Università degli studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base

Area Didattica di Scienze Matematiche Fisiche e Naturali

Dipartimento di Fisica "Ettore Pancini"

Laurea Triennale in Fisica

Quantum Machine Learning: implementazione quantum graph recurrent neural network

Relatore:

Giovanni Acampora

Candidato:

Davide Romano

Matr. *N85001247*

Anno Accademico 2019/2020

Indice

Capitolo 1

Introduzione

Il *quantum computing* è lo studio di un modello computazionale basato sulle leggi della meccanica quantistica. Il *machine learning* è la scienza che si occupa di rendere i computer in grado di apprendere dai dati, senza che siano esplicitamente programmati. É prevedibile che nei prossimi anni entrambe le discipline giocheranno un ruolo sempre più centrale nella gestione dell'informazione e dei big data, motivo per cui la forte spinta alla ricerca non viene unicamente dalla comunità scientifica ma anche dai colossi della tecnologia dell'informazione. Dal momento che il volume dei dati stoccati globalmente sta vivendo una crescita esponenziale - la IDC prevede che entro il 2025 si toccherà la soglia dei 175 zettabytes - urge la necessità di trovare degli approcci di machine learning innovativi e più efficienti. Un'idea che sta riscuotendo molto successo è quella di sfruttare il quantum computing per migliorare gli algoritmi di machine learning.

1.1 Background

Mentre i computer classici sono basati su circuiti elettronici che possono essere descritti con le leggi della fisica classica, i quantum computer si basano su sistemi microscopici quali fotoni, elettroni, atomi che obbediscono alle leggi della teoria quantistica. Per questo motivo fenomeni tipicamente quantistici, come la sovrapposizione o l'entanglement, sono propri dei quantum computer e in essi vengono sfruttati per risolvere alcune classi di problemi in modo più efficiente. Una delle principali difficoltà consiste

proprio nel capire in quali campi e per quali problemi risulta vantaggioso l'impiego di dispositivi quantistici.

In generale, si parla di *supremazia quantistica* se un dispositivo quantistico risulta in grado di risolvere uno specifico problema che alcun computer classico riesce a risolvere in un tempo ragionevole; più precisamente, è necessario che si abbia uno speedup superpolinomiale. Il raggiungimento della supremazia quantistica non implica l'obsolescenza dei computer classici: come si vedrà più avanti, dispositivi classici e quantistici spesso cooperano in modo che si possano sfruttare i punti di forza degli uni e degli altri.

1.2 Potenziali applicazioni del quantum computing

Gran parte dei risultati ad oggi ottenuti nel campo del quantum computing sono di tipo teorico. Come sarà meglio spiegato nella sezione 3.3.1, ciò è dovuto principalmente alle molte difficoltà che si incontrano nella realizzazione fisica di un hardware quantistico. Ad ogni modo, i progressi tecnologici degli ultimi anni sono incoraggianti: nel 2016 IBM realizza il primo hardware quantistico a 5 qubit accessibile ai ricercatori via cloud; nel 2019 Google e IBM annunciano la realizzazione di dispositivi a 20 qubit, mentre oggi entrambe le compagnie dispongono di dispositivi basati su 30-50 qubit.

A prescindere dai limiti tecnologici oggi esistenti, la ricerca ha già individuato alcuni campi che potranno trarre vantaggio dall'introduzione del quantum computing:

- **Crittonalisi**: la maggior parte dei sistemi di crittografia si basa sull'elevata complessità computazionale della fattorizzazione in numeri primi. L'algoritmo di Shor (sez. 2.5.1) mostra uno speedup esponenziale rispetto al più efficiente algoritmo classico di fattorizzazione, perciò con un computer quantistico sufficientemente potente risulterebbe possibile scardinare la maggior parte dei crittosistemi esistenti;

- **Simulazione**: la simulazione di sistemi quantistici su computer classici è sempre stata limitata dalla complessità intrinseca dei sistemi quantistici. I computer quantistici risulterebbero naturalmente ideali per la simulazione di fenomeni quantistici.

- **Machine learning**: grazie agli hardware sempre più potenti e alla continua evoluzione degli algoritmi di apprendimento, le tecniche di machine learning

diventano strumenti sempre piu utili per la ricerca di pattern nei dati. Dal momento che i sistemi quantistici producono dei pattern contro intuitivi e soprattutto difficili da riprodurre per i dispositivi classici, è ragionevole pensare che i computer quantistici possano risultare più performanti in alcune mansioni di machine learning.

Capitolo 2

Quantum computing

2.1 Qubit

Un bit è un'unità di informazione che descrive un sistema classico bidimensionale. Adottando la notazione di Dirac, possiamo indicare i due stati possibili come:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \text{e} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.1}$$

Un qubit è un'unità di informazione che descrive un sistema quantistico bidimensionale. Lo stato generico di un qubit è descritto da una *sovrapposizione* degli stati $|0\rangle$ e $|1\rangle$:

$$|\psi\rangle = c_0 |0\rangle + c_1 |1\rangle = c_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + c_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \equiv \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}, \tag{2.2}$$

dove c_0 e $c_1 \in \mathbb{C}$ sono le ampiezze di probabilità, dunque $|c_0|^2$ e $|c_1|^2$ le probabilità che il qubit si trovi nello stato $|0\rangle$ o nello stato $|1\rangle$ rispettivamente, per cui $|c_0|^2 + |c_1|^2 = 1$.

I vettori $|0\rangle$ e $|1\rangle$ formano una base ortonormale in $\mathbb{C}^2$, detta *base computazionale standard*.

2.1.1 Cambio di base

Si introducono di seguito le *matrici di Pauli*, matrici 2 x 2, unitarie, hermitiane:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{2.3}$$

le quali, con l'inclusione della matrice identità I_2, formano una base ortogonale per lo spazio di Hilbert delle matrici 2 x 2 complesse.

Si può osservare che gli stati $|0\rangle$ e $|1\rangle$ che si è detto costituire la base computazionale standard sono autovettori dell'operatore σ_z.

Si introducono infine la base formata dagli stati $|i+\rangle$ e $|i-\rangle$, autovettori dell'operatore σ_y:

$$|i+\rangle = \frac{|0\rangle + i\,|1\rangle}{\sqrt{2}} \qquad |i-\rangle = \frac{|0\rangle - i\,|1\rangle}{\sqrt{2}}, \tag{2.4}$$

e la base formata dagli stati $|+\rangle$ e $|-\rangle$, autovettori dell'operatore σ_x:

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \tag{2.5}$$

tutti rappresentati sulla sfera di Bloch in figura 2.1.

2.1.2 Registri quantistici

Dato un sistema di n bit, cioè un registro classico, i possibili 2^n stati possono essere ottenuti *uno per volta*. Se si considera invece un registro quantistico di n qubit, i 2^n stati sono in *sovrapposizione*, coesistono simultaneamente.

Per descrivere un sistema di 8 bit è sufficiente una stringa di 8 numeri che possono assumere valore 0 o 1, per esempio 00101101. L'analogo quantistico è invece descritto dal seguente prodotto tensoriale:

$$|00101101\rangle = |0\rangle \otimes |0\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle\,, \tag{2.6}$$

che è un vettore di 256 righe appartenente a $(\mathbb{C}^2)^{\otimes 8}$. Il generico stato di un sistema di 8 qubit è quindi esprimibile come:

$$|\psi\rangle = c_0\,|00000000\rangle + \cdots + c_{45}\,|00101101\rangle + \cdots + c_{255}\,|11111111\rangle\,, \tag{2.7}$$

dove $\sum_{i=0}^{255} |c_i|^2 = 1$. Questa discrepanza tra classical computing e quantum computing cresce esponenzialmente col numero di qubit. Un sistema di 32 qubit necessita di $2^{32} = 4,294,967,296$ numeri complessi per essere simulato su un computer classico. Questo enorme potenziale si esprime appieno in una proprietà, detta *parallelismo quantistico* (sez. 2.3), ampiamente sfruttata dagli algoritmi quantistici.

2.1.3 Stati entagled

Se un sistema di più qubit è esprimibile come prodotto tensoriale tra i vettori di stato dei singoli qubit (come nel caso 2.6) esso si dice essere in uno stato *separabile*. L'*entanglement* è una proprietà puramente quantistica per la quale un sistema - in questo caso di qubit - non può essere espresso come il prodotto tensoriale dei singoli vettori di stato. Un sistema siffatto si dice essere in uno stato *entangled*. Intuitivamente, ciò significa che tra gli stati dei singoli qubit esiste una correlazione, la quale non dipende dalla distanza che separa i qubit.

Tale correlazione si palesa nei processi di misura; si consideri il seguente stato entangled, detto *stato di Bell*:

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{2.8}$$

- prima di effettuare alcuna misura, entrambi i qubit hanno probabilità $\frac{1}{2}$ di essere nello stato $|0\rangle$ o nello stato $|1\rangle$;
- se si misura il 1°(2°) qubit e lo si trova nello stato $|0\rangle$, allora il 2°(1°) qubit è certamente nello stato $|0\rangle$;
- se si misura il 1°(2°) qubit e lo si trova nello stato $|1\rangle$, allora il 2°(1°) qubit è certamente nello stato $|1\rangle$.

L'entanglement è alla base di soluzioni di problemi in information-processing che non hanno analogo classico: un esempio è il *teletrasporto quantistico*.

2.1.4 Interpretazione geometrica

Sfruttando la *sfera di Bloch* (figura 2.1) si può ottenere una utile rappresentazione visiva del qubit, associando i punti della sfera ai possibili stati di un singolo qubit. In particolare il polo nord corrisponde allo stato $|0\rangle$ e il polo sud allo stato $|1\rangle$. I punti sulla linea equatoriale rappresentano stati di sovrapposizione uniforme del qubit, cioè stati per i quali i risultati $|0\rangle$ e $|1\rangle$ di una misura sono equiprobabili.

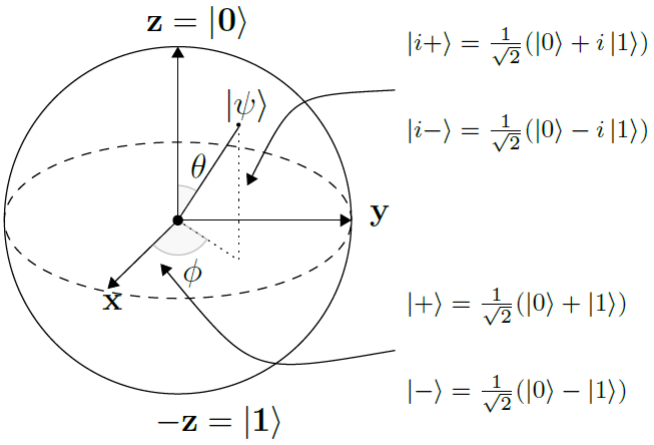


Figura 2.1: Sfera di Bloch

Un generico punto sulla sfera può essere espresso come:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle, \tag{2.9}$$

dove θ e $\phi \in \mathbb{R}$ sono le coordinate sferiche del punto. Risulta quindi chiara la corrispondenza biunivoca tra gli stati di un singolo qubit e i punti della sfera di Bloch.

2.1.5 Interpretazione fisica

Se la rappresentazione fisica di un bit può essere un qualsiasi sistema classico a due stati - come per esempio due diversi valori di tensione in un circuito elettrico - la rappresentazione fisica di un qubit è possibile sfruttando le proprietà di un qualsiasi sistema quanto-meccanico a due stati.

Esempi di supporti fisici sono:

- i due diversi stati di polarizzazione di un fotone;
- i due stati di spin di un elettrone o di un nucleo;
- due livelli energetici di un elettrone che orbita in un singolo atomo;

...e molti altri, su ognuno dei quali il controllo viene effettuato diversamente. Si consideri a titolo di esempio l'ultimo dei tre: il passaggio dell'elettrone da un orbitale all'altro viene controllato tramite impulsi laser di adeguata intensità, durata e lunghezza d'onda.

2.2 Quantum gates

Come nei computer classici i bit vengono manipolati per mezzo di porte logiche, così nei computer quantistici ci si serve di *porte logiche quantistiche* per manipolare lo stato dei qubit. Una delle principali differenze fra le porte logiche classiche e quantistiche è la reversibilità delle seconde, motivo per il quale il numero di qubit in input e in output è sempre lo stesso.

Da un punto di vista formale, le porte logiche quantistiche sono rappresentate da matrici unitarie che operano sui vettori di stato.

Si introducono di seguito le più comuni porte logiche quantistiche.

2.2.1 Single qubit gates

NOT gate

La porta `NOT` anche detta X è rappresentata dalla matrice di Pauli σ_x. L'effetto su un qubit è quello di invertire le ampiezze di probabilità:

$$X \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_0 \end{pmatrix}. \tag{2.10}$$

Il relativo simbolo circuitale è:

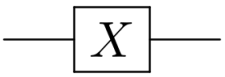


Figura 2.2: `NOT` gate

Z gate

La porta Z, rappresentata dalla matrice di Pauli σ_z, agisce solo sullo stato $|1\rangle$ cambiandone il segno dell'ampiezza di probabilità:

$$Z \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} c_0 \\ -c_1 \end{pmatrix}. \tag{2.11}$$

Il relativo simbolo circuitale è:

Figura 2.3: Z gate

Hadamard gate

La porta H è rappresentata dalla seguente matrice:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{2.12}$$

Essa trasforma gli stati della base computazionale standard $|0\rangle$ e $|1\rangle$ in una sovrapposizione di stati, più precisamente negli stati $|+\rangle$ e $|-\rangle$ rispettivamente:

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle, \tag{2.13}$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle. \tag{2.14}$$

Il relativo simbolo circuitale è:

Figura 2.4: H gate

Nella sfera di Bloch l'operazione H corrisponde ad una rotazione di ampiezza π intorno all'asse $x + z$.

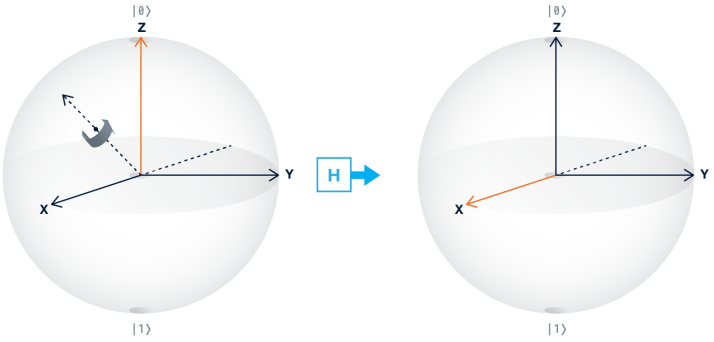


Figura 2.5: Visualizzione grafica dell'azione della porta di Hadamard sullo stato $|0\rangle$

Porte di rotazione

Le porte RX, RY e RZ effettuano una rotazione del qubit di un angolo ϕ rispettivamente attorno all'asse x, y e z della sfera di Bloch. L'angolo ϕ è un parametro della porta logica. Sono rappresentate dalle seguenti matrici:

$$RX(\phi) = e^{-i\phi\sigma_x/2} = \begin{pmatrix} \cos(\phi/2) & -i\sin(\phi/2) \\ -i\sin(\phi/2) & \cos(\phi/2) \end{pmatrix}, \tag{2.15}$$

$$RY(\phi) = e^{-i\phi\sigma_y/2} = \begin{pmatrix} \cos(\phi/2) & \sin(\phi/2) \\ \sin(\phi/2) & \cos(\phi/2) \end{pmatrix}, \tag{2.16}$$

$$RZ(\phi) = e^{-i\phi\sigma_z/2} = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix}. \tag{2.17}$$

Sono indicate con i seguenti simboli

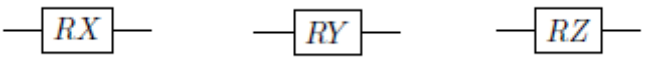


Figura 2.6

2.2.2 Multiple qubit gates

Le operazioni su registri quantistici a due o più qubit sono ottenute tramite le porte logiche a più qubit. Esse assumono particolare rilievo quando si parla di stati entan-

gled: non essendo questi scrivibili come prodotto tensoriale dei singoli vettori di stato, non è possibile operare sul registro mediante operazioni sui singoli qubit.

CNOT gate

La porta `CNOT` (controlled-`NOT`) è l'analogo quantistico della porta `XOR`: essa agisce su un sistema di due qubit di cui il primo è detto di *controllo* e il secondo è detto *target*. La porta `CNOT` è rappresentata dalla matrice:

$$\texttt{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.18}$$

Dati due qubit A e B, essa effettua la seguente operazione:

$$\texttt{CNOT}\,|A, B\rangle = |A, B \oplus A\rangle\,, \tag{2.19}$$

quindi secondo la logica seguente:

- se il controllo è 0 il target resta inalterato;
- se il controllo è 1 il target viene negato.

Il relativo simbolo circuitale è:

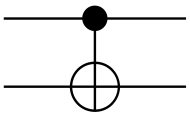


Figura 2.7: `CNOT` gate

SWAP gate

La porta $SWAP$ è rappresentata dalla seguente matrice:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.20}$$

Essa scambia i due qubit cui è applicata:

$$SWAP\,|01\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |10\rangle\,. \tag{2.21}$$

Il simbolo circuitale più diffuso per la porta $SWAP$ è:

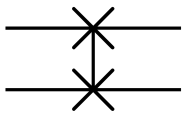


Figura 2.8: SWAP gate

CSWAP gate

La porta CSWAP o Fredkin gate agisce su tre qubit di cui il primo è di controllo e gli altri due sono qubit target. Se il qubit di controllo è $|0\rangle$ i tre qubit non subiscono alcuna alterazione, se invece il qubit di controllo è $|1\rangle$ allora gli altri due vengono scambiati:

$$CSWAP\,|0,\phi,\psi\rangle = |0,\phi,\psi\rangle\,, \tag{2.22}$$

$$CSWAP\,|1,\phi,\psi\rangle = |1,\psi,\phi\rangle\,. \tag{2.23}$$

La porta CSWAP è rappresentata dalla seguente matrice:

$$CSWAP = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.24}$$

Il relativo simbolo circuitale è:

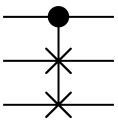


Figura 2.9: CSWAP gate

RZZ gate

La porta RZZ ha la seguente rappresentazione matriciale:

$$RZZ(\phi) = e^{-i\frac{\phi}{2}Z\otimes Z} = \begin{pmatrix} e^{-i\phi/2} & 0 & 0 & 0 \\ 0 & e^{i\phi/2} & 1 & 0 \\ 0 & 1 & e^{i\phi/2} & 0 \\ 0 & 0 & 0 & e^{-i\phi/2} \end{pmatrix}. \tag{2.25}$$

Essa produce un'interazione $Z \otimes Z$ fra i due qubit su cui agisce.

Set universale

Infine, si introduce il concetto di *universalità* per le porte logiche, facendo riferimento a [14]

> a set of quantum gates is said to be universal if any unitary transformation of the quantum data can be efficiently approximated arbitrarily well as sequence of gates in the set.

Dunque qualsiasi trasformazione unitaria può essere approssimata come una sequenza finita di porte di un set universale.

2.3 Parallelismo quantistico

Il parallelismo quantistico è una caratteristica di molti algoritmi quantistici per la quale una funzione può essere valutata su input multipli simultaneamente sfruttando la sovrapposizione.

Supponiamo di avere una funzione $f(\phi) : \{0,1\}^2 \to \{0,1\}$. Per quanto detto, la funzione f può essere realizzata tramite un set di porte logiche, che indichiamo con O. Quindi se lo stato iniziale è $|\psi_0\rangle = |\phi, 0\rangle$ lo stato finale sarà $|\psi_1\rangle = O\,|\psi_0\rangle = |\phi, f(\phi)\rangle$.

Sia lo stato $|\phi\rangle$ inizializzato come

$$|\phi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \tag{2.26}$$

allora lo stato iniziale è

$$|\psi_0\rangle = |\phi, 0\rangle = \frac{1}{2}\big(|00\rangle + |01\rangle + |10\rangle + |11\rangle \big) \otimes |0\rangle\,, \tag{2.27}$$

e lo stato finale

$$\begin{aligned} |\psi_1\rangle &= O\,|\phi, 0\rangle = \frac{1}{2}\big(O\,|00, 0\rangle + O\,|01, 0\rangle + O\,|10, 0\rangle + O\,|11, 0\rangle\big) \\ &= \frac{1}{2}\big(|00, f(00)\rangle + |01, f(01)\rangle + |10, f(10)\rangle + |11, f(11)\rangle \big). \end{aligned} \tag{2.28}$$

Come si può vedere la funzione f è stata valutata per i quattro input simultaneamente. La stessa operazione in un computer classico può essere eseguita serialmente per i singoli input. È importante chiarire che gli output sono anch'essi stati quantistici in sovrapposizione, non sono quindi simultaneamente accessibili in senso classico. Ad ogni modo, come si vedrà nell'algoritmo di Shor (sez. 2.5.1), il parallelismo quantistico viene sfruttato come step intermedio.

2.4 Circuiti quantistici

Il quantum computing comprende diversi modelli operazionali:

- circuiti quantistici;
- calcolo quantistico adiabatico;
- calcolo quantistico topologico;
- calcolo quantistico unidirezionale.

In questa sezione saranno brevemente introdotti i circuiti quantistici. I circuiti quantistici sono l'analogo più semplice se si pensa ai computer classici: linee di trasmissione connettono porte logiche quantistiche che operano sui qubit.

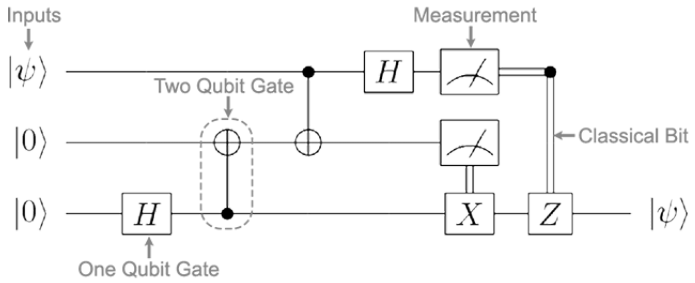


Figura 2.10: Esempio di circuito quantistico

Linee singole trasmettono stati quantistici, linee doppie bit classici. La misura è rappresentata dal simbolo  .

Le principali porte logiche quantistiche sono state illustrate nella sezione 2.2; altri due elementi fondamentali nella logica dei circuiti quantistici sono la codifica dei dati e il processo di misura.

2.4.1 Data encoding

Perché un circuito quantistico possa operare sui dati classici in input è necessario che questi siano codificati nello stato o nella dinamica di un sistema quantistico. Tale processo non è affatto banale e in esso risiede buona parte della complessità di un algoritmo quantistico; inoltre a causa della *decoerenza* (sez. 2.4.2) indotta dal processo di misura, la codifica è necessaria ogni qual volta un algoritmo quantistico viene ripetuto.

Per codifica si intende quindi il "caricamento" di un set di dati $\mathcal{X}$ in stati quantistici $\mathcal{D}_n$ di n qubit.

Di seguito saranno illustrate alcune delle strategie di codifica più diffuse; in ogni caso la codifica viene effettuata da un circuito S_x a partire dallo stato iniziale $|\phi\rangle = |0\rangle^{\otimes n}$ (figura 2.11).

Basis encoding

La codifica nella base consiste nell'associare a una stringa di n bit classici un vettore di stato della base computazionale di un sistema di n qubit; in altre parole, ogni bit viene sostituito da un qubit, quindi qualsiasi operazione agisce in parallelo su tutte le sequenze di bit in sovrapposizione. Formalmente, se si ha $x^{(m)} \in \{0,1\}^n$ il dataset

può essere scritto come la seguente sovrapposizione di stati:

$$|\mathcal{D}_n\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} \left|x^{(m)}\right\rangle . \tag{2.29}$$

Per esempio, nel caso $n = M = 4$, con $x^{(1)} = 0010$, $x^{(2)} = 0101$, $x^{(3)} = 0110$, $x^{(4)} = 1010$, si ha:

$$|\mathcal{D}_4\rangle = \frac{1}{\sqrt{4}} |0010\rangle + \frac{1}{\sqrt{4}} |0101\rangle + \frac{1}{\sqrt{4}} |0110\rangle + \frac{1}{\sqrt{4}} |1010\rangle . \tag{2.30}$$

Per questo tipo di codifica, un algoritmo quantistico ha lo scopo di accrescere la probabilità relativa allo stato che codifica la soluzione. Le ampiezze di probabilità infatti non codificano informazione ma semplicemente "segnano" il risultato di una computazione con una probabilità sufficientemente alta. Per esempio, se lo stato $|0101\rangle$ ha una probabilità $|a_{0101}|^2 > \frac{1}{2}$, è chiaro che esso, dopo ripetute esecuzioni dell'algoritmo e misure, sarà il risultato complessivo dell'algoritmo.

Amplitude encoding

La codifica nelle ampiezze permette di codificare 2^n caratteristiche nelle ampiezze di probabilità di uno stato di n qubit. Quindi un vettore 2^n-dimensionale è rappresentato dalle ampiezze di uno stato quantistico $|\psi_x\rangle$ di n qubit; ciò implica un risparmio esponenziale in termini di "spazio" ma anche un costo esponenziale per la preparazione dello stato in termini di "tempo".

La rappresentazione del dataset è del tipo:

$$|\mathcal{D}_n\rangle = \sum_{i=1}^{2^n} \alpha_i |x_i\rangle . \tag{2.31}$$

con il vincolo che si abbia $\sum_{i=1}^{2^n} \alpha_i = 1$

Se per esempio si volesse codificare il vettore $x = (2.7, 0.0, 4.2, 1.7) \in \mathbb{R}^4$ sarebbero necessari 2 qubit:

$$|\psi_x\rangle = \frac{1}{\sqrt{27.82}} \big(2.7 |00\rangle + 4.2 |10\rangle + 1.7 |11\rangle \big) \tag{2.32}$$

dove $\frac{1}{\sqrt{27.82}}$ è il fattore di normalizzazione.

2.4.2 Misura

Com'è noto dalla meccanica quantistica, dato un sistema quantistico in uno stato di sovrapposizione, la misura di un'osservabile implica la *decoerenza* del sistema. La

decoerenza è un processo irreversibile per il quale si ha la perdita dello stato di sovrapposizione coerente di un sistema quantistico. In particolare, nella logica del quantum computing, la decoerenza rappresenta un problema quando insorge spontaneamente a causa delle interazioni del quantum computer con l'ambiente esterno, in quanto essa implica la perdita di gran parte dell'informazione codificata nello stato quantistico.

Dato un vettore di stato, esso si evolve nel tempo deterministicamente. Più precisamente, sono le ampiezze di probabilità che evolvono deterministicamente e non una caratteristica misurabile del sistema.

Una quantità osservabile, come l'energia o il momento di una particella quantistica, è associata a un operatore matematico M. L'osservabile M è hermitiano e può essere espresso come:

$$M = \sum_i \alpha_i P_i, \tag{2.33}$$

dove P_i è un operatore di proiezione nell'autospazio di M con autovalore α_i. In altre parole, un osservabile è una somma pesata di proiettori. I possibili risultati di una misura sono gli autovalori α_i reali. All'atto della misura, tutte le componenti della sovrapposizione svaniscono, eccetto una. La probabilità di osservare l'autovalore α_i è

$$\mathbf{P}(\alpha_i) = \langle\psi|P_i|\psi\rangle\,. \tag{2.34}$$

Il risultato di una misura è intrinsecamente probabilistico.

Tornando ai circuiti quantistici, la codifica viene effettuata da un circuito S_x a partire dallo stato iniziale $|\phi\rangle = |0\rangle^{\otimes n}$, dopodiché il circuito quantistico vero e proprio effettua una trasformazione unitaria U_θ. Infine si ha la misura e quindi il postprocessing dei risultati.

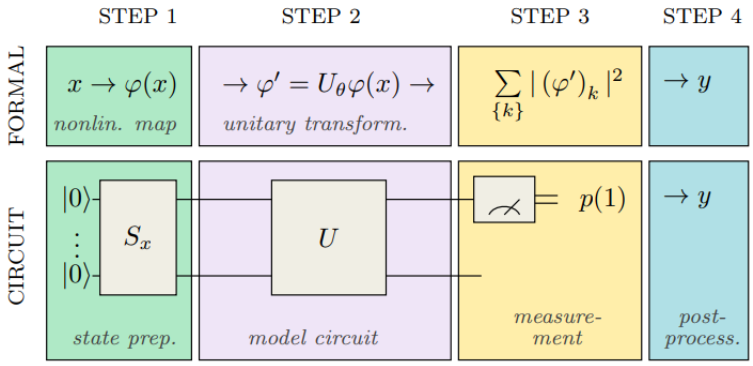


Figura 2.11: Schematizzazione delle principali fasi di un circuito quantistico generico (tratta da [15])

2.5 Algoritmi quantistici

2.5.1 Algoritmo di Shor

L'algoritmo di Shor fu ideato nel 1994 da Peter Shor per risolvere il problema della fattorizzazione in numeri primi. Fino a quel momento il quantum computing non aveva ricevuto particolari attenzioni se non da parte della ricerca accademica; con l'algoritmo di Shor e le sue implicazioni nel campo della crittoanalisi furono chiare le potenzialità del quantum computing anche da un punto di vista applicativo, potenzialità che suscitarono l'interesse dell'industria e di conseguenza una forte spinta alla ricerca.

La complessità del problema della fattorizzazione si basa su un'asimmetria della funzione $f(x) = a^x (\text{mod } N)$. Se calcolare $f(x)$ è facile se sono noti i parametri a e N, ricavare x a partire da $f(x)$ non è affatto banale se si considera inoltre che la funzione è periodica.

Il problema della fattorizzazione si può ricondurre al problema di trovare il periodo r della funzione, con N numero da fattorizzare e $a < N$ un intero coprimo ad N. Il teorema di Eulero per l'aritmetica modulare infatti afferma che se a ed N sono coprimi allora esiste $r > 0$ tale che:

$$a^r \equiv 1 (\text{mod } N). \tag{2.35}$$

r è proprio il periodo della funzione, infatti:

$$f(x+r) = a^{x+r} (\text{mod } n) = a^x a^r (\text{mod } n) = a^x (\text{mod } n) = f(x), \tag{2.36}$$

e se r è pari e $a^{\frac{r}{2}} \not\equiv 1(\text{mod } n)$ allora:

$$a^r - 1 \equiv 0(\text{mod } n) \rightarrow (a^{\frac{r}{2}})^2 - 1) \equiv 0(\text{mod } n), \tag{2.37}$$

da cui si ricava

$$(a^{\frac{r}{2}} - 1)(a^{\frac{r}{2}} + 1) \equiv 0(\text{mod } n). \tag{2.38}$$

Dunque il prodotto tra $(a^{\frac{r}{2}} - 1)$ e $(a^{\frac{r}{2}} + 1)$ è un multiplo di N, per cui i due termini contengono la scomposizione in fattori primi di N. Nel caso più semplice, $N = pq$ con p e q primi, si ha:

$$p = MCD(a^{\frac{r}{2}} - 1, N), \tag{2.39}$$

$$q = MCD(a^{\frac{r}{2}} + 1, N). \tag{2.40}$$

L'algoritmo di cifratura RSA sfrutta questa proprietà per generare una chiave pubblica (a, N) e una chiave privata (p, q); la sicurezza deriva dal fatto che trovare p e q per N sufficientemente grandi risulta un procedimento estremamente lungo per un normale computer. Con la disponibilità di un computer quantistico sufficientemente potente e l'utilizzo dell'algoritmo di Shor si avrebbe uno speedup esponenziale nella risoluzione di tale problema.

I passaggi che costituiscono l'algoritmo sono i seguenti:

1. Si stabilisce se il numero N da fattorizzare è o primo o l'elevamento a potenza di un numero primo: in tal caso esistono algoritmi classici efficienti per la fattorizzazione;

2. Si sceglie un intero h potenza di 2 tale che $N^2 < h < 2N^2$;

3. Si sceglie un numero x coprimo ad N;

4. Si creano due registri di memoria: il primo deve poter rappresentare gli interi da 0 a $h - 1$, il secondo gli interi da 0 a $N - 1$;

5. Nel primo registro si carica una sovrapposizione uniforme di tutti gli stati tra 0 e $h - 1$, il secondo registro è inizializzato con tutti 0

$$\frac{1}{\sqrt{h}} \sum_{x=0}^{h-1} |x, 0\rangle \, ; \tag{2.41}$$

6. Nel secondo registro si carica $f(x)$ per ogni x presente nel primo registro:

$$\frac{1}{\sqrt{h}} \sum_{x=0}^{h-1} |x, f(x)\rangle \,, \tag{2.42}$$

per farlo si sfrutta il parallelismo quantistico;

7. Si misura il secondo registro e si ottiene un valore k, quindi il primo registro collassa in un'equa sovrapposizione di tutti gli stati x' tra 0 e $h-1$ tali che $f(x') = k$:

$$\frac{1}{\sqrt{|X|}} \sum_{x' \in X} |x', k\rangle \,, \tag{2.43}$$

dove X è l'insieme di tutti gli x' tali che $f(x') = k$;

8. Si calcola la trasformata di Fourier quantistica del primo registro:

$$\frac{1}{\sqrt{|X|}} \sum_{x' \in X} \frac{1}{\sqrt{h}} \sum_{c=0}^{h-1} e^{\frac{2\pi i x' c}{h}} |c, k\rangle \,, \tag{2.44}$$

in questo modo si è massimizzata la probabilità di trovare nel primo registro un multiplo intero m di $\frac{h}{r}$, dove r è il periodo della funzione;

9. Si misura lo stato m del primo registro;

10. Si ricava il periodo r sapendo che

$$m = \lambda \frac{h}{r} \tag{2.45}$$

allora $\frac{m}{h} = \frac{\lambda}{r}$ con $\lambda \in \mathbb{Z}$, quindi se λ e r non hanno fattori in comune (cosa che per la teoria dei numeri accade con una probabilità $\geq \frac{1}{log(log(r))}$) si può ridurre la frazione ai minimi termini e ottenere r;

11. Trovato r si può procedere al calcolo dei due MCD p e q: se si ottiene un fattore non banale l'algoritmo si ferma, altrimenti si riparte dallo step 4.

I primi tre e gli ultimi due passaggi sono effettuati da un computer classico, mentre gli step dal 4 al 9 sono a carico del computer quantistico.

2.5.2 Algoritmo di Grover

L'algoritmo di Grover è un algoritmo quantistico che trova con elevata probabilità l'unico input per il quale una funzione $f(x)$, che si può considerare come una black

box, produce un particolare output. Se gli input x sono gli elementi di un database e la funzione $f(x)$ è tale che l'output sia, per esempio, "vero" per l'elemento desiderato e "falso" per tutti gli altri, allora l'algoritmo esegue la ricerca di un elemento in un database non strutturato. Sia il database costituito da N elementi, i relativi indici da 0 a $N-1$ sono esprimibili con $n = log_2 N$ bit. Sia $f(x)$ con $x \in [0, N-1]$ tale che $f(x) = 1$ se x è una soluzione al problema di ricerca e $f(x) = 0$ altrimenti. Si definisce *oracolo* l'operatore unitario O che agisce sulla base computazionale come:

$$|x\rangle |q\rangle \rightarrow |x\rangle |q \oplus f(x)\rangle , \tag{2.46}$$

dove $|q\rangle$ è il qubit oracolo e $|x\rangle$ l'indice del registro. Risulta utile avere il qubit oracolo inizialmente nello stato $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$ di modo che se x non è una soluzione lo stato del qubit oracolo resti invariato e se x è soluzione si ottenga $-\frac{|0\rangle - |1\rangle}{\sqrt{2}}$. L'azione dell'oracolo è allora:

$$|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \rightarrow (-1)^{f(x)} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.47}$$

dove lo stato del qubit oracolo resta invariato, quindi si può trascurare:

$$|x\rangle \rightarrow (-1)^{f(x)} |x\rangle . \tag{2.48}$$

Inizialmente ogni elemento della lista è un buon candidato, quindi si costruisce lo stato iniziale $|\psi\rangle$ tale che la probabilità sia la stessa per ogni indice. Si parte dallo stato $|0\rangle^{\otimes n}$ cui si applica la trasformata di Hadamard per ottenere una sovrapposizione uniforme:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle . \tag{2.49}$$

Successivamente si applica l'*operatore di Grover* ripetutamente. Tale operatore, indicato con G, consiste di quattro step:

1. applicazione dell'oracolo;
2. applicazione della trasformata di Hadamard $H^{\otimes n}$;
3. applicazione di uno shift di fase sullo stato eccetto per $|0\rangle$
   $|x\rangle \rightarrow -(-1)^{\delta_{x0}} |x\rangle$
4. nuova applicazione della trasformata di Hadamard $H^{\otimes n}$.

L'effetto degli step $2) - 4)$ è:

$$H^{\otimes n}(2 |0\rangle \langle 0| - I)) H^{\otimes n} = 2 |\psi\rangle \langle \psi| - I. \tag{2.50}$$

L'operatore di Grover è:

$$G = (2\,|\psi\rangle\,\langle\psi| - I)O. \tag{2.51}$$

Nel caso classico la complessità temporale è $O(N)$, mentre l'algoritmo di Grover mostra uno speedup quadratico con una complessità temporale $O(\sqrt{N}$.

2.5.3 Swap test

Lo *swap test* è un algoritmo quantistico, spesso presente come subroutine in algoritmi più complessi, che serve a stimare il valore assoluto del prodotto interno tra due stati quantistici. Il circuito che implementa tale procedura è il seguente:

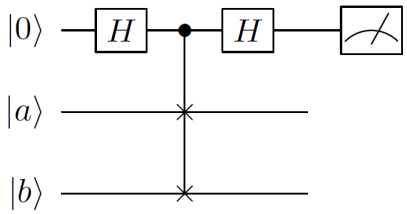


Figura 2.12: Circuito che esegue lo swap test.

Gli step della procedura sono i seguenti:

Inizializzazione : si inizializzano due stati $|a\rangle$ e $|b\rangle$ più un qubit di controllo $|0\rangle$.

$$|\psi_0\rangle = |0, a, b\rangle\,. \tag{2.52}$$

I due stati $|a\rangle$ e $|b\rangle$ descrivono n qubit ognuno.

Porta di Hadamard : si applica una porta di Hadamard al bit di controllo e si ottiene la seguente sovrapposizione

$$|\psi_1\rangle = (H \otimes I^{\otimes n} \otimes I^{\otimes n})\,|\psi_0\rangle = \frac{1}{\sqrt{2}}\big(\,|0, a, b\rangle + |1, a, b\rangle\,\big). \tag{2.53}$$

Porta SWAP : si applica la porta SWAP allo stato $|\psi_1\rangle$ e si ottiene lo stato

$$|\psi_2\rangle = \frac{1}{\sqrt{2}}\big(\,|0, a, b\rangle + |1, b, a\rangle\,\big). \tag{2.54}$$

Porta Hadamard : si applica nuovamente una porta Hadamard sul qubit di controllo e si ottiene

$$|\psi_3\rangle = \frac{1}{\sqrt{2}}\,|0\rangle\,\big(\,|a, b\rangle + |b, a\rangle\,\big) + \frac{1}{\sqrt{2}}\,|1\rangle\,\big(\,|a, b\rangle - |b, a\rangle\,\big). \tag{2.55}$$

Misura : si misura il qubit di controllo. La probabilità di trovare il qubit di controllo nello stato $|0\rangle$ è data da:

$$\begin{aligned} P(|0\rangle) &= \left| \frac{1}{2} \langle 0|0\rangle \left(|a,b\rangle + |b,a\rangle \right) + \frac{1}{2} \langle 0|1\rangle \left(|a,b\rangle - |b,a\rangle \right) \right|^2 \\ &= \frac{1}{4} \left| \left(|a,b\rangle + |b,a\rangle \right) \right|^2 \\ &= \frac{1}{4} \left(\langle b|b\rangle \langle a|a\rangle + \langle b|a\rangle \langle a|b\rangle + \langle a|b\rangle \langle b|a\rangle + \langle a|a\rangle \langle b|b\rangle \right) \\ &= \frac{1}{2} + \frac{1}{2} |\langle a|b\rangle|^2. \end{aligned} \tag{2.56}$$

In tal modo si può stimare la sovrapposizione tra due stati in termini della probabilità di trovare il qubit di controllo nello stato $|0\rangle$. Una probabilità $P(|0\rangle = \frac{1}{2}$ indica che i due stati sono ortogonali, mentre una probabilità $P(|1\rangle) = 1$ indica che i due stati sono identici. É chiaro che per la natura stessa, intrinsecamente probabilistica, dei fenomeni quantistici è necessario ripetere la funzione più volte perché la stima sia attendibile. Dal momento che la misura si effettua unicamente sul qubit di controllo non è necessario conoscere i due stati $|a\rangle$ e $|b\rangle$ a priori.

Capitolo 3

Quantum machine learning

3.1 Machine learning

Il *machine learning*, o *apprendimento automatico*, è una branca dell'intelligenza artificiale che fa riferimento a un vasto insieme di algoritmi. Per l'uomo, *apprendimento* significa trovare dei pattern significativi sulla base dell'esperienza pregressa che possano tornare utili nella gestione di situazioni non note; quando si parla di computer, l'esperienza si traduce in *dati*. Lo scopo principale del machine learning è quello di addestrare i computer a trovare e usare pattern nei dati, senza forzare alcun modello su di essi: l'approccio si dice perciò *data-driven*.

Lo *spazio delle caratteristiche*, o *feature space*, è una rappresentazione matematica dei dati sotto esame e costituisce il cuore degli algoritmi di apprendimento. La ricerca di pattern nello spazio delle caratteristiche può procedere sulla base di modelli statistici o sulla base della teoria dell'apprendimento algoritmico. Quest'ultima presenta il vantaggio di fare a meno di qualsiasi tipo di assunzione statistica, quindi permette una maggiore libertà nell'analisi di dataset reali, spesso molto rumorosi e dove la distribuzione non è nota.

A prescindere dall'approccio, il machine learning si divide in tre categorie principali:

- *Apprendimento supervisionato*: l'algoritmo sfrutta un dataset di addestramento, quindi dei dati già etichettati per cui le coppie input-output sono prestabilite,

sulla base del quale costruisce un modello che possa adattarsi a nuovi dati non classificati;

- *Apprendimento non supervisionato*: in questo caso non si sfrutta alcun dataset di addestramento. L'algoritmo ha il compito di trovare una struttura nei dati secondo caratteristiche che ricava dai dati stessi;
- *Apprendimento per rinforzo*: lo scopo è quello di creare un *agente* che attraverso le interazioni con l'ambiente sia in grado di migliorare le proprie performance. Il comportamento dell'agente è determinato da una routine di apprendimento basata su ricompensa e punizione.

In ogni caso, a prescindere dall'approccio statistico o non statistico, supervisionato o non supervisionato, ciò che è desiderabile per un modello di apprendimento è una buona *generalizzazione*. Per *generalizzazione* si intende l'adattabilità di un algoritmo a dati, esempi, compiti nuovi che non erano parte della fase di addestramento.

3.2 Reti neurali

Una rete neurale artificiale è costituita da unità, dette *neuroni*, comunicanti tramite *connessioni sinaptiche* pesate. Ogni unità riceve un input a seconda delle connessioni esistenti ed elabora una risposta in base alla *funzione di attivazione* che implementa.

Le singole unità sono componenti elementari in grado di svolgere operazioni semplici come esecuzioni di medie pesate e decisioni di soglia. La capacità di eseguire compiti complessi emerge dalla cooperazione delle molte unità interconnesse e operanti in parallelo.

La configurazione più semplice è quella in cui si ha un singolo neurone, detto *percettrone*. In generale, i molti neuroni che costituiscono una rete neurale sono dislocati in livelli:

- Livello di input;
- Livello nascosto (spesso più di uno);
- Livello di output.

Nelle reti neurali *feed-forward* il segnale si diffonde attraverso la rete nella direzione dalle unità di input a quelle di output. Nelle reti neurali *ricorrenti* il segnale circola nella rete fino al raggiugimento di una condizione stabile.

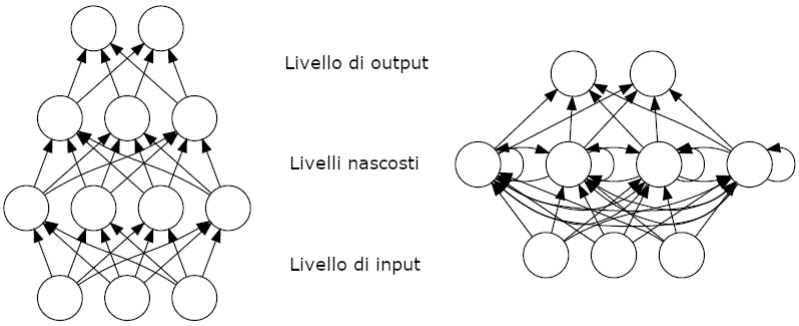


Figura 3.1: A sinistra una rete neurale feed-forward; a destra una rete neurale ricorrente.

La varietà di tipologie di reti neurali esistenti è estremamente ampia; di seguito si introdurrà il percettrone, dopodichè saranno introdotte le reti neurali ricorrenti e le reti neurali a grafo.

In ogni caso, il processo di apprendimento consiste nel rafforzamento o indebolimento selettivo delle connessioni, quindi nell'aggiornamento dei pesi delle connessioni, il quale si basa sulla minimizzazione di una *funzione di costo*.

3.2.1 Percettrone

Come si è detto, il percettrone è il caso piu semplice di rete neurale. Il modello funziona da classificatore binario facendo corrispondere a un input $\mathbf{x} \in \mathbb{R}^n$ un output binario prodotto da una funzione di attivazione:

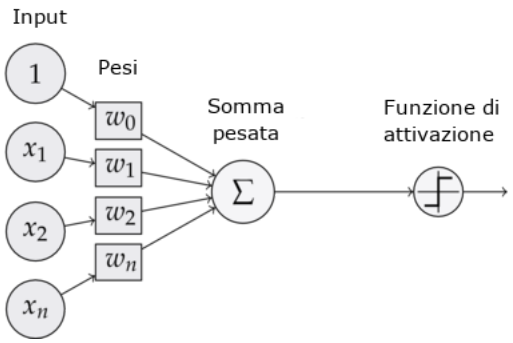


Figura 3.2: Per

In questo caso la funzione di attivazione è la funzione di Heaviside, dunque:

$$f(x) = \begin{cases} 1 & \text{se } \mathbf{w}^\mathsf{T}\mathbf{x} + b > 0 \\ 0 & \text{altrimenti} \end{cases} \tag{3.1}$$

dove $\mathbf{w} = (w_1, \ldots, w_n)$ è il vettore dei pesi e $b = w_0$ il termine di bias, utile per riaggiustare la soglia in fase di addestramento.

Il percettrone presenta una importante limitazione. Il valore di output è 0 o 1, quindi la scelta dei pesi individua un iperpiano a $n-1$ dimensioni nello spazio n-dimensionale dei valori di input. Per questo motivo il percettrone è utile unicamente nella risoluzione di problemi linearmente separabili, caso in cui la convergenza è assicurata.

Per la fase di training si sfrutta un set di dati per i quali i risultati y_i sono noti e si inizializzano i pesi e la soglia in modo casuale. La procedura di aggiornamento dei pesi è nota come *delta rule* e si basa sulla discesa del gradiente (sezione 3.4). Si definisce il termine E come la somma degli errori al quadrato:

$$E = \frac{1}{2}\sum_{i=1}^{n} \big(y_i - f(\mathbf{x}_i)\big)^2. \tag{3.2}$$

Si minimizza E calcolando la derivata parziale rispetto a w_j:

$$\frac{\partial E}{\partial w_j} = -\big(y_i - f(\mathbf{x}_i)\big)x_j \tag{3.3}$$

e aggiornando $\mathbf{w}$ in maniera proporzionale:

$$\Delta w_j = \eta(y_i - f(\mathbf{x_i})x_j, \tag{3.4}$$

dove η è il learning rate.

3.2.2 Reti neurali ricorrenti

In una rete neurale ricorrente i neuroni ammettono dei loop e possono essere connessi ai neuroni dello stesso livello, di un livello successivo o precedente. La ricorrenza dota la rete neurale quindi di una sorta di memoria; infatti l'output di un neurone può influenzare il neurone stesso ad un istante temporale successivo, così come può influenzare i neuroni di un altro livello che a loro volta faranno altrettanto. Per questo motivo l'uso delle RNN è particolarmente adatto nell'analisi di sequenze temporali, più in generale nei casi in cui i dati sono di tipo dinamico.

Si consideri una sequenza $\mathbf{x}$ di lunghezza T e una RNN con I unità di input, H unità nascoste e K unità di output. Sia x_i^t il valore dell'input i all'istante t, siano a_h^t e b_h^t rispettivamente il valore in input e in output dell'unità h all'istante t:

$$a_h^t = \sum_{i=1}^{I} w_{ih} x^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}, \tag{3.5}$$

$$b_h^t = f_h(a_h^t). \tag{3.6}$$

dove f è la funzione di attivazione dell'unità h. Analogamente è possibile calcolare il valore in input dalla rete all'unità di output k:

$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t. \tag{3.7}$$

Una utile rappresentazione visiva di una RNN è la seguente:

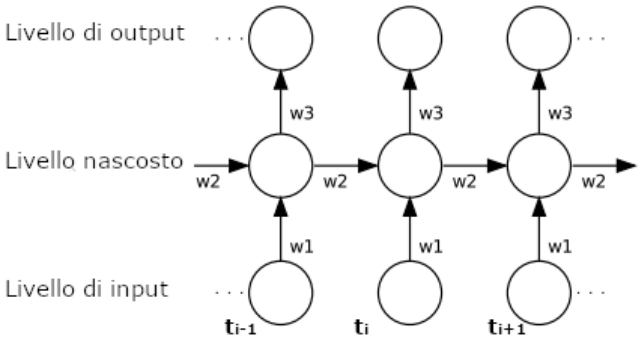


Figura 3.3: Rappresentazione unfolded della RNN: ogni nodo rappresenta un livello ad un certo istante.

Per quanto riguarda il processo di apprendimento, quindi l'aggiornamento dei pesi, uno degli algoritmi più diffusi è quello di retropropagazione nel tempo (BPTT da BackPropagation through time).

Sia $\mathcal{L}$ la funzione di costo, si vuole calcolare

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial a_j^t} b_i^t. \tag{3.8}$$

In questo caso la funzione di costo $\mathcal{L}$ dipende dall'attivazione del livello nascosto per l'influenza che esso esercita sia sul livello di output sia sul livello nascosto all'istante successivo. Dunque

$$\frac{\partial \mathcal{L}}{\partial a_h^t} = \frac{\partial f_h}{\partial a_h^t} \left(\sum_{k=1}^{K} \frac{\partial \mathcal{L}}{a_k^t} w_{hk} + \sum_{h'=1}^{H} \frac{\partial \mathcal{L}}{\partial a_{h'}^{t+1}} w_{hh'} \right). \tag{3.9}$$

Nota la quantità $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ si può procedere all'update dei pesi secondo diverse modalità (sezione 3.4). Nel caso più semplice si ha:

$$\Delta \mathbf{w}^n = -\eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}^n}, \tag{3.10}$$

dove $\Delta \mathbf{w}^n$ è l'n-esimo update, $\mathbf{w}^n$ è il vettore dei pesi prima che sia applicata la correzione $\Delta \mathbf{w}^n$ e η come al solito è il learning rate.

3.2.3 Graph neural networks

Un *grafo* è una struttura matematica costituita da *nodi* e *archi*. In generale, un grafo è definito come una coppia di insiemi $\mathcal{G} = \{H, E\}$, dove H è l'insieme di nodi ed E l'insieme di archi, tali che gli elementi di E siano coppie di elementi di H.

Grazie alla loro potenza espressiva, i grafi rappresentano spesso la modellizzazione più naturale per molti sistemi o fenomeni. Spesso la traduzione dei dati contenuti in un grafo in una forma che possa essere fruibile per un sistema di apprendimento automatico, oltre a essere tutt'altro che banale, implica la perdita di parte delle informazioni. Le reti neurali a grafo (o GNN da graph neural network) sono pensate per operare naturalmente su dati sottoforma di grafi.

Per meglio visualizzare una GNN si può immaginare la struttura di una RNN come un grafo diretto, cioè orientato, eliminando però le unità di input e le connessioni pesate, che saranno rispettivamente incorporate nello stato dei nodi e negli archi. Allo stesso modo si eliminino le unità di output. Infatti, in una GNN l'input è lo stato iniziale dei nodi e l'output è lo stato finale, successivo al processo di apprendimento.

In una GNN, ad ogni nodo v è associato un *embedding*, cioè un vettore di stato $\mathbf{h}_v \in \mathbb{R}^s$ ottenuto tramite una funzione f, detta di *transizione locale*, che aggrega sia gli stati dei nodi vicini che lo stato precedente dello stesso nodo v. Inizialmente, ogni nodo ha delle caratteristiche $\mathbf{x}_v$. Il vettore h_v è definito come

$$\mathbf{h}_v = f(\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{h}_{ne[v]}, \mathbf{x}_{ne[v]}), \tag{3.11}$$

dove $\mathbf{x}_{ne[v]}$ sono le caratteristiche dei nodi vicini, $\mathbf{x}_{co[v]}$ le caratteristiche degli archi connessi a v, $\mathbf{h}_{ne[v]}$ gli stati dei nodi vicini. L'output invece è definito dalla funzione g, detta di *output locale*:

$$\mathbf{o}_v = g(\mathbf{h}_v, \mathbf{x}_v). \tag{3.12}$$

Siano $\mathbf{H}, \mathbf{O}, \mathbf{X}$ e $\mathbf{X}_N$ i vettori formati raggruppando tutti gli stati, gli output, le caratteristiche degli archi e dei nodi rispettivamente. Allora in una forma più compatta

si ha:

$$\mathbf{H} = F(\mathbf{H}, \mathbf{X}), \tag{3.13}$$

$$\mathbf{O} = G(\mathbf{H}, \mathbf{X}_N), \tag{3.14}$$

dove F è la *funzione di transizione globale* e G la *funzione di output globale*. Il valore di $\mathbf{H}$ è il punto fisso dell'Eq. (3.13), univocamente definito con l'assunzione che F sia una contrazione. Lo schema iterativo per il calcolo degli stati è il seguente:

$$\mathbf{H^{t+1}} = F(\mathbf{H}^t, \mathbf{X}) \tag{3.15}$$

dove l'apice indica l'iterazione. Il sistema (3.15) converge alla soluzione dell'Eq. (3.13) per qualsiasi valore iniziale $\mathbf{H}(0)$. Per quanto riguarda l'addestramento, sia $\mathbf{t}_v$ il risultato target per il nodo v. Si può definire la funzione di costo come:

$$\mathcal{L} = \sum_{i=1}^{p} (\mathbf{t}_i - \mathbf{o}_i), \tag{3.16}$$

dove p è il numero di nodi soggetti a supervisione. Infine, l'algoritmo di apprendimento può essere basato sul metodo della discesa del gradiente (sez. 3.4).

3.3 Quantum machine learning

Il *quantum machine learning* è la branca che nasce dall'incontro tra quantum computing e machine learning. In generale, l'idea è quella di investigare le potenzialità della sinergia tra le due discipline; più precisamente, quando si intende sfruttare l'efficienza del quantum computing per la risoluzione di problemi tipici del machine learning si parla di *quantum enhanced machine learning*. L'espressione quantum-*enhanced* suggerisce che gli algoritmi quantistici siano in qualche modo più performanti rispetto alla loro controparte classica. Ciò, da un punto di vista teorico, è indiscutibilmente vero e dimostrato da diversi algoritmi: l'algoritmo di Shor e l'algoritmo di Grover visti in precedenza sono gli esempi più noti. Da un punto di vista pratico le cose sono ben diverse: le difficoltà che si incontrano nella realizzazione fisica di un computer quantistico sono molteplici e, sebbene lo sforzo e i progressi fatti negli ultimi anni, si è ancora lontani dalla realizzazione di macchine quantistiche scalabili che possano implementare algoritmi come i due appena citati.

3.3.1 Problematiche sperimentali

Molti dei risultati teorici ottenuti negli ultimi decenni si basano su dispositivi quantistici ideali. Il termine *NISQ* (*Noisy Intermediate-Scale Quantum*), coniato dal fisico teorico Jonh Preskill, descrive i dispositivi quantistici disponibili oggi e nel prossimo futuro: *Noisy* si riferisce al rumore intrinseco da cui sono affetti i qubit, responsabile della decoerenza; *Intermediate-Scale* fa riferimento alla dimensione in termini di qubit, quindi dai 50 alle poche centinaia.

Perché il tempo di decoerenza sia quanto più lungo possibile, è necessario isolare il computer quantistico tenendolo a temperature prossime allo zero assoluto. Inoltre, maggiore è il numero di qubit più è complicato preservare la coerenza del sistema. Ci si aspetta di poter preservare l'informazione tramite il principio della correzione quantistica dell'errore: l'idea è quella di codificare l'informazione in uno stato altamente entangled. La codifica dell'informazione in uno stato altamente entangled richiede però un gran numero di qubit fisici ausiliari, motivo per cui è prevedibile che i primi computer quantistici basati sul quantum error correction non saranno disponibili presto. Molte tecniche di error correction in uso nei computer classici non possono essere sfruttate nei computer quantistici per via dell'impossibilità di creare delle copie esatte di uno stato quantistico sconosciuto, come asserisce il *teorema di no-cloning quantistico*.

Altre limitazioni riguardano la qualità dei qubit, la connettività tra essi, la scalabilità.

3.3.2 Hybrid quantum-classical computing

Per quanto detto, è chiaro che i NISQ computer, da soli, non sono adatti per la risoluzione di problemi oltre un certo livello di complessità. L'ibridizzazione di algoritmi quantistici e classici risulta essere una soluzione promettente che consente di sfruttare il potenziale degli hardware quantistici attualmente disponibili. Tali algoritmi ibridi sfruttano computer quantistici e classici nel tentativo di trarre il meglio da entrambi: dei primi viene sfruttata la potenza del calcolo quantistico, mentre i secondi intervengono per risolvere le limitazioni proprie dei NISQ computer. Per esempio, i computer classici dispongono di memorie sufficientemente grandi, tali da poter immagazzinare l'intero problema, mentre gli hardware quantistici risultano spesso più performanti nella risoluzione di specifici compiti.

In generale, l'idea è quella di scomporre un problema in parti, risolvere i sottoproblemi dispendiosi dal punto di vista computazionale sui quantum computer e ricombinare le singole soluzioni sul computer classico per ottenere una soluzione globale.

3.3.3 Circuiti quantistici variazionali

I *circuiti variazionali* sono circuiti parametrizzati, costituiti quindi da porte logiche fisse, come le porte `CNOT`, e da porte logiche che dipendono da parametri, come le porte di rotazione.

La strategia più diffusa nell'ambito del machine learning consiste nel formulare un qualsiasi problema come un problema di ottimizzazione con approccio variazionale. In generale, secondo il paradigma basato sull'approccio ibrido, gli step fondamentali sono:

- il CC si occupa del pre-processing e di comunicare i dati al QC;
- il QC riceve i dati classici, li codifica in stati quantistici sui quali vengono effettuate delle operazioni a seconda della struttura del circuito, effettua delle misure di cui comunica i risultati al CC;
- il CC si occupa del post-processing e di aggiornare i parametri del circuito variazionale affinché venga minimizzata una opportuna funzione di costo. I nuovi parametri sono passati nuovamente al QC.

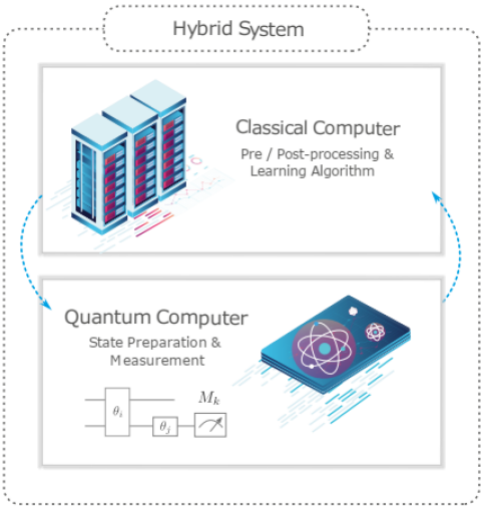


Figura 3.4: Schema modello ibrido (tratto da [4])

3.3.4 Quantum Graph Neural Networks

Si consideri un grafo $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, dove $\mathcal{V}$ è l'insieme dei vertici e $\mathcal{E}$ l'insieme degli archi. É possibile assegnare a ciascun vertice un sottosistema quantistico associato allo spazio di Hilbert $\mathcal{H}_v$; lo spazio di Hilbert globale sarà quindi $\mathcal{H}_\mathcal{V} = \bigotimes_{v\in\mathcal{V}} \mathcal{H}_v$. Ognuno dei sottosistemi può essere per esempio uno o più qubit. Gli archi del grafo indicano le interazioni tra i sottosistemi. Una configurazione di questo tipo è una *rete quantistica*, la cui topologia è data dal grafo.

L'ansatz più generale per una QGNN non è altro che un circuito variazionale su una rete costituita da una sequenza di Q evolutori temporali diversi, relativi alle hamiltoniane H^q, con l'intera sequenza ripetuta P volte:

$$\hat{U}_{QGNN}(\boldsymbol{\eta}, \boldsymbol{\theta}) = \prod_{p=1}^{P} \left[\prod_{q=1}^{Q} e^{-i\eta_{pq}\hat{H}^q(\boldsymbol{\theta})} \right], \tag{3.17}$$

dove il prodotto è temporalmente ordinato, $\boldsymbol{\eta}$ e $\boldsymbol{\theta}$ sono parametri variazionali; in particolare $\boldsymbol{\eta}_{pq}$ è il parametro temporale. Le hamiltoniane $\hat{H}^q(\boldsymbol{\theta})$ possono essere qualsiasi hamiltoniana parametrizzata la cui topologia delle interazioni corrisponde a quella del grafico:

$$\hat{H}^q(\boldsymbol{\theta}) \equiv \sum_{\{j,k\}\in\mathcal{E}} \sum_{r\in\mathcal{I}_{jk}} W_{qrjk} \hat{O}_j^{(qr)} \otimes \hat{P}_k^{(qr)} + \sum_{v\in\mathcal{V}} \sum_{r\in\mathcal{J}_v} B_{qrv} \hat{R}_j^{(qv)}, \tag{3.18}$$

dove W_{qrjk} e B_{qrv} sono parametri che formano la collezione $\boldsymbol{\theta} \equiv \cup_{q,r,j,k}\{W_{qrjk}\} \cup_{q,r,v} \{B_{qrv}\}$. Gli operatori $\hat{O}_j^{(qr)}$, $\hat{P}_j^{(qr)}$ e $\hat{R}_j^{(qv)}$ sono operatori hermitiani che agiscono sullo spazio di Hilbert $\mathcal{H}_j$ associato al j-esimo nodo del grafo. Infine, $\mathcal{I}_{jk}$ e $\mathcal{J}_v$ sono gli insiemi degli indici per i termini corrispondenti agli archi e ai nodi, rispettivamente.

Quantum Graph Recurrent Neural Network

Nel caso di una QGNN ricorrente (QGRNN) l'ansatz (3.17) resta pressoché invariato; l'unica differenza riguarda i parametri temporali, i quali restano invariati per ogni iterazione della sequenza, proprio come i pesi durante l'elaborazione di una sequenza in una RNN, perciò $\eta_{pq} \to \eta_q$:

$$\hat{U}_{QGRNN}(\boldsymbol{\eta}, \boldsymbol{\theta}) = \prod_{p=1}^{P} \left[\prod_{q=1}^{Q} e^{-i\eta_q \hat{H}^q(\boldsymbol{\theta})} \right]. \tag{3.19}$$

3.4 Algoritmi di ottimizzazione

In questa sezione si introducono diversi algoritmi che torneranno utili in seguito: *Adam* è un algoritmo classico di ottimizzazione ampiamente sfruttato per l'update dei parametri nelle reti neurali; il *VQE* è un algoritmo classico-quantistico utilizzato per trovare lo stato fondamentale di un sistema quantistico.

3.4.1 Adam optimizer

Il concetto di ottimizzazione è un punto fondamentale quando si parla di machine learning e di reti neurali in particolare. Ricapitolando, lo scopo principale degli algoritmi di machine learning è quello di costruire un modello di ottimizzazione che sia in grado di minimizzare una *funzione obiettivo* $J(\theta)$, anche detta *funzione di costo*, tramite l'iterazione di un processo. I metodi di ottimizzazione del primo ordine, tra cui Adam, si basano sul concetto di discesa del gradiente.

Discesa del gradiente

Il metodo della discesa del gradiente si basa sull'aggiornamento iterativo del parametro θ lungo la direzione opposta a quella del gradiente della funzione di costo $J(\theta)$. La

regola per l'aggiornamento del parametro è:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta)$$

dove η è il learning rate che determina l'ampiezza della variazione di θ. Nonostante la semplicità, il metodo presenta diversi svantaggi:

- risulta essere molto dispendioso dal punto di vista computazionale in quanto per il calcolo del gradiente viene impiegato l'intero dataset;
- nel caso di funzioni non convesse la soluzione trovata potrebbe essere un minimo locale;
- la scelta del parametro η non è banale.

Discesa del gradiente stocastico

Il metodo della discesa del gradiente stocastico non sfrutta l'intero dataset per il calcolo del gradiente bensì un singolo campione casualmente scelto ad ogni iterazione:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

In tal modo la complessità computazionale viene drasticamente ridotta.

La scelta di un singolo campione dai dati può determinare una forte oscillazione della direzione del gradiente. Tale problema è risolto con la variante *mini batch* del metodo della discesa del gradiente, la quale non sfrutta un singolo campione ma un insieme di n campioni:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}).$$

In tal modo si riduce la varianza del gradiente e si stabilizza la convergenza.

Metodo del momento

Il processo di aggiornamento di θ basato sui metodi appena illustrati risulta spesso particolarmente lento. Il concetto di *momento* nasce e viene introdotto proprio allo scopo di velocizzare tale processo. Molto brevemente, l'idea è quella di calcolare, ad ogni iterazione, una media mobile esponenziale dei gradienti storici e di sfruttare tale valore come direzione da seguire nell'aggiornamento di θ. La legge per l'aggiornamento

è:

$$v_t = \beta v_{t-1} - \eta \nabla_\theta J(\theta)$$
$$\theta_{t+1} = \theta_t + v_t$$

dove v è il momento e $\beta \in [0, 1)$ determina la velocità di decadimento dei contributi dei gradienti storici.

Metodi di learning rate adattivo

É evidente quanto una opportuna regolazione del learning rate sia determinante sull'efficacia dei metodi appena visti.

Il metodo *AdaGrad* implementa una regolazione dinamica del learning rate, il quale non risulta quindi costante durante il processo di aggiornamento di θ ma varia ad ogni iterazione sulla base dei valori storici del gradiente:

$$g_t = \nabla_\theta J(\theta),$$
$$v_t = \sqrt{\sum_{i=1}^{t} (g_i)^2 + \epsilon},$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{v_t} g_t.$$

Nel caso di training molto lunghi si può osservare che il learning rate tende a zero a causa di v_t, quindi il parametro θ può tendere a un valore stazionario non corretto. Tale problema è risolto con il metodo *RMSProp*: l'algoritmo utilizza una finestra temporale dei gradienti storici e ne calcola ad ogni iterazione il momento cumulativo del secondo ordine:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2.$$

Infine, il metodo *Adam* (ADAptive Moment estimation), introduce un'ulteriore miglioramento. Oltre a sfruttare la media mobile esponenziale dei quadrati dei gradienti v_t sfrutta anche la media mobile esponenziale dei gradienti m_t:

$$g_t = \nabla_\theta J(\theta),$$
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$$
$$\eta_t = \eta_{t-1} \frac{\sqrt{1-\beta_2}}{1-\beta_1},$$
$$\theta_{t+1} = \theta_t - \eta_t \frac{m_t}{\sqrt{v_t} + \epsilon}.$$

dove β_1 , $\beta_2 \in [0, 1)$ e m_t, v_t sono inizialmente 0.

3.4.2 Variational Quantum Eigensolver

Il VQE è un algoritmo ibrido in grado di trovare il minimo autovalore di una matrice hermitiana, quindi lo stato energetico fondamentale di un sistema quantistico. Senza scendere nei dettagli matematici, il processo quantistico è strutturato come di seguito:

- Si prepara l'autostato iniziale $|\psi_0(\eta)\rangle$;
- Si misura il valore di aspettazione; $\langle\psi_0(\eta)| H |\psi_0(\eta)\rangle$
- Si ripete la misura del valore di aspettazione facendo variare lo stato a poco a poco;
- Si sceglie il minimo valore di aspettazione $\langle\psi_{min}(\eta)| H |\psi_{min}(\eta)\rangle$.

Il parametro η è controllato nel processo classico. Per il principio variazionale il valore di aspettazione $\langle\psi_0(\eta)| H |\psi_0(\eta)\rangle$ è sempre maggiore o uguale del minimo autovalore, quindi $\langle\psi_{min}(\eta)| H |\psi_{min}(\eta)\rangle$ approssima il valore energetico dello stato fondamentale.

Capitolo 4

Implementazione QGRNN

4.1 Modello di Ising

Il modello di Ising è un modello studiato in meccanica statistica, nato in principio allo scopo di descrivere il magnetismo nei materiali. La modellizzazione vede gli spin come nodi di un reticolo, ognuno dei quali interagisce unicamente coi primi vicini. In questa sezione si introduce brevemente il modello di Ising unidimensionale in campo trasverso. L'hamiltoniana che descrive tale modello è la seguente:

$$\hat{H}_{\text{Ising}} = -\sum_{(i,j)\in E} J_{ij} Z_i Z_j - \mu \sum_i h_i Z_i - \mu \sum_i g_i X_i \tag{4.1}$$

L'insieme E contiene le sole coppie di primi vicini. Gli operatori Z e X nel caso di spin $\frac{1}{2}$ sono le matrici di Pauli σ^z e σ^x. Il termine J_{ij} ha dimensioni di un'energia e indica l'interazione tra primi vicini. Il termine μ è il momento magnetico, h_i e g_i indicano rispettivamente il campo magnetico esterno e il campo magnetico esterno trasverso interagenti col nodo i-esimo.

Tramite una semplice manipolazione dei coefficienti si può ottenere la seguente hamiltoniana:

$$\hat{H}_{\text{Ising}}(\boldsymbol{\theta}) = \sum_{(i,j)\in E} \theta_{ij}^{(1)} Z_i Z_j + \sum_i \theta_i^{(2)} Z_i + \sum_i X_i \tag{4.2}$$

dove $\boldsymbol{\theta} = \{\theta^{(1)}, \theta^{(2)}\}$.

Una hamiltoniana in questa forma si presta naturalmente ad essere descritta con un approccio tramite QGRNN:

- i qubit sono rappresentati dai nodi e le interazioni tra essi dagli archi del grafo;
- i parametri $\theta^{(1)}$ corrispondono ai pesi delle connessioni e i parametri $\theta^{(2)}$ corrispondono ai pesi dei singoli qubit.

Si ricordi che l'operatore di evoluzione temporale, essendo l'hamiltoniana indipendente dal tempo, può essere scritto come

$$\hat{U}_{\text{Ising}} = e^{-it\hat{H}_{\text{Ising}}(\boldsymbol{\theta})} = \exp\left(\sum_{(i,j)\in E} \theta_{ij}^{(1)} Z_i Z_j + \sum_i \theta_i^{(2)} Z_i + \sum_i X_i \right) \tag{4.3}$$

In generale, questo tipo di operazione è difficile da implementare in un circuito quantistico. Si può ricorrere alternativamente alla *decomposizione Trotter-Suzuki* che restituisce la seguente approssimazione:

$$\exp\left(\sum_{(i,j)\in E} \theta_{ij}^{(1)} Z_i Z_j + \sum_i \theta_i^{(2)} Z_i + \sum_i X_i \right) \approx \prod_{k=1}^{t/\Delta} \left[\prod_{j=1}^{Q} e^{-it\hat{H}_{\text{Ising}}^{j}(\boldsymbol{\theta})} \right] \tag{4.4}$$

dove $\hat{H}_{\text{Ising}}^{j}(\boldsymbol{\theta})$ è il termine j-esimo dell'hamiltoniana di Ising e $\Delta << 1$ il passo dell'approssimazione di Trotter-Suzuki.

Si ricordi che in generale l'approccio variazionale tramite QGRNN si presenta nella forma

$$\hat{U}_{QGRNN}(\boldsymbol{\eta}, \boldsymbol{\theta}) = \prod_{p=1}^{P} \left[\prod_{q=1}^{Q} e^{-i\eta_q \hat{H}^q(\boldsymbol{\theta})} \right], \tag{4.5}$$

Dal momento che i due approcci risultano equivalenti, è evidente che si può sfruttare una QGRNN per studiare la dinamica del sistema.

4.2 QGRNN per l'apprendimento della dinamica di un sistema quantistico

Si supponga di avere un sistema descritto da $\hat{H}_{\text{Ising}}(\boldsymbol{\alpha})$ con $\boldsymbol{\alpha}$ set di parametri target non noti e $\mathcal{G}$ grafico d'interazione anch'esso non noto. Si supponga inoltre di avere accesso a delle copie di uno stato a bassa energia $|\psi_0\rangle$, che non sia lo stato fondamentale, dalle quali si possono ottenere degli stati evoluti a diversi istanti di tempo $\{|\psi(t_1)\rangle, |\psi(t_2)\rangle, \ldots, |\psi(t_k)\rangle\}$ definiti come:

$$|\psi(t_k)\rangle = e^{-it_k \hat{H}_{\text{Ising}}(\boldsymbol{\alpha})} |\psi_0\rangle \,. \tag{4.6}$$

Insieme, la collezione di stati evoluti e lo stato $|\psi_0\rangle$, costituiscono i dati quantistici utili all'addestramento della rete. Per ogni stato evoluto all'istante t_k casualmente scelto dalla collezione lo si compara con

$$U_{QGRNN}(\boldsymbol{\mu}, \boldsymbol{\Delta}) |\psi_0\rangle \approx e^{-it_k \hat{H}(\boldsymbol{\mu})} |\psi_0\rangle \,. \tag{4.7}$$

Ciò viene realizzato inviando una delle copie di $|\psi_0\rangle$ alla QGRNN con un set di parametri $\boldsymbol{\mu}$ e un grafico di interazione $\mathcal{G}'$ casuali, dopodiché si sfrutta un algoritmo di ottimizzazione classico per massimizzare la somiglianza tra gli stati evoluti e gli stati in uscita dalla QGRNN.

Come gli stati preparati dalla QGRNN divengono sempre più simili agli stati evoluti presi dai dati, così i parametri $\boldsymbol{\mu}$ tendono ai parametri target $\boldsymbol{\alpha}$ che si vuole conoscere.

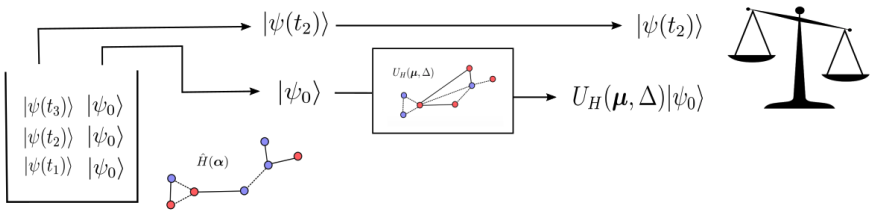
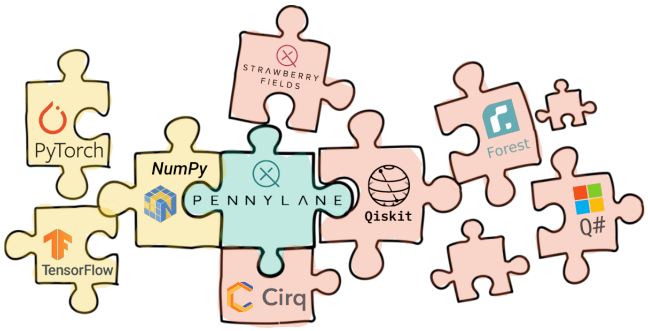


Figura 4.1: Rappresentazione di una singola esecuzione per un singolo stato.

4.3 Pennylane

Pennylane è un framework open-source per il calcolo quantistico ibrido che fornisce un'interfaccia tra diverse librerie per il machine learning (NumPy, PyTorch, Tensorflow) e hardware quantistici (Xanadu's Strawberry Fields, Rigetti's Forest, IBM's Qiskit, or Google's Cirq). La computazione di tipo quantistico viene assolta da un dispositivo che può essere un simulatore classico o un reale computer quantistico tra quelli appena elencati. In altre parole, Pennylane permette di addestrare un circuito variazionale sfruttando le classiche librerie per il machine learning.



4.4 Algoritmo e codice

Innanzitutto si importano gli strumenti necessari:

```
import pennylane as qml
from matplotlib import pyplot as plt
import numpy as np
import scipy
import networkx as nx
import copy
```

Si fissa il numero di qubit.

```
qubit_number = 4
qubits = range(qubit_number)
```

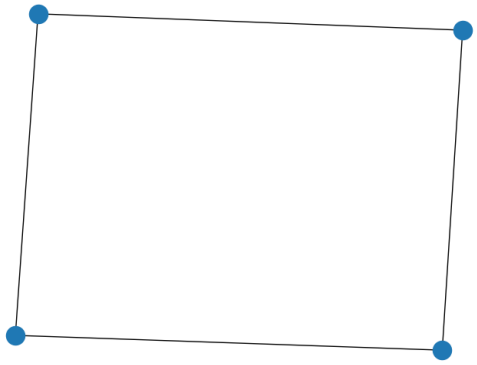
Prima di procedere è necessario generare i dati per l'addestramento della rete. Per farlo bisogna conoscere il grafo di interazione target $\mathcal{G}$ e l'hamiltoniana target.

Sia $\mathcal{G}$ il seguente grafo:

```
ising_graph = nx.cycle_graph(qubit_number)

print(f"Edges: {ising_graph.edges}")
nx.draw(ising_graph)
```

Out:

```
Edges: [(0, 1), (0, 3), (1, 2), (2, 3)]
```



I parametri target $\boldsymbol{\alpha} = \{\alpha^{(1)}, \alpha^{(2)}\}$ si scelgono casualmente da una distribuzione di probabilità uniforme nell'intervallo $[-2, 2]$.

```
    target_weights = [0.37, 0.90, 1.97, -1.45]
    target_bias = [0.77, -1.43, -1.17, 1.73]
```

dove, ricordando l'hamiltoniana di Ising (4.2) i `target_weights` corrispondono ai parametri $\theta^{(1)}$ e i `target_bias` ai parametri $\theta^{(2)}$.

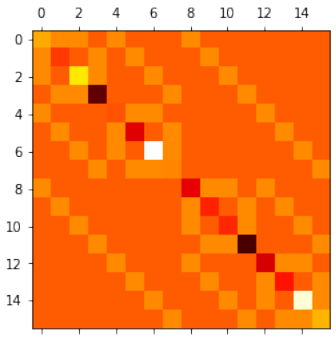
Infine, si genera la forma matriciale dell'hamiltoniana di Ising:

```
def create_hamiltonian_matrix(n_qubits, graph, weights, bias):

    full_matrix = np.zeros((2 ** n_qubits, 2 ** n_qubits))

    # Termine di interazione
    for i, edge in enumerate(graph.edges):
        interaction_term = 1
        for qubit in range(0, n_qubits):
            if qubit in edge:
                interaction_term = np.kron(interaction_term, qml.PauliZ.matrix)
            else:
                interaction_term = np.kron(interaction_term, np.identity(2))
        full_matrix += weights[i] * interaction_term
```

```
    # Termini di bias
    for i in range(0, n_qubits):
        z_term = x_term = 1
        for j in range(0, n_qubits):
            if j == i:
                z_term = np.kron(z_term, qml.PauliZ.matrix)
                x_term = np.kron(x_term, qml.PauliX.matrix)
            else:
                z_term = np.kron(z_term, np.identity(2))
                x_term = np.kron(x_term, np.identity(2))
        full_matrix += bias[i] * z_term + x_term

    return full_matrix
```

e se ne stampa una rappresentazione grafica:

```
ham_matrix = create_hamiltonian_matrix(qubit_number, ising_graph, target_weights, target_bias)
plt.matshow(ham_matrix, cmap="hot")
plt.show()
```



Preparazione dati

Si ricorda che i dati necessari sono:

- diverse copie di uno stato a bassa energia $|\psi_0\rangle$;
- una collezione di stati evoluti a diversi istanti t_k.

Lo stato $|\psi_0\rangle$ può essere ottenuto sfruttando l'algoritmo VQE (Variational Quantum Eigensolver) ma interrompendolo prima che converga allo stato fondamentale. Se si fosse sfruttato lo stato fondamentale al posto di uno stato a bassa energia la dipendenza dal tempo sarebbe stata triviale.

Il seguente è lo stato $|\psi_0\rangle$ che sarà utilizzato

```
low_energy_state = [
    (-0.054661080280306085 + 0.016713907320174026j),
    (0.12290003656489545 - 0.03758500591109822j),
    (0.3649337966440005 - 0.11158863596657455j),
    (-0.8205175732627094 + 0.25093231967092877j),
    (0.010369790825776609 - 0.0031706387262686003j),
    (-0.02331544978544721 + 0.007129899300113728j),
    (-0.06923183949694546 + 0.0211684344103713j),
    (0.15566094863283836 - 0.04760201916285508j),
    (0.014520590919500158 - 0.0044418878360784 86j),
    (-0.032648113364535575 + 0.009988590222879195j),
    (-0.09694382811137187 + 0.02965579457620536j),
    (0.21796861485652747 - 0.06668776658411019j),
    (-0.0027547112135013247 + 0.0008426289322652901j),
    (0.006193695872468649 - 0.0018948418969390599j),
    (0.018391279795405405 - 0.005625722994009138j),
    (-0.041350974715649635 + 0.012650711602265649j),
]
```

Si può verificare che si tratti effettivamente di uno stato a bassa energia calcolando numericamente il più piccolo autovalore dell'hamiltoniana e comparandolo al valore d'aspettazione di $|\psi_0\rangle$:

```
res = np.vdot(low_energy_state, (ham_matrix @ low_energy_state))
energy_exp = np.real_if_close(res)
print(f"Energy Expectation: {energy_exp}")


ground_state_energy = np.real_if_close(min(np.linalg.eig(ham_matrix)[0]))
print(f"Ground State Energy: {ground_state_energy}")
```

Out:

```
Energy Expectation: -5.505790611172773
Ground State Energy: -7.013556035723771
```

Per ottenere gli stati evoluti è sufficiente applicare allo stato $|\psi_0\rangle$ l'operatore di evoluzione temporale, che può essere definito in PennyLane come:

```
def state_evolve(hamiltonian, qubits, time):

    U = scipy.linalg.expm(-1j * hamiltonian * time)
    qml.QubitUnitary(U, wires=qubits)
```

Apprendimento dell'hamiltoniana

Una volta preparati i dati, si può procedere con la definizione dei diversi layer che costituiscono la QGRNN:

```
def qgrnn_layer(weights, bias, qubits, graph, trotter_step):

    # Livello di porte RZZ
    for i, edge in enumerate(graph.edges):
        qml.MultiRZ(2 * weights[i] * trotter_step, wires=(edge[0], edge[1]))

    # Livello di porte RZ
    for i, qubit in enumerate(qubits):
        qml.RZ(2 * bias[i] * trotter_step, wires=qubit)

    # Livello di porte RX
    for qubit in qubits:
        qml.RX(2 * trotter_step, wires=qubit)
```

Come si accenava nella sezione 4.2, la QGRNN sfrutta due registri. In uno sono preparati gli stati evoluti $|\psi(t)\rangle$, nell'altro gli stati $U_{\hat{H}}(\boldsymbol{\mu}, \boldsymbol{\Delta})\,|\psi_0\rangle$. Un passo fondamentale è la stima della somiglianza tra essi, stima che può essere ottenuta dal calcolo della quantità $\left|\langle\psi(t)|\,U_{\hat{H}}(\boldsymbol{\mu}, \boldsymbol{\Delta})\,|\psi_0\rangle\right|^2$, detta *fedeltà*. Per quanto detto nella sezione

2.5.3, applicando uno swap test a tali stati si ottiene:

$$P(|0\rangle) = \frac{1}{2} + \frac{1}{2}\left|\langle\psi(t)| U_{\hat{H}}(\boldsymbol{\mu}, \boldsymbol{\Delta}) |\psi_0\rangle\right|^2. \tag{4.8}$$

Si definisce uno swap test sui due registri come:

```
def swap_test(control, register1, register2):

    qml.Hadamard(wires=control)
    for reg1_qubit, reg2_qubit in zip(register1, register2):
        qml.CSWAP(wires=(control, reg1_qubit, reg2_qubit))
    qml.Hadamard(wires=control)
```

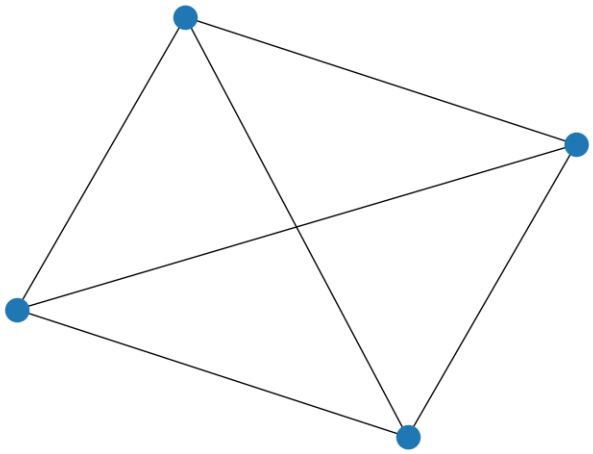
Ciò che si ottiene dalla misura, di fatto, è il valore di aspettazione $\langle Z \rangle$ relativo al qubit di controllo. La probabilità $P(|0\rangle)$ è quindi legata sia alla fedeltà che a $\langle Z \rangle$.

L'ultima operazione preliminare necessaria è definire un grafo predetto di partenza G'. Si sceglie di partire da un grafo completo cosicché qualunque sia il grafo di interazione target, esso sarà un sottografo di G' e i parametri $\boldsymbol{\mu}$ corrispondenti alle connessioni inesistenti tenderanno a zero.

```
reg1 = tuple(range(qubit_number))  # Primo registro
reg2 = tuple(range(qubit_number, 2 * qubit_number))  # Secondo registro

control = 2 * qubit_number  # Indice qubit di controllo
trotter_step = 0.01  # Passo approssimazione di Trotter

# Grafo d'interazione per il nuovo sistema di qubit

new_ising_graph = nx.complete_graph(reg2)

print(f"Edges: {new_ising_graph.edges}")
nx.draw(new_ising_graph)
```

Out:

```
Edges: [(4, 5), (4, 6), (4, 7), (5, 6), (5, 7), (6, 7)]
```



A questo punto non resta che ultimare la QGRNN:

```
def qgrnn(weights, bias, time=None):

    # Preparazione stato a bassa energia in ambo i registri
    qml.QubitStateVector(np.kron(low_energy_state, low_energy_state), wires=reg1 + reg2)

    # Si applica il circuito di evoluzione temporale al primo registro
    state_evolve(ham_matrix, reg1, time)

    # Si applica la QGRNN al secondo registro
    depth = time / trotter_step  # P = t/Delta
    for _ in range(0, int(depth)):
        qgrnn_layer(weights, bias, reg2, new_ising_graph, trotter_step)

    # SWAP test fra i due registri
    swap_test(control, reg1, reg2)

    # Risultati SWAP test
    return qml.expval(qml.PauliZ(control))
```

Ora che il circuito è completo si può definire la funzione di costo.

La quantità $\left|\langle\psi(t)|\, U_{\hat{H}}(\boldsymbol{\mu}, \boldsymbol{\Delta})\, |\psi_0\rangle\right|^2$ tende a 1 per stati simili e a 0 per stati ortogonali, quindi si può minimizzare la quantità $-\left|\langle\psi(t)|\, U_{\hat{H}}(\boldsymbol{\mu}, \boldsymbol{\Delta})\, |\psi_0\rangle\right|^2$. Dal momento che il calcolo va effettutato per N diversi campioni di dati, la funzione di costo sarà

la fedeltà negativa media tra i due registri:

$$\mathcal{L}(\boldsymbol{\mu}, \Delta) = -\frac{1}{N} \sum_{i=1}^{N} \left| \langle \psi(t) | \, U_{\tilde{H}}(\boldsymbol{\mu}, \boldsymbol{\Delta}) \, | \psi_0 \rangle \right|^2 \tag{4.9}$$

Quindi si definiscono N e il valore massimo di tempo

```
N = 15  # Numero di campioni per ogni step
max_time = 0.1  # Tempo massimo per ogni evoluzione temporale
```

e la funzione di costo

```
rng = np.random.default_rng(seed=42)

def cost_function(weight_params, bias_params):

    # Si scelgono casualmente gli istanti cui la QGRNN evolve
    times_sampled = rng.random(size=N) * max_time

    # Calcolo funzione di costo
    total_cost = 0
    for dt in times_sampled:
        result = qgrnn_qnode(weight_params, bias_params, time=dt)
        total_cost += -1 * result

    return total_cost / N
```

Si imposta l'algoritmo di ottimizzazione, Adam optimizer:

```
# Si definisce il nuovo dispositivo
qgrnn_dev = qml.device("default.qubit", wires=2 * qubit_number + 1)

# Si definisce il nuovo QNodo
qgrnn_qnode = qml.QNode(qgrnn, qgrnn_dev)

steps = 500

optimizer = qml.AdamOptimizer(stepsize=0.5)
```

```
weights = rng.random(size=len(new_ising_graph.edges)) - 0.5
bias = rng.random(size=qubit_number) - 0.5

initial_weights = copy.copy(weights)
initial_bias = copy.copy(bias)
```

e si esegue il loop di ottimizzazione:

```
for i in range(0, steps):
    (weights, bias), cost = optimizer.step_and_cost(cost_function, weights, bias)

    # Si stampano i valori della funzione di costo
    if i % 5 == 0:
        print(f"Cost at Step {i}: {cost}")
        print(f"Weights at Step {i}: {weights}")
        print(f"Bias at Step {i}: {bias}")
        print("-------------------------------------------")
```

Out:

```
Cost at Step 0: -0.9460448001301196
Weights at Step 0: [-0.22604193  0.43887117  0.85859761  0.69736699  0.0941771  -0.02437739]
Bias at Step 0: [ 0.76113097 -0.21393427  0.12811009  0.45038567]
-------------------------------------------
Cost at Step 5: -0.9957121598422894
Weights at Step 5: [-1.15713352  0.57369314  2.04062306  1.84441956  1.29770836 -1.2307227 ]
Bias at Step 5: [ 2.73513424 -1.39206097  1.17904007  1.6621921 ]
-------------------------------------------
    .
    .
    .
Cost at Step 490: -0.9997390793308107
Weights at Step 490: [ 0.39681112 -0.01696001  0.98328524  2.11565063  0.01423494 -1.56889149]
Bias at Step 490: [ 0.9765824  -1.34404702 -1.50454034  1.80746523]
-------------------------------------------
Cost at Step 495: -0.9998320209394237
```

```
Weights at Step 495: [ 0.33807451  0.0740421   0.97942133  2.12608291  0.00388393 -1.55964889]
Bias at Step 495: [ 0.89243953 -1.34918275 -1.48061772  1.79599584]
--------------------------------------------
```

Si può effettuare un confronto tra le rappresentazioni grafiche dell'hamiltoniana basate sui parametri target, iniziali e appresi:

```
new_ham_matrix = create_hamiltonian_matrix(
    qubit_number, nx.complete_graph(qubit_number), weights, bias)

init_ham = create_hamiltonian_matrix(
    qubit_number, nx.complete_graph(qubit_number), initial_weights, initial_bias)

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(6, 6))

axes[0].matshow(ham_matrix, vmin=-7, vmax=7, cmap="hot")
axes[0].set_title("Target", y=1.13)

axes[1].matshow(init_ham, vmin=-7, vmax=7, cmap="hot")
axes[1].set_title("Iniziale", y=1.13)

axes[2].matshow(new_ham_matrix, vmin=-7, vmax=7, cmap="hot")
axes[2].set_title("Appresa", y=1.13)

plt.subplots_adjust(wspace=0.3, hspace=0.3)
plt.show()
```

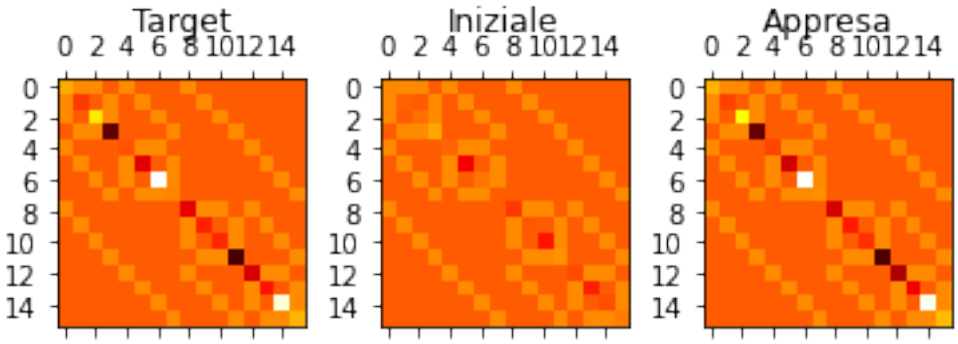


Figura 4.2: Confronto visivo

Le immagini risultano pressoché indistinguibili.

Si possono osservare infine i parametri ottenuti:

```
Parametri target     Parametri appresi
Pesi
-----------------------------------------
0.37                |  0.3647701752682651
0.9                 |  0.9637165895487491
1.97                |   2.121673254702519
-1.45               | -1.5482608065496295

Bias
-----------------------------------------
0.77                |  0.9211049393681469
-1.43               | -1.3411484532950293
-1.17               | -1.4721369551941315
1.73                |  1.7835349429661653

Parametri relativi alle connessioni inesistenti:
[0.05243482486848598, -0.007978503061897095]
```

I pesi corrispondenti alle connessioni inesistenti $(1, 3)$ e $(2, 0)$ sono prossimi a 0. Per il resto, i parametri appresi sono molto simili ai parametri target.

Capitolo 5

Conclusioni e possibili sviluppi futuri

Lo scopo del presente studio è stato quello di indagare sui punti di forza e i potenziali vantaggi del quantum computing, in particolare per quanto riguarda l'ambito del machine learning.

Nel secondo capitolo sono stati innanzitutto introdotti i concetti teorici fondamentali utili alla comprensione del quantum computing; successivamente sono state introdotte alcune delle più importanti porte logiche quantistiche, quindi è stata illustrata la struttura di un generico circuito quantistico con particolare attenzione ai processi di codifica dei dati e di misura. Infine sono stati presentati gli algoritmi di Shor, di Grover e lo swap test: i primi due - sia per il loro valore intrinseco che per quello storico - come esempi di algoritmi quantistici più performanti rispetto alle controparti classiche; lo swap test in quanto spesso presente come subroutine in molti algoritmi quantistici e perché fosse più chiara la sua utilità nella realizzazione della QGRNN.

Il terzo capitolo si è aperto con una panoramica generale sul machine learning e sulle reti neurali per poi focalizzarsi in particolare sul percettrone, sulle reti neurali ricorrenti e infine sulle GNN. Proprio queste ultime, nella loro variante quantistica, sono state protagoniste dell'applicazione presentata nel quarto capitolo. Prima di studiare le QGRNN è stato introdotto il quantum machine learning e si è parlato dell'approccio ibrido - in particolare dell'utilità dei circuiti variazionali nell'ambito

del machine learning - e del perché esso risulti una soluzione valida agli attuali limiti tecnologici della NISQ era.

In chiusura del capitolo è stato inserito un breve excursus sull'ottimizzazione basata sul metodo della discesa del gradiente, il quale si è concluso con l'introduzione dell'algoritmo Adam. Infine è stato brevemente illustrato il VQE. Entrambi gli algoritmi, Adam e VQE, sono serviti per l'applicazione realizzata nel capitolo successivo.

Nel quarto capitolo è stato introdotto il modello di Ising in campo trasverso, quindi è stata evidenziata l'utilità di un approccio tramite QGRNN per studiarne la dinamica. Successivamente è stato brevemente descritto il software usato per implementare la QGRNN, PennyLane, e infine è stato illustrato e spiegato l'algoritmo e il codice. La performance della QGRNN è promettente; i risultati sono soddisfacenti e soprattutto ciò che è interessante sottolineare sono le potenzialità delle QGNN in generale. Per esempio, l'uso delle QGNN potrebbe essere particolarmente utile nell'ambito della fisica nucleare - più precisamente per la ricostruzione delle tracce delle particelle emesse in una collisione - e della chimica quantistica. Inoltre, il modello di Ising è strettamente correlato alla classe di problemi QUBO (Quadratic Unconstrained Binary Optimization), di cui fanno parte molti problemi non trattabili classicamente; in questo senso l'approccio tramite QGRNN potrebbe risultare molto utile per le numerose potenziali applicazioni.

Come anticipato nell'introduzione, il quantum computing è ormai uno dei settori di ricerca più scottanti. Non solo le già citate Google e IBM, molte altre aziende stanno investendo sulle tecnologie quantistiche e stringendo collaborazioni con università e istituti di ricerca. Per esempio, Airbus ha lanciato nel 2019 la Airbus Quantum Computing Challenge, una competizione indetta allo scopo di risolvere alcune delle più importanti sfide dell'industria aerospaziale grazie alla potenza del calcolo quantistico; ancora dal settore aerospaziale, Lockheed Martin ha stretto accordi con la University of Southern California e con D-Wave Systems.

Dal settore automobilistico, BMW sta già sfruttando i computer quantistici H0 e H1 della Honeywell per minimizzare i costi della filiera dei suoi approvvigionamenti; Wolksvagen ha lanciato a Lisbona il primo progetto per l'ottimizzazione dei flussi di traffico sfruttando la potenza di calcolo dei dispositivi D-Wave.

Dai settori delle telecomunicazioni e della crittografia quantistica, rispettivamente,

SK Telecom sta già costruendo una rete quantistica in Corea del Sud; Toshiba ha sviluppato un sistema di distribuzione quantistica delle chiavi.

Nokia, proprietaria di Bell Laboratories - presso i quali Shor sviluppò il celebre algoritmo - è coinvolta in un progetto con l'Università di Oxford e Lockheed Martin per l'impiego del quantum computing nell'ambito del machine learning.

Anche l'accessibilità agli strumenti di sviluppo forniti da diverse società sta contribuendo alla diffusione del quantum computing. Nel 2017 Microsoft ha lanciato il Quantum Development Kit, composto da un framework di programmazione e il linguaggio Q#. Nello stesso anno IBM ha rilasciato Qiskit, un framework open-source che fornisce gli strumenti necessari allo sviluppo di software compatibilmente con il servizio di cloud quantum computing IBM. Nel 2018 il progetto Xanadu ha rilasciato PennyLane, il primo software per il quantum computing dedicato per il machine learning. Nel 2021 IBM ha rilasciato un nuovo modulo Qiskit: Qiskit Machine Learning si integra con le funzionalità preesistenti permettendo la creazione di circuiti parametrizzati e il calcolo automatico dei gradienti rispetto ai parametri circuitali.

Quanto detto attesta le alte aspettative e la grande fiducia riposte nel quantum computing da parte della comunità scientifica e dell'industria, aspettative che, pur tenendo conto degli importanti limiti propri della NISQ era, hanno motivo di esistere. Se è improbabile che i NISQ computer possano portare a una vera e propria rivoluzione, è pur vero che essi saranno fondamentali nel percorso che porta alla creazione di dispositivi quantistici più avanzati.

Per quanto riguarda il quantum machine learning, i vantaggi rispetto al machine learning classico sono i vantaggi propri del quantum computing. Oltre all'innovazione tecnologica, è importante apportare un cambio di paradigma: piuttosto che riprodurre versioni quantistiche di algoritmi classici è fondamentale lavorare allo sviluppo di modelli quantistici di apprendimento.

Bibliografia

[1] Verdon, G., McCourt, T., Luzhnica, E., Singh, V., Leichenauer, S., Hidary, J. (2019). Quantum Graph Neural Networks. arXiv preprint arXiv:1909.12264.

[2] Benedetti, M., Lloyd, E., Sack, S., and Fiorentini, M. (2019). Parameterized quantum circuits as machine learning models. Quantum Science and Technology, 4(4):043001.

[3] J. Choi, S. Oh and J. Kim, "A Tutorial on Quantum Graph Recurrent Neural Network (QGRNN)," 2021 International Conference on Information Networking (ICOIN), Jeju Island, Korea (South), 2021, pp. 46-49, doi: 10.1109/ICOIN50884.2021.9333917.

[4] Wittek, P. Quantum Machine Learning: What Quantum Computing Means to Data Mining (Academic Press, New York, NY, USA, 2014).

[5] Diederik P. Kingma, Jimmy Ba, Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980

[6] John Preskill, Quantum Computing in the NISQ era and beyond. arXiv preprint arXiv:1801.00862v3

[7] Maria Schuld, Quantum machine learning for supervised pattern recognition.

[8] R. Shaydulin, H. Ushijima-Mwesigwa, C. F. A. Negre, I. Safro, S. M. Mniszewski and Y. Alexeev, "A Hybrid Approach for Solving Optimization Problems on Small Quantum Computers," in Computer, vol. 52, no. 6, pp. 18-26, June 2019, doi: 10.1109/MC.2019.2908942.

[9] Maria Schuld, Ilya Sinayskiy and Francesco Petruccione, An introduction to quantum machine learning. arXiv preprint arXiv:1409.3097v1

[10] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, Maosong Sun, Graph Neural Networks: A Review of Methods and Applications. arXiv preprint arXiv:1812.08434v4

[11] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, "The Graph Neural Network Model," in IEEE Transactions on Neural Networks, vol. 20, no. 1, pp. 61-80, Jan. 2009, doi: 10.1109/TNN.2008.2005605.

[12] Kopczyk, D. (2018). Quantum machine learning for data scientists. arXiv preprint arXiv:1804.10068.

[13] Viraj Kulkarni, Milind Kulkarni, Aniruddha Pant, Quantum Computing Methods for Supervised Learning. arXiv preprint arXiv:2006.12025v1

[14] Abraham Asfaw, Luciano Bello, Y. B.-H. S. B. L. C. A. C. V. J. C. R. C. A. F. J. G. S. G. L. G. S. D. L. P. G. F. H. T. I. D. M. A. M. Z. M. R. M. G. N. P. N. A. P. M. P. A. R. J. S. J. S. J. S. K. T. M. T. S. W. J. W. (2020). Learn quantum computation using qiskit.

[15] Maria Schuld, Alex Bocharov, Krysta Svore, Nathan Wiebe, Circuit-centric quantum classifiers. arXiv preprint arXiv:1804.00633

[16] Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka, Hoe powerful are graph neural networks? arXiv preprint arXiv:1810.00826v3

[17] Yujia Li & Richard Zemel, Marc Brockschmidt & Daniel Tarlow, Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493v4