparare un pattern relativo a questa immagine:



Considerando questo pattern si può vedere come "si allena" il neurone, prima di tutto inizializzerà un vettore peso casuale w_o, il circuito relativo al neurone è eseguito, implementato nel precedente paragrafo; a seconda del risultato ci troveremo in una delle due possibilità precedenti, ovvero all'attivazione o no del percettrone. Nel caso in cui il pattern non verrà riconosciuto rispetto al vettore w_o come corretto alla prima iterazione, il vettore w_0 verrà modificato secondo l'algortimo visto in precedenza, ovvero cosi che dopo un numero finito di cicli il nostro percettrone sarà in grado di rispondere correttamente a i vari input fornitogli. Come

possiamo notare dopo un training il nostro percettrone sarà in grado di attivarsi al momento adatto. La seguente immagine mostra come , nel caso simulato esente da rumore, il nostro percettrone sia in grado di attivarsi all'arrivo di un solo pattern.



3.6 Quantum Simulation

Risulta ovvio che il candidato ideale atto ad compilare un sistema quantistico è per natura un hardware quantistico, infatti un computer classico è in grado di emulare un sistema quantistico ma, computazionalmente, il costo cresce esponenzionalmente al crescere della dimensione del sistema. Al fine di descrivere un sistema quantistico che consta di n differenti componenti è necessaria, su un computer classico, una memoria di c^n bit, invece su un computer quantistico kn, dove k e c sono costanti che dipendono dalle caratteristiche del sistema in considerazione. Ulteriore differenza da evidenziare è il fatto che l'informazione, nel caso quantistico, per essere letta necessità di essere misurata che restituiranno solo kn bit di informazione.

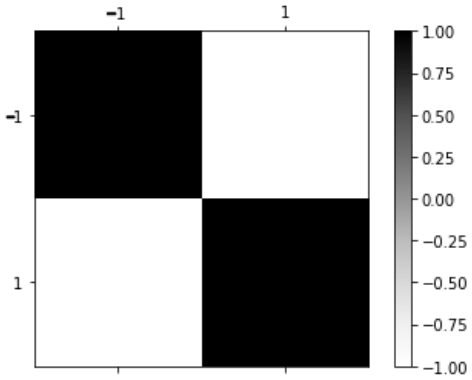
3.7 Iris dataset AllvsOne

Al fine di implementare il nostro neurone con un dataset reale, dobbiamo compiere qualche passaggio preliminare in modo da effettuare una codifica ad hoc per il dataset in studio. In questo caso, il più semplice, è stata effettuata una codifica tale da permettere un corrispondenza biunivoca tra i labels , definiti in precedenza come etichette, cosi da permettere di automatizzare il processo di codifica. In questo modo partendo da un dataset quadridimensionale formato da numeri reali passeremo ad un dataset binario quadridimensionale di facile interpretazione per il nostro neurone quantistico. Per rendere il dataset di facile comprensione per il lettore è stata effettuata un ulteriore codifica legandoli a griglie di pixel $2x2$.
Una volta definita le funzioni relative ai circuiti quantistici, passeremo all'acquisizione del dataset Iris, avente 150 esempi con tre possibili output: Setosa, Virginica, Versicolor. Studiando i dati che compongono i

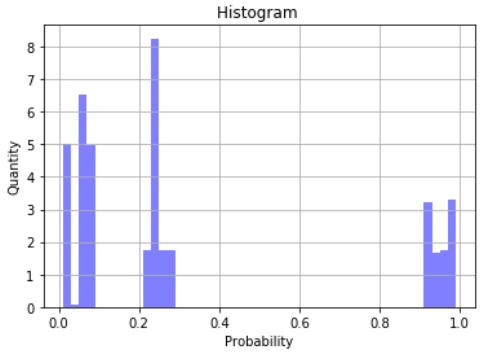
vettori input è possibile rendere i vettori quadridimensionali in forma binaria attraverso una conversione.

$$[0.5, 2.0, 5.4, 4.5] = [1., -1., 1., -1.] \tag{3.8}$$

In questo modo è possibile rappresentare i vettori dei dati in forma di griglie di pixel, come mostrato anche in precedenza.

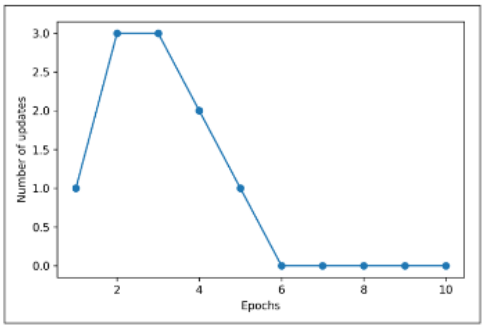


Attraverso questa codifica avremo tre possibili output e quindi tre possibili $|\psi_w$ che permetteranno al neurone di attivarsi. Mediante la creazione di una rete di 3 neuroni che lavorano in parallelo, è possibile sfruttare l'approccio AllvsOne, infatti in questo modo ogni neurone è risponderà in maniera indipendente al ricevere degli input. Partiamo dal dividere il dataset in due sezioni, una relativa al training e una relativa al riconoscimento di pattern. Durante la prima fase di training, inizializzando 3 vettori w_1, w_2, w_3 casuali si permetterà di modificarli cosi da renderli adatti al peso desiderato con un numero finito di passi. Quindi attraverso questo processo è possibile creare i pattern da riconoscere. In seconda sede si sono utilizzati i vettori relativi ai pesi , e quindi il corrispettivo circuito atto a ricrearli, per far attivare i neuroni. La rete creata conterrà tre neuroni, ad agnuno di essi sarà associato un particolare pattern da riconoscere. Possiamo notare nel seguente istogramma come per un dato pattern non sempre il corrispettivo neurone si attiva, ma è presente sempre un certo "rumore" nella misura.



Ogni volta dati una combinazione di input e peso il computer quantistico ricrea caso per il caso gli stati sempre poichè non è possibile copiare

direttamente gli stati ma solamente ricostruirli. Quindi dato un modello da riconoscere, qualunque esso sia. Dopo aver eseguito il codice, e mostrato attraverso un grafico l'attivazione del percettrone dedicato al caso del dataset Iris, è mostrato in seguito il grafico relativo alla correzione del vettore w_i, dove è scopo illustrativo essendo i vari vettori w_i inizializzati casualemente, vediamo come al più impiegherà circa 7 cicli per arrivare alla giusta classificazione del vettore w_i in considerazione.



Come possiamo notare il nostro percettrone converge dopo un numero finito di passi, in questo caso sei epoche ed è capace dopo questo passaggio classificare correttamente i campioni dati in input.

Conclusioni

In conclusione, è stato proposto un modello di percettrone implementato direttamente su un dispositivo dotato di processore quantistico, ed è stato sperimentalmente sul quantum computer IBM-QX Tenerife basato sulla tecnologia a superconduttori. Il nostro algoritmo presenta un vantaggio esponenziale rispetto al modello di un percettrone classico, come abbiamo gia visto durante questa trattazione, è capace di rappresentare e classificare una stringa di 4 bit, quindi 4 pattern, usando 2 soli qubit. Il problema del vantaggio esponenziale richiederebbe una discussione a se stante. Un generico stato quantistico o un generica trasformazione unitaria richiede un esponenziale numero di gate elementari al fine di essere implementati, e questo potrebbe portare a nascondere i vantaggli della computazione quantistica. Questo rappresenta il principale problema degli algoritmi riferiti al quantum machine learning. Inoltre con l'aumentare del numero di qubit, sorgono determinati problemi dovuta al fatto di dover implementare molteplici gate di controllo, con molteplici qubit come controller, dato che al crescere della complessità serviranno sempre più qubit ausiliari per eseguire una misura non disturbata da errori. Comunque questa complicazione, dovuta ai limiti hardware del computer in uso, dipende strettamente dalla topologia utilizzata, e quindi come i qubit sono interconessi tra loro; Infatti è possibile creare classi di stati quantistici , come degli stati campioni ricreati innumerevoli volte, con una arbitraria precisione utilizzando un metodo basato sull'uso di numerose rotazioni governate da multi-controlled gate. Con queste premesse possiamo notare come, con un evoluzione dell'hardware, l'ottimizzazione di sistemi di machine learning quantistici siano realizzabili, addirittura utilizzando un sistema di codifica ad hoc sia possibile sfruttare completamente il vantaggio della scala esponenziale migliore di qualsiasi implementazione basata su hardware classico. In questa trattazione è stato possibile seguire le linee guida indicate dai lavori di McCollought-Pitts e Rosenblatt al fine di creare una rete semplice, un neurone, ristetto al caso di vettori in input binari. Partendo da questa idea utilizzando un approccio semi-quantistico si è stato in grado di creare un sistema di multi neuroni capace di distinguere vari pattern di un dataset utilizzando un approccio AllvsOne. Nonostante questo approccio funzioni in maniera eccelsa per il dataset scelto, non può essere considerato come approccio universale per qualsiasi dataset, infatti il principale problema del neurone, che non dipende dal tipo di hardware utilizzato, è che un vettore

relativo al learning può convergere se e solo se le classi del dataset in studio sono separabili, in caso contrario il percettrone continuerà ad aggiornare i vettori all'infinito. Mediante una collaborazione tra computer classico e quantistico è stato possibile implementare un algoritmo ibrido che permette il confronto degli input da classificare con tutti i vettori di training, questa modifica ha permesso quindi di far evolvere l'algoritmo rispetto al solo caso quantistico. Questo codice ibrido permette il confronto tra tutti i dati di input con tutti i dati di training.

Sviluppi futuri

Una possibile strada da percorrere al fine di ottimizzare il nostro neurone quantisto come classificatore binario è quella di cambiare approccio riguardante l'encoding delle informazioni in input, durante questa trattazione la codifica è realizzata sfruttando i sign-flip e phase-flip i quali rispettivamente permettono di passare dalla base $|0>$ a alla base $|1>$ e viceversa mentre i secondi permettono di modificare la fase di π così da creare una corrispondenza biunivoca con una base binaria, si comprende come in questo caso durante il fenomeno di codifica sia limitato; un approccio al fine di ottimizzare la codifica sarebbe quello di sfruttare la capacita dei qubit di poter variare con continuità da una base all'altra, ovvero può assumere qualsiasi gradazione di grigio su una scala da bianco a nero, e proprio sfruttando questa caratterista si potrebbe pensare di sfruttare le varie "tonalità" di grigi al posto di sfruttare solo il "bianco o nero". Ulteriore caratteristica è la limitazione del nostro hardware quantistico reale che non ci permette di implementare un modello di percettrone che permette di riconoscere più classi contemporaneamente data l'impossibilità di realizzare collegamenti tra molteplici qubit. Al di la delle aspettative negli ultimi sono stati fatti passi da gigante nel campo della computazione quantistica, basti gaurdare che fino a qualche anno la sola realizzazione delle simulazioni richiedeva un'enormità di tempo per la preparazione del set, mentre oggi grazie allo sviluppo di Qiskit è possibile non solo simulare codice che potrebbe essere eseguito solamente da hardware quantistico ma addirittura lavorare da remoto.

Bibliografia

[1] Sebastian Raschka, Vahid Mirjalili (2017) *Python Machine Learning Machine and Deep Learning with Python, scikit-learn and Tensor Flow*

[2] Dr. Christine Corbett Moran (2019) *Mastering Quantum Computing with IBM QX*

[3] Yudong Cao, Gian Giacomo Guerrischi, Alan Aspuru-Guzik (2017) *Quantum Neuron: an elementary building block for machine learing on quantum computers*

[4] Francesco Tacchino, Chiara Macchiavello, Dario Gerace, Daniele Bajoni (2018) *An Artificial Neuron Implemented on an Actual Quantum Processor*

[5] Noson Yanofsky , Mirco Mannucci (2008) *Quantum computing for computer scientist*

Ringraziamenti

Ringrazio il mio relatore per avermi seguito durante il termine di questo percorso universitario e per avermi concesso la possibilità di partecipare ad un progetto coinvolgente, ringrazio i miei amici i quali mi hanno sostenuto. Sono grato ai miei genitori che mi hanno spinto a non mollare nemmeno per un secondo e spingermi verso la versione migliore di me, ringrazio la mia ragazza che mi ha sempre aiutato a vedere le cose da un punto di vista esterno così da comprendere al meglio il percorso effettuato. Ringrazio in fine me per essere arrivato a questo punto.