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 Magistrale in Fisica

Reti Neurali per il Mapping di Circuiti su Processori Quantistici

Relatori:
Acampora Giovanni

Candidato:
Schiattarella Roberto
Matr. N94000529

Anno Accademico 2019/2020

Indice

Introduzione **4**

1 Elementi di Quantum Computation e Quantum Compiling **7**
1.1 Introduzione alla Quantum Computation 7
1.1.1 Il Qubit 7
1.1.2 Sistemi di più Qubit 9
1.1.3 Circuiti quantistici 10
1.1.4 La Decoerenza 14
1.1.5 Stato dell'Arte degli Algoritmi Quantistici 15
1.2 Introduzione al Quantum Compiling 16
1.2.1 L'implementazione del Qubit 16
1.2.2 Il Processore Quantistico 17
1.2.3 Qiskit 19
1.2.4 Il Transpiler 20
1.2.5 Attuali Problematiche nel Processo di Compilazione . . . 25

2 Elementi di Machine Learning **26**
2.1 Definizione di Apprendimento per una Macchina 26
2.2 Il formalismo generale in un problema di Machine Learning . . . 27
2.3 Le Reti Neurali Artificiali 29
2.4 Il Training 31
2.4.1 La Stima dei Parametri via Backpropagation 33
2.5 Overfitting e Bias-Variance Tradeoff 34

3 Reti Neurali per il Mapping dei Qubit **40**
3.1 Una Rete Neurale per il Mapping dei Qubit: Setup del Modello 40
3.1.1 Il Modello Costruito 40
3.2 Analisi a Posteriori delle Predizioni 43

4 Analisi dei Risultati **46**
4.1 La Generazione del Dataset 46
4.1.1 Analisi del Dataset 49
4.2 Training e Testing del Modello 51
4.3 Valutazione del Modello 54

Conclusioni **57**

Elenco delle figure **60**

Bibliografia **62**

Introduzione

Il campo della computazione quantistica nasce con i lavori di Yuri Manin [1], Richard Feyman [2] e David Deutsch [3] nell'ultimo ventennio del secolo scorso. Lo scopo di un computer quantistico è quello di risolvere problemi computazionalmente intrattabili per calcolatori classici sfruttando principi di meccanica quantistica come l'entanglement o la sovrapposizione di stati.
La Quantum Computation ha già mostrato impatti significativi in diverse aree come la *Chimica Quantistica* [4][5], la *Crittografia* [6], il *Machine Learning* [7] ed altri. Sfortunatamente però, esiste ancora un grosso gap tra le risorse richieste dagli algoritmi quantistici proposti e le risorse disponibili negli attuali prototipi hardware. Gli attuali processori quantistici infatti, possono contare su un numero di qubit, gli equivalenti quantistici dei bit classici, molto limitato.
In quella che viene definita la *Noise Intermediate-Scale Quantum* (NISQ) computers era sono di pubblico accesso strutture hardware come quelle di IBM che contano fino a 20 qubit [8] mentre strutture con un numero di qubit maggiore sono state annunciate per i prossimi anni. Ci si aspetta nel breve e lungo periodo di avere a disposizione processori composti da centinaia se non migliaia di qubit: strutture del genere dovrebbero garantire quella che viene definita la *quantum supremacy* [9][10] a patto che si riesca a limitare in maniera adeguata l'errore commesso nell'esecuzione dei vari algoritmi.
Un numero di qubit cosi ridotto fa si che sia ancora impossibile implementare qualsiasi tipo di *error correction code* all'interno dei circuiti che codificano gli algoritmi da eseguire [11] e di conseguenza il ruolo svolto dal processo di compilazione di tali circuiti è fondamentale nella computazione quantistica attuale.
Un compilatore quantistico ha il compito di trasformare un dato circuito di input, che viene solitamente scritto dai programmatori senza tener conto dei vincoli e delle restrizioni fisiche che un attuale processore quantistico presenta, in un circuito quantistico di output che tali vincoli li rispetti e che possa quindi essere fisicamente eseguito sulla macchina a disposizione, minimizzando la probabilità di commettere errori durante l'esecuzione.
Dato il ruolo fondamentale che il processo di compilazione riveste attualmente nel campo della computazione quantistica, la ricerca nel mondo del *quantum compiling* si è intensificata molto negli ultimi anni, portando alla nascita di un efficiente compilatore per le architetture hardware di IBM, ossia il *transpiler* [12] di Qiskit, libreria software scritta in Python [13].
Tra le operazioni che questo svolge c'è quella di mappare i qubit virtuali di un dato algoritmo quantistico scritto sottoforma di circuito, su i qubit fisici di un dato processore. Questi ultimi, date le restrizioni tecnologiche che si hanno in

questa prima era di computer quantistici, non sono tutti connessi tra loro, ma per ogni processore è presente una *coupling map* che mette in relazione i diversi qubit. Qualsiasi operazione che coinvolge due qubit, tra cui la fondamentale operazione logica del CNOT, può essere fisicamente eseguita soltanto se i due qubit interagenti sono mappati in due qubit fisici accoppiati sulla coupling map del processore. Al contrario se gli stati dei due qubit coinvolti dal gate non sono collegati sulla coupling map devono essere spostati mediante l'inserimento di porte di SWAP all'interno del circuito, fino a quando non saranno mappati su due qubit fisici collegati. Questi inserimenti aumentano la probabilità di commettere errori durante l'esecuzione di un dato algoritmo e aumentano la profondità del circuito da eseguire minando la stabilità del sistema dettata dai tempi di decoerenza dei qubit. L'importanza di un mapping iniziale ottimale, che richiede quindi un inserimento minimo di SWAP durante l'esecuzione dell'algoritmo è di fondamentale importanza.
Nel presente lavoro di tesi si è proposto un innovativo metodo di mapping iniziale definendo un modello di rete neurale adatto allo scopo. Questo è il primo tentativo di inserire il machine learning nel processo di compilazione quantistica con l'obiettivo di ovviare al forte problema di scalabilità all'aumentare del numero di qubit di cui molti degli attuali algoritmi di mapping soffrono.

La combinazione di machine learning e quantum computing è, di recente, oggetto di intensi studi da parte della comunità scientifica, che hanno portato alla nascita del settore di ricerca noto come *quantum machine learning*, dove si tentano di definire algoritmi di machine learning da eseguire direttamente su computer quantistici. La simbiosi è quasi naturale se si pensa che anche il può semplice algoritmo di machine learning comporta il lavorare con un elevatissimo numero di parametri, i quali richiedono un forte potere computazionale per essere gestiti mentre i computer quantistici forniscono, in linea di principio, per loro natura tale potere. Questo lavoro di tesi, invece, tenta di seguire una strada inversa a quella del quantum machine learning, cercando di utilizzare il machine learning a supporto del processo di computazione quantistica. La possibilità di una rete neurale di lavorare con un numero di parametri elevato, permette a questa di apprendere modelli anche estremamente complessi, tra cui ad esempio quello che cerca di trovare un mapping ottimale tra tutti quelli forniti dal transpiler di IBM.
Il presente lavoro di tesi è allora volto a dimostrare che un applicazione di un sistema di rete neurale è possibile per effettuare un mapping iniziale quanto più vicino all'ottimale possibile.

L'elaborato si struttura in una prima parte di concetti basilari della computazione e della compilazione quantistica, passando poi ad una introduzione a quelli che sono i fondamenti del machine learning e dei modelli di rete neurale.
La seconda parte del lavoro è invece incentrata sul modello messo a punto. Si noti come, essendo questo il primo tentativo di applicare reti neurali per eseguire il mapping di circuiti quantistici su processori quantistici, il modello di rete è stato completamente sviluppato durante il lavoro di tesi, ottenendo seppure limitatamente al processore IBMQ_Burlington di IBM a 5 qubit risultati interes-

santi.
La parte finale dell'elaborato invece, è incentrata sulla costruzione del dataset ottenuto, proponendo un metodo di collezione di dataset utilizzabili da algoritmi di machine learning che potrà essere sfruttato facilmente anche da futuri lavori. La tesi si conclude con un'analisi dei risultati ottenuti.
Si vuole infine sottolineare che nel corso di tutto l'elaborato si è cercato, ove possibile, di non inserire i codici scritti sia per la parte di generazione del dataset che per la parte di costruzione ed utilizzo del modello di rete neurale. Il lettore interessato può trovare la parte di codice all'indirizzo github *Github Repository Link*

Capitolo 1

Elementi di Quantum Computation e Quantum Compiling

In questo capitolo saranno esaminati i concetti fondamentali della computazione quantistica. Dopo una prima parte dedicata alla descrizione delle unità di informazione quantistica, i *qubit*, si passerà alla descrizione di come gli algoritmi quantistici vengono implementati secondo una codifica circuitale e si analizzerà uno dei principali problemi della computazione quantistica, quale il fenomeno della *decoerenza*.
Tali nozioni saranno necessarie per capire a fondo il problema della compilazione quantistica, e come questa è necessaria ad oggi per limitare la probabilità di commettere errori in fase di esecuzione di un determinato algoritmo. L' ultima parte del capitolo sarà quindi riservata alla descrizione generale del processo di compilazione, con particolare attenzione all'operazione di mapping di qubit virtuali di un dato circuito quantistico sui qubit fisici di un processore. Questo processo è quello che si vuole eseguire mediante l'utilizzo di reti neurali profonde come proposto nei capitoli finali dell'elaborato.

1.1 Introduzione alla Quantum Computation

1.1.1 Il Qubit

Il bit è il concetto fondamentale per la computazione classica. La computazione quantistica è costruita su un concetto analogo, il *quantum bit* (*qubit*).
In questa sezione verrà data un interpretazione del qubit da un punto di vista prettamente matematico, il che permetterà di generalizzare i concetti esposti all'intera teoria della computazione quantistica non vincolandosi a uno specifico sistema di realizzazione.
Dunque se il bit classico è uno stato di un sistema binario, normalmente descritto da 0 ed 1, un qubit è uno stato di un sistema quantistico a due livelli (ad esempio lo stato di spin di una particella) che secondo la notazione di *Dirac* è usualmente

descritto come

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \quad con \quad |\alpha|^2 + |\beta|^2 = 1, \quad \alpha, \beta \in C \tag{1.1}$$

L'interpretazione del qubit è l'usuale interpretazione di uno stato quantistico: le quantità $|\alpha|^2$ e $|\beta|^2$ descrivono la probabilità che all'atto della misura il qubit vada a trovarsi rispettivamente nello stato 0 o nello stato 1.
Complessivamente, allora, si può concludere che lo stato di un qubit è descritto da un vettore in uno spazio di Hilbert H, la cui base è data dai vettori $\{|0\rangle, |1\rangle\}$.

Una rappresentazione molto utilizzata per descrivere lo stato e la natura di un qubit è una rappresentazione di tipo geometrica.
Sfruttando la proprietà $|\alpha|^2 + |\beta|^2 = 1$, è possibile riscrivere lo stato del qubit come

$$|\psi\rangle = e^{i\gamma}\left(cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}sen\frac{\theta}{2}\,|1\rangle\right) \tag{1.2}$$

Dove θ, γ e ϕ sono numeri reali. E' inoltre possibile trascurare il fattore di fase $e^{i\gamma}$ siccome questo non produce effetti osservabili e quindi complessivamente è possibile scrivere

$$|\psi\rangle = cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}sen\frac{\theta}{2}\,|1\rangle \tag{1.3}$$

I numeri θ e ϕ definiscono un punto su di una sfera tridimensionale come in Figura 1.2. Tale sfera è spesso chiamata **sfera di Bloch**.
Molte delle operazioni effettuabili su singoli qubit, descritte nella sezione 1.1.3, saranno facilmente intuibili grazie alla rappresentazione geometrica appena presentata. Tuttavia questa risulta limitata in quanto non è possibile utilizzarla per rappresentare stati composti da più qubit.

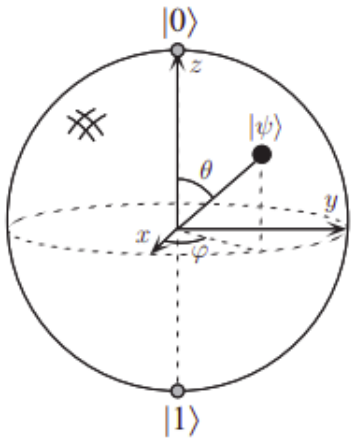

Figura 1.1: Rappresentazione grafica di un qubit

E' però errato pensare che siccome lo stato di un qubit è rappresentato da un punto sulla sfera di Bloch e i punti su una sfera sono infiniti, allora l'informazione rappresentata da un qubit è infinita. Tale considerazione non tiene conto del

processo di misura, che, come detto, è in grado di dare soltanto i valori 0 e 1. Inoltre va tenuto presente che il processo di misura stesso modifica lo stato del qubit secondo i postulati della meccanica quantistica [14], facendo si che questo collassi dallo stato di sovrapposizione $\alpha|0\rangle + \beta|1\rangle$ allo stato $|0\rangle$ o $|1\rangle$ corrispondente al valore misurato.

1.1.2 Sistemi di più Qubit

Lavorando con computer quantistici, difficilmente capiterà di lavorare con sistemi ad un singolo qubit, ma piuttosto ci si troverà a descrivere stati di sistemi composti da un numero di qubit che negli anni si spera possa diventare sarà sempre maggiore.
Da un punto di vista prettamente matematico se un qubit è un vettore di stato di uno spazio di Hilbert $\mathcal{H}_1$, lo stato di un sistema composto da N qubit, secondo i principi della meccanica quantistica [14], sarà un vettore dello spazio di Hilbert prodotto tensoriale dei singoli spazi di Hilbert $\mathcal{H}_i$

$$\mathcal{H} = \mathcal{H}_1 \otimes .. \otimes \mathcal{H}_N \tag{1.4}$$

Se ci si limita al caso semplice di un sistema formato da due qubit, allora lo spazio di Hilbert avrà quattro vettori di base $[|00\rangle, |01\rangle, |10\rangle, |11\rangle]$ ed il generico vettore di stato del sistema sarà individuato dalla sovrapposizione di questi stati

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \quad con \quad \sum_{x\in(0,1)^2} |\alpha_x|^2 = 1 \tag{1.5}$$

Tale stato verrà al solito interpretato secondo l'idea che un operazione di misura restituirà il risultato $x = (00, 01, 10, 11)$ con probabilità $|\alpha_x|^2$, facendo si che lo stato collassi nel vettore $|x\rangle$.

Un processo di misura su un sistema composto da più qubit avviene misurando i qubit in maniera separata. Se per semplicità si considera ancora una volta un sistema a due qubit e si va a misurare il primo qubit, questo darà 0 con probabilità $|\alpha_{00}|^2 + |\alpha_{01}|^2$ lasciando lo stato dopo la misura come

$$|\psi'\rangle = \frac{\alpha_{00}|00\rangle + \alpha_{01}|01\rangle}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} \tag{1.6}$$

Dove si noti che lo stato è stato riscalato della quantità $\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}$ affinché si mantenga la condizione di normalizzazione.

Quanto detto permette di formalizzare il concetto dell'*entanglement*. Formalmente infatti, due qubit si dicono entangled nel caso in cui lo stato complessivo non è esprimibile come prodotto tensoriale degli stati di singolo qubit.
Da un punto di vista pratico questo significa dire che l'operazione di misura su di un qubit influisce inevitabilmente sullo stato del secondo qubit, anche se questo si trova a distanza grande dal primo. Tale affermazione può essere elegantemente sintetizzata dicendo che la meccanica quantistica è una teoria fisica *non-locale*.

Generalizzando quanto detto a un sistema di N qubit, si più allora dire che la base del relativo spazio di Hilbert associato sarà della forma $|x_1, x_2.., x_N\rangle$ e lo stato di tale sistema sarà descritto da 2^N ampiezze. Per N = 500 questo numero è più grande del numero di atomi stimato nell'universo. Cercare di gestire un numero cosi grande di variabili sarebbe impensabile per qualsiasi computer classico. Questo enorme potenziale computazionale è quello che rende estremamente interessante la computazione quantistica.

1.1.3 Circuiti quantistici

In analogia con il modello circuitale classico, nell'ambito del quale si possono disegnare degli algoritmi utilizzando collegamenti e porte logiche, nel modello circuitale quantistico si utilizzano elementi analoghi per trasportare e manipolare l'informazione. E' utile a questo punto introdurre un formalismo matriciale, secondo il quale associamo ad un qubit singolo un vettore colonna

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \tag{1.7}$$

mentre rappresentiamo le porte agenti su un singolo qubit come matrici 2×2. Le porte logiche agiscono sui qubit secondo $|\psi'\rangle = U\,|\psi\rangle$; imponendo la conservazione della norma dello stato [14] si trova che le porte U accettabili sono unitarie.

$$\langle\psi'|\psi'\rangle = \langle\psi|\,U^\dagger U\,|\psi\rangle = \langle\psi|\psi\rangle = 1 \quad \forall\,|\psi\rangle \longrightarrow U^\dagger U = 1 \tag{1.8}$$

Questa è l'unica condizione richiesta per una porta logica quantistica (il ragionamento rimane valido anche per porte agenti su più qubit). L'unitarietà delle porte logiche ha delle implicazioni importanti per quanto riguarda l'entropia delle computazioni quantistiche: esse sono sempre invertibili.

Dal punto di vista grafico i collegamenti che trasportano i qubit si rappresentano come delle linee orizzontali, le porte logiche come dei quadrati e i circuiti si leggono da sinistra a destra (Figura 1.2)

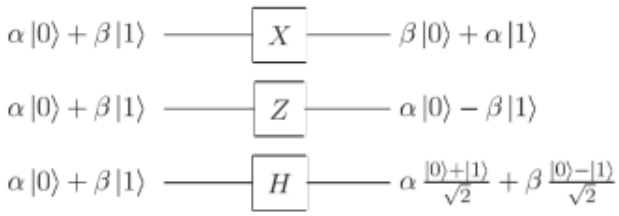

Figura 1.2: Tre esempi di circuiti quantistici, che illustrano l'azione delle porte logiche X, Z e H

Vi sono delle differenze fondamentali tra i circuiti logici classici e quelli quantistici. In primo luogo nei circuiti quantistici, non è permesso il feedback: essi si dicono *aciclici*.
In secondo luogo nei circuiti quantistici non può esistere il FANIN, operazione

in cui due o più qubit sono uniti in uno solo mediante una qualche porta logica. Operazioni come l'AND o l'OR classici, che mappano quattro stati (00,01,10,11) su due (0,1) non possono essere invertite, contrariamente a quanto richiesto dalla (1.8).
Per ultimo si osservi come il FANOUT, tramite il quale il ramo di un circuito logico classico si dirama in due o più bracci, comporti la copia dell'informazione trasportata dal ramo originale. Questo per il *no-cloning theorem* [14] è vietato, e dunque anche il FANOUT risulta impossibile.

Gates logici a singolo qubit

Se nel modello classico l'unica porta logica non banale di singolo bit è la porta NOT, nel modello quantistico esistono invece diverse operazioni di questo tipo. La naturale estensione del NOT classico alla logica circuitale quantistica è la porta X, che scambia i coefficienti dei vettori della base computazionale:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{1.9}$$

E' banale algebra la dimostrazione:

$$X \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} \tag{1.10}$$

Il simbolo X è stato scelto in riferimento alla matrice di Pauli $\sigma_x = X$.

Similmente sono definite, sempre in relazione alle rispettive matrici di Pauli, le porte:

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{1.11}$$

Un'altra porta molto utilizzata è la porta di *Hadamard*

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1.12}$$

essa trasforma un qubit preparato nello stato $|0\rangle$ nello stato $(|0\rangle + |1\rangle)/\sqrt{2}$.
E' possibile visualizzare l'operazione svolta dalla porta di Hadamard utilizzando la rappresentazione del qubit sulla sfera di Bloch. Come si vede dalla Figura 1.3 l'operazione svolta da questo gate corrisponde ad una rotazione del qubit attorno l'asse $\hat{y}$ di 90°, seguita da una rotazione attorno l'asse $\hat{x}$ di 180°

Le matrici unitarie 2×2 sono infinite. Dunque tendenzialmente esistono infiniti gate quantistici utilizzabili.

Va però sottolineato che esponenziando le matrici di Pauli X, Y, Z si ottengono tre *operatori di rotazione* in grado di far ruotare lo stato di un qubit, identificato dal punto (θ, ϕ) sulla sfera di Bloch, attorno gli assi $\hat{x}, \hat{y}, \hat{z}$ definiti dalle equazioni:

$$R_x(\theta) = e^{-i\theta X/2} = cos\left(\frac{\theta}{2}\right)\mathbf{I} - isen\left(\frac{\theta}{2}\right)\mathbf{X} = \begin{pmatrix} cos\frac{\theta}{2} & -isen\frac{\theta}{2} \\ -isen\frac{\theta}{2} & cos\frac{\theta}{2} \end{pmatrix} \tag{1.13}$$

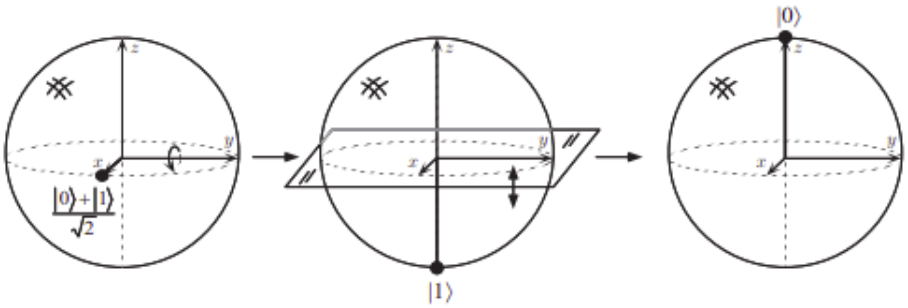

Figura 1.3: Visualizzazione sulla sfera di Bloch di una porta di Hadamard , agente su un stato di input $(|0\rangle + |1\rangle)/\sqrt{2}$

$$R_y(\theta) = e^{-i\theta Y/2} = cos\left(\frac{\theta}{2}\right)\mathbf{I} - sen\left(\frac{\theta}{2}\right)\mathbf{Y} = \begin{pmatrix} cos\frac{\theta}{2} & -isen\frac{\theta}{2} \\ sen\frac{\theta}{2} & cos\frac{\theta}{2} \end{pmatrix} \tag{1.14}$$

$$R_z(\theta) = e^{-i\theta Z/2} = cos\left(\frac{\theta}{2}\right)\mathbf{I} - isen\left(\frac{\theta}{2}\right)\mathbf{Z} = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \tag{1.15}$$

E' possibile dimostrare il seguente **teorema di *decomposizione Z-Y*** [15]: Se U è una matrice unitaria rappresentante un gate operante su un singolo qubit, allora esistono quattro numeri reali α, β, γ e δ tali che

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta) \tag{1.16}$$

Gates Logici a MultiQubit

Altro tipo di porte logiche fondamentali per la computazione quantistica, cosi come per quella classica, sono quelle che coinvolgono due qubit simultaneamente. E' di uso comune per questi gates l'utilizzo del termine *operazioni controllate*: una certa operazione viene eseguita condizionalmente a certi requisiti.
La più semplice di queste porte logiche è il CNOT (talvolta indicato con CX): questa esegue la negazione dello stato del qubit *target* a seconda dello stato del qubit di *controllo* (CNOT sta per *controlled-not*, ossia l'equivalente dello OR esclusivo classico).
La rappresentazione cirucitale di tale gate è quella presente in Figura 1.4, dove nella riga superiore è presente il qubit di controllo e in quella inferiore il target.

L'azione del CNOT può dunque essere riassunta come segue:

$$|00\rangle \to |00\rangle\,; \quad |01\rangle \to |01\rangle\,; \quad |10\rangle \to |11\rangle\,; \quad |11\rangle \to |10\rangle \tag{1.17}$$

E' facile verificare che la matrice unitaria associata al CNOT gate è rappresentata da

$$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{1.18}$$

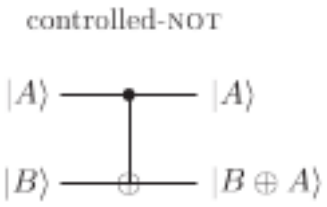

Figura 1.4: Rappresentazione circuitale del CNOT

E' possibile generalizzare l'idea del CNOT alla generica porta logica a doppio qubit U: se il qubit di controllo è 'alto' allora U è applicata al *target* qubit, altrimenti quest'ultimo è lasciato invariato.

$$|c\rangle\,|t\rangle \rightarrow |c\rangle\,U^c\,|t\rangle \tag{1.19}$$

Ancora più in generale, il numero di qubit di controllo N e quelli dei qubit bersaglio K può essere scelto a piacere. Un' esempio è rappresentato dalla porta di Toffoli riportata in Figura 1.5, dove i qubit di controllo sono N = 2, e se questi sono entrambi 'alti', il qubit di controllo viene negato.

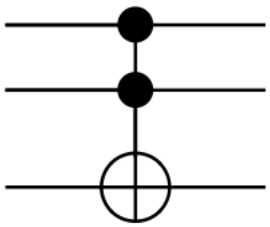

Figura 1.5: Rappresentazione circuitale della porta di Toffoli

Va detto che il gate CX è detto *entangling gate* [16], cioè la sua azione non può essere decomposta in nessuna azione di gate indipendenti agenti in maniera successiva e separata sullo stato dei qubit coinvolti dal CX.

Un importantissimo risultato teorico raggiunto dalla quantum computation è il fatto che l'insieme di tutte le porte a singolo qubit ed una *entangling gate* è sufficiente a descrivere tutte le porte realizzabili [15]. E siccome per quanto affermato nella (1.16) con una opportuna combinazione di $R_y(\theta)$ e $R_z(\phi)$ è possibile descrivere tutti i gates logici ad un qubit, allora $\{R_y(\theta)$, $R_z(\phi)$, CX$\}$ è una base univerale per le porte quantistiche: ovvero qualsiasi gate logico può essere decomposto in termini di queste.

Si riporta come esempio la decomposizione di una porta logica a due qubit molto utilizzata, la SWAP gate, in termini di CX. La porta di SWAP inverte lo stato dei due qubit su cui agisce, dunque se in entrata si ha lo stato $|a, b\rangle$

in uscita si avrà lo stato $|b, a\rangle$ in questo modo è possibile spostare qubit da un punto ad un altro del circuito quantistico senza violare il teorema del *no cloning*. La decomposizione dello SWAP consiste in tre CNOT consecutivi, dove il secondo ha il qubit di controllo e il qubit bersaglio invertiti rispetto al primo e al terzo (Figura 1.6)

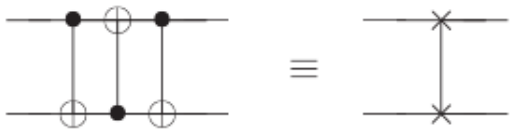

Figura 1.6: Decomposizione del gate di SWAP (a destra) in tre CX

Sfruttando l'interpretazione del CNOT come somma modulo due del qubit bersaglio con quello di target, ovvero :

$$CX\,|a, b\rangle = |a, a \oplus b\rangle \tag{1.20}$$

la dimostrazione della decomposizione diventa banale:

$$|a, b\rangle \to |a, a \oplus b\rangle \to |a \oplus (a \oplus b), a \oplus b\rangle \to |b, (a \oplus b) \oplus b\rangle = |b, a\rangle \tag{1.21}$$

1.1.4 La Decoerenza

Un concetto chiave che deve essere tenuto in considerazione per la realizzazione prima dei computer quantistici, poi degli algoritmi che su questi vengono implementati è quello della Decoerenza.
Con questo termine si intende la corruzione della desiderata evoluzione del sistema, ovvero il tempo in cui questo può esistere prima di perdere la coerenza degli stati. In altri termini si sta parlando essenzialmente della memoria quantistica del sistema.
Condizione importantissima per la computazione è dunque che il qubit mantenga la coerenza durante tutto il tempo necessario all'esecuzione di un'operazione logica. Questo vincolo si può esprimere con la disuguaglianza

$$\tau_Q \geq \tau_{op} \tag{1.22}$$

dove τ_Q è l'intervallo temporale durante il quale si ha coerenza quantomeccanica degli stati, e τ_{op} indica il tempo impiegato per un'operazione logica. Generalmente per la maggior parte dei sistemi questi due tempi sono tra loro correlati tanto che il rapporto ci fornisce una stima del numero di operazioni possibili prima che il qubit perda il suo set di condizioni iniziali

$$n_{op} = \frac{\tau_Q}{\tau_{op}} \tag{1.23}$$

grandezza che può variare notevolmente in un intervallo compreso tra 10^4 e 10^{14} operazioni a seconda del sistema preso in esame.

Va infine detto che le tipologie di decoerenza che si annoverano sono principalmente due:

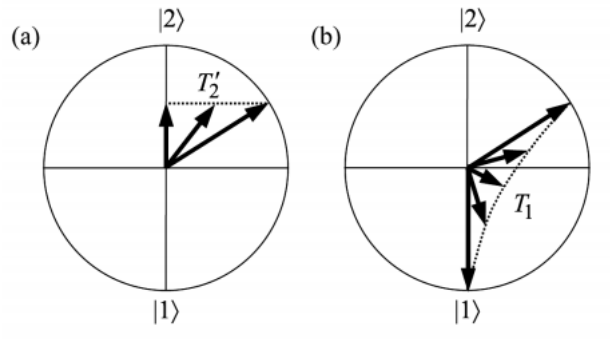

Figura 1.7: Rappresentazione geometrica del rilassamento trasverso a sinistra e del rilassamento longitudinale a destra

- **Rilassamento trasverso.** *Il rilassamento trasverso* è causato dalla perdita di coerenza tra le fasi relative delle ampiezze di uno stato quantistico. Il tempo di decoerenza che ne risulta si indica con T_2.
  E' possibile utilizzare il modello della sfera di Bloch per capire meglio questo tipo di fenomeno. Consideriamo uno stato $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ le cui ampiezze sono individuate da

$$\alpha = |\alpha|\, e^{i\phi_1} \quad \beta = |\beta| e^{i\phi_2} \tag{1.24}$$

  che geometricamente diventano

$$\alpha = sen\left(\frac{\theta}{2}\right) \quad \beta = e^{i\phi} cos\left(\frac{\theta}{2}\right) \quad con \quad \phi = \phi_2 - \phi_1 \tag{1.25}$$

  Dunque il punto sulla sfera di Bloch rappresentate lo stato del qubit è individuato dalle coordinate

$$x = 2\,\mathrm{Re}\,(\alpha\beta) \quad y = 2Im\,(\alpha\beta) \quad z = |\beta|^2 - |\alpha|^2 \tag{1.26}$$

  Variazioni casuali di ϕ possono portare all'annullamento dei termini $\alpha\beta$ e ciò proietta il vettore sull'asse z, non conservando nemmeno il modulo del vettore di Bloch (Prima immagine Figura 1.7. L'interpretazione geometrica spiega l'utilizzo del termine *trasverso* riferito a questo tipo di decoerenza.

- **Rilassamento longitudinale.** *Il rilassamento longitudinale* è dovuto al decadimento di popolazione: gli stati eccitati tendono a decadere spontaneamente allo stato fondamentale in un certo tempo tipico T_1. Esso è rappresentato dal secondo disegno in Figura 1.7: si può osservare che il rilassamento longitudinale implica anche il rilassamento trasverso

1.1.5 Stato dell'Arte degli Algoritmi Quantistici

Anche se una descrizione approfondita di algoritmi quantistici esula degli scopi di questo lavoro di tesi, è opportuno almeno riportare quale sia lo stato dell'arte attuale per questi ultimi.

Va inizialmente detto che al giorno d'oggi sono molteplici gli studi circa la cosiddetta *quantum supremacy* [9][10], ovvero la possibilità che la computazione quantistica possa effettuare operazioni che per i calcolatori classici sono impossibili. La promessa dei computer quantistici è quella di riuscire a svolgere certi tasks computazionali in un tempo esponenzialmente minore rispetto a quello impiegato da un processore classico [17].

Ci sono degli ambiti di ricerca in cui la computazione quantistica ha già mostrato importanti risultati. Questi campi sono del più disparato tipo. Si può ad esempio partire citando la capacità di computer quantistici di simulare sistemi quantistici. Questi calcolatori sono ad oggi già utilizzati ad esempio, nella *chimica quantistica* [4].
Altro campo di ricerca in cui si sono focalizzati gli studiosi di algoritmi quantistici è quello relativo alla computazione delle *trasformate di Fourier* [15] e, ancora, sono ad oggi sotto studio tra i vari, algoritmi di ricerca (*Groove's Algorithm*[15]), algoritmi di quantum machine learning [24], algoritmi di crittografia quantistica [6].

1.2 Introduzione al Quantum Compiling

Analizzati i concetti fondamentali della computazione quantistica quale il concetto di qubit, la codifica circuitale ad oggi utilizzata per l'implementazione di algoritmi quantistici ed i principali campi di applicazione di tali algoritmi, in questa sezione verranno discusse le specifiche hardware e software dei processori quantistici utilizzati durante il lavoro di tesi e gli step fondamentali alla compilazione, facendo riferimento alle tecnologie messe a disposizione da IBM.
Nella prima parte della sezione si troverà una breve introduzione all'hardware con il quale si è lavorato, ed una breve introduzione alla libreria software utilizzata. La restante parte del capitolo si sofferma sul formalizzare il problema della compilazione quantistica e su come questa è svolta oggi mediante il *transpiler* di Qiskit.

1.2.1 L'implementazione del Qubit

Il qubit fisico utilizzato da IBM è il cosiddetto *fixed-frequency superconducting transmon qubit* [18]. Un transmon qubit è un tipo di *charge qubit*, pensato per avere una ridotta sensibilità al charge noise.
Seguirà una breve spiegazione di come un charge qubit funziona, seppur un'analisi profonda esula dagli scopi di questo lavoro.

Un charge qubit è un qubit basato sul funzionamento di una giunzione Josephon [19][20]. Una giunzione Josephon consiste di due regioni superconduttive separate generalmente da un isolante. La bassa dissipazione che caratterizza i superconduttori rende possibile, in linea di principio, lunghi tempi di decoerenza. L'ingrediente fondamentale affinché un materiale si comporti da superconduttore è la presenza di coppie di Cooper: coppie di elettroni legati che formano sistemi a

spin intero comportandosi dunque come bosoni. Queste coppie in una giunzione possono fare effetto tunnel attraverso la barriera di isolante.

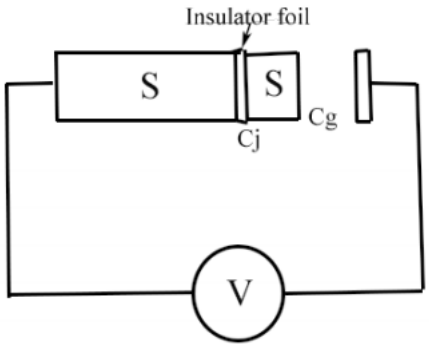

Figura 1.8: Rappresentazione circuitale di una giunzione Josephon

In Figura 1.8 è presente un esempio di circuito contente una giunzione Josephon: questo è composto da un isolante con una capacità intrinseca C_j, una seconda capacità C_g ed infine una tensione applicata V. Il superconduttore tra l'isolante e la capacità è chiamato *island*.
Detto n il numero quantico che descrive il numero di coppie di Cooper presenti nell'island, i due livelli del qubit sono da intendersi come interazione tra lo stato $|n = 0\rangle$ e $|n = 1\rangle$.

1.2.2 Il Processore Quantistico

Un processore quantistico è un insieme di qubit fisici che idealmente sono connessi tra loro. Nei fatti a causa di limitazioni spaziali non è possibile far comunicare tutti i qubit di un processore, ma piuttosto su questo è definita una *coupling map*. Un esempio è quello in Figura 1.9, dove è presente la struttura del processore '*ibmq_melbourne_16*'.
E' possibile notare che la connessione tra i qubit è rappresentata da una freccia: questa indica la direzione in cui un operazione che coinvolge due qubit può essere eseguita: la freccia va dal qubit di controllo al qubit bersaglio.

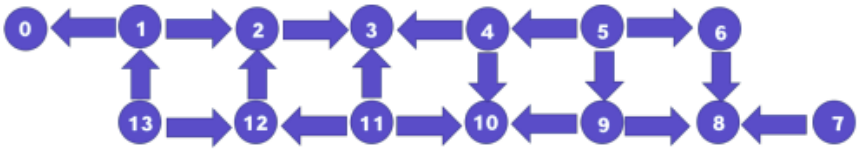

Figura 1.9: Coupling Map del processore 'ibmq_melbourne_16' a 16 qubit. Le frecce indicano la direzione in cui i gates a due qubit possono essere eseguiti.

Essendo il presente lavoro di tesi un punto di partenza per l'utilizzo di reti neurali (Capitolo 2) al problema della compilazione di algoritmi quantistici, si è scelto inizialmente di lavorare con un processore più piccolo di quello di Melbourne, quale il processore *ibmq_burlington* a 5 qubit, la cui coupling map è

rappresentata in Figura 1.10.

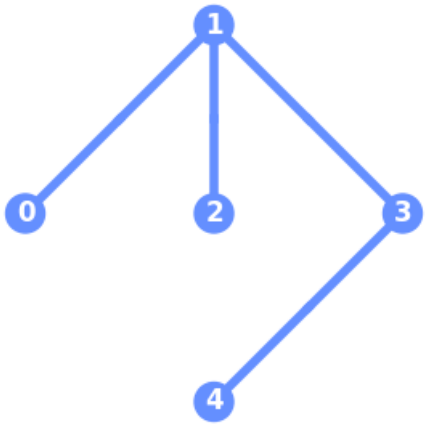

Figura 1.10: Coupling Map del processore 'ibmq_burlington' a 5 qubit.

Si noti come la non presenza di frecce tra i nodi del grafo, sta ad indicare che i qubit adiacenti possono interagire senza una direzione fissata. Ossia una qualsiasi porta a due qubit può essere definita tra due qubit collegati sulla coupling map senza dover preoccuparsi della direzione della porta.

I Gates di Base e gli Error Rates

Quando si scrive un circuito quantistico si è liberi di utilizzare qualsiasi quantum gate che rispetti la proprietà di essere descritto da una matrice unitaria, e qualsiasi operazione non legata a gates come operazioni di misura o di reset. Quando, però, si esegue un circuito su un processore quantistico reale non si ha più tale flessibilità.
A causa di limitazioni fisiche, come ad esempio l'interazione fisica tra i qubit, la difficoltà nell'implementazione di multi-qubit gates, controlli elettronici etc., un computer quantistico può supportare soltanto specifiche porte e specifiche operazioni non legate a porte.
Ovviamente quando si inizia a lavorare con un processore del tipo IBM Q, come ad esempio 'ibmq_burlington', è possibile sapere quali sono le porte native per quel processore.

Nel caso del processore 'ibmq_burlington' la base di gates nativi è rappresentata da:

- *u1*, che implementa un phase shift con angolo λ:

$$u1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix} \tag{1.27}$$

- *u2* che permette di creare stati di sovrapposizione

$$u2(\phi,\lambda) = \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\phi+\lambda)} \end{pmatrix} \tag{1.28}$$

  Un esempio di gate realizzato tramite la porta u2 è la porta di Hadamard: H = u2(0, π)

- *u3*, questa è la più generale trasformazione unitaria implementata

$$u3(\theta,\phi,\lambda) = \begin{pmatrix} cos(\theta/2) & -e^{i\lambda}sen(\theta/2) \\ e^{i\phi}sen(\theta/2) & e^{i(\phi+\lambda)cos(\theta/2)} \end{pmatrix} \tag{1.29}$$

  Un esempio di gate realizzato tramite la porta u3 è il NOT gate X: X = u3(π, 0, π)

- Il CNOT definito nella 1.18

E' importante sottolineare che l'applicazione di un certo gate su di un qubit fisico ha una probabilità non nulla di fallire. Ogni gate, quindi, ha un proprio *error rate* che è possibile definire come il rapporto $\frac{n_g^f}{n_g^s}$, dove n_g^f è il numero di applicazioni del gate risultate errate durante una fase di calibrazione, e n_g^s è invece il numero di applicazioni del gate risultate esatte durante la stessa fase. Questa fase di calibrazione è effettuata due volte al giorno per i dispositivi di IBM, e i risultati di tali calibrazioni sono ovviamente accessibili.
Tipicamente questi error gate di singolo qubit sono molto piccoli, ordine 10^{-4}. Alla stessa maniera, va sottolineato, che anche il CNOT (porta universale a due qubit) ha una probabilità di errore non nulla. Anzi, tipicamente il CNOT error rate è di uno/due ordini di grandezza maggiore rispetto al error rate delle porte a singolo qubit, essendo ordine 10^{-2}.
In Figura 1.11 è presente un esempio di schema di calibrazione fornito da IBM per il processore 'ibmq_burlington'.

L'applicazione fisica dei gates è fatta mediante impulsi sui qubit fisici. L'analisi degli steps che compongono un gate è aldilà degli scopi di questa tesi, ma è opportuno tener presente che queste operazioni richiedono un tempo non nullo di esecuzione. Ogni applicazione di un CNOT, ad esempio, ha un tempo tipico (*length*) che è dell'ordine delle centinaia di nanosecondi.

1.2.3 Qiskit

Qiksit [13] è l'interfaccia software, scritta come libreria Python, sviluppata per eseguire esperimenti e simulazioni sulle piattaforme di IBM. Questa è composta da quattro macroaree, chiamate *'elements'*:

1 *Qiskit Terra.* Il suo scopo è quello di fornire strumenti utili per comporre programmi quantistici a livello di circuiti ed impulsi, ed infine ottimizzare questi affinchè possano rispettare i vincoli fisici del dispositivo su cui andranno ad essere eseguiti.

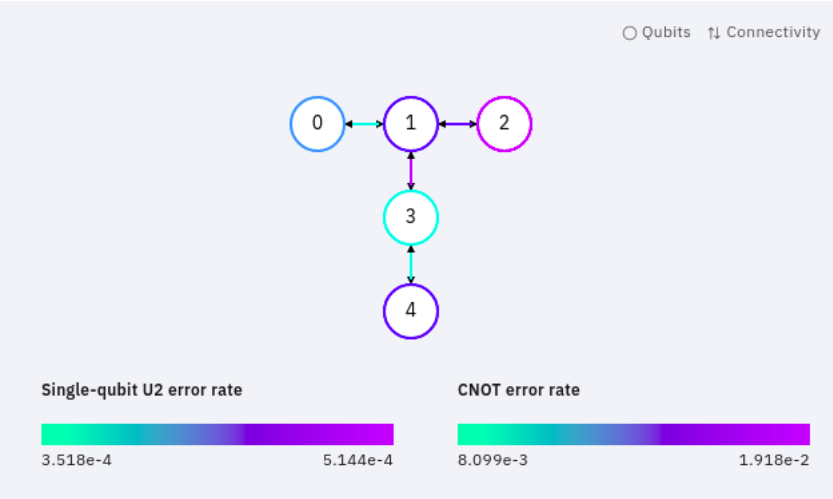

Figura 1.11: Coupling Map del processore 'ibmq_burlington' a 5 qubit, con schema di calibrazione

2 *Qiskit Aer.* Questa è la parte utile ad eseguire simulazioni numeriche. E' possibile utilizzare Aer per verificare che il processore funzioni nella maniera adeguata, simulando con metodi numerici l'evoluzione teorica del sistema.

3 *Qiskit Ignis.* Lo scopo di Ignis è quello di procedere alla limitazione di rumore ed errori. Questa infatti include strumenti di errors correction e strumenti utili a calcolare e prevedere la presenza di errore durante l'esecuzione di un determinato codice.

4 *Qiskti Aqua.* Questa parte è dedicata a tutte le possibili applicazioni della computazione quantistica, fornendo strumenti utili alla chimica, all'intelligenza artificiale o all'ottimizzazione.

La parte largamente utilizzata nel corso del lavoro è quella di qiskit Terra, che permette di interfacciarsi tra le varie, anche direttamente con il compilatore di cui sono provvisti i computer di IBM, detto *IBM Transpiler*.

1.2.4 Il Transpiler

Ricapitolando, un algoritmo quantistico viene scritto in termini di un circuito nel quale vengono inserite operazioni logiche, rappresentate da particolari gates, che permettono di ottenere, auspicabilmente, l'output desiderato.
Molto spesso però, visto i limiti fisici di un processore quantistico non è possibile lanciare il circuito virtuale sull'hardware senza prima rimodularlo nella maniera opportuna.
L'insieme di queste trasformazioni è il compito del compilatore quantistico: un circuito virtuale deve essere trasformato e ottimizzato per rispettare i vincoli hardware presenti e massimizzare la probabilità di successo dell'algoritmo, minimizzando quelli che sono gli effetti di rumore.
Questa operazione è un operazione molto complessa ed è stato provato che la

ricerca di un circuito ottimale, dato un certo circuito in input è un problema computazionale di tipo NP-completo[1] [21] .

Questa operazione di compilazione per circuiti scritti con Qiskit ed eseguiti su piattaforme IBM Q è svolta dal *transpiler*. Le macro-operazioni da questo svolte sono riassunte in Figura 1.12

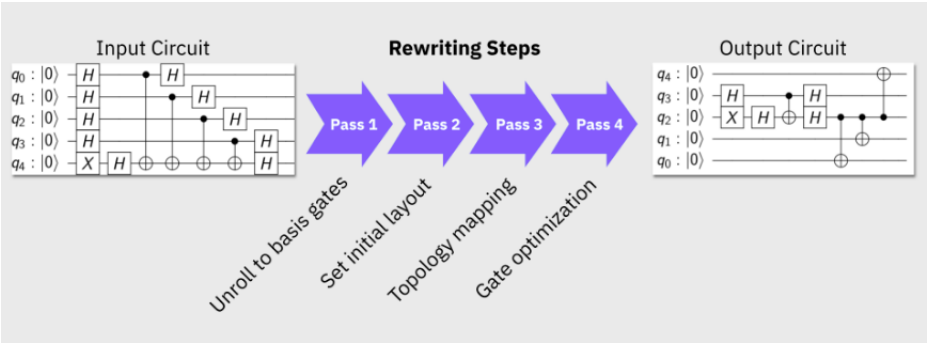

Figura 1.12: Operazione di compilazione di un circuito quantistico

Si proseguirà adesso analizzando i passaggi fondamentali eseguiti in ognuna di queste.

Unroll to Basis Gates

Come detto ogni processore quantistico ha solo una determinata collezione di gates che possono essere fisicamente implementati e nella sezione 1.2.2 si è visto quale fosse la collezione di porte di base disponibili sul processore in uso, 'ibmq_burlington' a 5 qubit.

Con l'operazione di *unrolling*, dunque, si intende la decomposizione dei gates presenti nel circuito virtuale di input, nelle porte fisicamente implementabili su quel determinato dispositivo.
Si consideri ad esempio, un circuito virtuale come quello in Figura 1.13.
In questo circuito sono presenti le porte H, X e controlled-U_1, nessuna delle quali è presente nel set di porte fisicamente implementabili in un hardware IBM Q.

[1]Nella teoria della complessità computazionale i problemi NP-completi sono i più difficili problemi della classe NP ('problemi non deterministici in tempo polinomiale') nel senso che, se si trovasse un algoritmo in grado di risolvere 'velocemente' (nel senso di utilizzare un tempo polinomiale) un qualsiasi problema NP-completo, allora si potrebbe usarlo per risolvere 'velocemente' ogni problema NP. Un esempio di problema NP-completo è il problema delle somme parziali, cioè: dato un insieme finito di numeri interi, determinare se esiste un sottoinsieme tale che la somma dei suoi elementi sia zero. E' evidente che è facile verificare velocemente se un sottoinsieme è o meno una soluzione del problema, ma non è noto alcun metodo per trovare una soluzione che sia sensibilmente più veloce di provare tutti i possibili sottoinsiemi tranne i due che contengono tutti numeri concordi(tutti i positivi o tutti i negativi), quelli formati da un solo numero negativo e da tutti i numeri positivi maggiori in valore assoluto al numero negativo e quelli formati da un solo numero positivo e da tutti i numeri negativi maggiori in valore assoluto al numero positivo.

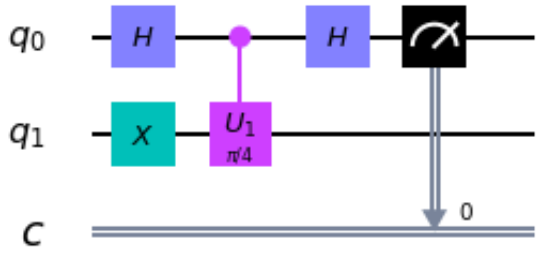

Figura 1.13: Esempio di circuito quantistico a due qubit

L'operazione di unrolling permette di ottenere il circuito in Figura 1.14, dove si può notare la decomposizione delle porte H ed X rispettivamente in $u_2(0, \pi)$ ed $u_3(\pi, 0, \pi)$.

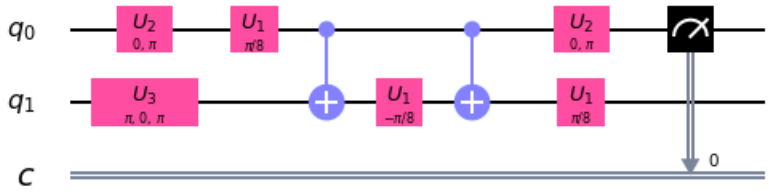

Figura 1.14: Esempio di unrolling

Vanno sottolineate due cose: la prima è il fatto che con l'operazione di unrolling il circuito diventerà più profondo, in questo esempio il numero di porte da eseguire ad istanti di tempo diversi (*depth* del circuito) passa da 4 a 7. La seconda cosa è che se anche in origine c'era una sola porta a due qubit, la decomposizione comporta la presenza di più di un singolo CNOT. In sintesi, l'operazione di unrolling aumenta la depth del circuito ed il numero di gates.

Layout Iniziale

I circuiti quantistici sono entità astratte, i cui qubit sono 'virtuali' rappresentazioni di qubit reali utilizzati nella computazione. L'operazione della scelta del layout iniziale da parte del transpiler consiste nel mappare in una maniera *one-to-one* i qubit virtuali in quelli fisici del dispositivo quantistico sul quale verrà lanciato l'algoritmo.

Graficamente l'operazione di mapping è quella presentata in Figura 1.15, dove un circuito quantistico da 5 qubit viene mappato su di un processore a 20 qubit.

L'operazione di mapping è la parte della compilazione su cui si è lavorato nel presente progetto di tesi. Si è infatti proposto una nuova tecnica di mapping basata sul funzionamento di una Deep Neural Network, utile a sostituire le tec-

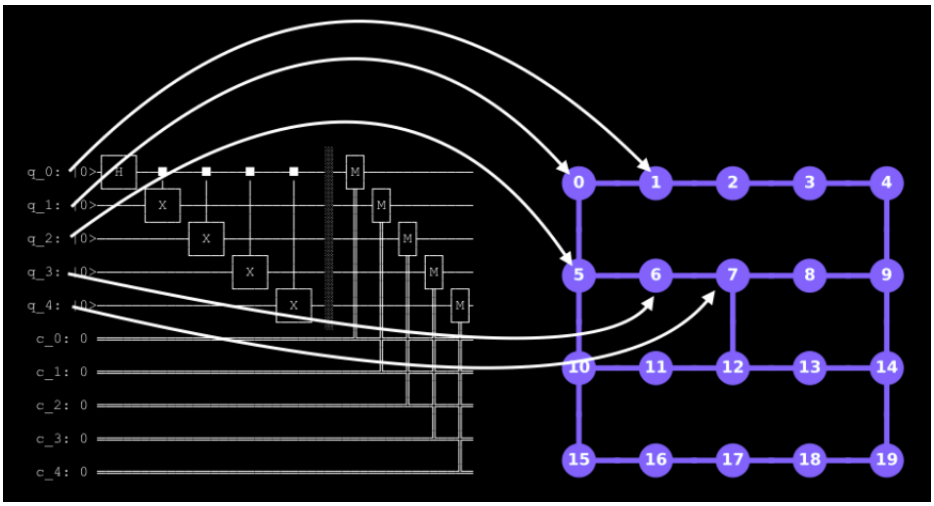

Figura 1.15: Rappresentazione grafica dell'operazione di mapping dei qubit virtuali di un circuito quantistico, su un dispositivo hardware a 20 qubit

niche ad oggi utilizzate, delle quali i limiti risultano evidenti nel momento in cui si lavora con un numero di qubit grande [22] [23].

Per capire l'importanza di un efficiente operazione di mapping basti pensare che durante l'esecuzione del circuito quantistico può capitare che siano presenti operazioni di CNOT tra qubit virtuali che non sono stati mappati su qubit fisici comunicanti in base alla topologia dell'hardware. Dunque affinchè sia resa possibile l'implementazione fisica di CNOT tra qubit non comunicanti, uno o più gates di SWAP devono essere inseriti all'interno del circuito, per spostare lo stato dei qubit interagenti mediante CNOT su slot comunicanti tra loro.
Come si è visto però, un gate di SWAP (Figura 1.6) è composto da tre porte CNOT. Questo ovviamente diminuisce la probabilità di corretta esecuzione dell'algoritmo per due ragioni fondamentali: la prima è il fatto che aggiungere gates all'interno del circuito aumenta la profondità di questo e quindi il suo tempo di esecuzione, che non può superare il tempo in cui lo stato dei qubit rimane coerente; la seconda è il fatto che l'inserimento di SWAP rappresenta un aumento molto importante del rumore a cui sono soggetti gli stati dei qubit a causa della decomposizione di questi in tre CNOT gates ognuno con error rate medio dell'ordine di 10^{-2}.
Dunque trovare il numero minimo di porte di SWAP necessario a mappare un dato circuito su un dato dispositivo è uno step fondamentale nel processo di esecuzione di un algoritmo.

L'operazione di mapping iniziale dunque, deve mirare a far si che il numero di SWAP da inserire durante l'esecuzione di un dato algoritmo sia il minore possibile.
Qiskit mette a disposizione diversi algoritmi utili a trovare la combinazione necessaria di SWAP utile ad eseguire correttamente un dato circuito quantistico [22].
Ad oggi gli approcci utilizzati per eseguire il mapping dei qubit sono di due tipi:

- Formulare il problema del mapping dei qubit come un problema di ottimizzazione equivalente ed utilizzare un *software solver*[2]. Questo tipo di approccio però fallisce quando si lavora con un numero di qubit elevato. Siccome la computazione quantistica sta entrando in quella che viene definita *Noisy Intermediate-Scale Quantum* (NISQ) *Era*, dove si avrà la possibilità di lavorare con computer quantistici con diverse decine, se non centinaia, di qubit questo approccio di mapping è destinato ad essere superato. Va infine sottolineato che il numero di qubit promessi nella NISQ Era è comunque troppo piccolo per pensare di introdurre dell' *Error Correction Code* (ECC) all'interno degli algoritmi quantistici. Questo fa si che il problema di un mapping ottimale risulterà negli anni a venire una sfida tecnologica pressante.

- Il secondo approccio, più moderno, è di tipo euristico. Questo tipo di approccio consiste nel definire una funzione di costo da minimizzare. Il problema di un approccio del genere è invece la generalità: una determinata funzione di costo potrebbe garantire un *success rate* molto alto per un determinato tipo di circuito o di processore, mentre potrebbe funzionare male per altri tipi di circuiti o processori.

Entrambi questi approcci tengono in genere conto dei dati di calibrazione che sono forniti dalla casa madre del processore, quali gli error rate per ogni arco sulla coupling maps o gli error rate per le operazioni di misura su singolo qubit. IBM fa una calibrazione giornaliera dei propri dispositivi fornendo i dati ottenuti agli utenti interessati.

Ottimizzazione dei Gates a Singolo e a MultiQubit

La decomposizione dei circuiti quantistici nel set di gates di base di un dispositivo IBM Q, e l'aggiunta di gates di SWAP richiesta per rispettare la topologia dell'hardware, fa aumentare la profondità e il numero di gates del circuito quantistico.
Fortunatamente ci sono diverse routine per ottimizzare i circuiti mediante la combinazione o l'eliminazione dei gates. In alcuni casi questi metodi sono talmente effettivi che il circuito di output ha una profondità minore di quello di input. In altri casi, non c'è molta possibilità di ottimizzazione, e la computazione risulta difficile per l'aumentare del rumore.
Il transpiler di qiskit mette a disposizione diversi livelli di ottimizzazione:

- *optimization_level = 0*: questo livello consiste nel solo mapping del circuito sul backend hardware, senza un esplicita ottimizzazione.

- *optimization_level = 1*: C'è il mapping del circuito ed anche un leggero livello di ottimizzazione facendo collassare gates adiacenti (ad esempio se si hanno due rotazioni successive attorno ad uno stesso asse, si trasforma il tutto in un unica rotazione, di un angolo che è la somma dei due)

[2]Questo tipo di programmi esegue risoluzione di problemi di ottimizzazione sotto vincoli. Un esempio è la soluzione di *sudoku*.

- *optimization_level = 2*: livello di media ottimizzazione, è possibile scegliere un layout che tenga conto anche del rate di errori del dispositivo hardware ed è presente una cancellazione
- *optimization_level = 3*: alto livello di ottimizzazione, con tutti gli steps precedentemente descritti, e una resintesi di gates a due qubit del circuito

1.2.5 Attuali Problematiche nel Processo di Compilazione

Il processo di compilazione quantistica attuale è stato ed è oggetto di intensi studi da parte della comunità scientifica. Il motivo sta nel fatto che oggi il compilatore quantistico è il mezzo più efficace che si ha a dispozione per abbattere la probabilità di commettere errori durante l'esecuzione di un algoritmo quantistico su i processori attualmente utilizzabili. Ad oggi la principale fonte di errore proviene dall'inserimento dei gate di SWAP all'interno dei circuiti per adattare questi alla topologia del processore in utilizzo. La ricerca di un mapping iniziale ottimale, che richiede quindi il minor numero di inserimenti di SWAP possibile risulta fondamentale. Gli approcci utilizzati oggi per trovare un mapping ottimale, visti nella sezione 1.2.4, sono soggetti a diverse problematiche. L'approccio basato su ottimizzatori SMT solver è un approccio non utilizzabile al crescere del numero di qubit poichè i tempi di compilazione richiesti aumentano esponenzialmente all'aumentare del numero di qubit, mentre oggi è fortemente utilizzato l'approccio basato sulla minimizzazione di funzioni di costo costruite ad hoc per il problema. Questi algoritmi però sono molteplici e nessuno di questi assicura performances migliori degli altri. Una possibile via da seguire allora consiste nel testare tutti gli algoritmi di mapping iniziale forniti da Qiskit ed eseguire un dato algortimo sul dato processore più volte per i diversi tipi di algoritmi di mapping, valutando cosi quale per lo specifico caso garantisca le performances migliori. Questo, dati i tempi di attese nelle code del cloud di IBM Quantum Experience, risulta essere un approccio difficilmente perseguibile. Da qui nasce l'idea di sviluppare una rete neurale che permetta di trovare in maniera automatica un layout iniziale per un dato circuito su di un dato processore più vicino all'ottimale possibile.

Nel corso dei prossimi capitoli si entrerà nel dettaglio di questa idea, prima analizzando quali sono gli aspetti fondamentali dell'apprendimento automatico esaminando nel dettaglio il funzionamento di una rete neurale, finendo poi per descrivere il modello costruito illustrando, infine, i risultati ottenuti.

Capitolo 2

Elementi di Machine Learning

In questo capitolo verrà definito il concetto di Machine Learning descrivendone le caratteristiche principali. Verrà quindi analizzato il workflow tipico di un algoritmo di machine learning e particolare attenzione sarà riservata, nella parte finale del capitolo, alla descrizione di Reti Neurali Profonde (*Deep Neural Network*), utilizzate poi per eseguire l'operazione di mapping di qubit virtuali sulla topologia fisica dell'hardware a disposizione, rispettando i vincoli descritti nel capitolo precedente.

2.1 Definizione di Apprendimento per una Macchina

Il machine learning è una branca della computer science che studia i sistemi e gli algoritmi che possono imparare dai dati, sintetizzando da essi nuova conoscenza. Questa branca è fondamentale nello studio e nello sviluppo delle intelligenze artificiali: un sistema basato sull'apprendimento automatico può migliorare la propria conoscenza del sistema da studiare dall'osservazione dei dati di input per poi fornire output più vicini a quelli desiderati.
In sintesi, quindi, un algoritmo di machine learning è un algoritmo capace di apprendere un dato modello a partire dai dati a disposizione senza alcuna conoscenza preliminare a riguardo.
E' allora opportuno formalizzare il concetto di *apprendimento* per una macchina. In letteratura è presente quest'efficace definizione di apprendimento [24]:

Un programma apprende da una certa esperienza E se nel rispetto di una classe di compiti T, con una misura della prestazione P, la prestazione P misurata nello svolgere il compito T è migliorata dall'esperienza E.

I compiti del machine learning sono spesso descritti in termini di come il sistema possa trattare un esempio, una collezione di features, o caratteristiche, quantitativamente misurate da alcuni oggetti o eventi che vogliamo il sistema elabori. Tipicamente l'input viene rappresentato da un vettore $x \in \mathcal{R}^n$, dove ogni x_i rappresenta una feature che il sistema può utilizzare per apprendere o prevedere.

Per valutare la qualità di un algoritmo di machine learning va stimata una misura quantitativa delle performances P dell'algoritmo. Spesso la misura di P è specifica per un certo compito T che deve compiere il sistema.
Complessivamente si possono individuare due macro categorie di algoritmi e dunque di compiti in cui il machine learning viene applicato:

- *Supervised Learning*: Gli algoritmi supervisionati dal tipo di esperienza svolgono compiti come la regressione o la classificazione. Quest'ultimo compito verrà largamente trattato nel corso del lavoro di tesi, dove una particolare codifica di classificazione permetterà ad una rete neurale di eseguire l'operazione di mapping di circuiti su processori quantistici.

- *Unspervised Learning*: gli algoritmi non supervisionati dal tipo di esperienza svolgono compiti come il Clustering, ovvero da dati non labellati si cerca di trovare pattern o insiemi di punti che hanno caratteristiche comuni.

Nel corso del capitolo si focalizzerà l'attenzione sul supervised learning, ed in particolare sul lavoro di classificazione. In generale l'obiettivo di un algoritmo di classificazione è quello di prevedere le classi di appartenenza di dati che sono del tutto ignoti all'algoritmo. Quest'obiettivo viene perseguito suddividendo il dataset a disposizione in un dataset di *training* ed un dataset di *test*: dal primo l'algoritmo imparerà ad effettuare il compito di classificazione per cui è stato progettato lavorando con istanze che sono labellate, ovvero l'algoritmo conosce a quali classi appartengono le istanze; dal secondo invece si otterrà una misura delle perfomances P dell'algoritmo, in quanto questo effettuerà una predizione della classe di ogni istanza e queste verranno confrontate con la classe reale, ignota all'algoritmo ma a disposizione nel dataset, definendo la cosiddetta *accuracy* della predizione (numero di predizioni esatte sul numero di predizioni complessive).

$$Accuracy = \frac{\#predizioni \quad esatte}{\#predizioni \quad totali} \tag{2.1}$$

2.2 Il formalismo generale in un problema di Machine Learning

In un generico problema di Machine Learning, tra cui un problema di classificazione come quello affrontato nel lavoro di tesi, si ha a disposizione un dataset $\mathcal{D}(\mathbf{X}, \mathbf{y})$ dove $\mathbf{X}$ è una matrice di variabili indipendenti detta *confusion matrix* (se si fanno n osservazioni della variabile $\mathbf{x} \in \mathcal{R}^p$ allora la confusion matrix $\mathbf{X}$ apparterrà ad $\mathcal{R}^{n \times p}$) ed $\mathbf{y}$ è un vettore di variabili dipendenti. Le variabili indipendenti $\mathbf{x}$ e le variabili dipendenti $\mathbf{y}$ sono collegate tramite un modello $f(\mathbf{x}|\mathbf{w})$, che rappresenta la funzione che vogliamo utilizzare per predire il valore di output in corrispondenza di un nuovo dato in input (*funzione di soglia o di attivazione*). Viene poi definita una funzione di costo $\mathcal{C}(\mathbf{y}, f(\mathbf{x}|\mathbf{w}))$ in modo che si possano giudicare le performances del modello. Il modello viene infine determinato minimizzando la funzione di costo rispetto ai parametri $\mathbf{w}$.
L'idea è quella di partizionare il dataset a disposizione in dataset di *training* utile ad imparare il modello, ed in un dataset di *test* su cui andare a verificare

le capacità predittive del modello appreso.

E' opportuno citare, poichè ritorneranno molto utili in seguito, alcuni esempi di funzioni $f(\mathbf{x}|\theta)$ comuni. Nel caso di classificazione binaria, ovvero nel caso in cui il dataset $\mathcal{D}(\mathbf{X}, \mathbf{y})$ è composto da sole due classi ($y_i \in [0, 1] \forall i$), è molto comune utilizzare come funzione di soglia la cosiddetta *logistic sigmoid function* (o sigmoide), dove definita con

$$z = \sum_{i=0}^{p} w_i x_i = \mathbf{w}^T \mathbf{x} \tag{2.2}$$

la combinazione lineare delle features di una istanza con i pesi del modello, allora la funzione sigmoide è definita dalla

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

Questa coincide con la probabilità condizionata che l'istanza $\mathbf{x}$ appartenga alla classe labellata con 1, $p(y = 1|\mathbf{x})$.
Dunque definiti i pesi l'attribuzione della classe viene fatta ponendo come valore di soglia $\phi(z)_{tr} = 0.5$, e se il valore della funzione calcolata è maggiore di $\phi(z)_{tr}$ allora verrà attribuita a quell'istanza la classe 1, altrimenti la classe 0.

La naturale generalizzazione a problemi multi-classe, in cui le classi disponibili non sono più solo due ma sono M si ottiene rimpiazzando la funzione di soglia sigmoidale con la *softmax function*. Questa fornisce la probabilità che l'istanza in questione appartenga ad ogni classe, dunque la somma delle probabilità è ovviamente normalizzata ad uno. Di conseguenza l'assegnazione della classe all'istanza viene fatta prendendo la probabilità massima tra quelle calcolata secondo l'espressione

$$p(y = j|z) = \frac{e^{z_j}}{\sum_{k=0}^{M} e^{z_k}} \tag{2.4}$$

Per quanto riguarda la funzione di costo utilizzata per stimare i pesi del modello la scelta è molto varia e dipende fortemente dal tipo di classificazione che occorre ottenere.

Molto comune è l'uso della *Cross-Entropy Loss* definita come

$$CE = -\sum_{i}^{C} t_i log(\phi(z_i)) \tag{2.5}$$

dove t_i è label reale della classe i-esima di C.

In una classificazione binaria, dove C = 2, la Cross Entropy si può scrivere come riportato nell'equazione (2.6), dove la funzione di attivazione è quella

sigmoidale.

$$CE = -\sum_{i=1}^{C=2} t_i log(\phi(z)_i) = -t_1 log(s_1) - (1 - t_1) log(1 - s_1) \tag{2.6}$$

Nel caso di classificazione multi-classe, come quella eseguita nel corso del presente elaborato, viene utilizzata l'espressione della cross-entropy ma si utilizza la predizione tramite softmax per calcolarla, piuttosto che quella sigmoidale.

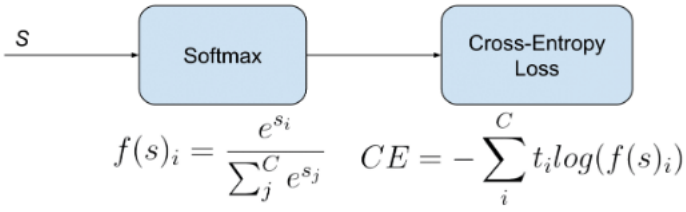

Figura 2.1: Activation function e Loss Function per classificazioni multi-classe

Molto spesso inoltre, nei problemi di classificazione multi-classe i label delle istanze vengono trasformati mediante *one-hot encoder*: per far si che tutte le classi vengano viste dall'algoritmo di classificazione alla stessa maniera si trasforma ogni label in un vettore di dimensione pari al numero di classi presenti nel dataset, si pongono a zero tutte le componenti diverse dalla classe propria dell'istanza in considerazione, il quale elemento nel vettore viene settato ad 1.
Se si suppone di lavorare con un dataset in cui sono presenti tre classi, allora la trasformazione dei label sarà la seguente

$$0 \rightarrow 1\ 0\ 0$$
$$1 \rightarrow 0\ 1\ 0$$
$$2 \rightarrow 0\ 0\ 1$$

Quando ci si trova con labels di tipo one-hot è possibile utilizzare come funzione di costo la *Categorical Cross Entropy*: siccome c'è un solo elemento del vettore di target t che è non zero $t_i = t_p$, si possono trascurare tutti gli elementi che hanno 0 come label nella somma che definisce la Cross Entropy e definire come funzione di costo

$$CE = -log(\phi(z)_p) = -log(\frac{e^{z_p}}{\sum_j^C e^{s_j}}) \tag{2.7}$$

Dove z_p è la combinazione lineare di features e pesi per la classe positiva.

2.3 Le Reti Neurali Artificiali

Capito come si definisce il concetto di apprendimento automatico per una macchina e definito il formalismo generale di un problema di machine learning, ci si concentrerà adesso sulla descrizione del particolare modello di apprendimento utilizzato nel corso del lavoro di tesi, quello di rete neurale.

Per capire a fondo il funzionamento di una rete neurale artificiale è utile analizzare brevemente il funzionamento di una rete neurale biologica. D'altronde l'idea di sviluppare questo tipo di architetture è storicamente presa in seguito all'analisi delle reti di neuroni presenti nei nostri cervelli.
Da un punto di vista biologico, i neuroni sono delle celle elettricamente attive ed il cervello umano ne contiene circa 10^{11}. La generica forma di un neurone è rappresentata in Figura 2.2: i dendriti rappresentano gli ingressi del neurone, mentre gli assioni ne rappresentano l'uscita.

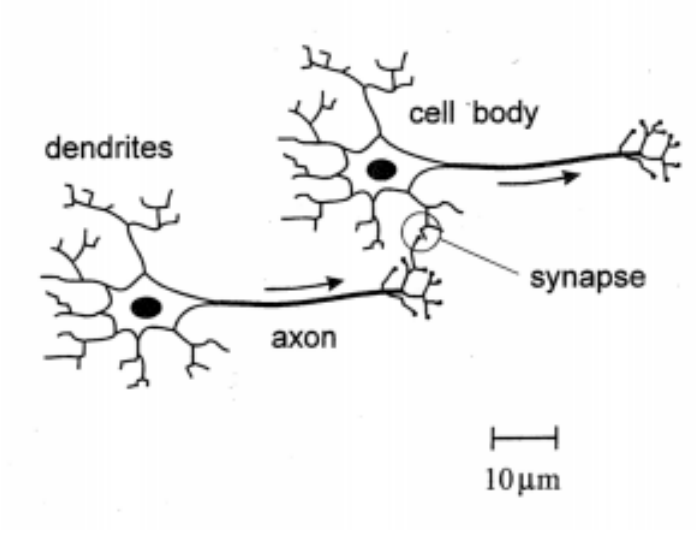

Figura 2.2: Schematizzazione di due neuroni biologici

La comunicazione tra i neuroni avviene tramite giunzioni chiamate sinapsi. Ogni neurone si può trovare principalmente in due stati: attivo o a riposo. Quando il neurone è attivo esso produce un potenziale di azione che viene trasportato lungo l'assione. Questo segnale viene poi fatto arrivare ai neuroni circostanti mediante le sinapsi. A seconda del segnale che arriva agli altri neuroni, questi possono a loro volta attivarsi o rimanere a riposo.
Riprendendo questo meccanismo, nel 1943, McCulloch e Pitts [25] proposero il loro modello di neurone artificiale (Figura 2.3): una serie di informazioni in ingresso producono l'attivazione del neurone se viene superata una certa soglia della funzione di attivazione scelta.

Si è visto nella sezione precedente del lavoro di tesi, quali possono essere gli scopi di tale neurone, quali sono possibili funzioni di attivazione mentre altre saranno incontrate durante il proseguo dell'elaborato.

Una rete neurale artificiale non è altro che l'insieme di più neuroni artificiali tra loro connessi.
Nel corso dei questo capitolo ci si concentrerà nella descrizione di Reti Neurali *Feedforward*: con questo termine si sta ad indicare quelle reti in cui l'informazione all'interno della rete viaggia in una sola direzione, senza loop, dallo strato di ingresso allo strato di uscita. Esistono, infatti, altri tipi di Neural Network come le *Recursive Neural Network* in cui i neuroni di output sono nuovamente

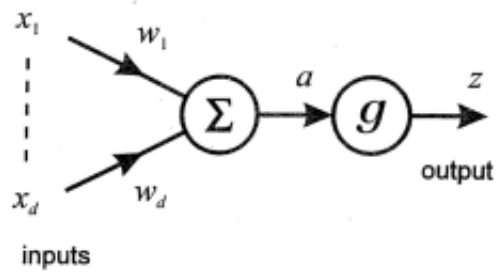

Figura 2.3: Modello di McCulloch-Pitts

collegati a quelli di input, ma la descrizione di questi modelli esula dagli scopi del presente elaborato.

Il più semplice modello di rete neurale è quello rappresentato in Figura 2.4: il *Multilayer Perceptron*, ovvero una rete di tre strati di neuroni fully-connected, nel senso che ogni neurone di uno strato è collegato con tutti i neuroni dello strato successivo.
Si indicherà con *unità*, il numero di neuroni facente parti di un determinato strato della rete. C'è quindi un primo layer che è detto *input layer* il cui numero di unità è in genere uguale al numero di features di input del dataset su cui si vuole far lavorare la rete neurale. L'ultimo strato è detto *output layer* e tipicamente il suo numero di unità è uguale al numero di classi che si vuole andare a classificare. Scegliendo ad esempio come funzione di attivazione per il layer di output la softmax analizzata nella sezione precedente, si avrà in uscita dalla rete neurale un vettore di n probabilità, dove n è il numero di unità che compongono lo strato. Ogni probabilità indica la probabilità che ognuno di questi neuroni di output risulti attivo, ovvero che l'istanza appartenga alla classe che il rispettivo neurone codifica. Se ad esempio si ha uno strato di output composto da due neuroni, l'output della rete sarà un vettore del tipo [p0, p1] dove p0 è la probabilità che l'istanza in questione appartenga alla classe 0 e p1 che appartenga alla classe 1. Si attribuirà come classe di quell'istanza quella a probabilità maggiore.

Infine tutti gli strati che non sono ne di input e ne di output sono detti *hidden layer* e il numero di unità che li compone è un iperparametro del modello.
Altro iperparametro è il numero di hidden layer che compongono la rete utilizzata: per modelli che hanno più di un solo hidden layer si parla di Reti Neuarli Profonde(***Deep Neural Network***).
Va inoltre specificato che molto spesso ogni neurone ha un ingresso aggiuntivo detto *bias* che serve a regolare la soglia oltre la quale il neurone risulterà attivo.

2.4 Il Training

Come introdotto nella sezione 2.2, la scelta dei pesi nel modello $f(\mathbf{x}|w)$ che esegue l'operazione di classificazione è fatta nella fase di training, minimizzando

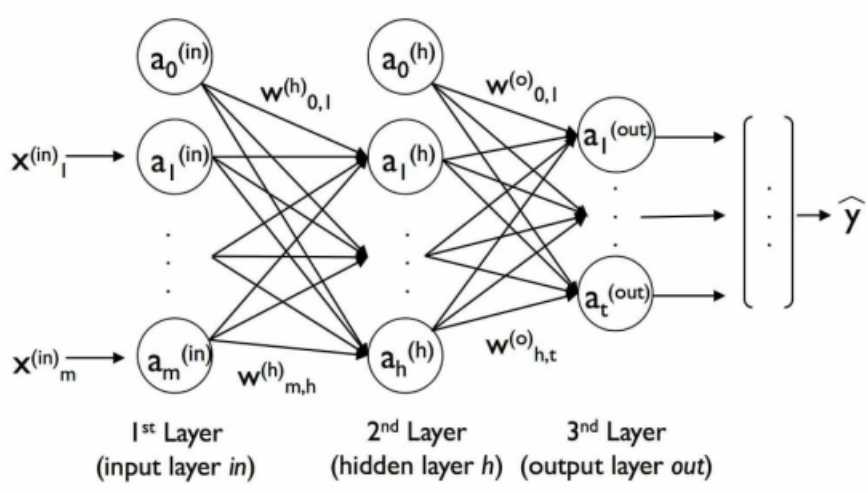

Figura 2.4: Schematizzazione di una Rete artificiale fully-connected Multilayer a tre strati

una funzione di costo, che sarà indicata nel corso del paragrafo con E(w)
Fin'ora però non si è accennato a come questa procedura di minimizzazione viene eseguita, sia nel caso di singolo neurone, sia nel caso più complesso di una rete neurale. Lo scopo di questa sezione è proprio quello di portare il lettore alla conoscenza degli algoritmi utilizzati per compiere il processo di minimizzazione. Il più semplice algoritmo di minimizzazione che può essere utilizzato in queste situazioni è il ***Gradient Descent***: esso consiste in una procedura di minimizzazione che fissato un valore iniziale dei pesi, tende a muoversi lungo la direzione opposta alla direzione del gradiente, il quale per definizione, punta nella direzione in cui la funzione di costo cresce.

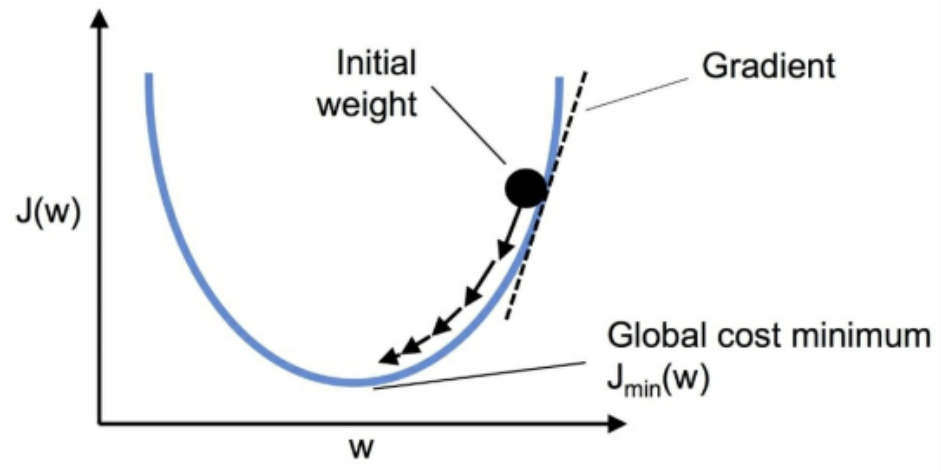

Figura 2.5: Rappresentazione grafica del metodo della discesa del gradiente [26]

L'algoritmo della discesa del gradiente è quindi riassumibile nei seguenti steps:

- Fissare un punto iniziale
$$w_0$$
- Definire il passo
$$\nu_t = \eta_t \nabla_w E(w_t)$$
- Passare al punto successivo secondo la regola
$$w_{t+1} = w_t - \nu_t = w_t - \eta_t \nabla_w E(w_t)$$

Il parametro η_t è detto *learning rate* e definisce il passo (uniforme in tutte le direzioni del gradiente) ad ogni iterazione. Questo è un altro iperaparametro su cui andrà quindi effettuato un *tuning*, per trovare il valore che garantisce le migliori performance dell'algoritmo in relazione al tipo di problema che si sta affrontando.
Intuitivamente, se il learning rate è molto piccolo ci si muoverà nelle direzioni in cui la funzione diminuisce molto lentamente, andando a convergenza ma probabilmente con un numero elevato di iterazioni. Se invece il learning rate è troppo grande, si corre il rischio di evitare dei minimi locali nel processo di esplorazione.
Molto spesso si preferisce lavorare con una versione *stocastica* del Gradient Discent: si preferisce, cioè, divide il dataset complessivo di training, in un insieme di minibatch, costituiti ciascuno da un numero molto minore di istanze, e di utilizzare il metodo della discesa del gradiente, come appena presentato, ma calcolando ad ogni iterazione il gradiente della funzione di costo utilizzando i dati una certa minibatch (in questo risiede la stocasticità del metodo).
Il metodo dello *Stochastic Gradient Discent* riduce cosi la probabilità che l'algoritmo si blocchi all'interno di minimi locali della funzione di costo.

Una versione più robusta del metodo del Gradient Descent è dato dall'ottimizzatore ***ADAM***. Si rimanda alla referenza [27] per una descrizione approfondita dell'algoritmo, ma l'idea di base che c'è dietro l'ottimizzazione ADAM è quella per cui il passo fatto ad ogni iterazione viene aggiornato tenendo conto del valore medio del gradiente: se il gradiente è molto alto si è lontani dal minimo e dunque ha senso compiere un passo nell'iterazione successiva molto grande, al contrario del caso in cui il valore del gradiente è molto piccolo e presumibilmente si è vicino ad un minimo.

2.4.1 La Stima dei Parametri via Backpropagation

Ricapitolando, si sono per ora presentati alcune metodologie di approccio che permetto di stimare i parametri di un certo modello di apprendimento automatico trovando il valore minimo di una funzione di costo E[y, f(x;**W**)]. Formalmente quindi lo scopo del training di un dato algoritmo è quello di risolvere l'equazione

$$\hat{\mathbf{W}} = argmin_{\mathbf{W}} \left\{ \frac{1}{n} \sum_{i=1}^{n} E[y_i, f(x_i; \mathbf{W})] \right\} \tag{2.8}$$

Il problema delle reti neurali è che il numero di parametri con cui ci si trova a lavorare è enorme. Anche per modelli semplici, si parla di migliaia o decine di migliaia di parametri da stimare.
Bisogna infatti ricordare che ogni arco di una rete neurale è rappresentato da un peso, quindi se si ha una rete con L strati, la rete possiede L matrici di parametri W^l dove detti rispettivamente p_l e p_{l+1} il numero di unità componenti lo strato l-esimo e l+1-esimo, W^l contiene $p_l \times p_{l+1}$ coefficienti.
Ad ogni iterazione dell'algoritmo di minimizzazione della funzione di costo, il calcolo del gradiente di ordine primo richiede un numero di operazioni pari al numero di coefficienti, mentre le operazioni richieste per il calcolo di un eventuale gradiente secondo crescono in maniera quadratica all'aumentare del numero di parametri.
La forza dell'algoritmo di *backpropagation* sta proprio nel fatto che questo utilizza soltanto il gradiente di ordine primo per arrivare ad ottenere una soluzione dell'equazione 2.8. Si rimanda alla referenza [26] per una descrizione algebrica dell'algoritmo, che risulta essere non di breve durata. Si preferisce quindi, enunciare per grandi linee quali sono le caratteristiche fondamentali di questo importante algoritmo.
Esso alterna due passi in modo iterativo: con il *passo in avanti*(dallo strato di input allo strato di output) si ottiene il valore di $f(x_i; \mathbf{W})$ per un dato valore dei pesi $\mathbf{W}$, mentre con un *passo all'indietro* (dallo strato di output a quello di input) si ottengono i gradienti e vengono aggiornati i parametri mediante tecniche come quella del Gradient Discent o Adam. Ogni iterazione dell'algoritmo viene chiamata *Epoca*.

2.5 Overfitting e Bias-Variance Tradeoff

Visto che cosa è una rete neurale, che cosa vuol dire allenare una rete neurale e come questa può effettuare delle predizioni, in questa sezione ci si soffermerà brevemente sul problema della complessità che un modello di apprendimento, in particolare una rete neurale, deve avere affinché le sue performances siano quelle ottimali.
La complessità del modello in generale è proporzionale al numero di paramentri che esso posside, dunque, per una rete neurale è direttamente collegata alla profondità della rete, intesa come numero di *hidden layers*, e al numero di unità di cui ognuno di questi strati è composto.
Si potrebbe erroneamente pensare che reti più profonde o reti più larghe, ovvero con molti neuroni per ogni strato, siano più performanti di reti più piccole: la realtà dei fatti invece è ben diversa da questa semplificazione errata.
In Figura 2.6 è presente un esempio di *overfitting*, dove si può vedere come nel caso di una regressione (ma il discorso è analogo per il caso di classificazione) il modello più complesso non sia quello più performante. La regressione polinomiale con un polinomio di grado maggiore, seppur sembra adattarsi meglio ai dati di training (Figura a sinistra), non è la migliore sul dataset di test. Ovvero, il modello più complesso non è quello che, in questo caso, riesce meglio a fare previsioni su dati sconosciuti. Bensì è un modello di media complessità ad ottenere capacità predittive maggiori, in questo specifico caso un polinomio di terzo

grado (in giallo) riesce a fittare meglio i punti della distribuzione di test rispetto ad un polinomio di decimo grado (in verde)

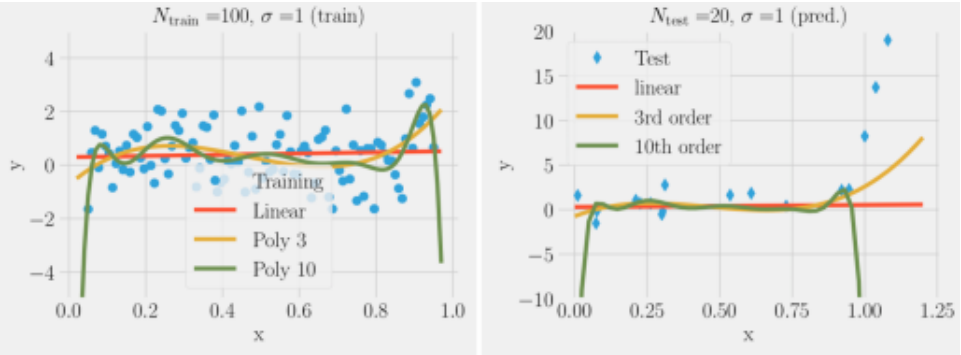

Figura 2.6: Esempio di *overfitting* per un problema di regressione: il modello polinomiale di decimo grado, seppur si vede essere quello che meglio si adatta ai dati di allenamento(Figura a sinistra), non ha le capacità predittive che ha il modello polinomiale di terzo grado (Figura a destra)

Questo breve esempio, permette di capire il significato dell'*overfitting*: modelli molto complessi possono talvolta essere troppo legati ai dati di training ma avere poi una bassa capacità predittiva.

Una formalizzazione maggiore di questo problema la si può ottenere guardando a quella che in letteratura è descritta come ***Bias-Variance Tradeoff***.
Si può definire *In-Sample-Error* il valore assunto dalla funzione di costo sul training set, mentre si può definire *Out-of-Sample-Error* il valore assunto dalla funzione di costo sul test set:

$$E_{in} = C(\mathbf{y}_{train}, \mathbf{f}(\mathbf{X}_{train}|\mathbf{w})) \qquad E_{out} = C(\mathbf{y}_{test}, \mathbf{f}(\mathbf{X}_{test}|\mathbf{w})) \tag{2.9}$$

Si definisce, *Bias* l'errore intrinseco del modello utilizzato per la predizione. Effettuando un numero infinito di osservazioni ci si aspetta di ridurre eventuali errori statistici a zero, di conseguenza il bias rappresenta il limite per la dimensione del dataset che tende all'infinito dell' Out-of-Sample-Error:

$$Bias = \lim_{N \to \infty} E_{out} \tag{2.10}$$

Si definisce, invece, *Varianza* l'errore dovuto alla presenza di un campione di osservazioni di dimensione finita (errore stocastico), questa quindi tende a zero quando la dimensione del Dataset tende a zero.
Complessivamente allora si può dire che

$$E_{out} \quad = \quad Bias \quad + \quad Varianza \tag{2.11}$$

E' facile analizzare l'andamento del bias-variance tradeoff mediante i seguenti grafici.
Nel grafico in Figura 2.7 è rappresentato l'andamento dell' in-sample error e dell'

out-of-sample error in funzione della dimensione del Dataset. Fissata la complessità del modello l'in-sample error aumenta all'aumentare del numero di punti, perchè il modello ha bisogno di maggiore complessità per imparare le fluttuazioni che questi aggiungono. Al contrario, però l'out of sample error diminuisce perchè diminuisce l'errore statistico e tende asintoticamente al bias.

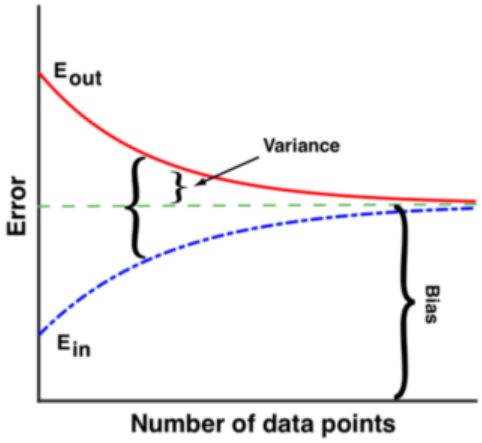

Figura 2.7: Grafico 1

Altro grafico da analizzare è quello in Figura 2.8 dove è rappresentato l'andamento dell' out-of-sample error in funzione della complessità del modello, fissata la dimensione del Dataset:

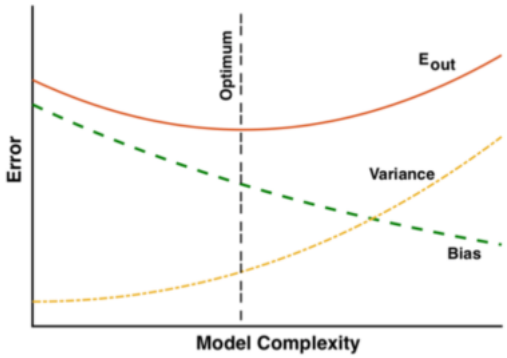

Figura 2.8: Grafico 2

un modello troppo semplice ha una bassa varianza, nel senso che anche con un campione di dimensione non eccessiva si riesce ad abbattere il rumore statistico (il modello non ha abbastanza parametri per seguire le fluttuazioni e vede solo l'andamento prevalente), mentre l'errore intrinseco, ovvero quello che si commetterebbe per N dimensione del Dataset tendente all'infinito è grande, perché il modello non è abbastanza complesso da adattarsi al modello reale.
Se invece si utilizza un modello estremamente complesso si ha che il bias risulta piccolo poiché asintoticamente il modello tende a quello reale, ma la varianza è

grande, perché è necessario un campione di dimensione infinita per abbattere il rumore statistico e 'costringere' il modello a non adattarsi alle fluttuazioni.

Si propone infine un terzo grafico, quello in Figura 2.9 esplicativo sulla questione del bias-variance tradeoff. I punti in verde rappresentano valori di parametri per un modello ad alta complessità, mentre quelli in nero i valori di parametri per un modello a bassa complessità, tutto ciò per tanti training set diversi di dimensione N fissata.
Le fluttuazioni del modello complesso intorno al modello reale sono grandi a causa della dimensione finita del campione, mentre il modello semplice ha piccole fluttuazioni ma non tende esattamente al modello reale, ovvero anche per N grandi il bias rimarrà grande.

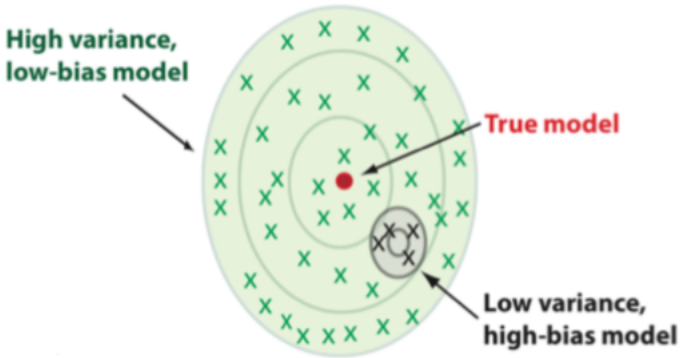

Figura 2.9: Grafico 3

Alla luce di quanto detto è giusto concludere che a seconda della dimensione del campione, è necessario scegliere in maniera opportuna tra i diversi modelli per migliorare l'accuratezza delle predizioni.

Gestire l'Overfitting nelle Reti Neurali

La forze delle Neural Network come metodo di apprendimento automatico sta nella capacità di queste di approssimare sostanzialmente qualsiasi modello dato il vastissimo numero di parametri che si ha a disposizione.
Per quanto appena visto, però, la possibilità di arrivare a descrivere modelli cosi complessi, rende le Reti Neurali molto prone al problema dell'overfitting.
In questa breve sezione si andranno ad analizzare due metodi che sono molto utilizzati proprio per fronteggiare questo problema. Ovviamente oltre i metodi che si è prossimi ad enunciare valgono le prescrizioni viste in precedenza: aumentare la dimensione del Dataset a disposizione è certamente la soluzione migliore, ma va da sè, che questa non è sempre una strada percorribile.
Per tali casi si enunciano due modalità di lavoro che permettono di limitare, quando possibile, il problema dell'overfitting lavorando sui parametri piuttosto che sul Dataset.

La prima possibile soluzione sta nella ***Regolarizzazione*** dei parametri. Con questa metodologia si intende il processo per cui viene aggiunto alla funzione di

costo un elemento extra, che sostanzialmente vincola lo spazio dei parametri in cui cercare la soluzione, limitando la complessità del modello. Due popolari versioni di questo metodo sono la *regolarizzazione L1* (Least Absolute Deviations - LAD) e la *regolarizzazione L2* (Least Square Errors - LS).
Da un punto di vista matematico queste tecniche di regolarizzazione consistono nell'aggiungere i fattori della 2.12 alla funzione di costo del modello

$$L2 : \lambda||\mathbf{w}||_2^2 = \lambda \sum_j w_j^2 \qquad L1 : \lambda||\mathbf{w}||_1 = \lambda \sum_j |w_j| \tag{2.12}$$

Il valore λ è un iperparametro che sostanzialmente regola il peso della regolarizzazione: maggiore sarà λ maggiore saranno gli effetti della regolarizzazione. Per capire a fondo cosa comporta aggiungere un termine di regolarizzazione al modello di apprendimento è possibile utilizzare una schematizzazione grafica nel caso semplice di soli due parametri (Figura 2.10). La norma L2 definisce in questo caso una sfera di raggio inversamente proporzionale a λ nello spazio dei parametri, dunque il valore minimo della funzione di costo che si può raggiungere con il vincolo della regolarizzazione è individuato dall'intersezione della sfera di regolarizzazione e la curva di livello più vicina al minimo della funzione di costo senza penalità. Questo si traduce nel limitare i valori che i parametri possono assumere nel modello.

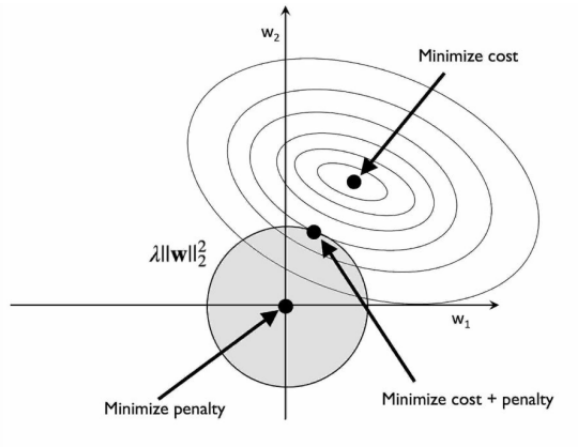

Figura 2.10: Rappresentazione grafica dell'effetto di una regolarizzazione L2 [26]

Alla stessa maniera, l'aggiunta di un termine di regolarizzazione L1, siccome questo è dato dalla somma di tutti i valori assoluti dei coefficienti e non è un termine quadratico come nel caso della norma L2, vincola lo spazio dei parametri non con una sfera ma con un rombo (Figura 2.11). In questo caso è molto probabile che la soluzione, vista come intersezione del rombo con le curve di livello della funzione di costo priva di regolarizzazione, sia molto vicina ad uno degli assi, facendo si che molti dei parametri vengano ad annullarsi. Di conseguenza l'aggiunta di un termine di regolarizzazione L1 fa si che la matrice di parametri diventi una matrice sparsa.

Soluzione diversa è invece rappresentata dai ***layers di Dropout***. L'idea di base di uno strato di Dropout è molto semplice: inibire ogni neurone dello strato precedente a quello di Dropout secondo una probabilità p, che diventa un

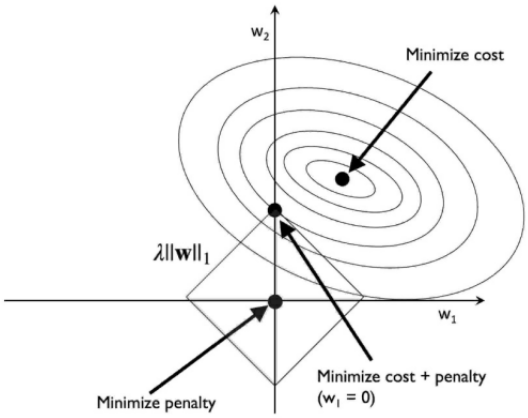

Figura 2.11: Rappresentazione grafica dell'effetto di una regolarizzazione L1 [26]

iperparametro del modello. Dunque ad ogni iterazione saranno randomicamente selezionati neuroni che verranno eliminati momentaneamente dal modello.
Un metodo come questo, potrebbe a prima vista sembrare controproducente, ma l'idea che c'è alla base è quella per cui ogni neurone prende in input le informazioni che gli vengono passate dai neuroni dello strato precedente, se randomicamente alcuni di questi vengono eliminati, i neuroni successivi impareranno a non dar peso maggiori ad alcuni neuroni piuttosto che ad altri.

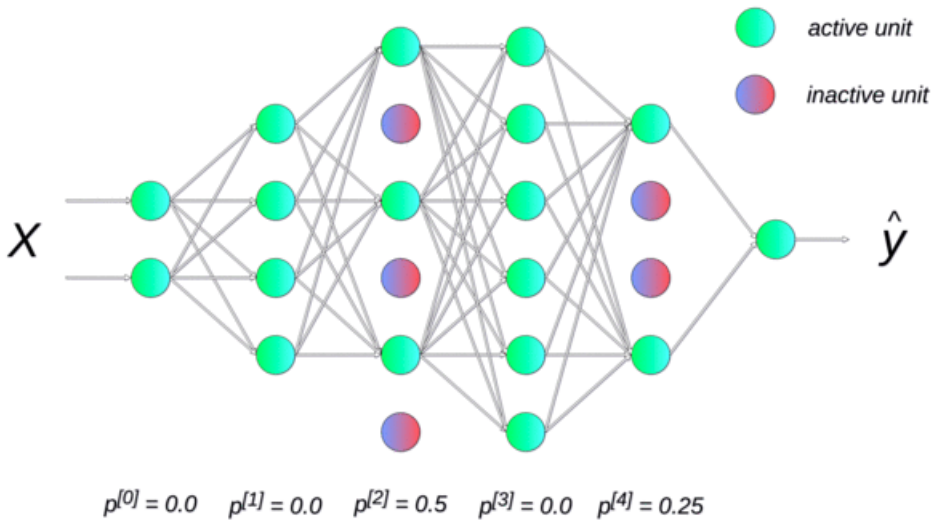

Figura 2.12: Rappresentazione grafica del Dropout. Alla referenza [28] è presente una animazione completa che meglio illustra la funzionalità di tale metodo

Illustrati i diversi accorgimenti da fare per limitare il problema dell'overfitting in una rete neurale, il cui funzionamento è stato analizzato partendo dal paragone con una rete neurale biologica, nel successivo capitolo si andrà ad illustrare il modello di rete neurale costruito per eseguire l'operazione di mapping dei circuiti sui processori quantistici.

Capitolo 3

Reti Neurali per il Mapping dei Qubit

In questo capitolo verrà descritta la struttura della rete neurale costruita per eseguire l'operazione di mapping dei qubit virtuali di un dato circuito su un processore quantistico a 5 qubit. La prima parte del capitolo sarà incentrata sulla composizione della rete, analizzando i diversi layer di cui questa è costituita. La parte finale sarà invece utile a comprendere i due approcci utilizzati nel corso del lavoro di tesi per verificare che la rete rispettasse i vincoli logici imposti dal problema: due qubit virtuali diversi devono essere mappati in due slot fisiche diverse.
Come nel resto dell'elaborato la parte di codice utilizzata per costruire il modello è soltanto in minima parte mostrata. Si rimanda il lettore interessato alla cartella Github *Github Repository Link*

3.1 Una Rete Neurale per il Mapping dei Qubit: Setup del Modello

L'idea che c'è dietro l'utilizzo di una Neural Network per l'operazione di mapping è quella di avere un numero di layers di output pari al numero di qubit fisici che compongono il processore da utilizzare. Ognuno di questi darà come output l'indice del qubit virtuale che deve essere mappato nel corrispondente slot fisico. Nel caso in cui ci fosse un circuito quantistico con un numero di qubit minore del numero di qubit fisici a disposizione con il processore in uso, allora i layers di output corrispondenti a slot in cui non va mappato nessun qubit daranno in uscita un valore prefissato: ad esempio utilizzando un processore a 5 qubit, un algoritmo quantistico può utilizzare al massimo 5 qubit, nel caso l'algoritmo ne preveda soltanto 3, due dei layers di output dovranno dare in uscita un valore di riferimento scelto essere 5 (i qubit sono indicizzati da 0 a 4).

3.1.1 Il Modello Costruito

Nel corso del lavoro si è utilizzata la libreria *Keras* [29]. Keras è una libreria open source per l'apprendimento automatico e le reti neurali, scritta in Python. Questa è progettata come un'interfaccia a un livello di astrazione superiore di

altre librerie simili di più basso livello, e supporta come back-end tra le varie librerie, quella *TensorFlow* sviluppata da Google.
Il vantaggio di lavorare con Keras è l'alta facilità che si ha nel creare modelli personalizzati ed ad hoc per il problema da affrontare.
Lavorando con il processore IBM Burlington a 5 qubit la rete sviluppata ha cinque layers di output e la sua struttura completa è presentata nella figura sottostante (Figura 3.1)

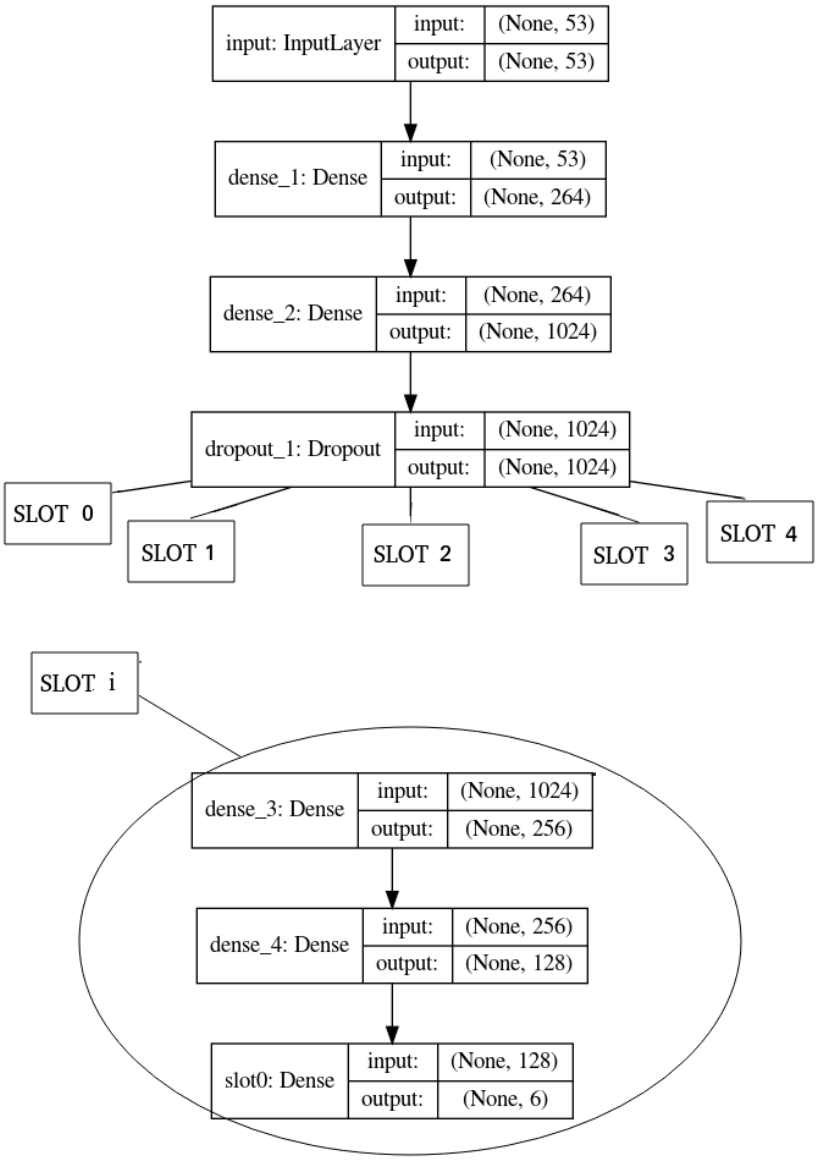

Figura 3.1: Rappresentazione grafica della Rete Neurale costruita

Lo schema illustrato in Figura 3.1 permette di capire nel dettaglio il modello messo a punto. Per ragioni dovute alla dimensione dell'immagine costruita da Keras si è raggruppato nei blocchi chiamati *'Slot'* l'ultima catena di layer che porta agli strati di output. La struttura di tutte le Slot è uguale ed è quella presentata nello zoom a fine immagine.
A sinistra di ogni blocco è presente il nome del layer dato all'interno del codice seguito dal tipo di layer che il blocco rappresenta. Compaiono in tutto il modello soltanto tre tipi di strati, l' *InputLayer* ovvero lo strato di input della rete, gli strati *Dense* che sono gli strati standard di una rete neurale semplice ed infine lo strato *Dropout* utile a limitare il problema dell'overfitting come già analizzato nella sezione 2.5. Gli strati Dense sono strati di unità fully-connected con le unità degli strati successivi.
Per capire a fondo, invece, l'ultima colonna di ogni blocco presente nel grafico del modello, va sottolineato che TensorFlow passa l'informazione tra i successivi strati della rete sottoforma di tensori fatti da un numero di righe pari al numero di istanze del dataset ed ad un numero di colonne pari al numero di unità di ogni strato. Di conseguenza nel passaggio dallo strato *dense_1* a quello dense_2, un tensore con 264 colonne sta entrando come input ad uno strato composto da 1024 unità che darà quindi in uscita un tensore con 1024 colonne.
Come si può notare dalla figura gli strati di output che si trovano al termine di ognuno dei blocchi *Slot* hanno tensori con 6 colonne in uscita, ciò è dovuto al fatto che le unità che costituiscono gli strati di output sono sei. Ovvero ogni output layer ha 6 neuroni e si accenderà quello relativo all'indice del qubit che deve essere mappato nella slot che quel layer rappresenta.

Con il fine di analizzare alcune caratteristiche del modello costruito si riporta di seguito la porzione di codice relativa all'implementazione della rete. Siccome la struttura delle slot di output è la medesima per ognuna delle cinque presenti in Figura 3.1, si è scelto di riportare a titolo di esempio il codice relativo all'implementazione della sola slot0

```
# Codice Rete Neurale

A0 = Input(shape=(X_train.shape[1], ), name='input')
A1 = Dense(264, activation='relu')(A0)
A2 = Dense(1024, activation='relu',
       kernel_regularizer=keras.regularizers.l1(0.001))(A1)
drop = Dropout(0.65)(A2)

#Slot 0
A3_0 = Dense(256, activation='relu')(drop)
A4_0 = Dense(128, activation='relu')(A3_0)
slot0 = Dense(units=6, activation='softmax', name='slot0')(A4_0)
```

Ogni riga del codice corrisponde ad uno strato della rete. E' allora utile soffermarsi a descrivere le caratteristiche principale del modello.
Tutti gli hidden layers (ovvero gli strati che non sono quelli di output) hanno come *activation function* una funzione fin'ora non trattata, la *ReLu* - Rectified Linear activation function. Questa è una funzione definita come la parte positiva

del suo argomento

$$f(x) = x^+ = max(0, x) \tag{3.1}$$

in altri termini essa è sempre 0 se l'argomento della funzione è negativo; assume invece un valore diverso da 0 se è positivo. La funzione quindi è lineare (derivata costante) per valori maggiori di zero e ciò la rende preferibile a molte altre in strutture di Deep Learning in quanto la Relu rende agevole il calcolo dei pesi mediante backpropagation. Questa caratteristica permette alla ReLu di essere una funzione di attivazione largamente utilizzata in strutture di reti neurale profonde, al contrario di funzioni non lineari come la sigmoide, in cui la difficoltà computazionale nel calcolo del gradiente risulta elevata [30].
Per quel che riguarda lo strato di Dropout il numero in parentesi indica la percentuale di unità che rimangono randomicamente spente nello strato precedente. Si noti come la presenza di uno strato di dropout è molto importante in una rete neurale larga come quella nel modello presentata. Modelli con tale numero di unità fanno si che il numero di parametri a disposizione sia vastissimo, e che quindi il problema dell'overfitting risulti pressante.
A contrastare ancor di più l'overfitting del modello nello strato più largo A3 è stato inserito un regolarizzatore L1, il cui numero in parentesi tonda (0.005) indica il valore del parametro λ presentato nella sezione 2.5.
In ultima analisi ci si può soffermare sui layer di output, nominati all'interno del codice *slot_i*, dove i è riferito all'indice del qubit fisico all'interno del processore IBM_Q Burlington che tale slot rappresenta. La funzione di attivazione scelta per gli strati di output è la *softmax activation function* presentata nella sezione 2.2. Ogni strato di output quindi, darà 6 probabilità che devono essere interpretate come la probabilità che il qubit virtuale i-esimo debba essere mappato nel qubit fisico che tale slot rappresenta (dove i va da 0 a 5, e la probabilità posta come elemento 5 del vettore è la probabilità che tale slot debba essere lasciata vuota, cosa che capita con circuiti più piccoli del numero massimo di qubit utilizzabile da un dato processore).

Come si è detto nel capitolo precedente la peculiarità delle reti neurali profonde come quella appena descritta è quella di lavorare con un numero enorme di parametri. E' interessante allora sottolineare il numero di parametri da cui è composto il modello di apprendimento che la rete andrà ad elaborare nella sua fase di training.
Keras fornisce una funzione che ritorna esattamente il numero di parametri presenti nel modello costruito e tale numero in questo caso è 1,765,966. Ciò è indicativo di quanto un modello di intelligenza artificiale possa eseguire compiti che sono per l'uomo ineseguibili.
Il prezzo da pagare per lavorare con reti cosi profonde è però quello di aver bisogno di dataset sufficientemente grandi cosi come si è visto nei grafici del *bias variance tradoff*.

3.2 Analisi a Posteriori delle Predizioni

L'output di ognuna delle slot presenti in Figura 3.1 è un vettore di 6 elementi, indicizzati da 0 a 5 che rappresenta la probabilità che il qubit virtuale i-esimo

debba essere mappato nella relativo qubit fisico (5 sta per qubit fisico vuoto).
E' a questo punto naturale porsi la domanda di come controllare che la rete effettivamente riesca a mappare qubit virtuali diversi in qubit fisici diversi, ovvero nasce il problema di un controllo dell'autenticità di un output.
Siccome alla rete neurale non viene data nessuna indicazione a priori sulla esclusività che ogni slot fisica deve avere nella fase di mapping (ad esempio il qubit virtuale 2 non deve e non può essere mappato contemporaneamente nel qubit fisico 0 e nel qubit fisico 4) un controllo a posteriori dell'output complessivo restituito dalla rete neurale risulta necessario.
L'output complessivo della rete consiste quindi in 5 vettori contenenti 6 probabilità con il significato prima enunciato. A questo punto il *layout finale*, ovvero il mapping di interesse può essere ottenuto in due modi

- ***Layout senza controllo a posteriori***: In questo approccio di lavoro si prende come layout finale quello ottenuto prendendo gli indici corrispondenti al massimo delle probabilità da ogni vettore di output.
  Se per semplicità di notazione supponiamo di lavorare con algoritmi a 3 qubit e con processori a 3 qubit allora riadattando la struttura della rete al caso preso in esempio si avrebbero tre vettori di output del tipo

$$Slot_0 = (p_{00}, p_{01}, p_{02}) \quad con \quad \sum_{i=0}^{2} p_{0i} = 1 \tag{3.2}$$

$$Slot_1 = (p_{10}, p_{11}, p_{12}) \quad con \quad \sum_{i=0}^{2} p_{1i} = 1 \tag{3.3}$$

$$Slot_2 = (p_{20}, p_{21}, p_{22}) \quad con \quad \sum_{i=0}^{2} p_{2i} = 1 \tag{3.4}$$

  Se si indica con *argmax (v)* l'indice del vettore v corrispondente all'elemento massimo in esso contenuto; si esprimerà il layout ottenuto come output dalla rete neurale come

$$QubitFisico_i \rightarrow argmax \quad (Slot_i) \qquad con \quad i = 0, 1, 2 \tag{3.5}$$

  In tale approccio viene quindi lasciato alla rete il compito, non affatto banale, di apprendere l'esclusività del layout iniziale. Se il massimo di due vettori risulta essere in una posizione uguale del vettore di output l'algoritmo darebbe come layout iniziale un layout in cui un qubit virtuale riuslta essere mappato in due slot fisiche giungendo ad un assurdo.
  E' per tale motivo che questo approccio è stato utilizzato soltanto in una prima fase di studio, per poi essere sostituito da un approccio più robusto che non porta ad assurdi non fisici che minano l'applicabilità sperimentale del risultato.

- ***Layout con controllo a posteriori (c.a.p.):*** Per garantire la presenza di un output sempre sensato si è aggiunto un algoritmo di controllo a posteriori sul layout fornito dalla rete.

E' possibile riassumere il funzionamento di questo controllo a posteriori come segue (ancora una volta per semplicità di notazione si utilizzerà l'esempio di un processore e di un circuito che deve essere mappato a 3 qubit, quanti saranno quindi gli strati di output di una ipotetica rete neurale per tale processore):

1) La rete da come output i tre vettori presentati nelle 3.2,3.3,3.4
2) Si parte dal valore di probabilità più alto tra tutti i vettori $\bar{p}_{xy}$ (dove x è indice del qubit fisico e y indice del qubit virtuale) e si mappa il relativo qubit virtuale nel relativo qubit fisico secondo la 3.5
3) Si esclude allora il vettore x dall'algoritmo in quanto quello slot fisico è stato appena occupato e si ritorna al punto 2), dove però la $\bar{p}_{xy}$ è la probabilità più alta relativa ad una y non ancora mappata

– Se il circuito da mappare è più piccolo della dimensione del processore da utilizzare si inzializzano prima i qubit fisici in cui la probabilità di NaN al loro interno è la più alta, poi si passa all'algoritmo come fin'ora descritto.

Nel corso del prossimo capitolo si andranno ad analizzare i risultati ottenuti con i due diversi approcci: l'approccio con un controllo a posteriori garantisce che seppure il layout della rete non corrisponde esattamente con il layout target del dataset, sia fisicamente e logicamente implementabile, garantendo cosi una robustezza maggiore all'algoritmo proposto.

Capitolo 4

Analisi dei Risultati

In questo capitolo saranno analizzati i risultati ottenuti tramite il modello proposto.
La prima parte del capitolo sarà dedicata al dataset utilizzato per allenare e testare il modello, mostrando come questo è stato generato e quali sono le features che lo compongono. La parte finale del capitolo, invece, sarà relativa all'analisi dei risultati ottenuti in termini di accuarcy con il test set a disposizione. Saranno quindi confrontati i diversi approcci con o senza controllo a posteriori del layout predetto dalla rete, e sarà proposta una leggera modifica alla struttura della neural network vista in Figura 3.1, tenendo conto della composizione del dataset a disposizione.

4.1 La Generazione del Dataset

La prima parte del lavoro di tesi è stata quella di pensare e sviluppare una possibile strategia di collezione per un dataset utilizzabile dal modello di rete neurale basandosi sulla possibilità di generare circuiti quantistici random tramite la libreria Qiskit.
Ogni riga del dataset ottenuto sarà quindi relativa all'analisi di un circuito generato in maniera random mediante la funzione *random_circuit* di Qiskit.
Le features che compongono il dataset sono divisibili in tre macro-categorie: *circuit-related*, *qubit-related* e *backend-related*. Ogni riga sarà poi labellata con il miglior mapping possibile per il circuito generato sul processore utilizzato secondo gli algoritmi forniti dal transpiler di Qiskit.
Si procederà illustrando brevemente qual'è il workflow dell'algoritmo di generazione del dataset:

- **Generazione Circuito**:
  Generazione di un circuito random mediante la funzione *random_circuit* di Qiskit. Questa permette di specificare il numero di qubit da cui tale circuito deve essere composto ed il numero di gates che si vuole compaiano nel circuito. L'operazione immediatamente successiva è quella di eseguire l'operazione di unrolling sui gates di base, ottenendo cosi un circuito che fungerà da riga del dataset. In Figura 4.1 è presente un esempio di circuito random a 5 qubit generato durante la creazione del dataset.

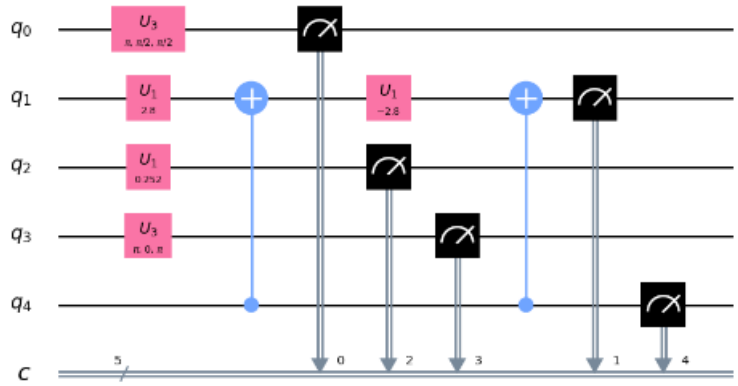

Figura 4.1: Esempio di circuito generato a 5 qubit. Si noti come su tale circuito sia già stata effettuata l'operazione di unrolling sui gate di base.

- **Features *circuit-related***:
  Si passa quindi all'estrapolazione delle features *circuit-related*. Dall'analisi del circuito è possibile ricavare alcune features che sono fondamentali nell'operazione di mapping. Nell'ordine si estrapolano il *numero di qubit* che compone il circuito (nel caso dell'esempio in Figura 4.1 questo è quindi 5); il *numero di porte CNOT* che compaiono nell'algoritmo generato (nel circuito in Figura 4.1 quindi 2); il *numero di operazioni di misura* a cui è sottoposto ciascun qubit del circuito (nell'esempio 1 per ognuno dei qubit); il numero di CNOT tra tutte le coppie di qubit e le sue inverse (nell'esempio in figura 4.1 si avrà quindi cx_14 = 2, mentre cx_ij = 0 $\forall \quad i \neq 1$ e $j \neq 4$. Il motivo per cui si va a considerare sia il numero cx_ij che il numero cx_ji risiede nel fatto che per alcuni processori la direzione in cui i gates a due qubit possono essere eseguiti non è bi-direzionale, come nell'esempio del processore di Melbourne in Figura 1.9)

- **Features *qubit-related***:
  Si estrapolano poi le features relative allo stato dei qubit fisici che compongono il processore da utilizzare. In contemporanea al processo di generazione del circuito si sceglie in maniera random una data e si va a vedere i dati di calibrazione dei qubit del processore in questione per il giorno scelto. Quest'operazione è necessaria poichè gli algoritmi come *Noise Adaptive Layout*[23] e in parte anche il *Dense Layout*[12] del transpiler di Qiskit tengono conto dello stato fisico dei qubit per eseguire l'operazione di mapping. Diventa dunque necessario inserire nel dataset i dati di calibrazione forniti giornalmente da IBM.
  In questo gruppo di features si includono allora i tempi caratteristici *T1 e T2* per ogni qubit del processore (nel caso del processore di Burlington a 5 qubit queste sono complessivamente 10 features) e il valore dell'*readout error* fornito da IBM per ogni qubit fisico del processore.

- **Features *backend-related***:
  Queste features sono sempre estrapolate dai dati di calibrazione forniti da

IBM sullo stato del processore il giorno scelto in maniera random come anticipato nel punto precedente. Per features backend-related si intendono i parametri *edge_length* ed *edge_error* di ogni arco presente nella coupling map. Ovvero nei dati giornalmente forniti da IBM ci sono la lunghezza di esecuzione dei gates a due qubit tra due slot fisiche accoppiate sulla coupling map (tempo in nanosecondi) e la probabilità di errore degli stessi gates. Questi parametri sono considerati dal mapping fornito dagli algoritmi del transpiler di qiskit e vengono perciò inseriti nel dataset utilizzato. Cosi come nel caso delle features relative al numero di CNOT del circuito random prima descritte, anche in questo caso le features backend-related sono composte dalle *edge_length* ed *edge_error* di tutte le possibili coppie e le relative inverse. A tutte le coppie che non sono collegate sulla coupling map viene attribuito un valore di riferimento posto a 100000 per entrambi i parametri. Ciò è necessario per mettere in evidenza l'eventuale importanza della direzionalità dei gates a multiqubit in taluni processori.

- **Scelta del Label**:
  L'operazione finale è quella di estrapolare il miglior label possibile fornito dai due algoritmi messi a disposizione dal transpiler di Qiskit. L'idea è quella di scegliere come label per la riga del dataset un vettore di un numero di elementi pari al numero di qubit fisici del processore da utilizzare (quindi un vettore con 5 elementi nel caso del processore ibmq_burlington) dove ogni elemento corrisponde all'indice del qubit virtuale che deve essere mappato nel qubit fisico del processore indicizzato come la posizione dell'elemento nel vettore. Considerata la Figura 4.2 in blu è presente la coupling map del processore di Burlington dove si evincono gli indici dei relativi qubit fisici, in nero è invece presente il mapping ottenuto per un dato circuito, dove gli indici sono invece relativi ai qubit virtuali dello stesso. Il label di tale riga nel dataset è allora il vettore [1,4,0,2,3].

  Il mapping finale è scelto tra quello fornito dagli algoritmi *Noise Adaptive Layout*(NAL) e *Dense Layout*(DL) del transpiler di qiskit, scegliendo quello che richiede il minor numero di SWAP da inserire durante l'esecuzione dell'algoritmo. Per calcolare tale numero esistono diverse tecniche messe a disposizione da qiskit. Le principali due tecniche sono il *LookhaedSwap*[31] e lo *StochasticSwap*[32], la cui analisi del funzionamento è rimandata alle referenze. Il principio che si è seguito è stato quello di:

  1) Ottenere il layout iniziale sia mediante il NAL che il DL
  2) Per entrambi i layout calcolare la Depth del circuito finale inserendo gli Swap sia con il LookhaedSwap algorithm che con lo StochasticSwap algorithm
  3) Utilizzare come layout iniziale quello che tra le quattro combinazioni possibili fornisse il circuito con Depth minore

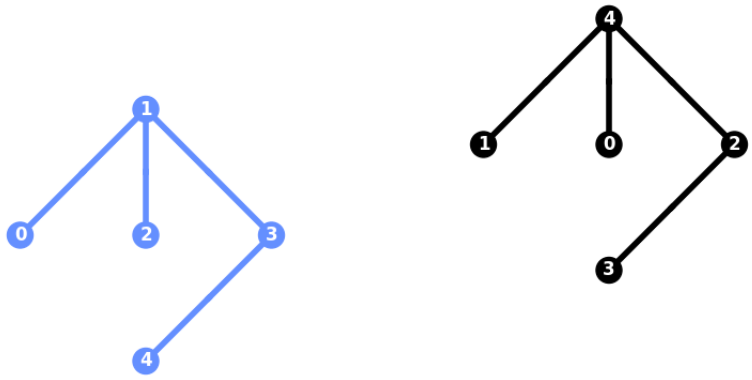

Figura 4.2: Il grafo in blu è la coupling map del processore ibmq_burlington, dove quindi gli indici sono i relativi qubit fisici. Il grafo in nero è invece il layout fornito dall'operazione di mapping, gli indici quindi sono relativi ai qubit virtuali che devono essere mappati in quelle slot.

4.1.1 Analisi del Dataset

Il dataset utilizzato, costruito nella maniera appena spiegata, consiste di 42039 circuiti quantisti random costituiti da 5 qubit e un numero complessivo di CNOT dopo la fase di unrolling minore o uguale a 10. Ognuno di questi è stato per un giorno random dell'anno 2020 mappato sul backend del processore IBMQ Burlington a 5 qubit secondo il miglior algoritmo di mapping fornito dal transpiler di Qiskit, dove si ricordi che per migliore si intende il mapping fornito dall'algoritmo che richiede il minor numero di inserimenti di SWAP durante l'esecuzione del circuito.

La difficoltà di effettuare classificazione multilabel come quella che il modello messo a punto mira ad eseguire, è quella di avere a disposizione dataset bilanciati, ovvero dataset in cui i label di ognuna delle classi per ogni layer di output siano in frequenza confrontabili. Questa richiesta è necessaria per non forzare un determinato layer di uscita a riconoscere bene delle classi, quelle più frequenti, meno bene altre, meno frequenti.
Con lo scopo di effettuare tale analisi si è allora proceduto ai plot di istogrammi di frequenza per ognuna delle slot di uscita (Figura 4.3): questi consentono di esaminare velocemente quante volte all'interno del dataset a disposizione i diversi qubit virtuali sono stati mappati nel relativo slot fisico.
Si evince uno sbilanciamento del dataset in particolare per il qubit fisico numero 4 (SLOT 4). La causa di tale sbilanciamento è attribuibile con buona probabilità alla randomicità del processo di creazione del dataset.

Per sopperire alla mancanza di un dataset perfettamente bilanciato, tuttavia, è possibile effettuare alcuni accorgimenti nella fase di training della rete neurale.

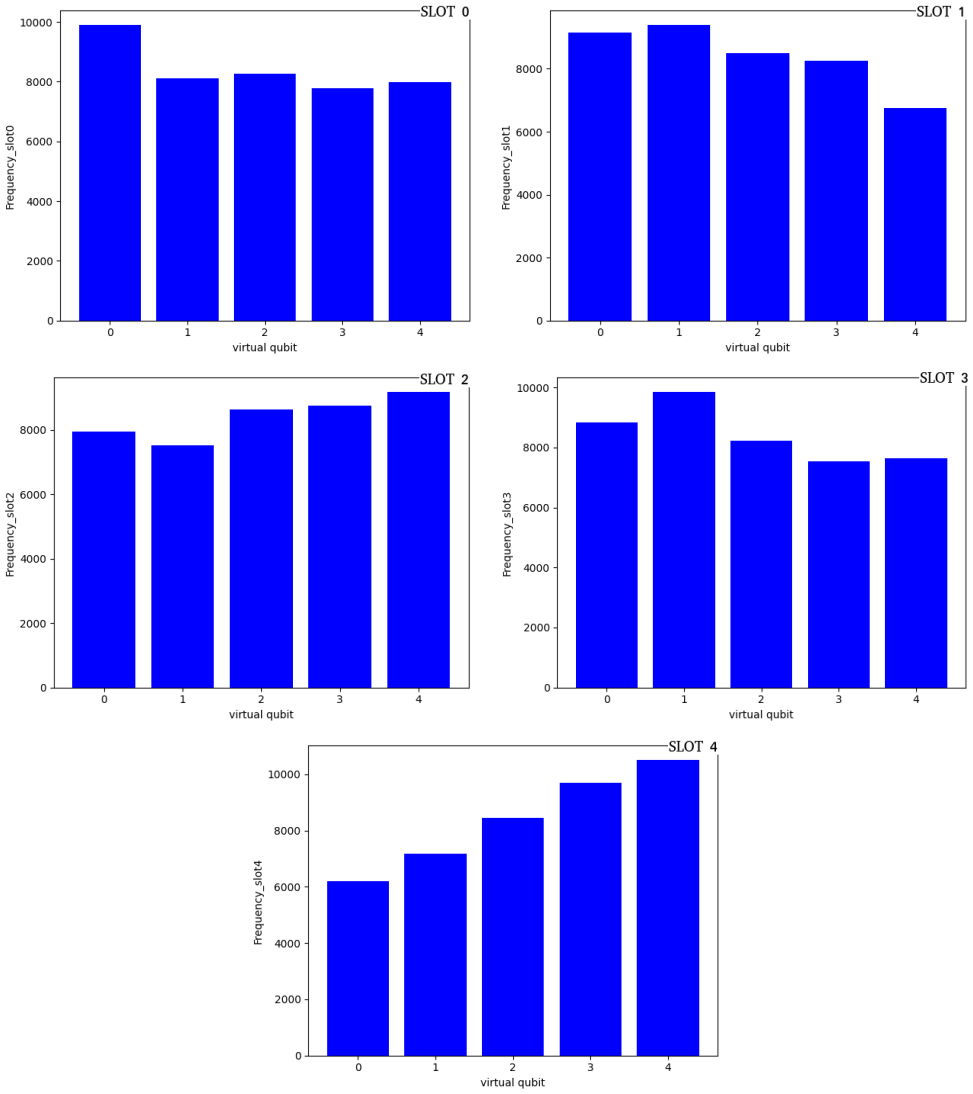

Figura 4.3: Istogrammi di frequenza comparsa classi nel dataset. Ogni istogramma è relativo ad una determinata slot, ed ogni barra indica quante volte il qubit virtuale i-esimo è stato mappato nella relativa slot fisica.

Il principale accorgimento fatto è stato quello di attribuire dei pesi alle diverse classi per ogni slot, in base alla loro frequenza relativa all'interno del dataset [26]. Questo permette di definire una funzione di costo pesata con i pesi trovati mediante la funzione *class_weight* di Keras secondo la relazione:

$$L = \frac{1}{n}\sum_{i=1}^{n} w_i L_i \tag{4.1}$$

dove n è il numero di classi, in questo caso n=5 ed L_i la funzione di costo calcolata per una singola classe
Il valore dei pesi è calcolato in maniera tale da sopperire allo sbilanciamento delle classi. Se si suppone per esempio di lavorare con una classificazione binaria, le cui classi sono 0 e 1, e si suppone di avere un dataset con 5000 istanze di 0 e 45000 di 1, allora i pesi trovati da *class_weight* saranno {0: 5, 1: 0.5}.
Per il dataset utilizzato nel lavoro di tesi, invece, si può riportare come esempio la combinazione di pesi trovati da Keras per la slot più sbilanciata, SLOT 4:

$$SLOT4\_Weights \rightarrow [1.36, 1.17, 0.99, 0.87, 0.80] \tag{4.2}$$

4.2 Training e Testing del Modello

Esaminata la struttura del dataset ci si concentrerà ora sulla fase di training del modello e si definiranno i concetti fondamentali utili alla verifica delle performances di questo.
Secondo il processo di costruzione del dataset presentato nella sezione precedente, questo viene costruito con 91 features. Il primo passaggio eseguito è stato quindi quello di analizzare la possibilità di eliminare alcune features da questo dataset utilizzando cosi solo quelle colonne di esso che realmente codificano l'informazione utile alla operazione di mapping.

Data Preprocessing

Al fine di snellire il dataset passando alla rete soltanto le features che realmente contenessero informazioni si sono per prima cosa eliminate le colonne che rappresentano gli edge errors e gli edge lengths tra le coppie di qubit non presenti sulla coupling map del processore IBMQ_burlington utilizzato. Si è poi visto che eliminando la colonna relativa al numero di misure complessive fatte all'interno del circuito, la cui informazione è comunque dipendente dal numero di misure fatte su ogni qubit, le performances della rete migliorano ulteriormente.
Complessivamente allora il dataset utilizzato consiste di 61 features raggruppabili nelle tre macrocategorie *circuit-related*, *qubit-related* e *qubit-related*.

Il secondo passo fondamentale nel preprocessing dei dati a disposizione è stato quello della **standardizzazione** delle features [26]. Tale processo è di fondamentale importanza in molti degli algoritmi di machine learning per la classificazione. Quest'operazione infatti, garantisce che tutte le features del dataset vengano trattate dall'algoritmo allo stesso modo: se si avessero due features una che varia tra 1 e 10 e l'altra che varia da 1 a 100000 la scrittura di una funzione di

costo come lo scarto quadratico tra predizione e label vero delle istanze sarebbe molto più influenzata dalle features con grossa varianza rispetto che a quelle con poca varianza. Per questo motivo si è utilizzata la standardizzazione ***Standard Scaler***[33] fornita dalla libreria *Scikit-Learn* di Python [34].
Questa calcola per ogni feature del dataset il valore medio μ e la deviazione standard σ, per applicare poi la trasformazione

$$x_{st} = \frac{x - \mu}{\sigma} \tag{4.3}$$

ad ogni x appartenente alla data colonna.

Effettuata la parte di data preprocessing si è quindi proceduto al training vero e proprio del modello. Per far questo si è definito un validation set del 10% del dataset di training. Lo scopo del validation set è quello di fornire uno strumento utile a verificare durante il processo di allenamento della rete neurale, se questa effettivamente sta imparando il modello o se c'è una configurazione di iperparametri completamente sbagliata e ha senso interrompere il training per ridefinire tali iperparametri.
L'idea è quella di avere delle istanze che alla fine di ogni epoca per il training del modello fungano da test in maniera da sottolineare i progressi fatti dalla rete. Questo è uno strumento utile a capire anche dopo quante epoche fermare il processo di allenamento della rete: è infatti possibile generare i due plot in Figura 4.4 e in Figura 4.5 che rappresentano i valori di accuratezza e funzione di costo per ognuna delle slot al variare del numero di epoche. Ovviamente l'obiettivo è quello di ottenere un plot di accuratezza che cresca quanto più possibile, in quanto questo rappresenta la percentuale di predizioni corrette fatte da ogni slot nel validation set, mentre si cerca di abbassare quanto più possibile il valore della loss function, collegato all'errore commesso dal modello. In questo contesto allora il numero di epoche in cui far allenare il modello è facilmente intuibile da questi due plot, in quanto arrivati al plateaux di entrambe le figure non ha senso continuare la fase di training della rete siccome le performances predittive, fissata la complessità del modello, non riescono a migliorare ulteriormente.

Dai due plot in Figura 4.4 e in Figura 4.5 si evidenzia come la Slot 4, in viola, sia la slot che abbia le performances migliori. Tale comportamento è probabilmente attribuibile al pronunciato sbilanciamento che sussiste nel dataset per la Slot 4 (Figura 4.3).
Dall'osservazione di tali andamenti è nata l'idea di modificare leggermente la struttura dello Slot 4 inserendo un layer di dropout tra gli strati *dense_3* e *dense_4* presenti in Figura 3.1. Ci si riferirà al modello contente lo strato di dropout nella Slot 4 come *'Rete Dropout SLOT 4'*.

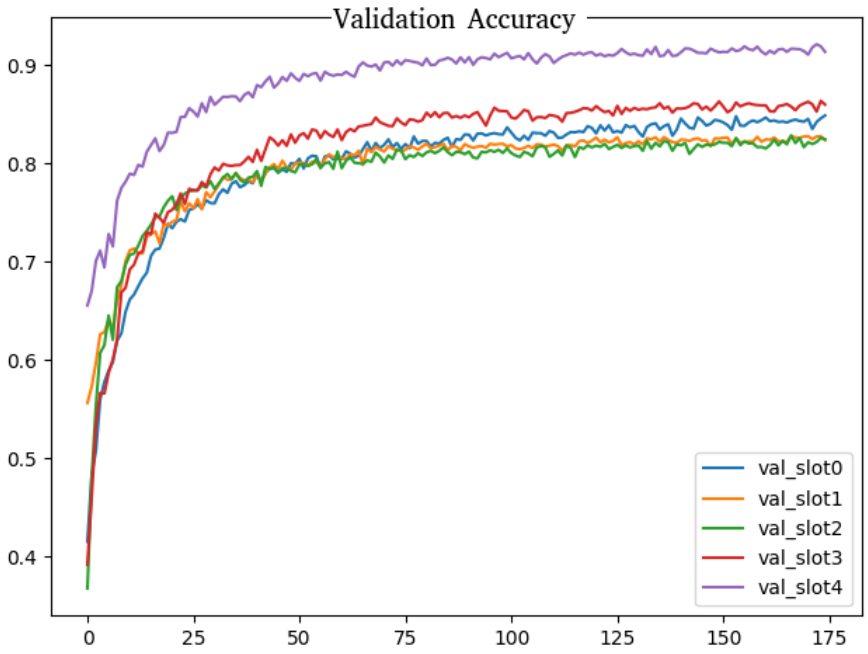

Figura 4.4: Plot dell'accuratezza sul validation set al variare delle epoche di training per tutte e 5 le slot della rete

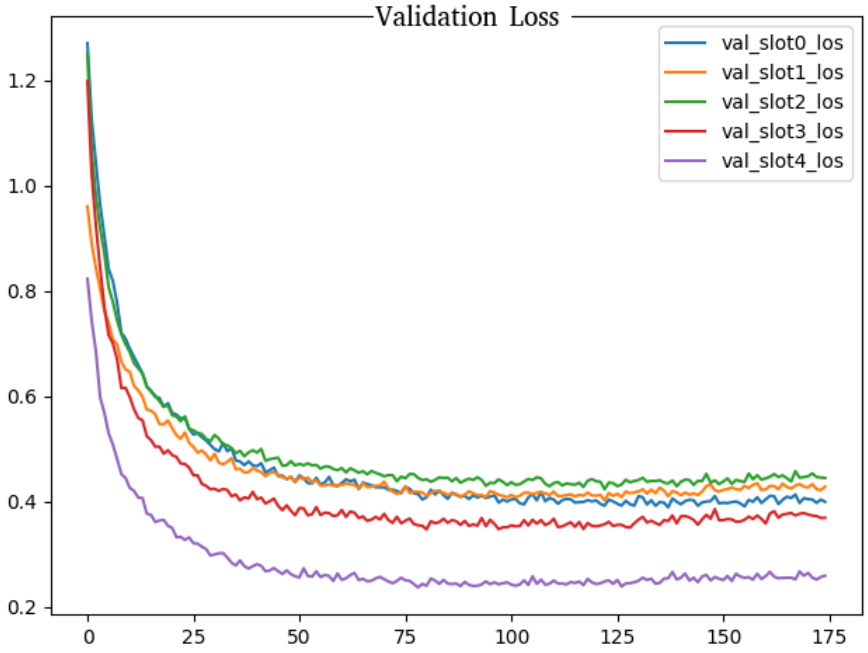

Figura 4.5: Plot del valore della funzione di costo sul validation set al variare delle epoche di training per tutte e 5 le slot della rete

4.3 Valutazione del Modello

In questa sezione saranno presentati i risultati ottenuti dopo la fase di training del modello.
Il parametro fondamentale per valutare le perfomances della rete è l'*accuracy*. Questa è la percentuale di volte in cui la predizione del modello risulta essere in accordo con il mapping presente nel dataset.

$$accuracy = \frac{\#predizioni\_corrette}{\#istanze\_dataset} \tag{4.4}$$

Ci si riferirà ad accuracy di training ed ad accuracy di test a seconda del dataset in considerazione. Siccome l'obiettivo della rete è quello di fare predizione, il parametro realmente interessante è l'accuarcy di test.
Si noti come una predizione viene considerata corretta se e soltanto se tutte e cinque le slot hanno fatto una predizione corretta, ovvero se l'intero vettore di cinque componenti [predizione_0, .., predizione_5] risulta uguale al vettore label dell'istanza [qubit_0, .. , qubit_5]. Ciò abbasserà le percentuali di singole accuracy che hanno le diverse slot separatamente in quanto basta una sola predizione errata delle cinque per considerare l'intero layout non corretto.
Nella Tabella 4.1 sono presentati tutti i valori di accuracy ottenuti.

Tabella 4.1: Tabella riassuntiva dei valori di accuracy per i dataset di training e di test. In grassetto il risultato migliore ottenuto. Con Rete semplice si intende il modello presentato in Figura 3.1, mentre con Rete Dropout SLOT 4 il modello presentato in Figura 3.1 con l'aggiunte di un layer di Dropout tra dense_3 e dense_4 nella struttura della SLOT 4

	Test Acc.	**Train. Acc.**
Rete Semplice	0.7041	0.8471
Rete Semplice con controllo a posteriori	0.7548	0.8867
Rete Dropout SLOT 4	0.7188	0.8295
Rete Dropout SLOT 4 con controllo a posteriori	**0.7645**	0.8752

Analizzando i risultati in 4.1 si evince come l'aggiunta del layer di dropout nella slot 4 va ad aumentare di un punto percentuale l'accuaracy sul test, diminuendo della stessa percenutale l'accuracy sul training facendo intendere una riduzione relativa all'errore di overfitting. Va comunque sottolineato che il miglioramento sulla percentuale di accuratezza risulta troppo piccolo per concludere che l'aggiunta di un layer di dropout nella slot 4 migliori le performances predittive del modello. Per asserire che il modello sia migliorato si dovrebbe andare ad analizzare un dataset di test più grande delle solo oltre 4000 istanze utilizzate.

Con tali limitazioni sulla dimensione del dataset queste fluttuazioni potrebbero essere di natura statistica: un dataset di test di dimensione maggiore potrebbe fare chiarezza su questo risultato.

E' invece opportuno sottolineare come il controllo a posteriori sul layout proposto dalla rete incrementi notevolmente le performances del modello, indicando che con il dataset a disposizione la rete non è in grado di imparare a pieno i vincoli logici che si celano dietro il problema del mapping. Questo limite potrebbe essere in linea di principio superato andando ad aumentare la dimensione del dataset a disposizione.
Per ora il 5% di differenza sui valori di accuracy tra i modelli con e senza controllo a posteriori consiste molto probabilmente, in quei circuiti del dataset in cui il modello di rete neurale da come output un layout logicamente incorretto. In più tutti i layout predetti dal modello con controllo a posteriori dell'output mi garantiscono il rispetto dei vincoli logici di cui sopra, facendo si che anche se la predizione sia diversa dal target del dataset, questa possa essere fisicamente implementata. Una successiva analisi dovrà essere quella di confrontare i layout predetti dalla rete con controllo a posteriori con quelli del dataset per i circuiti in cui i mapping risultano diversi (circa il 25% delle istanze del test). Questa analisi permetterà di capire se la rete neurale riesce a migliorare la qualità del layout iniziale, in termini di inserimenti di SWAP, rispetto a quelli proposti da Qiskit.

Analizzate le capacità predittive complessive dei diversi modelli di reti neurali proposti, è interessante analizzare nel dettaglio come si comportano le singole slot che compongono gli strati di output dei modelli. Un modo di analizzare le capacità predittive delle singole slot è quello che si ottiene mediante le *confusion matrix*[26]. Queste sono matrici quadrate che riportano il conteggio dei True Positive (TP), True Negative (TN), False Positive(FP) e False Negative (FN). Nel caso di una classificazione binaria l'aspetto di tali matrici è quello riportato in Figura 4.6.

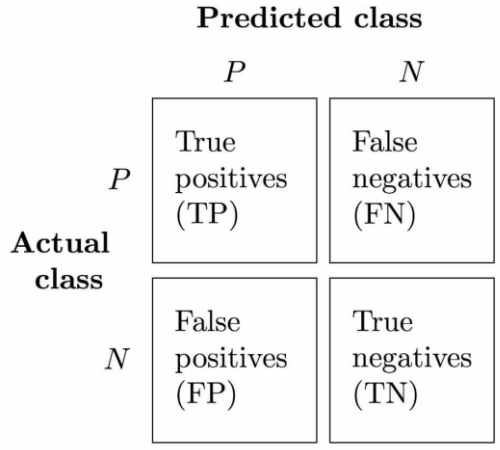

Figura 4.6: Confusion Matrix per una classificazione binaria

Essendo i True Positive quelle classi labellate con 1 e riconosciute dal classificatore come tali, cosi come i True Negative quelle classi labellate con 0 e riconosciute dal classificatore come tali, allora un ottimo classificatore binario presenterebbe una matrice di confusione esattamente diagonale con tutti 0 fuori diagonale, dove i False Negative sono le istanze 1 classificate come 0 e i False Positive sono le istanze 0 classificate come 1.
Nel caso delle slot presenti nel modello di Rete Neurale proposto, la classificazione è tra 5 classi dunque la forma della matrice sarà 5x5 dove la diagonale principale rappresenta le classificazioni corrette, mentre gli elementi fuori diagonale le classificazioni errate, ad esempio l'elemento 01 sarà il numero di volte in cui la slot in questione ha predetto la classe 1 quando la classe dell'istanza è lo 0, e cosi via per tutti gli altri elementi della matrice aventi indici i e j con i≠ j.

In Figura 4.7 sono rappresentate le confusion matrix per ognuna delle slot del modello indicato con drop4. Il numero di elementi sulla diagonale maggiore rispecchia l'alto livello di accuracy delle singole slot di output.
Queste matrici forniscono anche un importante strumento su come andare a migliorare il dataset su cui allenare il modello di rete neurale. L'idea è infatti quella di aumentare la dimensione del dataset di training con istanze i cui label siano nelle righe delle matrici in cui sono presenti più errori.

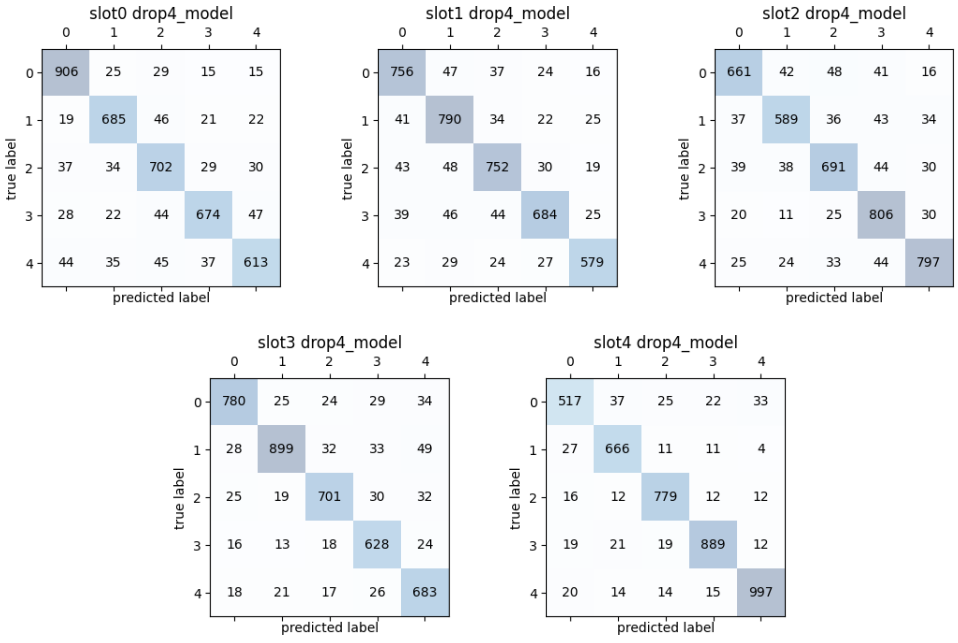

Figura 4.7: Confusion Matrix per ognuna delle slot di output nel modello drop4. Sulla diagonale principale è presente il numero di volte in cui la predizione coincide con il target di test. Gli elementi fuori diagonali sono invece il numero di volte in cui un qubit target i-esimo è stato classificato come qubit j-esimo, con i ≠ j

Conclusioni

Il presente lavoro di tesi si inserisce nell'ambito della ricerca che tenta di utilizzare il machine learning e la quantum computation in maniera combinata. Esso inverte la linea di tendenza seguita nel quantum machine learning, dove la computazione quantistica è a supporto dell'intelligenza artificiale, andando ad utilizzare le tecniche di apprendimento automatico a supporto della computazione quantistica.
Questo elaborato è uno dei primi tentativi di utilizzare il machine learning come strumento utile al processo di compilazione quantistica, in particolare ci si è concentrati nello studiare e migliorare il processo di mapping dei qubit di un dato circuito quantistico su un determinato processore, utilizzando la tecnologia messa a disposizione da *IBM Quantum Experience.*

I risultati presentati nel lavoro di tesi sono da intendersi come punto di partenza per sviluppi futuri: si è dimostrato che un applicazione di sistemi di rete neurale per il mapping dei qubit su processori NISQ (Noise Intermediate Scale Quantum) a 5 qubit è possibile. Si è quindi raggiunto il duplice scopo di trovare, in primis, un metodo di mapping alternativo a quelli esistenti che fosse in generale più versatile, ovvero meno legato al tipo di circuito e al tipo di processore che si utilizza per una data elaborazione quantistica, e in secundis un metodo che possa essere maggiormente scalabile con l'aumentare del numero di qubit rispetto ai metodi di ottimizzatori sotto vincoli (*SMT Solver*) utilizzati. La strada percorsa in tal senso è appena agli inizi, ma presenta già promettenti risultati.
Lo sviluppo naturale del presente lavoro consiste allora in una prima fase di valutazione dei circuiti in cui c'è discrepanza tra i layout predetti dalla rete e quelli target presenti nel dataset. La rete con controllo a posteriori è infatti in grado di fornire sempre un layout che sia implementabile fisicamente, e questa essendo stata allenata utilizzando target sempre ottimali rispetto agli algoritmi a disposizione, potrebbe in linea di principio fornire layout non in accordo con quelli target ma che comunque richiedono un numero di inserimento di SWAP minore di questi ultimi.
Lo sviluppo successivo consiste nel passare ad analizzare la scalabilità di questo approccio nel caso di processori più grandi di quello IBMQ Burlington a 5 qubit. Procedendo, quindi, col testare le capacità predittive di una Deep Neural Network per circuiti più larghi di quelli considerati nel lavoro di tesi, analizzando la qualità dei layout proposti in termini di inserimento di SWAP necessari, e confrontando i tempi richiesti per la predizione tramite rete neurale con quelli richiesti dagli attuali algoritmi di mapping. In tal senso sarà quindi necessario

procedere a costruire un nuovo dataset adatto al processore con più qubit che si intende utilizzare, generando circuiti più grandi di quelli fin'ora utilizzati e in ultima battuta, riadattare la struttura della rete, nella sua parte di output, per a questi nuovi casi.

Il miglioramento del processo di compilazione quantistica è fondamentale per lo sviluppo della quantum computation nel breve e nel lungo periodo. Le restrizioni tecnologiche che ci si trova e ci si troverà ad affrontare con gli attuali e con i futuri processori quantistici vincolano e vincoleranno le performances che gli algoritmi quantistici possono e potranno ottenere. Basti pensare come in un algoritmo variazionale che nel caso quantistico alterna ad ogni iterazione una fase di compilazione ad una di computazione, quanto una compilazione rapida ed efficace, in termini di minimizzazione dell'errore, sia importante [16].
Dimostrando che le reti neurali sono in grado di eseguire l'operazione di mapping e proponendo un metodo di sviluppo di dataset utilizzabile anche per circuiti e processori più grandi, si apre una strada anche nel lungo periodo: se in futuro un dato gruppo di ricerca sarà in grado di fornire un nuovo algoritmo di mapping che in taluni casi potrà funzionare meglio di quelli già utilizzabili, si potranno inserire tali casi nel dataset di training della rete neurale e questa sarà quindi in grado di fornire un mapping tenendo conto anche del nuovo algoritmo proposto. Questo approccio fa si che il problema della scelta dell'algoritmo di mapping da utilizzare per una data esecuzione di un circuito quantistico su un processore quantistico, sia tolta all'utente, e scelta in maniera ottimale dalla rete andando a minimizzare la probabilità di errore durante il running dell'algoritmo, minimizzando il numero di gate di SWAP da inserire nel circuito da eseguire.
Se si tiene infine conto che porzioni di Error Correction Code (ECC) non sono inseribili all'interno degli attuali e dei futuri (almeno nel breve periodo) algoritmi quantistici [11] data il piccolo numero di qubit disponibile per la computazione, la minimizzazione dell'errore in fase di esecuzione risulta necessaria se si vuole ottenere quella che è stata definita *quantum supremacy* [10], giustificando tutti gli sforzi fatti dalla comunità scientifica (e non solo) nell'ambito della computazione quantistica.

Elenco delle figure

1.1 Sfera di Bloch 8
1.2 Gates X, Z ed H 10
1.3 Schematizzazione Grafica dell'azione di H 12
1.4 Rappresentazione circuitale del CNOT 13
1.5 Rappresentazione circuitale della porta di Toffoli 13
1.6 Decomposizione dello SWAP in CNOT 14
1.7 Tempi di Decoerenza 15
1.8 Rappresentazione circuitale di una giunzione Josephon 17
1.9 Copuling Map IBMQ_Melbourne_16 17
1.10 Coupling Map IBMQ_Burlington 18
1.11 Schema Calibrazione IBMQ_Burlington 20
1.12 Operazione di compilazione di un circuito quantistico 21
1.13 Esempio di circuito quantistico a due qubit 22
1.14 Esempio di unrolling 22
1.15 Rappresentazione grafica dell'operazione di Mapping 23

2.1 Activation e Loss Function 29
2.2 Neuroni Biologici 30
2.3 Modello di McCulloch-Pitts 31
2.4 Rete Neurale Artificiale 32
2.5 Gradient Descent 32
2.6 Esempio di Overfitting 35
2.7 Bias-Variance tradeoff pt.1 36
2.8 Bias-Variance tradeoff pt.2 36
2.9 Bias-Variance tradeoff pt.3 37
2.10 Regolarizzazione L2 38
2.11 Regolarizzazione L1 39
2.12 Dropout Layer 39

3.1 Rete Neurale per il Mapping 41

4.1 Esempio circuito random generato 47
4.2 Layout sul processore IBMQ Burlington 49
4.3 Istogrammi di Frequenza Classi 50
4.4 Validation Accuracy vs Numero di Epoche 53
4.5 Validation Loss Function vs Numero di Epoche 53
4.6 Confusion Matrix 55

4.7 Confusion Matrix per ogni Slot 56

Bibliografia

[1] Classical computing, quantum computing, and Shor's factoring algorithm YI Manin - arXiv preprint quant-ph/9903008, 1999 - arxiv.org

[2] R. P. Feynman, "Simulating physics with computers," International Journal of Theoretical Physics, vol. 21, no. 6, pp. 467–488, 1982. [Online]. Available: https://doi.org/10.1007/BF02650179

[3] D. Deutsch, Quantum theory, the Church-Turing principle and the universal quantum computer,Proc. R. Soc. Lond. A 400:97–117 (1985).

[4] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware- efficient variational quantum eigensolver for small molecules and quantum magnets. Nature, 549:242 EP –, Sep 2017

[5] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. Nature Communications, 5:4213 EP –, Jul 2014. Article

[6] P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Review, 41(2):303–332,1999.

[7] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. Nature, 549:195 EP –, Sep 2017.

[8] IBM. IBM Quantum Devices. https://quantumexperience.ng.bluemix.net/qx/devices, 2018. Accessed: 2018-05-16.

[9] Arute, F., Arya, K., Babbush, R. et al. Quantum supremacy using a programmable superconducting processor. Nature 574, 505–510 (2019). https://doi.org/10.1038/s41586-019-1666-5

[10] Sergio Boixo, Sergei V. Isakov, Vadim N. Smelyanskiy, Ryan Babbush, Nan Ding, Zhang Jiang, Michael J. Bremner, John M. Martinis, Hartmut Neven. Characterizing Quantum Supremacy in Near-Term Devices. arXiv:1608.00263 [quant-ph]

[11] John Preskill. Quantum Computing in the NISQ era and beyond, 2018.

[12] https://qiskit.org/documentation/stubs/qiskit.compiler.transpile.html

[13] https://qiskit.org/

[14] Introduzione alla fisica teorica. Piero Caldirola, Giovanni Prosperi, e al. — 26 lug. 2003 Utet Editore

[15] Michael A. Nielsen and Isaac L. Chuang. 2011. Quantum Computation and Quantum Information: 10th Anniversary Edition (10th ed.). Cambridge University Press, New York, NY, USA

[16] Pranav Gokhale, Yongshan Ding, Thomas Propson, Christopher Winkler, Nelson Leung, Yunong Shi, David I. Schuster, Henry Hoffmann, Frederic T. Chong. Partial Compilation of Variational Algorithms for Noisy Intermediate-Scale Quantum Machines arXiv:1909.07522 [quant-ph]

[17] Feynman, R.P. Simulating physics with computers. Int J Theor Phys 21, 467–488 (1982). https://doi.org/10.1007/BF02650179

[18] Koch, J., Terri, M. Y., Gambetta, J., Houck, A. A., Schuster, D., Majer, J., Blais, A., Devoret, M. H., Girvin, S. M., and Schoelkopf, R. J. (2007) Charge-insensitive qubit design derived from the cooper pair box. Physical Review A, 76, 042319.

[19] Martinis, J. M. and Osborne, K. (2004) Superconducting qubit and the physics of josephson junctions. arXiv preprint cond-mat/0402415.

[20] Benenti, G., Casati, G., and Strini, G. (2007) Principles of quantum computation and information: Volume II: Basic Tools and Special Topics. World Scientific Publishing Company

[21] Adi Botea, Akihiro Kishimoto, Radu Marinescu. On the Complexity of Quantum Circuit Compilation

[22] https://qiskit.org/documentation/apidoc/transpiler_passes.html#routing

[23] Prakash Murali, Jonathan M. Baker, Ali Javadi Abhari,Frederic T. Chong, Margaret Martonosi. Noise-AdaptiveCompiler Mappings for Noisy Intermediate-Scale Quan-tum Computers. arXiv:1901.11054 [quant-ph]

[24] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. Machine learning: An artificial intelligence approach. Springer Science & Business Media, 2013.

[25] McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943). https://doi.org/10.1007/BF02478259

[26] Sebastian Raschka, Vahid Mirjalili - Python Machine Learning. 2nd ed-Packt Publishing (2017)

[27] A Method for Stochastic Optimization, Diederik P. Kingma and Jimmy Lei Ba, 2014. https://arxiv.org/abs/1412.6980.

[28] https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a

[29] https://it.wikipedia.org/wiki/Keras

[30] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[31] https://qiskit.org/documentation/stubs/qiskit.transpiler.passes.LookaheadSwap.html

[32] https://qiskit.org/documentation/stubs/qiskit.transpiler.passes.StochasticSwap.html

[33] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[34] https://scikit-learn.org/stable/