Università degli Studi di Padova

Dipartimento di Tecnica e Gestione dei Sistemi Industriali
Corso di Laurea Magistrale in Ingegneria Meccatronica

Tesi di Laurea Magistrale

Implementazione di una Rete Neurale su un Microcontrollore Industriale per la Stima del Modello Magnetico di Motori Sincroni

Relatore: Ing. Fabio Tinazzi

Laureando: Lorenzo Alessandro Parise
1237241

ANNO ACCADEMICO: 2021/2022

Implementazione di una Rete Neurale su un Microcontrollore Industriale per la Stima del Modello Magnetico di Motori Sincroni

Lorenzo Alessandro Parise

INDICE

ELENCO DELLE TABELLE

ELENCO DELLE FIGURE

INTRODUZIONE

A causa delle numerose ripercussioni negative sull'ambiente provocate dall'industrializzazione del secolo scorso, prima fra tutte il riscaldamento globale causato dagli elevati livelli di anidride carbonica emessi nell'atmosfera, la tendenza attuale è quella di privilegiare, in ogni ambito, soluzioni ad elevata efficienza. Allo stato attuale, circa il 45% della potenza elettrica prodotta a livello mondiale viene consumata da attuatori elettrici, di cui oltre il 90% sono motori trifase ad induzione. Data la robustezza e la facilità di controllo che li caratterizzano, spesso questi dispositivi sono collegati direttamente a rete, lavorando, soprattutto nella fase di avviamento, con un rendimento estremamente basso a causa delle elevate correnti assorbite. L'obiettivo è quello di sostituire questi dispositivi con azionamenti a basso costo ma ad elevata efficienza. Per alcuni anni il motore elettrico di riferimento è stato quello a magneti permanenti, comunemente detto "brushless", in grado di produrre coppie elevate con ingombri minimi. I materiali che lo compongono però, ovvero le terre rare, sono difficili da reperire e da estrarre, perciò hanno un costo elevato e un impatto ambientale non trascurabile. Per tale motivo risulta poco adatto a sostituire un motore robusto ed economico come quello asincrono trifase, limitando il suo ambito di utilizzo ad applicazioni di fascia alta o al settore dell'automotive.
La direzione intrapresa a livello globale è volta a prediligere l'impiego di attuatori in cui la coppia non venga interamente generata dal flusso dei magneti permanenti. Tra le soluzioni più consone per adempiere a tale scopo si trovano i motori a magneti interni e i motori a riluttanza, in cui è presente il contributo di coppia dato dall'anisotropia costruttiva del dispositivo. Il problema di queste tecnologie è che, al contrario dei motori a magneti permanenti superficiali, la cui riluttanza è simile all'aria, presentano un circuito magnetico soggetto a saturazione e un mutuo accoppiamento tra gli assi. A causa di ciò, la relazione flusso-corrente, fondamentale per un efficiente controllo del motore, risulta complicata da ricavare per la forte non linearità che la caratterizza.
La ricerca di tale relazione è un trade-off tra complessità progettuale, e, di conseguenza, costi da sostenere, e accuratezza nel controllo, la quale si traduce in una maggior efficienza energetica. Una prima soluzione è rappresentata dall'analisi agli elementi finiti. Il principale svantaggio di questa metodologia è il livello di accuratezza richiesto nella creazione del modello per ottenere un risultato attendibile, soprattutto per quanto riguarda materiali e dimensioni del motore analizzato. Una possibile alternativa è basarsi su misure sperimentali raccolte nel corso di vari test, spesso realizzati con l'ausilio di un banco di prova costruito ad-hoc. Su questo secondo approccio sono stati effettuati numerosi studi a livello globale, che hanno portato alla creazione di molteplici metodologie applicative. Una delle più recenti ed innovative è rappresentata dall'impiego di una rete neurale che, a partire dalle informazioni su tensioni, correnti e velocità raccolte durante il normale funzionamento del dispositivo, permette di ricavare una mappa del flusso magnetico che

caratterizza il motore con un approccio di tipo "black box", ovvero senza conoscere a priori la struttura con cui è realizzato.
Questa soluzione progettuale presenta numerosi vantaggi: innanzitutto la possibilità di lavorare con il sistema completo, motore con relativo inverter, nella configurazione in cui andranno ad operare. Questo permette di ottenere una caratteristica che includa tutte le non linearità e che sia particolarizzata per il singolo dispositivo, non genericamente per un campione di prodotti. Inoltre, non è necessario un banco di test, la prova può essere effettuata nel luogo d'installazione durante il normale funzionamento del sistema, oltre al fatto che può essere ripetuta nel tempo per controllare le condizioni operative del motore ed eventuali variazioni parametriche che possano essere sintomo di danneggiamenti.
Il principale svantaggio di questo approccio è che, solitamente, le reti neurali richiedono una certa potenza di calcolo per un training efficacie: spesso, infatti, si effettua una fase di raccolta dati "in-place" per poi ricorrere ad un calcolatore esterno, con prestazioni elevate, per addestrare la rete ed elaborarne l'output.
L'obiettivo della tesi è quello di implementare una rete neurale in grado di stimare, in modo sufficientemente accurato, il modello magnetico di un motore sincrono mediante l'utilizzo di un sistema di tipo industriale, una scheda a microcontrollore, caratterizzato da un costo di alcuni ordini di grandezza inferiore rispetto ad un elaboratore professionale.
Nella prima parte dell'elaborato si andranno a descrivere le principali tipologie di reti neurali e di algoritmi di addestramento, per meglio comprendere la natura delle scelte effettuate nelle varie fasi della ricerca. Nella seconda parte si andranno poi ad applicare le nozioni teoriche al caso in esame, riportando la struttura dei test eseguiti e i dati ottenuti, oltre alle principali criticità riscontrate. Infine, si procederà con una post-elaborazione dei risultati al fine di affinare quanto ottenuto per un controllo del motore più accurato.

1

INTRODUZIONE ALLE RETI NEURALI

Le reti neurali artificiali, comunemente abbreviate in *reti neurali*, sono una tipologia di algoritmi ad apprendimento automatico con una struttura ispirata a quella del cervello umano, con il quale condivide l'elevato numero di interconnessioni tra le celle computazionali elementari. Questa particolare caratteristica dona a tali algoritmi la straordinaria abilità di "imparare" dall'esperienza mediante la modifica delle interconnessioni tra i neuroni, quindi della struttura stessa con cui è realizzata la rete.
In questo capitolo si andranno a descrivere struttura e principali proprietà che accomunano gli algoritmi artificiali con la rete neurale per eccellenza, il cervello umano.

1.1 *Il cervello umano*

Il cervello umano è composto di decine di miliardi di unità interconnesse, i neuroni, da ciascuno dei quali si ramificano migliaia di terminazioni che li collegano tra loro e con il mondo esterno, formando percorsi di lunghezza variabile da pochi nanometri fino ad alcuni metri.

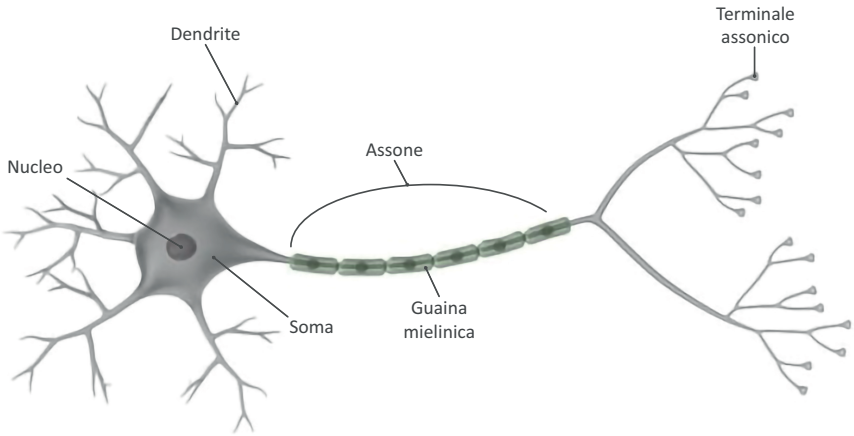


Figura 1.1. Struttura di un neurone.

Ogni neurone è composto da tre elementi fondamentali, come si evince in figura (1.1): un corpo cellulare, o soma, nel quale risiedono il nucleo e gli altri organelli deputati alle principali funzioni cellulari, i dendriti, delle terminazioni nervose che si ramificano attorno al corpo cellulare e che servono a trasportare l'impulso elettrico entrante (anche detto

stimolo) verso la cellula, e l'assone, una singola fibra nervosa di lunghezza relativamente elevata che veicola l'informazione uscente dal nucleo (anche detta *reazione*) verso gli altri neuroni collegati. Questi ultimi, in realtà, non sono posti direttamente a contatto tra loro, ma sono interfacciati da speciali membrane, le sinapsi.

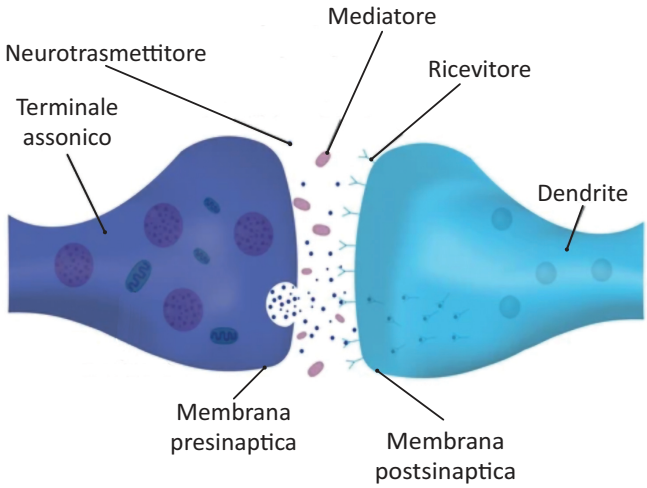


Figura 1.2. Struttura di una sinapsi [20a].

Come si evince in figura (1.2), ogni sinapsi è suddivisa in *membrana presinaptica*, appartenente al neurone trasmettente, e *membrana postsinaptica*, appartenente al neurone ricevente. Quando un segnale elettrico arriva nella sinapsi, viene rilasciato un "trasmettitore" che trasporta una sostanza detta *mediatore* dalla membrana presinaptica a quella postsinaptica, generando una variazione della permeabilità di quest'ultima. Se la somma algebrica dei potenziali in ingresso supera una determinata soglia, quest'ultimo si attiva e trasmette a sua volta l'impulso ai neuroni collegati.
In particolare, in ogni cellula è presente una sostanza ricca di potassio, che si accumula all'interno grazie alla struttura stessa della membrana, la quale ne permette il passaggio. All'aumentare del potenziale ai capi della sinapsi, tale membrana passa dall'essere permeabile agli ioni di potassio al prediligere gli ioni di sodio, presenti in maggiore quantità all'esterno del neurone. Quando queste particelle vengono iniettate nel nucleo, ne determinano l'attivazione.
Grazie a questo meccanismo, variando l'intensità dei singoli legami e il modo in cui sono connessi i corpi cellulari, si modifica la funzione della rete neurale. Alcune, fondamentali, sono definite già dalla nascita, altre si formano con l'esperienza, alterando via via la chimica delle interconnessioni tra i vari neuroni. Ed è proprio questa capacità di apprendere dall'esperienza, di "imparare per generalizzare", tipica del comportamento umano, che si vuole trasmettere alle reti artificiali.
Questo processo è reso possibile dalla comprensione del meccanismo con cui apprende il cervello umano: si è infatti notato che esperienze simili portano all'attivazione delle medesime sequenze di neuroni. Inoltre, la probabilità che due neuroni siano interconnessi decresce esponenzialmente all'aumentare della distanza tra i due. Sono perciò i collegamenti stessi che fungono da "memoria", permettendo agli esseri umani di adattarsi a nuove situazioni semplicemente applicando ciò che hanno già vissuto, con un livello

di *generalizzazione* che aumenta all'aumentare della distanza tra l'esperienza pregressa e quella in corso d'opera.
Vi è però una seconda fondamentale caratteristica del cervello umano che i moderni elaboratori digitali non sono in grado di emulare, ovvero il fortissimo livello di parallelismo con cui vengono eseguiti i calcoli. Un singolo impulso nervoso, infatti, contiene una quantità di informazioni ridottissima, assimilabile ad un bit, e viene processato dai vari neuroni in modo relativamente lento, in un tempo che mediamente si attesta sui 200 millisecondi. Replicando però il singolo impulso per le decine di miliardi di neuroni, interconnessi da fibre nervose che sommate arrivano ad un'estensione dell'ordine dei 10^{14} metri, si ottiene una capacità di calcolo inavvicinabile per qualsiasi dispositivo artificiale mai creato fino ad oggi.

1.2 *Le reti neurali artificiali*

Le reti neurali artificiali non si avvicinano nemmeno lontanamente alla complessità e all'estensione del cervello umano. Vi sono però alcuni aspetti chiave con i quali si cerca di emularne il comportamento. Innanzitutto la struttura con cui sono realizzate, che presenta un elevato numero di unità fondamentali fortemente interconnesse. In secondo luogo il fatto che la funzione della rete sia strettamente legata alle interconnessioni stesse tra i nodi. E proprio questa struttura "parallela" rende le reti neurali estremamente veloci nel trovare soluzione a problemi che un normale calcolatore digitale impiegherebbe anni per risolvere.
Se si analizzano inoltre le principali caratteristiche che accomunano tutte le reti artificiali, si possono individuare quattro parametri di classificazione: il tipo di neuroni utilizzati, l'architettura delle interconnessioni, l'algoritmo di apprendimento adottato, l'algoritmo di richiamo di cui la rete si serve per generare l'output.

1.2.1 Storia delle reti neurali

L'idea delle moderne reti neurali nacque a metà del secolo scorso con il lavoro di Warren McCulloch e Walter Pitts, rispettivamente neurofisiologo e matematico statunitensi, i quali modellizzarono il primo neurone artificiale, riportato in figura (1.3).
Come si nota, si compone principalmente di tre elementi: una serie di ingressi, o sinapsi, ciascuno caratterizzato da un proprio peso, un nodo sommatore, il cui valore iniziale può essere opportunamente traslato grazie all'ingresso di bias, e una funzione di attivazione φ, che definisce la forma dell'output in relazione agli ingressi. Quest'ultima può essere di vari tipi, come si andrà ad analizzare nel paragrafo (1.2.2). Sfruttando questo modello, McCulloch e Pitts dimostrarono come una rete di neuroni artificiali potesse, teoricamente, emulare ogni funzione logica o aritmetica esistente.
Pochi anni dopo, lo psicologo canadese Donald Hebb pose le basi per comprendere il meccanismo di apprendimento di ogni tipo di rete neurale, ovvero la creazione di interconnessioni durature tra i medesimi neuroni in presenza di stimoli uguali e ripetuti. Quella che viene ricordata come "La regola di Hebb", può essere riassunta nella frase "Le cellule che si attivano insieme si collegano insieme".
Queste teorie portarono alla creazione della prima rete neurale, il percettrone di Frank Rosenblatt. Tale rete si serviva dei neuroni artificiali teorizzati da McCulloch e Pitts per

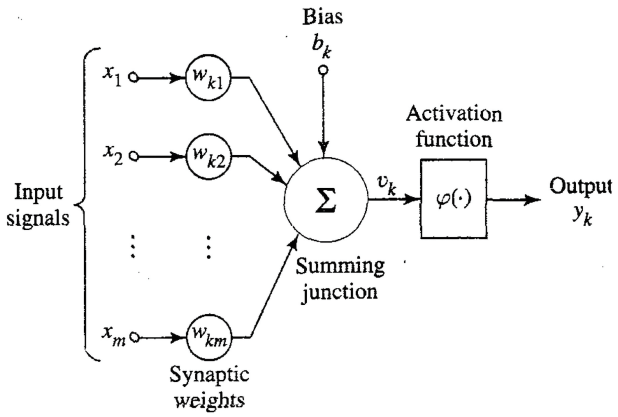


Figura 1.3. Modello di McCulloch e Pitts di un neurone artificiale [Hay98].

riconoscere un modello ricorrente in una serie di dati. Il contributo fondamentale di Rosenblatt fu la stesura dell'algoritmo di apprendimento, basato sulla minimizzazione dell'errore commesso tra l'uscita della rete e quella attesa mediante la variazione del peso delle connessioni tra i nodi.
Negli stessi anni, Bernard Widrow, all'epoca professore di ingegneria elettronica a Stanford, e un suo studente, Ted Hoff, crearono una rete neurale denominata ADALINE, ADAptive LInear NEuron, molto simile al percettrone di Rosenblatt nella struttura ma che differiva per il tipo di funzione di attivazione utilizzata, lineare anziché "a soglia". Ciò che si ricorda però, è l'algoritmo di apprendimento che progettarono, il quale viene utilizzato ancora oggi nell'elaborazione di segnali digitali. Basato sulla minimizzazione dell'errore quadratico medio tramite successive iterazioni, tale algoritmo si rivelò particolarmente efficacie nel filtraggio di segnali rumorosi, motivo per cui Widrow negli anni successivi abbandonò le ricerche sulle reti neurali per concentrarsi sulle telecomunicazioni.
Entrambe le reti create in quegli anni, però, risultarono particolarmente limitate riguardo la varietà di problemi che erano in grado di risolvere. Infatti, a causa della struttura stessa a singolo layer, ovvero con uno o più neuroni in parallelo ma mai in cascata, tali reti potevano trovare soluzione solo di quesiti linearmente classificabili, ovvero in cui i dati risultanti sono idealmente separabili da una retta.
Rosenblatt e Widrow cercarono di correggere i problemi delle loro reti introducendo delle soluzioni più complesse, con un'architettura a più layer. Non riuscirono però ad adattare i loro algoritmi di training alle nuove reti, complice la scarsa potenza di calcolo degli elaboratori dell'epoca. Questo spinse vari ricercatori ad abbandonare gli studi sulle reti neurali artificiali.
A partire dagli anni ottanta, però, grazie agli strumenti di calcolo più potenti disponibili in commercio, vi fu un rinnovato interesse per la materia. Questo portò all'introduzione di un algoritmo di training rivoluzionario ad opera di David Rumelhart e James McClelland, il cosiddetto "algoritmo di retropropagazione dell'errore", che permise di addestrare reti più complesse rispetto a quelle introdotte trent'anni prima, anche con più layer di neuroni in cascata. Il percettrone multistrato, o "multilayer perceptron", addestrato con

tale algoritmo, è ancora oggi la rete neurale più diffusa ed utilizzata, grazie alla grande varietà di funzioni che è in grado di replicare.

1.2.2 Funzioni di attivazione

Il primo dei parametri che caratterizza qualsiasi rete neurale artificiale è il tipo di funzione di attivazione utilizzata. Quest'ultima può presentare vari andamenti in base al comportamento che si vuole ottenere dalla rete. Alcuni dei quali sono riportati in figura 1.4.

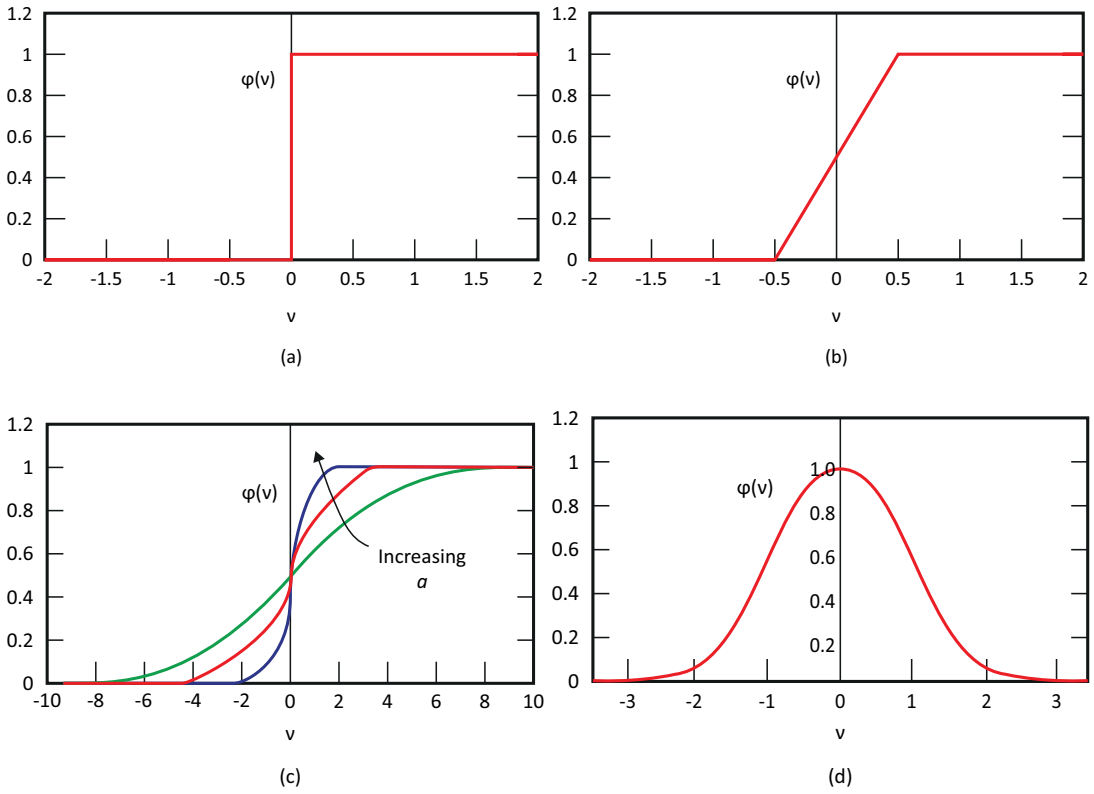


Figura 1.4. Funzioni di attivazione più utilizzate nelle reti neurali: (a) Soglia, (b) Costante a tratti, (c) Sigmoidale, (d) Gaussiana.

La più semplice (1.4.a) è la funzione di soglia, che porta l'uscita al valore unitario una volta che l'ingresso ha superato lo zero:

$$\varphi(v) = \begin{cases} 1 \text{ if } v \geq 0 \\ 0 \text{ if } v < 0 \end{cases} \tag{1.1}$$

Si passa poi ad una funzione costante a tratti, (1.4.b), assimilabile ad un amplificatore lineare saturato, che si può rappresentare mediante il seguente sistema di equazioni:

$$\varphi(v) = \begin{cases} 1 \text{ se } v \geq +\frac{1}{2} \\ v \text{ se } -\frac{1}{2} < v < +\frac{1}{2} \\ 0 \text{ se } v \leq -\frac{1}{2} \end{cases} \tag{1.2}$$

Questa funzione di attivazione consente una transizione più graduale rispetto alla funzione di soglia, risultando però anch'essa non differenziabile, caratteristica problematica in molte applicazioni. Per ovviare a tale inconveniente, una possibile soluzione è rappresentata dalla funzione sigmoidale riportata in figura (1.4.c), strettamente crescente, derivabile, che può essere modulata variandone il parametro *a*. L'equazione che la descrive è la seguente:

$$\varphi(v) = \frac{1}{1 + exp(-av)} \tag{1.3}$$

Infine, in determinate applicazioni può essere utile utilizzare una funzione di attivazione gaussiana, (1.4.d), al fine di dare un carattere locale alla rete. La forma normalizzata è la seguente:

$$\varphi(v) = e^{-\frac{(v-\mu)^2}{2\sigma^2}} \tag{1.4}$$

1.2.3 Architettura della rete

Oltre alla struttura dei singoli nodi, il funzionamento della rete è strettamente correlato all'architettura delle interconnessioni tra i neuroni.
Una prima distinzione può essere fatta tra le reti *autoassociative* e le reti *eteroassociative*: alla prima categoria appartengono le strutture in cui i neuroni di input coincidono con quelli di output, mentre nella seconda vi rientrano tutte le reti composte da più layer, nelle quali i neuroni sono suddivisi in livelli a seconda della loro funzione. Il vantaggio di inserire uno o più layer "nascosti", o *hidden layers*, tra i nodi di ingresso e i nodi di uscita è rappresentato dalla possibilità di approssimare funzioni con un grado di complessità più elevato grazie all'elaborazione operata dai neuroni intermedi. Inoltre questo tipo di strutture offre l'opportunità di operare con un numero di ingressi relativamente elevato rispetto al numero di uscite del sistema. Una delle reti più diffuse è infatti il *percettrone multilayer*, il quale, con un primo stadio costituito da neuroni a funzioni sigmoidali e un secondo strato con funzioni di attivazione lineari, è in grado di approssimare in modo sufficientemente accurato la maggior parte delle funzioni matematiche esistenti.
Un esempio di rete a più strati è riportato in figura (1.5.a). Come si evince dall'immagine, i dieci nodi d'ingesso entrano nei quattro neuroni del layer nascosto, le cui uscite costituiscono l'input dello stadio successivo. Si può inoltre osservare che, questo tipo di rete, si definisce *fully connected*, o interamente connessa, in quanto i neuroni di un layer sono collegati a tutti i neuroni di quello seguente, mentre, se alcune connessioni sinaptiche risultano mancanti, la rete viene detta *partially connected*, o parzialmente connessa.
Un'ulteriore distinzione è quella tra reti con architettura *feedforward* e reti con architettura *a feedback*. Le prime non presentano connessioni di ritorno dagli output agli input del sistema, quindi l'uscita dipende solo dallo stato di attivazione attuale dei neuroni, come avviene nella rete precedentemente analizzata. Le seconde, invece, presentano una linea di ritorno che parte dall'output e si aggiunge agli input, come si evince dalla struttura (1.5.b). In questo caso, lo stato attuale dipende anche dall'uscita al passo precedente e ciò viene schematizzato con la presenza degli operatori di ritardo ad un passo. Si nota inoltre che nella rete riportata non sono presenti anelli di *auto-feedback*, ovvero anelli in cui l'uscita del neurone rientra come ingresso del medesimo neurone, ma solo di *cross-feedback*.

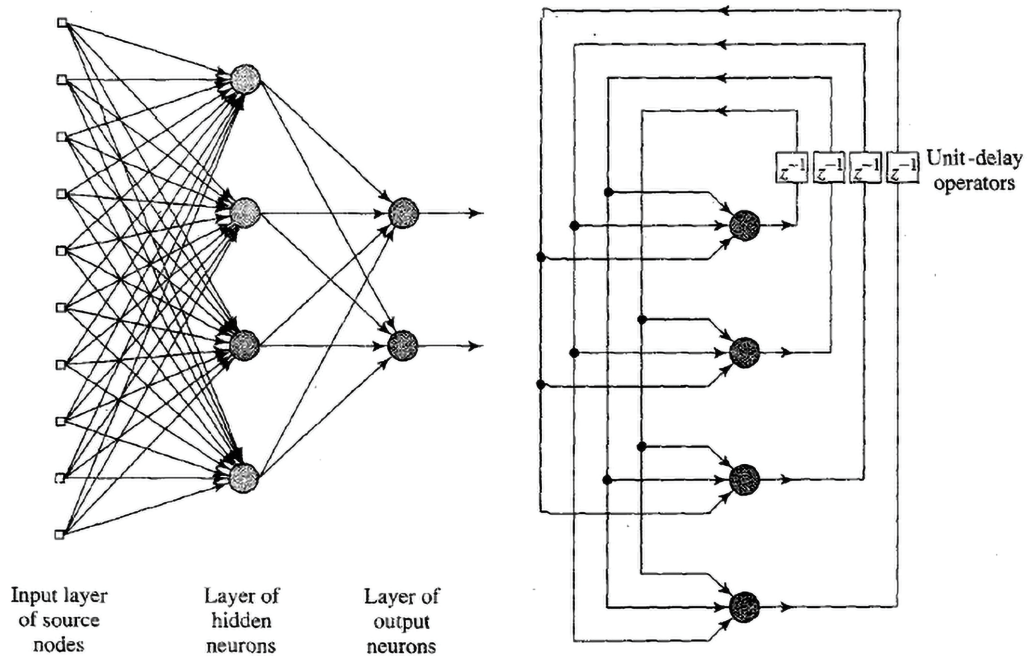


Figura 1.5. Esempi di reti neurali: (a) percettrone multilayer, (b) rete con feedback [Hay98].

1.2.4 Algoritmo di apprendimento

Prima di poter fornire un output coerente con le aspettative, qualsiasi rete neurale deve essere sottoposta ad un preciso processo di apprendimento che ne modifichi i parametri liberi, ovvero i *pesi*, rappresentanti l'intensità del legame tra i neuroni.
Una possibile definizione di *apprendimento* è quella fornita da Mendel e McClaren nel 1970:

> *L'apprendimento è un processo secondo cui i parametri liberi di una rete neurale vengono modificati grazie a una serie di stimoli attuati dall'ambiente in cui la rete è immersa. Il tipo di apprendimento determina il modo in cui la variazione dei parametri prende luogo.*

Questa generica definizione descrive una precisa sequenza di eventi che devono verificarsi prima di considerare la rete "addestrata": innanzitutto è necessaria una stimolazione esterna, con una serie di input coerenti con l'ambiente in cui la rete andrà inserita. Alla stimolazione deve seguire una variazione dei parametri della rete affinché il processo di apprendimento possa considerarsi efficace. Infine si deve osservare un cambiamento nella risposta della rete in seguito al training, con l'obiettivo di avvicinare il più possibile tale risposta all'output desiderato.
In generale, gli algoritmi di training si possono classificare in tre categorie:

- Apprendimento "Supervisionato", o *supervised learning*, in cui durante l'addestramento vengono forniti alla rete sia il set di dati in ingresso che l'output desiderato per ogni input

- Apprendimento "Non Supervisionato", o *unsupervised learning*, in cui alla rete vengono resi disponibili solo gli ingressi, utile nelle applicazioni di pattern recognition
- Apprendimento "Per Rinforzo", o *reinforcement learning*, una combinazione dei due precedenti, in cui vengono inseriti dei dati nella rete e in base alla qualità del risultato fornito vengono rafforzate o affievolite le connessioni tra i neuroni.

Tra le principali regole di apprendimento, invece, si ricordano le cinque fondamentali dalle quali sono derivate tutte le altre: *error-correction learning*, *memory-based learning*, *Hebbian learning*, *competitive learning* e *Boltzmann learning*.

Error-Correction Learning

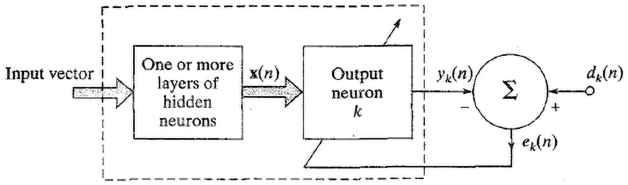


Figura 1.6. Schema a blocchi dell'algoritmo di correzione dell'errore, operante su un singolo neurone [Hay98].

Uno dei metodi di apprendimento più utilizzati è il training basato sulla correzione dell'errore tra l'output desiderato e il risultato fornito dalla rete, il cui principio di funzionamento è riportato in figura (1.6). Indicando con $y_k(n)$ il risultato prodotto dal k-esimo neurone di uscita della rete al passo di apprendimento n-esimo e con $d_k(n)$ l'output desiderato, vale:

$$e_k(n) = d_k(n) - y_k(n) \tag{1.5}$$

con $e_k(n)$ l'errore commesso al passo n dal neurone k. L'obiettivo è quello di ridurre il più possibile tale errore e ciò può essere realizzato minimizzando una funzione di costo, definita, ad esempio, come:

$$F(n) = \frac{1}{2}e_k^2(n) \tag{1.6}$$

L'apprendimento viene ripetuto finché $F(n)$ non raggiunge uno stato di regime. In tale condizione non si trarrebbe vantaggio da un'ulteriore variazione dei pesi.
La minimizzazione di tale funzione segue la regola di Widrow-Hoff, o *regola del delta*, teorizzata da Widrow e Hoff nel 1960. Indicato con w_{kj} il valore del peso del neurone k stimolato con l'ingresso j del vettore di input $\mathbf{x}(n)$, vale:

$$\Delta w_{kj}(n) = \alpha e_k(n) x_j(n) \tag{1.7}$$

con α *grado di apprendimento*, costante positiva, un parametro che determina la rapidità di convergenza verso la soluzione finale. Vale infatti che se α è piccolo (e.g. 0.1), gli step di variazione sono ridotti ed è necessario compiere più iterazioni per arrivare ad un risultato soddisfacente, mentre se α è grande (e.g. 0.9) c'è la possibilità che si verifichino overshoot

e oscillazioni attorno al valore finale. Nel 1992 Eaton e Olivier proposero un'equazione approssimativa per il calcolo di tale coefficiente nel caso di reti progettate per classificare oggetti, la cui formulazione è la seguente:

$$\alpha = \frac{1.5}{\sqrt{\sum p_i^2}} \tag{1.8}$$

con p_i numero di oggetti appartenenti all'i-esima classe di output.
Quanto descritto finora risulta particolarmente adatto nel caso in cui si voglia tarare il peso di un neurone la cui uscita sia accessibile dall'esterno. Nel caso in cui la rete sia dotata di più layer, però, il procedimento da applicare non è altrettanto immediato. Numerosi ricercatori del secolo scorso, infatti, hanno lavorato alla ricerca di una soluzione a tale problema, tra cui si ricordano Rumelhart (1986) e Werbos (1990), arrivando a proporre il *backpropagation algorithm*, o algoritmo di retropropagazione dell'errore. Tale procedura si compone di due passi per ogni ciclo di addestramento, o *epoca*: il primo, detto *forward pass*, ripercorre quanto descritto in precedenza per l'algoritmo "standard", ovvero, dopo aver fornito un ingresso alla rete, si calcola la differenza tra l'output e l'uscita desiderata. La novità è rappresentata dal secondo passo, il *backward pass*, in cui l'errore in out è moltiplicato per i pesi dei nodi connessi al neurone in uscita al fine di trovare l'errore Err_j commesso dal j-esimo nodo dell'hidden layer della rete, errore da minimizzare modificando le interconnessioni.

Memory-Based Learning

Negli algoritmi basati sulla *memorizzazione*, tutte (o quasi) le coppie input-output fornite fino a quel momento alla rete sono salvate in tabelle consultabili dal sistema. Quando arriva un nuovo ingresso, il sistema risponde analizzando i dati memorizzati che più gli si avvicinano, definendo un output coerente con quanto proposto in precedenza per i dati individuati.
In particolare, in questo tipo di algoritmi è necessario scegliere due parametri fondamentali: il primo è il criterio usato per definire i dati che più assomigliano al nuovo input, il secondo è il modo in cui viene calcolato l'output una volta definiti i campioni attinenti. Tra le tecniche più semplici si ricordano la *nearest neightbor rule*, o regola del vicino più prossimo, secondo la quale il dato da testare $\mathbf{x}_{test}$ è assimilato a quello salvato in memoria che con la minima distanza euclidea.
In figura (1.7) è riportata una tecnica di classificazione simile, chiamata "k-nearest neighbor". Tale algoritmo si basa sull'identificare un numero intero di pattern, tra quelli memorizzati, che si avvicinano all'ingresso $\mathbf{x}_{test}$ fornito, al fine di assegnarlo alla classe che si presenta più frequentemente nell'intorno considerato. In questo caso l'ingresso *d* viene assegnato alla classe 1, nonostante sia più vicino ad un dato classificato come 0. Quest'ultimo viene invece definito "valore anomalo", o *outlier*, in quanto si presenta meno frequentemente nell'intorno. Questo valore, grazie alla tecnica descritta, non va ad influenzare il risultato, come invece avverrebbe con la nearest neighbor rule.
Una volta associata la nuova coppia ingresso-uscita, anch'essa verrà aggiunta alla memoria. Il processo, che teoricamente potrebbe continuare all'infinito, è ovviamente limitato dalla quantità di spazio di archiviazione disponibile.

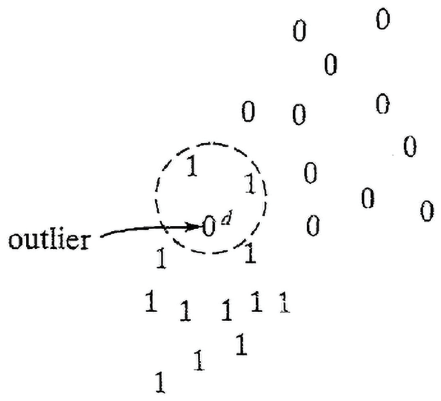


Figura 1.7. Esempio di classificazione di un dato d basata sulla tecnica "k-nearest neighbor", con $k = 3$ [Hay98].

Hebbian Learning

Il postulato di Hebb sull'apprendimento è il primo e forse il più famoso, nominato in onore del neuropsicologo Donal Hebb, sul cui lavoro è ispirata la tecnica che si andrà a descrivere in questo paragrafo. Come scriveva nella sua opera "The Organization of Behavior" del 1949, "quando l'assione della cellula A è abbastanza vicino da stimolare ripetutamente la cellula B, l'interconnessione tra le due viene rafforzata in modo che l'attivazione reciproca avvenga in modo più efficiente". Questo postulato pose le basi per una serie di ricerche in ambito neurobiologico, ma rapidamente venne applicato anche alle reti neurali. Prima Stent nel 1973 e, successivamente, Changeux e Danchin nel 1976 formularono il seguente assioma in due parti, di cui solo la prima riferita al lavoro di Hebb:

1. Se due neuroni agli estremi di una sinapsi si attivano simultaneamente, l'intensità dell'interconnessione viene rafforzata
2. Se due neuroni agli estremi di una sinapsi si attivano in modo asincrono, tale collegamento si indebolisce o viene eliminato

Questo tipo di sinapsi viene detta "Sinapsi Hebbiana", e le sue caratteristiche sono: una dipendenza diretta dal tempo in cui avviene l'attivazione, un forte carattere locale e un meccanismo di affinamento strettamente correlato alla presenza o meno di interazioni tra i neuroni ai suoi capi.
La forma più semplice dell'apprendimento hebbiano è descritta dall'equazione:

$$\Delta w_{kj}(n) = \alpha y_k(n) x_j(n) \tag{1.9}$$

con α grado di apprendimento, $y_k(n)$ l'output del neurone k in riposta al j-esimo ingresso $x_j(n)$ alla ripetizione n dell'algoritmo.
La retta superiore nel grafico di figura (1.8) mostra però il limite di questa tecnica: l'applicazione ripetuta di un ingresso x_j, alla quale consegue una crescita di $y_k(n)$, porta ad un aumento incontrollato di w_{kj} fino ad un livello in cui la sinapsi risulta priva di

informazione.
Una possibile soluzione a tale problematica consiste nell'utilizzare la covarianza:

$$\Delta w_{kj}(n) = \alpha(x_j(n) - \overline{x})(y_k(n) - \overline{y}) \tag{1.10}$$

con $\overline{x}$ e $\overline{y}$ valori medi di input e output. In questo modo i pesi smettono di variare quando, in seguito ad un ingresso $x_j(n) = \overline{x}$, la risposta si assesta al suo valore medio $\overline{y}$.

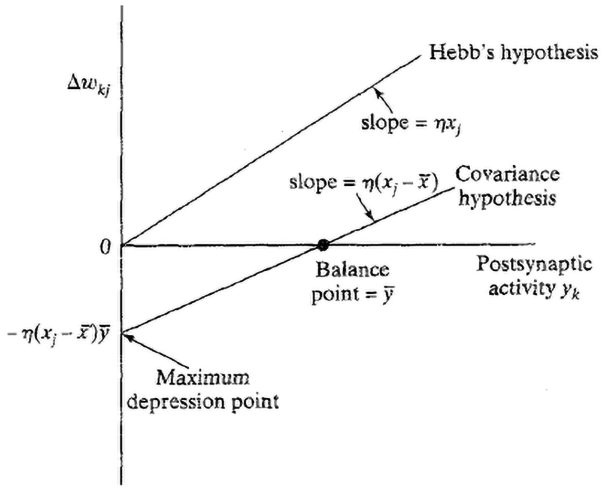


Figura 1.8. Addestramento basato sull'ipotesi di Hebb confrontato con l'addestramento basato sulla covarianza [Hay98].

Competitive Learning

Come indica il nome, nel *competitive learning* i neuroni devono competere tra loro per attivarsi. Mentre in altri tipi di algoritmi più neuroni possono attivarsi contemporaneamente, nell'apprendimento competitivo un solo neurone di uscita per volta può attivarsi. Questo lo rende particolarmente adatto per il riconoscimento di caratteristiche ricorrenti e per la classificazione di set di dati.
In figura (1.9) è riportata una semplice rete per l'apprendimento competitivo. Come si nota, è presente un unico layer di neuroni, la cui struttura delle interconnessioni è realizzata in modo da attuare il meccanismo di inibizione che porterà un solo nodo per volta ad attivarsi. Il segnale in uscita da ogni neurone è quindi riassumibile nel seguente sistema:

$$y_k = \begin{cases} 1 \text{ se } v_k > v_j \quad \forall j, j \neq k \\ 0 \quad \text{altrimenti} \end{cases} \tag{1.11}$$

in cui v_k rappresenta la sommatoria degli ingressi nel neurone k.
Supponendo ora di assegnare un valore fisso alla somma dei pesi di tutte le connessioni sinaptiche di un neurone, ad esempio

$$\sum_j w_{kj} = 1 \tag{1.12}$$

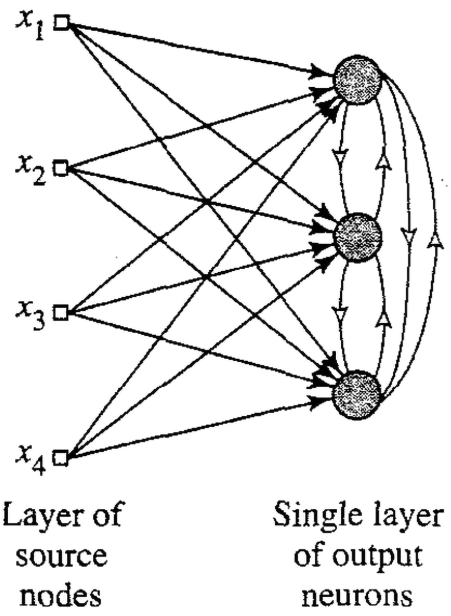


Figura 1.9. Esempio di rete neurale a singolo layer costruita per un addestramento di tipo competitivo [Hay98].

il processo di apprendimento del neurone prende luogo con lo spostamento del peso dalle linee di ingresso inattive a quelle attive. Se, ad esempio, le caratteristiche dell'ingresso sono tali da non richiedere l'attivazione del nodo, esso non parteciperà all'apprendimento, mentre quello "vincente" sposterà i pesi verso le linee d'ingresso attive, in modo da ottenere, una volta completato l'addestramento, un insieme di ingressi simili che porteranno alla sua attivazione.
La regola di apprendimento risulta quindi:

$$\Delta w_{kj} = \begin{cases} \alpha(x_j - w_{kj}) & \text{se il neurone } k \text{ vince la competizione} \\ 0 & \text{se il neurone } k \text{ perde la competizione} \end{cases} \tag{1.13}$$

Boltzmann Learning

L'ultima tipologia di algoritmi di training prende spunto dalla termodinamica per aggiungere una componente stocastica all'addestramento. In particolare, in una cosiddetta *Macchina di Boltzmann*, una rete neurale addestrata con l'omonimo algoritmo, i neuroni operano in modo binario: se attivi assumono il valore +1, altrimenti −1.
La macchina è caratterizzata da una funzione di energia, E, definita come:

$$E = -\frac{1}{2} \sum_j \sum k w_{kj} x_k x_j, j \neq k \tag{1.14}$$

con x_j e x_k stati dei due neuroni collegati dalla sinapsi di peso w_{kj}. La condizione $j \neq k$ si traduce semplicemente nella mancanza di anelli di auto-feedback nella rete.
Durante l'addestramento, lo stato di un neurone k, scelto in modo casuale, viene invertito

da x_k a $-x_k$ con probabilità:

$$P(x_k \rightarrow -x_k) = \frac{1}{1 + exp(-\Delta E_k / T)} \tag{1.15}$$

con ΔE_k variazione di energia derivante da tale operazione e T *pseudotemperatura* del sistema, ovvero un parametro che permette di controllare il "rumore" nelle sinapsi, riconducibile all'incertezza con cui si attiveranno i neuroni. Riducendo T ad un valore prossimo allo zero, si degenera nel caso di un sistema deterministico, privo di rumore, detto *modello di McCulloch-Pitts*.
Iterando la (1.15), la macchina si porterà verso uno stato di "equilibrio termico", condizione in cui la rete potrà considerarsi addestrata.

1.2.5 Algoritmo di richiamo

Una volta che la rete è stata addestrata, è necessario implementare un algoritmo in grado di sfruttare i "pesi" che caratterizzano le interconnessioni tra i neuroni per proporre, in corrispondenza di un determinato input, un'uscita coerente con il risultato desiderato. Nella maggior parte dei casi, quest'ultimo si discosterà dal set utilizzato per il training, rendendo necessaria una fase di *generalizzazione*, punto di forza delle reti neurali. Per fare ciò, si sfrutta il principio *similar stimuli cause similar reactions*, ovvero di fronte ad un input x' che assomigli ad x_{test} utilizzato nell'apprendimento, verrà proposta in uscita una grandezza y' che si avvicina a y_{test}, mediante l'attivazione dello stesso pattern di neuroni. Come verrà discusso meglio in seguito, questo meccanismo di funzionamento permette di ottenere una relazione ingresso-uscita continua, non discreta, particolarmente vantaggiosa nel caso si voglia effettuare una post-elaborazione sui dati, ad esempio una derivata del primo ordine.

2

ERROR-CORRECTION LEARNING

Come accennato nel capitolo precedente, uno dei metodi più utilizzati per addestrare i pesi dei vari layer delle reti neurali è l'algoritmo di training basato sulla correzione dell'errore.
Per comprendere meglio il suo funzionamento, verrà fatta una trattazione matematica sulla minimizzazione di funzioni quadratiche, le più utilizzate come funzioni di costo, concentrandosi principalmente sul metodo di Newton, dal quale è stato derivato uno dei algoritmi di training più efficienti e rapidi nella convergenza, il *Levenberg-Marquardt*.
Prima di partire con la dimostrazione, si ricorda la forma generica di una funzione quadratica:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c \tag{2.1}$$

il cui gradiente vale:

$$\Delta F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d} \tag{2.2}$$

mentre la matrice Hessiana:

$$\Delta F^2(\mathbf{x}) = \mathbf{A} \tag{2.3}$$

con **A** matrice simmetrica.

2.1 *Minimizzazione di una funzione quadratica*

Il primo passo è quello di trovare un meccanismo per minimizzare una generica funzione $F(\mathbf{x})$ agendo sull'argomento $\mathbf{x}$. Si inizia sempre da un valore di partenza $\mathbf{x}_0$ scelto casualmente, per poi procedere in modo iterativo come segue:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k = \mathbf{x}_k + \alpha_k\mathbf{p}_k \tag{2.4}$$

dove il vettore $\mathbf{p}_k$ rappresenta la *direzione* in cui si va a ricercare il minimo della funzione, che, nel caso scalare, si riconduce alla scelta "variazione positiva" o "variazione negativa", mentre α_k è il *learning rate*, o grado di apprendimento, scalare positivo che determina l'ampiezza dello step di apprendimento.
Quando si ricerca il minimo di una funzione nel minor numero di passi possibile, si desidera che venga rispettata la condizione:

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k) \tag{2.5}$$

Lo scopo è ora definire una direzione $\mathbf{p}_k$ che, per un learning rate α_k sufficientemente ridotto, permetta di rispettare sempre la (2.5).
Consideriamo innanzitutto l'espansione di Taylor al primo ordine della F(**x**):

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T\Delta\mathbf{x}_k \tag{2.6}$$

con $\mathbf{g}_k$ gradiente di F valutato al passo k:

$$\mathbf{g}_k = \nabla F(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_k} \tag{2.7}$$

Per la (2.5) deve valere:

$$\mathbf{g}_k^T \Delta\mathbf{x}_k = \alpha_k \mathbf{g}_k^T \mathbf{p}_k < 0 \tag{2.8}$$

che, con α_k costante positiva (learning rate), si riconduce a:

$$\mathbf{g}_k^T \mathbf{p}_k < 0 \tag{2.9}$$

Ogni vettore $\mathbf{p}_k$ che soddisfa tale equazione è denominato "direzione discendente".
L'obiettivo diviene ora trovare la direzione più rapida per la convergenza, ovvero quella in cui $\mathbf{g}_k^T \mathbf{p}_k$ è "il più negativo possibile". Per trovare il minimo della funzione lungo l'asse identificato da un generico vettore $\mathbf{p}$, è necessario ricorrere alla definizione di *derivata direzionale* di $F(\mathbf{x})$:

$$\frac{\mathbf{p}^T \nabla F(\mathbf{x})}{||\mathbf{p}||^2} \tag{2.10}$$

Si nota che il valore massimo si ha con i due vettori $\mathbf{p}$ e $\nabla F(\mathbf{x})$ che puntano nella stessa direzione, quindi:

$$\mathbf{p}_k^T = -\mathbf{g}_k \tag{2.11}$$

Si ricava quindi l'equazione del metodo definito di "discesa più ripida", o *steepest descend*:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k \tag{2.12}$$

In questo caso rimane da definire il valore di α_k, ricordando che se ridotto saranno richieste più iterazioni per la convergenza, se elevato potrebbero formarsi delle oscillazioni attorno al valore ottimo.
Una possibile soluzione consiste nel fissare α pari ad una costante positiva ed effettuare delle prove sperimentali. Un'alternativa più raffinata è andare a ricercare il minimo di F lungo delle traiettorie rettilinee (al variare di α), ovvero:

$$min[F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)] \tag{2.13}$$

In figura (2.1) è riportato un esempio di minimizzazione di funzione quadratica seguendo traiettorie rettilinee a partire da un punto casuale $\mathbf{x}_0 = [0.8; -0.25]$. Si nota immediatamente come venga rispettata la condizione (2.5), oltre al fatto che, dato che la direzione di ricerca è il negato del gradiente, la ricerca successiva sarà ortogonale alla precedente, per l'equazione (2.10).

2.2 *Metodo di Newton*

L'algoritmo della discesa più ripida si basava sul primo ordine della serie di Taylor della funzione $F(\mathbf{x})$. Il metodo di Newton, invece, si basa su un'espansione al secondo ordine:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k^T \mathbf{A}_k \Delta\mathbf{x}_k \tag{2.14}$$

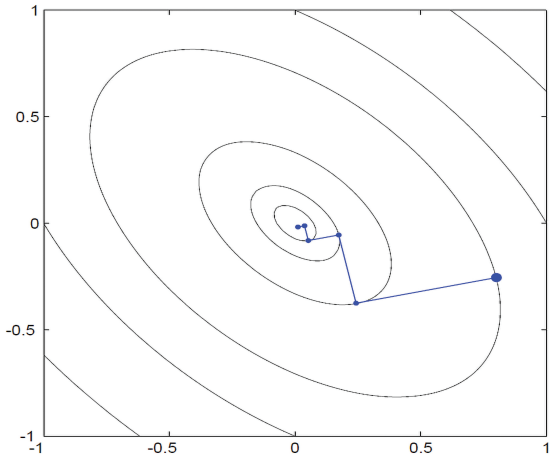


Figura 2.1. Metodo della discesa più rapida, minimizzando in linea retta [Hag+14].

con $\mathbf{A}_k$ *matrice Hessiana* valutata al passo k:

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k} \tag{2.15}$$

L'obiettivo del metodo di Newton è quello di trovare il punto stazionario (minimo) di tale approssimazione quadratica. Usando l'espressione del gradiente (2.2) ed eguagliandola a zero per cercare il minimo della funzione, si trova:

$$\mathbf{g}_k + \mathbf{A}_k \Delta\mathbf{x}_k = 0 \tag{2.16}$$

che risolvendo per $\Delta\mathbf{x}_k$ diventa:

$$\Delta\mathbf{x}_k = -\mathbf{A}_k^{-1}\mathbf{g}_k \tag{2.17}$$

ottenendo perciò il metodo di Newton:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k \tag{2.18}$$

É possibile dimostrare che, con questo metodo, si trova sempre il minimo di una funzione quadratica in un solo passo. Ciò avviene in quanto è stato ideato per approssimare qualunque funzione con una quadratica e poi trovarne il punto stazionario. Se la funzione è già quadratica, la minimizzazione avverrà in un solo step.
Il problema di questo metodo è che richiede il calcolo della matrice Hessiana, e quindi delle derivate seconde della funzione. La sua invertibilità, inoltre, non è sempre garantita.

2.3 *Levenberg-Marquardt Algorithm*

L'algoritmo di *Levenberg-Marquardt* è una variante del metodo di Newton per funzioni quadratiche, quindi è particolarmente adatto a il training di reti neurali basato sulla minimizzazione dell'errore quadratico.
Si parte quindi dall'assunzione che F sia una somma di N funzioni quadratiche:

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i^2(\mathbf{x}) = \mathbf{v}^T(\mathbf{x})\mathbf{v}(\mathbf{x}) \tag{2.19}$$

Il j-esimo elemento del gradiente si definisce come:

$$[\nabla F(\mathbf{x})]_j = \frac{\partial F(\mathbf{x})}{\partial x_j} = 2\sum_{i=1}^{N} v_i(\mathbf{x})\frac{\partial v_i(\mathbf{x})}{\partial x_j} \tag{2.20}$$

che si può abbreviare nella forma matriciale:

$$\nabla F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{v}(\mathbf{x}) \tag{2.21}$$

con **J(x)** detta "matrice Jacobiana" e definita come segue:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial v_1(\mathbf{x})}{\partial x_1} & \frac{\partial v_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial v_2(\mathbf{x})}{\partial x_1} & \frac{\partial v_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial v_N(\mathbf{x})}{\partial x_1} & \frac{\partial v_N(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_N(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{2.22}$$

Il prossimo passo è il calcolo della matrice Hessiana $\mathbf{A} = \nabla^2 F(\mathbf{x})$, il cui generico elemento k,j-esimo si calcola come:

$$[\nabla^2 F(\mathbf{x})]_{k,j} = \frac{\partial^2 F(\mathbf{x})}{\partial x_k \partial x_j} = 2\sum_{i=1}^{N} \frac{\partial v_i(\mathbf{x})}{\partial x_k}\frac{\partial v_i(\mathbf{x})}{\partial x_j} + v_i(\mathbf{x})\frac{\partial^2 v_i(\mathbf{x})}{\partial x_k \partial x_j} \tag{2.23}$$

che, in forma matriciale, diventa:

$$\nabla^2 F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + 2\mathbf{S}(\mathbf{x}) \tag{2.24}$$

con $\mathbf{S}(\mathbf{x})$ definita come:

$$\mathbf{S}(\mathbf{x}) = \sum_{i=1}^{N} v_i(\mathbf{x})\nabla^2 v_i(\mathbf{x}) \tag{2.25}$$

Assumendo che $\mathbf{S}(\mathbf{x})$ sia trascurabile all'interno del calcolo dell'Hessiana in quanto spesso i contributi "incrociati" di due differenti elementi x_j e x_k sono nulli, o molto ridotti, si ottiene:

$$\nabla^2 F(\mathbf{x}) \approx 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) \tag{2.26}$$

Sostituendo quindi le equazioni (2.21) e (2.26) dentro la (2.18), si ottiene il metodo di *Gauss-Newton*:

$$\begin{aligned} \mathbf{x}_{k+1} &= x_k - \left[2\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)\right]^{-1} 2\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) \\ &= x_k - \left[\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)\right]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) \end{aligned} \tag{2.27}$$

Si può notare immediatamente uno dei vantaggi di questo metodo, ovvero il fatto che non sia necessario il calcolo della derivata seconda grazie alle approssimazioni, comunque marginali, sul gradiente.
Uno dei problemi di questa tecnica, però, si nota nella prima parte dell'equazione (2.27): affinché il problema sia risolvibile, è necessario che la matrice $\mathbf{H} = \mathbf{J}^T\mathbf{J}$ sia invertibile, e

tale ipotesi non è sempre verificata.
Una possibile soluzione consiste nel sostituire l'Hessiana con una sua versione modificata:

$$\mathbf{G} = \mathbf{H} + \mu\mathbf{I} \tag{2.28}$$

in cui il termine $\mu\mathbf{I}$ rappresenta una matrice diagonale che fornisce un contributo costante all'Hessiana in modo da renderla invertibile.
Questo porta all'algoritmo di *Levenberg-Marquardt*:

$$\mathbf{x}_{k+1} = x_k - \left[\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}\right]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) \tag{2.29}$$

Analizzando quanto ottenuto, si possono distinguere le due situazioni opposte che si vengono a creare in base alla scelta di μ_k: in caso di un valore molto ridotto l'algoritmo si riconduce al Gauss-Newton descritto in precedenza, con la problematica dell'invertibilità, mentre con un μ_k elevato si converge verso l'approccio steepest descend, descritto dalla (2.12):

$$\mathbf{x}_{k+1} \simeq \mathbf{x}_k - \frac{1}{\mu_k}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) = \mathbf{x}_k - \frac{1}{2\mu_k}\nabla F(\mathbf{x}) \tag{2.30}$$

3

RADIAL BASIS FUNCTION NEURAL NETWORK

Come accennato nel capitolo precedente, una delle reti neurali più diffuse è il *multilayer perceptron*, o percettrone multistrato, solitamente costituito da uno o più livelli di nodi con funzioni di attivazione sigmoidali, e da un layer di uscita lineare. La sua diffusione è dovuta alla possibilità di approssimare qualsiasi funzione matematica con elevata precisione grazie, appunto, alla combinazione di diversi livelli, ognuno con una propria funzione caratteristica. Una rete a singolo strato, invece, può risolvere solamente problemi classificabili linearmente.
Il prezzo da pagare è ovviamente una maggior complessità strutturale, che si traduce in un training più oneroso. Tra gli algoritmi che adempiono a tale scopo si ricorda il *back-propagation*, il quale però, per un addestramento efficacie, richiede numerose iterazioni, ognuna delle quali si compone di più passi, come descritto al paragrafo (1.2.4).
Un'alternativa alla rete a percettroni è la *Radial Basis Function Network*, spesso abbreviata in *RBF*.
Le *funzioni radiali di base* sono delle mappe a variabili reali il cui valore dipende esclusivamente dalla distanza tra l'ingresso e un punto (fissato) del dominio. Il primo a servirsi di tali funzioni fu il matematico inglese Michael Powell, il quale, verso la metà degli anni ottanta, le utilizzò per interpolare dati in uno spazio multidimensionale. Pochi anni più tardi, Broomhead e Lowe proposero il primo modello di rete neurale con funzioni radiali, il cui punto di forza era la capacità di generalizzare in modo accurato in presenza di nuovi dati in ingresso. Per tale motivo, queste reti risultano particolarmente adatte ad essere impiegate per attuare processi di curve fitting, anche in corrispondenza di set di dati in ingresso limitati o affetti da rumore.

3.1 *Struttura*

La struttura di una rete neurale basata su funzioni radiali di base è riportata in figura (3.1), in cui si possono distinguere R ingressi, S^1 neuroni nel primo layer e S^2 nel secondo.
Come si evince dall'immagine, tale rete si compone di due strati: nel primo viene calcolata la distanza, solitamente euclidea, tra l'ingresso e il vettore $\mathbf{W}^1$, contenente le coordinate dei nodi. Queste ultime si traducono nei "centri" delle funzioni radiali, in questo caso gaussiane, in quanto, sebbene siano centrate sull'asse delle ascisse, il calcolo della distanza funge da traslazione per l'ingresso.
Si nota poi la prima differenza rispetto ad una rete a percettroni: al contrario di quanto avviene in quest'ultima, in una radial basis l'ingresso di *bias* viene moltiplicato, non sommato. Ciò avviene in quanto tale grandezza viene utilizzata per ampliare o restringere il campo d'azione delle singole gaussiane, in modo da aumentare o diminuire il grado di sovrapposizione in base alla risposta desiderata.

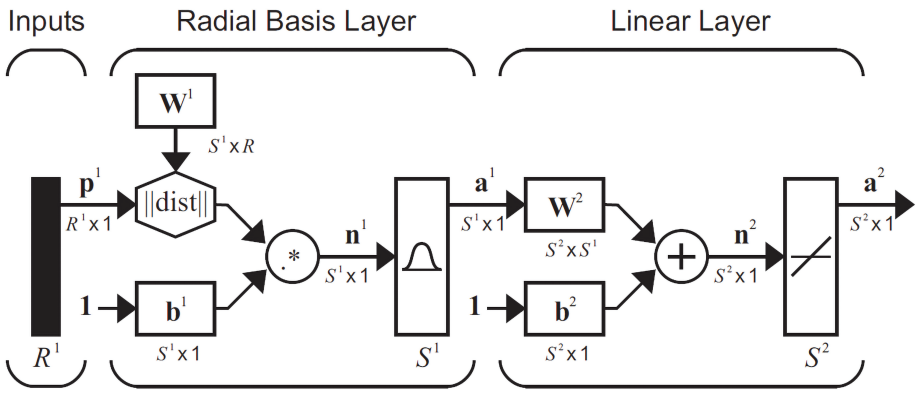


Figura 3.1. Struttura di una Radial Basis Function Network [Hag+14].

L'input per l'i-esimo neurone avrà quindi la seguente espressione:

$$n_i^1 = ||\mathbf{x} - \mathbf{w}_i^1||b_i^1 \tag{3.1}$$

con $\mathbf{x}$ vettore degli ingressi, $\mathbf{w}_i$ coordinata dell'i-esimo nodo della rete e b_i ingresso di bias.

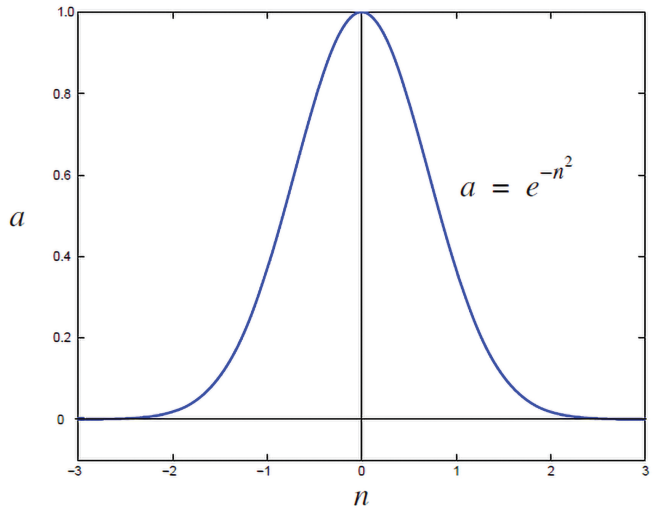


Figura 3.2. Funzione radiale di base gaussiana [Hag+14].

Il valore che ne risulta viene poi passato in ingresso alla funzione di attivazione del neurone. Al contrario di quanto avviene nella multilayer perceptron, la quale utilizza funzioni sigmoidali, nella radial basis si prediligono geometrie che conferiscano un carattere locale, come ad esempio gaussiane, il cui andamento è riportato in figura (3.2). Le prime, infatti, tendono ad un valore prossimo all'unità anche per ingressi lontani dall'origine, fornendo ai neuroni un carattere globale, mentre le seconde degradano esponenzialmente all'aumentare della distanza, conferendo ai neuroni un raggio di intervento limitato, secondo

l'equazione:

$$a_i^1 = e^{-n_i^2} \tag{3.2}$$

É immediato osservare come un argomento n_i quasi nullo, ovvero un ingresso centrato sul nodo i-esimo, porti ad un'uscita prossima all'unità nella funzione di attivazione e, di conseguenza, ad un neurone che parteciperà in modo marcato alla definizione dell'uscita. Si osserva inoltre l'effetto del bias b_i, il quale va ad aumentare o a diminuire la deviazione standard della gaussiana, ovvero la sua estensione.
Il carattere locale della rete ha un impatto diretto sulla sua struttura: mentre in una multilayer perceptron bastano pochi neuroni in quanto tutti partecipano all'output della rete, nella RBF i centri devono essere distribuiti in tutto il range di ingressi in cui si prevede opererà la rete, adeguatamente scalati mediante il bias in modo che vi sia una parziale sovrapposizione tra i nodi. Questo porta ad avere un maggior numero di neuroni da addestrare, i quali, inoltre, necessitano di un input in grado di attivarli per tararne correttamente i pesi. Da qui sorge la necessità di trovare un set di training che copra l'intero working range.
Bisogna però osservare che, con pochi neuroni attivi per volta, si ottiene un addestramento più rapido, che converge ad un valore ottimale dopo poche iterazioni. Inoltre, per la struttura stessa della rete, i pesi di ciascuno dei due layer sono indipendenti, così come i relativi training, come verrà discusso meglio in seguito.
Per quanto riguarda il secondo strato della rete, è costituito semplicemente da una sommatoria lineare, il cui argomento è l'output di ciascuna gaussiana, opportunamente pesato per ottenere la mappa desiderata. In uscita dalla rete si avrà quindi l'espressione:

$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2 \tag{3.3}$$

in cui gli apici fanno riferimento al layer di appartenenza.

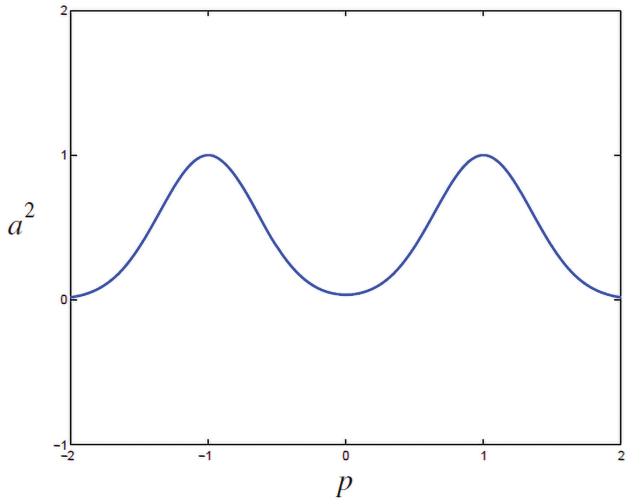


Figura 3.3. Risposta della rete RBF con i parametri di base [Hag+14].

In figura (3.3) è riportata la risposta di una rete RBF con due neuroni nell'hidden layer e uno nello strato di uscita, con la seguente configurazione parametrica:

$$w_1^1 = -1,\ w_2^1 = 1,\ b_1^1 = 2,\ b_2^1 = 2,\ w_1^2 = 1,\ w_2^2 = 1,\ b^2 = 0$$

Come si nota, i pesi del primo layer identificano i centri delle gaussiane: quando l'ingresso si avvicina a tali coordinate, l'uscita assume un valore crescente, fino a raggiungere l'unità. La loro modifica comporta la variazione che si evince in figura (3.4.b). I bias del primo layer, inoltre, servono a definire l'ampiezza della gaussiana, anche identificata come *deviazione standard*, e influenzano il grado di sovrapposizione delle funzioni (figura (3.4.a)). Dato che, al contrario della vera deviazione standard che caratterizza l'espressione normalizzata delle gaussiane, i bias entrano nel numeratore dell'esponente, all'aumentare del loro valore tali funzioni si restringono. I pesi del secondo strato, invece, servono a scalare in ampiezza la risposta dei singoli nodi, come si comprende in figura (3.4.c), mentre i bias traslano l'intero andamento lungo l'asse delle ordinate (3.4.d).

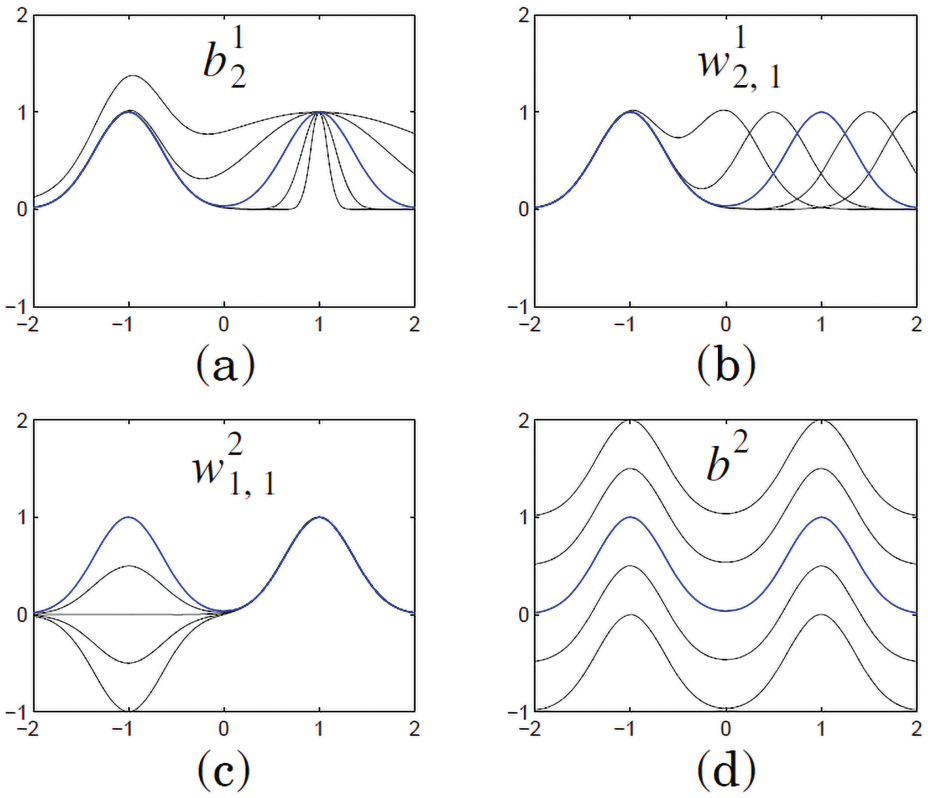


Figura 3.4. Risposta della rete RBF al variare dei parametri [Hag+14].

3.2 *Addestramento della rete*

Per quanto riguarda il training della rete, è importante osservare come i due layer possano essere addestrati in modo indipendente: in un primo momento si posizionano i nodi del primo strato, successivamente si procede con la taratura dei pesi del secondo.

3.2.1 Primo Layer

Per posizionare i centri dei nodi, si possono seguire diverse metodologie. L'opzione più semplice, ma spesso più efficacie, è equispaziare le coordinate in tutto il range in cui la rete andrà ad operare, scegliendo un bias che consenta un certo grado di sovrapposizione tra le gaussiane. Il problema in questo caso è che, all'aumentare delle dimensioni dell'area di lavoro, il numero di nodi cresce quadraticamente, a causa della dipendenza geometrica del problema.
In caso di funzioni fortemente non lineari da approssimare, una possibile strategia per ottenere un risultato più accurato sarebbe concentrare un maggior numero di gaussiane nell'area dove la mappa desiderata è più complessa. Ciò è giustificato anche dal fatto che, in molti casi, durante il normale funzionamento del sistema non si utilizza quasi mai l'intera area su cui viene fatto il training, quindi molte gaussiane non parteciperanno mai all'output.
Una soluzione più raffinata, ma anche più complessa da attuare, è quella di creare una griglia con un numero possibile di nodi sufficientemente elevato, teoricamente uno per ogni input del vettore di ingresso scelto per il training. Ad ogni passo della procedura, l'algoritmo sceglie un centro e lo aggiunge al layer, valutandone l'influenza sull'output della rete in base a un criterio a scelta, come ad esempio la riduzione dell'errore quadratico. La procedure si ripete finché non si raggiunge una risoluzione soddisfacente nella mappa d'uscita.
Una volta fissata la posizione del centri, è possibile calcolare un valore per il bias che permetta una parziale sovrapposizione tra le gaussiane, onde evitare di creare delle zone in cui nessun neurone viene attivato dall'input. La formula proposta da David Lowe nella sua pubblicazione del 1989 [Low89], è la seguente:

$$b_i^1 = \frac{\sqrt{N}}{d_{max}} \tag{3.4}$$

in cui N è il numero totale di nodi nel primo layer e d_{max} è la distanza massima coperta dai nodi della rete. Con questa soluzione tutti i bias hanno un valore costante, risulta perciò particolarmente adatta ad essere applicata ad una rete con i nodi a distanza fissa.

3.2.2 Secondo Layer

Una volta determinati i parametri del primo layer, si passa all'addestramento del secondo. Questo può essere fatto con qualunque algoritmo basato sulla correzione dell'errore in uscita, senza necessità di ricorrere alla retropropagazione, in quanto si applica ad un unico strato della rete, al contrario di quanto avviene nelle multilayer perceptrons. Un algoritmo adatto a svolgere tale funzione è il Levemberg-Marquardt che, come descritto nel paragrafo (2.3), si presta particolarmente ad essere utilizzato nel caso si voglia minimizzare una funzione di costo quadratica, in quanto permette di ottenere un risultato attendibile in un solo passo di training.
Le reti RBF presentano un ulteriore vantaggio sotto questo punto di vista: il loro carattere locale consente, in determinate condizioni, di attuare un training "incrementale", ovvero di addestrarne un sottoinsieme di neuroni alla volta, in base a dove sono localizzati i dati

in input. Ciò permette di effettuare un affinamento della rete mentre viene utilizzata per il normale funzionamento del sistema.

4

RETE NEURALE PER LA STIMA DEL MODELLO MAGNETICO DI UN MOTORE SINCRONO

In questo capitolo si andrà ad applicare quanto descritto finora sulle reti RBF e sull'algoritmo di Levenberg-Marquardt al caso in esame di stima del modello magnetico di un motore sincrono. Quanto riportato vale sia per i motori isotropi a magneti permanenti superficiali che per gli IPM, oltre che per i motori sincroni a riluttanza.

4.1 *Modello del motore*

Innanzitutto è necessario definire un modello per il motore elettrico, il quale verrà utilizzato come riferimento per il training della rete. A tale scopo, è possibile considerare il *bilancio di tensione* nel sistema rotante *dq*, sincronizzato con l'angolo elettrico del rotore:

$$\begin{aligned} u_d &= R_s i_d + \frac{\mathrm{d}\lambda_d(i_d, i_q)}{\mathrm{d}t} - \omega_{me}\lambda_q(i_d, i_q) \\ u_q &= R_s i_q + \frac{\mathrm{d}\lambda_q(i_d, i_q)}{\mathrm{d}t} + \omega_{me}\lambda_d(i_d, i_q) \end{aligned} \tag{4.1}$$

dove u_d, u_q sono le tensioni di statore, i_d, i_q le correnti di statore e λ_d, λ_q i flussi magnetici concatenati. $\omega_{me} = p\omega_m$ è la velocità angolare meccanico-elettrica del rotore, calcolata conoscendo il numero di coppie polari del motore p e la velocità meccanica ω_m.
É importante evidenziare la dipendenza dei flussi magnetici dalle correnti i_d, i_q: tale relazione non è lineare in quanto, all'aumentare della corrente, si incorre nel fenomeno della saturazione magnetica, che provoca una diminuzione del gradiente del flusso per correnti elevate.
Conoscendo il legame tra flusso e corrente, è possibile definire l'*induttanza differenziale*, la cui espressione è la seguente:

$$L_d^{diff}(i_d, i_q) = \frac{\partial\lambda_d(i_d, i_q)}{\partial i_d}\ ,\ L_q^{diff}(i_d, i_q) = \frac{\partial\lambda_q(i_d, i_q)}{\partial i_q} \tag{4.2}$$

Inoltre, se i percorsi magnetici sono condivisi, la saturazione di una porzione di materiale ferromagnetico dovuta ad una delle due correnti, provoca variazioni di flusso anche sull'altro asse, indipendentemente dall'altra corrente. Questo fenomeno prende il nome di *cross-saturation*, o saturazione incrociata, e provoca la dipendenza del flusso da entrambe le correnti, esprimibile tramite l'induttanza differenziale mutua:

$$L_{dq}^{diff}(i_d, i_q) = L_{qd}^{diff} = \frac{\partial\lambda_d(i_d, i_q)}{\partial i_q} = \frac{\partial\lambda_d}{\partial i_q} \tag{4.3}$$

Infine si ricorda che, l'eventuale contributo dato da un magnete permanente Λ_{mg}, viene assimilato nel flusso magnetico concatenato λ_d. Ciò è giustificato dal fatto che l'asse d del sistema sincrono viene convenzionalmente posizionato lungo l'asse del flusso del magnete, in caso sia presente, o lungo il percorso a minor riluttanza, nel caso di motori SynR. Il bilancio di tensione (4.1) è infatti valido per tutte le tipologie di motori sincroni, che siano a magneti permanenti interni, superficiali o a riluttanza.

4.2 *Ricerca della relazione flusso-corrente*

Una volta definito un modello, è possibile costruire un sistema basato sulle reti neurali che permetta di approssimare la relazione flusso-corrente di qualsiasi motore sincrono. La sua conoscenza, infatti, permette di controllare con assoluta precisione ogni tipo di attuatore, raggiungendo livello di efficienza energetica impossibili da ottenere con un controllo lineare. Tale relazione però è, nella maggior parte dei casi, di difficile individuazione, a causa delle non linearità discusse in precedenza. Anche un'analisi agli elementi finiti risulta di ardua applicazione, in quanto richiede un modello accurato di geometrie e composizione dei materiali, oltre ad una conoscenza accurata del loro comportamento in presenza di campi magnetici applicati esternamente.
Una possibile soluzione è un approccio di tipo *black-box* con l'utilizzo di una rete neurale, ad esempio a funzioni radiali di base, per replicarne l'andamento. Una volta eseguita la fase di training, si andrà ad ottenere una mappa del flusso per ogni punto di lavoro, dalla quale sarà possibile, tra le altre cose, ricavare in tempo reale i parametri da applicare al controllore per ottenere la risposta desiderata, in termini di margine di fase e banda passante, in ogni condizione operativa. Ripetendone la stima nel tempo sarà inoltre possibile tracciare la variazione parametrica dell'azionamento, con la possibilità di valutare lo stato di salute del motore e prevedere eventuali guasti che potrebbero verificarsi nell'immediato futuro.

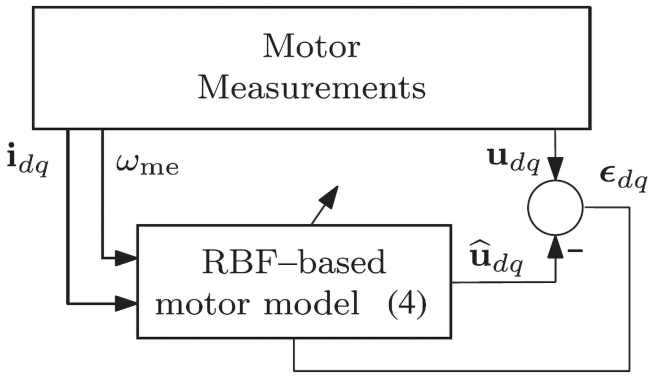


Figura 4.1. Schema di principio del modello utilizzato per addestrare la rete neurale [Ort+20].

Il sistema utilizzato per l'addestramento è riportato in figura (4.1). La struttura è simile ad un *model reference adaptive system*, in cui si utilizzano i parametri noti e le misure sperimentali per confrontare l'uscita del dispositivo reale con l'uscita del sistema stimato, al fine di correggere in retroazione il modello adattativo finché non si ottiene il medesimo output.

In particolare, l'addestramento viene eseguito in modo da minimizzare l'errore di stima, definito come $\boldsymbol{\epsilon}_{dq} = [\boldsymbol{\epsilon}_d \, \boldsymbol{\epsilon}_q]^T = \mathbf{u}_{dq} - \hat{\mathbf{u}}_{dq}$. Le tensioni stimate vengono calcolate secondo il modello:

$$\begin{aligned} \hat{u}_d &= R_s i_d + \frac{\mathrm{d}\hat{\lambda}_d(i_d, i_q)}{\mathrm{d}t} - \omega_{me}\hat{\lambda}_q(i_d, i_q) \\ \hat{u}_q &= R_s i_q + \frac{\mathrm{d}\hat{\lambda}_q(i_d, i_q)}{\mathrm{d}t} + \omega_{me}\hat{\lambda}_d(i_d, i_q) \end{aligned} \tag{4.4}$$

dove $\hat{\lambda}_d(i_d, i_q)$ e $\hat{\lambda}_q(i_d, i_q)$ sono i flussi magnetici concatenati stimati dalla rete per una determinata coppia di correnti. Ne risulta quindi una coppia di errori di stima che risponde alle seguenti relazioni:

$$\begin{aligned} \varepsilon_d &= u_d - \hat{u}_d = u_d - R_s i_d - \frac{\mathrm{d}\hat{\lambda}_d(i_d, i_q)}{\mathrm{d}t} + \omega_{me}\hat{\lambda}_q(i_d, i_q) \\ \varepsilon_q &= u_q - \hat{u}_q = u_q - R_s i_q - \frac{\mathrm{d}\hat{\lambda}_q(i_d, i_q)}{\mathrm{d}t} - \omega_{me}\hat{\lambda}_d(i_d, i_q). \end{aligned} \tag{4.5}$$

Note le tensioni, le correnti, la velocità angolare e la resistenza di statore, dall'azzeramento di tali errori si ottiene un flusso concatenato stimato pari a quello reale.
Ovviamente il risultato differirà dal valore ideale, anche in modo marcato, se i parametri forniti alla rete non sono strettamente aderenti quelli reali. Infatti, un qualsiasi errore sulle misure di tensione o di corrente, oppure sulla stima stessa di R, si ripercuoterà in un errore nella mappa dei flussi magnetici risultante. Questo accade in quanto il modello considerato è quello riportato nella (4.5), che considera come variabile da stimare i soli flussi concatenati, assumendo che gli altri dati siano corretti.

4.3 *Struttura della Rete Neurale*

Come descritto nel capitolo (1), le reti neurali possono essere considerate degli approssimatori universali. Ne esistono di diverse strutture, dalle multilayer perceptrons alle radial basis function networks, ognuna con determinate caratteristiche che le rendono più o meno adatte ad applicazioni differenti. Inoltre, ogni rete dev'essere addestrata con un algoritmo progettato ad-hoc in base a numero di neuroni, di layer e di interconnessioni tra i medesimi.
Per questa applicazione si è scelto di utilizzare una rete neurale a funzioni radiali di base, o RBF, grazie al carattere locale che la contraddistingue. Questa proprietà, infatti, porta con sé i vantaggi discussi nel capitolo (3), tra cui una buona approssimazione di mappe in più dimensioni, oltre a un addestramento relativamente rapido.
La struttura scelta è riportata in figura (4.2). Come si nota, la rete si compone di due layer, uno "nascosto", o *hidden*, in quanto non direttamente collegato con l'uscita, e il layer di *output*, da cui derivano i flussi concatenati in uscita.

4.3.1 Hidden Layer

Trattandosi di una rete RBF, il primo layer si compone di neuroni le cui funzioni di attivazione sono delle gaussiane, in sostituzione delle tradizionali sigmoidali utilizzate

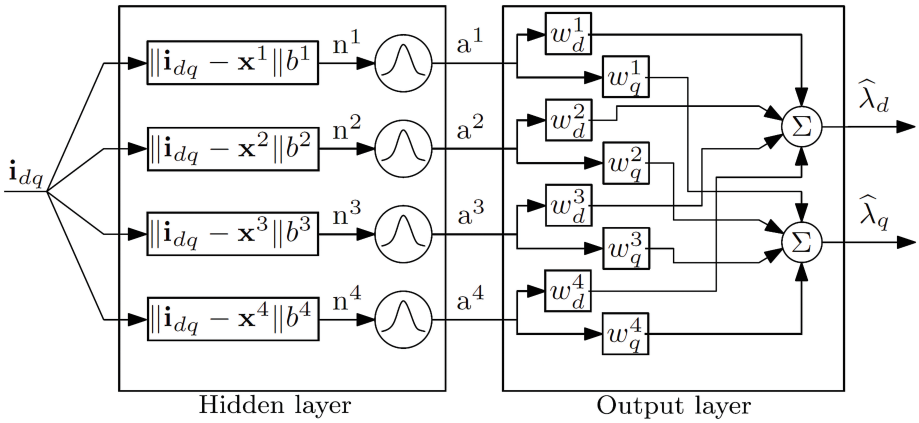


Figura 4.2. Struttura delle rete neurale a funzioni radiali di base [Ort+20].

nelle reti a percettroni. Sono proprio queste funzioni a conferire il carattere locale alla rete, grazie alla conformazione che decresce esponenzialmente all'aumentare della distanza dal centro. Ciò si traduce nel fatto che solo pochi neuroni alla volta si attivano, semplificandone notevolmente il training, altrimenti reso laborioso nel caso in cui un singolo ingresso andasse a variare l'intero set di pesi.
L'input della rete è la coppia di correnti i_d, i_q, che vanno a definire una regione di lavoro circolare all'interno dell'area delimitata dalla corrente nominale del motore, come si evince nell'immagine (4.3). Per un'implementazione più agevole, la regione considerata è stata estesa al quadrato che circoscrive tale area e, al suo interno, sono state posizionate le gaussiane equispaziate che compongono il primo layer.

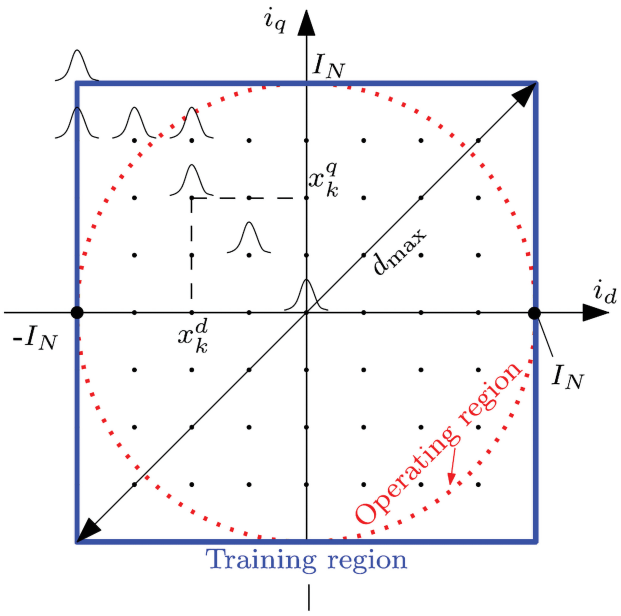


Figura 4.3. Area di training della rete neurale [OTZ18].

Questa scelta è stata fatta in quanto una diversa distribuzione è sensata solo nel caso in cui sia necessario approssimare funzioni con un diverso grado di complessità nelle differenti aree di lavoro. Non è questo il caso, in quanto il flusso magnetico presenta la stesso grado di complessità in tutta l'area di training. Eventualmente si potrebbe scegliere di aggiungere nodi nella zona dove andrà a posizionarsi la curva MTPA del motore, in cui, presumibilmente, l'azionamento andrà più spesso a lavorare. In questa fase, però, si è scelto di non farlo, in quanto si vuole creare una soluzione che possa essere applicata a tutti i tipi di motori sincroni.
Anche il numero di nodi è una scelta che dipende dalla complessità della funzione da approssimare. Un compromesso valido può essere posizionare una gaussiana ogni 25% della corrente nominale del motore, per un totale di $N_g \times N_g = 9 \times 9$ neuroni nel piano.
A questo punto è necessario definire il grado di sovrapposizione tra le funzioni. Una scelta ragionevole è quella suggerita dall'equazione:

$$b_i^k = \frac{\sqrt{N}}{d_{max}} \tag{4.6}$$

nella quale K è il numero totale di nodi e d_{max} è la massima estensione del piano in diagonale, come riportato nella figura (4.3). Secondo tale espressione, ogni nodo copre l'intera area circostante al suo centro, fino a sovrapporsi parzialmente ai neuroni adiacenti. Con questa scelta si esclude la presenza di aree non coperte da alcuna gaussiana, in modo da avere una funzione in uscita continua su tutto il piano. Con le funzioni sigmoidali, invece, ciascun nodo si sovrapporrebbe a tutti i nodi della rete, per il carattere globale che le contraddistingue.
Il funzionamento della rete RBF si riconduce, appunto, a quello delle funzioni radiali di base, la cui uscita dipende dalle coordinate dell'ingresso. Lo scopo di questo layer è di individuare quali neuroni sono coinvolti nel calcolo del flusso stimato e, per fare ciò, il primo passo è quello di computare la distanza euclidea che intercorre tra il vettore di corrente $\mathbf{i}_{dq}$ ed ognuno dei neuroni della rete, il cui k-esimo ha coordinate $\mathbf{x}^k = (x_d^k, x_q^k)$. L'espressione che ne risulta è quindi:

$$n^k = ||\mathbf{i}_{dq} - \mathbf{x}^k||b^k, \quad k = 1 \dots K \tag{4.7}$$

Dato che le singole gaussiane decrescono esponenzialmente all'aumentare della distanza dall'origine dell'ingresso, solo quelle in cui la (4.7) risulta sufficientemente ridotta parteciperanno al valore in uscita, in base al bias definito nella (4.6).
L'output del primo layer sarà quindi il seguente:

$$a^k = e^{-(n^k)^2} = e^{-(||\mathbf{i}_{dq} - \mathbf{x}^k||b^k)^2}, \quad k = 1 \dots K \tag{4.8}$$

in cui ognuno degli a^k costituirà un ingresso per il secondo livello della rete (k indice del nodo, non esponente).

4.3.2 Output Layer

Il secondo strato è invece quello che convenzionalmente si adotta in ogni rete neurale, comprese le multilayer perceptron. É di tipo lineare, come riportato in figura (4.2), e il

suo scopo è quello di computare la sommatoria pesata degli output del primo layer, una per ogni asse.
L'equazione che lo caratterizza è la seguente:

$$\hat{\lambda}_d = \sum_{k=0}^{K} w_d^k e^{-(n^k)^2} \qquad \hat{\lambda}_q = \sum_{k=0}^{K} w_q^k e^{-(n^k)^2}. \tag{4.9}$$

I pesi w_d e w_q applicati ad ogni nodo sono oggetto del training che verrà descritto nel seguente paragrafo. É però già possibile osservare come il risultato ottenuto nella (4.9) sia una funzione continua, al contrario di quanto si otterrebbe con una semplice look-up table, nella quale sarebbe necessario interpolare i dati tra due valori successivi per ottenere un risultato analogo.
É inoltre importante ricordare che, come descritto nel paragrafo (3.2), i parametri dei due layer sono indipendenti. Infatti una volta posizionati i centri delle gaussiane e imposte le varianze, l'addestramento è limitato alla ricerca del vettore $\mathbf{w}_{dq}$ che minimizzi l'errore $\boldsymbol{\epsilon}_{dq}$.

4.4 *Addestramento della rete*

L'addestramento del secondo layer della rete, ovvero la ricerca del vettore di pesi ottimo $\mathbf{w}_{dq}$, si compone di due fasi: la prima in cui viene effettuata la raccolta dati, ovvero vengono salvate in memoria le informazioni su correnti, tensioni e velocità angolare, la seconda in cui avviene effettivamente il training della rete mediante un algoritmo di minimizzazione dell'errore, il Levemberg-Marquardt.

4.4.1 Raccolta dati

Per ottenere un risultato attendibile dall'algoritmo di training è necessario raccogliere una quantità sufficiente di dati, il più possibile distribuiti nell'area in cui si andrà ad addestrare la rete. A tale scopo, si è deciso di suddividere la linea temporale come riportato in figura (4.4).

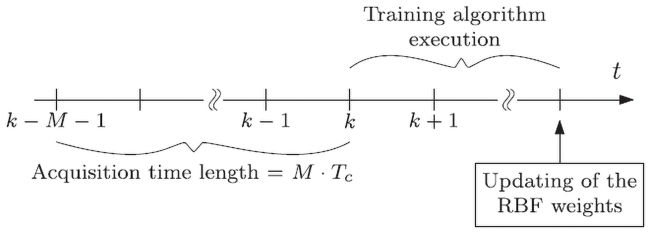


Figura 4.4. Fasi di training della rete neurale [Ort+20].

Innanzitutto vengono salvati in memoria M campioni, uno ad ogni T_c, periodo di controllo del sistema. Ogni campione contiene cinque differenti grandezze:

- le due correnti misurate, $i_{d,meas}$, $i_{q,meas}$

- le due tensioni di riferimento che vengono proposte dai controllori PI di corrente, $u_{d,ref}$, $u_{q,ref}$
- la velocità elettrica del rotore ω_{me}, derivata dalla misura di posizione dell'encoder collegato all'albero del motore.

La scelta di utilizzare le tensioni di riferimento si riconduce al fatto che un sistema in grado di misurare tali grandezze in modo rapido ed accurato porterebbe ad un aumento del costo dell'azionamento non indifferente, principalmente a causa dell'ADC a due canali che sarebbe necessario aggiungere [ATZ15]. Per tale motivo, sarà necessario adottare una serie di strategie per fare in modo che le tensioni di riferimento e quelle che effettivamente arrivano alle fasi del motore coincidano. Come verrà analizzato nel capitolo (5), questo non è sempre vero a causa delle non idealità dell'inverter e delle approssimazioni che si vanno ad introdurre nel controllo. L'errore commesso, però, dev'essere il più piccolo possibile, in quanto può portare ad un addestramento della rete non ottimale.
In generale, il metodo utilizzato non necessita di condizioni di funzionamento particolari per il motore, essendo applicabile sia in condizioni di regime, o *steady state*, con correnti e velocità costanti, che durante i transitori, grazie al modello del motore (4.1) utilizzato nella sua forma generale, senza assunzioni a priori. Come verrà evidenziato nella sezione (8.1), però, la mancanza di un sistema di misura delle tensioni porta a commettere degli errori sempre più marcati con l'aumentare della velocità di rotazione del motore, motivo per cui è preferibile effettuare l'addestramento dove ω_m presenta un valore sufficientemente ridotto. Nel caso l'attuatore sia dotato di magneti permanenti, però, le prove non possono essere effettuate a velocità nulla, altrimenti si andrebbe a perdere la componente costante di λ_d nel bilancio di tensione, rendendo impossibile per la rete stimare il flusso del magnete.
Per quanto riguarda il numero di misure da acquisire, è strettamente correlato alla quantità di memoria disponibile per il salvataggio dei dati. Considerando, ad esempio, di raccogliere campioni in formato floating-point a singola precisione (32 bit), con una frequenza di campionamento di 10 kHz, 5 dati alla volta ($i_{dq}, u_{dq}, \omega_{me}$), si otterrà un'occupazione di memoria di circa 200 kB al secondo. Come accennato in precedenza, un numero di campioni più elevato si traduce in un addestramento più accurato, complice la maggior area di training coperta. Bisogna però considerare anche le limitazioni hardware, come la quantità di memoria a disposizione e la velocità di calcolo del microprocessore, onde evitare di ottenere tempi di training eccessivamente elevati. Ovviamente il fatto di effettuare un addestramento offline consente di trascurare il problema del tempo di esecuzione, ma non quello della memoria, quest'ultimo strettamente legato alla configurazione hardware di cui si dispone.
Come verrà analizzato nel paragrafo successivo, un algoritmo di questo tipo difficilmente può essere eseguito in real-time in quanto computazionalmente oneroso, complice l'inversione di matrice di dimensioni considerevoli richiesta nell'addestramento dei pesi. Una possibile strategia è quella di eseguire il codice durante i tempi morti del microprocessore, il quale solitamente rimane in idle per la maggior parte del periodo di controllo, dopo aver effettuato i calcoli necessari per comandare l'inverter in modo opportuno.
Un'alternativa è quella di trasmettere i dati in remoto ad un secondo dispositivo adibito unicamente a tale scopo. Questa seconda metodologia è applicabile in tutti quegli azionamenti che prevedano un collegamento alla rete, ormai sempre più diffusi nelle industrie

moderne.

4.4.2 Algoritmo di training

Il training della rete avviene dopo la conclusione della raccolta dati, come illustrato in figura (4.4).
Come descritto in precedenza, l'obiettivo è quello di tarare il set di $2K$ pesi, $\mathbf{w}_d = \left[w_d^1 \dots w_d^K\right]^T$ e $\mathbf{w}_q = \left[w_q^1 \dots w_q^K\right]^T$, affinché i flussi stimati $\hat{\lambda}_d$ e $\hat{\lambda}_q$ coincidano con quelli reali. Per fare ciò, è stato scelto un algoritmo di tipo *error-correction*, che vada a minimizzare la funzione di costo quadratica definita come segue:

$$E(\mathbf{w}_d, \mathbf{w}_q) = \frac{1}{2} \sum_{i=1}^{M} (\varepsilon_d^i)^2 + (\varepsilon_q^i)^2 = \frac{1}{2} \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \tag{4.10}$$

dove il vettore degli errori $\boldsymbol{\epsilon} = [\varepsilon_d^1 \dots \varepsilon_d^M \; \varepsilon_q^1 \dots \varepsilon_q^M]$ è ottenuto con l'equazione (4.5).
Dato che l'informazione sul flusso compare in entrambe le equazioni, la ricerca dei pesi che minimizzino il costo non può essere separata tra i due assi, a meno di porre la condizione di regime stazionario, ovvero a correnti costanti. Questa ipotesi permetterebbe di eliminare il termine di derivata del flusso, perciò con la tensione di asse d si andrebbe ad identificare il solo flusso concatenato di asse q e viceversa.
Escludendo questa ipotesi, è necessario ricercare il vettore dei pesi nel suo insieme, ovvero $\mathbf{w} = [\mathbf{w}_d^T; \mathbf{w}_q^T]^T$. Per l'addestramento si è scelto di utilizzare l'algoritmo di training di Levemberg-Marquardt descritto nel paragrafo (2.3), il quale, al passo h, evolve secondo l'equazione:

$$\mathbf{w}^{h+1} = \mathbf{w}^h - [\mathbf{J}^T\mathbf{J} + \mu^h\mathbf{I}]^{-1}\mathbf{J}^T\boldsymbol{\epsilon} \tag{4.11}$$

dove $\mathbf{J} \in \mathbb{R}^{2M \times 2K}$ è la matrice Jacobiana del sistema, mentre μ^h, imposta costante ad ogni passo, serve a garantire l'invertibilità della matrice $[\mathbf{J}^T\mathbf{J} + \mu^h\mathbf{I}]$.
Tale algoritmo si contraddistingue per la sua rapidità di convergenza, infatti, dopo una sola epoca, il risultato ottenuto è più che soddisfacente. Per una maggior precisione, è comunque possibile continuare l'apprendimento fino a che il costo definito dalla (4.10) non smette di decrescere tra un'iterazione e la successiva, fissando una soglia, ad esempio, pari a $\mathbf{E}_{th} = 1\%$, compromesso tra l'accuratezza della stima e il tempo di esecuzione.
Si è scelto inoltre di porre il valore iniziale dei pesi $\mathbf{w}^0$ pari ad un vettore nullo. In alternativa, poteva essere preimpostato considerando in prima istanza un motore lineare, con un unico valore per l'induttanza differenziale di ciascun asse.
Gli elementi della Jacobiana, invece, sono calcolati come le derivate degli errori di stima di ciascun asse rispetto ai pesi soggetti al training, ovvero quelli del secondo layer della

rete:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \varepsilon_d^1}{\partial w_d^1} & \cdots & \frac{\partial \varepsilon_d^1}{\partial w_d^K} & \frac{\partial \varepsilon_d^1}{\partial w_q^1} & \cdots & \frac{\partial \varepsilon_d^1}{\partial w_q^K} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_d^M}{\partial w_d^1} & \cdots & \frac{\partial \varepsilon_d^M}{\partial w_d^K} & \frac{\partial \varepsilon_d^M}{\partial w_q^1} & \cdots & \frac{\partial \varepsilon_d^M}{\partial w_q^K} \\ \frac{\partial \varepsilon_q^1}{\partial w_d^1} & \cdots & \frac{\partial \varepsilon_q^1}{\partial w_d^K} & \frac{\partial \varepsilon_q^1}{\partial w_q^1} & \cdots & \frac{\partial \varepsilon_q^1}{\partial w_q^K} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_q^M}{\partial w_d^1} & \cdots & \frac{\partial \varepsilon_q^M}{\partial w_d^K} & \frac{\partial \varepsilon_q^M}{\partial w_q^1} & \cdots & \frac{\partial \varepsilon_q^M}{\partial w_q^K} \end{bmatrix} \tag{4.12}$$

Ne risulta una matrice di $2M$ righe, M numero di campioni acquisiti, e $2K$ colonne, K numero di nodi, divisa in quattro blocchi, $\frac{\partial \boldsymbol{\varepsilon}_d}{\partial \mathbf{w}_d}$, $\frac{\partial \boldsymbol{\varepsilon}_d}{\partial \mathbf{w}_q}$, $\frac{\partial \boldsymbol{\varepsilon}_q}{\partial \mathbf{w}_d}$, $\frac{\partial \boldsymbol{\varepsilon}_q}{\partial \mathbf{w}_q}$.
Ogni elemento della matrice (4.12) può essere calcolato derivando le equazioni della (4.5) rispetto ai pesi w_d e w_q. Di seguito è riportato un esempio di calcolo per ciascun blocco della Jacobiana.
Innanzitutto il primo blocco nella diagonale inversa viene calcolato come:

$$\begin{aligned} \frac{\partial \varepsilon_d^1}{\partial w_q^1} &= \frac{\partial (u_d - R_s i_d - \frac{\mathrm{d}\hat{\lambda}_d(i_d,i_q)}{\mathrm{d}t} + \omega_{me}\hat{\lambda}_q(i_d,i_q))}{\partial w_q^1} \\ &= \omega_{me}\frac{\partial \hat{\lambda}_q(i_d,i_q)}{\partial w_q^1} = \omega_{me}\frac{\partial \sum_1^K w_q^k a^k}{\partial w_q^1} = \omega_{me}a^1 \end{aligned} \tag{4.13}$$

mentre il secondo:

$$\begin{aligned} \frac{\partial \varepsilon_q^1}{\partial w_d^1} &= \frac{\partial (u_q - R_s i_q - \frac{\mathrm{d}\hat{\lambda}_q(i_d,i_q)}{\mathrm{d}t} - \omega_{me}\hat{\lambda}_d(i_d,i_q))}{\partial w_d^1} \\ &= -\omega_{me}\frac{\partial \hat{\lambda}_d(i_d,i_q)}{\partial w_d^1} = -\omega_{me}\frac{\partial \sum_1^K w_d^k a^k}{\partial w_d^1} = -\omega_{me}a^1 = -\frac{\partial \varepsilon_d^1}{\partial w_q^1} \end{aligned} \tag{4.14}$$

Passando poi agli elementi dei blocchi nella diagonale principale, il calcolo richiede un passaggio ulteriore a causa della dipendenza dal tempo:

$$\begin{aligned} \frac{\partial \varepsilon_d^1}{\partial w_d^1} &= \frac{\partial (u_d - R_s i_d - \frac{\mathrm{d}\hat{\lambda}_d(i_d,i_q)}{\mathrm{d}t} + \omega_{me}\hat{\lambda}_q(i_d,i_q))}{\partial w_d^1} \\ &= -\frac{\partial}{\partial w_d^1}\frac{\mathrm{d}\hat{\lambda}_d(i_d,i_q)}{\mathrm{d}t} = -\frac{\partial}{\partial w_d^1}\frac{\mathrm{d}\sum_1^K w_d^k a^k}{\mathrm{d}t} = -\frac{\mathrm{d}a^1}{\mathrm{d}t} \end{aligned} \tag{4.15}$$

Si ha però che gli elementi tra i due blocchi coincidono:

$$\begin{aligned} \frac{\partial \varepsilon_q^1}{\partial w_q^1} &= \frac{\partial (u_d - R_s i_d - \frac{\mathrm{d}\hat{\lambda}_q(i_d,i_q)}{\mathrm{d}t} - \omega_{me}\hat{\lambda}_d(i_d,i_q))}{\partial w_q^1} \\ &= -\frac{\partial}{\partial w_q^1}\frac{\mathrm{d}\hat{\lambda}_q(i_d,i_q)}{\mathrm{d}t} = -\frac{\partial}{\partial w_q^1}\frac{\mathrm{d}\sum_1^K w_q^k a^k}{\mathrm{d}t} = -\frac{\mathrm{d}a^1}{\mathrm{d}t} = \frac{\partial \varepsilon_d^1}{\partial w_d^1} \end{aligned} \tag{4.16}$$

In queste equazioni si assume che R_s sia nota e indipendente dal tempo. Il suo valore può essere ricavato dai dati di targa del motore, supponendo che la resistenza degli avvolgimenti del motore reale coincida con il valore nominale, oppure misurato, ad esempio attuando una misura a quattro fili mediante un multimetro collegato tra le fasi.
Per ottenere un valore che più si avvicini alla resistenza reale del circuito collegato ai capi della scheda di controllo, comprendente quindi sia il motore che l'inverter, si può attuare la strategia illustrata nel paragrafo (5.3), che prevede vengano iniettati, tra due fasi del motore, una serie di gradini di corrente e che, a transitorio concluso, se ne misuri la risposta in tensione, al fine di ottenere una mappa tensione-corrente il più aderente possibile alla caratteristica del motore, la cui derivata è, appunto, la resistenza delle fasi.
Per quanto riguarda il calcolo della derivata del termine a^k nelle equazioni (4.15) e (4.16), è possibile ottenere il risultato desiderato ricordando l'equazione (4.8), arrivando all'espressione:

$$\begin{aligned}\frac{\mathrm{d}a^k}{\mathrm{d}t} &= \frac{\mathrm{d}(e^{-(n^k)^2})}{\mathrm{d}t} = \frac{\mathrm{d}(e^{-((i_d-x_d^k)^2+(i_q-x_q^k)^2)(b^k)^2})}{\mathrm{d}t} \\ &= -2a^k(b^k)^2\left[(i_d - x_d^k)\frac{\mathrm{d}i_d}{\mathrm{d}t} + (i_q - x_q^k)\frac{\mathrm{d}i_q}{\mathrm{d}t}\right]\end{aligned} \tag{4.17}$$

Come si nota, l'equazione (4.17) richiede il calcolo delle derivate di corrente. Dato che i campioni acquisiti sono, nella quasi totalità dei casi, affetti da un certo rumore di misura sovrapposto, è preferibile applicarvi un filtraggio di tipo passa basso prima di effettuare l'operazione di derivazione descritta, onde evitare degli impulsi di entità elevata nella $\frac{\mathrm{d}a^k}{\mathrm{d}t}$.
Qualsiasi filtro, però, introduce un ritardo nella misura, rendendo le varie quantità coinvolte nell'addestramento sfasate nel tempo. Per tale motivo, si è scelto di applicare il medesimo filtro a tutte le grandezze, $\mathbf{i}_{dq}$, $\mathbf{u}_{dq}$ e ω_{me}. In questo caso, la frequenza di taglio è stata impostata a 1 kHz, compromesso tra il rumore residuo nelle misure e la perdita di informazione nel segnale.
Per completezza, è riportato di seguito uno pseudocodice del programma utilizzato per il training della rete:

Algorithm 1 RBF Training

1: Calcolo matrice Jacobiana $\mathbf{J}$ (4.12)
2: Inizializzazione vettore dei pesi $\mathbf{w}^0$
3: **do**
4: Calcolo dei flussi stimati (4.9)
5: Calcolo errori di stima (4.5)
6: Computazione funzione di costo (4.10)
7: Esecuzione algoritmo di Levemberg-Marquardt (4.11)
8: **while** $E(\mathbf{w}_d, \mathbf{w}_q) > E_{th}$

5

SET-UP SPERIMENTALE

In questo capitolo verranno descritti i vari componenti del banco di prova utilizzato. In particolare, si andranno a giustificare le scelte fatte riguardo ai parametri della rete neurale in relazione alle risorse hardware a disposizione, come memoria volatile, potenza di calcolo del microprocessore e sensori per la raccolta dati.
Lo schema di principio del banco utilizzato per testare il funzionamento dell'algoritmo è riportato in figura (5.1).

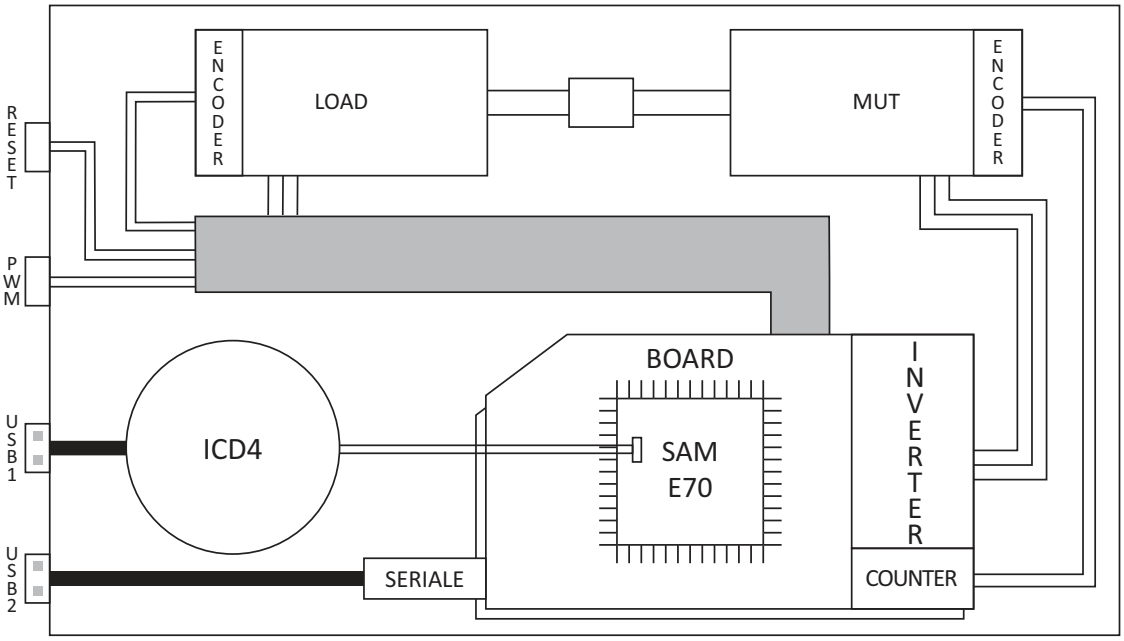


Figura 5.1. Schema di principio del banco di prova utilizzato.

Nella configurazione testata sono presenti due motori sPMSM, uno *under test* e uno che funge da carico, ciascuno dotato del proprio encoder incrementale. Il collegamento tra i due è effettuato mediante un giunto elastico. Ogni motore è controllato da una scheda dedicata dotata di inverter, porte seriali e un microprocessore industriale. Per la programmazione è stato utilizzato un dispositivo apposito fornito da *Microchip*, denominato *ICD4*, in grado di leggere e scrivere la memoria del micro, oltre ad essere dotato della modalità di debug.
Infine, il banco presenta due pulsanti, oltre allo switch di alimentazione, uno per accendere la PWM e uno per il reset delle due schede.

5.1 *Scheda di controllo*

Sono innanzitutto presenti due schede di controllo a bassa tensione *MCLV-2 Development Board* fornite da *Microchip*, una per ogni motore, di cui si riassumono le caratteristiche più rilevanti:

- Alimentazione della scheda da 16 V a 24 V
- Inverter trifase in grado di pilotare motori fino a 48 V, con una corrente nominale di 10 A
- Interfaccia per i sensori di Hall/Encoder in quadratura per il controllo *sensored* di motori
- Resistenza di *shunt* sul bus in continua
- Resistenze di *shunt* sulle fasi del motore per la misura della corrente
- Interfaccia per la comunicazione *CAN-bus*
- Interfaccia per la comunicazione seriale *USB* e *RS-232*

L'alimentatore utilizzato è in grado di stabilizzare la tensione del bus in continua a 16 V, sufficienti per il controllo dei motori in analisi (descritti nel paragrafo (5.4)).
La configurazione della scheda permette la misura della posizione mediante l'encoder a due canali (in quadratura) calettato all'albero motore. Per la corrente, invece, si utilizzano le resistenze di shunt presenti in due delle tre fasi. Il segnale che ne deriva dovrà poi essere convertito in digitale dall'ADC del microcontrollore, prima di poter essere utilizzato per il controllo del motore. Come sistema di misura non è tra i più accurati. Il dato risultante, infatti, avrà un certo rumore sovrapposto, motivo per cui si ricorre al filtro passa basso descritto nel paragrafo (4.4.2) prima di applicarvi l'operazione di derivata.
Passando poi alla connessione tra i dispositivi, le due schede comunicano tra loro grazie ad un protocollo seriale. In particolare, viene sfruttato lo standard *CAN-bus*, in cui la scheda che controlla il motore under test viene identificata come *master*, mentre la scheda collegata al motore che funge da carico è lo *slave*. Nella soluzione implementata, il master si limita a comunicare allo slave quando effettuare l'allineamento iniziale del motore e con che parametri pilotare il carico, ovvero se effettuare un controllo di coppia o di velocità, e con che percentuale rispetto al valore nominale.
Per applicazioni più sviluppate, una comunicazione di questo tipo può essere sfruttata per trasferire dati tra le due schede, al fine di "raddoppiare" la potenza di calcolo e la memoria a disposizione. Ovviamente il limite di una struttura di questo tipo è la velocità con cui vengono trasferiti i bit. In questo caso il ricetrasmettitore integrato è un *MCP2551*, il quale è in grado di arrivare fino a *1 Mb/s*, una velocità che può essere considerata sufficientemente elevata per la maggior parte delle applicazioni industriali, a condizione di filtrare adeguatamente i disturbi sovrapposti sulla linea.
L'interfaccia *USB*, invece, serve per comunicare con il PC collegato alla scheda master. Attraverso la linea vengono trasmessi i comandi per la scheda che partono dall'interfaccia *Simulink* del programma, ad esempio l'enable per iniziare il test per l'addestramento della rete neurale o il modulo della corrente da fornire al motore, mentre il microcontrollore invia al computer i dati da visualizzare in output sul monitor seriale.

5.2 *Microcontrollore*

Il microcontrollore integrato nella scheda è un *SAM E70* prodotto da Microchip, basato sul processore *Arm Cortex-M7*, con architettura a 32 *bit*.
Le principali caratteristiche del dispositivo sono le seguenti:

- Arm Cortex M7 con clock a 300 MHz
- 16 kB di cache per le istruzioni, 16 kB per i dati
- Unità *floating-point* hardware con singola e doppia precisione
- 2048 kB di memoria Flash integrata per il programma
- 384 kB di memoria dati SRAM integrata
- Timer/Counter a 16 bit e tre canali, con possibilità di operare in modalità *Capture*, *Compare*, *Waveform* e *PWM*, oltre che per la lettura di encoder in quadratura
- 74 canali per richieste di interrupt con 8 livelli di priorità
- Generatore di *dead Time* operante sull'uscita PWM
- 1 *DAC* a 2 canali a 12 bit
- 2 *ADC* a 2 canali a 12 bit
- 3 ricevitori/trasmettitori *USART*, 2 *SPI*, 2 *CAN* e 1 *USB*

Oltre alla potenza di calcolo derivante dai 300 MHz di clock, più che sufficienti nella maggior parte delle applicazioni industriali, le caratteristiche più rilevanti per questa applicazione sono la possibilità di operare in virgola mobile, in singola e doppia precisione, e la quantità di memoria a disposizione. Se da un lato la porzione dedicata alle istruzioni è più che sufficiente, 2048 kB, la *SRAM* per i dati è in realtà piuttosto limitata, "soli" 384 kB.
Il problema nasce dalla necessità di calcolare e memorizzare fino al termine dell'addestramento la matrice Jacobiana che, come descritto nel paragrafo (4.4.2), presenta un'estensione di $2M$ righe, M numero di campioni, e $2K$ colonne, K numero di nodi. Supponendo di suddividere il piano di addestramento in 36 nodi, di aver raccolto 200 campioni e di utilizzare un formato a 32 bit per la costruzione della matrice, si ottiene un'occupazione di memoria di circa 120 kB, circa un terzo di quella disponibile, solo per la Jacobiana.
A questa va aggiunta:

- la memoria occupata per i campioni, $5M$
- quella della matrice $\mathbf{A}$ con le uscite del primo layer, necessaria per il calcolo dei flussi, di estensione $M \times K$
- quella occupata dalle matrici temporanee, $\mathbf{J}^T\mathbf{J}$, di dimensione $2K \times 2K$
- l'inversa della stessa, di pari estensione, $2K \times 2K$
- dei vettori degli errori, ognuno di lunghezza M
- dei flussi stimati, anch'essi $2M$
- dei pesi al passo corrente e al passo precedente, $2 \times 2K$.

Con i parametri ipotizzati in precedenza, $M = 200$ e $K = 36$, si ottiene un'occupazione di memoria di circa 275 kB per le sole variabili inerenti al training della rete, considerando

di lavorare con floating point in doppia precisione, esclusa la Jacobiana che, per limitarne l'occupazione, è stata calcolata in singola precisione. Quanto appena descritto è riassunto in tabella (5.1).

Variabile	**Elementi**	**Formato**	**Dimensione**
$i_{dq}, u_{dq}, \omega_{me}$	$5 \times M$	$64bit$	8 kB
$\frac{\mathrm{d}i_d}{\mathrm{d}t}, \frac{\mathrm{d}i_q}{\mathrm{d}t}$	$2 \times M$	$64bit$	3.2 kB
$\mathbf{x}$	$K \times 2$	$64bit$	0.6 kB
$\mathbf{A}$	$M \times K$	$64bit$	57.6 kB
$\mathbf{J}$	$2M \times 2K$	$32bit$	115.2 kB
$\mathbf{J}^T\mathbf{J} + \mu I$	$2K \times 2K$	$64bit$	41.5 kB
$[\mathbf{J}^T\mathbf{J} + \mu I]^{-1}$	$2K \times 2K$	$64bit$	41.5 kB
$\hat{\lambda}_d, \hat{\lambda}_q$	$2 \times M$	$64bit$	3.2 kB
$\boldsymbol{\epsilon}_d, \boldsymbol{\epsilon}_q$	$2 \times M$	$64bit$	3.2 kB
$\mathbf{w}_d, \mathbf{w}_q$	$2 \times K$	$64bit$	0.6 kB
$\mathbf{w}_{d,prec}, \mathbf{w}_{q,prec}$	$2 \times K$	$64bit$	0.6 kB

Tabella 5.1. Occupazione di memoria.

A ciò vanno aggiunte tutte le aree di memoria occupate dalle variabili singole, a 32 o 64 *bit*, arrivando ad un'occupazione di memoria quasi totale.
Per cercare di limitare il problema, sono state adottate alcune strategie alternative: innanzitutto si è cercato di gestire la memoria allocando dinamicamente il maggior numero di variabili possibile, liberando lo spazio di quelle inutilizzate appena consentito. Questo però provocava una forte imprevedibilità nel software, presumibilmente per le difficoltà della scheda a leggere e scrivere continuamente in ampie aree di memoria dinamica. Si è quindi cercato di limitare lo spazio occupato effettuando tutte le operazioni con variabili a singola precisione. Adottando questa strategia, però, la funzione di costo dell'algoritmo di training divergeva, rendendo impossibile la ricerca di un minimo. Si è constatato che questo inconveniente era dovuto principalmente al fatto che, con una matrice di esponenziali $\mathbf{A}$ con poca risoluzione, si commetteva un errore sulla Jacobiana (e nella sua trasposta) tale per cui l'operazione matriciale $\mathbf{J}^T\mathbf{J}$ portava, a causa della sommatoria intrinseca in questo calcolo, ad un errore non trascurabile. Si è risolto tale inconveniente aumentando nuovamente la risoluzione dei calcoli in virgola mobile, a costo di un'occupazione di memoria superiore, e limitando il numero di nodi e di campioni acquisiti, per risparmiare spazio.

Per la PWM, invece, si è scelto di utilizzare un timer a 16 *bit* con una frequenza di clock di 150 MHz. I registri sono stati settati per effettuare un conteggio *up&down*, con l'inversione in corrispondenza delle soglie 0 e 9375 unità. In questo modo si ottiene un'onda triangolare come quella riportata in figura (5.2), con una frequenza di 8 kHz.
Calcolando adeguatamente le soglie di commutazione di ciascuno dei tre canali uscenti dal timer, *OC0*, *OC1*, *OC2*, è possibile utilizzare tali segnali per pilotare l'inverter con una modulazione PWM di tipo simmetrico.

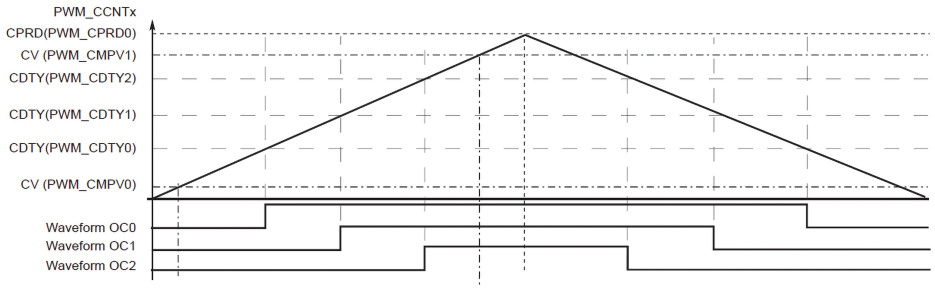


Figura 5.2. Segnale generato dal counter per la modulazione PWM [20b].

5.3 *Inverter*

Per quanto riguarda l'inverter integrato nella scheda, lo schema è quello trifase a sei interrutori riportato in figura (5.3).

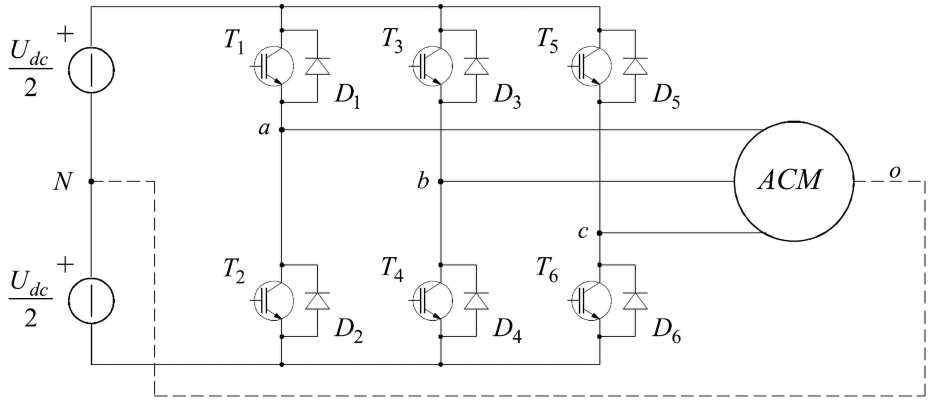


Figura 5.3. Schema dell'inverter trifase [PZ11].

L'alimentazione viene prelevata da un bus in continua, in questo caso l'alimentatore a 16 V che alimenta la scheda. Sono poi presenti sei interruttori di potenza che formano i tre rami dell'inverter, ognuno con il proprio diodo di *freewheeling* in antiparallelo. La funzione di questi componenti, a volte realizzati sfruttando le proprietà di conduzione inversa del MOSFET stesso, è quella di permettere la circolazione di corrente quando l'interruttore si apre, onde evitare di interrompere bruscamente il flusso di elettroni e causare, di conseguenza, picchi di tensione potenzialmente distruttivi.
Comandando opportunamente la chiusura o l'apertura dei vari interruttori si vanno ad applicare dei vettori di tensione al motore: gli stati possibili sono otto, di cui due corrispondenti al vettore nullo, che generano una stella di vettori spaziali. Nel caso in esame, la modulazione attuata è di tipo simmetrico, ovvero la suddivisione temporale dei vettori applicati è speculare all'interno del periodo. In questo modo ogni T_c inizia e si conclude con il vettore nullo, limitando al minimo le armoniche di tensione.
Nella configurazione circuitale di figura (5.3) si nota un potenziale problema: i due switch dello stesso ramo non devono mai essere attivi contemporaneamente per evitare corto-

circuiti distruttivi del bus in continua. Dato che gli interruttori elettronici presentano dei tempi di commutazione da uno stato ad un altro non nulli, è necessario prevedere dei *tempi morti* tra lo spegnimento di uno e l'accensione dell'altro. Questo accorgimento, però, provoca una distorsione nella tensione generata, come verrà analizzato a breve.
Le proprietà più rilevanti dei MOSFET utilizzati, la cui sigla è *FQP55N10,* sono riportate in tabella (5.2). Si nota dai parametri elettrici ($V_{DSS,max} = 100\,V$, $I_{D,max} = 55\,A$) come gli

Electrical Characteristics			
Symbol	**Parameter**	**Max Value**	**Unit**
V_{DSS}	Drain-Source Max Voltage	100	$[V]$
I_D	Drain Current -Continuous	55	$[A]$
I_{DM}	Drain Current -Pulsed	220	$[A]$
V_{GSS}	Gate-Source Max Voltage	±25	$[V]$
$V_{GS(th)}$	Gate-Source Threshold Voltage	4	$[V]$
$R_{DS(on)}$	Drain-Source On Resistance	0.026	$[\Omega]$
V_{SD}	Drain-Source Diode Forward Voltage	1.5	$[V]$
Switching Characteristics			
$t_{d(on)}$	Tempo di accensione	60	$[ns]$
$t_{d(off)}$	Tempo allo spegnimento	230	$[ns]$
t_r	Tempo di salita all'accensione	510	$[ns]$
t_f	Tempo di discesa allo spegnimento	290	$[ns]$
t_{rr}	Tempo di recovery inverso del diodo	100	$[ns]$

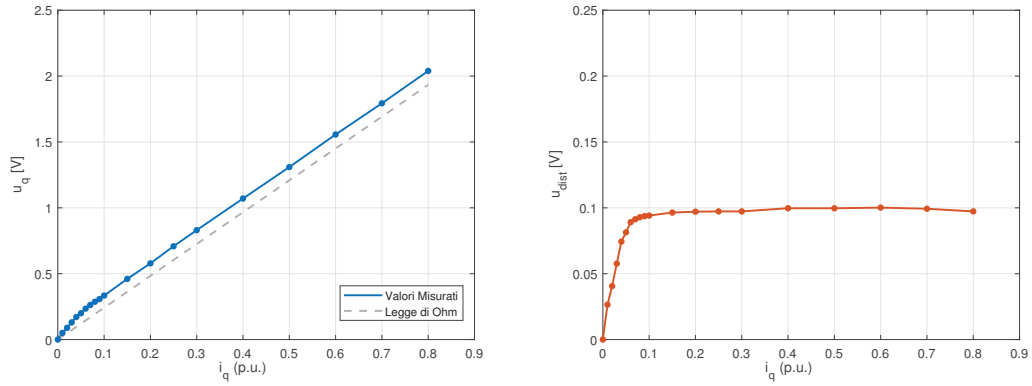
Tabella 5.2. Parametri dei MOSFET.

switch utilizzati siano sovradimensionati rispetto all'applicazione. Questo però non è da considerarsi uno svantaggio, in quanto, con una corrente che vi scorre ridotta rispetto a quella massima, la caduta sul componente in conduzione sarà trascurabile. Lo stesso non vale per i diodi in antiparallelo, i quali, essendo interni al componente e non ottimizzati per la conduzione, presentano una caduta superiore ($V_{SD} = 1.5\,V$) ad un normale diodo posizionato esternamente all'interruttore.
I parametri su cui però è necessario effettuare un'analisi più approfondita sono i tempi di accensione e di spegnimento dei dispositivi, oltre che i tempi di salita e di discesa, che intercorrono tra la commutazione del comando sul Gate e l'effettivo cambio di stato del MOSFET. Questi ritardi sono dovuti alla presenza delle capacità parassite tra il Drain e il Source del componente, oltre a quella sul gate, perciò non sono eliminabili. Il costruttore può solo limitarsi a ridurre al minimo il loro impatto nel funzionamento.
Come accennato in precedenza, questa caratteristica intrinseca degli interruttori elettronici obbliga a prevedere un certo tempo (morto) t_D tra lo spegnimento e l'accensione dei MOSFET di uno stesso ramo. Questo porta ad avere una distorsione nella tensione che effettivamente arriva al motore che, per un controllo più accurato, dev'essere adeguatamente compensata in feedforward.
É possibile dimostrare che la variazione di tensione dovuta ai tempi morti è, in modulo,

pari a:

$$|\boldsymbol{u}_{dist}| = \frac{4}{3} u_{bus} \frac{t_D}{T_{PWM}} sign(\mathbf{i}) \tag{5.1}$$

Come accennato in precedenza, in questi esperimenti la tensione non viene misurata ma alla rete vengono forniti i campioni inerenti al riferimento, e più si avvicinano alla grandezza reale, più il risultato sarà accurato. É quindi necessario identificare l'effetto dei tempi morti.
Per fare ciò, a motore fermo, si applica una tensione continua al motore e, dopo un certo transitorio, si osserva la risposta di corrente, anch'essa continua. Il rapporto tra le due dovrebbe dipendere solamente dalla resistenza degli avvolgimenti di statore, e qualunque valore che si discosti dalla legge di Ohm è imputabile alla presenza di tempi morti nell'inverter.
Uno dei modi più immediati per ottenere questo risultato è applicare una certa corrente di asse q, che si tradurrà in una quantità positiva sulla fase b e negativa sulla fase c, di pari modulo. L'intera prova dev'essere eseguita imponendo un angolo θ_{me} pari a zero, per evitare di porre il motore in rotazione a velocità elevate.
Il risultato sperimentale della procedura è riportato in figura (5.4.a).



(a) (b)

Figura 5.4. Procedura di identificazione dei tempi morti: (a) Misure sperimentali, (b) Tensione di distorsione.

Come si nota, la curva ottenuta si discosta dalla legge di Ohm, riportata in tratteggio, presentando un primo tratto non lineare, per correnti ridotte, per poi procedere parallela alla curva ideale, con un certo offset.
L'andamento iniziale è imputabile al fatto che, a basse tensioni, le capacità parassite dei MOSFET tendono a compensare il vuoto di tensione dovuto ai tempi morti, e questo fenomeno si osserva fino a circa il 10% della corrente nominale.
Da questo test, inoltre, si ricava facilmente la resistenza vista ai capi del controllore, comprendente, oltre al motore, l'impedenza dei collegamenti e degli switch. Per calcolarla è sufficiente ricavare la pendenza della curva ottenuta.
Una volta fatto ciò, sottraendo alle misure di tensione la componente dovuta alla caduta resistiva, rimane solamente quella inerente ai tempi morti (figura (5.4.b)) che, sommata in feedforward nelle tre fasi, permette di ottenere una tensione sul motore più aderente al

riferimento desiderato.
Le strategie di compensazione che è possibile attuare sono molteplici: la più semplice è una compensazione a gradino, prendendo come riferimento la media dei valori ottenuti con correnti maggiori di $\frac{I_N}{10}$. Un'alternativa più raffinata è quella di compensare con una rampa nel primo tratto non lineare, per poi proseguire con la compensazione a gradino precedente.
La metodologia che si è scelto di attuare, invece, prevede l'utilizzo di una *Lookup table*, in cui sono salvati i valori di tensione di compensazione per tutto il range di correnti, da 0 a I_N, in modo da ottenere il risultato più accurato possibile. Ovviamente quanto riportato nell'immagine (5.4) riguarda solo le correnti positive. Per l'intervallo da $-I_N$ a 0, si ripropone il medesimo andamento ma con valori di tensione negativi.
Qui sorge una problematica che è necessario affrontare: se la misura di corrente presenta un certo disturbo sovrapposto, ad ogni attraversamento per lo zero la tensione di compensazione passerà da positiva a negativa molteplici volte. E una compensazione di segno sbagliato porta ad un errore doppio rispetto a non compensare affatto. Per ovviare a tale problema, si è scelto di iniziare a compensare solamente da una certa corrente in poi, circa il 2% della nominale, generando una fascia in cui l'azione è nulla. L'effetto negativo di questa soluzione è trascurabile, in quanto, come descritto in precedenza, per piccole correnti la caduta dovuta ai tempi morti è parzialmente attenuata dalle capacità parassite, rendendo la compensazione superflua.

5.4 *Motori Elettrici*

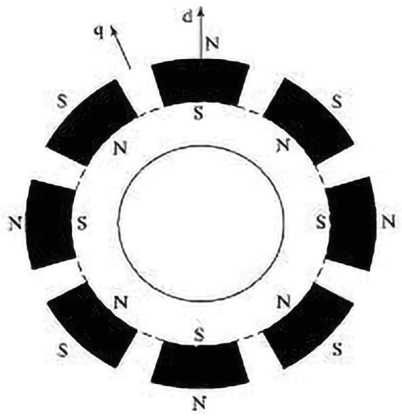


Figura 5.5. Schema del motore sincrono a magneti permanenti superficiali [Car+17].

Il dispositivo sul quale sono stati eseguiti i vari test è un motore sincrono a magneti permanenti superficiali, la cui struttura è riportata in figura (5.5). Come si nota, è caratterizzata da una distribuzione dei magneti uniforme, essendo incollati attorno alla circonferenza del rotore. Questa proprietà permette di considerare gli *sPMSM* isotropi, grazie al fatto che i magneti permanenti presentano una permeabilità magnetica simile all'aria. Non influenzano quindi la riluttanza del percorso magnetico, rendendo l'induttanza dei due

assi pressoché identica.
Inoltre, per la proprietà appena descritta, questo tipo di motori è poco soggetto alla saturazione, permettendo, almeno in fase di progettazione del sistema di controllo, di semplificare le equazioni (4.1) come segue:

$$\begin{aligned} u_d &= R_s i_d + \frac{\mathrm{d}\lambda_d(i_d, i_q)}{\mathrm{d}t} - \omega_{me}\lambda_q(i_d, i_q) = R_s i_d + L_d \frac{\mathrm{d}i_d}{\mathrm{d}t} - \omega_{me} L_q i_q \\ u_q &= R_s i_q + \frac{\mathrm{d}\lambda_q(i_d, i_q)}{\mathrm{d}t} + \omega_{me}\lambda_d(i_d, i_q) = R_s i_q + L_q \frac{\mathrm{d}i_q}{\mathrm{d}t} + \omega_{me}(L_d i_d + \Lambda_{mg}) \end{aligned} \tag{5.2}$$

Ovviamente questa approssimazione porta a commettere un certo errore, soprattutto nelle regioni di lavoro dove l'induttanza differenziale diminuisce a causa della saturazione. Per tale motivo, al fine di ottenere un controllo più preciso ed efficiente, è utile avere a disposizione la mappa completa del flusso magnetico concatenato.
Per quanto riguarda i parametri del motore, alcuni di essi possono essere ricavati dal datasheet fornito dal costruttore. Spesso, però, molti sono mancanti o approssimativi, in quanto rilevati a campione o solo stimati. Per tale motivo, prima di utilizzare un nuovo motore, è utile effettuare delle rilevazioni aggiuntive, al fine di ricavare dei dati precisi.
Per quanto riguarda la resistenza delle fasi di statore, è sufficiente una misurazione mediante un multimetro in modalità *quattro fili* tra due fasi del motore, per poi dividere quanto ottenuto per un fattore due, al fine di ottenere la resistenza sulla singola fase. Una strategia alternativa è quella illustrata nel paragrafo (5.3), che permette di includere nel risultato anche la resistenza dei collegamenti e degli switch dell'inverter.
L'induttanza delle fasi, invece è stata ricavata partendo dalla conoscenza del circuito elettrico equivalente del motore, riportato in figura (5.6).

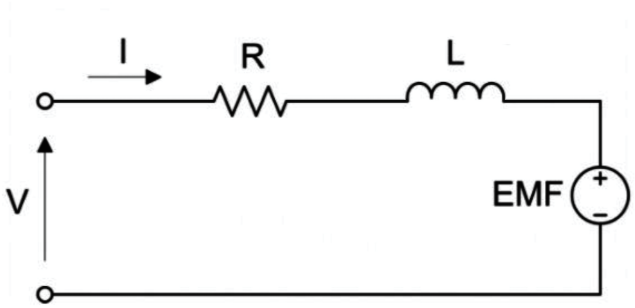


Figura 5.6. Schema elettrico equivalente del motore.

L'espressione fasoriale, a motore fermo, si riconduce all'equazione:

$$\dot{U}_d = R_s \dot{I}_d + j\Omega L_s \dot{I}_d = \dot{z}\dot{I}_d \tag{5.3}$$

con $\dot{z} = R_s + j\Omega L_s$ impedenza della fase.
La prova consiste nell'iniettare una sinusoide di tensione ad una certa frequenza su uno dei due assi, solitamente l'asse *d*, in modo da non produrre coppia, per poi misurare la risposta in corrente del motore. Il modulo dell'impedenza si ricava come rapporto tra la tensione e la corrente, mentre la fase si trova conoscendo lo sfasamento temporale tra le

due, con il legame $\varphi = \Omega\,\Delta t$. Dall'espressione di $\dot{z}$ è immediato identificare la parte reale come la componente resistiva del circuito, mentre la parte immaginaria, che equivale al modulo moltiplicato al seno dell'angolo di fase, è riconducibile a:

$$|\dot{z}|\sin\varphi = |\dot{z}|\sin(\Omega\,\Delta t) = \Omega L \tag{5.4}$$

dalla quale, dividendo per Ω, si ricava infine il valore dell'induttanza.
Il flusso del magnete Λ_{mg}, invece, è stato ottenuto sfruttando quanto descritto dal modello (5.2). A morsetti aperti, infatti, vale che la tensione tra due fasi è da imputare alla sola forza controelettromotrice che caratterizza il motore in rotazione. Vale perciò che il termine $\omega_{me}\Lambda_{mg}$ possa essere misurato sotto forma di una tensione fase-fase trifase trascinando il motore con un secondo attuatore ad esso collegato. Tale tensione va poi trasformata nell'equivalente grandezza monofase, per ricondursi allo schema di figura (5.6).
L'espressione che ne risulta è la seguente:

$$\Lambda_{mg} = \frac{V_{pp}(@\omega_m)}{2\sqrt{3}\omega_{me}} \tag{5.5}$$

I parametri dei due motori utilizzati sono riportati in tabella (5.3). Entrambi sono dei

Simbolo	Parametro	MUT	Carico	Unit
R_s	Resistenza di fase	0.56	0.342	$[\Omega]$
L_s	Induttanza di fase	1.6	0.32	$[mH]$
λ_{mg}	Flusso concatenato magnete permanente	0.0278	0.0082	$[V \cdot s]$
J	Inerzia del rotore	0.12	0.177	$[kgcm^2]$
I_N	Corrente nominale	3.5	4.4	$[A]$
Ω_N	Velocità nominale	5000	2800	$[rpm]$
T_N	Coppia nominale	0.268	0.22	$[Nm]$
p	Numero di coppie polari	2	5	–
N	Tacche encoder	512	250	$[ppr]$

Tabella 5.3. Parametri dei motori *Under Test* e *Load*.

Brushless DC motors e presentano delle caratteristiche simili.
Il primo, numero di serie *MB057GA240*, è prodotto dall'azienda *Speeder Motion*, mentre il secondo, *DMA0204024B101*, è fabbricato dalla *Hurst*. La differenza principale tra i due trasduttori è il numero di coppie polari con cui sono realizzati, ovvero il numero di volte in cui il flusso concatenato tra rotore e statore si ripete uguale all'interno dell'angolo meccanico. Il motore Hurst presenta più del doppio delle coppie polari rispetto all'altro, e ciò gli consente di essere realizzato con magneti più piccoli a parità di coppia prodotta, che in questo tipo di motori vale $\tau = \frac{3}{2}p\Lambda_{mg}$.
Il fatto che le grandezze elettriche si ripetano più volte all'interno dell'angolo giro meccanico, però, ne penalizza la precisione nel controllo. Infatti, dividendo il numero di gradi elettrici che il motore percorre ad ogni rotazione meccanica per il numero di tacche dell'encoder, nel primo caso si ottiene una risoluzione elettrica di $\frac{360°\times 2}{512} = 1.4°$, mentre nel secondo di $\frac{360°\times 5}{250} = 7.2°$, causando una serie di problematiche che verranno analizzate

nelle varie prove sperimentali.
La struttura dell'encoder è comunque la medesima nei due motori, entrambi dotati di trasduttori ottici di tipo incrementale. Il funzionamento di questi dispositivi è piuttosto semplice, essendo composti di un disco forato in rotazione che permette o meno, a seconda della posizione, il passaggio di infrarossi tra un emettitore e un ricevitore, generando in uscita un onda quadra alla frequenza di rotazione dell'albero. Conteggiando il numero di fronti, e conoscendo il numero di tacche dell'encoder, è possibile risalire all'incremento angolare di posizione.
Ciascun trasduttore, inoltre, presenta due uscite "in quadratura", ovvero due segnali, generati da altrettante coppie emettitore-ricevitore, sfasati di 90° tra loro, grazie alla diversa disposizione dei fori nel disco. Questa caratteristica permette di quadruplicare la risoluzione dei dispositivi, grazie alla lettura dei fronti di entrambi i segnali.
Ne risulta perciò una risoluzione elettrica di circa 0.35° per il primo encoder e di 1.8° per il secondo.

6

SOFTWARE

In questo capitolo si andranno a descrivere le soluzioni software adottate, con particolare riguardo per gli accorgimenti utilizzati per adattare l'algoritmo della rete neurale al microcontrollore.
In particolare, il software si compone di due parti: il programma che viene eseguito a livello hardware è realizzato in linguaggio *C*, mentre l'interfaccia è stata costruita con *Simulink*, in modo da rendere più agevole l'interazione con l'utente.
La conversione di tale interfaccia in un linguaggio comprensibile per il microcontrollore è attuata mediante il software *X2C*, che trasforma lo schema *Simulink* in una serie di librerie da includere nel progetto.

6.1 *Software Microcontrollore*

Per la programmazione del microcontrollore è stato utilizzato il software *MPLAB X*, un ambiente di sviluppo fornito da *Microchip* che integra gli strumenti per scrivere, compilare e scaricare nel micro qualsiasi programma scritto in linguaggio *C* o *C++*. Inoltre, permette di monitorare la quantità di memoria flash occupata dal codice e dai dati, oltre ad offrire la possibilità di effettuare il debug in tempo reale per visualizzare il valore assunto dalle variabili durante l'esecuzione.
Per quanto riguarda il codice, si è scelto di attuare la scansione temporale illustrata in figura (6.1).

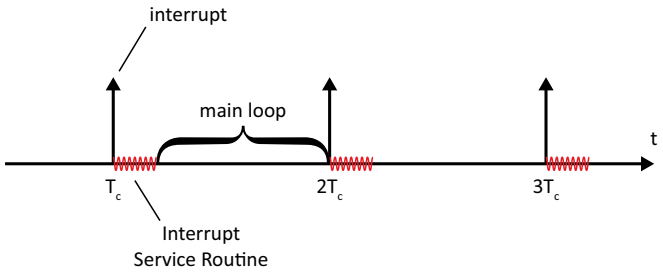


Figura 6.1. Suddivisione temporale del tempo di ciclo.

Come si nota, tutto ciò che riguarda il controllo del motore viene eseguito ad ogni $T_c = 125\,\mu s$, periodo scelto per la PWM, grazie all'interrupt dato dal timer descritto nel paragrafo (5.2). Quando tale interruzione viene generata, il micro esegue le funzioni riportate nell'algoritmo (2).
Come si nota, il processo inizia con l'acquisizione delle misure di corrente (2.2), convertite dall'ADC sincronizzato con la PWM, e delle misure di tensione, per poi passare alla misura

Algorithm 2 Control Loop

```
1: function InterruptTasks( )
2:     CurrentMeasurement( );
3:     PositionMeasurement( );
4:     SignalTransformation( );
5:     GetSwitchState( );
6:     X2CReadInports( );
7:     X2CUpdate( );
8:     X2CWriteOutports( );
9:     VoltageMeasurement( );
10:    TestTimerCan1( );
11: end function
```

di posizione fornita dal counter collegato all'encoder (2.3).
A questo punto si hanno tutti i dati necessari per passare da un sistema di riferimento in *abc* ad uno sincrono in *dq*, nel quale operano gli anelli di controllo del motore (2.4).
Si passa poi alla lettura dei pulsanti del banco (2.5), per verificare il segnale di accensione o spegnimento della PWM o un eventuale comando di reset.
La fase successiva include tutte le funzioni necessarie per interfacciarsi con i blocchi in *Simulink*. Infatti, vengono prima letti gli ingressi dello schema (2.6), ovvero le misure acquisite dal micro, per poi eseguire le funzioni dei vari blocchi (2.7) ed infine scriverne i valori risultanti nelle uscite programmate (2.8).
Le ultime due funzioni eseguite nell'interrupt sono la misura della tensione sul BUS in continua (2.9), sempre convertita tramite uno dei canali dell'ADC, e un check sulla comunicazione in seriale tra le due schede (2.10).
Tra una routine di interrupt e la successiva, il microcontrollore esegue le funzioni che sono incluse nel main loop, riportate nell'algoritmo (3).

Algorithm 3 Control Loop

```
1: while true do
2:     X2CCommunicate( );
3:     HandleCan1State( );
4:     RBF( );
5: end while
```

Tali funzioni presentano una caratteristica di particolare interesse per questa applicazione: oltre ad essere eseguite nel tempo di *idle* del microcontrollore, ogni volta che si attiva un segnale di interrupt il processore automaticamente salva il contesto attuale, ovvero variabili ed indirizzo dell'istruzione che stava eseguendo, per riprendere la sua esecuzione una volta terminata la funzione richiamata dall'interrupt stesso. Questo permette di inserire nel main loop anche funzioni della durata di molteplici T_c senza generare problemi nella sequenza di esecuzione del sistema.
La prima funzione include le istruzioni per comunicare con il computer tramite *USB*. In particolare, vengono letti i parametri settati nei vari blocchi sullo schema simulink

del terminale e, ogni volta che viene apportata qualche modifica, viene comunicata in seriale al micro. Inoltre viene trasmesso al PC un buffer che la scheda, ad ogni T_c, aggiorna con un nuovo valore. Questo buffer contiene il segnale che verrà mandato in output nel monitor del PC sull'oscilloscopio virtuale dell'interfaccia. Viene infine resettato un timer che, se raggiunge il suo valore massimo, interrompe ogni comunicazione e invia un segnale di errore di tipo *TimeOut*, il quale indica che è passato troppo tempo tra due comunicazioni successive. Il flag di mancata comunicazione viene attivato dopo circa sette secondi. É importante ricordare questo limite temporale in quanto, nonostante le istruzioni del main loop possano essere eseguite anche in molti T_c, se avviene il *TimeOut* della seriale sarà impossibile raccogliere informazioni utili dal micro una volta terminato il processo, in quanto la comunicazione si interrompe.
La seconda funzione, invece, gestisce la connessione tra le due schede di controllo dei motori. Come descritto in precedenza, tale comunicazione è implementata mediante un protocollo *CAN-Bus*, che permette una velocità di comunicazione fino a 1 $Mbit/s$. Nella linea transitano otto byte alla volta, più che sufficienti data la ridotta quantità di informazione scambiata tra le due schede.
In questo contesto è stata inserita la funzione inerente all'addestramento della rete, la cui struttura è riportata nell'algoritmo (4).

6.1.1 Posizionamento nodi della rete

Innanzitutto vengono posizionati i nodi della rete, salvandone le coordinate nel piano dq in una matrice. Dopo aver ottenuto il numero totale di nodi, viene calcolato il bias con l'equazione (4.6).
A questo punto la rete rimane in attesa finché non ottiene il permesso di eseguire. L'abilitazione viene fornita dalla funzione che genera i segnali di riferimento per il training, dopo che si è conclusa la fase di raccolta dati.

6.1.2 Data processing

Le operazioni successive riguardano i campioni raccolti, che vengono a loro volta ricampionati, nel caso sia necessario (si ricorda la scelta di limitare la dimensione dei dati a 200 unità per problematiche di occupazione della memoria RAM), e filtrati. In particolare, si utilizza un passa basso del primo ordine con frequenza di taglio di 1 kHz, discretizzato con il metodo di Eulero *in avanti*.
La struttura utilizzata è del tipo:

$$y(k) = \alpha x(k) + (1 - \alpha) y(k - 1) \tag{6.1}$$

in cui il termine moltiplicativo risultante dalla discretizzazione vale $\alpha = \frac{\tau}{T_c + \tau}$, con τ costante di tempo del filtro ($\frac{1}{2\pi FreqTaglio}$).
Una volta filtrati i campioni possono essere derivati. Per semplicità, si è scelto di utilizzare una derivata discreta ad un passo, del tipo:

$$y(k) = \frac{x(k) - x(k - 1)}{T_s} \tag{6.2}$$

Algorithm 4 RBF Training in C

1: Posizionamento dei nodi della rete e calcolo dei parametri
2: **if** *enable* **then**
3: Downsampling dei dati e successivo filtraggio passa basso
4: Calcolo delle derivate
5: **for** *tutti gli M dati* **do**
6: **for** *tutti i K nodi* **do**
7: Calcolo distanza euclidea (4.7)
8: Calcolo uscita gaussiana (4.8)
9: Calcolo elemento [m,k] della Jacobiana $\frac{\partial \varepsilon_d^m}{\partial w_d^k}$ (4.15)
10: Copia elemento [m+M,k+K] della Jacobiana $\frac{\partial \varepsilon_q^m}{\partial w_q^k} = \frac{\partial \varepsilon_d^m}{\partial w_d^k}$ (4.16)
11: Calcolo elemento [m,k+K] della Jacobiana $\frac{\partial \varepsilon_d^m}{\partial w_q^k}$ (4.13)
12: Copia elemento [m+M,k] della Jacobiana $\frac{\partial \varepsilon_q^m}{\partial w_d^k} = -\frac{\partial \varepsilon_q^m}{\partial w_d^k}$ (4.14)
13: **end for**
14: **end for**
15: Inversione matrice $[\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}]$
16: Inizializzazione vettore dei pesi $\mathbf{w}^0 = NULL$
17: **while** numero cicli<100 **do**
18: Calcolo dei flussi stimati (4.9)
19: Calcolo errori di stima (4.5)
20: Computazione la funzione di costo (4.10)
21: **if** $E(\mathbf{w}_d, \mathbf{w}_q) < E_{th}$ **then**
22: *Break*
23: **end if**
24: Esecuzione algoritmo di Levenberg-Marquardt per trovare il nuovo vettore di pesi (4.11)
25: numero cicli++
26: **end while**
27: Scrittura pesi risultanti nel buffer del monitor seriale
28: **end if**

in cui T_s rappresenta il tempo tra due campioni dopo aver effettuato il downsampling, calcolato come $T_c * SampleFactor$.

6.1.3 Calcolo Jacobiana

Il passo successivo riguarda il calcolo della matrice Jacobiana, che si suddivide nelle tre fasi elencate nell'algoritmo: calcolo della distanza del campione $\mathbf{i}_{dq}^{k}$ da ciascun nodo, "filtraggio" per la gaussiana ed, infine, calcolo delle quattro componenti della matrice Jacobiana, una per blocco.
Ciò che risulta particolarmente oneroso dal punto di vista computazionale, però, è l'inversione della matrice $[\mathbf{J}^T\mathbf{J}+\mu\mathbf{I}]$, principalmente a causa della sua estensione $(2M\times 2K)$. L'algoritmo di Levenberg-Marquardt prevedrebbe di partire con un coefficiente μ molto piccolo per poi incrementarlo gradualmente ad ogni passo del training finché la matrice non risulta invertibile. Per eseguire l'inversione una sola volta e rendere l'addestramento più veloce, si è invece scelto di renderlo costante. Nel capitolo sui risultati sperimentali verranno eseguite alcune comparazioni dei risultati al variare di μ (7.6).
Per quanto riguarda l'inversione vera e propria, in un primo momento si è optato per un algoritmo basato sul metodo dei cofattori. Considerando di avere una matrice di partenza pari a:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \tag{6.3}$$

applicando tale metodo, si ottiene:

$$\mathbf{A}^{-1} = \frac{1}{det(\mathbf{A})} \begin{bmatrix} Cof(a_{11}) & Cof(a_{12}) & \cdots & Cof(a_{1n}) \\ Cof(a_{21}) & Cof(a_{22}) & \cdots & Cof(a_{2n}) \\ \vdots & \vdots & \ddots & \vdots \\ Cof(a_{n1}) & Cof(a_{n2}) & \cdots & Cof(a_{nn}) \end{bmatrix}^T \tag{6.4}$$

dove $Cof(a_{ij}) = (-1)^{i+j} \cdot det(\mathbf{A}_{ij})$ indica il cofattore, o complemento algebrico, relativo all'elemento a_{ij}, con $\mathbf{A}_{ij}$ pari alla matrice di partenza privata della riga i-esima e della colonna j-esima. Questo metodo è semplice da trasformare in un algoritmo ricorsivo, ma il calcolo risulta tutt'altro che rapido, soprattutto per matrici di estensione elevata. Per l'inversione della matrice in esame, con 400 righe e 72 colonne, sono state impiegate due ore e sette minuti.
Si è quindi scelto di cambiare approccio, optando per un algoritmo simile a quello utilizzato da MATLAB, che sfrutta la decomposizione di Doolittle, o *decomposizione LU*. In particolare, per una generica matrice quadrata, vale $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$, vale:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ a'_{21} & a'_{22} & \cdots & a'_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a'_{n1} & a'_{n2} & \cdots & a'_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{6.5}$$

Se poniamo la matrice $\mathbf{A}^{-1}$ come una matrice di incognite e ne consideriamo una colonna alla volta, ne risulta un sistema lineare nella forma $\mathbf{A}x = b$. Ad esempio, per la prima colonna, vale:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} a'_{11} \\ a'_{21} \\ \vdots \\ a'_{n1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.6}$$

Esistono molteplici modi per risolvere un sistema lineare. Una possibile strategia, sufficientemente semplice da implementare in un algoritmo in C, consiste nel trasformare la matrice di partenza in una matrice a scala e risolvere il sistema che si genera procedendo a ritroso, dall'ultima riga (con una sola incognita), alla prima, sostituendo progressivamente quanto trovato nelle varie equazioni, in modo da ricondursi sempre ad un problema con una variabile.
L'algoritmo implementato è il (5).

Algorithm 5 Linear system

1: Unisci la matrice **A** alla colonna dei termini noti
2: *Ordina le righe della matrice dall'elemento maggiore al minore, procedendo a scala:*
3: **for** Ogni elemento [i,i] della diagonale della matrice **do**
4: **for** Ogni elemento j nella colonna al di sotto di i **do**
5: **if** L'elemento j è più grande di [i,i] **then**
6: Scambia le righe i e j
7: **end if**
8: **end for**
9: **end for**
10: *Eliminazione Gaussiana:*
11: **for** Ogni elemento [i,i] della diagonale della matrice **do**
12: **for** Ogni riga j al di sotto di i **do**
13: Sottrai alla riga j la riga i moltiplicata per il coefficiente $f = A[j][i]/A[i][i]$, in modo da ottenere zero nei primi i elementi di j
14: **end for**
15: **end for**
16: *Sostituzione all'indietro per trovare le incognite:*
17: **for** Ogni elemento [i,i] della diagonale della matrice a scala, dall'ultimo al primo **do**
18: risultato[i] = termine noto del sistema
19: **for** Ogni elemento j=i+1 della riga i **do**
20: Sottrai al risultato tutte le j incognite trovate finora, moltiplicate per i relativi coefficienti
21: **end for**
22: Dividi il valore per il coefficiente dell'incognita [i,i] cercata
23: **end for**

In un primo momento si scambiano le righe della matrice $[\mathbf{A}|b]$ in modo da avere sempre l'elemento maggiore pivot nella diagonale (*pivot*). Successivamente si va a sottrarre ad

ogni riga quella del pivot, moltiplicata per il primo elemento diviso per il pivot, in modo da ottenere la scala di zeri. Infine si trovano le incognite, dall'ultima alla prima, sottraendo ai termini noti le incognite già trovate, con i relativi coefficienti moltiplicativi, e dividendo per il coefficiente dell'elemento cercato.
Con questo algoritmo, il medesimo problema, che prima impiegava oltre due ore, viene risolto in circa due secondi.

6.1.4 Levenberg-Marquardt

A questo punto si hanno tutti gli elementi per applicare l'algoritmo di Levenberg-Marquardt. Inizialmente si suppone nullo il vettore dei pesi. Di conseguenza, nel primo ciclo, si otterranno dei flussi stimati nulli (4.9). Il vettore degli errori, quindi, si comporrà dei soli termini $\mathbf{u}_{dq} - R * \mathbf{i}_{dq}$. Applicando poi l'equazione (4.11) si calcolano i nuovi pesi, considerati già attendibili dopo un solo passo. L'algoritmo comunque viene iterato finché il costo non smette di decrescere, con un limite massimo a 100 cicli per evitare loop infiniti.
Infine, il vettore di pesi "ottimo" viene inserito nel buffer, che verrà inviato sul monitor del PC, per verificarne la correttezza. Tale vettore, inoltre, costituisce la base per eseguire la rete neurale in real time nel microprocessore ed estrarre il valore esatto del flusso per ogni coppia di correnti. Questa strategia verrà descritta nel paragrafo (9.5).

6.2 *Interfaccia Simulink*

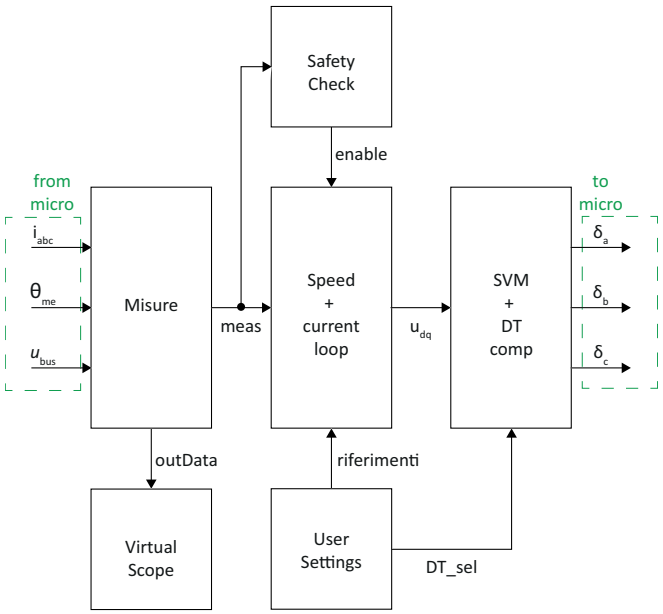


Figura 6.2. Schema di principio dell'interfaccia *Simulink*.

Il principio di funzionamento dello schema *Simulink* che funge da interfaccia è riportato in figura (6.2).
Come descritto nel paragrafo (6.1), tale schema viene eseguito come una routine a sé stante all'interno del programma del microcontrollore, con la quale avviene una sorta di

"comunicazione" asincrona. Sono infatti presenti dei buffer nei quali il micro inserisce le misure raccolte, oltre a delle variabili dove i blocchi simulink possono salvare i risultati computati. Il tutto viene aggiornato ogni T_c, sfruttando l'interrupt del timer della PWM. Nella prima parte dello schema vengono effettuate una serie di elaborazioni matematiche sulle grandezze acquisite dal micro. Innanzitutto viene rimosso l'eventuale offset di misura dalle correnti, acquisito prima di avviare la PWM. Infatti, all'accensione del micro, le correnti misurate dovrebbero essere nulle. Se ciò non accade, l'eventuale differenza dallo zero viene imputata ad un offset e sottratta durante l'intero funzionamento. Nel blocco di misura, inoltre, viene calcolata la velocità angolare del rotore grazie ad una derivata filtrata (con banda passante di 50 Hz) della misura di posizione.
É poi presente una sezione dedita a controllare che il sistema non superi i parametri nominali del motore, corrente e velocità. Se ciò accade, viene generato un allarme che spegne la *PWM*, onde evitare danneggiamenti ai dispositivi utilizzati.
Nella sezione di controllo, invece, vengono calcolate le uscite dei PI, prima dell'anello di velocità, se attivo, poi dell'anello di corrente. Per quanto riguarda quest'ultimo, si è scelto di utilizzare una taratura basata sul metodo di *Bode*, grazie alla conoscenza, anche approssimata, dei parametri dell'attuatore.
Si ricorda che, per applicarlo, è necessario imporre pulsazione di attraversamento, ω_{gc} , e margine di fase, φ_m, della funzione ad anello aperto del sistema, $L(s)$, generando le condizioni:

$$\begin{cases} |L(j\omega_{gc})| = 1 \\ \underline{/L(j\omega_{gc})} = -\pi + \varphi_m \end{cases} \tag{6.7}$$

Tale funzione è definita come:

$$L(s) = R(s)D(s)M(s). \tag{6.8}$$

dove $M(s)$ è la funzione di trasferimento del circuito equivalente del motore, supposto lineare:

$$M(s) = \frac{1}{R + sL}, \tag{6.9}$$

$D(s)$ è quella dell'inverter, considerato un ritardo approssimato ad un primo ordine:

$$D(s) \cong \frac{1}{1 + s\tau_D}, \tag{6.10}$$

infine, $R(s)$ è quella del regolatore PI:

$$R(s) = k_p + \frac{k_i}{s} \tag{6.11}$$

In particolare, si è scelto di porre $\omega_{gc} = 2\pi \cdot 200\, rad/s$ e $\varphi_m = 80°$, ottenendo $k_p = 1.93\, \frac{V}{A}$ e $k_i = 1041.95\, \frac{V}{A \cdot s}$. Quest'ultimo andrà poi moltiplicato per T_c quando si andrà a discretizzare il controllo con il metodo di Eulero *in avanti*.
Il riferimento di tensione generato sarà l'ingresso della sezione dedicata all'inverter, nella quale vengono calcolati i *duty cycle* che serviranno per pilotare gli switch. Prima di fare ciò, come descritto nel paragrafo (5.3), ai riferimenti di tensione viene sommato il contributo che si andrà a perdere a causa dei tempi morti.

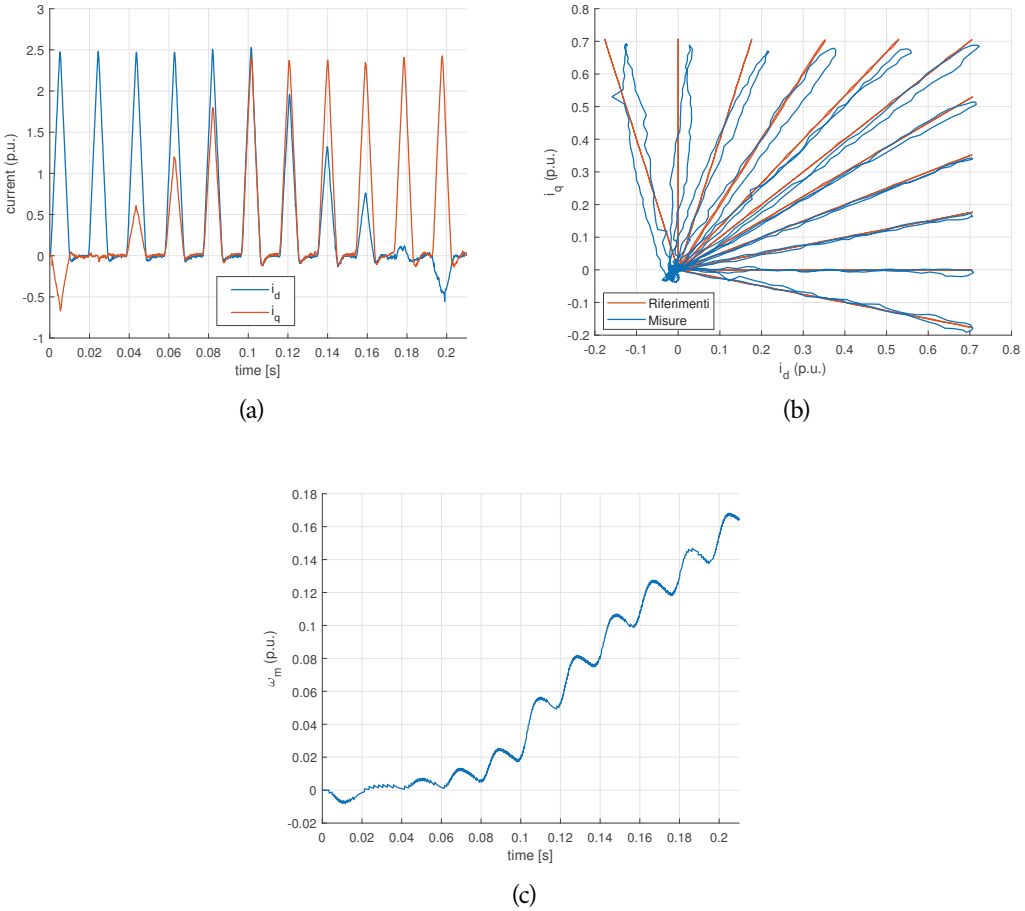
Infine, è presenta una serie di blocchi che fungono da interfaccia con l'utente. Questa sezione comprende un selettore per scegliere se controllare il motore in corrente o in velocità, due blocchetti di input che permettono di impostare il riferimento di velocità o di corrente, un parametro libero che permette di scegliere il metodo di compensazione dei tempi morti e un enable per avviare l'algoritmo di addestramento della rete.
A ciò si aggiunge un interfaccia grafica che simula un oscilloscopio virtuale, nel quale è possibile visualizzare gli andamenti dei vari segnali che arrivano dal micro.
Come in ogni oscilloscopio reale, è possibile avviare o fermare l'acquisizione, scegliere il tempo di campionamento, con conseguente aumento o diminuzione della finestra temporale osservata, impostare una modalità e un livello di trigger, con relativo ritardo, e scegliere quali tracce visualizzare.

7

RISULTATI SPERIMENTALI ADDESTRAMENTO SPMSM UNDER TEST

In questo capitolo si andranno a riportare e a descrivere i risultati ottenuti dall'addestramento della rete implementata sul microcontrollore, utilizzando come attuatore in analisi il "Motor Under Test".
In particolare, si andranno ad analizzare gli effetti delle scelte effettuate in termini di parametri della rete, numero di nodi, taratura dei PI e compensazione dei tempi morti.

7.1 *Riferimento di corrente*



(a) (b)

(c)

Figura 7.1. Riferimenti di corrente imposti al motore: (a) andamenti temporali, (b) nel piano dq, (c) velocità angolare meccanica.

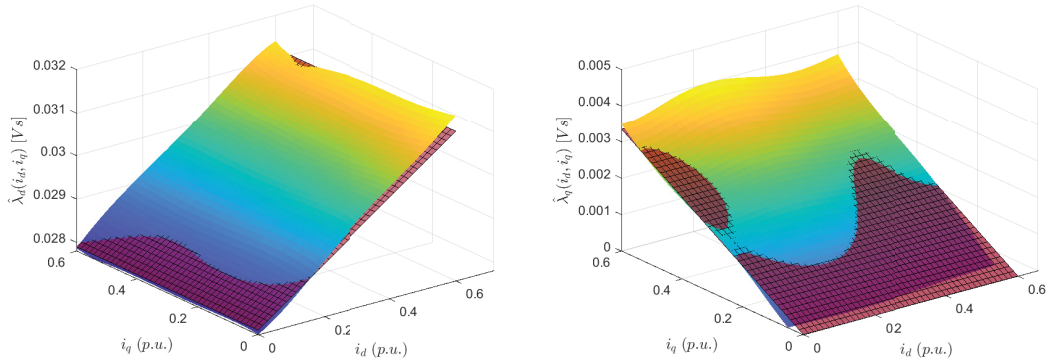
L'algoritmo descritto può operare in qualsiasi condizione di lavoro del motore, compresi i transitori di corrente e velocità. L'unico limite è l'area del piano *dq* che viene addestrata. Infatti, se uno o più neuroni non vengono interessati dalle correnti $\mathbf{i}_{dq}$, il peso a loro associato rimarrà prossimo allo zero. Questa situazione non è affatto da escludere in questo tipo di reti, il cui carattere è fortemente localizzato a causa della funzione di attivazione gaussiana utilizzata. Al fine di ottenere una mappa di flusso il più omogenea possibile, si è scelto di iniettare una serie di rampe di corrente che interessassero l'intero primo quadrante.
I riferimenti sono riportati in figura (7.1). Nella (7.1.a) sono riportati gli andamenti temporali: si può notare come la durata delle rampe sia limitata, $10\,ms$, così come l'ampiezza delle stesse, $2.5\,A$. Il primo parametro è stato scelto in modo da mantenere una velocità ridotta, in quanto, all'aumentare di quest'ultima, la tensione di riferimento tende ad allontanarsi da quella realmente applicata al motore, provocando un errore che si ripercuote sul training. Oltre a ciò, l'anello di controllo di velocità è scollegato, quindi tale grandezza evolve in modo incontrollato. Limitando la durata delle rampe si evita di raggiungere la velocità nominale, che porterebbe all'interruzione della prova. L'andamento è riportato nella (7.1.c). Come si nota, la massima velocità che si raggiunge è di circa $180\,rad_{ele}/s$, che equivalgono a $860\,rpm$ meccanici (due coppie polari), quasi $\frac{1}{5}$ di quella nominale.
L'ampiezza delle rampe, invece, è limitata dalla potenza che è in grado di fornire l'alimentatore, $16\,V \cdot 5\,A$. La tensione sulle fasi, infatti, aumenta all'aumentare della corrente sia per la maggior componente resistiva, che per le elevate velocità raggiunte grazie alla maggior coppia generata.
Il fatto che l'attuatore in esame acceleri in modo così repentino è giustificato dall'inerzia ridotta che caratterizza il sistema. Infatti, la coppia motore-carico raggiunge appena gli $0.297\,kgcm^2$.
Nella figura (7.1.b) si può invece osservare l'area coperta dai riferimenti di corrente (in rosso), sovrapposti alla quantità realmente misurata sugli avvolgimenti del motore. Come si nota, viene interessato dall'addestramento l'intero primo quadrante, con le due grandezze raffigurate che differiscono a causa dei tempi di risposta dell'anello di controllo. Le rampe che sforano nel secondo e nel quarto quadrante servono per fornire dati attendibili ai neuroni alle estremità dell'area di training, altrimenti marginalmente interessati dal riferimento e, di conseguenza, addestrati in modo approssimativo.

7.2 *Flusso Magnetico Concatenato Stimato*

L'output dell'addestramento della rete è il set di pesi $\mathbf{w}_{dq}$. Per ricavarne effettivamente il flusso concatenato $\hat{\lambda}_{dq}$, è necessario fornire in ingresso alla rete una coppia di correnti i_d, i_q, individuare i nodi che si attivano attraverso il calcolo della distanza di $\mathbf{i}_{dq}$ da ciascuno di essi ed, infine, ricavare il valore flusso, seguendo quanto riportato nell'algoritmo (6).
Come si descriverà più dettagliatamente nel paragrafo (9.5), questo processo è abbastanza dispendioso dal punto di vista computazionale, perciò non è immediato da eseguire in real time.
Il flusso risultante è riportato in figura (7.2). Nel grafico di sinistra si può osservare il flusso stimato $\hat{\lambda}_d$, comprensivo del magnete permanente. La correttezza della stima si può verificare confrontando quanto ottenuto con il flusso "teorico" costruito con i parametri nominali del motore, ovvero $L_d = L_q = 1.595\ [mH]$ e $\Lambda_{mg} = 0.0278\ [V\,s]$,

Algorithm 6 Esecuzione RBF

1: **for** tutti i K nodi **do**
2: Calcolo distanza euclidea (4.7)
3: Calcolo uscita gaussiana (4.8)
4: **end for**
5: Calcolo dei flussi stimati (4.9)



(a) (b)

Figura 7.2. Flussi magnetici concatenati su un sPMSM stimati dalla rete RBF.

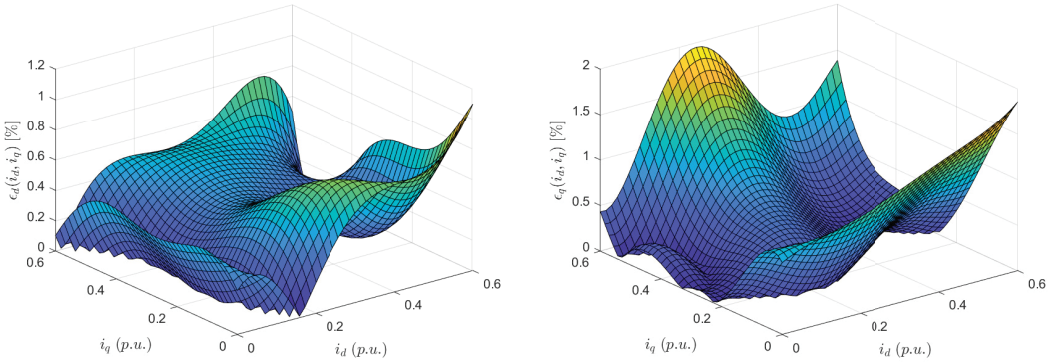
supponendo il motore lineare come da (5.2). La mappa che ne risulta è riportata in rosso nel grafico. Si nota come, sull'asse *d*, a corrente nulla, il flusso concatenato sia dovuto alla sola presenza del magnete, mentre, all'aumentare della corrente, cresca con la pendenza dettata dall'induttanza differenziale del motore. É evidente come la stima quasi coincida con il risultato teorico, differendo per un fattore di circa $10^{-4}[V\ s]$.
Quanto ottenuto rispetta le attese dato che il motore sotto test è un *sPMSM*, con induttanze ridotte e saturazione quasi assente. Questa struttura porta inoltre ad avere una cross-induttanza quasi nulla, che si traduce in un influenza reciproca dei due assi praticamente assente. Infatti, la corrente di asse *q* non induce variazioni nel flusso di asse *d* e viceversa.
In figura (7.3) sono riportati gli errori percentuali commessi sul flusso, calcolati come segue:

$$\epsilon_d = \frac{\hat{\lambda}_d - \lambda_{d,th}}{\lambda_{max}} \cdot 100 \qquad \epsilon_q = \frac{\hat{\lambda}_q - \lambda_{q,th}}{\lambda_{max}} \cdot 100 \tag{7.1}$$

con $\lambda_{max} = 0.0335\ [Vs]$, pari al massimo flusso teorico.
Si può notare come nell'asse *d* l'errore sia inferiore al punto percentuale, mentre nell'asse *q* arrivi a sfiorare il 2%. In entrambi gli assi, l'errore residuo potrebbe essere ulteriormente ridotto con una misura sperimentale delle tensioni, non prevista nella scheda utilizzata.

7.3 *Compensazione dei tempi morti*

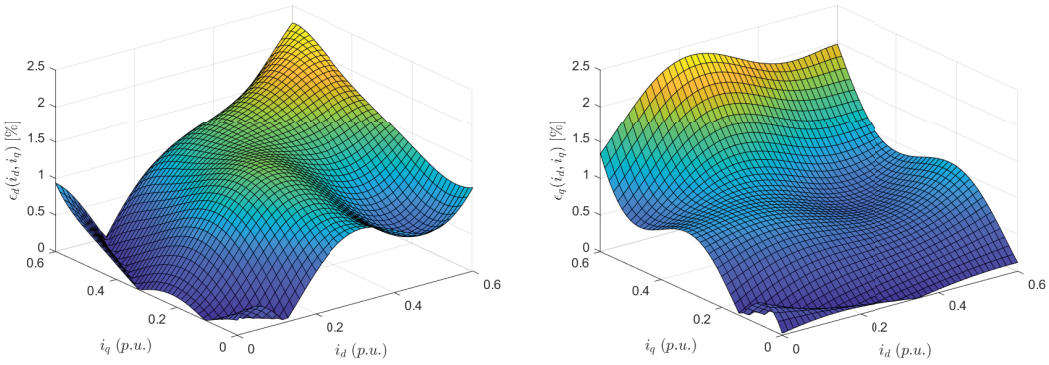
Come descritto nel paragrafo (5.3), per un controllo più preciso del motore è consigliabile compensare la tensione che viene persa a causa dei tempi morti nel comando degli switch.



(a) (b)

Figura 7.3. Errore percentuale commesso sulla stima dei flussi magnetici concatenati.

In questo caso, tale compensazione assume una rilevanza ancora maggiore, in quanto la tensione utilizzata per l'addestramento non è misurata ma è il riferimento in uscita dai PI di corrente. Se una parte della tensione letta è in realtà la quota aggiuntiva che i controllori devono inserire perché verrà decurtata dai tempi morti, le misure fornite all'algoritmo di Levenberg-Marquardt non sono più coerenti tra loro e il modello presenterà un errore più marcato, come si evince in figura (7.4).



(a) (b)

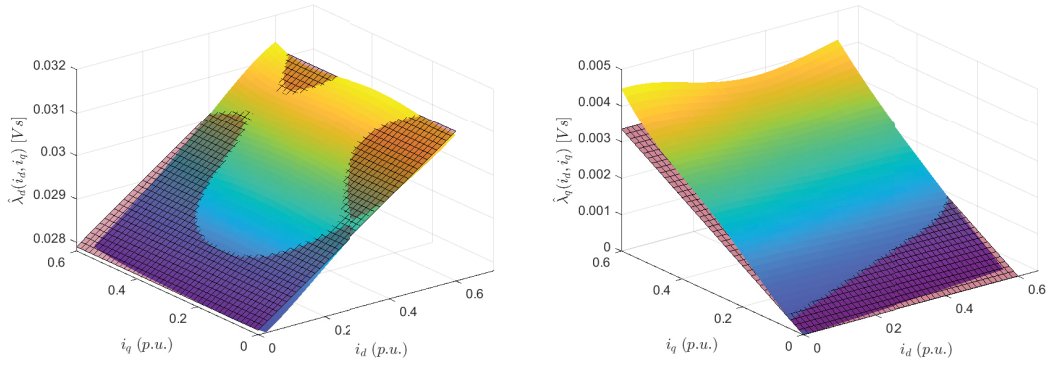
Figura 7.4. Errore percentuale commesso sul flusso concatenato senza compensare i tempi morti.

L'errore sul flusso concatenato di asse *d* è più che raddoppiato, mentre sull'asse *q* è aumentato quasi del 30%.
Ovviamente l'algoritmo per la compensazione non è privo di impatto sulle prestazioni del microcontrollore. Come descritto nel paragrafo (5.3), per la sua attuazione è richiesta la lettura di tre differenti lookup table indicizzate con il valore delle correnti di fase, che richiedono un tempo di esecuzione che aumenta all'aumentare della dimensione, e quindi della risoluzione, richiesta.
Nella soluzione implementata, sono state utilizzate *LUT* a 65 valori, portando ad un

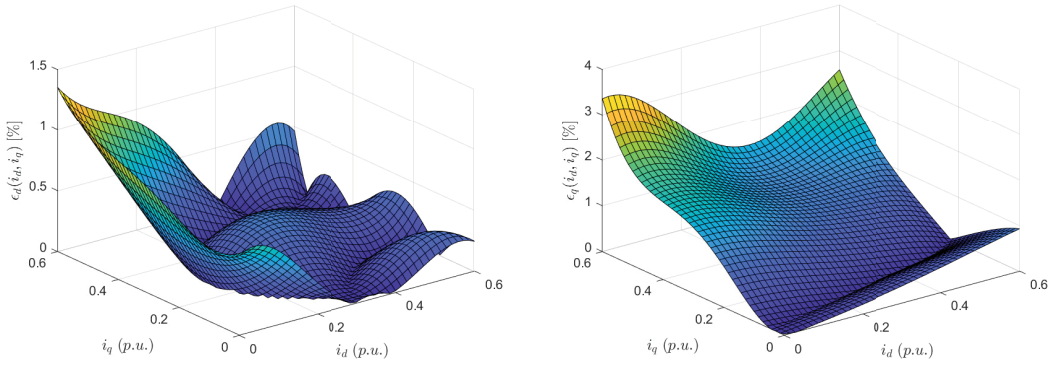
incremento del tempo di esecuzione di circa 21 μs, quasi il 20% del periodo di ciclo (125 μs). Questo intervallo temporale è stato stimato rilevando il valore del conteggio raggiunto dal timer della PWM una volta terminata l'esecuzione delle varie funzioni incluse nell'algoritmo (2). Inserendo nel loop di controllo la compensazione dei dead-time si è rilevato un incremento da 4300 a 6000 impulsi di clock (ricordando che il valore raggiunto alla fine del periodo è pari a 9375 unità).

7.4 *Numero di nodi*

Il numero di nodi, invece, ha un impatto diretto sulla risoluzione della mappa. Infatti, un set eccessivamente ridotto non è in grado di replicare con precisione le variazioni di pendenza tra un nodo e l'altro. Per tale motivo, se la funzione da approssimare è di grado elevato, la rete dovrà contenere un maggior numero di neuroni per replicarla con accuratezza. In questo caso, la mappa di flusso si può considerare lineare, almeno nell'area considerata, perciò il risultato è comunque accettabile, come raffigurato in figura (7.5.a) .



(a)



(b)

Figura 7.5. Flusso stimato (a) ed errore percentuale commesso (b) diminuendo il numero di nodi a quattro per quadrante.

Quanto illustrato è stato ottenuto diminuendo il numero di nodi da cinque a quattro per quadrante, asse compreso, perciò la risoluzione è stata ridotta da $\frac{I_N}{4}$ a $\frac{I_N}{3}$. Anche se l'andamento stimato sembra corretto, analizzando l'errore si osserva un valore quasi

raddoppiato, soprattutto sull'asse q, come raffigurato nel grafico (7.5.b).
Il vantaggio che si ottiene, però, si misura in termini di tempo di addestramento e di esecuzione della rete stessa. Come riportato in figura (7.6), l'andamento che si ottiene all'aumentare del numero di nodi non è lineare ma quadratico, con un tempo di addestramento quasi decuplicato nel passare dai 36 nodi utilizzati finora a 64 (da 6 a 8 nodi per asse). Questo è dovuto all'aumento della dimensione del problema, che si ripercuote in tutte le fasi del training, dal calcolo della distanza di $\mathbf{i}_{dq}$ da ciascun nodo, all'inversione della Jacobiana che passa da 400×72 a 400×128 elementi, al calcolo del vettore dei pesi $\mathbf{w}_{dq}$.

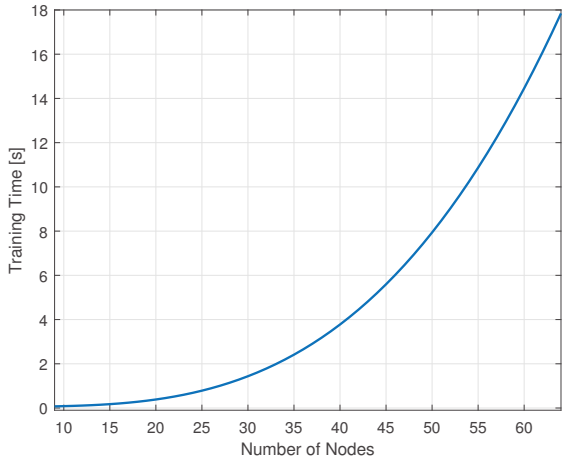
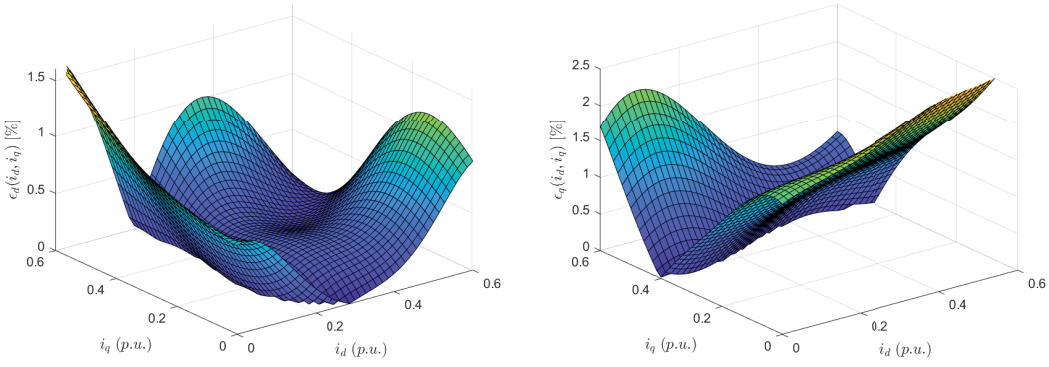


Figura 7.6. Tempo di addestramento della rete al variare del numero di nodi.

7.5 *Taratura dei PI*



(a) (b)

Figura 7.7. Errore percentuale commesso sul flusso concatenato con una taratura non ottimale dei PI.

Come descritto in precedenza, il metodo utilizzato per l'addestramento della rete non dipende dalle condizioni operative del motore. Ciò che si vuole dimostrare ora è che la taratura dei PI non influenza l'addestramento della rete. Per fare ciò, ne sono stati modificati i parametri, cercando di diminuire la componente proporzionale per evitare di ridurre eccessivamente il margine di fase. I nuovi parametri scelti sono $k_p = 1.0\,\frac{V}{A}$ e $k_i = 850\,\frac{V}{A \cdot s}$, con una riduzione, rispettivamente, di quasi il 50% e del 22% rispetto a quanto calcolato in sezione (6.2).
L'errore percentuale ottenuto sul flusso stimato, calcolato sempre con le equazioni (7.1), è riportato in figura (7.7). Come si nota, gli errori commessi, e, di conseguenza, gli andamenti dei flussi concatenati, sono molto simili a quelli ottenuti con la taratura corretta (immagine 7.3), attestandosi sull'unità percentuale sull'asse *d* e poco sopra al 2% sull'asse *q*.
Ciò che varia è il modo in cui il sistema insegue le rampe di riferimento. Come si evince in figura (7.8), vi sarà una differenza maggiore tra le due grandezze, modificando leggermente l'area di training (rendendola, oltretutto, più ampia), senza influenzare l'addestramento della rete.

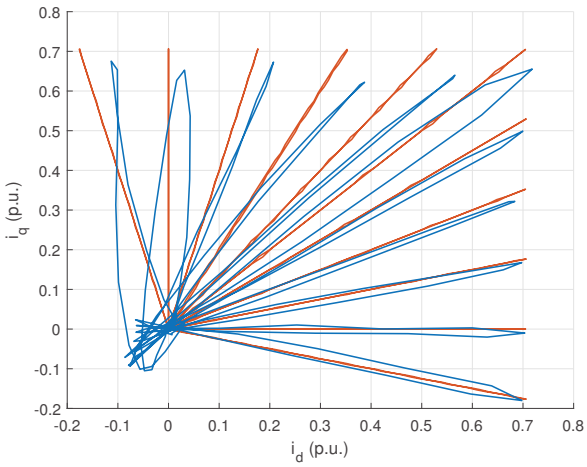
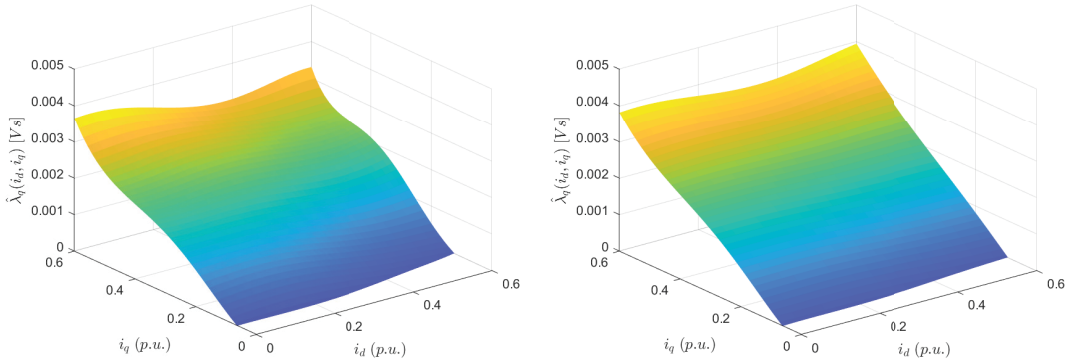


Figura 7.8. Area coperta dall'addestramento con una taratura dei PI non ideale.

7.6 *Variazione del parametro μ*

É stato infine analizzato il comportamento al variare del parametro μ nell'espressione $[\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}]$. Come descritto nel paragrafo (2.3), un valore ridotto porta a problemi di invertibilità, un valore elevato degrada l'algoritmo ad uno *steepest descend*, perdendo i vantaggi sulla convergenza portati dal metodo di Newton.
Nella rete in esame si è scelto di porre tale parametro ad un valore costante e pari all'unità, compromesso tra una buona approssimazione dei flussi concatenati e il tempo necessario per la convergenza verso una soluzione finale, che aumenta esponenzialmente se la matrice è prossima alla singolarità (non invertibile).
In figura (7.9) sono riportati i flussi di asse *q* stimati ponendo prima μ pari a 0.01, (7.9.a), e poi a 10, (7.9.b). Si nota come nel primo caso la stima sia meno lineare, con molte increspature che ne alterano l'andamento, mentre nel grafico ottenuto con $\mu = 10$ la



(a) (b)

Figura 7.9. Flusso stimato di asse *q al variare del parametro* μ, *(a)* $\mu = 0.01$, *(b)* $\mu = 10$.

superficie risulti "filtrata", con molte meno oscillazioni. Si può quindi concludere che tale parametro vada a modificare gli autovalori della matrice da invertire rendendola più o meno sensibile alle variazioni nella superficie da replicare, agendo come un filtro in due dimensioni sul risultato finale.

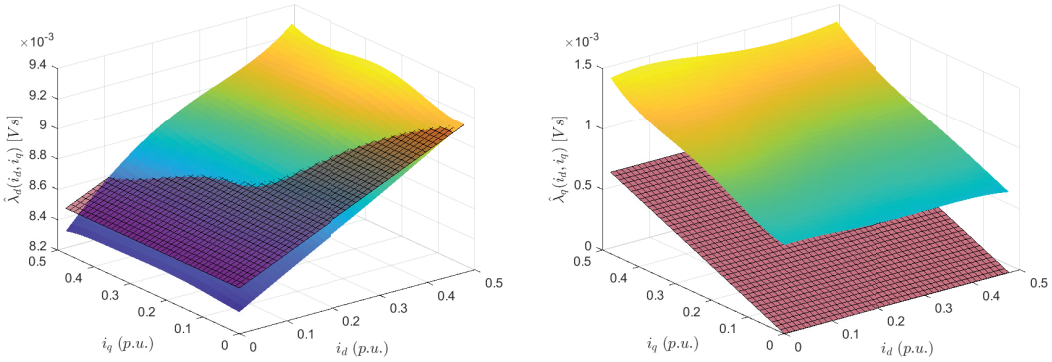
8

RISULTATI SPERIMENTALI ADDESTRAMENTO SPMSM LOAD E SYNR

Per verificare il corretto funzionamento della rete, si è deciso di ripeterne l'addestramento anche sul secondo SPMSM installato sul banco, oltre che su un motore sincrono a riluttanza. Lo scopo delle due prove è quello di validare l'ipotesi iniziale secondo cui, con il modello adottato, la rete è in grado di operare con qualsiasi motore sincrono, a magneti permanenti o SynR.

8.1 *Load*

In questa prima prova, si desiderava testare il comportamento dell'algoritmo in presenza di un sistema di misura meno accurato. Si ricorda infatti che la risoluzione sulla posizione elettrica di questo azionamento è di 1.8° per ogni impulso, come descritto nel paragrafo (5.4), trattandosi di un motore a cinque coppie polari dotato di un encoder incrementale con 250 tacche per ogni giro.
Per il solo controllo, in velocità o in corrente, tale risoluzione è più che sufficiente, considerando inoltre che, grazie alle due uscite in quadratura, il numero di impulsi per giro può considerarsi quadruplicato. Per questa applicazione, però, anche un minimo disallineamento può causare dei problemi, come verrà analizzato in questo capitolo.



(a) (b)

Figura 8.1. Flussi magnetici concatenati sul sPMSM *Load* stimati dalla rete RBF.

I flussi concatenati stimati dalla rete sono riportati in figura (8.1). L'asse *d* risulta coerente con quanto atteso, con una stima del flusso del magnete permanente che si discosta di meno del 2% rispetto a quello reale. Per quanto riguarda l'asse *q*, invece, si nota immediatamente come risulti corretto dal punto di vista dell'induttanza differenziale (perfettamente

parallelo a quello teorico), ma presenti un offset non trascurabile, che lo porta ad un valore massimo quasi triplicato rispetto a quello atteso.
Dato che l'addestramento della rete si basa sulle misure sperimentali che gli vengono fornite, si è scelto di iniziare a ricercare la sorgente del problema con l'analisi di tali grandezze. Innanzitutto le correnti sono misurate direttamente, e l'eventuale offset viene compensato con la strategia illustrata nel paragrafo (6.2), quindi si è escluso a priori che possano influire negativamente sul risultato.
Per verificare la correttezza della velocità angolare, è stato utilizzato un misuratore ottico esterno che, grazie a degli appositi adesivi riflettenti incollati sul rotore, ha permesso di confrontare il valore rilevato con quello derivato dalla posizione letta sull'encoder. Dato che i due valori coincidevano, è stato escluso anche un errore su ω_{me}.
L'ultima grandezza da analizzare è la tensione, ricordando che u_d e u_q non vengono misurate ma se ne utilizzano i riferimenti generati dai PI di corrente.

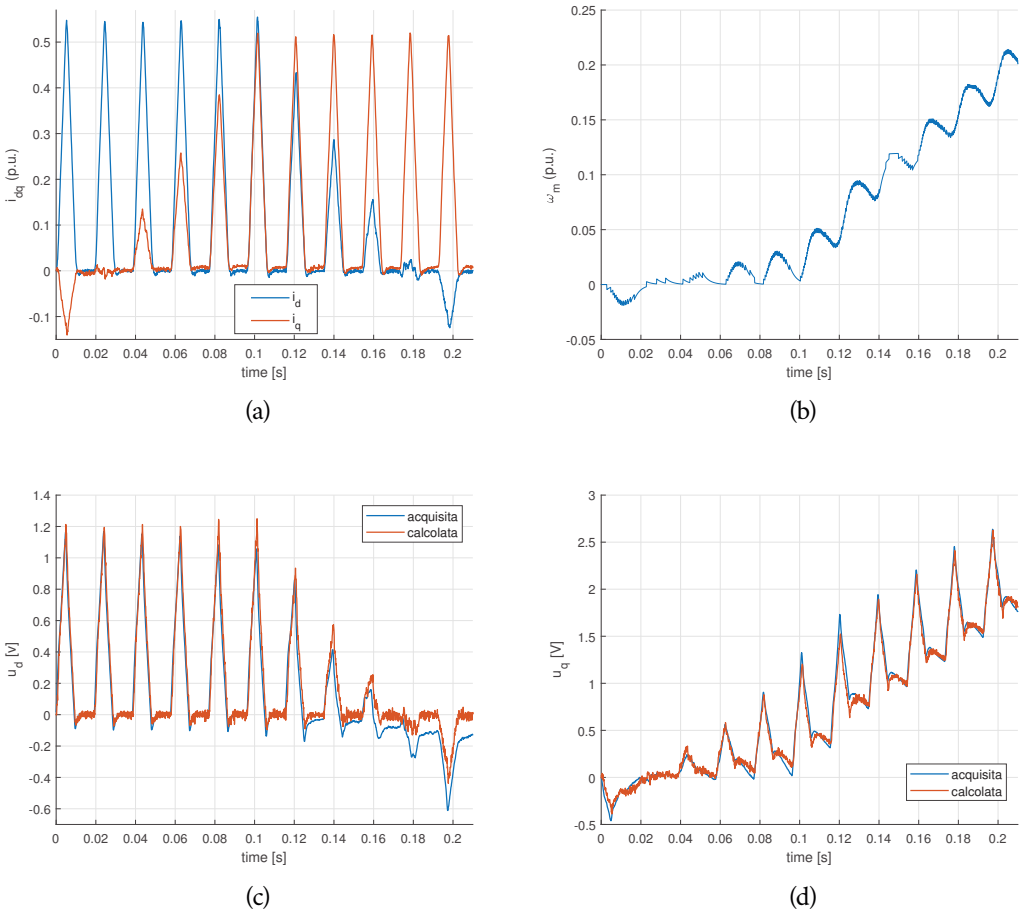


Figura 8.2. Andamenti temporali delle principali grandezze nel motore *Load*, (a) i_{dq}, (b) ω_m, (c) u_d, (d) u_q.

In figura (8.2) sono riportate le principali grandezze che caratterizzano il motore *Load*. In particolare, è utile focalizzarsi sul confronto tra le tensioni acquisite (riferimenti) e le tensioni calcolate con il modello lineare (5.2), conoscendo i vari parametri del motore. Come si nota, la tensione di asse q (5.2.d), caratterizzata dal contributo dato dal magnete

permanente, è coerente con quella teorica, mentre la tensione di asse *d* (5.2.c), all'aumentare della velocità (5.2.b), si discosta da quella calcolata, soprattutto nei transitori tra una rampa di corrente e la successiva (5.2.a).
Vale infatti che, in questo tipo di motori, quando i_d e i_q sono entrambe nulle, la tensione di asse *d* dovrebbe essere teoricamente nulla, non essendoci il contributo dato da $\omega_{me}\Lambda_{mg}$. Si nota invece che ciò non accade, essendoci una deriva negativa di u_d all'aumentare della velocità, come se il contributo Λ_{mg} fosse parzialmente suddiviso in entrambi gli assi. Più precisamente, si è ricavato che circa il 5% del flusso dovuto al magnete permanente va generare una tensione (negativa) di asse *d*.
Ciò che accade si può spiegare con la figura (8.3).

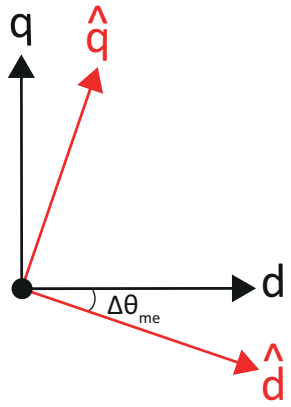
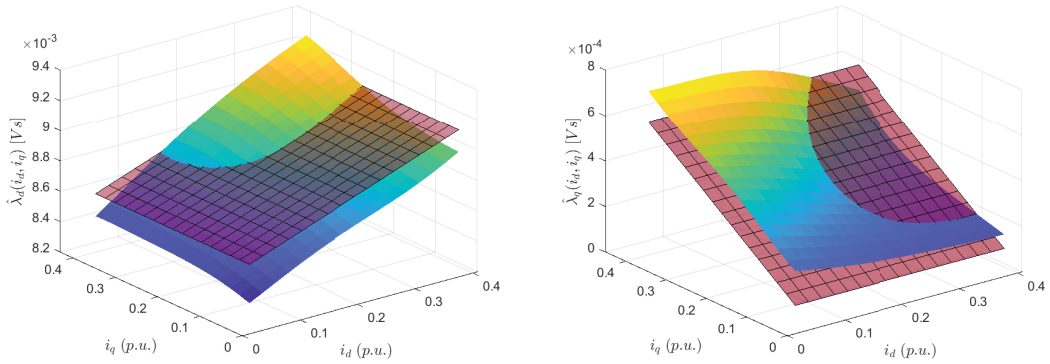


Figura 8.3. Sistema di riferimento sincrono reale e sistema stimato.

Il sistema di riferimento *dq reale* (riportato in nero), con il flusso del magnete permanente allineato sull'asse *d*, ruota con la stessa velocità del rotore, essendo, per costruzione, sincrono con quest'ultimo. Lo stesso non si può dire per il sistema "stimato" dal microcontrollore, utilizzato per calcolare le uscite dei PI dell'anello di corrente. Lo sfasamento tra i due determina lo spostamento di una parte del flusso sulla tensione di asse *d*, in quanto si suppone di controllare il motore in *dq* ma il sistema utilizzato è in ritardo di $\Delta\theta_{me}$.
In un primo momento si è pensato che la causa di tale sfasamento fosse interamente dovuta alla scarsa risoluzione dell'encoder (circa 1.8°). Ripetendo le prove con un encoder più accurato, però, la situazione è marginalmente migliorata, facendo supporre che il problema fosse da ricercarsi altrove.
Analizzando i dati a disposizione, si può intuire come il problema sia causato dall'incoerenza tra le tensioni di riferimento in uscita dai PI di velocità e le altre grandezze. Si ricorda che la frequenza di aggiornamento del sistema di controllo è di $125\,\mu s$. In tale intervallo, la posizione elettrica del sistema *dq* in rotazione alla velocità elettrica di $200\,rad/s$ si sposta di circa 1.5°. Se si somma tale sfasamento alla risoluzione dell'encoder, 1.8°, si ottengono poco più di tre gradi, sufficienti a giustificare l'errore commesso sulla tensione. Vale infatti:

$$\Lambda_{mg,\hat{d}} = \Lambda_{mg} \cdot sin(3^\circ) = \Lambda_{mg} \cdot 0.052, \tag{8.1}$$

esattamente il 5% di scostamento rilevato sul flusso. Eliminare completamente il problema risulta impossibile con le risorse a disposizione, in quanto è causato principalmente dall'elevato numero di coppie polari del motore analizzato. Questa caratteristica, infatti, porta ad avere un numero elevato di gradi elettrici all'interno della rotazione meccanica, rendendone difficoltoso il controllo accurato.



(a) (b)

Figura 8.4. Flussi magnetici concatenati sul sPMSM *Load* con T_c dimezzato.

Per porre rimedio alla problematica è stato sostituito l'encoder a 250 tacche con uno a 512 tacche, sempre con le due uscite in quadratura, migliorando la risoluzione fino a 0.9 gradi elettrici per impulso. Si è inoltre scelto di incrementare la frequenza di controllo del sistema a 15 kHz, per diminuire il tempo di aggiornamento del sistema sincrono $\hat{dq}$. Il risultato ottenuto, in termini di flusso concatenato stimato, è riportato in figura (8.4).

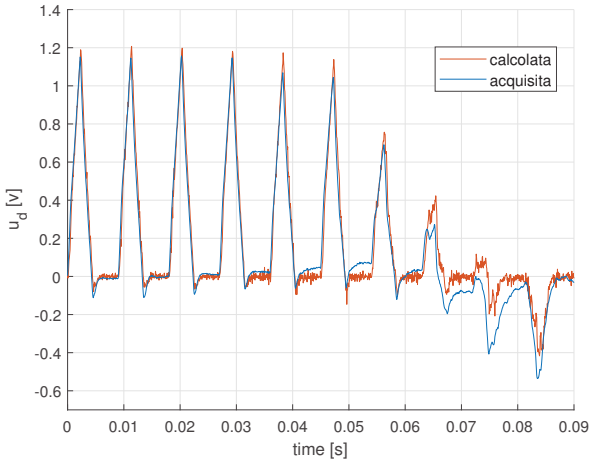


Figura 8.5. Confronto tra la tensione u_d acquisita e quella teorica con T_c dimezzato.

Si nota come la stima sia molto più aderente al valore atteso, nonostante tenda a discostarsi dal flusso teorico nella zona di training a velocità più elevata, ovvero l'area attorno all'asse q. Il miglioramento ottenuto è strettamente legato ad un controllo più accurato, che porta

la tensione di riferimento ad un valore prossimo a quella che realmente arriva al motore, come si evince in figura (8.5).

8.2 *SynR*

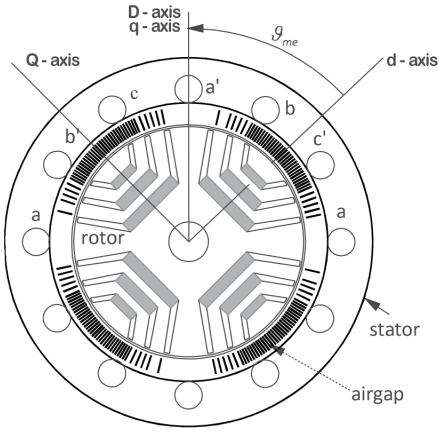


Figura 8.6. Struttura di un motore sincrono a riluttanza [Zig19].

Si è deciso poi di effettuare un'ulteriore prova, analizzando i dati raccolti in un esperimento precedente su di un motore sincrono a riluttanza, la cui struttura è riportata in figura (8.6).
La prima caratteristica di interesse è l'assenza di magneti permanenti. La coppia, infatti, viene generata dall'anisotropia costruttiva che caratterizza il motore, secondo l'equazione:

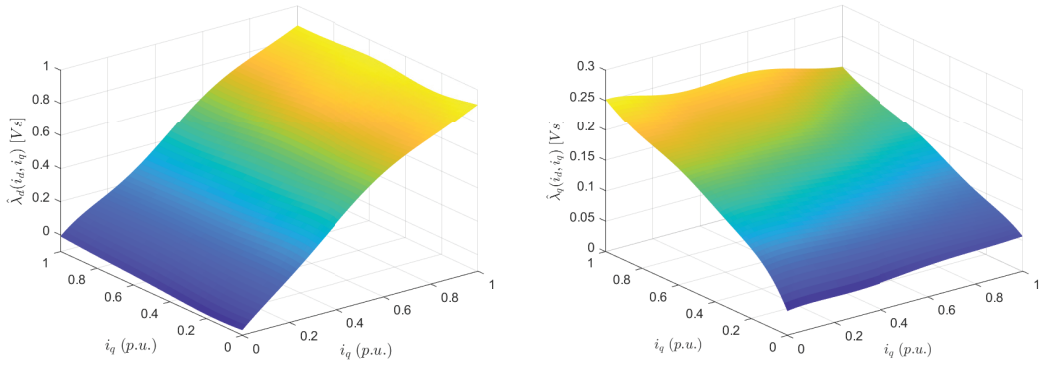
$$\tau = \frac{3}{2} p(\lambda_d i_q - \lambda_q i_d) \tag{8.2}$$

Sono quindi le sole correnti a generare i flussi concatenati, rendendo questi motori particolarmente adatti a sostituire i trifase ad induzione. La robustezza e il costo ridotto sono due punti a favore di questi attuatori, soprattutto se comparati con quelli a magneti permanenti. Sicuramente il peso e l'ingombro sono superiori, a parità di coppia generata, venendo meno il contributo dovuto al magnete. Inoltre, il fatto che la struttura sia interamente realizzata in materiale ferroso, rende questi motori particolarmente soggetti alla saturazione, motivo per cui non è efficiente pilotarli con un controllore lineare, ma è preferibile attuare tecniche di controllo più evolute.
Una possibilità è implementare un *gain scheduling*, ovvero realizzare un controllore adattativo che, in base al punto di lavoro, modifichi i parametri con cui agisce sul sistema in modo da mantenere costante la banda passante e il margine di fase. Questa strategia può essere implementata in modo "statico", con delle lookup table contenenti i parametri del controllore, indicizzate con la coppia di correnti del punto di lavoro, o mediante una rete neurale, che, dinamicamente, calcola le induttanze differenziali dei due assi e le utilizza per ricavare i nuovi guadagni. Per questi motori assume quindi una maggiore rilevanza una stima accurata del flusso magnetico.
In particolare, il motore analizzato presenta le caratteristiche riportate in tabella (8.1).

Simbolo	Parametro	SynR	Unit
R_s	Resistenza di fase	4.76	$[\Omega]$
L_d	Induttanza di asse d	380	$[mH]$
L_q	Induttanza di asse q	85	$[mH]$
I_N	Corrente nominale	4.0	$[A]$
Ω_N	Velocità nominale	1500	$[rpm]$
T_N	Coppia nominale	5.5	$[Nm]$
p	Numero di coppie polari	2	–

Tabella 8.1. Parametri del motore sincrono a riluttanza.

Si nota immediatamente la differenza tra le induttanze dei due assi sincroni. Tale anisotropia deriva dalla struttura stessa del motore ed è strettamente necessaria per la generazione di coppia.
Come si nota in figura (8.6), esistono due diverse strategie per il posizionamento del sistema di riferimento rotante: la prima, seguendo quanto fatto per i motori a magneti permanenti, prevede di porre l'asse d dove le *ribs* generano l'airgap, ovvero dove c'è una maggior riluttanza (quindi $L_d < L_q$). La seconda prevede di considerare l'asse D come quello con una maggior intensità di flusso, ovvero $L_d > L_q$, come in questo caso.
Per addestrare la rete neurale con i dati di questo nuovo tipo di motore, è sufficiente modificare il parametro R_s utilizzato per stimare gli errori, dato che, come descritto finora, il modello è in grado di funzionare con qualsiasi tipo di motore sincrono. Le mappe risultanti dal training sono riportate in figura (8.7).



(a) (b)

Figura 8.7. Flussi magnetici concatenati del SynR stimati dalla rete RBF.

Si nota immediatamente come gli andamenti siano molto meno lineari rispetto ai motori a magneti permanenti superficiali, a causa della saturazione che caratterizza i materiali ferrosi. Inoltre è evidente come nell'asse d l'intensità del flusso sia nettamente superiore rispetto all'asse q, che nel suo percorso magnetico include l'airgap che ne riduce l'induttanza.

Infine si osserva come, essendo un motore in grado di erogare una coppia nettamente superiore rispetto ai due PMSM installati sul banco di prova, il modulo dei flussi stimati sia nettamente superiore a quanto ottenuto nei test precedenti, arrivando a $\lambda_{d,max} = 0.898\,[Vs]$, alla corrente nominale, per l'asse d e a $\lambda_{q,max} = 0.252\,[Vs]$ per l'asse q. Nonostante questa differenza nelle caratteristiche dell'attuatore, la rete è stata in grado di replicarne gli andamenti con precisione, arrivando alla soluzione ottima riportata nei grafici sopracitati.

9

DATA AUGMENTATION ED ESECUZIONE REAL-TIME

Come descritto nei capitoli precedenti, l'addestramento della rete è stato eseguito su una porzione ridotta del primo quadrante del piano *dq*. Il fatto di disporre di una quantità di dati che copre solo un'area dell'intero piano, non è una situazione inusuale. Infatti, come descritto nelle prove effettuate finora, raccogliere una grande quantità di campioni richiede tempo e spazio di archiviazione, oltre che una potenza di calcolo sufficiente ad elaborarli.
Però, se si vuole implementare un controllo che sfrutti l'esecuzione online di una rete neurale, è necessario che quest'ultima sia addestrata in tutte le aree del piano (i_d, i_q) in cui si prevede il motore verrà fatto funzionare, onde evitare di coinvolgere neuroni dal comportamento indefinito. Il requisito minimo è che la rete sia in grado di fornire un'uscita coerente lungo la curva MTPA (Maximum Torque per Ampere), la traiettoria di corrente in cui si produce coppia con la massima efficienza.
Per espandere l'area di training senza raccogliere una maggiore quantità di dati è stato attuato un processo di *data augmentation*, ovvero, a partire dal flusso stimato con i campioni a disposizione e dalla conoscenza della struttura del motore, sono stati generati dei nuovi dati, con i quali la rete è stata riaddestrata.
Il procedimento utilizzato prevede di estendere la mappa del flusso per poter applicare le equazioni del bilancio di tensione (4.1) in un range ampliato di correnti, imponendo dei valori fittizi di velocità.

9.1 *Calcolo delle induttanze differenziali*

Il primo passo per ampliare il range di neuroni addestrati è il calcolo delle induttanze differenziali nell'area già interessata dal training. Le grandezze calcolate verranno poi utilizzate per estendere il flusso stimato, in modo da proseguirne il trend presente lungo il margine dell'area addestrata.
Innanzitutto è necessario calcolare le induttanze $\mathbf{L}_d^{diff}$ e $\mathbf{L}_q^{diff}$. Per fare ciò si sfrutta ancora una volta l'equazione del flusso concatenato che viene ricavato dalla rete:

$$\hat{\lambda}_d = \sum_{k=0}^{K} w_d^k a^k \qquad \hat{\lambda}_q = \sum_{k=0}^{K} w_q^k a^k. \tag{9.1}$$

L'induttanza differenziale è, per definizione, la derivata del flusso rispetto alla corrente.

Considerando, ad esempio, l'asse d, vale:

$$\begin{aligned} L_d^{diff} &= \frac{\partial \hat{\lambda}_d}{\partial i_d} = \frac{\partial \sum_{k=0}^{K} w_d^k a^k}{\partial i_d} \\ &= \frac{\partial \sum_{k=0}^{K} w_d^k e^{-(||\mathbf{i}_{dq} - \mathbf{x}^k||b)^2}}{\partial i_d} \\ &= \frac{\partial \sum_{k=0}^{K} w_d^k e^{-[(i_d - x_d^k)^2 + (i_q - x_q^k)^2]b^2}}{\partial i_d} \\ &= \sum_{k=0}^{K} -2b^2 (i_d - x_d^k) w_d^k e^{-[(i_d - x_d^k)^2 + (i_q - x_q^k)^2]b^2} \\ &= -2b^2 \sum_{k=0}^{K} (i_d - x_d^k) w_d^k e^{-[(i_d - x_d^k)^2 + (i_q - x_q^k)^2]b^2} = -2b^2 \sum_{k=0}^{K} (i_d - x_d^k) w_d^k a^k \end{aligned} \tag{9.2}$$

Il calcolo dell'induttanza potrà quindi essere inserito nell'algoritmo che ad ogni ingresso associa un valore di flusso (6), una volta addestrata la rete. L'aumento del tempo di esecuzione che ne deriverà sarà marginale, infatti tutte le grandezze che compongono l'equazione appena ricavata vengono già calcolate per trovare il flusso stimato.
Per quanto riguarda l'induttanza di asse q, applicando il medesimo procedimento si ricava:

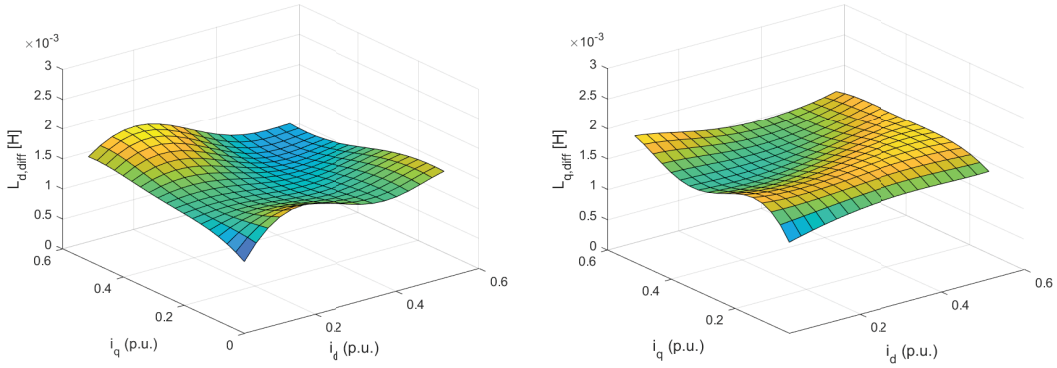
$$L_q^{diff} = \frac{\partial \hat{\lambda}_q}{\partial i_q} = -2b^2 \sum_{k=0}^{K} (i_q - x_q^k) w_q^k a^k \tag{9.3}$$

Mentre per le cross-induttanze vale:

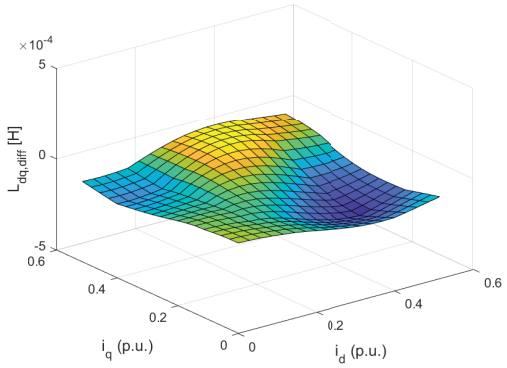
$$L_{dq}^{diff} = \frac{\partial \hat{\lambda}_d}{\partial i_q} = \frac{\partial \sum_{k=0}^{K} w_d^k e^{-[(i_d - x_d^k)^2 + (i_q - x_q^k)^2]b^2}}{\partial i_q} = -2b^2 \sum_{k=0}^{K} (i_q - x_q^k) w_d^k a^k \tag{9.4}$$

$$L_{qd}^{diff} = \frac{\partial \hat{\lambda}_q}{\partial i_d} = \frac{\partial \sum_{k=0}^{K} w_q^k e^{-[(i_d - x_d^k)^2 + (i_q - x_q^k)^2]b^2}}{\partial i_q} = -2b^2 \sum_{k=0}^{K} (i_d - x_d^k) w_q^k a^k \tag{9.5}$$

Ricordando però che $L_{dq}^{diff} = L_{qd}^{diff}$, il calcolo può essere eseguito per una sola delle due grandezze.
In figura (9.1) sono riportati i valori di induttanza L_d^{diff} e L_q^{diff} calcolati con le equazioni sopra riportate.
Si può notare come nell'asse d sia prossima al valore nominale di fase, $L = 1.595\,mH$, mentre nell'asse q si ottenga una quantità leggermente superiore. L'errore commesso può essere imputabile al modulo estremamente ridotto della tensione u_d con il quale, trascurando per un istante il termine di derivata del flusso, $\hat{\lambda_q}$ è strettamente legato (bilancio di tensione (4.1)).
Infatti, come si evince in figura (9.2.a), i vari termini che compongono u_d tendono a compensarsi, mentre all'asse q (9.2.b) è imputato il contributo di forza conetroelettromotrice generata dal magnete permanente che contribuisce ad aumentarne il modulo.



(a)



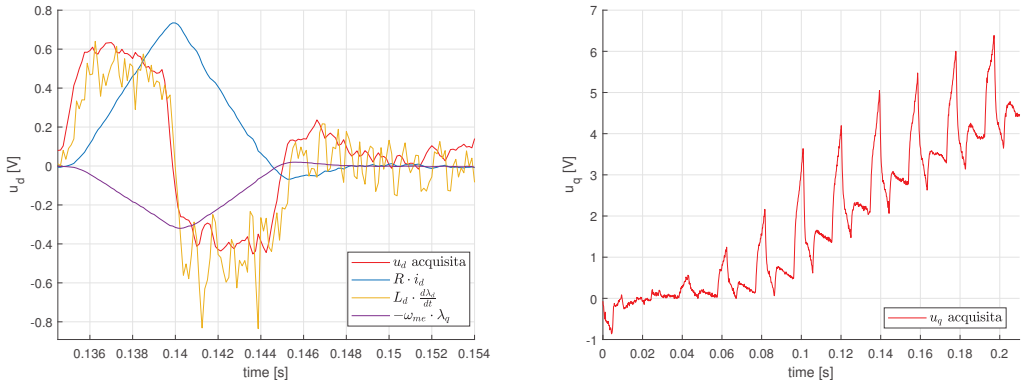
(b)

Figura 9.1. Induttanza differenziale (a) di asse d, (b) di asse q, (c) mutua dq.

Ricordando ora che tali grandezze non vengono misurate ma si assume che al motore arrivi il riferimento imposto, è intuitivo immaginare che un eventuale errore dovuto a questa ipotesi abbia un impatto certamente superiore su u_d, e quindi su $\hat{\lambda}_q$, rispetto a u_q, superiore in modulo.
Infine, come ci si attendeva per questo tipo di motori, la mutua induttanza L_{dq}^{diff} assume un valore prossimo allo zero, in quanto l'influenza reciproca tra i due assi è molto limitata.

9.2 *Estensione delle mappe di flusso*

Prima di generare i nuovi dati è necessario estendere le mappe di flusso $\hat{\lambda}_d$ e $\hat{\lambda}_q$. Per fare ciò, si è scelto di seguire gli step riportati in figura (9.3): innanzitutto, con le misure sperimentali è stata ricavata l'area 1, che si estende fino a poco più della metà della corrente nominale. Nonostante i riferimenti arrivassero a 2.5 A, si è scelto di limitare l'area addestrata a 2 A in quanto, oltre tale soglia, c'è una ridotta quantità di dati che portano ad avere valori di flusso poco attendibili, con induttanze differenziali spesso errate lungo il margine dell'area di training. E le grandezze scelte per attuare l'estensione sono proprio le induttanze $\mathbf{L}_d^{diff}$, $\mathbf{L}_q^{diff}$, $\mathbf{L}_{dq}^{diff} = \mathbf{L}_{qd}^{diff}$, calcolate, rispettivamente, con le



(a)

Figura 9.2. Tensione di riferimento (a) di asse *d*, scomposta nelle varie componenti (b) di asse *q*.

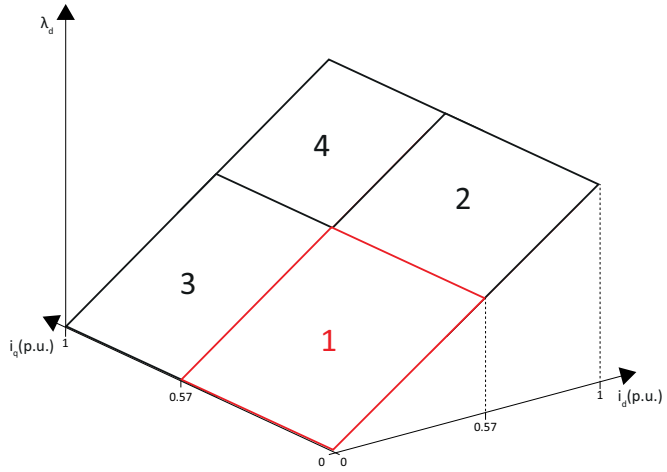
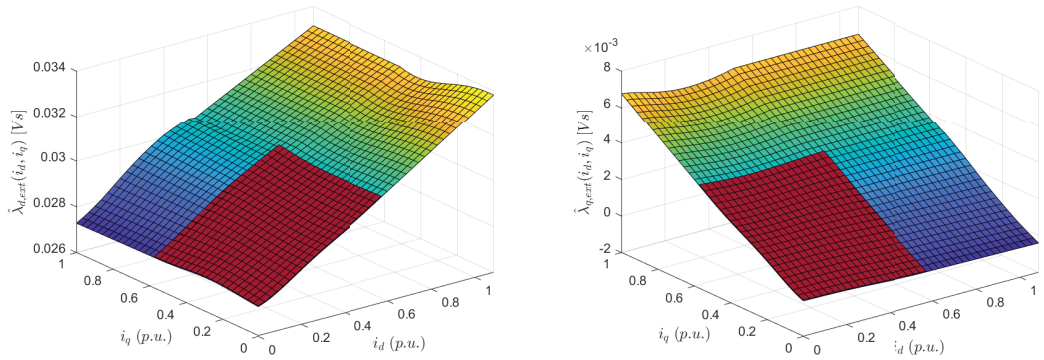


Figura 9.3. Aree di estensione del flusso concatenato stimato.

equazioni (9.2), (9.3) e (9.4).
In particolare, l'induttanza differenziale di asse *d* calcolata lungo il margine del flusso $i_d = 2\,A$, $i_q = 0 : 2A$, verrà utilizzata per espandere la mappa $\hat{\lambda}_d$ nell'area 2. L'andamento ottenuto si basa sull'ipotesi che la saturazione del motore non aumenti oltre al livello presente a $i_d = 2\,A$, ipotesi semplificativa che, però, presumibilmente non porterà ad un errore marcato, trattandosi di un sPMSM, in cui la saturazione è poco accentuata grazie alla disposizione dei magneti.
Per estendere il flusso $\hat{\lambda}_d$ nell'area 3, quindi lungo l'asse *q*, in prima approssimazione si potrebbe ipotizzare che il valore rimanga quello lungo il margine dell'area 1, data la mutua induttanza estremamente ridotta del motore. Per un risultato più preciso, invece, è stata calcolata l'induttanza $\mathbf{L}_{dq}^{diff}$ a $i_q = 2\,A$, in modo da proseguire la mappa con un andamento più aderente a quello del margine dell'area 1.
Infine, per l'area 4, si è scelto di utilizzare la prima delle due strade appena descritte, estendendo l'area 2 con il valore ottenuto a $i_q = 2\,A$. Questa scelta è dovuta a due fattori: il primo è che non si ha a disposizione $\mathbf{L}_{dq}^{diff}$ dell'area 2, essendo una semplice

estensione matematica dell'area 1. Sarebbe quindi necessario eseguire il training della rete includendo i dati delle aree 1, 2 e 3, per poi calcolare le induttanze e, infine, applicare la nuova estensione, computazionalmente dispendioso per un'area così ridotta rispetto al totale. Il secondo è che, dato il valore estremamente ridotto di mutua induttanza (10^{-6}), l'errore commesso sull'area 2 , frutto di un'estensione e non di dati reale, potrebbe generare valori poco attendibili.
I flussi "estesi" ottenuti sono riportati in figura (9.4), in cui si può notare la differenza con l'area precedentemente addestrata (in rosso).



(a) (b)

Figura 9.4. Flussi magnetici estesi nell'intero primo quadrante sfruttando le induttanze differenziali.

9.3 *Generazione dei nuovi dati*

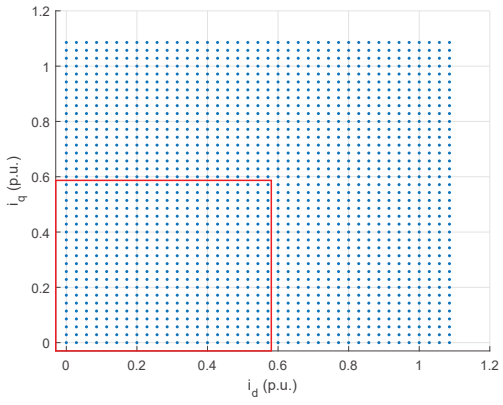


Figura 9.5. Punti di lavoro stazionari nel piano *dq*.

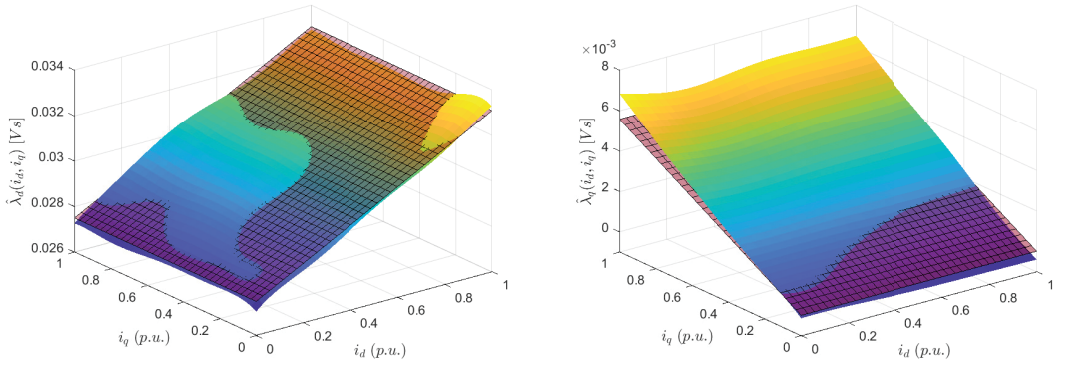
A questo punto si hanno tutti gli elementi per definire il nuovo set di dati che servirà per eseguire l'ulteriore addestramento della rete nell'area ampliata. Dato che, grazie al modello implementato (4.5), l'algoritmo di training è in grado di operare sia con misure

stazionarie che dinamiche, per semplicità si è scelto di generare punti di lavoro stazionari, che rispondono alle equazioni:

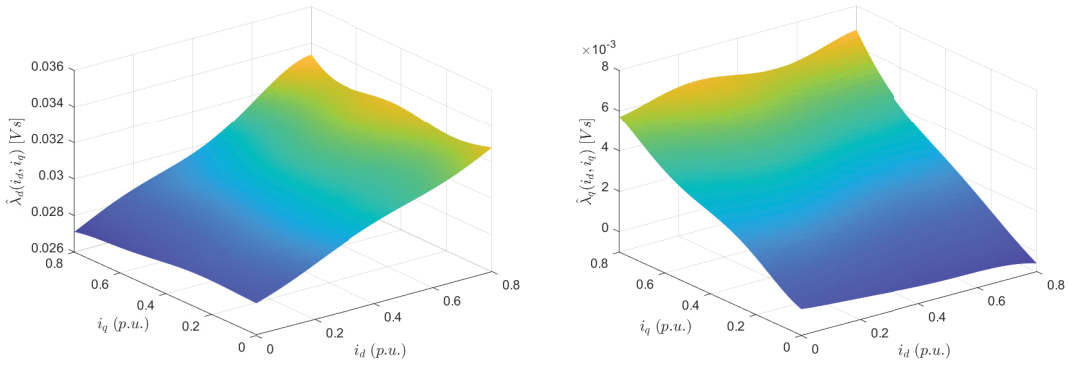
$$\begin{aligned} U_d &= R_s I_d - \Omega_{me}\lambda_{q,ext}(I_d, I_q) \\ U_q &= R_s I_q + \Omega_{me}\lambda_{d,ext}(I_d, I_q) \end{aligned} \tag{9.6}$$

In questo modo si elimina il termine di derivata, evitando di dover inserire un filtro passa basso prima dell'addestramento. Si suppone inoltre che la velocità rimanga costante per tutta la "prova", imponendo un valore di 100 rad/s, in modo da ottenere un termine $\Omega_{me}\boldsymbol{\lambda}_{dq}$ dello stesso ordine di grandezza rispetto a $R_s\mathbf{I}_{dq}$.
In figura (9.5) è riportata la nuova area di lavoro in dq. Come si nota, è stato definito un nuovo campione ogni 0.1 $[A]$, per un totale di 1521 valori.

9.4 *Nuovo addestramento della rete*



(a)



(b)

Figura 9.6. Flussi stimati (a) dall'addestramento della rete con i nuovi campioni generati, (b) campionando rampe di ampiezza massima 3 A.

L'ultimo passo è stato eseguire un ulteriore addestramento della rete con i nuovi campioni generati. I flussi ottenuti sono riportati in figura (9.6.a).
Dal confronto con quelli teorici, sempre ricavati con le equazioni lineari (5.2), si nota

come tali flussi siano coerenti con le aspettative. Il fatto che l'andamento sia molto simile è anche dovuto all'assunzione che la saturazione sia trascurabile, come spesso accade in questo tipo di motori.
Per cercare di verificare tale ipotesi, è stata aumentata la tensione del bus in continua, riuscendo ad arrivare ad un'ampiezza massima delle rampe pari a 3 A, prossima alla corrente nominale del motore analizzato (3.5 A). Le mappe di flusso ottenute sono riportate in figura (9.6.b). Come si nota, l'andamento non presenta alcun termine riconducibile alla saturazione. Il picco che si nota con i_d e i_q vicine ai 3 A, o 0.8 ($p.u.$), è dovuto al fatto che i neuroni in tale zona sono al limite dell'area di training, quindi marginalmente interessati dall'addestramento.

9.5 *Esecuzione della Rete in Real-Time*

Quanto ottenuto finora è utile per la modellizzazione del motore elettrico ma non è ancora stato effettivamente implementato nell'algoritmo controllo. Infatti, i pesi risultanti dall'addestramento sono stati utilizzati per ricostruire il flusso magnetico concatenato stimato, per una generica coppia di correnti $\mathbf{i}_{dq}$, senza però intervenire nei parametri del controllore.

9.5.1 Parametri dei Controllori

Come descritto nella sezione (6.2), i PI di corrente sono stati tarati con il metodo di Bode, assimilando il motore al suo modello lineare, con un'induttanza differenziale costante in ciascuno dei due assi. Però, come si nota chiaramente dai risultati ottenuti analizzando il motore sincrono a riluttanza (sezione (8.2)), questo non è sempre vero, a causa del fenomeno della saturazione magnetica del ferro, che porta il flusso a crescere meno che linearmente all'aumentare della corrente iniettata.
Per aumentare l'efficienza del controllo, è necessario ricalcolare i parametri dei PI in base all'induttanza differenziale che caratterizza il generico punto di lavoro nel piano dq. Questo processo, denominato *gain-scheduling*, può essere eseguito *offline*, creando una lookup table in due dimensioni contenente i diversi guadagni da applicare al variare delle correnti applicate, oppure in real-time, aggiungendo nell'algoritmo di esecuzione della rete neurale, (6), le equazioni (9.2) e (9.3).
Una volta calcolate le induttanze, i nuovi parametri dei controllori possono essere ricavati dinamicamente mediante l'espressione:

$$\tau_c = \frac{tan\left(m_{phi} - \frac{\pi}{2} + tan^{-1}\left(\frac{\omega_c}{R} L^{diff}\right) + tan^{-1}\left(\omega_c T_d\right)\right)}{\omega_c} \tag{9.7}$$

che permette di calcolare i guadagni:

$$k_i = \frac{\omega_c R \sqrt{1 + \left(\frac{\omega_c}{R}\right)^2 \left(L^{diff}\right)^2} \sqrt{1 + \omega_c^2 T_d^2}}{\sqrt{1 + \omega_c^2 \tau_c^2}} \tag{9.8}$$

$$k_p = \tau_c k_i \tag{9.9}$$

Queste equazioni sono state ricavate dall'articolo di Riccardo Antonello "*Advanced Current Control of Synchronous Reluctance Motors*" [Ant+17].
L'algoritmo (7) riassume i passaggi necessari per il funzionamento online della rete neurale.

Algorithm 7 Esecuzione RBF

1: **for** tutti i K nodi **do**
2: Calcolo distanza euclidea (4.7)
3: Calcolo uscita gaussiana (4.8)
4: **end for**
5: Calcolo induttanza L_d^{diff} (9.2)
6: Calcolo induttanza L_q^{diff} (9.3)
7: Calcolo costanti di tempo $\tau_{c,d}$, $\tau_{c,q}$ (9.7)
8: Calcolo guadagni integrali $k_{i,d}$, $k_{i,q}$ (9.8)
9: Calcolo guadagni proporzionali $k_{p,d}$, $k_{p,q}$ (9.9)

Il calcolo della distanza dell'ingresso dai nodi ripercorre quanto proposto per la ricerca del flusso concatenato, attuata nel capitolo (7.2). A questa si aggiunge il calcolo dell'induttanza differenziale per ogni asse, seguito dalla ricerca dei guadagni dei controllori. É immediato constatare la dinamicità della procedura, in quanto questi ultimi variano in modo continuo in base alla coppia di correnti in ingresso.

9.5.2 Risultati sperimentali dell'algoritmo Real-Time

Il limite del processo descritto è, appunto, il fatto che il risultato dev'essere fornito entro un tempo massimo, in questo caso il periodo di controllo T_c. Quanto appena riportato è la definizione di *Real-Time task*, in cui non basta che il calcolo sia corretto, è anche necessario rispettare precisi vincoli temporali.
I passaggi da attuare, però, sono computazionalmente onerosi, includendo, tra gli altri, l'esponenziale che caratterizza le gaussiane. Infatti, inserendo nel codice il gain scheduling dei PI con la rete a 36 nodi su cui è stato eseguito l'addestramento, l'esecuzione delle varie funzioni viene completata quando il counter della PWM è arrivato a circa 8500 unità. Ricordando che il limite di conteggio è settato a 9375, il valore raggiunto corrisponde a quasi il 90% del tempo utile a disposizione. Nel caso tale limite venisse superato, inizierebbe l'esecuzione dell'interrupt successivo prima di concludere con quello attuale, generando un errore di sistema.
Una possibile strategia per ridurre la complessità progettuale, consiste nell'approssimare l'esponenziale con un polinomio. Così facendo, però, si perde accuratezza nella stima delle induttanze, e questo si traduce in una minor efficienza del sistema. Per questo motivo è stata utilizzata la funzione *exp* presente nella libreria *math.h*, che riproduce fedelmente l'andamento della funzione matematica in modo ottimizzato, nel minor numero possibile di istruzioni macchina.
Risulta inoltre fondamentale la scelta del numero di nodi della rete, compromesso tra la risoluzione con cui si vogliono approssimare le mappe e il tempo a disposizione per il calcolo. I 36 neuroni implementati, infatti, permettono di ottenere un'esecuzione suffi-

cientemente rapida e, al tempo di stesso, una buona accuratezza nella risposta. In aggiunta a ciò, è necessario ricordare che l'addestramento era stato eseguito nel primo quadrante, quindi le prove di esecuzione online sono state effettuate nel medesimo settore, onde evitare di includere nel risultato dati inerenti a nodi non addestrati.
Nelle prove eseguite, infine, è stato rimosso l'algoritmo di compensazione dei tempi morti, il quale, come riportato nel paragrafo (7.3), richiederebbe un ulteriore 20% di T_c per l'esecuzione.

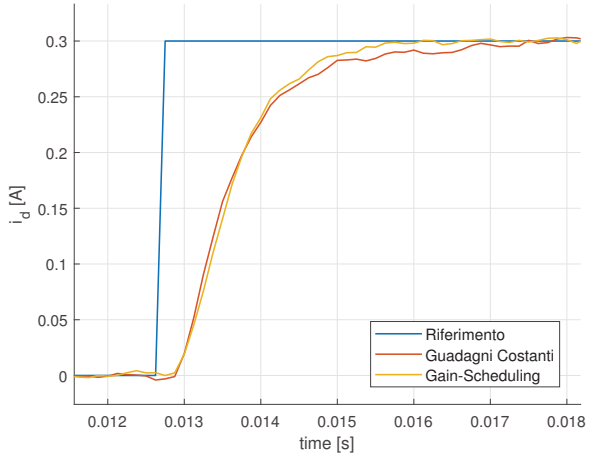


Figura 9.7. Risposta del sistema al gradino di corrente i_d con e senza gain-scheduling.

Le induttanze trovate sono comunque in linea con quelle riportate in figura (9.1), nell'area in cui la rete è stata addestrata. Per verificare la correttezza della soluzione implementata, si è deciso di comparare il risultato ottenuto con il gain scheduling con quello ottenuto con la taratura lineare dei PI, riportata in sezione (6.2). Trattandosi di un motore sincrono a magneti permanenti isotropo, la caratteristica del flusso concatenato segue un andamento lineare, almeno finché il punto di lavoro non si avvicina alla corrente nominale. Per tale motivo, la risposta del sistema con le due soluzioni implementate dovrebbe essere molto simile. Come riferimento da seguire si è scelto di imporre un gradino di corrente di asse d, per mantenere il motore fermo, al fine di testare la sola risposta dell'anello di corrente, senza dover includere dinamiche meccaniche.
Il risultato ottenuto è riportato in figura (9.7). Come ci si attendeva, le due risposte sono molto simili a causa della caratteristica lineare del motore, con una convergenza del sistema basato sul gain-scheduling verso lo stato di regime leggermente più rapida. Inoltre, per entrambi gli andamenti, è presente un ritardo di quasi $2T_c$ rispetto alla variazione del riferimento. Questo è dovuto al controllo discreto attuato nel sistema (T_c), al ritardo introdotto dall'inverter, in cui si assume che la tensione media nel periodo sia pari al riferimento imposto ($0.5T_c$), e al tempo di risposta dell'anello di corrente (circa $0.5T_c$).

CONCLUSIONI

Prima di procedere con i commenti conclusivi, è necessario ricordare qual era lo scopo iniziale della tesi (descritto nell'introduzione) oltre a riassumere brevemente il percorso logico che ha permesso di ottenere i risultati riportati nei vari capitoli.
L'obiettivo della ricerca era verificare la possibilità o meno di implementare una rete neurale, finalizzata alla stima del legame flusso-corrente di un motore elettrico, su di un microcontrollore industriale, dal costo di alcuni ordini di grandezza inferiore rispetto agli elaboratori con cui solitamente vengono addestrate le reti.
Per fare ciò, è stato innanzitutto necessario scegliere i parametri della rete, come numero di nodi e quantità di dati forniti alla stessa per il training. La principale limitazione di un sistema di questo tipo è rappresentata dalla quantità di memoria a disposizione, oltre che dalla potenza di calcolo. Per tale motivo per cui si è reso necessario ridimensionare il problema, riducendo il numero di campioni e limitando l'analisi ai quadranti di interesse, in questo caso il primo. Inoltre, come si è dimostrato nell'analisi delle soluzioni software del capitolo (6), si è reso necessario prestare attenzione alla metodologia utilizzata nell'implementazione delle funzioni matematiche più onerose, come l'inversione di matrice che caratterizza l'addestramento, onde evitare di ottenere un programma poco efficiente e un tempo di esecuzione eccessivamente prolungato.
Passando all'analisi dei risultati, consultabili nei capitoli (7) e (8), si può intuire come la rete implementata porti effettivamente ad ottenere delle mappe più che soddisfacenti, nonostante le risorse hardware limitate. Questo apre la strada alla possibilità di controllare in modo efficiente i motori elettrici caratterizzati da un legame flusso-corrente non lineare, come i sincroni a riluttanza, anche in applicazioni con un costo contenuto. Non è infatti immaginabile sostituire un motore ad induzione, magari collegato direttamente a rete, con un azionamento che necessiti un costoso sistema di controllo, ad elevata potenza di calcolo, per funzionare correttamente. Ed è da escludere anche una soluzione che preveda un controllo lineare su di un attuatore caratterizzato da una forte saturazione magnetica, onde evitare una perdita di efficienza o, nel peggiore dei casi, una perdita del controllo del sistema.
Come analizzato all'inizio di questo paragrafo, per migliorare la qualità dei risultati ottenuti bisognerebbe aggiungere una certa quantità di memoria SRAM nell'unità di controllo, in modo da compiere un training più accurato grazie ad una mole maggiore di dati raccolti. Esiste però una seconda soluzione che in molti casi può migliorare la mappa ottenuta in uscita, che permette di evitare i problemi incorsi nei test eseguiti col motore a cinque coppie polari. Questa soluzione consiste nel misurare la tensione effettivamente applicata sulle fasi del motore e non assumere semplicemente che sia pari al riferimento imposto dai PI di corrente. Ovviamente, così facendo, si complica ulteriormente la struttura del sistema, inserendovi due nuovi sensori, oltre che impiegare un ADC a due

canali aggiuntivo. É quindi necessario valutare attentamente il rapporto costi-benefici in base all'applicazione in cui si andrà ad implementare la rete e alla risoluzione che si vuole ottenere in uscita.
Quanto implementato nel progetto è finalizzato a verificare la fattibilità di una soluzione di questo tipo in un sistema a costi contenuti, perciò le varie fasi del processo, ovvero l'addestramento e il funzionamento online, sono state implementate in modo separato. Con modifiche marginali all'algoritmo, però, si può ottenere un importante vantaggio che deriva dall'impiego di una rete neurale, ovvero la possibilità di operare in modo completamente autonomo in ogni fase. É infatti possibile realizzare un software che, senza alcun intervento da parte dell'installatore dell'azionamento, caratterizzi completamente il motore collegato all'inverter e, automaticamente, modifichi i parametri del controllore in modo da ottenere la massima efficienza.
Un algoritmo di questo tipo andrebbe ad attuare, in modo sequenziale, le varie prove descritte nell'elaborato, ovvero:

- Test per tarare la compensazione dei tempi morti e ricerca della resistenza di fase (5.4)
- Iniezione dei riferimenti di corrente (7.1)
- Addestramento della rete neurale (4)
- Funzionamento online della rete neurale (9.5)

Infine, eseguendo a intervalli temporali prefissati il training e confrontando ogni volta il nuovo risultato ottenuto con quello precedente, è possibile tracciare l'eventuale variazione dei parametri dell'attuatore nel tempo. Questa strategia permette di verificare lo stato di salute del motore, aiutando a prevenire eventuali guasti prima che insorgano, al fine di programmarne la sostituzione nel momento più consono, onde evitare interruzioni indesiderate del processo produttivo dell'azienda.

BIBLIOGRAFIA

[Low89] David Lowe. «Adaptive radial basis function nonlinearities, and the problem of generalisation». In: *First IEE International Conference on Artificial Neural Networks* (1989).

[Hay98] Simon O. Haykin. *Neural Networks: A Comprehensive Foundation*. Pearson Education (US), 1998.

[PZ11] Luca Peretti e Mauro Zigliotto. «Tecniche di Modulazione Vettoriali». In: *UniPd* (2011).

[Hag+14] Martin T. Hagan et al. *Neural Network Design, 2nd edition*. 2014. URL: https://hagan.okstate.edu/NNDesign.pdf.

[ATZ15] Riccardo Antonello, Fabio Tinazzi e Mauro Zigliotto. «Benefits of Direct Phase Voltage Measurement in the Rotor Initial Position Detection for Permanent-Magnet Motor Drives». In: *IEEE Transactions on Industrial Electronics* 62.11 (2015), pp. 6719–6726. DOI: 10.1109/TIE.2015.2448514.

[Ant+17] Riccardo Antonello et al. «Advanced Current Control of Synchronous Reluctance Motors». In: *IEEE PEDS* (2017). DOI: 10.1109/PEDS.2017.8289150.

[Car+17] M. Caruso et al. «Enhanced Loss Model Algorithm for Interior Permanent Magnet Synchronous Machines». In: *AEIT International Annual Conference* (2017).

[OTZ18] Ludovico Ortombina, Fabio Tinazzi e Mauro Zigliotto. «Magnetic Modeling of Synchronous Reluctance and Internal Permanent Magnet Motors Using Radial Basis Function Networks». In: *IEEE Transactions on Industry Applications* 65.2 (2018), pp. 1140–1148.

[Zig19] Mauro Zigliotto. «Azionamenti per motori sincroni a riluttanza». In: *UniPd* (2019).

[20a] 2020. URL: https://www.studyread.com/synapse/.

[Ort+20] Ludovico Ortombina et al. «Magnetic Model Identification of Synchronous Motors Considering Speed and Load Transients». In: *IEEE Transactions on Industry Applications* 56.5 (2020), pp. 4945–4954.

[20b] *SAM E70/S70/V70/V71 Family*. Microchip. 2020.