Università degli Studi di Milano-Bicocca

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Triennale in Fisica

Sviluppo di una Rete Neurale Convoluzionale Quantistica per classificazione binaria

Candidato: Giuliano Trapani

Matricola: 846287

Relatore: Prof. Angelo Enrico Lodovico Nucciotti
Correlatore: Dr. Roberto Moretti

Anno accademico 2022-2023

Sommario

La teoria della meccanica quantistica, nata agli inizi del XX secolo, cambiò radicalmente la nostra comprensione dell'Universo, fornendo modelli adatti a descrivere fenomeni su scala atomica e subatomica. Negli anni '80, i fisici statunitensi Murray Gell-Mann e Richard Feynman intravidero la possibilità di fondare un nuovo paradigma dell'informazione sfruttando le proprietà intrinseche di sistemi quantistici. Nasceva così l'idea di sviluppare calcolatori basati sulle proprietà intrinseche della fisica quantistica quali l'entaglement, la sovrapposizione e l'interferenza tra funzioni d'onda per eseguire sequenze di istruzioni. Queste macchine operano su unità di informazione, dette qubit, che consistono di sistemi quantistici a due livelli energetici chiamati $|0\rangle$ e $|1\rangle$ (la controparte quantistica del bit). Lo stato di un qubit è esprimibile come sovrapposizione di tali autostati. Questo permette ad un calcolatore quantistico di accedere a nuovi algoritmi in grado di ridurre significativamente la complessità di alcuni problemi rispetto ad un approccio classico, quali la scomposizione in fattori primi e la ricerca in database non strutturati. Lo sviluppo di computer quantistici affidabili ad alto numero di qubit rappresenta oggi una sfida tecnologica che coinvolge gruppi di ricerca e aziende tra cui IBM e Google. Quest'ultima annunciò nel 2019 il raggiungimento della "supremazia quantistica", con il suo processore Sycamore (una macchina a 53 qubit) eseguendo correttamente un algoritmo in pochi minuti, contro i 10 000 anni stimati per un super computer classico.

Questa tesi si colloca nel contesto del Quantum Machine Learning (QML), ossia del quantum computing applicato ad algoritmi di apprendimento supervisionato. Il machine learning è una disciplina dell'intelligenza artificiale che si occupa di studiare e creare sistemi in grado di apprendere la struttura di dati forniti in input per assolvere diversi tipi di compiti, come classificazione, regressione o generazione di nuovi dati.

Il machine learning viene applicato con successo in campi come la medicina, il riconoscimento vocale, i social media e nell'analisi di dati sperimentali. In particolare, questo lavoro di tesi si concentra sullo sviluppo di reti neurali artificiali, ossia un insieme di nodi, chiamati neuroni, che possono essere organizzati in varie architetture. E' utile schematizzare una rete neurale come una sequenza di strati (detti layer) di nodi, ciascuno dei quali è interconnesso ai nodi del layer successivo. L'efficienza delle reti neurali e la rapidità del loro apprendimento dipende considerevolmente dalla dimensione del dataset iniziale e dalla sua architettura. Nel lavoro di tesi si costruisce e si caratterizza una rete neurale classica, per poi modificarne l'architettura inserendo layer quantistici. In questo elaborato, un layer quantistico corrisponde ad un circuito quantistico, ossia una sequenza ordinata di operazioni di controllo su stati multi-qubit parametrizzata da pesi, i cui valori ottimali sono definiti attraverso l'addestramento. L'allenamento della rete avviene tramite i meccanismi di forward

e backward propagation, che permettono di correggere iterativamente i parametri del modello, in modo da minimizzare una funzione costo, o loss function, tramite un metodo di discesa del gradiente. Nello specifico, la rete neurale sviluppata in questa tesi è una rete neurale di tipo convoluzionale, contraddistinta dall'uso di particolari layer specializzati per la Computer Vision, denominati *convoluzionali* e *pooling*. Tale tipologia di rete è stata scelta in funzione del problema di analisi considerato, che consiste nella identificazione del Plasmodium, parassita responsabile della malaria, all'interno di globuli rossi. I dati a disposizione sono immagini di globuli rossi ingranditi al microscopio trattate con il colorante Giemsa II per evidenziare la presenza del parassita. Nel lavoro di tesi si è confrontata l'accuratezza di classificazione del modello classico con quella di diversi modelli quantistici, considerando circuiti a due e tre qubit ed al variare della quantità di entanglement. I risultati ottenuti dalla rete neurale quantistica risultano essere competitivi con quelli della rete classica, con un'accuratezza di classificazione compresa tra il 72% ed il 74% per entrambe le tipologie di modello

Indice

Capitolo 1

Quantum computing

1.1 Introduzione al quantum computing

Era il 1964 quando durante una conferenza alla Cornell University Richard Feynman esponeva ai suoi studenti uno dei suoi più celebri pensieri sulla teoria quantistica: 'nobody understands Quantum Mechanics'. Chiaramente il famoso scienziato non intendeva dire che la meccanica quantistica fosse qualcosa di incomprensibile, ma piuttosto che la sua comprensione andasse contro la intuizione umana [1].

La meccanica quantistica si sviluppa nel primo quarto del secolo scorso e la teoria per come la conosciamo oggi possiamo dire sia nata grazie a Werner Karl Heisenberg ed Erwin Schrödinger e formalizzata da Paul Dirac, Neumann, e Weyl nei primi anni 30. Nel corso degli anni la meccanica quantistica ci ha aiutato ad esempio a spiegare la struttura dell'atomo, la superconduttività di alcuni materiali, la fusione nucleare nelle stelle e ci ha portati allo sviluppo di teorie più complesse come la Elettrodinamica Quantistica o lo sviluppo del Modello Standard. In questo scenario, negli anni '80 nasce la computazione quantistica e la teoria dell'informazione quantistica che si propone, tra gli altri,anche l'obiettivo di migliorare la nostra intuizione su fenomeni di natura quantistica [2]. La computazione quantistica si concentra sulla realizzazione di computer quantistici (macchine che utilizzano le proprietà quantistiche della materia come superposizione, entanglement e interferenza di funzioni d'onda) e sugli algoritmi che possono essere eseguiti grazie ad essi. L' informazione quantistica studia invece come criptare, manipolare e trasmettere informazioni usando i sistemi quantistici [3]. Ad esempio, uno dei principali risultati della informazione quantistica è il teorema no-clonig il quale stabilisce che non è possibile creare una copia di uno stato quantistico arbitrario sconosciuto.

Il primo modello di computer quantistico fu proposto da Paul Benioff nel 1980: la macchina di Turing quantistica [4]. Nel corso degli anni sono poi stati proposti diversi altri modelli (Measurement based quantum computing MBQC, Adiabatic quantum computing, Topological quantum computing). Tra questi, quello ad aver riscosso più successo è il "circuito quantistico" [5].

La realizzazione di dispositivi di calcolo quantistico presenta tuttora numerose sfide tecnologiche. Tuttavia nell'ultimo decennio questo settore di ricerca ha visto un forte sviluppo e sono nati i primi veri processori quantistici (si veda sezione 1.4). In questo ambito due dei principali competitor sono IBM e Google con quest'ultima che nell'ottobre del 2019 ha pubblicato un articolo su Nature [6] in cui afferma di aver raggiunto la supremazia quantistica (dall'inglese quantum primacy). La

supremazia quantistica si raggiunge quando un computer quantistico è in grado di eseguire un'operazione in modo esponenzialmente più veloce rispetto a un computer classico .

Il problema principale del risultato raggiunto da Google è che l'algoritmo implementato non ha un'utilità pratica. Tralasciando i possibili dibattiti sul fatto che la quantum primacy sia stata effettivamente raggiunta (IBM ha dimostrato che invece che impiegare diversi anni come affermava google un sumpercomputer classico poteva completare il calcolo in 2 giorni circa [7]) l'attenzione per certi versi è adesso concentrata sull'ottenere il vantaggio quantistico. Si parla di vantaggio quantistico quando un algoritmo di utilità pratica è in grado di risolvere un problema in modo più efficiente di quello classico.

In particolare il lavoro di questa tesi riguarda lo sviluppo di una rete neurale convoluzionale che sfrutti sia layer classici che quantistici per affrontare un problema di classificazione binaria, confrontando i risultati ottenuti con quelli da un modello puramente classico.

1.2 Qubit

Per poter parlare di calcolo quantistico è utile partire dalla definizione di qubit. Un qubit ricopre per la informazione quantistica lo stesso ruolo che ricopre il bit in ambito classico ossia rappresenta l'unità di base dell'informazione. Il primo postulato della meccanica quantistica afferma che ad ogni sistema quantistico è associato uno spazio di Hilbert: uno spazio vettoriale sul campo complesso munito di metrica e distanza, tale spazio è detto spazio degli stati del sistema. Il sistema è poi completamente descritto dal suo vettore di stato appartenente allo spazio degli stati.

Il qubit è un sistema quantistico ed il suo stato è descrivibile utilizzando un vettore $|\psi\rangle$ (seguendo la notazione di Dirac). Lo stato fisico di un qubit può essere $|0\rangle$ o $|1\rangle$ (il bit classico si può trovare solo in 2 stati o 0 o 1). La sostanziale differenza tra un bit classico ed un qubit è che lo stato più generale di quest'ultimo è una sovrapposizione degli stati $|0\rangle$ e $|1\rangle$, ovvero in altri termini lo stato più generale del qubit è una combinazione lineare degli elementi costitutivi della base ortonormale associata allo spazio di Hilbert del sistema. Tale proprietà è detta superposizione e in notazione di Dirac è esplicitata come segue:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.1}$$

con α e β complessi tali che

$$|\alpha|^2 + |\beta|^2 = 1 \tag{1.2}$$

Ciò che è importante tenere in considerazione è che nel momento della misura lo stato del qubit collassa in uno tra gli stati della sua base (collasso della funzione d'onda). Di fatto i valori $|\alpha|^2$ e $|\beta|^2$ rappresentano rispettivamente le probabilità di trovarsi nei due autostati. In altri termini la caratteristica fondamentale che distingue un qubit da un bit è la sua indeterminatezza intrinseca: prima della misura è possibile solo sapere quali sono le probabilità che il qubit si trovi nello stato $|0\rangle$ o $|1\rangle$.

Lo stato $|\psi\rangle$ è rappresentabile graficamente su quella che è detta sfera di Bloch, ovvero una sfera di raggio unitario i cui punti sulla superficie sono in corrispondenza biunivoca con gli stati del sistema quantistico a 1-qubit appena descritto (figura 1.1).

Se $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ allora è possibile trovare tre angoli γ,δ,θ tali che

$$\alpha = e^{i\gamma}\cos\left(\frac{\theta}{2}\right) \tag{1.3}$$

$$\beta = e^{i\delta}\sin\left(\frac{\theta}{2}\right) \tag{1.4}$$

Tale rappresentazione risulta essere rindondante in quanto i fattori di fase globale non influiscono sugli stati fisici. Risulta quindi possibile riscrivere lo stato come segue

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{1.5}$$

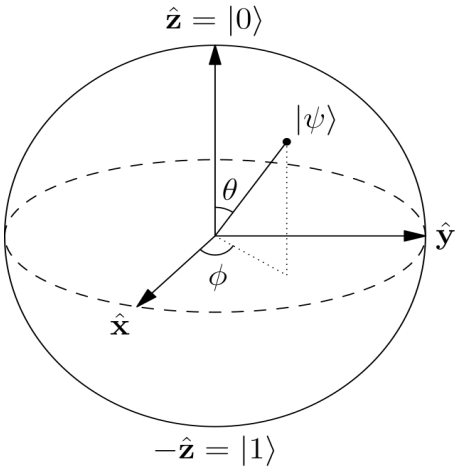


Figura 1.1: Rappresentazione di un sistema a 2 livelli su sfera di Bloch

Quanto abbiamo appena detto su si può estendere a sistemi quantistici formati da più qubit, ad esempio nel caso di due qubit avremo che ognuno di essi si potrà trovare nello stato $|0\rangle$ o $|1\rangle$ e il sistema complessivo potrà trovarsi in quattro stati possibili $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. Lo stato più generale del sistema sarà allora

$$|\Psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle \tag{1.6}$$

con i coefficienti tali che

$$\sum_{a,b=0}^{1} |\alpha_{ab}|^2 = 1$$

La probabilità di osservare il primo qubit nello stato $|0\rangle$ sarà data dalla somma delle probabilità di trovare il sistema complessivo nello stato $|01\rangle$ e $|00\rangle$ e sarà quindi nel nostro caso $|\alpha_{01}|^2 + |\alpha_{00}|^2$.

1.3 Quantum gates

La controparte quantistica delle porte logiche sono i gate quantistici (o quantum gates). Questi, altro non sono che l'estensione delle porte logiche applicabili a stati quantistici, che permettono di controllare uno o più qubit e come tali permettono di operare sui qubit. Per comprendere la loro rappresentazione matematica è necessario un altro dei postulati della meccanica quantistica, quello riguardante l'equazione di Schrödinger che in una delle sue formulazioni stabilisce che l'evoluzione temporale di un sistema quantistico isolato è descritta da una trasformazione unitaria, che connette lo stato del sistema iniziale $|\psi(t_1)\rangle$ al tempo t_1 con lo stato finale $|\psi(t_2)\rangle$ al tempo t_2 attraverso un operatore unitario U [5]. In altre parole

$$|\psi(t_2)\rangle = U\,|\psi(t_1)\rangle \tag{1.7}$$

dove U è un operatore lineare unitario che rappresenta l'evoluzione temporale del sistema. I quantum gates sono quindi matrici unitarie ovvero $U : UU^\dagger = 1$. La caratteristica di unitarietà è necessaria per conservare le probabilità. Tali matrici si applicano agli stati e permettono quindi di trasformarli.

Nel caso specifico di sistemi quantistici a 1-qubit le matrici sono 2x2 e le più importanti sono le seguenti:

- X-GATE:

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

  tale gate modifica gli autostati come segue: $X\,|0\rangle = |1\rangle\,, X\,|1\rangle = |0\rangle$

- Z-GATE:

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

  tale gate modifica gli autostati come segue: $Z\,|0\rangle = |0\rangle\,, Z\,|1\rangle = -\,|1\rangle$

- Y-GATE:

$$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

  tale gate modifica gli autostati come segue: $Y\,|0\rangle = i\,|1\rangle\,, Y\,|1\rangle = -i\,|0\rangle$

- H-GATE:

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

  tale gate modifica gli stati come segue: $H\,|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), H\,|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

Si noti inoltre che la rappresentazione di Bloch può essere utile anche in questo caso per comprendere meglio come agisce un operatore unitario a livello intuitivo. La applicazione di un gate al sistema può essere vista come rotazione dello stato quantistico di un certo angolo attorno a un determinato asse della sfera di Bloch. Così ad esempio si avrà che il gate X rappresenta una rotazione di π attorno all'asse x, il gate Y rappresenta una rotazione di π attorno all'asse y, mentre il gate Z rappresenta una rotazione di π attorno all'asse z. Più in generale, detti $R_x(\theta), R_y(\theta), R_z(\theta)$ i gate di rotazione che agiscono sullo stato ruotandolo di un angolo theta rispetto al relativo asse. Allora è possibile dimostrare che esistono angoli α, β, γ tali che un qualunque 1-qubit gate U risulti scrivibile come [5]

$$U = R_x(\alpha)R_y(\beta)R_z(\gamma) \tag{1.8}$$

Nella maggior parte delle applicazioni è necessario lavorare con sistemi a più qubit. Nel caso più semplice, a due qubit, si utilizzano matrici unitarie 4x4. Il modo più semplice per creare un "two-qubit gate" è quello di utilizzare due "one-qubit gate". Siano detti A e B due matrici che lavorano separatamente sui singoli qubit. In tal caso la matrice che rappresenta tale gate è una matrice unitaria 4x4 data dal prodotto tensoriale $A \otimes B$.

Non tutti i gate che operano su due qubit risultano però fattorizzabili come sopra, un esempio è il CNOT GATE che nella base computazionale di un sistema a due qubit ($|00\rangle, |01\rangle, |10\rangle, |11\rangle$) sarà scrivibile come:

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

tale gate agisce sul secondo qubit (target qubit) modificandolo nel caso il primo qubit (control qubit) sia 1, lasciandolo inalterato altrimenti. Il CNOT GATE combinato con l'H-GATE permette la creazione dei più semplici stati-entangled: gli stati di bell.

$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{1.9}$$

$$\left|\Phi^-\right\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \tag{1.10}$$

$$\left|\Psi^+\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \tag{1.11}$$

$$\left|\Psi^-\right\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \tag{1.12}$$

La peculiarità di tali stati (stati quantistici di due qubit) è quella di non essere scrivibili come prodotto di due singoli stati, in altri termini non esistono stati $|\psi_1\rangle$ e $|\psi_2\rangle$ (stati rispettivamente del primo e del secondo qubit) tali che:

$$|\psi\rangle = |\psi_1\rangle\,|\psi_2\rangle \tag{1.13}$$

1.4 Qubit a livello hardware

Abbiamo definito il qubit come un sistema quantistico a due livelli, ma come è possibile creare un sistema del genere a livello hardware? In generale per creare un qubit, è necessario avere la capacità di controllare e manipolare le proprietà quantistiche di un sistema fisico. I metodi sono vari, ad esempio si può utilizzare lo spin di ioni confinati in un campo magnetico (ion traps), dove la base 0 e 1 è associata all'orientamento "su" o "giù" dello spin. Un altro metodo è quello di utilizzare la polarizzazione di un singolo fotone, dove gli stati sono rappresentati dalla direzione di polarizzazione del fotone. Esistono poi i qubit di tipo superconduttivo: dispositivi simili a circuiti LC ma con un'induttanza non lineare, chiamata giunzione Josephson, in grado di conferire al sistema la non-linearità per realizzare un qubit. I qubit appena descritti, infatti, possono essere visti come degli oscillatori quantistici anarmonici: il potenziale devia dalla forma parabolica del caso armonico, in modo da rendere disuguali le separazioni energetiche tra diversi livelli. La caratteristica di anarmonicità consente al qubit di avere un'energia di transizione tra il suo stato fondamentale e il primo stato eccitato che è più grande rispetto alle altre transizioni e risulta quindi possibile isolare uno spazio di lavoro formato dai primi due livelli e trattarli come base computazionale $|0\rangle$ e $|1\rangle$. Nello specifico, i computer IBM utilizzano i qubit di tipo transmon che presentano una separazione dei livelli energetici molto stabile anche in presenza di rumore di carica.

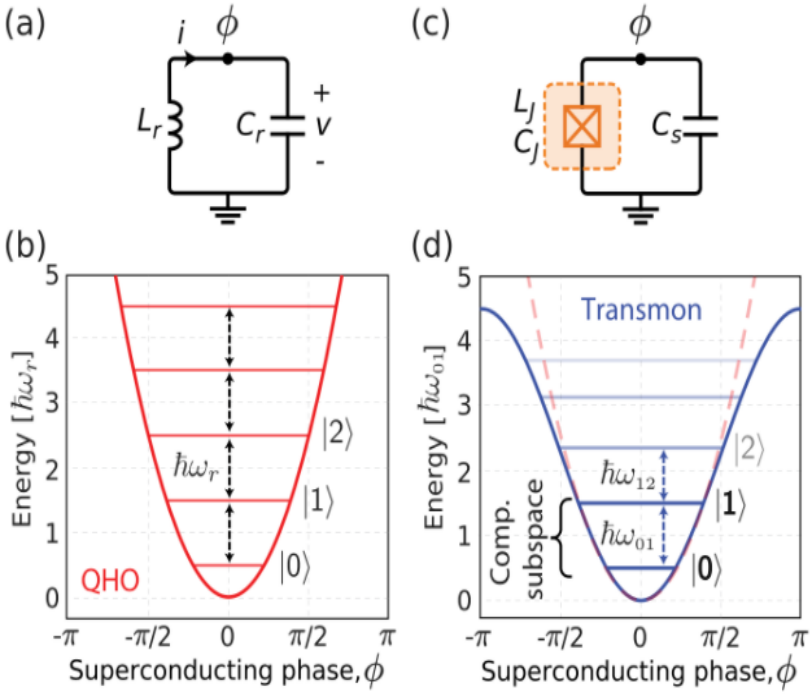


Figura 1.2: A sinistra potenziale armonico dovuto al circuito LC classico, a destra potenziale anarmonico dovuto all'aggiunta della giunzione Josephson

1.5 Creazione di un circuito quantistico con Qiskit

L'insieme delle operazioni effettuate in sequenza dai quantum gates sui qubit sono detti algoritmi quantistici, rappresentabili graficamente come circuiti quantistici. Nella rappresentazione grafica di un circuito quantistico i 1-qubit gate sono rappresentati da riquadri con all' interno il simbolo identificativo mentre i gate che operano su più qubit hanno rappresentazioni leggermente più complesse. Per esempio, il CNOT-gate è descritto da una linea che parte dal qubit di controllo e termina sul qubit target con il simbolo "+" (figura 1.3)

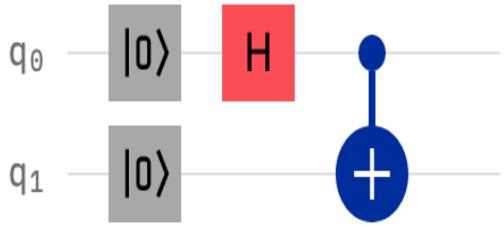


Figura 1.3: Creazione dello stato di Bell $|\Phi^+\rangle$ applicando H-gate e CNOT-GATE ad un sistema nello stato iniziale $|00\rangle$

Qiskit è un framework open source che consente di creare e manipolare programmi e algoritmi quantistici utilizzando il linguaggio di programmazione Python [8]. La libreria è progettata per funzionare sui dispositivi di IBM Quantum Experience, una piattaforma online che mette a disposizione computer quantistici e simulatori di IBM. Questi strumenti possono essere utilizzati tramite il cloud, basta collegarsi al backend e attendere il proprio turno nella coda. Nel lavoro di tesi, è stato utilizzato il simulatore di default di Qiskit, ovvero il simulatore Statevector. Questo simulatore consente di simulare il comportamento di un circuito quantistico e di calcolare lo stato del sistema quantistico alla fine dell'esecuzione del circuito. Rispetto ai computer quantistici reali, il simulatore Statevector ha il vantaggio di essere molto più veloce ed efficiente, poiché non richiede l'uso di hardware quantistico fisico. Tuttavia, il simulatore Statevector ha anche dei limiti, poiché può simulare solo un numero limitato di qubit e non tiene conto degli effetti di rumore e decoerenza che si verificano nei computer quantistici reali Di seguito si mostra (figura 1.4) come creare il circuito quantistico mostrato in figura (1.3)

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

myQuantumRegister = QuantumRegister(2, 'q')
myClassicalRegister = ClassicalRegister(4, 'c')
myCircuit = QuantumCircuit(myQuantumRegister, myClassicalRegister)

circuit.reset(myQuantumRegister[0])
circuit.reset(myQuantumRegister[1])
circuit.h(myQuantumRegister[0])
circuit.cx(myQuantumRegister[0], myQuantumRegister[1])
```

Figura 1.4: Creazione circuito quantistico con qiskit

Vengono creati un registro quantistico di due qubit (ogni qubit è referenziato come myQuantumRegister[nQubit]) e un registro classico di due bit. Alla riga sei viene creato il circuito quantistico "myCircuit" utilizzando i registri creati in precedenza come input. Le righe successive del codice definiscono le operazioni che saranno applicate ai qubit. In particolare, le prime due righe resettano i qubit, portandoli nello stato $|0\rangle$. Nelle righe successive, il circuito applica un gate H sul primo qubit e un CNOT-gate tra il primo e il secondo qubit.

Capitolo 2

Machine Learning

Nel 1950 il famoso matematico Alan Turing cercando di rispondere al quesito "può una macchina pensare?" proponeva il famoso test che prende il suo nome. Il test veniva superato dalla macchina se un interlocutore umano, dopo aver posto delle domande alla macchina, non era in grado di determinare se avesse parlato con un computer o un essere umano. Per superare tale test il computer doveva possedere quattro capacità [9]:

- natural language processing: la capacità di comunicare attraverso il linguaggio umano.
- knowledge representation: la capacità di immagazinare la "conoscenza"
- automated reasoning: la capacità di rispondere alle domande e trarre conclusioni.
- machine learning: la capacità di adattarsi a nuove circostanze ed estrapolare pattern dai dati

Queste quattro discipline, unite alla computer vision e alla robotica, costituiscono ad oggi l'Intelligenze Artificiale (spesso abbreviata in AI dall'inglese Aritficial Intelligence). L'apprendimento automatico (traduzione di ML) è quindi una disciplina dell'AI che si occupa di far imparare ai computer, attraverso i dati, come risolvere problemi senza ricorrere alla programmazione esplicita [10].

Un tipo di modello di apprendimento automatico sono le reti neurali (NN da neural network in inglese). In modo analogo alle reti neurali biologiche, le NN artificiali sono un insieme di neuroni interconnessi tra di loro. I neuroni sono fondamentalmente funzioni matematiche che prendono in input uno o più valori e producono un output.

Le reti neurali sono comunemente utilizzate in contesti in cui i dati possono contenere rumore o errori, o in cui i modelli analitici tradizionali possono essere insufficienti. Alcuni esempi di applicazioni delle reti neurali sono i software di riconoscimento ottico dei caratteri (OCR), i sistemi di riconoscimento vocale, l'analisi finanziaria e meteorologica. Le neural network sono poi anche ampiamente utilizzate nel riconoscimento di immagini. Questa tesi si occupa di sviluppare una rete neurale in grado di riconoscere, attraverso l'analisi di immagini contenenti foto di globuli rossi, se in quest'ultimi è presente o meno il virus della malaria.

2.1 Struttura di una rete neurale

Per comprendere come una rete neurale funzioni è utile partire dagli elementi costitutivi di essa: i neuroni artificiali. Un primo esempio di neurone artificiale è il cosidetto Perceptron, sviluppato negli anni tra il 1950 e il 1960 (ad oggi tale modello di neurone artificiale è caduto in disuso, ma le caratteristche principali sono estendibili ai modelli più moderni). Un perceptron (percettrone in italiano) è un oggetto in grado di prendere diversi input binari e produrre un singolo output binario. I neuroni sono raccolti all'interno di strutture dette layer ed il collegamento di ciascun neurone con quelli dei layer precedenti o successivi viene garantito attraverso dei parametri che pesano l'importanza di ciascun input nel collegamento con lo specifico neurone.

Siano detti x_1,x_2,x_3,...,x_n input del perceptron e w_1,w_2,w_3,...,w_n pesi del perceptron [11] allora $\mathbf{x}$ e $\mathbf{w}$ saranno i vettori che hanno per componenti rispettivamente gli input e i pesi di un determinato neurone: $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\mathbf{w} = (w_1, w_2, \ldots, w_n)$. L' output del singolo neurone (indicato con a) potrà assumere valore 0 o 1 a seconda che la somma pesata sia maggiore o minore di un valore soglia detto bias indicato con b.

$$\text{a} = \begin{cases} 0 & \text{se } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1 & \text{se } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

Variando i valori dei pesi e dei bias è possibile ottenere diversi modelli di "decision-making".

Un tipo di distinzione tra layer, come rappresentato in figura 2.1, è tra input, output e hidden layer, a seconda della loro posizione all'interno della rete neurale.

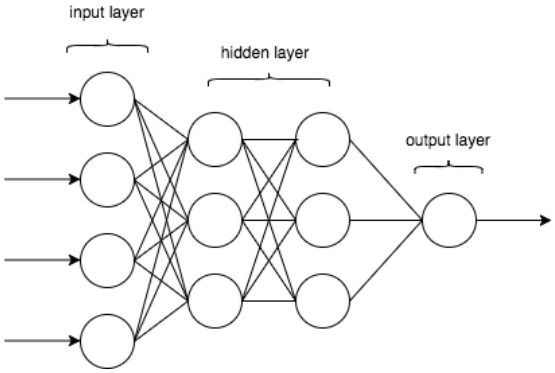


Figura 2.1: Rappresentazione delle layer che costituiscono una rete neurale

Non esiste una regola precisa per determinare il numero ottimale di layer o di neuroni in ciascun layer per risolvere un generico compito. La scelta di questi parametri è un processo sperimentale.

Il problema delle reti neurali contenenti percettroni è che una piccola modifica nei pesi o nei bias di un singolo percettrone può causare un cambiamento completo del suo output (facendolo passare da zero a uno). Questo può causare un cambiamento complesso nel comportamento del resto della rete, rendendo difficile capire come modificare gradualmente i pesi e i bias in modo che la rete apprenda [11]. Tale problema può essere aggirato aggiungedo la possibilità agli input di assumere valori

non più binari ma continui. Analogamente i singoli output potranno assumere valori continui attraverso l'applicazione di quella che è detta funzione di attivazione, ossia:

$$\mathrm{a} = f(\mathbf{w} \cdot \mathbf{x} + \mathbf{b}) \tag{2.1}$$

Si noti che nel caso di funzione di attivazione a gradino si avrebbe una rete di perceptron, in quanto gli output sarebbero zero o uno a seconda che $\mathbf{w} \cdot \mathbf{x} + \mathbf{b}$ sia positivo o negativo. L'importanza delle funzioni di attivazione nella costruzione delle reti neurali è dovuta alla possibilità di scegliere funzioni non lineari. La non linearità della funzione di attivazione è ciò che ci consente di costruire reti complesse. Ciò è garantito dal teorema di approssimazione universale il quale stabilisce che una rete con due layer di neuroni, il primo non lineare e il secondo lineare, è in grado di approssimare ogni funzione continua ad un arbitrario grado di accuratezza [9].

Tra le funzioni di attivazione più comuni ci sono la sigmoid-function (o funzione sigmoide) indicata con σ e la tangente iperbolica, mentre la funzione ReLU (Rectified Linear Unit) è considerata una delle più performanti e ampiamente utilizzate nelle reti neurali in quanto la sua derivata è semplice da calcolare, il che rende l'allenamento della rete più efficiente. Un'altra funzione interessante è la funzione softplus, simile alla ReLU ma differenziabile in ogni punto. La figura (2.2) mostra esempi delle funzioni di attivazione discusse.

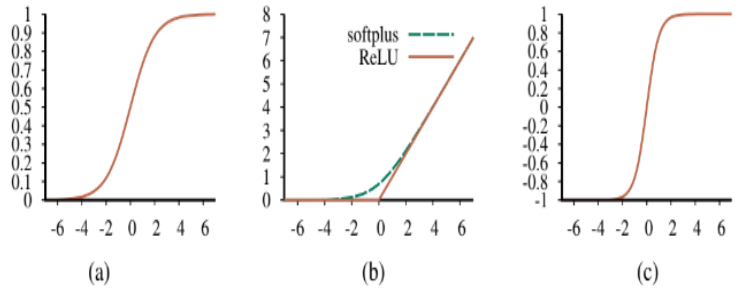


Figura 2.2: Funzioni di attivazione più comuni: (a) la funzione sigmoide; (b) la funzione ReLU e la funzione softplus; (c) la funzione tanh.

2.2 Addestramento delle reti neurali

Per gli scopi di questo lavoro di tesi, consideriamo il tipo di apprendimento detto "supervisionato". In questo contesto, l'addestramento di una rete neurale (training) è la fase in cui la rete neurale viene esposta ai dati di training, ovvero un insieme di dati etichettati utilizzati per allenare la rete e aggiornare i suoi parametri (training input). Questa è la fase in cui effettivamente la rete impara. Al training segue una seconda fase: la validation. La validation (validazione) è la fase in cui la performance della rete neurale viene valutata su un insieme di dati separato, che non fa parte dei dati di training. L'obiettivo della validazione è quello di valutare le prestazioni della rete neurale su dati "nuovi" e non visti durante il training allo scopo di ottimizzare l'architettura ed il numero di parametri della rete. Infine esiste una terza fase: la fase di test, in cui la rete neurale viene testata su un insieme di dati completamente separato, che non fa parte né dei dati di training né dei dati di validazione. Tale

fase serve per fare delle prove pratiche sul modello già finito e allenato. I dataset utilizzati nelle tre fasi sono detti rispettivamente training set, validation set, test set [12].

Per stabilire quanto una rete sia ben allenata è quindi necessario definire una cost function (o loss function): una funzione che quantifichi quanto le previsioni della rete siano buone.

Si noti che nel resto della sezione x rappresenta i singoli training input (ad esempio nel caso di addestramento di una rete neurale per il riconoscimento di numeri scritti a mano, x potrà essere un'immagine contenente il numero 6) e che il vettore $\mathbf{x}$ rappresenta il vettore contenente tutti i training input.

Un esempio semplice di cost function è la quadratic cost function (utilizzata come esempio in tutto il capitolo):

$$C(\mathbf{w}, \mathbf{b}) \equiv \frac{1}{2n} \sum_{i=1}^{n} ||\mathbf{y}(x_i) - \mathbf{a}(x_i)||^2 \tag{2.2}$$

Dove x_i rappresenta i singoli training input, mentre $\mathbf{y}(x_i)$ è il vettore di output della rete quando x_i è il training input. La somma è su tutti gli n elementi del training set. Il vettore $\mathbf{a}$ contiene l'output di riferimento, ossia l'obiettivo che la rete deve raggiungere dato x_i, infine n è il numero di training (o validation) input.

Nella cost function d'esempio quando $\mathbf{y}(x) \simeq \mathbf{a}$ si avrà $C(\mathbf{w}, \mathbf{b}) \simeq 0$. Al contrario quando $\mathbf{y}(x)$ differisce molto da $\mathbf{a}$, $C(\mathbf{w}, \mathbf{b})$ cresce. Obiettivo dell'algoritmo di apprendimento della rete sarà quindi quello di minimizzare la cost function, in altri termini trovare i parametri b e w tali per cui si ha un minimo della loss function. Per far ciò si può agire per via analitica, calcolando gradiente ed hessiana della cost function per trovarne il minimo della funzione. Tale processo risulta però complesso per funzioni a più variabili come nel caso delle reti neurali. Per questo motivo per trovare il minimo della funzione si preferisce usare quello che è detto algoritmo del gradiente. L'algoritmo del gradiente consiste nell'aggiornare i parametri verso la direzione di decrescita, tale direzione è ricavata dal gradiente: il vettore delle derivate parziali (rispetto ai w e b).

$$\nabla C(x) = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial b_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_n} \\ \frac{\partial C}{\partial b_n} \end{bmatrix} \tag{2.3}$$

Nell'addestramento delle reti neurali l'algoritmo utilizzato per calcolare il gradiente di C è l'algoritmo di backpropagation.

Per comprendere come funzioni l'algoritmo di backpropagation è utile usare la seguente notazione. Sia ω^l la matrice con entrate i valori di $\omega^l_{j,k}$, il peso della connessione tra il k-esimo neurone del layer (l-1) e il j-esimo neurone nel l-esimo layer . Sia $\boldsymbol{b^l}$ il vettore con entrate i bias del l-esimo layer e siano $\boldsymbol{a^l}$ e $\boldsymbol{in^l}$ l'analogo per i valori di attivazione e di input del l-esimo layer. Si avrà in notazione matriciale

per un fissato l:

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \dots & w_{n,m} \end{bmatrix} \begin{bmatrix} in_1 \\ in_2 \\ \vdots \\ in_m \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \right) \tag{2.4}$$

Osservando poi che l'input del layer l coincide con l'output del layer l-1 ovvero $\boldsymbol{in^l} = \boldsymbol{a^{l-1}}$ varrà che per un generico layer l:

$$\boldsymbol{a^l} = \sigma(\boldsymbol{w^l a^{l-1}} + \boldsymbol{b^l}) \tag{2.5}$$

espressione che mette in relazione i valori di attivazione del layer l con i valori di attivazione del layer (l-1). Per rendere più compatta la formula (2.5) si definisce il weighted input

$$\boldsymbol{z^l} \equiv \boldsymbol{w^l a^{l-1}} + \boldsymbol{b^l} \tag{2.6}$$

E quindi la (2.5) diventa

$$\boldsymbol{a^l} = \sigma(\boldsymbol{z^l}) \tag{2.7}$$

L'equazione (2.2) che definisce la loss function sarà riscrivibile come segue

$$C = \frac{1}{2n} \sum_{i=1}^{n} \left\| \mathbf{y}(x_i) - \boldsymbol{a^L}(x_i) \right\|^2 \tag{2.8}$$

dove L denota il numero di layer della rete e quindi $\boldsymbol{a^L}$ è il vettore di attivazione dell'output-layer.

L'Obiettivo dell'algoritmo di backpropagation è calcolare le derivate parziali rispetto ad ogni valore di w,b della loss function. Affinchè l'algoritmo di backpropagation funzioni serve che la funzione di costo C sia scrivibile come una media sulle funzioni di costo C_x relative ai singoli training input x. Questa condizione si verifica nella funzione di costo quadratica, in cui il costo per un singolo evento di training è espresso come $C_x = \frac{1}{2} \left\| \mathbf{y} - \mathbf{a}^L \right\|^2$ e non è difficile osservare che $C = \frac{1}{n} \sum_x C_x$. La ragione per cui questa assunzione è necessaria è che l'algoritmo di backpropagation permette di calcolare le derivate parziali $\frac{\partial C_x}{\partial w}$ e $\frac{\partial C_x}{\partial b}$ rispetto ai pesi w e ai bias b per un singolo traininig C_x. Successivamente grazie alla linearità della derivata è possibile ricavare $\frac{\partial C}{\partial w}$ e $\frac{\partial C}{\partial b}$.

Per comprendere come la variazione dei pesi e dei bias influenzi il cambiamento nella cost function si introduce l'errore del neurone j nel layer l

$$\delta_j^l \equiv \frac{\partial C_x}{\partial z_j^l} \tag{2.9}$$

Definita tale quantità è possibile dimostrare le 4 equazioni fondamentali della backpropagation utilizzando la chain rule [11]

- Equazione per l'errore nell'output layer, $\delta^{\mathbf{L}}$:

$$\delta^{\mathbf{L}} = \nabla_a C_x \odot \sigma'(\boldsymbol{z^L}) \tag{2.10}$$

  dove il simbolo $\odot$ indica l'Hadamard product anche detto element wise product

- Equazione per l'errore in $\delta^{\mathbf{l}}$ in termini dell'errore del layer successivo $\delta^{\mathbf{l+1}}$

$$\delta^{\mathbf{l}} = ((\boldsymbol{w}^{\boldsymbol{l+1}})^T \delta^{\mathbf{l+1}}) \odot \sigma'(\boldsymbol{z_l}) \tag{2.11}$$

- Equazione che mette in relazione errore con derivata rispetto a qualunque bias

$$\frac{\partial C_x}{\partial b_j^l} = \delta_j^l \tag{2.12}$$

- Equazione che mette in relazione errore con derivata rispetto a qualunque peso

$$\frac{\partial C_x}{\partial w_{jk}^l} = a_k^{(l-1)} \delta_j^l \tag{2.13}$$

Dalle 4 equazioni fondamentali della backpropagation è possibile scrivere l'algoritmo utile alla determinazione del gradiente: l'algoritmo di backpropagation. Gli step dell'algoritmo sono i seguenti:

1. Input: Si settano i valori di attivazione dell'input layer ovvero i valori corrispondenti alle entrate di $\boldsymbol{a}^{\mathbf{1}}$
2. Feedforward: Per ogni layer l=1,2,...,L si calcolano i weighted input $\boldsymbol{z}^{\boldsymbol{l}}$ e i valori di attivazioni $\boldsymbol{a}^{\boldsymbol{l}} = \sigma(\boldsymbol{z}^{\boldsymbol{l}})$
3. Output error: Si calcola il vettore $\delta^{\mathbf{L}}$ usando la formula (2.10)
4. Backpropagate the error: Per ogni layer l=L-1,L-2,...,1 si calcola $\delta^{\mathbf{l}}$ usando la (2.11)

Completato il quarto step le entrate del gradiente della funzione C_x sono facilmente determinate utilizzando la (2.13) e la (2.12).

Una volta calcolato il gradiente della loss function, ognuno dei parametri viene aggiornato scalandolo di un fattore proporzionale al gradiente della funzione. Tale fattore di proporzionalità è detto learning rate (LR). La scelta del learning rate è importante in quanto se il LR fosse troppo piccolo l'aggiornamento dei parametri sarebbe troppo lento e rischierebbe di convergere su minimi locali della funzione costo, se invece il LR fosse troppo grande si potrebbe non trovare il minimo della funzione. Il processo di aggiornamento dei parametri avviene in maniera iterativa all'interno di un training loop nel quale l'algoritmo analizza più volte l'intero dataset di training. Ciascuna iterazione è detta epoca.

2.3 Il framewrok Pytorch: un primo esempio

In questa tesi per lo sviluppo della rete neurale si utilizza Pytorch: una libreria di machine learning open source basata su Phyton [13]. PyTorch è una libreria che si basa sull'efficiente manipolazione di strutture tensoriali, che permette l'addestramento di reti neurali sfruttando, dove possibile, la parallelizzazione su GPU.

Uno dei principali vantaggi di PyTorch è la sua componente autograd, una componente di PyTorch che si occupa di gestire in modo automatico il calcolo del gradiente e l'esecuzione dell'algoritmo di backpropagation durante l'addestramento dei modelli di deep learning. In pratica, autograd tiene traccia delle operazioni eseguite sui tensori durante il forward pass della rete e, a partire da queste informazioni, calcola il gradiente della loss function rispetto ai parametri del modello. PyTorch offre inoltre anche il modulo torch.optim che consente di utilizzare diversi algoritmi di ottimizzazione per aggiornare automaticamente i parametri di un modello. Il LR è settato all'interno del torch.optim. È possibile scegliere l'algoritmo di ottimizzazione più adatto alle proprie esigenze. Esempi sono, l'algoritmo "SGD" e "Adam".

Per comprendere meglio è possibile studiare un semplice esempio (si veda figura 2.3).

```
3  # Definizione della rete neurale
4  class Net(torch.nn.Module):
5      def __init__(self):
6          super(Net, self).__init__()
7          self.weight = torch.nn.Parameter(torch.randn(1))
8
9      def forward(self, x):
10         return x * self.weight
11
12 # Definizione del dataset di esempio
13 x_data = torch.tensor([[1.0], [2.0], [3.0], [4.0]])
14 y_data = torch.tensor([[2.0], [4.0], [6.0], [8.0]])
15
16 loss_fn = torch.nn.MSELoss()
17
18 optimizer = torch.optim.SGD(myModel.parameters(), lr=0.01)
19
20 # Addestramento della rete neurale
21 myModel = Net()
22 for epoch in range(1000):
23     # Forward pass
24     y_pred = myModel(x_data)
25
26     # Calcolo della funzione di perdita
27     loss = loss_fn(y_pred, y_data)
28
29     optimizer.zero_grad()
30     #backward è uno dei principali metodi di autograd:
31     #computa il gradiente tramite backpropagation
32     loss.backward()
33     #applica il gradiente
34     optimizer.step()
```

Figura 2.3: Esempio di utilizzo dei moduli di Pytorch

In primo luogo, viene definita una semplice rete neurale utilizzando la classe Net che eredita dalla classe torch.nn.Module. La rete neurale ha un singolo neurone con un singolo parametro (il peso), che viene inizializzato casualmente nella funzione __init__() della classe. Successivamente, viene definito un dataset di esempio `x_data` e `y_data`. I valori di `x_data` sono gli input training definiti nella sezione precedente mentre i valori di `y_data` sono i valori di output attesi.

Viene poi definita la loss function utilizzata per valutare la bontà del modello durante l'addestramento. In questo caso, viene utilizzata la funzione Mean Squared Error (MSE) implementata dalla classe torch.nn.MSELoss(). Infine, viene definito un ottimizzatore (in questo caso, lo Stochastic Gradient Descent - SGD) che verrà utilizzato per aggiornare i pesi della rete neurale durante l'addestramento. All'interno del loop di addestramento, viene eseguito il forward pass utilizzando la rete neurale definita in precedenza (riga 24). Per ogni epoca nella variabile `y_pred` si salveranno gli output della rete. Viene poi calcolata la loss function (riga 27) con-

frontando gli output attesi con quelli ottenuti dalla rete. Successivamente, viene calcolato il gradiente della loss function rispetto ai parametri del modello, utilizzando il metodo backward() (riga 32, questo è un metodo di autograd). Il gradiente viene quindi utilizzato dall'ottimizzatore per aggiornare i pesi (riga 34).

2.4 Implementazione della rete neurale: le reti convoluzionali

Obiettivo di questa tesi è sviluppare una rete neurale in grado di riconoscere la presenza del Plasmodium (parassita della malaria) all'interno dei globuli rossi. Per fare ciò si addestra la rete neurale in modo supervisionato con un campione di immagini contenenti fotografie di globuli rossi al microscopio, sia sani che infetti. In tali foto i globuli rossi infetti sono distinguibili in quanto il parassita è colorato in viola grazie all'utilizzo del colorante Giemsa. Le immagini sono prese dal sito del National Institutes of Health (NIH). Un'immagine altro non è che una matrice contenente i valori dei pixel. Supponendo ad esempio di star analizzando immagini in scala di grigi 8k, si avrà una matrice 7680x4320 nella quale ogni entrata corrisponderà al valore di intensità di grigio associato all'immagine. Nel caso particolare di immagini 8k si avrebbero 3.3×10^7 input da gestire per singolo training input. Una rete neurale convoluzionale è particolarmente efficiente a gestire la mole di informazioni utilizzando una serie di particolari layer che implementano filtri (anche detti kernel). Queste reti sono in grado di apprendere caratteristiche "locali" (ossia che coinvolgono pixel vicini tra loro) come piccole forme, cambi di colore e di texture.

I layer principali di una CNN sono [14]:

- Convolutional Layer: il Layer Convoluzionale esegue un'operazione di convoluzione tra l'immagine di input e il kernel. Durante l'operazione di convoluzione, il kernel scorre sull'immagine di input e moltiplica ciascun pixel dell'immagine per i valori corrispondenti del kernel (hadamard product tra le matrici kernel e la porzione di matrice di pixel su quale il kernel sta scorrendo). Questo processo produce una nuova matrice, chiamata Convolved Feature (si veda figura 2.4). Il filtro scorre sull'immagine con passo fissato, detto *stride*.

- Pooling Layer: ha lo scopo di ridurre la dimensione della Convolved Feature. Esistono due tipi di Pooling Layer: Max Pooling e Average Pooling. Il primo restituisce il valore massimo della porzione dell'immagine coperta dal kernel, il secondo restituisce la media di tutti i valori della porzione dell'immagine coperta dal kernel.

In una CNN, l'informazione estratta da uno "stack" di più layer convoluzionali viene mappata da layer lineari in modo da minimizzare la funzione costo.

In figura viene mostrata un'implementazione della rete neurale utilizzata nel lavoro di tesi

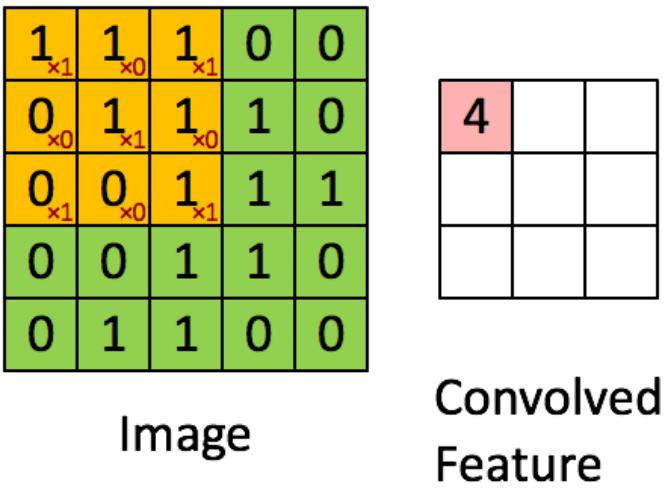


Figura 2.4: La matrice verde contiene i valori dei pixel, la matrice gialla è il kernel, a destra invece si ha la Convolved Feature

Capitolo 3

Sviluppo e test dei modelli

Al momento, i dispositivi quantistici si trovano nella cosiddetta era Noisy Intermediate-Scale Quantum (era NISQ), e come suggerisce il nome, possono eseguire circuiti quantistici con un numero limitato di qubit e di entanglement ma con tassi di errore ridotti. Questi dispositivi, attualmente composti da circa 10-100 qubit, possono essere utilizzati per ricercare il vantaggio quantistico, introdotto in sezione 1.1, con la limitazione però di limitare gli algoritmi a pochi qubit. L'apprendimento automatico è spesso indicato come uno dei candidati promettenti a mostrare vantaggio quantistico [10]. Il Quantum Machine Learning (QML) nasce dall'esigenza di trovare algoritmi in grado di mostrare vantaggio quantistico e la necessità di ottimizzare l'efficacia delle reti neurali.

È utile osservare che nell'accezione più generale, il termine QML racchiude quattro tipologie di applicazioni, a seconda che l'algoritmo implementato sia classico o quantistico e che i dati stessi da elaborare siano di natura quantistica o meno. In questo lavoro di tesi si studia il caso di dati classici e computazione quantistica, tramite l'utilizzo di una rete neurale ibrida.

In generale in una rete neurale ibrida i dati in ingresso vengono elaborati da layer classici, poi vengono passati ad un layer quantistico che li elabora con operazioni quantistiche e infine ritornano a un layer classico per la fase di output (si veda figura 3.1).

3.1 Rete neurale ibrida

Per costruire una rete neurale convoluzionale ibrida si parte dalla costruzione di una rete neurale classica, in cui i dati in input vengono elaborati da una serie di layer nascosti. All'interno della rete fra due layer lineari, viene insierito un layer quantistico. Il layer quantistico è costituito da un circuito quantistico parametrico, ovvero un insieme di operazioni che regolano la dinamica di uno stato multi-qubit (come spiegato nella sezione 1.3), definito da due diversi set di parametri. Il primo set x corrisponde all'output del layer lineare precedente, mentre il secondo θ è un insieme di pesi che vengono addestrati durante la fase di backpropagation descritta nella sezione 2.2 tramite il metodo di discesa del gradiente, allo stesso modo di tutti gli altri parametri della rete.

Come già introdotto nella sezione 1.3, i gate su singolo qubit possono essere interpretati come rotazioni del raggio vettore sulla sfera di Bloch. I parametri del layer quantistico sono quindi legati agli angoli di queste rotazioni [15]. Tuttavia, è im-

portante sottolineare che questa interpretazione è corretta solo se non sono presenti gate che implementano entanglement, ovvero se tutti i qubit del circuito evolvono in modo indipendente gli uni dagli altri. In caso contrario, la rappresentazione sulla sfera di Bloch non è più possibile e la corrispondenza tra parametri e rotazioni sussiste (in modo non più intuitivo) solo nello spazio di Hilbert dello stato entangled.

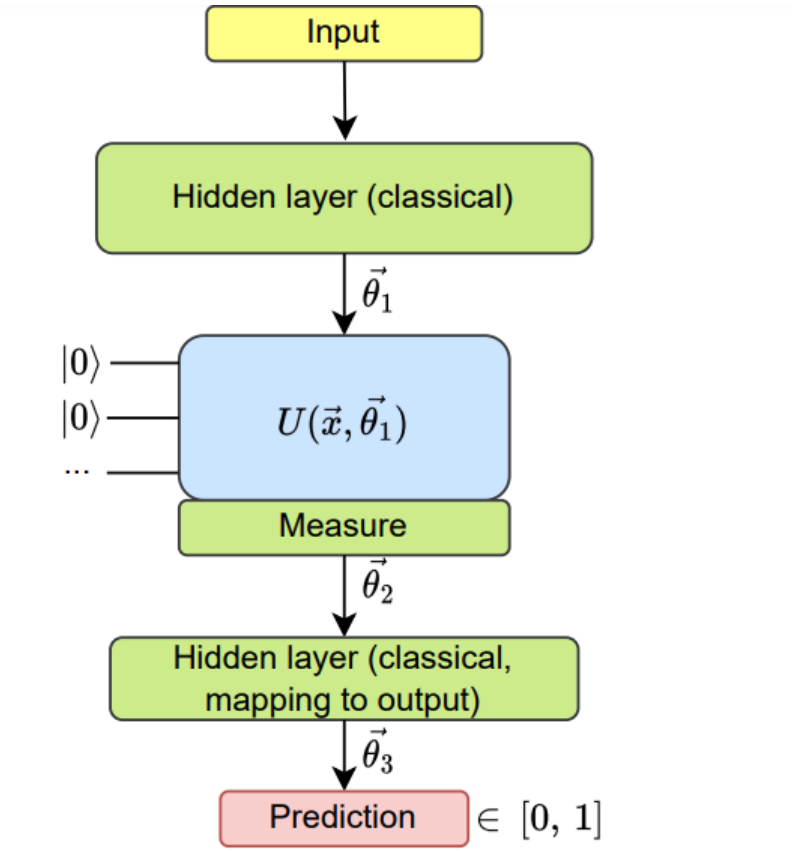


Figura 3.1: Rappresentazione schematica del funzionamento di una rete neurale ibrida

Qiskit mette a disposizione la libreria QiskitMachineLearning per implementare algoritmi di apprendimento automatico quantistico. Una volta implementati i circuiti quantistici questi vengono testati tramite l'utilizzo di del simulatore statement vector della piattaforma qiskit. In particolare, per creare reti neurali ibride in questa tesi è stata utilizzata la classe EstimatorQNN della libreria appena citata.

Nella figura (3.2) viene mostrato il codice utilizzato per creare la rete neurale classica.

```
1  class Net(Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          # Defining layers that are specific of the CNN
5          self.cnn_layers = Sequential(
6              # Defining a 2D convolution layer
7              Conv2d(1, 5, kernel_size=5, stride=3, padding=4),
8              BatchNorm2d(5), # A normalization layer
9              ReLU(inplace=True), # The activation funcion (ReLU)
10             MaxPool2d(kernel_size=2, stride=2), # Pooling layer
11             # Defining another 2D convolution layer
12             Conv2d(5, 5, kernel_size=3, stride=1, padding=1),
13             BatchNorm2d(5),
14             ReLU(inplace=True),
15             MaxPool2d(kernel_size=2, stride=2),
16         )
17         # Defining a linear layer of neurons
18         self.linear_layers = Sequential(
19             Linear(125, 10),
20             ReLU(inplace=True),
21             Linear(10, 2),
22         )
23         self.qnn = TorchConnector(qnn)
24         self.linear_layers2 = Sequential(
25             Linear(1,1)
26         )
```

Figura 3.2: Codice modello classico

Alla riga 27 viene inserito il circuito quantistico creato tramite l'utilizzo del linguaggio Qiskit.

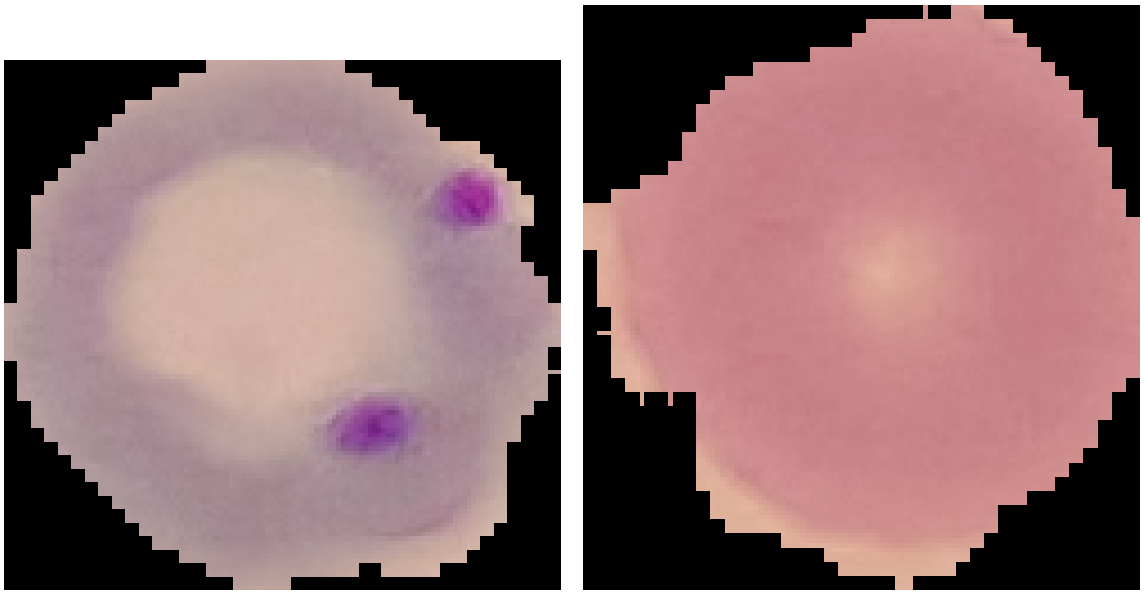
In figura (3.3) si mostra un esempio commentato sul caso standard RR + ZA (circuito quantistico spiegato meglio nella sezione successiva) per comprendere come si costruiscono le quantum neural network (QNN) utilizzando EstimatorQNN. Alla riga 12 si crea l'oggetto QNN, all'interno del quale sono definiti i parametri da allenare (weight_params) e i parametri di input (input_params). Questi ultimi sono i parametri di output del layer classico precedente, che verranno utilizzati come angoli di rotazione.

```
1  from qiskit.circuit.library import RealAmplitudes, ZZFeatureMap
2  from qiskit_machine_learning.neural_networks import EstimatorQNN
3  from qiskit import QuantumCircuit
4  def create_qnn():
5      feature_map = ZZFeatureMap(2)
6      ansatz = RealAmplitudes(2, reps=1)
7      qc = QuantumCircuit(2)
8      qc.compose(ansatz, inplace=True)
9      qc.compose(feature_map, inplace=True)
10     # remember to set input_gradients=True for
11     # enabling hybrid gradient
12     qnn = EstimatorQNN(
13         circuit=qc,
14         input_params=feature_map.parameters,
15         weight_params=ansatz.parameters ,
16         input_gradients=True,)
```

Figura 3.3: Codice per creare oggetto qnn

3.2 Dataset

Nel presente lavoro di tesi è stata sviluppata una rete neurale convoluzionale capace di individuare la presenza del virus della malaria in una cellula di globulo rosso. Per raggiungere tale scopo, la rete è stata addestrata su un set di immagini di cellule prese al microscopio, dove il virus era stato evidenziato attraverso l'impiego del colorante di Giemsa. Tali immagini sono state ottenute dal sito del National Institutes of Health (NIH) ed è possibile trovarle nel dataset "Cell Images for Detecting Malaria". Il dataset in questione è composto da circa 27.000 immagini, le quali presentano una



(a) Immagine di cellula infetta (b) Immagine di cellula sana

Figura 3.4

risoluzione approssimativamente di 140 x 140 pixel, tali immagini sono a colori e in formato JPEG. Dalle 27.000 immagini presenti nel dataset sono state selezionate in modo casuale 800 immagini per la fase di training, 200 per quella di validazione e 400 per quella di test. Al fine di migliorare le prestazioni delle reti neurali, si è deciso di ridurre la risoluzione delle immagini e di trasformarle in scala di grigi. Questa scelta è motivata dalla necessità di ridurre il tempo di addestramento del modello. Infatti, addestrare una rete neurale ibrida richiede (con gli strumenti oggi a disposizione) molto più tempo di una rete interamente classica. Si noti inoltre che al fine di semplificare il dataset si è deciso di ridurre la risoluzione delle immagini ad una risoluzione di 64 x 64 pixel e di trasformarle in scala di grigi.

3.3 CNN classica

Vengono ora presentati i risultati ottenuti con la rete neurale classica, utili poi per il confronto con tutte le reti neurali ibride implementate. La rete neurale classica utilizzata in questo lavoro di tesi è costituita da 2 layer convoluzionali e 2 layer di pooling (figura 3.2). Come evidenziato nella sezione 2.2 l'ottimizzazione dell'architettura e del numero di neuroni nella rete è un processo sperimentale.

Uno degli obbiettivi che si volevano ottenere durante l'ottimizzazione della rete neurale classica era quello di evitare l'overfitting. L'overfitting è un comportamento indesiderato della rete neurale (o più in generale di modelli di ML) che consiste nell'apprendere caratteristiche eccessivamente specifiche del particolare dataset di training a discapito della capacità di generalizzazione, ossia fornire previsioni attendibili di nuovi dati non appartenenti all'insieme di training. Per monitorare l'overfitting durante l'apprendimento della rete, è utile ricorrere al dataset di validation, valutando ad ogni epoca la discrepanza tra le performance di training e di validation.

Si tenga presente che per ridurre l'overfitting si possono applicare i seguenti accorgimenti (nel lavoro di tesi sono stati applicati i primi due, e testato il terzo con scarsi risultati) [16]:

- La semplificazione del modello: eliminare neuroni e/o layer può portare a una diminuzione dell'overfitting

- Early stopping: interrompere l'addestramento quando le performance sul set di validazione smettono di migliorare

- Aggiunta di layer di dropout: tecnica che consiste nell'eliminare casualmente alcuni nodi del modello durante l'addestramento, ciò riduce la possibilità che alcuni nodi siano troppo dipendenti da altri

- Aumento dei dati: dati di addestramento aggiuntivi aiutano il modello a imparare caratteristiche più generali. Di contro, aumentare il numero di elementi di training comporta un costo computazionale maggiore

Si riportano in tabella i parametri (descritti nelle sezioni precedenti) utilizzati in questo esperimento e in tutti quelli successivi :

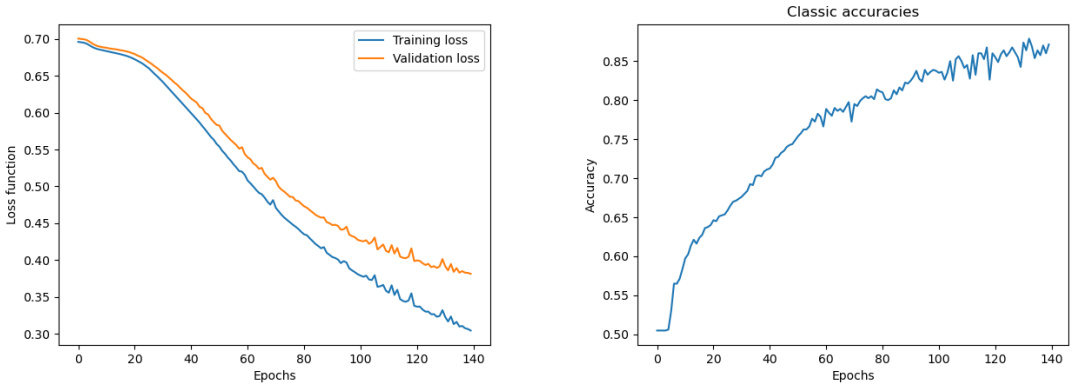
Tabella 3.1: Parametri dell'addestramento

LR	Optimizer	Loss function	Train-set	Val-set	Test-set	N epochs
0.0001	Adam	Binary Cross Entropy	800	200	400	140

Questa configurazione è risultata essere il miglior compromesso tra le risorse computazionali a disposizione (in quanto il training del modello ibrido richiede più tempo del classico) e performance ottenute.

La metrica di riferimento utilizzata per valutare la performance della rete neurale è l'accuracy (accuratezza), definita come la frazione di previsioni corrette sul totale. In altre parole, l'accuracy indica la percentuale di esempi nel set di dati che sono stati correttamente classificati dalla rete neurale rispetto al numero totale di esempi nel set di dati.

Nella figura 3.5, sono riportati i grafici che mostrano l'andamento dell'accuracy e della funzione costo durante l'addestramento della rete neurale classica. Analizzando il grafico 3.5(b), si può notare che l'accuracy di training non ha raggiunto il valore di convergenza. Questo comportamento è attribuibile alla scelta di limitare a 140 il numero di epoche di addestramento, che è stata effettuata per evitare l'overfitting utilizzando la tecnica dell'early stopping.



(a) Curve training loss e validation loss

(b) Curva training accuracy

Figura 3.5

In tabella 3.2 vengono riportati i valori delle accuracy finali per il training, la validation e il test del modello classico. La discrepanza tra l'accuracy di validation e di test può essere spiegata dal fatto che il modello è stato ottimizzato in modo da massimizzare l'accuratezza su un set di validation molto ridotto (200 immagini).

Tabella 3.2: Accuracy finali per modello classico

Training accuracy	Validation accuracy	Test accuracy
0.8712	0.840	0.7470

3.4 Circuiti a 2 qubit

In questa sezione, con lo stesso approccio di attenzione all'overfitting utilizzato per il modello classico, sono stati definiti circuiti quantistici a due e tre qubit e diverse reti neurali ibride sono state addestrate su di essi. La scelta del circuito quantistico da utilizzare è stata basata su considerazioni euristiche, poiché non esiste al momento una regola universale per stabilire quale circuito sia più adatto per un determinato problema di QML.

3.4.1 Modello RA + ZZ (Modello 1)

Come primo circuito quantistico implementato, si è deciso di partire da un circuito già utilizzato in altre applicazioni per la classificazione dei dati binari [17]. Real Amplitude è composto da gate R_y e CNOT-gate (descritti in sezione 1.3) alternati tra loro [18].

Il circuito ZZ Feature Map è composto da gate R_z e CNOT-gate alternati tra loro, e corrisponde a un caso specifico di Pauli Feature maps [17].

In generale, i circuiti Real Amplitude e ZZ Feature Map possono essere ripetuti più volte consecutivamente. In figura 3.6 è mostrato un esempio di Real Amplitude a due ripetizioni su tre qubit.

In figura 3.7 si mostra un esempio di Feature Map nel caso di una ripetizione per tre qubit (R_Z ed $U1$ possono essere usati in modo interscambiabile per eseguire rotazioni di fase su un singolo qubit)

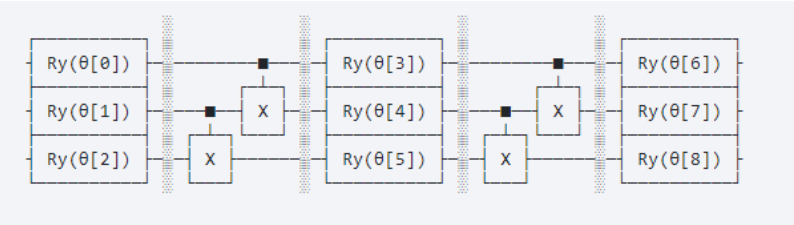


Figura 3.6: Circuito Real Amplitude

In figura 3.8 si mostra una rappresentazione grafica del circuito prodotta usando le librerie di qiskit per il circuito RA + ZZ.

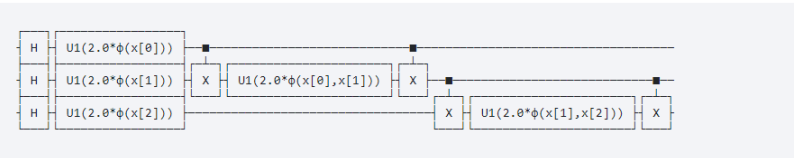


Figura 3.7: Circuito ZZ Feature Map

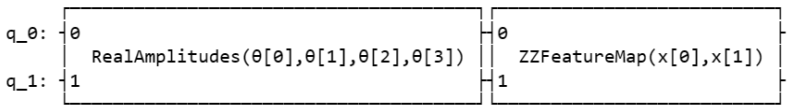


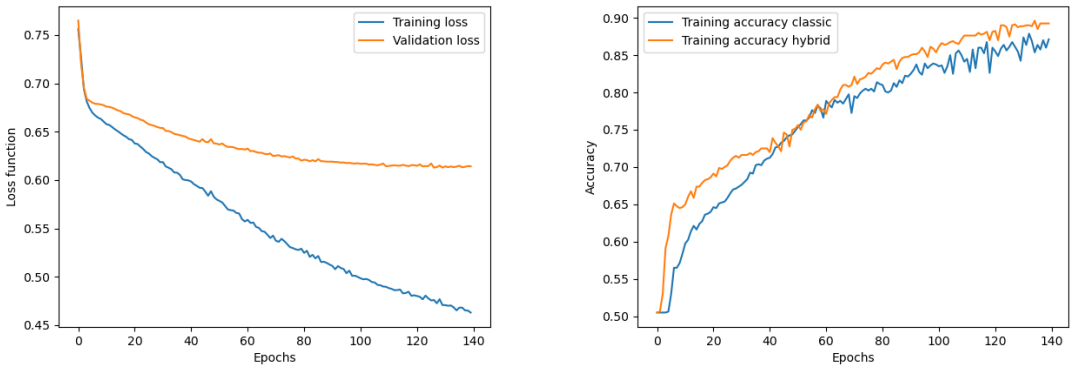
Figura 3.8: Rappresentazione grafica del circuito del modello 1

In figura 3.9, si riportano i grafici relativi al modello in studio.

Il primo grafico 3.9(a) mostra l'andamento della funzione di loss valutata sul set di training e sul set di validation al variare delle epoche. Il secondo grafico 3.9(b) confronta le accuracies del training tra il modello classico e il modello ibrido in analisi. Dal primo grafico, si nota come questo modello tenda a overfittare già dopo circa venti epoche. Nel secondo grafico, è possibile osservare come il modello ibrido RA + ZZ si comporti particolarmente bene per un basso numero di epoche (minore di 50). In tabella 3.3, vengono riportati i valori delle accuracy finali per il training, la validation e il test. Ciò che si può notare è che l'accuracy sul training risulta simile a quella del modello classico, mentre per la validation e il test si hanno valori nettamente inferiori rispetto a quelli ottenuti dal modello classico.

3.4.2 Modello 2

Il secondo circuito implementa il gate U3 (rotazione 3D), facendolo agire sul primo qubit del circuito (figura 3.10). Questo gate è dipendente da tre parametri che corrispondono agli angoli di Eulero. Gli input parameters, come nell'esempio precedente, sono inseriti nella componente ZZFeatureMap del circuito. Tale gate risulta



(a) Curve training loss e validation loss (b) Curve classic accuracy e hybrid accuracy

Figura 3.9: Grafici modello 1

Tabella 3.3: Accuracy finali per il modello 1

Training accuracy	Validation accuracy	Test accuracy
0.8920	0.6550	0.640

importante da studiare in quanto, come visto nella sezione 1.3 qualunque gate quantistico agente su un singolo qubit può essere rappresentato da U3 opportunamente parametrizzato.

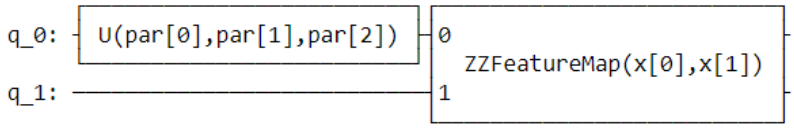


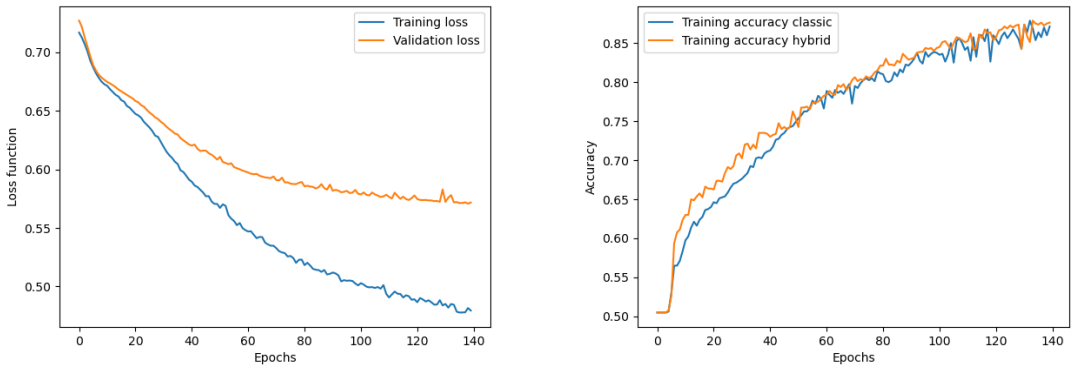
Figura 3.10: Rappresentazione grafica del circuito del modello 2

In figura 3.11(a) si confrontano le curve di loss della validation e del training. Come nell'esempio precedente, l'overfitting è presente già dopo poche epoche. Per quanto riguarda l'accuracy del modello ibrido, in questo caso le accuratezze risultano essere sempre migliori del modello classico.

In tabella 3.4 vengono riportate le accuracy finali : si nota un deciso miglioramento rispetto al modello ibrido RA + ZZ per quanto riguarda i valori sul validation e sul test che in questo caso risultano entrambi essere competitivi con il modello classico.

Tabella 3.4: Accuracy finali per il modello 2

Training accuracy	Validation accuracy	Test accuracy
0.8775	0.780	0.7475



(a) Curve training loss e validation loss (b) Curve classic accuracy e hybrid accuracy

Figura 3.11: Grafici modello 2

3.4.3 Modello 3

Nel terzo circuito si è deciso di testare il caso con entanglement. Il circuito è costituito da due gate di rotazione, R_x agente sul primo qubit e R_y agente sul secondo qubit. Viene poi aggiunto il CNOT-GATE che agisce tra i due qubit (figura 3.12). Si noti inoltre che questo circuito dipende da due soli weight-parameters.

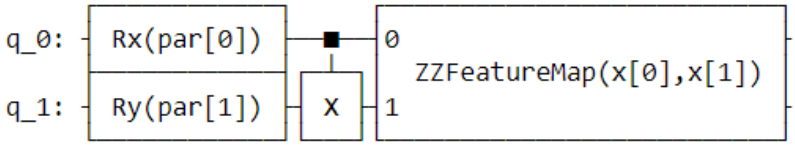
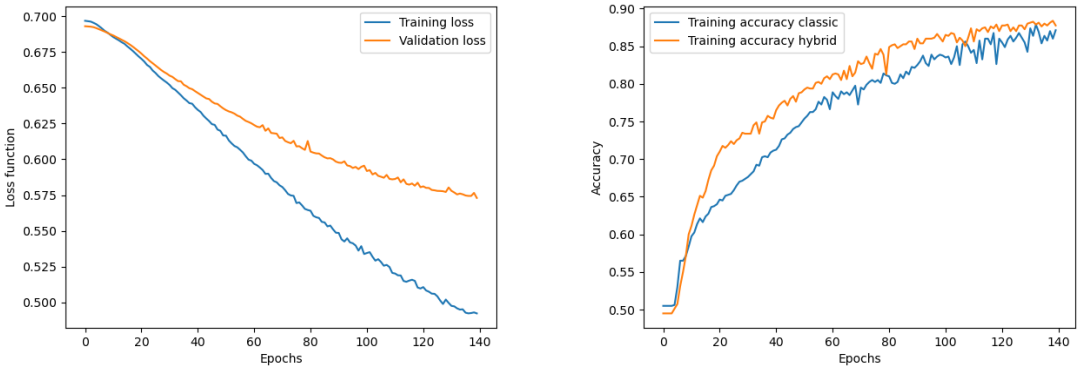


Figura 3.12: Rappresentazione grafica del circuito del modello 3

I grafici relativi a questo modello sono presentati in figura 3.13. Analogamente ai primi due casi si osserva overfitting e una accuracy maggiore del modello ibrido rispetto a quello classico al variare delle epoche. In tabella 3.5 vengono riportati i valori di accuracy finale.



(a) Curve training loss e validation loss (b) Curve classic accuracy e hybrid accuracy

Figura 3.13: Grafici modello 3

Tabella 3.5: Accuracy finali per il modello 3

Training accuracy	Validation accuracy	Test accuracy
0.8770	0.7150	0.730

3.4.4 Modello 4

Nel quarto circuito si è voluto testare se i risultati potessero variare all'aumentare del numero di gate che implementano entanglement. In questo circuito sono stati implementati due gate che costituiscono una generalizzazione del CX gate rispetto a quanto descritto nella sezione 1.3: il CRY gate ed il CRX gate. Analogamente al CX, questi ultimi implementano rotazioni Ry ed Rx condizionate dal qubit di controllo.

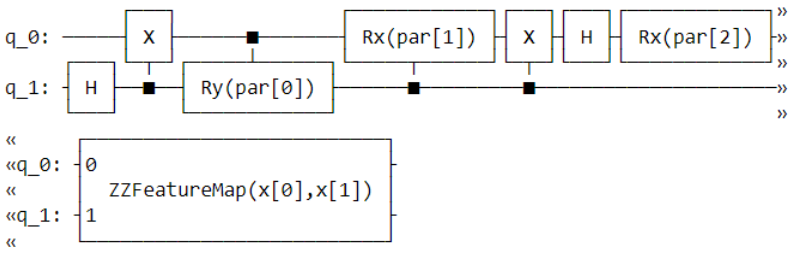
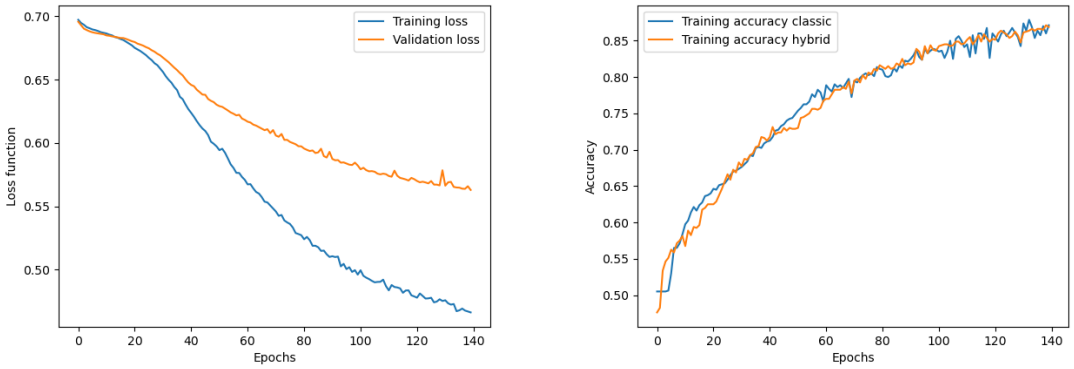


Figura 3.14: Rappresentazione grafica del circuito del modello 4

Come mostrato in figura 3.15(a),anche in questo caso si evidenzia overfitting. In figura 3.15(b) si osserva che le due curve di apprendimento per il modello 4 e il modello classico sono confrontabili. I valori di accuracy per validation e test (tabella 3.6) inoltre non evidenziano miglioramenti apprezzabili rispetto ai modelli con meno entanglement.



(a) Curve training loss e validation loss (b) Curve classic accuracy e hybrid accuracy

Figura 3.15: Grafici modello 4

Tabella 3.6: Accuracy finali per il modello 4

Training accuracy	Validation accuracy	Test accuracy
0.8690	0.740	0.7325

3.5 Circuiti a 3 qubit

3.5.1 Modello RA + ZZ a 3 qubit (Modello 5)

In questo caso si è applicato il modello RA + ZZ descritto nella sezione 3.4.1 a tre qubit.

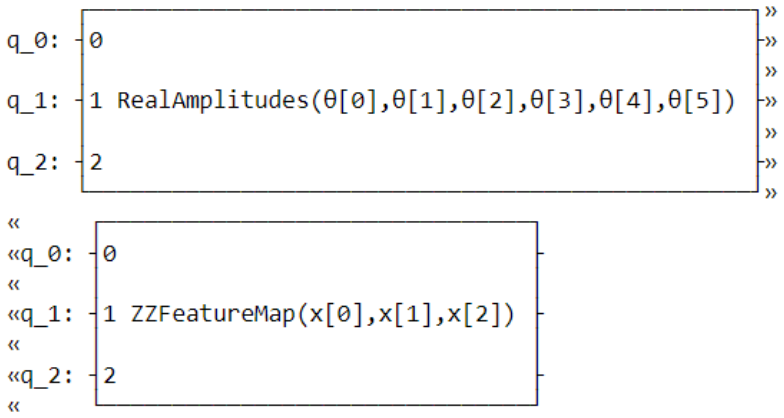
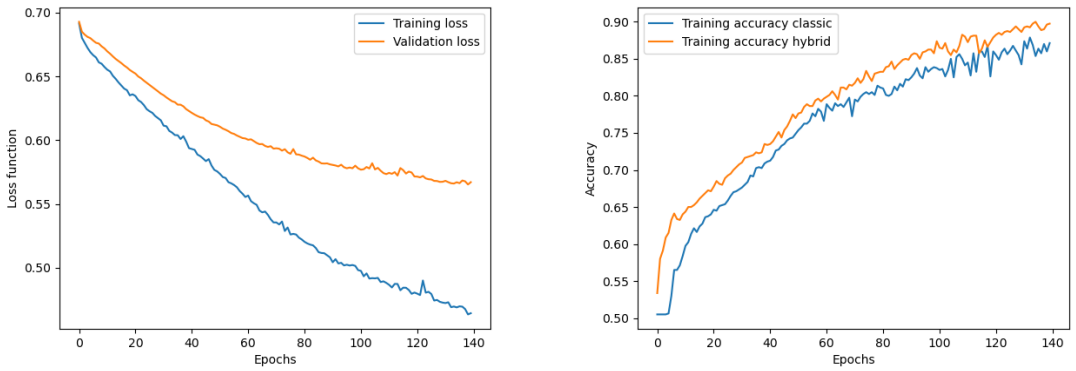


Figura 3.16: Rappresentazione grafica del circuito del modello 5



(a) Curve training loss e validation loss (b) Curve classic accuracy e hybrid accuracy

Figura 3.17: Grafici modello 5

Confrontando il modello 5 con il modello 1 (RA + ZZ), non si notano miglioramenti significativi riguardo all'overfitting, come mostrato nella figura 3.17(a). Tuttavia, i valori di accuratezza ottenuti sui dati di validazione e test sono migliori, come indicato nella tabella 3.7. Inoltre, entrambi i valori di accuratezza sono confrontabili con quelli del modello classico.

Tabella 3.7: Accuracy finali per il modello 5

Training accuracy	Validation accuracy	Test accuracy
0.8975	0.730	0.7350

3.5.2 Modello 6

Nel modello 6 è stato testato un circuito quantistico a tre qubit composto da un gate R_y sul primo e terzo qubit e un gate R_x sul secondo qubit (figura 3.18).

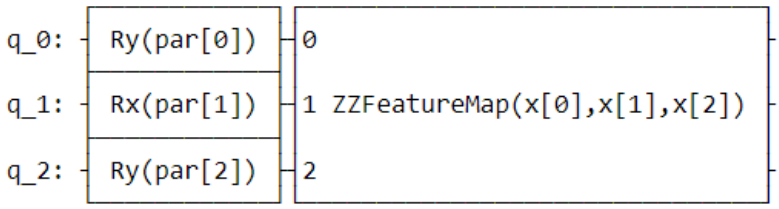
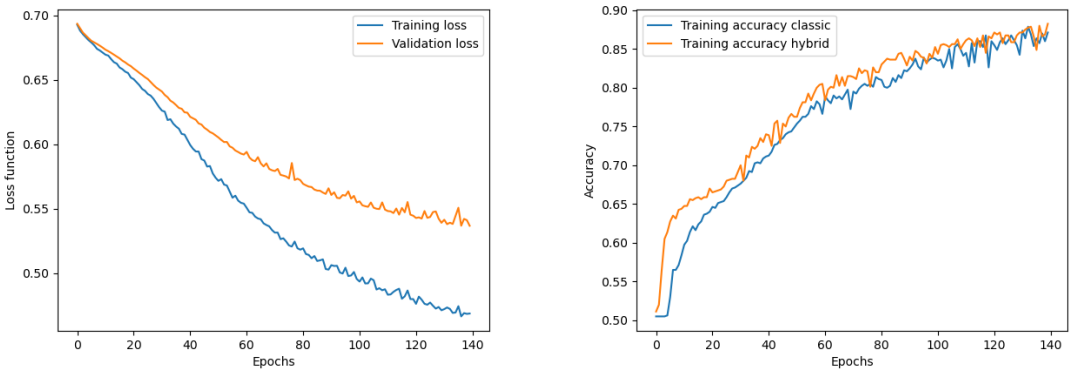


Figura 3.18: Rappresentazione grafica del circuito del modello 6

I grafici rappresentati in figura 3.19 non presentano caratteristiche distintive rispetto ai modelli precedenti, in quanto si osserva anche qui un certo livello di overfitting. Tuttavia, per quanto riguarda i valori di accuracy sulla validation e sul test, il modello 6 presenta i risultati migliori tra tutti i modelli testati, come riportato nella tabella 3.8. In particolare, l'accuracy sul test risulta essere migliore anche dell'accuracy sul test del modello classico. È importante tuttavia considerare le fluttuazioni statistiche sulla metrica di accuracy, dovuta al dataset di piccole dimensioni.



(a) Curve training loss e validation loss (b) Curve classic accuracy e hybrid accuracy

Figura 3.19: Grafici modello 6

Tabella 3.8: Accuracy finali per il modello 6

Training accuracy	Validation accuracy	Test accuracy
0.8250	0.785	0.7750

Capitolo 4

Commento ai risultati e conclusioni

4.1 Commento ai risultati e conclusioni

Il presente lavoro di tesi aveva l'obiettivo di sviluppare e confrontare le capacità di una rete neurale convoluzionale ibrida rispetto a quelle di una rete neurale convoluzionale classica. Inizialmente si è posta particolare attenzione all'ottimizzazione della rete classica per limitare l'overfitting. Tale processo di ottimizzazione ha dovuto tener conto della limitazione del dataset di training a soli 800 immagini, limitazione dovuta alle risorse computazionali disponibili.

Il primo modello analizzato è stato il Real Amplitudes + ZZ Feature Map (RA + ZZ), già noto in letteratura ed utilizzato nella classificazione binaria. Successivamente sono stati sviluppati cinque modelli ibridi, seguendo un approccio sperimentale in quanto non esiste una regola universale per stabilire a priori quale modello sia il migliore.

I risultati di accuracy del modello RA + ZZ sono stati superati da tutti i cinque modelli ibridi successivi. I cinque modelli ibridi proposti in questo lavoro di tesi hanno fornito risultati competitivi con quelli del modello classico sulle accuratezze finali (di validation, test, training). In particolare, uno dei modelli si è dimostrato essere più efficace del modello classico per quanto riguarda l'accuratezza sul dataset di test, anche se tale risultato non è conclusivo a causa delle fluttuazioni statistiche dovute al dataset di dimensione limitata.

Dai grafici è emersa una caratteristica comune a tutti e 6 i modelli, ovvero la presenza di overfitting già dopo poche epoche. Per quanto riguarda le capacità di apprendimento sul training set, i modelli ibridi si sono comportati meglio del modello classico per un basso numero di epoche. In particolare, tre dei sei modelli hanno dimostrato un'accuracy decisamente migliore del modello classico per un numero di epoche intorno a 20.

È importante sottolineare che lo studio delle accuratezze andrebbe esteso anche al validation set, al fine di stabilire se esiste un effettivo vantaggio quantistico per un basso numero di epoche.

I risultati dei modelli ibridi sono stati ottenuti utilizzando il simulatore statevector della piattaforma qiskit. Si assume che tali risultati rispecchino fedelmente quelli ottenibili utilizzando un moderno processore quantistico. Ciò è dovuto al fatto che i circuiti quantistici sviluppati e utilizzati per la simulazione sono di piccole dimen-

sioni, ossia comprensivi di un numero ridotto di qubit (due e tre) e di un limitato numero di porte logiche. Questi circuiti sono quindi adatti a dispositivi "noisy-scale intermediate", ossia dispositivi quantistici in cui si verifica una certa percentuale di errore nelle operazioni eseguite, generalmente nell'ordine dell'1% o inferiore.

Infine, di seguito vengono elencati alcuni aspetti attraverso i quali lo studio potrebbe essere approfondito e migliorato:

- Aumento del dataset di training: ciò avrebbe comportato una maggiore capacità di generalizzazione e, di conseguenza, una riduzione dell'overfitting di tutti i modelli.
- Aumento del numero di epoche fino alla convergenza ad un valore fissato della curva di apprendimento. Nel lavoro di tesi, questo non è stato possibile fare, poiché a causa del dataset limitato si è fatto ricorso all'early stopping.
- Ottimizzare ulteriormente le reti neurali: sarebbe interessante comprendere come i risultati variano al variare dei parametri come l'optimizer, il LR, la loss function.
- Considerando che il dataset è stato preprocessato e quindi alcune informazioni sono andate necessariamente perse (sezione 3.2), sarebbe interessante comprendere come ciò influisce sui risultati di accuratezza.

Bibliografia

[1] Sean Carroll. *Even Physicists Don't Understand Quantum Mechanics*. 2019. URL: https://www.nytimes.com/2019/09/07/opinion/sunday/quantum-physics.html.

[2] Micheal A. Nielsen e Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge, 2010, pp. 2–3.

[3] Masahito Hayashi. *Quantum Information: An Introduction*. Springer Berlin e Heidelberg, 2004.

[4] Paul Benioff. "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines". In: *Journal of Statistical Physics* (1980), pp. 563–591.

[5] Elias Fernandez-Combarro Alvarez. *A Practical Introduction to Quantum Computing: From Qubits to Quantum Machine Learning and Beyond*. 2020. URL: indico.cern.ch/event/970903/.

[6] Babbush R. et al. Arute F. Arya K. "Quantum supremacy using a programmable superconducting processor". In: *Nature 574, 505–510* (2019).

[7] Douglas Busvine. *Google claims 'quantum supremacy'; others say hold on a qubit*. 2019. URL: https://www.reuters.com/article/alphabet-quantum-idUSL5N278466.

[8] *Qiskit Documentation*. URL: https://qiskit.org/documentation/.

[9] Stuart Russell e Peter Norvig. *Artifical Intelligence: A Modern Approach*. Prentice Hall, 2020.

[10] Maria Schuld e Francesco Petruccione. *Machine Learning with Quantum Computers*. Springer, 2021.

[11] Michael Nielsen. *Neural Networks and Deep Learning*. 2019. URL: http://neuralnetworksanddeeplearning.com/index.html.

[12] Ignazio Barraco. "Valutazione delle prestazioni delle Reti Neurali Convoluzionali su immagini affette da distorsione". Politecnico di Torino, 2019. URL: https://webthesis.biblio.polito.it/13125/1/tesi.pdf.

[13] URL: https://pytorch.org/docs/stable/index.html.

[14] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[15] Maham Shafiq. *Quantum Machine Learning: Hybrid quantum-classical Machine Learning with PyTorch and Qiskit*. 2020. URL: https://medium.com/@mahamshafiq98/quantum-machine-learning-hybrid-quantum-classical-machine-learning-with-pytorch-and-qiskit-d03da758d58b.

[16] Abhinav Sagar. *5 Techniques to Prevent Overfitting in Neural Networks*. 2019. URL: https://www.kdnuggets.com/2019/12/5-techniques-prevent-overfitting-neural-networks.html.

[17] Temme K. et al. Havlíček V. Córcoles A. "Supervised learning with quantum enhanced feature spaces". In: *Nature 567, 209–212* (2019).

[18] *Qiskit Documentation Gates*. URL: https://qiskit.org/documentation/apidoc/circuit_library.html.