Politecnico di Torino

Facoltà di Ingegneria

Dipartimento di Ingegneria Meccanica e Aerospaziale

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Sviluppo di un sensore sintetico per la stima dell'angolo d'attacco tramite rete neurale *Generalized Radial Basis Functions*

Candidato:
Simone Giannattasio
Matr. 253907

Relatore: Angelo Lerro
Correlatore: Piero Gili

Anno Accademico 2019/2020

Ringraziamenti

A mia madre, a mio padre e a mia sorella che mi hanno sempre supportato anche nei momenti più difficili. Senza di voi non ce l'avrei mai fatta, siete le mie stelle polari.

A tutte le persone che mi sono state sempre vicine e che mi vogliono bene. Ve ne voglio anche io.

Sommario

Il principale problema con cui si ha a che fare nella progettazione dell'apparato sensoristico di un velivolo ultraleggero è quello di ottenere una buona ottimizzazione del velivolo in termini di costo, peso, ingombro aerodinamico senza, tuttavia, rinunciare ai requisiti di sicurezza.

Nuovi sistemi digitali di controllo come il Flight By Wire (FBW) sono applicati con successo ai grandi aeroplani e saranno, nei prossimi anni, lo standard anche per piccoli velivoli da trasporto. Le nuove soluzioni avioniche digitali permettono, infatti, l'installazione anche su velivoli ultraleggeri di sistemi di controllo avanzati.

Uno dei segmenti più critici per un sistema di controllo è l'ADS (Air Data System) che fornisce dati critici dall'esterno per la sicurezza del velivolo. I sempre più stringenti requisiti di aeronavigabilità prevedono l'installazione, su ogni velivolo, di ADS in ridondanza che possano fungere da supporto per il sistema di controllo principale. È evidente che inserire un ulteriore sistema ingombrante e pesante (come quelli presenti attualmente sul mercato) su un velivolo che fa della leggerezza la sua caratteristica peculiare, risulta estremamente complicato e fonte di non poche problematiche. In tal senso, i sensori sintetici che integrano reti neurali possono costituire una valida soluzione: ciò permetterebbe di ridurre la probabilità di un'errata diagnosi di failure da parte dell'ADS principale.

Una corretta progettazione dei sensori sintetici passa inevitabilmente attraverso la progettazione di reti neurali performanti, che riescano a stimare le grandezze di interesse con notevole precisione, anche in condizioni di volo estreme. Quest'ultimo aspetto è fondamentale perché il sensore virtuale che ci si propone di progettare riesca ad ottenere una certificazione aeronautica necessaria per diventare operativo a tutti gli effetti sul velivolo.

Gli angoli aerodinamici sono ricavati da dati inerziali, comandi di volo e velocità relativa ottenuta dal sistema statico di Pitot. La relazione tra que-

sti parametri e gli angoli aerodinamici rappresenta una funzione complessa e non lineare difficile da determinare per via analitica, costruendo dei modelli di aeromobile. L'obiettivo principale di questo lavoro, che è mirato alle applicazioni su velivoli ultraleggeri di piccole dimensioni, è quello di analizzare il sistema del velivolo e sviluppare un sensore virtuale che sfrutti metodi di soft computing, come le tecniche di predizione neurale, per misurare l'angolo d'attacco nell'ordine di valutare la fattibilità di questo tipo di sistema neurale.

In quest'ottica il presente lavoro mira ad analizzare le prestazioni di una rete neurale GRBF (Generalized Radial Basis Functions) per la determinazione dell'angolo d'attacco del velivolo utilizzando dati raccolti in ambiente reale su un ultraleggero G -70 della Nando Groppo Srl.

La questione può essere considerata di notevole interesse se si tiene presente che le MLP (Multi-Layered Preceptrons) e le altre architetture neurali convenzionali sono state abbondantemente testate per problemi di questo tipo, mentre lo stesso non si può dire per le reti neurali RBF.

La trattazione proposta mira, dunque, a chiarire il funzionamento generale di queste reti andando a scoprire ed analizzare la loro natura intrinseca e le caratteristiche peculiari che le rendono ottimi 'approssimatori universali' [1], alla pari delle MLP.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Gli *Air Data Systems* (ADS) sono adottati su veicoli aerei con lo scopo di misurare una serie di dati dall'ambiente esterno. Nei tempi più recenti, grazie anche allo sviluppo di più moderne tecnologie, la tendenza è quello di spostare l'ADS verso soluzioni digitali che consentano una migliore integrazione con l'avionica digitale più moderna di tipo *Fly-by Wire*. Tale sistema di controllo, ormai sempre più applicato soprattutto ad aeromobili di grandi dimensioni, ha sostituito i tradizionali comandi di volo costituiti da attuatori meccanici con un sistema di controllo elettronico digitale. La barra di comando è composta da una serie di trasduttori e sensori il cui compito è quello di inviare ai computer di bordo i segnali elettrici relativi ai comandi. Questi segnali, insieme ad altri dati sono successivamente rielaborati dal computer di bordo in modo da essere compatibili con una serie di attuatori appositi che mettono in moto le superfici servendosi di altri attuatori elettromeccanici.

Nell'ambito del Flight Path 2050, la transizione tecnologica al Fly-by-wire è una strada necessaria per raggiungere gli obbiettivi definiti dalla comunità europea. L'industria aeronautica ha lanciato diversi programmi per far fronte alla sfida del fly-by-wire, anche per quei sistemi che sembravano meno coinvolti in questa rivoluzione, come l'ADS. Per digitalizzare l'ADS e superare tutti gli eventuali inconvenienti legati alla connessione pneumatica di sonde e palette agli ADM e quindi di ogni ADM al sistema di controllo di volo, si è reso necessario lo sviluppo di una serie di sonde con trasduttori incorporati al loro interno.

Lo stesso approccio è condiviso con quei veicoli aerei di dimensioni ridotte, ad esempio la categoria di piccoli aerei per il trasporto (SAT), i velivoli aeronautici senza pilota (UAV) e veicoli per la mobilità aerea urbana (UAM), dove il FBW è necessario per un sistema più integrato.

Gli sforzi spesi dai governi di tutto il mondo per regolare il lavoro aereo relativamente a queste categorie sono sempre più consistenti, vista e consi-

derata l'importanza cruciale di questi veicoli per spostamenti a corto raggio e collegamenti tra comunità locali (solo per fare alcuni esempi). Dunque, l'interesse in tale ambito è sempre crescente e potrebbe subire un'impennata negli anni a seguire.

Soffermandosi sull'ADS, questi velivoli presentano alcune criticità comuni nell'installazione di sistemi convenzionali per la misurazione dei dati esterni. I problemi principali sono legati all'ingombro e al peso dei classici sistemi presenti sul mercato, visto lo spazio limitato per un'istallazione sulla fusoliera. Questo problema è ancor più rilevante se si considera che spesso è necessaria un'istallazione multipla di ADS per soddisfare i requisiti di aeronavigabilità che prevedono una ridondanza dei sistemi cruciali sul velivolo. L'ADS, infatti, è adibito alla misurazione di molte grandezze la cui perdita può essere considerata critica per il velivolo stesso, nel senso che rappresenta una criticità per la sicurezza dell'aeromobile al punto da causarne la *failure*. L'architettura comunemente utilizzata per rientrare nei parametri di sicurezza è quella di un triplex ADS. Dunque, è facile comprendere come un sistema invasivo e pesante quale un ADS convenzionale risulti difficilmente installabile con ridondanza su un veicolo ultraleggero.

Da quanto detto, si comprende che è sempre più forte l'interesse, nonché la necessità di trovare delle soluzioni alternative per migliorare l'affidabilità in termini di controllo su questi velivoli senza intaccare però ingombro e peso, caratteristiche fondamentali perché un ultraleggero sia classificabile come tale.

Per ovviare al problema appena esposto, la tendenza è quella di sostituire dispositivi hardware pesanti, ingombranti e costosi con codici di software eseguibili. Un esempio concreto consiste nella 'ridondanza analitica', intesa come la sostituzione di alcuni dei sensori effettivi (ad esempio le sonde per la misurazione dell'AoA) con sensori virtuali. Lo scopo è quello di fornire dati alternativi come metro di paragone per rilevare eventuali errore o incongruenze nelle misurazioni dei sensori hardware ed evitare, in questo modo, le modalità di failure più comuni delle moderne ADS. Questa soluzione sarebbe la più valida ed utile ad innalzare il livello di sicurezza su un ultraleggero, conformandosi alle severe norme di aeronavigabilità. Già negli anni 70 gli studi sulla ridondanza vennero affidati a computer digitali. L'interesse si incrementò nel decennio successivo quando apparvero una serie di articoli centrati proprio sulla gestione di sistemi di sensori ridondanti. Negli ultimi due decenni, con l'affinamento delle tecniche computazionali, sono state pro-

poste soluzioni algoritmiche basate su tecniche *model based* che tentavano di creare un modello matematico per schematizzasse l'aeromobile. Proprio l'inevitabile discrepanza tra il modello matematico e quello reale, in un sistema altamente complesso come quello di un aereo, costituisce il limite principale di questi metodi.

Un esempio di sensore virtuale *model based*, progettato e brevettato da Wise, è effettivamente utilizzato sugli aerei Boeing X-45° per stimare gli angoli aerodinamici. Per questo scopo, il sensore si serve dei dati inerziali, di un modello accurato del velivolo e di un filtro di Kalman. Per superare il problema del *model based* si è passati ad un approccio *model learned* che ha coinvolto l'uso delle reti neurali per la stima degli angoli aerodinamici servendosi degli altri dati misurati sul velivolo: Rohlo, Samy e Green, ad esempio, hanno creato una rete in grado di determinare completi set di parametri di volo a partire dalle misurazioni della pressione statica sulla fusoliera, senza bisogno di dati inerziali.

Tutti gli esempi virtuali condividono l'uso della pressione dinamica, che è chiaramente un dato molto complicato da stimare indirettamente. Una possibile soluzione sarebbe quella di stimare la pressione dinamica per via indiretta tramite un altro sensore virtuale. In ogni caso, la maggior parte dei sensori virtuali basati su reti neurali che sono oggetto di studio, riceve la pressione dinamica dal sistema ADS principale, i dati inerziali dal sistema AHRS (attitude and heading referece system) e i comandi dal FCS. Questi dati sono sfruttati per determinare le grandezze di interesse: nel caso in esame l'angolo d'attacco. L'intento è quello di creare un ADS parzialmente basato su sensori sintetici. Un esempio è costituito dall'architettura MIDAS sviluppata al Politecnico di Torino nell'ambito del progetto *Clean Sky 2*[13].

Dunque, la 'ridondanza analitica' rappresenta una soluzione innovativa che permette di utilizzare dei sensori virtuali basati su tecniche neurali apportando benefici in termini di costo, peso e ingombro relativamente a velivoli ultraleggeri. Inoltre, l'introduzione di sensori virtuali permetterebbe di superare alcuni problemi legati all'incongruenza di informazioni tra ADS ridondanti, che spesso sono è causa di failure per i velivoli più moderni. Lo scopo del qui presente lavoro è quello di stabilire la fattibilità di un sensore virtuale *model learned*, cioè che utilizza una rete neurale, per la stima dell'angolo d'attacco in ambiente reale. La rete che si progetterà per supportare tale studio è una rete *Generalized Radial Basis Functions*. I dati in ingresso alla rete sono stati raccolti durante una campagna di volo condotta

da un ultraleggero G-70 della Nando Groppo Srl. Quindi l'interesse si concentrerà sulla valutazione delle performance della rete addestrata e testata in ambiente reale con lo scopo di verificare il comportamento della rete in un ambiente particolarmente ostico in cui i risultati potrebbero degradarsi a causa di raffiche di vento, fenomeni di turbolenza e vibrazioni del motore.

Capitolo 1

Air data system (ADS)

Con la sigla ADS (*Air Data System*) si intende l'insieme di sensori e sonde che sono disposti sull'aereo e che servono a misurare e monitorare costantemente, in modo diretto, tutte le principali grandezze in volo. Questo sistema sensoristico fornisce successivamente i dati ad un ADC (*Air Data Computer*) la cui funzione è quella di convertire tali dati in segnali elettrici da trasferire successivamente al FCS (*Flight Control System*) per il controllo automatico dell'aeroplano. I segnali elettrici provenienti dall'ADC comunicano, dunque, con il FCS che provvede a gestire automaticamente le superfici di controllo attraverso altri segnali elettrici. Il percorso può anche essere inverso, nel senso che il pilota può effettuare un comando a sua volta convertito in segnale elettrico e passato alle superfici di controllo che lo 'attuano'. L'ADC contiene degli algoritmi di correzione, calibrazione e conversione dei dati in ingresso dai sensori che servono appunto a trattare tali valori e calcolare tutte le grandezze necessarie prima che siano convertite in segnali elettrici e fornite al FCS come informazioni per il controllo.

Di solito ogni velivolo è disposto di 2-3 ADS perché la ridondanza è fondamentale, come ben noto in ambito aeronautico, per conformarsi alle severissime normative di aeronavigabilità e sicurezza. Ciò consente di incrementare quanto basta il fattore di sicurezza. Molti incidenti aerei sono stati causati proprio da una *failure* fatale dell'ADS o da alcuni difetti di integrazione dell'ADS con il sistema di controllo automatico.

Uno dei più recenti incidenti, risalente al 2019, riguarda il Being 737 Max. Così titolava il Sole 24 ore, il 12 Dicembre 2019: *"Boeing, aviazione Usa sotto accusa: sapeva che il 737 Max rischiava incidenti ogni 2-3 anni"*; Mr. Sully Sullengerger, ex capitano della US airways affermava: *"the fatally*

flawn design of the MCAS was a death trap." e ancora: *"The MCAS design shuold never have been approved, not by Boeing, and not by the FAA."*

Il problema era legato al fatto che i computer di bordo erano alimentati indipendentemente da due sensori AoA duplicati il che creava un messaggio di errore "*AoA disagree*" sul pannello di controllo del pilota. In condizioni particolari di volo, si attivava il sistema di controllo MCAS con l'obiettivo di ridurre la tendenza del velivolo a cabrare. L'incongruenza che si generava per la presenza dei due sensori AoA indipendenti sull'aeroplano, faceva andare in tilt il sistema di controllo MCAS. Boeing non riteneva necessario indicare "AOA DISAGREE" in quanto in presenza di tale errore si sarebbero attivati anche altri flag di errore ad avvisare l'equipaggio. Pertanto, il sistema AOA e relativo flag errore rimasero opzionali. Secondo la compagnia Boeing, l'equipaggio avrebbe dovuto manualmente disattivare il sistema MCAS così come successo su altri voli con equipaggi più preparati. Ma questo molte volte non è avvenuto, causando numerosi incidenti.

Figura 1.1: tipico messaggio di errore sul Boeing 737 Max

È chiaro, a questo punto, come una corretta progettazione dell'ADS in modo che esso sia perfettamente compatibile con il FCS sia un argomento di estrema importanza e delicatezza nel mondo aeronautico, a maggior ragione se si ha a che fare con velivoli ultraleggeri per i quali, come vedremo in seguito, le norme sono ancora più stringenti.

L'architettura di un sistema ADS standard è mostrata nella figura 1.2.Come già accennato tale sistema è dotato di alcune sonde e sensori che prendono le informazioni dall'ambiente esterno e le passano all'ADC, il quale le converte in segnali elettrici.Di solito sono presenti due *static flush ports* per la misurazione della pressione statica e un tubo di pitot per misurare la pressione totale, in più è presente una sonda per la misurazione diretta dell'angolo

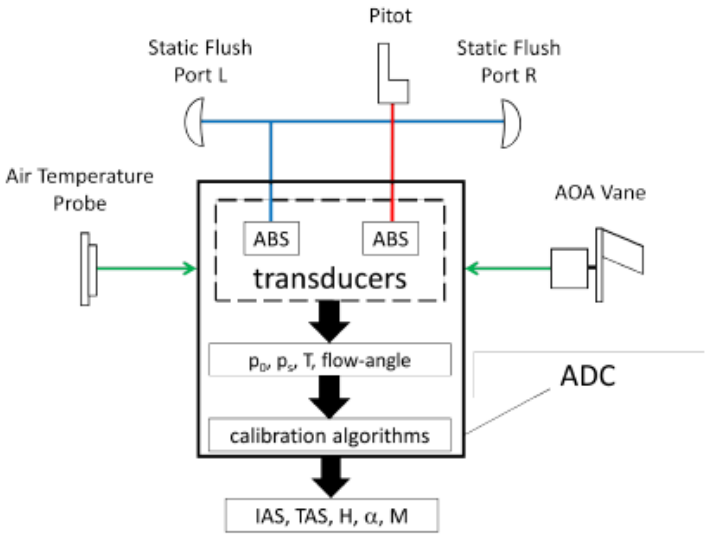


Figura 1.2: Schema dell'ADS convenzionale

d'attacco e un'altra sonda per la misurazione diretta della temperatura. Tutti questi sensori sono disposti nei pressi del naso della fusoliera.

Le grandezze misurate direttamente sono dunque:

- La pressione statica p
- La pressione totale p_o
- La temperatura totale T_o
- La temperatura statica T
- L'angolo d'attacco α (rispetto agli assi di riferimento della fusoliera)

Dalla pressione statica e quella totale è possibile calcolare la velocità indicata (IAS) come segue:

$$p_0 = p + \frac{1}{2}\rho \left|\langle v \rangle\right|^2 \tag{1.1}$$

$$IAS = \sqrt{\left(\frac{2\left(p_0 - p\right)}{\rho_{SL}}\right)} \tag{1.2}$$

Da questa è possibile poi passare alla velocità calibrata (CAS) e alla velocità vera (TAS). Sempre dalla pressione statica e da quella totale si calcola il numero di Mach come segue:

$$\mathrm{M} = \sqrt{\frac{2}{\gamma - 1}\left[\left(\frac{p_0}{p}\right)^{\frac{\gamma-1}{\gamma}} - 1\right]} \tag{1.3}$$

Gli angoli aerodinamici locali α_{loc} e β_{loc} sono misurati da sonde apposite. I valori all'infinito dell'angolo di derapata e dell' angolo d'attacco sono poi ottenuti con delle particolari leggi di calibrazione. Più raramente, invece, β può essere calcolato dalle misurazioni della pressione statica a destra e sinistra della fusoliera mediante la seguente legge di calibrazione:

$$\beta = K_\beta(p_{,L} - p_{,R}) \tag{1.4}$$

$$K_\beta = f(\alpha, M, \beta) \tag{1.5}$$

1.1 Architetture alternative ADS

L'architettura standard di un ADS, utilizzata per la maggior parte su aerei civili, può essere sostituita da architetture alternative più moderne e convenienti da punto di vista dell'ingombro ma non sempre di costi, come si vedrà.

La prima architettura che si analizza è chiamata ADS avanzato: essa è simile all'ADS standard con la differenza che la maggior parte dei sensori e delle sonde, eccetto il sensore di temperatura, sono sostituite da un *multifunction probe* cioè da un sensore multifunzione in grado di misurare più grandezze contemporaneamente. Inoltre l'ADC è sostituita con un ADU. Tale architettura è utilizzata ad esempio sull'Airbus A400M oppure sull'Alenia Aermacchi M346. Il vantaggio principale consiste nel fatto che si riducono le sonde, aspetto positivo da un punto di vista aerodinamico. Gli svantaggi riguardano il costo e l'impossibilità di calcolare l'AoS. Per calcolare anche l'AoS bisognerebbe inserire due ADS avanzati e connetterli entrambi al FCS.

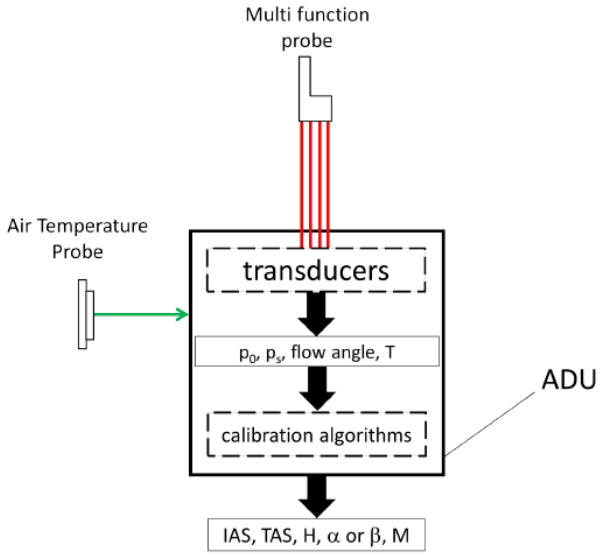


Figura 1.3: schema di un'architettura *multifunction probe*

La FADS (*Flush air data system*), invece, è un'architettura abbastanza innovativa, costituita da sensori diretti di pressione non intrusivi chiamati *flush ports*. L'ADC processa solo dati relativi alle pressioni e alle temperature e poi con particolari algoritmi di conversione riesce a tirare fuori tutti i dati necessari. Il vantaggio, in questo caso, è proprio la poca intrusività.

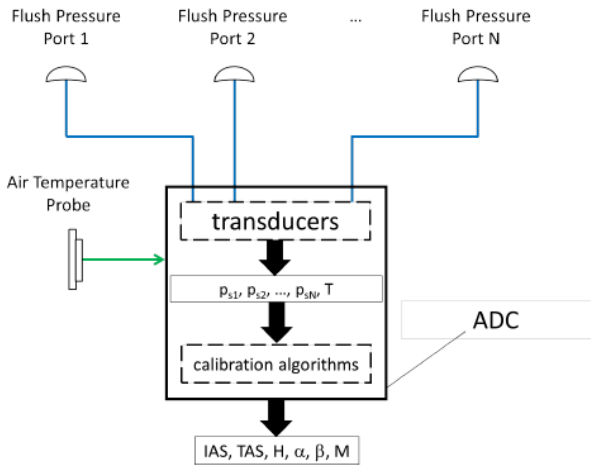


Figura 1.4: Schema di un'architettura FADS

Infine, si fa menzione di un'ulteriore configurazione dell'ADS molto utiliz-

zata negli ultraleggeri. Ci si riferisce alla *ADS Boom architecture*, costituita da un *Pitot Boom* cioè da un'asta per la misurazione dei dati sulla quale sono disposti dei tubi di Pitot. Al contrario, per un'architettura del genere il reale problema è costituito dall'intrusività.

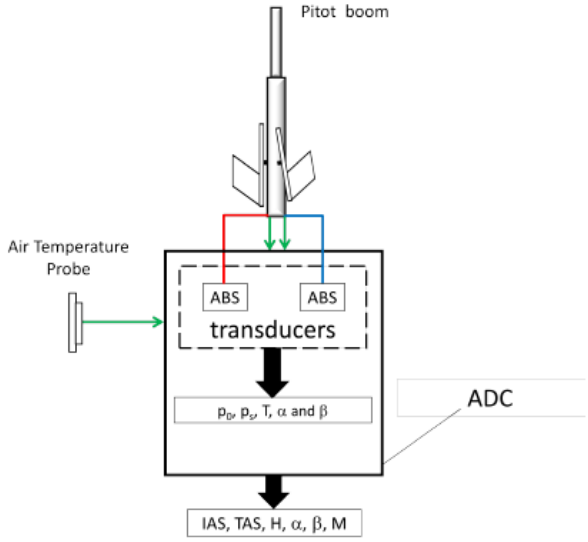


Figura 1.5: Schema di un'architettura *Boom*

Per tale ragione una configurazione più probabile per gli ultraleggeri può essere l'ADS convenzionale con due static flush port per stimare anche l'angolo di derapata all'infinito che, nei velivoli *unmanned*, è una grandezza da tenere in forte considerazione.

1.2 Calibrazione

Prima di entrare effettivamente in funzione l'ADS deve essere calibrato per correggere eventuali imprecisioni nelle misurazioni legate al posizionamento dei vari sensori sul velivolo, al numero di Mach e agli angoli d'assetto. A tal proposito vengono utilizzate delle leggi di calibrazione da applicare alle misurazioni locali dei sensori. Per correggere ogni grandezza misurata si scelgono dei fattori di compensazione:

- Per la p statica il fattore di compensazione tiene conto dell'errore di posizione, dell'errore legato al Mach e di quello legato agli angoli.

- Per la p_o invece l'errore di posizione è basso mentre invece è più alto quello legato al Mach e agli angoli di assetto soprattutto in transonico.
- Per la temperatura il fattore di compensazione va a correggere errori legati alla viscosità del fluido.

La stima del fattore di compensazione viene effettuata attraverso dei metodi di calibrazione, i quali possono essere diretti e indiretti. I metodi diretti, rispetto a quelli indiretti, stimano il fattore di compensazione utilizzando delle sonde per la misurazione diretta e sono di solito utilizzati per correggere prevalentemente errori di posizionamento.

I metodi indiretti sono 2:

- ***Tower Flyby***. La p viene calibrata rispetto ad una pressione standard a livello pista (a 100 piedi a velocità costante in condizioni atmosferiche standard). La funzione che lega l'altezza barometrica H alla pressione statica calibrata rispetto alla p_{st} in condizioni ISA è definita (una parabola). Nota la pressione statica calibrata dell'aereo in pista (QFE), sempre in condizioni ISA, l'altezza barometrica della pista rispetto al livello del mare $H_{rw,std}$ e quella dell'aereo $H_{A/C,std}$, è possibile entrare sfruttare la relazione per calcolare la pressione statica dell'aereo calibrata in condizioni ISA. Questo metodo è estremamente accurato.
- ***Radar tracking***. È un' estensione del tower flyby a velocità supersoniche. L'accuratezza è minore.

I metodi indiretti sono i seguenti:

- ***Flight test air data Boom***. Utile per calibrare ADS in presenza di manovre dinamiche. È molto utile per calibrare le misurazioni dell'ADS sugli angoli aerodinamici (è il solo metodo che permette di farlo).
- ***Trailing cone***. Un lungo tubo forato, di lunghezza variabile, posto posteriormente sul velivolo misura le condizioni del flusso libero lontano dall'aeroplano. È utilizzato per le correzioni sulla pressione statica e quindi completa il tower flyby per altitudini anche maggiori.
- ***Pacer aircraft***. Questo metodo coinvolge un ulteriore aeroplano con un sistema ADS accuratamente calibrato che serve come riferimento per calibrare il sistema del velivolo interessato. Entrambi gli aerei, durante la calibrazione, volano alla stessa altitudine e alla stessa velocità.

Capitolo 2

Sensori virtuali

Dal capitolo precedente risulta chiaro che un'alternativa all'ADS convenzionale veramente valida e soddisfacente per i velivoli MALE (*Medium Altitude Long Endurance*) non esista. La soluzione a cui si potrebbe ricorrere per inserire un sistema che possa fornire dei dati di supporto e confronto al sistema principale ADS su un velivolo ultraleggero è quella di mettere appunto dei sensori virtuali. Lo scopo, infatti, è quello di aumentare l'affidabilità, tenendo inalterati costi e ingombro che sono parametri di fondamentale importanza per un ultraleggero. Quanto detto spesso risulta estremamente complicato in quanto esiste un conflitto di fondo tra la necessità di mantenere i costi bassi (che è un aspetto importante per gli ultraleggeri), la necessità di progettare un aereo che sia conforme, in termini di peso, agli standard degli ultraleggeri, e la necessità di soddisfare le normative di sicurezza che impongono la ridondanza degli apparati sensoristici. Inoltre, ci sono altri requisiti in materia di aeronavigabilità che dovrebbero essere presi in considerazione come, ad esempio, quelli relativi al *bird strike*, che stabiliscono alcuni vincoli sulle installazioni della fusoliera. Infatti, il 60% degli impatti in volo con uccelli avviene nella parte frontale dell'aereo e i sensori disposti in quella zona sono più vulnerabili a possibili impatti in volo. Per tale ragione vanno distanziati accuratamente lungo la fusoliera del velivolo in modo da ridurre il rischio di impatto. Tuttavia, su un ultraleggero, per mancanza di spazio, è spesso difficile installare sonde e sensori ad un adeguata distanza in modo da essere conformi alla certificazione *bird strike*.

La tecnica che può essere utilizzata per superare questi problemi, introdotta per la prima volta negli anni 80, è quella che in letteratura viene indicata con il nome di 'ridondanza analitica' : per alcuni velivoli che pre-

sentano notevoli problemi di spazio e di peso, come gli ultraleggeri, sarebbe preferibile inserire dei sensori basati su tecniche di *soft computing*.

Sebbene questi discorsi fossero prematuri quando furono introdotti, suscitarono notevole interesse in ambito di ricerca tanto che molti studiosi si adoperarono per cercare delle soluzioni innovative che si fondassero sul concetto di 'ridondanza analitica'. Al giorno d'oggi, questi studi di ricerca continuano e il background aeronautico, grazie anche al notevole incremento della potenza dei moderni calcolatori, sembra pronto per accogliere un'innovazione così importante. Questo è ancor più vero se si considera che i più recenti studi in questo campo hanno condotto a dei risultanti molto promettenti mostrando che un grado di precisione molto vicino ai sistemi convenzionali potrebbe essere raggiunto.

Il vero elemento di novità introdotto dai sensori virtuali in ambito aeronautico è stata la possibilità di misurare indirettamente gli angoli aerodinamici a differenza dei sensori presenti attualmente sul mercato, per i quali la misurazione degli angoli aerodinamici coinvolge una serie di misurazioni di pressione provenienti da sonde specializzate disposte sulla fusoliera dell'aereo. Il sistema innovativo presentato in questo elaborato si propone di misurare nello specifico l'AoA in una modalità indiretta, servendosi cioè delle misurazioni provenienti dall'ADS (come la pressione dinamica), di quelle inerziali provenienti dall'AHRS e dei segnali di comando provenienti dal FCS. La tecnologia implementata per la progettazione del sensore può essere di tipo *model based* o di tipo *model learned*: la differenza e la scelta migliore tra le due filosofie sarà oggetto del paragrafo successivo.

2.1 Sensori *model based* e *model learned*

Esistono due tipologie di sensori virtuali:

- **Sensori *model based***, che sfruttano un modello matematico precostituito.
- **Sensori *model learned***, che sfruttano il *machine learning* trovando la soluzione più corretta tenendo memoria degli errori commessi.

È bene introdurre un breve esempio che permetta di comprendere la differenza tra le due metodologie di approccio: Aldo è un giovane adolescente che si è trasferito da poco in una nuova città che non conosce affatto. Dopo il

primo giorno di scuola sfortunatamente entrambi i genitori, a causa di alcuni imprevisti, non riescono a passare a scuola per riportare Aldo a casa, per tale motivo Aldo dovrà cercare di tornare da solo a casa in qualche modo. Il giovane studente ha due possibilità: la prima è quella di comprare una mappa della città con i pochi spiccioli che ha in tasca e di studiarla per imparare il percorso più breve che lo riporti a casa, la seconda è quella di procedere per tentativi finché non troverà la strada giusta.

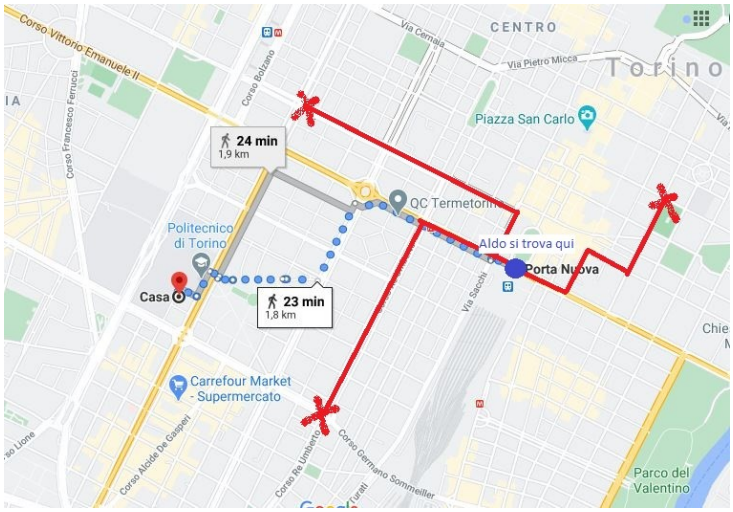


Figura 2.1: Esempio per comprendere la differenza tra logica *model based* e logica *model learned*

Questo esempio evidenzia perfettamente la differenza tra i due approcci: nel primo caso la mappa è sostitutiva del modello matematico (approccio *model based*), nel secondo caso la procedura per tentativi è quella che rispecchia la modalità di apprendimento delle reti neurali (approccio *model learned*). Nei paragrafi successivi si delineeranno le caratteristiche principali delle due tipologie di sensori per poi farne un raffronto, la cui finalità sarà quella di comprendere perché la tecnologia sensoristica *model learned* è ritenuta più adatta per un' applicazione aeronautica e, più in particolare, per il problema in esame in questo elaborato di tesi.

2.2 Sensori model based

I sensori *model based* sono anche conosciuti come osservatori di stato. Gli osservatori di stato sono sistemi adibiti alla misurazione di una grandezza

che non può essere misurata direttamente. Un sistema si dice misurabile solo se si è in grado di determinare gli input e gli output misurabili e da questi risalire ad una stima della grandezza da misurare, la quale può essere più o meno precisa (cioè più vicina al valore reale) a seconda del tipo di osservatore che decidiamo di utilizzare. Per un sensore *model based*, come indica il nome, ci si serve di un modello matematico del sistema in esame. Tale modello, per quanto approssimato, deve essere comunque abbastanza vicino al modello reale in modo da garantire una soluzione più o meno corretta. Dunque, tali sensori stimano la grandezza tramite un modello precostituito, cioè seguendo un pattern già pronto.

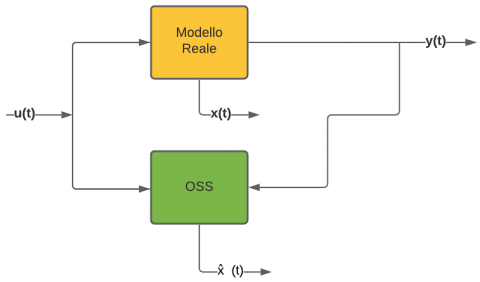


Figura 2.2: Schema di un osservatore di stato

Poiché il modello matematico è semplificato risulta ovvio che $x\,(t) \neq \hat{x}\,(t)$, dove $x\,(t)$ è la grandezza vera da misurare e $\hat{x}\,(t)$ è la stima di tale grandezza ottenuta dall'osservatore di stato. Allo scopo di di avvicinare il valore stimato a quello vero, l'osservatore di stato è dotato di un *closed loop* che elimina l'errore tra $y\,(t)$ e $\hat{y}\,(t)$ modulando l'input in ingresso al modello matematico grazie ad un controllore K.

Equazioni Modello reale:

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases} \tag{2.1}$$

Equazioni Modello con Osservatore:

$$\begin{cases} \dot{\hat{x}} = A\ \hat{x} + Bu + K\,(y - \hat{y}) \\ \hat{y} = C\ \hat{x} \end{cases} \tag{2.2}$$

Effettuando la differenza delle rispettive equazioni per i due sistemi si ottiene:

$$\dot{x} - \dot{\hat{x}} = Ax - A\,\hat{x} + Bu - Bu - K\,(y - \hat{y}) \tag{2.3}$$

$$y - \hat{y} = C\,(x - \hat{x})) \tag{2.4}$$

Da cui, combinando le due scritture, si ricava:

$$\dot{x} - \dot{\hat{x}} = A\,(x - \hat{x}) - KC\,(x - \hat{x}) = (A - KC)\,(x - \hat{x}) = (A - KC)\ e_{obs} \tag{2.5}$$

Da cui si nota che il controllore K agisce in modo da ridurre l'errore: tanto più $|A - KC| > 0$, tanto più e_{obs} tende a zero velocemente. Un esempio di sensore *model based* è il filtro di Kalman.

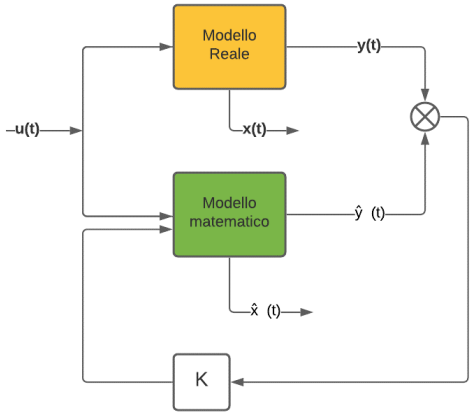


Figura 2.3: Schema di un osservatore di stato *model based*

2.3 Sensori model learned

Tali sensori stimano la grandezza con una logica che si basa sul metodo '*trial and error*': non è presente alcun modello precostituito che definisce un pattern preciso da seguire, ma si arriva alla soluzione con più tentativi imparando dagli errori commessi nei precedenti tentativi. Questo tipo di logica si basa dunque sul *machine learning* e sfrutta la capacità delle reti neurali di "apprendere con intelligenza" il miglior percorso da seguire. Tali

reti riescono a stimare la grandezza dopo un processo di apprendimento in cui sono note non solo $u\,(t)$(comandi di volo) e $y\,(t)$(grandezze osservabili), ma anche $x\,(t)$(grandezza da osservare). Dopo l'apprendimento la rete dovrebbe essere in grado di generalizzare e ricavare $\widehat{x(t)}$ che più si avvicina al valore reale per un qualsiasi set di u$(t)\,ey\,(t)$.

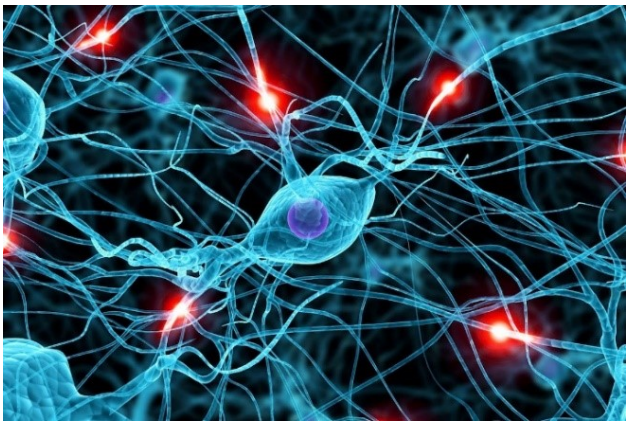


Figura 2.4: Visualizzazione di una rete neuronale biologica. Immagine tratta da `https://leganerd.com/2020/03/12/il-cervello-viene-plasmato-morfologicamente-dalle-esperienze/`

Le reti neurali si ispirano alle reti biologiche di neuroni. Lo sviluppo della tecnologia ha consentito di simulare un circuito neurale biologico dal punto di vita elettronico, meglio conosciuto come "elaboratore elettronico". Un elaboratore elettronico simula le sinapsi con cavi, resistenze e capacità mentre i neuroni postsinaptici sono accumulatori di segnali che liberano impulsi in uscita solo se il loro valore raggiunge una determinata soglia.

Attraverso tale sistema la rete neurale, servendosi di specifici algoritmi di apprendimento, può adattare i propri pesi agli input in ingresso simulando complicate relazioni input-output. Alcune volte, infatti, le relazioni tra grandezze in ingresso e di uscita risultano molto difficili da determinare analiticamente, dato l'elevato livello di complessità del sistema il quale potrebbe dipendere da molti parametri (questo è il caso di un aeroplano, dove le variabili che influenzano ogni fenomeno sono molteplici).

2.4 Model based VS Model learned

In questo paragrafo si cerca di confrontare le due metodologie mettendo su una bilancia i pro e i contro dell'una e dell'altra.

Per ciò che concerne il *model based*, la necessità di un modello matematico che sia affidabile e applicabile a tutto l'inviluppo di volo e l'inevitabile complessità che ne deriva, rende tale approccio poco adatto ad un'applicazione in *real time*. In più un altro aspetto negativo è legato all'elevata sensibilità ai disturbi esterni il che rappresenta un problema non trascurabile nel caso in cui si disponga di dati di volo con un notevole rumore di sottofondo (si veda paragrafo 5.3).Quando si parla di rumore di sottofondo si intende un disturbo dei dati causato da quattro principali fattori: raffiche di vento, turbolenza, rumore elettronico legato ai sensori, vibrazioni strutturali generate principalmente dal motore. Questo fenomeno genera un notevole peggioramento nei risultati se non trattato adeguatamente. Per tale ragione alla base del *model based* vi è una filosofia stocastica che tiene in considerazione di una possibile variabilità dei dati in input (proprio per ovviare al suddetto problema di sensibilità) e fornisce i risultati in termini probabilistici. L'approccio *model based* presenta anche alcuni vantaggi: innanzitutto non si perde mai di vista la fisica del problema che è alla base del modello matematico e in più i disturbi in ingresso sono facilmente modulabili.

Il *model learned*, invece, non ha bisogno di alcun modello matematico ma si serve delle tecniche di autoapprendimento neurali. Questa strada, tuttavia, porta alla perdita di informazioni del modello fisico. Per tale ragione occorre opportunamente addestrare la rete su tutto l'inviluppo di volo per far sì che vengano acquisite abbastanza informazioni sulla fisica del fenomeno e che la rete possa trarne le giuste conclusioni. Ci sono altri due svantaggi nell'applicazione di questo metodo: le reti neurali garantiscono risultati affidabili solo se vengono addestrate con dati sperimentali; in più, se non addestrate correttamente mediante tecniche mirate alla riduzione del problema di *overfitting*, le reti neurali possono essere molto sensibili ai disturbi esterni e ciò potrebbe condurre ad un significativo peggioramento dei risultati (il problema dell'*overfitting* sarà affrontato più nello specifico all'interno del paragrafo 4.4.4). Per il resto questa soluzione presenta numerosi aspetti positivi: il metodo è deterministico, ciò vuol dire che il risultato è univoco e non deve essere interpretato in termini probabilistici; le reti neurali sono particolarmente adatte ad applicazioni in tempo reale vista la velocità ele-

vata di calcolo (maggiore dei sensori *model based*). Se in più si aggiunge che, anno dopo anno i calcolatori sono sempre più potenti e consentono di effettuare una sempre maggiore mole di calcoli con performance molto elevate, si comprende perché l'interesse della ricerca tenda prepotentemente nella direzione dei sensori *model learned*.

	Model Based	*Model Learned*
Esempi	• Osservatori di stato (Filtro di Kalman)	• Reti neurali • Altre tecniche di machine learning
Vantaggi	• La fisica del problema è alla base dello sviluppo • Modellazione dei possibili disturbi	• Metodo deterministico • Adatto ad applicazioni in tempo reale
Svantaggi	• Necessità di avere un modello matematico del velivolo affidabile ed applicabile a tutto l'inviluppo di volo • Metodo stocastico (non deterministico) • Sensibilità ai disturbi esterni	• Perdita delle informazioni del modello fisico • Necessità di "insegnare" il problema fisico con dati di volo (simulati o sperimentali) su tutto l'inviluppo di volo • Sensibilità ai disturbi esterni • Solo i dati sperimentali garantiscono adeguate prestazioni durante la vita operativa.

Tabella 2.1: Differenze tra logica *model based* e *model learned*

2.5 Requisiti di progetto

I requisiti di prestazione relativamente alla misurazione di AoA e AoS non sono fissati a priori. Non esistono dei requisiti univoci che permettano di

stabilire entro quali limiti l'errore commesso è accettabile. Infatti, sebbene l'angolo di attacco debba conformarsi alla normativa AS403A, le prestazioni prescritte dalla stessa normativa sono riferite esclusivamente alla protezione dello stallo e non all'intera gamma di valori che l'angolo d'attacco assume durante il volo. Quando gli angoli aerodinamici in uscita dal sensore virtuale sono richiesti dall'ADS, i requisiti particolari che ne stabiliscono l'accettabilità sono fissati da altre funzionalità. Spetta ai responsabili del progetto stabilire i requisiti di precisione nell'ottica più ampia della funzione che il sensore sintetico dovrà assumere. Ad esempio, se il sensore è adibito a fornire dati alle leggi di controllo, l'imprecisione massima della misurazione su tali grandezze sarà stabilito in base alle prestazioni di navigazione desiderate.

Una strategia per fissare i requisiti dell'angolo di attacco può essere quella di stabilire la banda di tolleranza considerando tre contributi all'errore, i quali vanno sommati:

- Errore relativo alla banderuola per la misurazione corrente di AoA ($\pm 0.4°$)
- Errore relativo all'algoritmo di calibrazione ($\pm 0.3°$)
- Errore relativo all'istallazione ($\pm 0.3°$)

La banda di tolleranza totale raggiunge il valore di circa ($\pm 1°$) [2]. Valore che è destinato ad aumentare se si considera l'effetto del rumore di sottofondo (si veda paragrafo 5.3) il quale degrada la precisione dei risultati. La banda di tolleranza che si considererà in codesto elaborato è di ($\pm 2°$), proprio perché si ha a che fare con dati raccolti in ambiente reale corrotti dal rumore di sottofondo. Un altro aspetto che può influenzare la larghezza della fascia di accettabilità dei dati è relativo all'inviluppo di dati utilizzati per il *training* e il test della rete e dunque al tipo di manovra da cui si attingono i dati. Durante le manovre in condizioni normali di volo o in emergenza le performance richieste sono differenti se si considera un inviluppo di dati ristretto(LFE che sta per *Limited Flight Envelope*), un inviluppo di dati esteso (EFE che sta per *Extended Flight Envelope*) oppure un inviluppo di dati relativi a manovre stazionarie (SSFE che sta per *Steady-State flight Envelope*). Proprio durante questo ultimo tipo di manovre i requisiti sono più stringenti perché si prevede un peggioramento non indifferente delle prestazioni del sensore.

2.6 Vantaggi e svantaggi di un sensore virtuale *model learned*

In questa sezione si effettuerà un confronto tra la tecnologia proposta e i principali competitor in produzione, nell'ordine ottenere un bilancio tra vantaggi e svantaggi. I principali vantaggi di un sensore con rete neurale integrata sono classificabili, come già discusso, in termini di:

- Costo
- Peso
- Ingombro
- Affidabilità

Infatti, confrontando l'architettura di un ADS convenzionale con quella di un sensore virtuale *model learned* (figura 2.5) si nota, in quest'ultima, una notevole riduzione di complessità che è da attribuirsi alla presenza di un unico *Pitot static system* per la ricezione dei dati in ingresso oltre che ad una riduzione di cavi per trasferimento dati e del cablaggio per la distribuzione energia (si dimezza). È chiaro che una minore complessità si traduce in costi più contenuti, il che renderebbe un sensore di questo tipo molto appetibile sul mercato. Il risparmio in termini di ingombro è anch'esso in qualche modo legato allo snellimento dell'architettura perché il sensore non necessità più delle sonde per la misurazione diretta dell'angolo d'attacco. Codeste sonde, per il loro ingombro e per il loro peso, peggiorano l'efficienza aerodinamica del velivolo e riducono il valore del massimo carico utile rappresentando lo svantaggio più rilevante del sistema ADS. Inoltre, un'architettura più snella significa maggiore affidabilità in quanto il numero dei componenti si riduce di pari passo al numero degli interventi di manutenzione (il che porta anche un ulteriore risparmio dei costi). In aggiunta, un aspetto che può far pendere la scelta a favore dei sensori virtuali è che essi non necessitano di un ADC dedicata. L'ADC viene sostituita da un ADU, la quale presenta un costo più contenuto perché non ha alcuna capacità di calcolo e presenta solo un set di trasduttori.

Oltre ai vantaggi di tipo tecnico appena menzionati, un sensore virtuale porta notevoli benefici in termini ambientali garantendo una riduzione di potenza e di emissioni nocive. La principale difficoltà nella progettazione di un *virtual sensor* consiste nel mettere appunto dei modelli matematici o

degli algoritmi di apprendimento abbastanza efficaci e tali da poter ridurre al minimo l'errore sulla grandezza rispetto alla stima dei rispettivi sensori diretti.

Per contro, il determinante vantaggio degli ADS è che sono attualmente disponibili sul mercato e caratterizzati da una più che consolidata tecnologia.

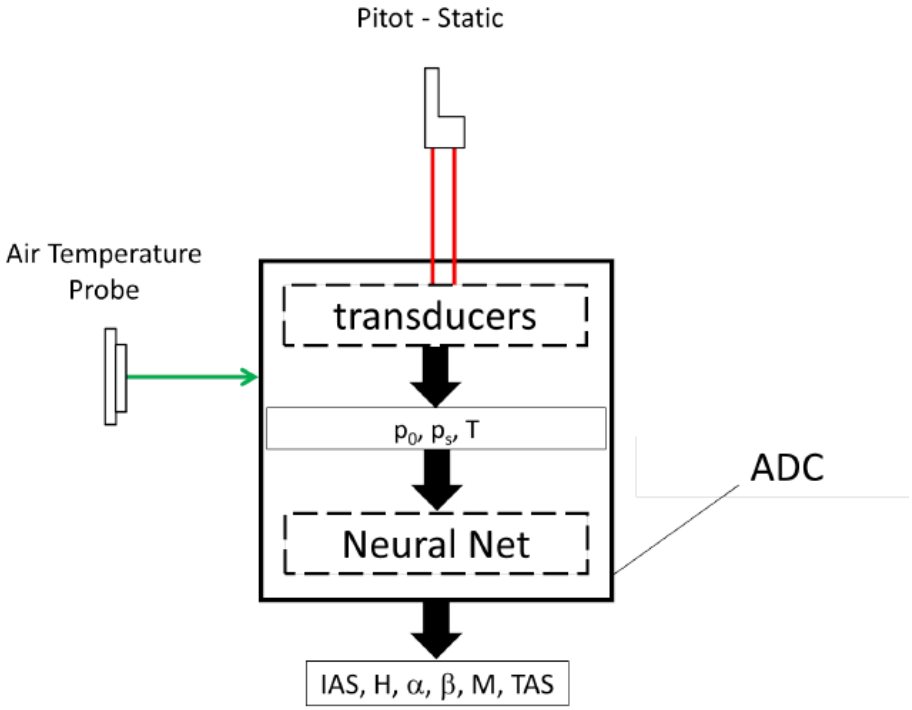


Figura 2.5: Architettura ADS integrata con rete neurale

Nella tabella 2.2 sono riassunti i cruciali vantaggi che la scelta di un sensore virtuale comporterebbe in ambito tecnico, economico e ambientale.

Nel complesso, per quanto riguarda la ridondanza analitica, un sensore virtuale, basato su reti neurali, è stato identificato come la migliore soluzione per ridurre la complessità dei sensori ADS ridondanti correnti.

	Benefici
Tecnici	• Sensori effettivi sostituiti con sensori virtuali: l'architettura del sistema è notevolmente semplificata • Riduzione cavi per trasferimento dati e cablaggio per distribuzione energia (almeno 50%), tubazioni pneumatiche • Riduzione del peso dal 50% all'80% (da 1,5 Kg a 3,6 Kg) • Aumentare l'affidabilità e la sicurezza complessive del sistema
Economici	• Minori costi di progettazione e integrazione • Costo di acquisto inferiore: risparmio di circa il 30% (da 6 K € a 35 K €) • Minori costi operativi: riduzione del 55% circa (0,3 Kg di carburante per ora di volo) • Le sonde esterne sono COTS (*Commercial of the shelf*)
Ambientali	• Riduzione di potenza dal 50% al 70% (principalmente per scopi di riscaldamento, da 0,2 KWh fino a 0,35 KWh) • Riduzione delle emissioni di CO2 dal 50% al 70% e NOx dal 50% al 70% • Meno parti di sistema in base alla progettazione e pezzi di ricambio da acquistare

Tabella 2.2: Principali benefici dell'introduzione di un sensore virtuale

Capitolo 3

Cenni teorici sulle reti neurali

Le reti neurali sono una rielaborazione elettronica dei circuiti neurali biologici. Ogni rete neurale presenta un certo numero di unità neuronali che apprendono sulla base della filosofia *model learned* e che sono in grado di applicare quanto appreso con una capacità di generalizzazione adattiva. Lo scopo dell'intelligenza artificiale è, dunque, quello di riprodurre il funzionamento della mente umana.

3.1 Storia

Nel 1943 Warren McCulloch e Walter Pitts realizzano una piccola rete neurale sfruttando circuiti elettrici: è la prima volta che le conoscenze sul funzionamento delle mente umana vengono utilizzate per produrre una struttura simile ad un circuito neuronale biologico. Il neurone artificiale proposto da Pitts e McCulloch è presentato nel ben noto lavoro *"A logical calculus of the ideas immanent in nervous activity"* : si tratta di una rete neurale embrionale che prende dati binari multipli in entrata per restituire un singolo dato binario in uscita.

Nel 1949, nel suo *" The Organization of Behaviour "*, Donal Hebb afferma che le connessioni tra neuroni sono tanto più forti quante più volte essi sono usati nello stesso momento. Questo studio mostra effettivamente come avviene il processo di apprendimento nella mente umana e propone alcuni modelli per spiegarne il suo funzionamento.

La sempre più approfondita conoscenza dei processi che governano il cervello umano apre nuove frontiere per l'applicazione degli stessi negli ambiti

più disparati. Combinatamente, lo sviluppo e il perfezionamento dei calcolatori negli anni 50 consentono di poter immaginare per la prima volta dei calcolatori che possano avvicinarsi alle strutture neurali biologiche, sfruttandone le capacità di apprendimento.

È proprio al 1950 che risale il primo tentativo (seppur fallito), ad opera di Nathanial Rochester, di simulare una rete neurale.

Nel 1955, McCarthy conia il termine 'intelligenza artificiale' riferendosi a tutti quei calcolatori che sono in grado di riprodurre il meccanismo di ragionamento e apprendimento della mente umana: da questo momento in poi l'interesse per questo settore cresce esponenzialmente.

Bisognerà attendere il 1959 per vedere il primo modello concreto di rete neurale ad opera di Bernard Widrow e Marcian Hoff della Stanford Univesity: i due mettono appunto dei modelli di nome "ADALINE" (*Adaptive Linear Elements*) e "MADALINE" (*Multilayered Adaptive Linear Elements*) che vengono direttamente applicati per risolvere problemi di eco sulle linee telefoniche.

Nello 1958 J. Von Neumann esprime le sue perplessità riguardo la bassa precisione dei metodi precedentemente proposti legata appunto alla scarsa potenza delle macchine di calcolo dell'epoca (*"The computer and the brain"*).

Nello stesso anno Rosemblatt, nel suo libro *"Psychological review"*, presenta il 'precettrone' che può essere considerato il vero è proprio padre delle reti neurali *feed-forward*. Il precettrone è un classificatore binario che, dato un vettore di dati in ingresso x, restituisce uno scalare $f(x)$.

$$f(x) = \begin{vmatrix} w_1 + b & \dots & w_n + b \end{vmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{3.1}$$

Dove i w_i sono i pesi del precettrone e b è il cosiddetto valore di 'bias'. Il 'bias' è un valore costante e indipendente dal vettore di input.

Il metodo di apprendimento messo appunto per il precettrone è il conosciuto algortimo di *back-propagation* o retro-propagazione dell'errore. La risposta della rete è confrontata con l'output desiderato generando un segnale d'errore. Tale segnale successivamente si propaga nella direzione inversa rispetto a quella delle connessioni sinaptiche, modificando i pesi sinaptici del precettrone concordemente con il tipo di addestramento scelto e fino a che non è soddisfatto il criterio di convergenza.

Il modello proposto da Rosenblatt suscita notevole interesse. Qualche anno dopo, tuttavia, non pochi studiosi dimostrano i limiti del percettrone come la sua incapacità di riconoscere e approssimare funzioni diverse da quelle linearmente separabili (la funzione XOR, ad esempio, non poteva essere approssimata con un precettrone).

Si capisce subito che il problema poteva essere risolto incrementando il numero dei precettroni in modo tale da creare una rete multistrato in cui i precettroni fossero interconnessi. Ma il discorso è troppo prematuro per quegli anni, e la potenza di calcolo ancora troppo bassa. Solo quando i calcolatori sarebbero diventati più performanti allora questa soluzione sarebbe ritornata prepotentemente in voga. Nel 1987 Hecht-Nielsen, Lippmann and Spreecher propongono l'utilizzo del teorema di Kolmogorov per l' approssimazione di funzioni multivariate. Tale intuizione ha dato un contributo notevole allo sviluppo delle reti neurali e è alla base del loro funzionamento: il teorema afferma che le funzioni continue di n varabili possono essere rappresentate come sovrapposizione di funzioni di una variabile:

'Esistono n costanti $w_i > 0$ *tali che* $\sum_{i=1}^{n} w_i \leq 1$ *e* $2n+1$ *funzioni continue strettamente crescenti* $\Phi_j \ [0,1] \to [0,1]$*, tali che ogni funzione continua f di n variabili su* $[0,1]^n$ *si può rapprentare nella forma:*

$$f\left(x_1, \ldots .., x_n\right) = \sum_{j=1}^{2n+1} \phi_j \ (\sum_{i=1}^{n} w_{ji} x_i); \tag{3.2}$$

per qualche $\phi_j \in C\left[0,1\right]$*, dipendente da* f*.'*

Le funzioni ϕ_j sono funzioni continue di una variabile dette anche funzioni di attivazione, mentre le funzioni w_{ji} sono funzioni continue monotone crescenti indipendenti da f. Girosi e Poggio manifestano la loro disapprovazione per il teorema di Kolmogorov, sostenendo che le funzioni di attivazione possano essere anche funzioni non monotone (come le funzioni radiali basiche).

Dopo vari anni di dibattiti riguardo l'affidabilità delle reti neurali multistrato per l'approssimazione di funzioni a più variabili, Cybenko, Hornik et al and Funahashi, dimostrano analiticamente che una rete *feed-forward* con 1 strato nascosto può approssimare una funzione a più variabili.

Dagli anni 80 in poi l'interesse per l'intelligenza artificiale e le reti neurali è cresciuto esponenzialmente anche grazie allo sviluppo di calcolatori più performanti. L'attenzione verso questo campo ha prodotto col tempo una vera

e propria rivoluzione nel modo di approcciarsi alla risoluzione di problemi complessi del mondo reale.

Nel 1994 Zadeh parla di *Soft Computing* riferendosi ad una modalità di approccio tipicamente umana basata sull'esperienza, che accoglie in sé la capacità di decidere, controllare, sbagliare e l'incertezza che caratterizza proprio l'essere umano. Secondo Lotfi Zadeh le comuni tecniche e modelli di analisi dei sistemi erano basati su un'accuratezza eccessiva che rendeva impossibile l'analisi di sistemi complessi del mondo reale. Zadeh sostiene l'importanza di introdurre dei metodi fondati sull'approssimazione e l'incertezza e afferma: *"una tendenza di visibilità crescente è costituita dall'uso della logica fuzzy in combinazione con il calcolo neurale e gli algoritmi genetici. Più in generale, la logica fuzzy, le reti neurali e gli algoritmi genetici possono considerarsi i principali costituenti di ciò che potrebbe essere definito calcolo soft. A differenza dei metodi di calcolo tradizionali o hard, il soft computing si prefigge lo scopo di adattarsi alla pervasiva imprecisione del mondo reale. Il suo principio guida può così esprimersi: sfruttare la tolleranza per l'imprecisione, l'incertezza e le verità parziali in modo da ottenere trattabilità, robustezza e soluzioni a basso costo. Nei prossimi anni, il soft computing è probabilmente destinato a giocare un ruolo sempre più rilevante nella concezione e progettazione di sistemi il cui MIQ (Quoziente Intellettivo di Macchina) sia di gran lunga più alto di quello dei sistemi convenzionali. Tra le varie combinazioni di metodologie di soft computing, quella avente maggiore visibilità in questo frangente è la fusione di logica fuzzy e calcolo neurale, che conduce ai cosiddetti sistemi neuro-fuzzy. Nel contesto della logica fuzzy, tali sistemi rivestono un ruolo particolarmente importante nel processo d'induzione delle regole a partire dall'osservazione"* [3].

Secondo Kohonen, peraltro, può sostenersi che: *"soft computing, real-world computing ecc. sono denominazioni comuni per certe forme di elaborazione naturale di informazione che hanno la loro origine in biologia. La logica fuzzy e probabilistica, le reti neurali, gli algoritmi genetici, d'altra parte, sono formalismi teorici alternativi mediante i quali si possono definire schemi computazionali e algoritmi per questi scopi"* [4].

Il *Soft Computing* si basa su tre branche fondamentali, di ispirazione 'umana':

- ***Artificial neural networks*** (ANN) o ***Simply neural network*** (NN), ispirate ai circuiti neurali biologici e in grado di riprodurre la

capacità di scelta data dall'esperienza del cervello umano servendosi di calcoli matriciali.

- ***Algoritmi genetici*** (GA), basati sui processi di evoluzione e mutazione della specie umana.
- ***Fuzzy logic***, (tradotto vuol dire 'logica sfumata') che rappresenta una logica polivalente, estensione della logica booleana. A tal proposito sempre Zadeh afferma nel 1994 : *«Il termine logica fuzzy viene in realtà usato in due significati diversi. In senso stretto è un sistema logico, estensione della logica a valori multipli, che dovrebbe servire come logica del ragionamento approssimato. Ma in senso più ampio logica fuzzy è più o meno sinonimo di teoria degli insiemi fuzzy cioè una teoria di classi con contorni indistinti. Ciò che è importante riconoscere è che oggi il termine logica fuzzy è usato principalmente in questo significato più vasto».*

Al giorno d'oggi l'intelligenza artificiale ha applicazione nei più disparati campi della tecnologia raggiungendo risultati prima impensabili. Tuttavia, le performance della mente umana sono ancora lontanissime.

3.2 La mente umana

Il sistema nervoso umano è costituito da tre componenti fondamentali: i recettori che convertono gli input provenienti dall'ambiente esterno o dal corpo umano in segnali elettrici, il cervello che elabora questi segnali elettrici e ne produce degli altri come risposta agli impulsi ricevuti e infine gli effettori che convertono gli impulsi elettrici in azioni effettive del nostro corpo.

Il cervello umano può essere considerato come un enorme rete neurale in grado di elaborare milioni di informazioni in tempi brevissimi e performance elevatissime. Le notevoli capacità processive ed elaborative del cervello sono da ricercarsi non tanto nella presenza massiva di neuroni all'interno del cervello, quanto nella rete di interconnessioni presenti tra le unità neuronali del cervello stesso. Infatti, i neuroni del cervello sono circa sei volte più lenti delle unità elaborative di un chip in silicio. Per fare qualche numero, gli eventi in un chip di silicio avvengono ogni $10^{-9}s$ mentre gli eventi neurali avvengono ogni $10^{-3}s$; nella corteccia ci sono approssimativamente 10 bilioni di neuroni e 60 trilioni di sinapsi. Per tale ragione l'efficienza energetica del cervello è altissima (circa $10^{-16}J$) [1].

Più nello specifico ogni circuito neuronale biologico è costituito da un neurone presinaptico e un neurone postsinaptico che combinatamente riescono a generare l'impulso elettrico da trasmettere poi ad altri neuroni. Il neurone presinaptico genera un impulso elettrico iniziale quando sollecitato da un segnale elettrico in ingresso. L'impulso generato dal neurone presinaptico passa in seguito attraverso una sinapsi che ha la capacità di 'graduare' l'eccitazione del neurone postsinaptico, regolando in questo modo il segnale in uscita. Il peso sinaptico è fondamentale per gestire l'output del neurone. Durante l'apprendimento il peso sinaptico viene costantemente modificato per poi essere conservato in memoria fino al momento dell'uso.

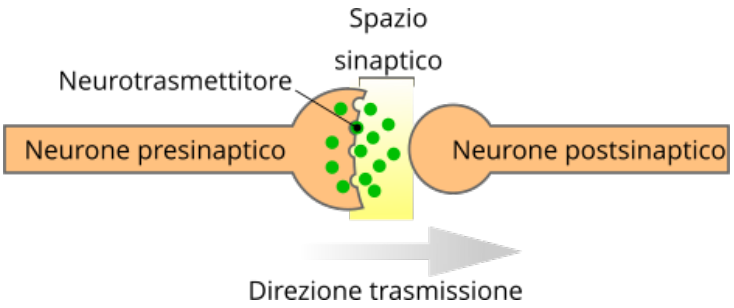


Figura 3.1: Schema di un circuito neuronale biologico. Immagine tratta da `https://biologiawiki.it/media/showimage/447-neurone-presinaptico-e-postsinaptico-nella-sinapsi-chimica/`. Autore: Crisafulli.

I neuroni artificiali utilizzati nelle reti neurali sono molto meno complessi e più primitivi dei neuroni biologici; nonostante tutto, ciò che fa ben sperare sono gli enormi progressi che l'intelligenza artificiale sta compiendo anno dopo anno.

3.3 Costruzione di una rete neurale

3.3.1 Modello di un neurone

Di seguito si presenta lo schema generico alla base di un modello neurale. Il classico modello di neurone artificiale si rifà al modello di precettrone proposto da Rosenblatt combinato ad una funzione di attivazione, come suggerito dal teorema di Kolmogorov. Il modello è composto da tre elementi fondamentali:

- Un insieme di sinapsi o collegamenti, ognuno dei quali è caratterizzato da un da un peso sinaptico specifico. In particolare, il peso w_{ji} connette il segnale di input i-esimo e il neurone j-esimo. Dunque, il primo indice si riferisce al neurone mentre il secondo indice si riferisce al segnale di input. Ovviamente nel caso in cui la struttura fosse costituita da un singolo neurone, il primo indice verrebbe meno.
- Un operatore somma che funge da combinatore lineare sommando i prodotti tra i vari input e i rispettivi pesi sinaptici che confluiscono nel neurone j-esimo
- Una funzione di attivazione Φ, il cui scopo è quello di filtrare o limitare l'ampiezza dell'output di un neurone.
- Un bias, la cui funzione è quella di ridurre o aumentare l'input della rete della funzione di attivazione. Di solito ha la funzione di limitare i problemi di generalizzazione in fase di test legati alla corruzione dei dati per effetto del rumore di sottofondo.

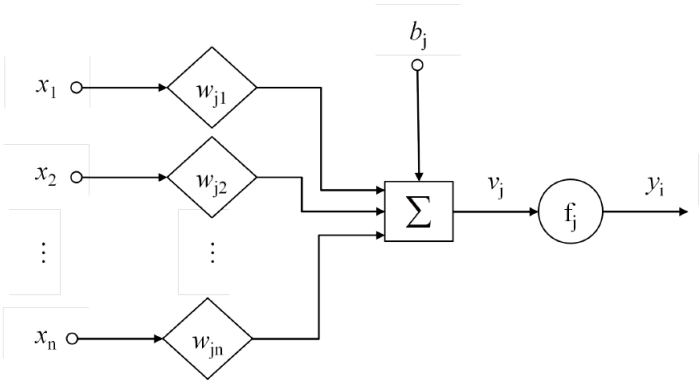


Figura 3.2: schema del segnale di uscita del neurone j-esimo

L'equazione dalla quale si ottiene l'output del modello neuronale è la seguente:

$$f(x_1, \ldots .., x_n) = y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{3.3}$$

Il modello proposto è rappresentato nella figura 3.2: l'output del j-esimo neurone si ottiene prima combinando linearmente i valori di input pesati, poi

aggiungendo un bias, quindi applicando una funzione di attivazione. Tale modello è non lineare, se la funzione di attivazione è non lineare. Per ottenere delle reti neurali efficienti che riescano ad approssimare nel migliore dei modi la funzione di interesse è fondamentale:

- scegliere opportunamente la funzione di attivazione.
- mettere appunto un efficace algoritmo di *learning* che modifichi iterativamente i pesi sinaptici in modo che convergano nel più breve tempo possibile.
- scegliere opportunamente il bias della rete

3.3.2 Funzioni di attivazione

Le funzioni di attivazione, come già detto, sono importanti da un lato per limitare gli output dei singoli neuroni, dall'altro perché, se scelte opportunamente, riescono a conferire alla rete una capacità di generalizzazione e di estrapolazione oltre i limiti dei *training patterns*. Le funzioni di attivazione più comuni sono le seguenti:

1. **Lineare.**

$$\phi(v_j) = v_j \tag{3.4}$$

   Tale funzione conferisce alla rete notevole capacità di estrapolazione ma tende a far divergere l'output. Per tale ragione spesso non è utilizzata. Si noti che v_j rappresenta la somma pesata degli input (detta anche *weighted input*) in ingresso al neurone j-esimo.

2. **Sigmoidale**.

$$\phi(v_j) = (1 + exp(-av_j))^{-1} \tag{3.5}$$

   Tale funzione ha la caratteristica di essere una funzione continua e derivabile, che ha una derivata prima non negativa e dotata di un minimo locale ed un massimo locale. Proprio perché è limitata permette all'output della rete di non divergere, anche se non consente di mantenere le capacità di estrapolazione che si ottengono con la funzione lineare. Variando il parametro a si ottengono delle funzioni sigmoidee di inclinazione variabile.

3. **Log-sigmoidale**.

$$\phi(v_j) = log(1 + exp(-v_j))^{-1} \tag{3.6}$$

Risulta essere poco performante per l'approssimazione di funzioni.

4. **Hard-limiter**.

$$\phi(v_j) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \end{cases} \tag{3.7}$$

Tae funzione ha il principale difetto di non essere differenziabile in tutti i punti.

5. **Tangente-iperbolica**.

$$\phi(v_j) = \tanh(v_j) \tag{3.8}$$

6. **Radiale basica.** Esistono vari tipi di funzioni radiali basiche; la più comune è la funzione gaussiana

$$\phi(r_j) = e^{(-\epsilon r_j)^2} \tag{3.9}$$

con $r_j = ||x - c_j||$, c_j centro della j-esima funzione radiale della rete. Le funzioni radiali basiche saranno analizzate più dettagliatamente nel capitolo successivo.

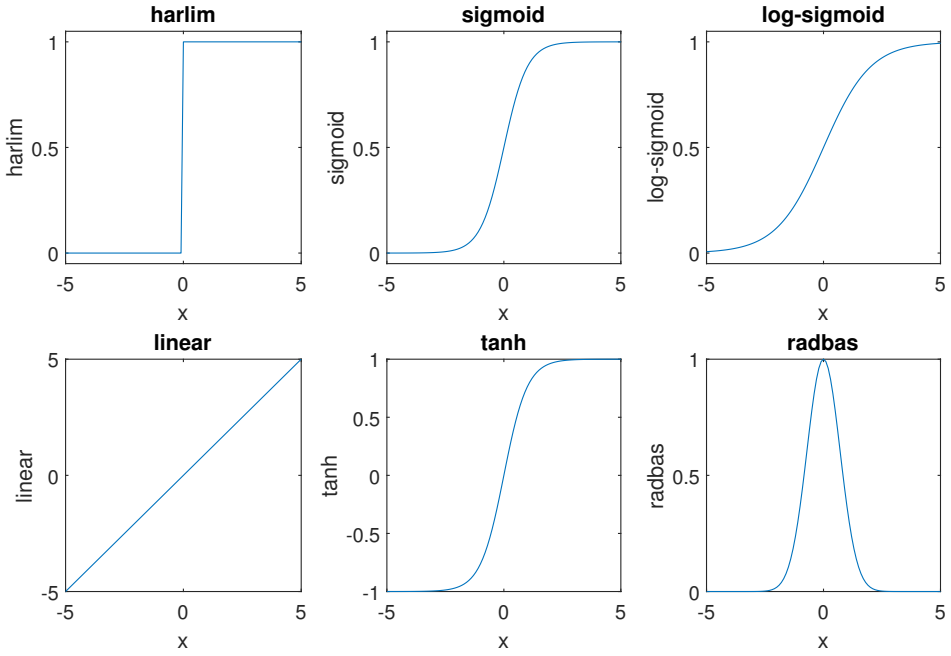


Figura 3.3: Principali tipologie di funzioni di attivazione

3.3.3 Tipi di architetture di reti neurali

In questo paragrafo si analizzano i principali tipi di architetture di reti neurali. Tutte le architetture comunemente usate riprendono più unità neuronali elementari (presentate nel paragrafo 3.3.1) e le combinano in modo tale da ottenere uno o più strati di neuroni. Esistono due tipologie principali di architetture:

1. Reti neurali *feed-forward*.
2. Reti neurali ricorrenti (*recurrent networks*).

Le reti feed-forward sono delle reti acicliche nel senso che non c'è alcun ciclo di feedback della rete. All'interno di questa categoria si distinguono due sottoclassi:

- Le *single layer feedforward preceptron networks*, dette anche SLP, che sono caratterizzate sa uno strato di neuroni di input a cui non è affidato alcun compito computazionale ma ha l'unica funzionalità di recepire i dati di input. L'unico strato che ha la funzione di calcolatore è lo strato

di uscita (da ciò deriva il nome *single layer*). L'output l-esimo della rete si calcola come segue:

$$f_l\left(x_1, \ldots .., x_n\right) = y_l = \sum_{j=1}^{n} \phi_j \left(\sum_{i=1}^{n} w_{ji} x_i + b\right) \tag{3.10}$$

Si noti che il bias è stato scelto costante per tutti i neuroni della rete.

- Le *multi layer feedforward preceprton networks*, dette anche MLP, invece sono caratterizzate da uno o più strati nascosti che processano i calcoli ed elaborano le informazioni. Questi strati sono cioè il 'cervello' della rete. Inoltre sono anche presenti uno strato di input che ha la sola funzione di 'immagazzinamento' cioè di trattare i dati di input per renderli compatibili con i neuroni dello strato nascosto ; uno strato di output la cui funzione è quella di convertire i dati di output in modo da renderli confrontabili (durante il training) con gli output desiderati. Una MLPs può essere totalmente connessa se tutti i neuroni di ogni strato sono connessi con i neuroni degli strati immediatamente contigui; se questo non accade si dice parzialmente connessa.

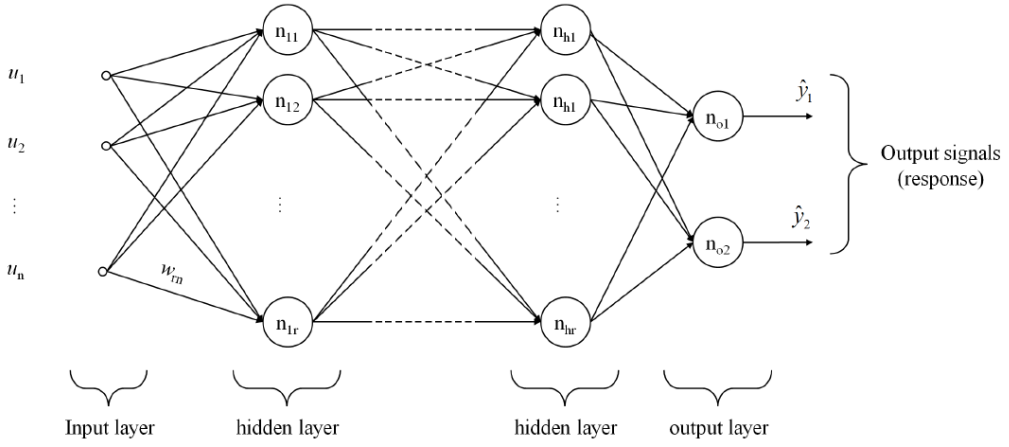


Figura 3.4: Schema di una MLP generica

L'output l-esimo di una rete MLP a 2 strati nascosti si ottiene come combinazione lineare degli output dei neuroni relativi al secondo strato nascosto

ognuno dei quali è dato dal prodotto della rispettiva funzione di attivazione per la combinazione lineare degli output dello strato precedente:

$$f_l(x_1, \ldots, x_n) = y_l = (\sum_{p=1}^{n} \phi_{2p} \, (\sum_{j=1}^{n} \phi_{1j} \, (\sum_{i=1}^{n} w_{ji} x_i + b))) \tag{3.11}$$

Una MLP può essere interpretata come una mappatura non lineare tra l'input e il target.

Le reti neurali ricorrenti sono delle particolari reti che hanno la capacità di contestualizzare le informazioni in ingresso alla rete utilizzando la memoria interna per processare una sequenza di input. I dati in ingresso, dunque non sono trattati indipendentemente l'uno dall'altro ma sono inseriti in un contesto più generale che consente alla rete di predire un risultato relazionando gli input tra loro. Questo tipo di apprendimento è particolarmente utilizzato per analisi predittive su sequenze di dati come, ad esempio, il riconoscimento della grafia o quello vocale. Si immagini di dover predire una parola all'interno di una frase: è chiaro che per farlo c'è bisogno che la rete ricordi tutte le relazioni tra le precedenti parole e sulla base di queste riesca a prevedere la successiva parola nel modo migliore. La figura 3.5 schematizza quanto detto: l'output della rete all'istante corrente è il risultato di un'auto addestramento della rete sulla base di una sequenza di test agli istanti precedenti. Per meglio intendersi, l'input della rete all'istante t è costituito non solo da x_t ma anche da x_{t-1} e O_{t-1}. La rete varia il suo comportamento in modo dinamico sulla base del contesto in cui è inserita.

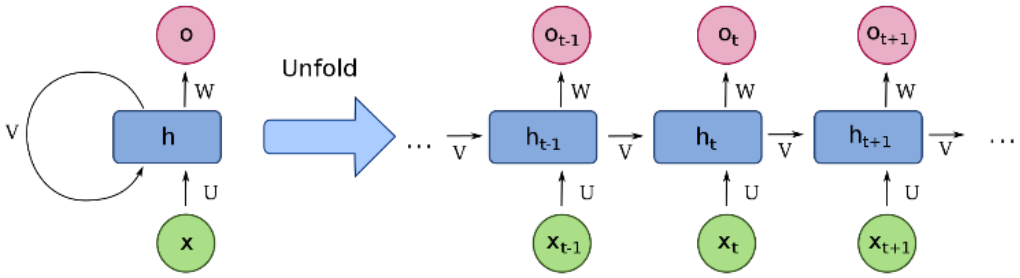


Figura 3.5: Schema di una rete neurale ricorrente. Immagine tratta da `https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg`. Autore: fdeloche

Un' altra classificazione che di solito si utilizza per le reti neurali si basa sul numero di dati in ingresso e in uscita. Distinguiamo:

- Le reti SISO (*single input- single output*).
- Le reti MISO (*multi input- single output*).
- Le reti MIMO (*multi input- multi output*).

3.4 Apprendimento di una rete neurale

Una delle filosofie di apprendimento più utilizzate per 'allenare' la rete è quella che si basa sulla valutazione dell'errore locale che commette la rete istante per istante nell'approssimare il valore di target. Dunque, il segnale di output stimato $\widehat{f_j}(\bar{x})$ relativo al neurone j-esimo è confrontato con l'output desiderato $f_j(\bar{x})$ della rete relativo proprio a quel neurone j-simo, in modo da ottenere il valore dell'errore j-esimo della rete quando è sollecitata dal vettore di input $\bar{x}$ valutato in un certo instante di tempo t.

$$e_j(t) = f_j(\bar{x}) - \widehat{f_j}(\bar{x}) \tag{3.12}$$

In questo modo si instaura un meccanismo di controllo della soluzione che passa, appunto, per la valutazione dell'errore locale. Lo scopo è quello di minimizzare l'errore in modo che la soluzione approssimata possa convergere verso la soluzione reale. In realtà la minimizzazione dell'errore avviene attraverso minimizzazione della funzione di costo detta anche funzione energia dell'errore. Nel caso più semplice di modello lineare,

$$f(x) = \sum_{j=1}^{n} w_j \phi_j(x,t) \tag{3.13}$$

la funzione di costo può essere espressa come segue:

$$\mathcal{E}(t) = \frac{1}{2}\sum_{j=1}^{n} e_j^2(t) \tag{3.14}$$

Si noti che di solito, in casi più complessi si aggiunge un ulteriore termine che è legato alle tecniche di regolarizzazione della rete. L'espressione 3.14 si semplifica se la rete ha un solo neurone:

$$\mathcal{E}(t) = \frac{1}{2} e_j^2(t) \tag{3.15}$$

Nel caso di un apprendimento off-line (*batch learning*) si valutano gli errori relativi a più istanti di tempo e la funzione di costo assume la seguente forma:

$$\mathcal{E} = \frac{1}{2p} \sum_{t=1}^{p} \sum_{j=1}^{n} e_j^2(t) \tag{3.16}$$

con p numero degli istanti temporali. In generale per problemi di regressione le possibili funzioni di costo utilizzate sono 3:

1. L'errore quadratico medio (MSE)

$$\mathcal{E} = \frac{1}{2p} \sum_{t=1}^{p} e^2(t) \tag{3.17}$$

2. La radice dell'errore quadratico medio (RMSE).

$$\mathcal{E} = \sqrt{\frac{1}{2p} \sum_{t=1}^{n} e^2(t)} \tag{3.18}$$

3. L'errore assoluto medio (MAE)

$$\mathcal{E} = \frac{1}{2p} \sum_{t=1}^{n} |e(t)| \tag{3.19}$$

In questo lavoro si è scelto di utilizzare come funzione di costo il MSE. Si noti inoltre la presenza di un fattore 1/2 all'interno di tutte le funzioni di costo. I pesi sinaptici vengono aggiornati iterativamente Δw_{ji} di un in modo da rendere minima tale funzione.

$$\Delta w_{ji}(t) = -\eta \frac{\partial \mathcal{E}}{\partial w_{ji}} \tag{3.20}$$

La variazione istantanea del peso sinaptico dipende dal tipo di algoritmo scelto per l'apprendimento. Tale valore è la variazione del peso sinaptico

relativo al neurone k-simo eccitato dall'input j-esimo all'istante t, dove η è una costante. Il peso è aggiornato come segue all'istante successivo.

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t) \tag{3.21}$$

L'aggiornamento dei pesi, e quindi il training della rete si conclude quando la rete arriva ad una condizione di stabilità e cioè quando la funzione di costo non diminuisce più perché si è raggiunto un valore di minimo. Tale metodo è anche conosciuto come *Delta-Rule*.

La caratteristica fondamentale di questo metodo è appunto la sua 'località' nel senso che l'errore è relativo ad un neurone specifico della rete e tutte le correzioni dei pesi sinaptici avvengono attorno a quel neurone.

3.4.1 Apprendimento supervisionato e non supervisionato

L'**apprendimento supervisionato** è una tecnica di apprendimento automatico il cui scopo è quello di addestrare una rete neurale fornendo alla rete stessa una serie di esempi che costituiscono l'output desiderato della rete in corrispondenza di un determinato set di input. In questo caso quindi la rete deve semplicemente imparare a riconoscere la relazione tra le coppie input-output fornite in fase di training. È come se la rete fosse guidata e controllata da un insegnante che supervisiona i suoi progressi e la guida nell'apprendimento.

Il set di *training* è costituito, dunque, da un vettore di input $x(t)$ ed un valore di output $f(x(t))$.

$$\mathcal{T} = (x(t), f(x(t)) \tag{3.22}$$

Il valore dell'output di cui si dispone di solito non è quello reale, ma è corrotto da un rumore di sottofondo. Per calcolare il valore reale dell'output da inserire nella rete per addestrarla bisognerebbe modellizzare il rumore utilizzando delle tecniche statistiche, il che spesso risulta estremamente complicato. L'errore che si commette considerando i dati di input corrotti dal rumore è trascurabile; al contrario l'errore commesso sull'output stimato dalla rete (considerando dei dati di input corrotti) cresce non di poco.

In tale approccio si utilizza una funzione di verosimiglianza perché la rete comprenda al meglio la relazione input- output. In pratica, l'apprendimento

supervisionato modifica i pesi della rete utilizzando come informazioni il vettore di training e l'errore commesso in ogni istante e inserendo quest' ultimo in un ciclo di feedback. La procedura è quella descritta nel paragrafo precedente : si immagini una funzione di costo multidimensionale che deve essere modificata andando iterativamente ad aggiornare i pesi. In un apprendimento supervisionato i pesi sono aggiornati utilizzando l'utile informazione del gradiente della funzione di costo.

La funzione di costo può essere minimizzata attraverso vari metodi che saranno discussi successivamente. I principali metodi sono: il metodo dei minimi quadrati (LS) lineare e non, il metodo dei minimi quadrati ortogonale (OLS) e ancora, per le MLP, l'algoritmo di *backpropagation* e i *descent methods*. In questo capitolo ci limiteremo ad accennare il funzionamento dei suddetti algoritmi delle MLP. Il LS e l'OLS saranno analizzati nel capitolo successivo, come principali metodi di ottimizzazione per le *RBF Neural Networks*.

L'**apprendimento non supervisionato**, invece, è una tecnica di apprendimento automatico per il quale non si hanno delle coppie di input-output ma di solito si ha un set di input che la rete deve imparare, attraverso varie tecniche, a auto classificare in apposite categorie sulla base di caratteristiche comuni in modo da effettuare delle previsioni sugli input futuri. La differenza con l'apprendimento supervisionato consiste nel fatto che non c'è legame tra input e output mostrati alla rete in fase di *training*. Gli output che si forniscono alla rete durante l'addestramento fanno parte dell'esperienza che forniamo alla rete stessa: sta alla rete comprendere a quale classe quell'output appartiene.

Un esempio tipico di apprendimento non supervisionato consiste nei motori di ricerca, i quali sono programmati per fornire una serie di link che l'algoritmo ritiene attinenti alla ricerca effettuata. L'algoritmo ha dunque il compito di classificare la ricerca e comprendere quali link presentare in uscita.

Questi algoritmi lavorano, in sintesi, effettuando continui confronti tra i dati a disposizione così da comprendere i nessi. Perciò sono più efficaci con dati numerici o con dati che comunque posseggono un ordinamento preciso e a cui è possibile applicare delle tecniche statistiche per la classificazione. I Principali algoritmi di apprendimento non supervisionato sono:

- Il *clustering*.

- Le regole di associazione.

In particolare, uno degli algoritmi più conosciuti è il *k-means clustering*: Gli algoritmi di classificazione, come il *k-means*, creano dei pattern prestabiliti in base alla classe a cui appartengono gli input. Molto spesso i dati di input non sono facilmente classificabili nell'iperspazio perché non sono linearmente separabili, per questo si utilizza il 'trucco del kernel' che consiste nell'applicare ai dati di input una funzione non lineare in modo da renderli linearmente separabili e, dunque, facilmente classificabili. Allora la rete riconosce i pattern da assegnare a quei dati di training e classifica il tipo di problema adattando l'output proprio al tipo di problema individuato.

3.4.2 Apprendimento *batch* e on-line

Un'altra classificazione sulla tipologia di apprendimento di solito dipende dalla capacità della rete di apprendere in modo incrementale.

Per meglio intendersi si dice che una rete apprende online se l'apprendimento avviene in modo incrementale, cioè se ad ogni istante di tempo la rete apprende. In questo modo il comportamento della rete è mutabile e dinamico e si adatta agli esempi che istante per istante le forniamo. Inoltre, tale metodo è anche computazionalmente molto vantaggioso perché non c'è bisogno di tenere in memoria tutti i pattern di *training* ma, una volta appreso un singolo pattern, esso risulterà successivamente inutile e potrà essere scartato.

Si dice che la rete apprende in *batch learning* (off-line) se, al contrario, forniamo alla rete tutti gli esempi che ha bisogno per essere addestrata nello stesso momento. Dal punto di vista computazionale ciò è molto più dispendioso perché devo allocare memoria per tutti i *training patterns*. Inoltre, la rete, una volta concluso l'addestramento risulta immutabile (il che può anche essere un vantaggio alcune volte, se si vuole evitare di incorrere in comportamenti poco prevedibili).

3.4.3 Algoritmi di apprendimento per le MLP

In questo paragrafo si accennerà ai principali metodi di apprendimento per le MLP. I metodi a cui si fa riferimento sono tutti supervisionati in quanto partono tutti dalla valutazione della funzione di costo che, presuppone la conoscenza di coppie input-output durante l'allenamento della rete. Le reti MLP sono delle reti *feed-forward* (si veda il paragrafo 3.3.3) che aggiornano

i pesi sulla base del gradiente della funzione di costo. L'aggiornamento dei pesi avviene limitatamente all'input e l'output in un determinato istante, i quali influenzano il gradiente della funzione di costo. Per tale ragione queste reti sono classificate come *feed-forward*.

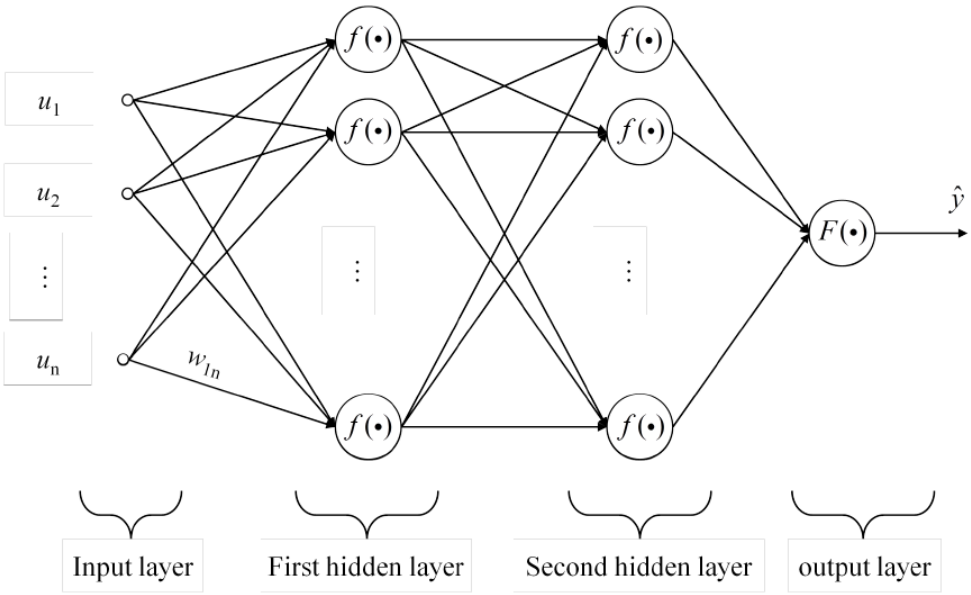


Figura 3.6: Schema di una MLP a 2 strati nascosti

Si preferisce generalizzare l'espressione relativa all'output del neurone j-esimo supponendo che faccia parte di una rete a più strati nascosti e che quindi riceva in ingresso l'output del neurone precedente. Nella formula 3.23, l'input al neurone (prima indicato con x_i) sarà sostituito con y_i.

$$f_j\left(x_1, \ldots .., x_n\right) = y_j = \phi_j\left(\sum_{i=1}^{n} w_{ji} y_i + b\right) \tag{3.23}$$

Gli algoritmi di apprendimento per le MLP sono:

- Algoritmo di retropropagazione (*backpropagation*)
- Metodi 'di discesa' (*descent methods* o *gradient based methods*), tra i quali possiamo distinguere:

    1. *Steepest Descent method*
    2. *Newton's Method*

- Algoritmo di Levenberg-Marquardt

L'algoritmo di *backpropagation*, in modo simile all'algoritmo LMS (*Least Mean Square*), applica una correzione ai pesi sinaptici della rete che è direttamente proporzionale alla derivata della funzione di costo della rete (relativa all'istante di tempo t) rispetto al peso sinaptico considerato, secondo la *Delta Rule*.

$$\Delta w_{ji}(t) = \eta \frac{\partial \mathcal{E}(t)}{\partial w_{ji}} \tag{3.24}$$

La regola del Delta dimostra che la suddetta derivata è uguale al prodotto di una funzione $\delta_j(t)$ detta gradiente locale relativa neurone j-esimo e l'input i-esimo allo stesso neurone $y_i(t)$.

$$\Delta w_{ji}(t) = -\eta \frac{\partial \mathcal{E}(t)}{\partial w_{ji}} = \eta \delta_j(t) y_i(t) \tag{3.25}$$

Il gradiente locale assume due valori diversi a seconda che il neurone j-esimo sia un neurone nascosto oppure un neurone di output

$$\begin{cases} \delta_j(t) = e_j(t)\,\phi'_j(v_j(t)) & \textit{neurone output} \\ \delta_j(t) = \phi'_j(v_j(t)) \sum_k \delta_k(t)\, w_{kj}(t) & \textit{neurone hidden} \end{cases} \tag{3.26}$$

Nella seconda espressione l'indice k si riferisce ad un neurone output, per evitare di fare confusione con l'indice j che invece si riferisce al neurone *hidden*.

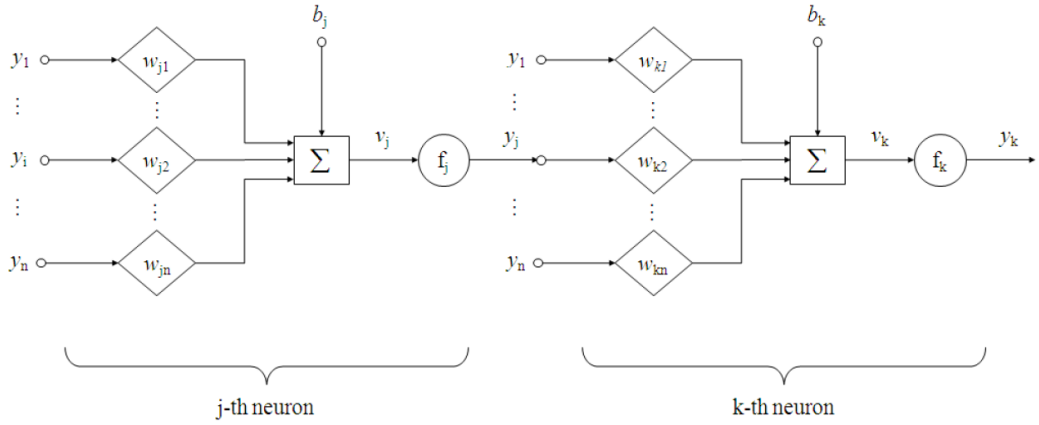


Figura 3.7: Schema di due neuroni nascosti consecutivi in una MLP

In caso di *batch learning* la trattazione è praticamente uguale con la sola differenza che la funzione di costo si modifica come segue:

$$\mathcal{E} = \frac{1}{2p} \sum_{t=1}^{p} \sum_{j=1}^{n} e_j^2 (t) \tag{3.27}$$

I ***descent methods*** sono dei metodi basati sul gradiente cioè cercano il minimo della funzione di costo annullando il gradiente di quest'ultima aggiornando in questo modo i pesi sinaptici. Si sottolinea che nella presentazione dei successivi metodi si farà riferimento ad un apprendimento di tipo on-line che presenta una trattazione praticamente equivalente ad un apprendimento off-line.

Si definisca il vettore gradiente $g(t)$ della funzione costo in relazione al vettore dei pesi sinaptici

$$g(t) = \frac{\partial \mathcal{E}(t)}{\partial \bar{w}(t)} = \left[\frac{\partial \mathcal{E}(t)}{\partial w_1(t)}, \ldots\ldots, \frac{\partial \mathcal{E}(t)}{\partial w_n(t)} \right]^T \tag{3.28}$$

Allora la variazione del vettore dei pesi sinaptici

$$\Delta \bar{w}(t) = -\eta \, G \frac{\partial \mathcal{E}}{\partial \bar{w}(t)} \tag{3.29}$$

Dove G è una matrice definita positiva. Lo ***steepest descent method*** minimizza la funzione di costo avvicinandola al minimo più vicino: non è detto che la funzione sia minimizzata globalmente, dipende tutto dalle condizioni iniziali. Per questo metodo $G = I$.

Il **metodo di Newton** ricava il vettore aggiornato dei pesi partendo dall'espressione dello sviluppo di Taylor. In questo caso si ricava $G = -H^{-1}$, dove H è la matrice hessiana. Se la matrice Hessiana non fosse definita positiva potrebbe presentarsi anche in questo caso il problema di minimo locale.

L'algoritmo di Levenberg-Marquardt è utilizzato per risolvere il problema dei minimi quadrati non lineare. In questo caso si ha:

$$\Delta\bar{w}(t) = -\left(H(t) + \lambda I\right)^{-1} \frac{\partial \mathcal{E}}{\partial \bar{w}(t)} \tag{3.30}$$

Con λ fattore di regolarizzazione che sarà discusso più approfonditamente nel capitolo successivo. Volendo confrontare il BP e il LM, numerosi studi hanno attestato che il BP è più veloce per ogni iterazione ma è più lento nel trovare il minimo. Al contrario, per il LM le iterazioni sono più lente ma il minimo è trovato più velocemente. Inoltre, il LM risolve il problema del minimo locale.

3.5 Verifica della rete: problema del minimo locale e *overfitting*

Una volta che la rete è stata addestrata, tocca testarla per verificare che effettivamente risponda nel modo corretto quando sollecitata da una qualsiasi set di input. In effetti, la fase di test risulta essere fondamentale per il corretto funzionamento della rete e per ottimizzarla. In questa fase si cerca infatti di ovviare a dei classici problemi che potrebbero compromettere la sua capacità di approssimazione con un basso errore. I problemi più comuni sono:

- *Overfitting.*
- Problema del minimo locale.

L'*overfitting* si verifica nel momento in cui la rete subisce un' addestramento eccessivo (*overtraining*) cioè se si forniscono alla rete troppi pattern d'esempio oppure se costruisco la rete con un numero eccessivo di neuroni, penalizzando di fatto la sua capacità di generalizzazione: se troppo addestrata, la rete potrebbe 'imparare a memoria' i pattern d'esempio senza riuscire ad applicare ciò che ha appreso a dei contesti più generici. Ma l'*overfitting*

può anche essere determinato dal un mal condizionamento insisto nella natura stessa del problema. Per evitare che questo accada si ricorre a delle tecniche che consentono di trovare il numero di neuroni che ottimizzi le performance della rete in termini di generalizzazione. Queste tecniche sono il *pruning* e il *growing* : la prima prevede che si parta da un numero sovrastimato di neuroni per poi studiare come varia il comportamento della rete quando se ne sottraggono alcuni; viceversa, per il *growing* si aggiungono man mano neuroni alla rete e si verifica se ci sono miglioramenti nella capacità di estrapolazione e generalizzazione.

I metodi di *pruning* generalmente si fondano si tecniche di regolarizzazione la cui idea generale è quella di migliorare la capacità di generalizzazione della rete aggiungendo un termine di 'regolarizzazione' appunto che permetta di individuare i neuroni che risultano debolmente attivati e cioè che sono inutili, per poi eliminarli.

Esistono però altre tecniche che permettono di risolvere con efficacia il problema dell'*overfitting* e sono chiamate *model selection critera*. Tali metodi minimizzano l'errore di generalizzazione della rete attraverso un confronto tra architetture simili in cui si fanno variare alcuni parametri scelti arbitrariamente. La filosofia è quella di minimizzare l'errore della rete in fase di test attraverso una predizione di tale errore in fase di addestramento utilizzando dei sottoinsiemi dei dati di training.

Altre tecniche, dette *subset selection* possono essere sfruttate unitamente ai *model selection criteria* per effettuare procedimenti di *growing* e *pruning*. Sia i *model selection critera*, sia i la teoria di regolarizzazione saranno descritte più approfonditamente nel capitolo successivo.

Per quanto riguarda il problema di minimo locale, esso è legato alle condizioni di inizializzazione della rete: a seconda delle condizioni iniziali scelte, la rete tende al minimo locale cioè più vicino. Questo problema può essere evitato con delle tecniche di computazione avanzate che però di solito non sono utilizzate perché troppo lente. L'alternativa è quella di effettuare più addestramenti in parallelo con CI diverse e selezionare le reti che si avvicinano alla soluzione commettendo un errore minore oppure, come già detto, utilizzare l'algoritmo LM che permette di bypassare tale problema.

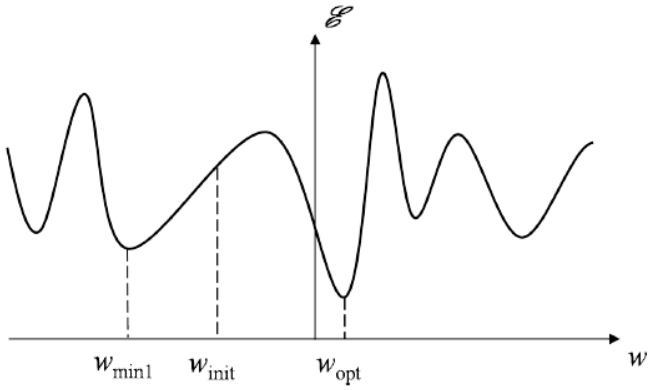


Figura 3.8: Rappresentazione di un minimo locale in una funzione generica

Capitolo 4

Reti neurali *Radial Basis Functions*

Nel capitolo precedente si era visto come costruire una generica rete neurale ripercorrendo tutte le possibili tecniche di *training* per le reti supervisionate e non. In particolare, ci si è concentrati sulle tecniche di apprendimento di una rete supervisionata MLP, mostrando brevemente la filosofia alla base del processo di training. L'approccio applicato, basato sulla correzione iterativa dei pesi sinaptici è anche conosciuto come approssimazione stocastica, nel senso che la rete cerca la soluzione compiendo delle scelte statisticamente ottimali sulla base di ciò che ha appreso e sulla base delle caratteristiche degli input in ingresso sfruttando tecniche di inferenza statistica. Tale approccio può essere molto utile nel caso di dati soggetti a rumore che quindi contengono in sé una natura stocastica ma spesso non è quello più adatto ad approssimare delle funzioni di un sistema complesso che dipendono da un elevato numero di parametri.

In questo capitolo, invece, l'approccio adottato è totalmente differente e fa riferimento al problema del *curve fitting* in uno spazio ad elevato numero di dimensioni. In parole povere, in questo approccio 'apprendere' vuol dire cercare la funzione che 'calza meglio' ai dati di *training* in uno spazio altamente dimensionale. In questo caso la statistica è coinvolta nella valutazione per capire quanto bene l'output stimato (ovvero la curva multidimensionale cercata) interpoli l'output desiderato.

4.1 Il teorema di Cover e la separabilità lineare dei patterns

Nell'ambito delle reti neurali è possibile, dunque, assegnare una funzione radiale ad ogni unità dello strato nascosto: in quest' ottica le funzioni radiali possono essere considerate delle basi per i pattern di input, che servono alla rete per interpolare funzioni multivariate. Nella fase di test la capacità di generalizzazione della rete dipende da come la curva, interpolante i pattern di training, interpola i dati di test.

Un teorema fondamentale per lo studio delle reti neurali RBF è il teorema di Cover (1965). Cover dimostra che:

'A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.' [5]

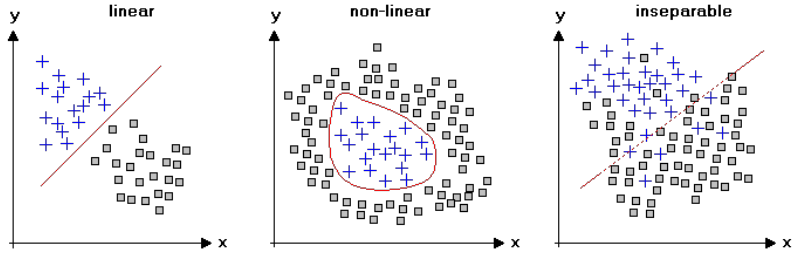


Figura 4.1: Tipologie di separabilità di un insieme di dati in uno spazio 2D. Immagine tratta da `http://www.statistics4u.com/fundstat_eng/cc_data_structure.html`

Cover, cioè, afferma che applicando una trasformazione non lineare (in questo caso una trasformazione radiale) ad un set di dati di *training* non linearmente separabile, è possibile proiettare questi ultimi in uno spazio multidimensionale in modo tale che essi assumano una distribuzione che li renda linearmente separabili.

Questo trucco è meglio conosciuto come 'trucco del Kernel'. Il metodo è invece conosciuto come mappatura non lineare. Per comprendere meglio di cosa si parla è bene fare un esempio nel caso bidimensionale in modo che sia facilmente visualizzabile : si supponga di avere un set di dati di input classificabili in base al colore come in figura 4.1. È ovvio che in uno spazio

bidimensionale gli input non sono linearmente separabili nelle due classi, il che rende molto complessa la classificazione. Per tale ragione potremmo applicare ai dati di input una funzione del tipo $z = x^2 + y^2$ in modo che lo spazio degli input sia linearmente separabile. Ciò consente una classificazione più agevole che semplifica notevolmente l'approssimazione della funzione di output e quindi il problema di *curve fitting*.

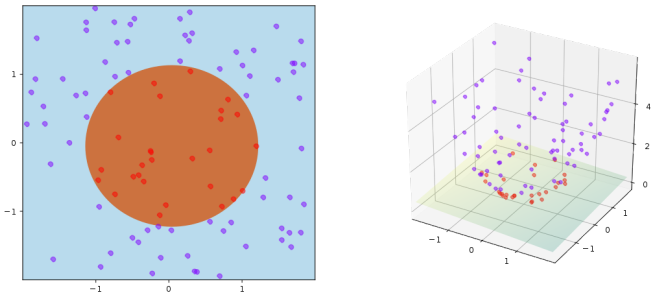


Figura 4.2: Separabilità lineare con 'trucco del Kernel'. Immagine tratta da `https://commons.wikimedia.org/wiki/User:Shiyu_Ji#/media/File:Kernel_trick_idea.svg`. Autore: Shiyu Ji

Trasportando quanto detto in uno spazio multidimensionale (ovvero un iperspazio che in matematica è un qualsiasi spazio multidimensionale a più di 3 dimensioni), si dice che una funzione multivariata è linearmente separabile nell'iperspazio se esiste almeno un iperpiano (cioè un qualsiasi sottospazio la cui dimensione è inferiore di uno all'iperspazio corrispondente) che separa i dati di *training* tra tutti i possibili iperpiani che separano l'iperspazio in semi iperspazi. L'applicazione del 'trucco del kernel', afferma ancora Cover, rende il problema linearmente separabile con una probabilità molto più alta se l'iperspazio di proiezione è altamente multidimensionale.

In particolare, un importante corollario del teorema di Cover, afferma che la capacità di separabilità di una famiglia di curve o funzioni intesa come il numero massimo di vettori (di input alla rete nel nostro caso) linearmente separabili in uno sazio di dimensionalità m, è uguale a $2m$. Dunque, è ovvio che più la multidimensionalità dell'iperspazio di proiezione dei dati è elevata, più sale il numero massimo di vettori linearmente separabili così come la probabilità che questo evento accada.

Applicando quanto detto alle reti neurali si comprende che, incrementando di molto il numero delle unità nascoste, ci si riconduce nelle condizioni

precedentemente enunciate nel teorema di Cover: l'iperspazio in cui proiettiamo i dati di input durante il *training* è costituito da un numero elevato di dimensioni e la probabilità che il problema sia linearmente separabile è più elevata.

Ne si deduce una caratteristica fondamentale delle reti neurali RBF, e cioè che sono costituite da un numero di unità nascoste in media molto più elevato delle reti MLP.

4.2 Funzioni Radiali Basiche

Una funzione radiale basica (RBF) è una particolare funzione reale ϕ che dipende dalla distanza tra un valore di input x e un punto fisso x_i che è il centro della funzione radiale. Nel caso in cui si abbiano più *pattern* di input, x è un vettore e la distanza tra tale vettore e il centro della funzione radiale basica coincide con la distanza euclidea.

$$\phi(x) = \phi(||x - \mu||) \tag{4.1}$$

Dati due vettori n-dimensionali $\bar{p}$ e $\bar{q}$, la distanza euclidea tra entrambi si calcola come segue:

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2} \tag{4.2}$$

La distanza euclidea tra il vettore x e il centro μ può anche essere interpretata come raggio r della funzione radiale.

$$r = ||x - \mu|| \tag{4.3}$$

Di seguito si elencano le varie tipologie di funzioni RBF in funzione di un parametro di forma ϵ :

- Gaussian:

$$\varphi(r) = e^{(-\epsilon r)^2} \tag{4.4}$$

- Multiquadric:

$$\varphi(r) = \left(1 + (r\epsilon)^2\right)^{\frac{1}{2}} \tag{4.5}$$

- Tin plate spline:

$$\varphi(r) = r^2 \log r \tag{4.6}$$

- Inverse multiquadric:

$$\varphi(r) = \frac{1}{\left(1 + (r\epsilon)^2\right)^{\frac{1}{2}}} \tag{4.7}$$

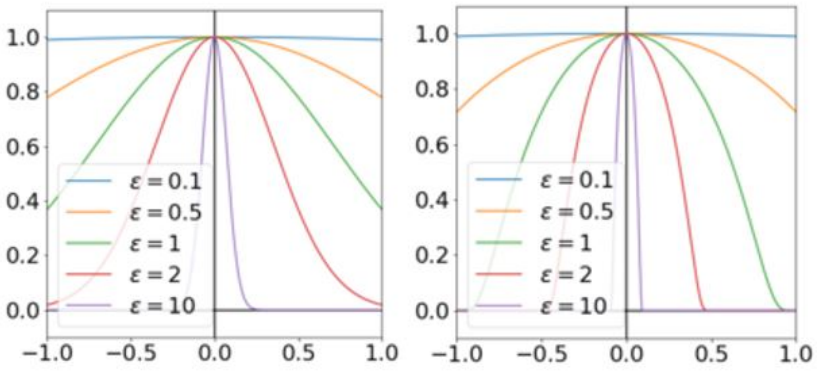


Figura 4.3: Tipologie di funzioni radiali basiche al variare di ϵ. Immagine tratta da `https://en.wikipedia.org/wiki/Radial_basis_function#/media/File:Gaussian_function_shape_parameter.png`. Autore: Shawsa7

Tali funzioni sono tutte strettamente positive, da qui l'uso del parametro di forma. Di solito la RBF più utilizzata è la funzione gaussiana.

4.3 Architettura di una rete neurale RBF

Il problema dell'interpolazione multivariata in uno spazio ad elevate dimensioni è stato oggetto di numerosi studi. Facendo riferimento alla teoria di Davis del 1963, si può parlare di interpolazione rigorosa quando la superficie interpolante passa per tutti i punti di *training*. Questa soluzione è solo ipotizzabile per via teorica perché vorrebbe dire che dovremmo avere tante funzioni radiali e tante unità nascoste, quanti sono i pattern di input presentati alla rete. Per risolvere tale problema un metodo di interpolazione

approssimato verrà presentato nel paragrafo 4.4. Chiamando $\hat{f}$ la funzione interpolante, allora deve risultare che:

$$\hat{f}\ (x_i) = d_i \tag{4.8}$$

Dove x_i è un generico valore del vettore di input e d_i è l'output desiderato corrispondente. La tecnica di interpolazione delle funzioni radiali basiche prevede che l'output stimato dalla rete sia calcolato come combinazione lineare della produttoria tra i pesi e le funzioni radiali basiche (dipendenti dal raggio e cioè dalla distanza tra il vettore input e il centro della funzione radiale) Si noti che i dati del vettore di input, sono scelti proprio come centri delle funzioni radiali basiche in questo approccio. Vedremo, tuttavia, che esistono anche altri criteri per la scelta dei centri delle RBF.

$$\hat{f}\,(x) = \sum_{i=1}^{n} w_i \ \ \phi_i\,(||x - x_i||) \tag{4.9}$$

Quanto visto equivale a costruire una rete con un unico strato nascosto, uno strato di input ed uno di output. Questa rete è progettata per effettuare una mappatura non lineare dei dati di input (nello strato nascosto) tramite l'applicazione delle funzioni RBF (come visto nel paragrafo introduttivo del capitolo) insieme ad una mappatura lineare (che consiste nel moltiplicare l'output del neurone j-esimo dello strato *hidden* per un peso sinaptico). Le funzioni radiali basiche sono diverse per ogni neurone e sono distribuite in modo da coprire tutto l'iperspazio degli input. In questo lavoro si testerà una rete le cui RBF coincidono con delle gaussiane:

$$\phi_i\,(x) = exp\left(-\frac{||x - \mu\ _i||}{2\sigma_i^2}\right) \tag{4.10}$$

Dove σ è la varianza della gaussiana che si calcola come:

$$\sigma_i^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j - x_i)^2 \tag{4.11}$$

Dove n è il numero totale dei pattern della rete, cioè delle variabili di input.

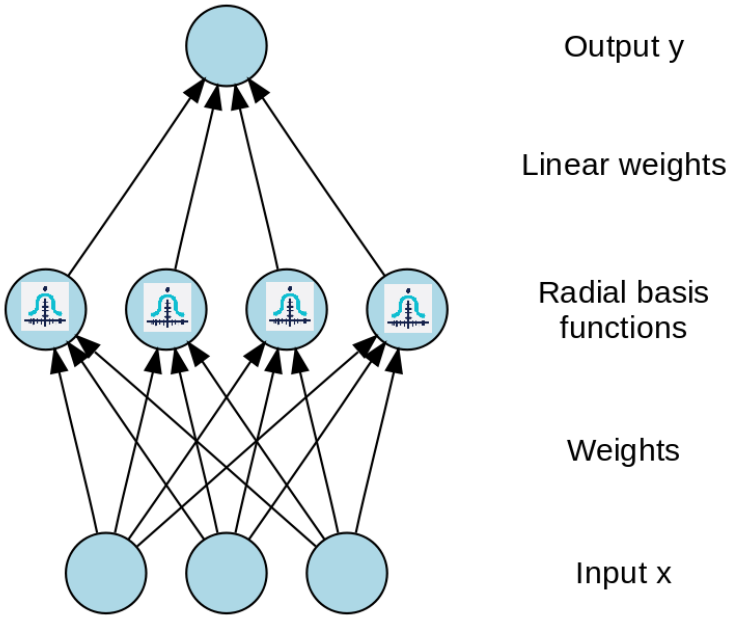


Figura 4.4: Architettura di una rete neurale RBF. Immagine tratta da `https://en.wikipedia.org/wiki/Radial_basis_function_network#/media/File:Radial_funktion_network.svg`. Autore: SebDE

4.4 Apprendimento delle reti RBF

Come visto in generale per le reti neurali, anche per le reti RBF la filosofia utilizzata per l'apprendimento della rete è quella che si basa sulla minimizzazione dell'errore e quindi della funzione di costo. I parametri che caratterizzano il processo di addestramento della rete sono i seguenti:

- Il numero di neuroni nello strato nascosto
- Le coordinate del centro di ciascuna funzione RBF di livello nascosto.
- Il raggio (diffusione) di ciascuna funzione RBF in ciascuna dimensione.
- I pesi applicati agli output della funzione RBF quando passano al *summation layer*.

Le RBF possono essere addestrate seguendo varie tecniche. L'approccio più comune è quello di utilizzare un algoritmo ibrido che prevede un apprendimento non supervisionato per la determinazione dei centri e delle estensioni

delle RBF unitamente con un apprendimento supervisionato per la determinazione dei pesi ottimali della rete. I passaggi fondamentali da compiere sono:

1. Fissare il numero di unità nascoste.
2. Assegnare centri e raggi alle RBF delle unità.
3. Calcolare i pesi ottimali (che minimizzano la funzione di costo).
4. Verificare la capacità di generalizzazione della rete ed eventualmente applicare delle tecniche di ottimizzazione dell'architettura.

Per quanto riguarda la determinazione non supervisionata di centri e varianze delle funzioni radiali, l'algoritmo a cui di solito riferimento è il K-means. Lo scopo è quello di trovare prima i centri dei cluster che vengono poi utilizzati come centri per le funzioni RBF. Tuttavia, il *clustering* K-means è computazionalmente intenso e spesso non genera il numero ottimale di centri. Un altro approccio consiste nell'utilizzare un sottoinsieme casuale di dati *training* come centri.

Il calcolo dei pesi ottimali tra i neuroni nello strato nascosto e nello strato di sommatoria viene eseguito utilizzando di solito il LS oppure l'OLS: per le reti RBF non ci si deve preoccupare del fatto che il problema sia mal posto perché sono delle reti auto-regolarizzate che, come dimostrato da Tichonov, hanno la caratteristica intrinseca di risolvere i problemi mal posti. Una procedura iterativa calcola il parametro λ di regolarizzazione ottimale in base al modello scelto per la previsione dell'errore.

La risoluzione del problema di *overfitting* è affidata a metodi di regolarizzazione di ordine superiore a quello di Tichonov e tecniche che selezionano l'architettura più ottimizzata della rete chiamati *model selection criteria*.

Il numero di neuroni dello strato nascosto non è per forza un parametro predeterminato: tutto dipende dal tipo di approccio utilizzato. La scelta più comune, quella che ottimizza nel miglior modo la rete, consiste nell'adottare un algoritmo di addestramento che utilizza un approccio evolutivo basato su una logica *forward selection* o *backward selection*. Questo metodo monitora le performance della rete aggiungendo o sottraendo volta per volta un singolo neurone. L'aggiunta dei neuroni (nel caso *forward*) si ferma nel momento in cui l'errore inizia ad aumentare a causa di *overfitting*.

4.4.1 Storia delle tecniche di *learning*

Pioggio e Girosi furono i primi a proporre, alla luce dei loro studi sulle tecniche di regolarizzazione delle reti, una rete neurale le cui funzioni kernel fossero costituite da funzioni radiali basiche [6]. Essi infatti dimostrarono come tali reti fossero un caso particolare di reti neurali regolarizzate. Con questo approccio è possibile comprendere meglio come funziona una rete RBF. Nelle reti regolarizzate il numero di unità nascoste è uguale al numero di input del problema ed è dunque fissato a priori. Per la determinazione dei pesi è possibile utilizzare un metodo *gradient descent* che prevede la minimizzazione di una funzione di costo per avvicinare la funzione approssimata alla funzione desiderata. I centri delle unità sono scelti arbitrariamente dal set dei dati di input. Spesso per ridurre la complessità della rete si utilizza un'approssimazione della funzione regolarizzata come si vedrà in seguito.

Un altro punto di vista che mette in risalto un ulteriore vantaggio delle RBFNN fu dato da Moody e Darken che studiarono gli effetti dell'applicazione di funzioni *locally tuned* sulle reti neurali. Grazie al carattere locale di tali funzioni di attivazione si riusciva a comprendere quali unità fossero localmente attivate (quelle che restituivano un valore di output non trascurabile e cioè quelle per cui i centri erano vicini ai dati di input) in corrispondenza di un determinato input ed eventualmente omettere le unità non necessarie. Anche in questo studio il numero dei neuroni è fissato a priori e i centri sono scelti arbitrariamente a patto che siano uniformemente distribuiti nel dominio.

Dunque, gli studi precedentemente descritti hanno mostrato che le reti RBF possono essere estremamente vantaggiose per l'approssimazione di funzioni sia perché sono delle reti regolarizzate, il che riduce notevolmente l'errore e risolve eventuali problemi di mal condizionamento e *overfitting*, sia per il loro carattere locale che consente di ridurre il costo computazionale e di comprendere quali unità sono localmente attivate per poi applicare delle tecniche di *pruning* volte a ridurre la complessità della rete.

Riguardo le tecniche di apprendimento di queste reti, alcuni studi hanno messo in mostra che un metodo *fully supervised* non mostra molti più vantaggi del classico algoritmo di *back propagation* e che non mantiene la località delle funzioni radiali perché queste assumono varianze elevate. Per tali ragioni si è deciso di utilizzare un algoritmo di apprendimento ibrido che comprende un *K-means clustering* per la determinazione dei centri delle unità

(*self orgnized*) e il classico metodo dei minimi quadrati per la determinazione dei pesi (*supervised* perché i pesi sono determinati tramite tale metodo dalla conoscenza di input e output). La località delle funzioni è garantita da tale algoritmo il che assicura una convergenza abbastanza veloce e migliore del metodo *back propagation*.

Molti tuttavia notarono l'inadeguatezza dell'utilizzo di un metodo *k-means* per la determinazione dei centri in quanto in quanto due campioni di dati vicini tra loro non hanno necessariamente lo stesso valore di output. Per tale ragione si preferisce utilizzare la tecnica dell'OFR (*orthogonal forward regression*)proposta da Chen et Al per selezionare i centri dei neuroni nascosti: tale tecnica fa riferimento ad un metodo chiamato *orthogonal least square* in cui si utilizza un 'progettone' (si veda 4.4.4) per selezionare i centri migliori uno alla volta. Il numero dei neuroni è sempre predeterminato e raggiunto tale numero tale algoritmo si ferma.

Tuttavia, spesso non si conosce a priori il numero ottimale di neuroni che possa ottimizzare le performance della rete. In tal senso è stato sviluppato un algoritmo chiamato *hyerarchically self organizing learning* (HSOL) che aggiunge automaticamente nuovi neuroni per migliorare il comportamento della rete, qualora il nuovo campione non ricada entro i limiti di influenza delle unità già esistenti in quel momento. Si parte in assenza di unità nascoste per poi proseguire con una logica di *growing* passando da regioni di influenza più larghe, che restituiscono una soluzione generalmente corretta ma più rozza, a regioni di influenza più strette che restituiscono una soluzione più corretta. L'alternativa è quella di partire da un numero di neuroni pari al numero di input e sottrarre neuroni con una logica di *pruning*(Musavi).

Oltre all'algoritmo OFR, visto precedentemente, Chen e Billings formularono un ulteriore algoritmo utilizzato soprattutto per l'on line *learning* basato su una tecnica di *clustering* per determinare la posizione più corretta dei centri e un metodo ricorsivo ai minimi quadrati per determinare i pesi. Anche in questo caso il numero di neuroni è già prefissato e, per trovarne il valore ottimale, bisogna effettuare numerose prove.

Successivamente sono stati sviluppati numerosi algoritmi basati su una logica simile a quella *forward selection* (discussa nel paragrafo 4.4.6) che hanno permesso di superare il suddetto problema e cioè di aggiungere in modo adattivo dei neuroni alla rete e di valutarne iterativamente le performance fino a raggiungere una rete ottimizzata dal punto di vista del numero di unità. Tra questi l'algoritmo RAN (Resource Allocating Network)che fu proposto da

Platt, un altro algoritmo ONSAHL (*On-line Structural Adaptive HybridLearning*) messo appunto da Junge e Unbehauen, l'algoritmo RANEKF(*Extended Kalman Filters Resource Allocating Network*) e l'algoritmo EMRAN (*Extended Resource Allocating Network*), introdotti con lo scopo di ottimizzare le performance dell'algoritmo RAN. Questi algoritmi verranno discussi in modo più dettagliato nel successivo capitolo.

4.4.2 Ottimizzazione dei pesi sinaptici tramite LS

Quando il problema seguente non è ben posto (sia a causa della non esistenza, sia per la non unicità della soluzione interpolante un set di dati x)

$$Ax = b \tag{4.12}$$

allora l'approccio usuale è noto come metodo dei minimi quadrati lineari e consiste nel minimizzare lo scarto S totale della funzione detto anche *sum squared error*. La matrice A è quella matrice che linearizza il legame tra input noti e output desiderati.

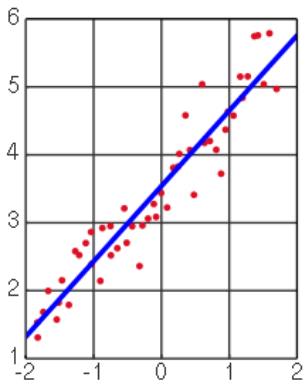


Figura 4.5: Visualizzazione grafica del LS lineare. Immagine tratta da `https://it.wikipedia.org/wiki/Metodo_dei_minimi_quadrati#/media/File:Linear_least_squares.svg`. Autore: Oleg Alexandrov

Nel caso in esame:

$$\left\{\hat{f}\,(x)\right\} = Ax = \Phi(x)\,\{w\} \tag{4.13}$$

$$b = \{f(x)\} \tag{4.14}$$

$$\Phi = \begin{bmatrix} \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \vdots & \ddots & \vdots \\ \phi_1\ (x_n) & \cdots & \phi_n(x_n) \end{bmatrix} \tag{4.15}$$

Dove $\Phi(x)$ è la matrice di design della rete le cui riga i-esima è costituita dalle n funzioni RBF relative alle n unità della rete calcolate nella generica x_i di input ; la matrice w è la matrice dei pesi che contiene tanti pesi quante sono le unità della rete; il vettore b è il vettore dei valori veri della funzione cioè quelli desiderati; il vettore $\hat{f}\,(x)$ è il vettore dei valori approssimati calcolati dalla rete neurale. In merito alle funzioni radiali di base elencate nel paragrafo 4.2 è bene menzionare il teorema di Micchelli (1986) che afferma:

Esiste una classe C di funzioni di base tale che, se i punti di interpolazione $x_1,\ldots\ldots x_p$ *in* R^m *sono distinti, per ogni* ϕ *in C la corrispondente matrice di interpolazione* Φ *in C è non singolare.*

La condizione perché la matrice di interpolazione sia non singolare è che tutti i punti siano distinti. Solo in questo caso è possibile applicare il metodo ai minimi quadrati invertendo la matrice di interpolazione per il calcolo dei pesi sinaptici ottimali della rete. La minimizzazione del *sum squared error* che può essere calcolato come:

$$S = \sum_{i=1}^{n} \left|\left| \hat{f}\,(x_i) - f\,(x_i) \right|\right|^2 \tag{4.16}$$

dove $||\ \ ||$ è la norma euclidea, conduce alla seguente equazione che permette di calcolare i pesi sinaptici ottimali.

$$\{w\} = A^{-1}\Phi^T\ \left\{\hat{f}\,(x)\right\} \tag{4.17}$$

Dove A è la matrice di covarianza e si definisce come:

$$A = \Phi^T\Phi \tag{4.18}$$

Riassumendo, considerare la somma degli scarti quadratici come funzione di costo e minimizzarla equivale ad applicare al sistema un'ottimizzazione nel senso dei minimi quadrati.

4.4.3 Problemi mal posti: Regolarizzazione di Tichonov

Nella maggior parte dei problemi reali le condizioni del teorema di Michelli non sarebbero soddisfatte, non essendo tutti i punti di interpolazione distinti. In questo caso la matrice di covarianza risulta singolare e il problema è irrisolvibile: un problema del genere si dice mal posto.

La regolarizzazione di Tichonov rappresenta una delle prime teorie di regolarizzazione di problemi mal posti (*ill-posed problems*) [7]. Per le reti neurali RBF questa trattazione è di fondamentale importanza perché mostra che queste ultime hanno la caratteristica intrinseca di essere auto-regolarizzate e di risolvere i problemi mal posti senza ricorrere a tecniche particolarmente complesse. Da qui la spiegazione del perché queste reti siano particolarmente adatte all'interpolazione di funzioni multivariate in un contesto sistemistico altamente complesso. Infatti, si vede che la funzione regolarizzata che approssima l'output desiderato (della rete) F_λ, calcolata minimizzando la funzione di costo di Tichonov, è data dalla combinazione lineare proprio di funzioni radiali basiche. Il procedimento che porta a questa conclusione è estremamente complesso [1], per questo ci si limiterà ad evidenziarne i passaggi chiave. La funzione di costo di Tichonov si esprime come segue:

$$\mathcal{E}\left(F\right) = \frac{1}{2}\sum_{i=1}^{l}\left(d_i - F\left(x_i\right)\right)^2 + \frac{1}{2}\lambda\left|\left|DF_\lambda\left(x\right)\right|\right| \tag{4.19}$$

Il secondo termine è detto termine di regolarizzazione: in esso compare il parametro λ i di regolarizzazione e un operatore differenziale D. Risolvere il problema di regolarizzazione vuol dire trovare la funzione F_λ che minimizza la funzione di Tichonov $\mathcal{E}(F)$.

Per minimizzare la funzione di costo ottenuta precedentemente si utilizza il differenziale di Frétches e lo si inserisce nell'identità di Green per poi ricavare l'equazione di Eulero Lagrange da cui, sfruttando la definizione di funzione di Green si ottiene l'espressione di F_λ:

$$F_\lambda\left(x\right) = \frac{1}{\lambda}\sum_{i=1}^{n}\left[d_i - F\left(x_i\right)\right]G(x, x_i) \tag{4.20}$$

Dove possiamo scegliere come funzioni di Green proprio delle multivariate funzioni gaussiane (che soddisfano le proprietà delle funzioni di Green)

centrate nei punti dei set di input :

$$G\left(x, x_i\right) = \exp\left(-\frac{1}{2\sigma_i^2}\left|\left|x - x_i\right|\right|^2\right) \tag{4.21}$$

Dunque, il valore approssimato della funzione sottoposta a regolarizzazione può essere visto come sovrapposizione lineare di funzioni basiche radiali (in questo caso gaussiane multivariate). In tal senso le reti neurali RBF possono essere viste come reti neurali regolarizzate il che si traduce in notevoli vantaggi nell'approssimazione di funzioni non lineari (Poggio e Girosi, 1990)[6].

$$F_\lambda\left(x\right) = \sum_{i=1}^{n} w_i \exp\left(-\frac{1}{2\sigma_i^2}\left|\left|x - x_i\right|\right|^2\right) \tag{4.22}$$

Dove, confrontando l'espressione 4.20 e l'espressione 4.22, si ottiene il valore dello i-esimo peso sinaptico detto anche coefficiente di espansione di Tichonov:

$$w_i = \frac{1}{\lambda}[d_i - F_\lambda\left(x_i\right)] \tag{4.23}$$

Determinazione dei pesi di Tichonov

Come si è visto, la formulazione della funzione di Tichonov conduce ad un'espressione dei pesi che dipende dalla stessa funzione regolarizzata che stiamo cercando.

$$w_i = \frac{1}{\lambda}[d_i - F_\lambda\left(x_i\right)] \tag{4.24}$$

Per determinare i pesi bisogna eliminare questa dipendenza eguagliando l'equazione precedente scritta in forma vettoriale e l'equazione di approssimazione della funzione regolarizzata, scritta sempre in forma vettoriale.

$$\mathbf{w} = \frac{1}{\lambda}\left(\mathbf{d} - \mathbf{F}_\lambda\right) \tag{4.25}$$

$$\mathbf{F}_\lambda = \mathbf{\Phi w} \tag{4.26}$$

Si ottiene:

$$\left(\mathbf{\Phi} + \lambda\mathbf{I}\right)\mathbf{w} = \mathbf{d} \tag{4.27}$$

Da cui si ricava

$$\mathbf{w} = (\mathbf{\Phi} + \lambda \mathbf{I})^{-1} \mathbf{d} \tag{4.28}$$

Dove, proprio per l'introduzione del termine di regolarizzazione, la matrice $\mathbf{\Phi} + \lambda \mathbf{I}$ utilizzata per il calcolo dei pesi è definita positiva, ed è pertanto invertibile. Il problema, da mal posto è diventato ben posto.

Reti neurali regolarizzate

Il risultato ottenuto è rilevante se si considera che ci è bastato esprimere la funzione approssimante come combinazione lineare di funzioni gaussiane per risolvere il problema. Guardando la questione da un'altra prospettiva appare evidente che da questo approccio sia nata l'idea si costruire delle reti neurali regolarizzate (di cui le RBFnn sono una sottocategoria) utilizzando l'architettura descritta nel paragrafo 4.3. In altre parole tale approccio giustifica quello che precedentemente era stato chiamato come metodo di interpolazione RBF (si veda equazione 4.9).

In sintesi le reti regolarizzate presentano due caratteristiche fondamentali:

- Sono degli 'approssimatori universali' [1] di qualsiasi funzione multivariata, per la capacità di auto-separazione lineare dei dati di input forniti alla rete, purché il numero di neuroni nascosti sia abbastanza elevato.
- Sono delle reti ottimali, nel senso che auto- minimizzano la funzione di costo, con la capacità di risolvere qualsiasi problema mal-posto.

Reti RBF generalizzate

L'interpolazione rigorosa attuata nella suddetta strategia di interpolazione prevede che tutti i punti siano distinti il che vuol dire che dovremmo avere tante funzioni radiali e tante unità nascoste, quanti sono i dati presentati alla rete. Questa soluzione è solo ipotizzabile per via teorica. Nei problemi reali, il numero dei dati è molto maggiore rispetto al numero di basi ovvero di unità della rete (gradi di libertà): il problema si dice sovradeterminato.

Per risolvere questo problema si applica il metodo di Galerkin che esprime la funzione desiderata in forma approssimata come combinazione lineare di un certo numero finito di m basi, per poi assicurare la convergenza minimizzando il residuo tra la funzione approssimata e quella desiderata. Nel caso in esame

la funzione desiderata è la funzione regolarizzata di approssimazione (della rete) F_λ; la funzione approssimata di F_λ la si chiama come F^*

$$F^*(x) = \sum_{i=1}^{m} w_i \phi(x - t_i) \tag{4.29}$$

Le basi, cioè le unità della rete utili per approssimare la F_λ sono ridotte ad m<n (con n numero di dati di input) e centrate in punti scelti arbitrariamente nello spazio degli input $(t_1, \ldots\ldots, t_m)$. La soluzione di un problema ai minimi quadrati sovradeterminato con metodo di Galerkin conduce ad una soluzione simile a quella vista nell'equazione 4.28. La differenza è che la matrice Φ ha dimensioni $n * m$ e che si aggiunge un'ulteriore matrice Φ_0 ovvero la matrice delle RBF con centri nei punti arbitrariamente scelti $(t_1, \ldots\ldots, t_n)$ calcolata negli stessi centri.

$$\mathbf{w} = \left(\Phi^T \Phi + \lambda \Phi_o\right)^{-1} \Phi^T \mathbf{d} \tag{4.30}$$

$$\Phi = \begin{bmatrix} \phi(x_1, t_1) & \cdots & \phi(x_1, t_m) \\ \vdots & \ddots & \vdots \\ \phi\ (x_n, t_1) & \cdots & \phi(x_n, t_m) \end{bmatrix} \tag{4.31}$$

$$\Phi_0 = \begin{bmatrix} \phi(t_1, t_1) & \cdots & \phi(t_1, t_m) \\ \vdots & \ddots & \vdots \\ \phi\ (t_n, t_1) & \cdots & \phi(t_n, t_m) \end{bmatrix} \tag{4.32}$$

In questo modo il costo computazionale può scendere perché la rete può ammettere un numero di neuroni minore dei dati di input.

Questa trattazione apre delle riflessioni interessanti sulla scelta dei centri più adatti per le RBF nello spazio degli input. Su questo ci si soffermerà nel paragrafo successivo.

Strategie di selezione dei centri

Come detto, l'introduzione delle reti RBF generalizzate, ottenute utilizzando il metodo di Galerkin, dà la possibilità di scegliere arbitrariamente centri delle RBF: al contrario delle reti regolarizzate convenzionali il numero i centri non sono più scelti coincidenti con i dati di input. Di seguito saranno elencate le strategie più comuni di selezione dei centri delle funzioni radili basiche:

- **Selezione randomica** dei centri: si stabilisce un valore massimo della distanza tra m centri scelti randomicamente d_{max} e si stabilisce che la varianza delle gaussiane scelte sia pari al valore $\sigma = d_{max} / \sqrt{m}$, per assicurassi che le gaussiane non siano ne troppo piatte ne troppo pronunciate.
- **Selezione non supervisionata** (automatica) dei centri, tramite utilizzo del *k-means clustering* che dispone i centri nelle regioni dello spazio in cui c'è una rilevante concentrazione degli input. La limitazione di tale algoritmo consiste nel fatto che l'ottimizzazione è locale nel senso che dipende fortemente dalla scelta iniziale dei cluster centers. L'algoritmo *ehnanced k-means clustering* supera in un certo senso questo problema.
- **Selezione supervisionata** dei centri, tramite logica di minimizzazione dell'errore sfruttando il metodo *gradient descent* descritto nel capitolo precendente. I centri sono dei parametri liberi della rete che vengono fatti variare finchè il gradiente della funzione di costo risulta minimizzato. Questo metodo può essere affetto da problemi di minimo locale.

4.4.4 Problema di *overfitting* nelle reti RBF

Come già detto nel capitolo 4, il problema di *overfitting* coincide con l'incapacità di estrapolazione e generalizzazione della rete che non riesce ad applicare correttamente ai test quanto appreso in fase di addestramento, conducendo così ad una soluzione affetta da un notevole errore. Ciò è ancora più visibile se i dati sono soggetti ad un rumore di sottofondo. Nell'introduzione del corrente capitolo si era posta l'attenzione sul fatto che l'utilizzo delle reti RBF per risolvere problemi di interpolazione riduce già di per sé l'*overfitting* per l'intrinseca capacità di rendere i pattern di *training* linearmente separabili, come enunciato dal teorema di Cover. Sebbene ciò migliori notevolmente l'attitudine alla generalizzazione di queste reti, tuttavia non elimina il problema dell'*overfitting*. Per tale ragione si ricorre a dei metodi ad hoc:

- **La tecnica di regolarizzazione** che, si ribadisce, agisce sulla funzione di costo attraverso un termine di regolarizzazione cercando di individuare i patterns (e dunque i pesi) che sono poco importanti per la soluzione fino a 'spegnerli' definitivamente

- I ***model selection criteria*** che selezionano tra varie architetture simili in cui si fanno variare determinati parametri, l'architettura ottimale (in termini di errore). Tali tecniche hanno come compito quello di minimizzare l'errore della rete in fase di test effettuando una predizione di tale errore utilizzando dei sottoinsiemi dei dati di training.

Le tecniche di regolarizzazione possono essere considerate a tutti gli effetti delle tecniche di *pruning*, mentre i *model selection critera* sono applicabili più in generale a tutti i parametri della rete: ad esempio, possono essere applicate alle tecniche di regolarizzazione per determinare il parametro di regolarizzazione ottimale. Per determinare il numero più adatto di unità che minimizzano l'errore di generalizzazione è possibile applicare delle tecniche dette *subset selection* unitamente ai *model selection criteria*.

Nei paragrafi successivi si analizzeranno più nel dettaglio questi metodi; ci si soffermerà sulle modalità di applicazione di questi ultimi alle RBF in modo da comprendere come possano beneficiare alle performace della rete stessa.

Interpretazione statistica dell'apprendimento: dilemma dei bias e delle varianze

Il problema dell'*overfitting* può essere compreso appieno approfondendo uno dei principali problemi dell'apprendimento delle reti neurali che coinvolge il corretto bilanciamento tra i bias e le varianze della rete. Per comprendere di cosa si parla è bene fare un passo indietro per parlare dell'approccio stocastico all'apprendimento.L'approccio probabilistico è fondamentale per stimare quello che è chiamato errore medio atteso della rete. Lo scopo è quello di stabilire quanto la funzione target di regressione f si discosta dalla funzione di approssimazione calcolata dalla nostra rete. Se si considera un set X di variabili indipendenti scelte randomicamente nello spazio degli input e un vettore D di variabili dipendenti scelte sempre randomicamente, detta f la funzione di regressione (funzione deterministica) della rete, allora il vettore errore atteso è uguale a:

$$\epsilon = D - f(X) \tag{4.33}$$

È possibile dimostrare che il valore medio dell'errore atteso dati tutti i possibili sottoset di input, stimato con un operatore di aspettativa statistico

E su tutti i sottoset di input possibili, è nullo e che non dipende dalla funzione di regressione.

$$E\left[\epsilon|x\right] = 0 \tag{4.34}$$

$$E\left[\epsilon f\left(X\right)\right] = 0 \tag{4.35}$$

Dove la funzione di regressione può essere espressa in funzione di un sottoset x di input mediante l'operatore di stima statistico E:

$$f\left(x\right) = E[D|x] \tag{4.36}$$

Per stimare tale funzione si utilizza il metodo del 'Kernel' (si veda paragrafo 4.4.7). È possibile utilizzare queste relazioni statistiche per esprimere in modo diverso la funzione di costo non regolarizzata :

$$\frac{1}{2}E_m\left(\left(d - F\left(x\right)\right)^2\right) \tag{4.37}$$

Dove si è preferito esprimere la sommatoria con un operatore media E_m. Aggiungendo e sottraendo il termine $f\left(x\right)$ ed esprimendolo utilizzando la relazione 4.36, si ottiene:

$$\mathcal{E}\left(w\right) = \frac{1}{2}E_m\left(\left(d - F\left(x\right)\right)^2\right) + \frac{1}{2}E_m\left(\epsilon^2\right) \tag{4.38}$$

Effettuando poche altre manipolazioni si vede che la funzione di costo può essere espressa come segue:

$$\begin{aligned}\mathcal{E}\left(w\right) = \frac{1}{2}\left\{B^2\left(w\right) + V_F(w) + V_f\right\} = \ & \frac{1}{2}\{(E_m\left[F\left(x\right)\right] - E\left[D|x\right]) \\ & +(E_m(F\left(x\right) - E_m\left[F\left(x\right)\right]) + E_m\left[\epsilon^2\right]\}\end{aligned} \tag{4.39}$$

Dove :

- $B(w) = \ (E_m\left[F\left(x\right)\right] - E\left[D|x\right])$ è il bias della rete ed misura effettivamente di quanto la funzione di approssimazione si avvicina a quella di regressione. Può essere visto come un errore di approssimazione relativo al livello di infedeltà della soluzione.
- $V_F\left(w\right) = (E_m(F\left(x\right) - E_m\left[F\left(x\right)\right])$ la varianza della funzione di approssimazione misurato sul set di training. Può essere visto come un errore

di stima della funzione di approssimazione rispetto a quella di regressione relativo all'inadeguatezza e l'incertezza dei dati di *training*. In parole semplici misura l'oscillazione della soluzione attorno al valore corretto e, dunque, la 'ruvidità' della soluzione.

- V_f è la varianza del'errore del modello di regressione e non dipende dai pesi della rete.

La somma $B^2(w) + V_F(w)$ può essere vista come il valore medio dell'errore che la funzione di approssimazione commette stimando la funzione di regressione.

Il dilemma dei bias e delle varianze nell'apprendimento, consiste proprio nello scegliere il giusto bilanciamento tra i due per avere una soluzione quanto più corretta possibile minimizzando l'errore di stima della funzione di approssimazione.

Se la rete non ha bias vuol dire che la soluzione sarà 'approssimativamente' quella corretta ma potrebbe produrre notevoli oscillazioni se la varianza è grande: sarebbe molto sensibile a piccole variazioni come ad esempio il rumore. Il significato dei bias e delle varianze può considerarsi centrale nella teoria di regolarizzazione per la risoluzione dei problemi mal condizionati.

Il progettone

Un altro elemento fondamentale per comprendere come si comporta la rete in fase di generalizzazione è il progettone, una particolare matrice P che proietta i vettori di input di dimensione p in uno spazio m-dimensionale, con m numero di gradi di libertà della rete. Se la rete non è regolarizzata allora, la proiezione avverrà in uno spazio pari al numero di unità della rete.

$$P = I_p - \Phi A^{-1} \Phi^T \tag{4.40}$$

con $A = \Phi^T \Phi$ e Φ matrice delle RBF della rete (si veda equazione 4.15) e I_p vettore di input. Si dimostra che il sum square error può essere espresso in funzione del progettone:

$$S = \hat{y}^T P^2 \ \hat{y} \tag{4.41}$$

Se la rete è regolarizzata, allora l'espressione del progettone è la seguente:

$$P = I_p - \Phi A^{-1} \Phi^T \tag{4.42}$$

con $A = \Phi^T\Phi + \lambda I$ (come si vedrà in seguito).

Teoria della regolarizzazione applicata all'*overfitting*

La regolarizzazione di Tichonov, per quanto intrinseca nelle reti RBF, è una regolarizzazione abbastanza rozza ed elementare, non adatta a risolvere del tutto i problemi di mal condizionamento. Per migliorare ancora le performance della rete esistono vari tipi di regolarizzazione di ordine superiore. Statisticamente parlando, per regolarizzazione si intende l'introduzione di un'informazione aggiuntiva con la finalità di risolvere un problema mal condizionato o che, parlando in termini di *deep learning*, tende ad adattarsi eccessivamente ai dati di *training* con una conseguente incapacità di generalizzazione. Quando si parla di mal condizionamento, in matematica, ci si riferisce alla relazione che esiste tra il risultato atteso e l'incertezza sui dati che hanno generato quel risultato. Per meglio intendersi, nelle reti neurali un problema si dice mal condizionato se, sottoponendo i dati a delle piccole variazioni si ottiene una soluzione altamente instabile. Il sistema è, dunque, molto sensibile all'incertezza dei dati di input. Viceversa, se la soluzione non si allontana molto da quella desiderata, anche con l'applicazione di piccole perturbazioni, si parla di problema ben condizionato. Un problema mal condizionato determina una difficoltà nel calcolo della matrice inversa utile a determinare i pesi sinaptici che si traduce nell'impossibilità nel risolvere il problema, a meno che non si aggiungano delle informazioni le sono fornite proprio dai metodi di regolarizzazione. In ambito aeronautico è di fondamentale importanza che il problema sia ben condizionato, in quanto i dati di volo sono molto spesso soggetti ad un rumore di sottofondo percepito dai sensori e, dunque, continuamente soggetti a piccole perturbazioni le quali, se il problema è mal condizionato, possono condurre ad una previsione errata della rete in fase di *testing*. Per le reti neurali, il termine correttivo o di regolarizzazione viene applicato alla funzione di costo. L'aggiunta di tale termine si basa su una logica volta a minimizzare il rischio di errore ovvero la funzione di performance della rete (funzione di costo), e quindi fa parte di un approccio probabilistico all'apprendimento.

$$\mathcal{E}(w) = \mathcal{E}_s + \lambda\mathcal{E}_c(w) \tag{4.43}$$

Dove il primo termine è l'espressione classica della funzione di costo, mentre il secondo termine rappresenta il temine di regolarizzazione che dipende

dal prodotto tra il parametro di regolarizzazione λ e un termine che è funzione dei pesi sinaptici della rete. Il termine $\mathcal{E}_c(w)$ si differenzia in base all'approccio utilizzato. Il primo approccio a cui si fa riferimento è detto *weight decay*. Per tale approccio risulta:

$$\mathcal{E}_c\,(w) = ||w||^2 = \sum_i w_i^2 \tag{4.44}$$

Il termine di regolarizzazione in questo caso ha la funzione di forzare a zero i pesi sinaptici meno attivi della rete; infatti, elevando al quadrato dei pesi che sono già di per se piccoli, il loro valore tende a zero. È come se stessimo disattivando i neuroni meno sollecitati dai pattern di *training* applicando di fatto un 'potatura' o *pruning* alla rete. Questa regolarizzazione è anche conosciuta come regolarizzazione di Ridge.

Nel secondo approccio, chiamato *weghit elemination*, il termine $\mathcal{E}_c\,(w)$ si esprime come segue:

$$\mathcal{E}_c\,(w) = \sum_i \frac{\left(\frac{w_i}{w_0}\right)^2}{1 + \left(\frac{w_i}{w_0}\right)^2} \tag{4.45}$$

Se $\frac{w_i}{w_0} \ll 1$, allora il contributo i-esimo ad $\mathcal{E}_c\,(w)$ tende a zero e il neurone è da considerarsi 'disattivo'. Perciò deve essere eliminato dalla rete per migliorare le performance. La scelta ottimale del parametro di regolarizzazione è affidata ai *model selection criteria*, che saranno discussi nel paragrafo 4.4.5. Tale scelta è condotta in modo da risolvere il problema del giusto bilanciamento tra bias e varianze del modello.

Regolarizzazione di Ridge

Una delle forme più comuni, nonché più semplici e computazionalmente più leggere di regolarizzazione è la regolarizzazione di Ridge. La funzione di costo assume la seguente forma:

$$S = \frac{1}{2}\sum_{i=1}^{n}\left(d_i - F_\lambda(x_i)\right)^2 + \sum_{j=1}^{n}\lambda\; w_j^2 \tag{4.46}$$

Tale metodo coincide con il *weight decay* discusso nel paragrafo precedente. Per evitare di ripetere le stesse considerazioni si rimanda a tale paragrafo.

Oltre alla regolarizzazione di Ridge convenzionale esiste anche la regolarizzazione di Ridge locale che consiste nello scegliere un parametro di regolarizzazione diverso relativamente ad ogni neurone della rete. La funzione di costo diventa:

$$S = \frac{1}{2}\sum_{i=1}^{n}(d_i - F_\lambda(x_i))^2 + \sum_{j=1}^{n}\lambda_j \; w_j^2 \tag{4.47}$$

Questo tipo di regolarizzazione di Ridge controlla localmente il bilanciamento tra i bias e la varianza, perciò risulta essere più performante per problemi la cui distribuzione dei dati nell'iperspazio degli input è altamente irregolare.

Il significato di λ

Introdurre un termine di regolarizzazione nella funzione di costo equivale ad introdurre un bias aggiuntivo nell'equazione 4.19 o 4.47 che penalizza i pesi più piccoli a favore di quelli più grossi: la soluzione risultante 'disattiva' le unità con i pesi più piccoli stimolando ancora di più le unità con i pesi più grossi.

Ciò vuol dire che una soluzione con λ elevato è più 'rozza' perché coinvolge solo i pesi più grandi. In questo modo si influenza non solo il bias della rete, ma anche la varianza perché disattivando alcuni pesi si riduce il numero di paramenti effettivi della rete. Infatti, nel caso di rete regolarizzata il numero di gradi di libertà della rete è minore del numero di unità presenti nella rete stessa e di solito si calcola come:

$$\gamma = p - traccia(P) \tag{4.48}$$

Dove p sono il numero delle osservazioni temporali relativi agli *input pattern* e P è il progettone (che dipende da λ).

$$\widehat{\sigma^2} = \frac{\hat{S}}{p - \gamma} \tag{4.49}$$

Per riassumere, una rete regolarizzata con λ elevato riduce il numero di unità attive e aumenta la varianza di queste ultime producendo una soluzione più 'ruvida' (più oscillante) ma meno 'infedele' mentre, una soluzione con λ piccolo è meno 'ruvida' ma più 'infedele' perché coinvolge anche i pesi più

piccoli. Richiamando l'espressione 4.19, per la scelta del λ ottimale bisogna trovare un compromesso tra:

- Il livello di 'infedeltà' della soluzione misurato dal termine $\sum_{i=1}^{l} \left(f\left(x_i\right) - \hat{f}_\lambda\left(x_i\right)\right)^2$
- Il livello di 'ruvidezza' della soluzione misurato dal termine $\left\|D\widehat{f_\lambda}\left(x\right)\right\|$

Dunque, l'introduzione del parametro di regolarizzazione va a modificare il bilanciamento originale tra bias e varianza della rete, influenzando la soluzione finale.

4.4.5 Model selection criteria applicati all'*overfitting*

I *model selection criteria* sono delle strategie di addestramento della rete che si basano sulla predizione dell'errore di generalizzazione alla rete per sfruttare questa informazione in modo intelligente scegliendo l'architettura, tra tante architetture simili, che ottimizza (in termini di errore di generalizzazione, la rete stessa). La logica è quella di analizzare delle architetture simili, dove per simili si intende architetture in cui si fanno variare dei parametri con lo scopo di comprendere come tali parametri influenzano le performance della rete in fase di test. Il cuore di questi metodi, dunque, si fonda su una corretta stima dell'errore di generalizzazione della rete e lo scopo è quello di selezionare una configurazione che minimizzi tale errore: da qui il nome di *model selection criteria*. L'errore di generalizzazione della rete è definito in termini probabilistici come la probabilità che dato un set di input di test, il valore predetto della rete sia differente da quello desiderato.

$$\epsilon\left(F\right) = P\left(F\left(x\right) \neq d\right) \tag{4.50}$$

dove F è la funzione di approssimazione della rete, x è un vettore di input scelti nell'intervallo $[0, 1]$ con una distribuzione incognita di probabilità, d è un vettore di output randomici nell'intervallo $[0, 1]$. Tra i *model selection criteria* i più utilizzati ci sono il metodo *Cross-Validation* (CV) e il metodo *Generalized Cross Validation* (GCV).

La formulazione più generale del metodo di Cross-validation prevede che il set di dati di *training* sia diviso in due parti: una adibita alla stima del modello più performante, l'altra per la validazione e quindi il test. Il sottoset di dati di *training* viene utilizzato per selezionare il modello migliore : si fanno cioè variare i pesi (e quindi le varianze) e i bias della rete e si confrontano

le varie configurazione in modo da selezionare quella di minimo rischio, cioè quella che minimizza l'errore di generalizzazione misurato sul suddetto sotto-set di dati training. I restanti dati di *training* vengono utilizzati per testare e validare il modello precedentemente selezionato. Di solito il procedimento che si segue è chiamato metodo dell'*early stopping* che alterna periodi di selezione (*training*) a periodi di validazione (*testing*) facendo variare i parametri durante i periodi di selezione e fissandoli in quelli di validazione. Il ciclo si ripete più volte. La curva che stima il livello di apprendimento della rete mostra che in realtà, a differenza del *training*, per il test l'apprendimento migliora fino ad un certo numero di iterazioni oltre il quale la rete continua ad apprendere solo informazioni legate al rumore dei dati, le quali peggiorano la soluzione.

Una variabile del metodo CV chiamata *Leave One Out Cross Validation*(LOO CV) consiste nel dividere il *training* set in un certo numero di sottoinsiemi e di allenare la rete su tutti i sottoinsiemi eccetto uno che viene lasciato fuori e utilizzato per la validazione. Questa procedura è ripetuta lasciando alternativamente fuori tutti i sottoset. La stima dell'errore di generalizzazione si ottiene calcolando l'errore quadratico medio su tutti i sottoset di validazione ed effettuandone una media rispetto al numero di partizioni del set di training. Per il metodo LOO la varianza dell'errore di predizione si calcola in funzione del vettore degli ouput approssimati e del progettone.

$$\hat{\sigma}^2_{LOO} = \frac{\hat{y}^T P \; diag\,(P)^{-2} \, P\hat{y}}{p} \tag{4.51}$$

con p numero degli esperimenti effettuati. Il metodo GCV è molto simile al LOO ma introduce delle correzioni nel calcolo della media dell'errore quadratico medio, risultando computazionalmente più vantaggioso.

$$\hat{\sigma}^2_{GCV} = \frac{\hat{y}^T p \; P^2 \hat{y}}{(trace\,(P))^2} \tag{4.52}$$

I *model selection criteria* possono essere utilizzati, oltre che per allenare la rete facendo variare bias e pesi della stessa, anche per altri scopi come calcolare il valore ottimale del parametro di regolarizzazione che minimizza l'errore di generalizzazione: si confrontano più reti neurali in cui si fa variare il parametro di regolarizzazione in modo da selezionare quella che restituisce le performance migliori [8].

4.4.6 Forward selection e backward selection

Una strategia alternativa per ridurre il problema dell'*overfitting* nelle reti RBF prevede l'utilizzo dei cosiddetti metodi *subset selection*. Come comprensibile dal nome, i *subset selection* confrontano delle reti costituite da differenti sottoset di funzioni basiche all'interno di un set prestabilito. Considerare diversi sottoinsiemi di funzioni di base equivale a confrontare delle reti con differente numero di unità nascoste. Dunque, la filosofia di questi metodi è quella di far variare il numero delle unità nascoste e di valutare come variano le performance della rete, stimate con uno dei *model selection criteria*. Di solito si fa riferimento alla GCV per la determinazione dell'errore di generalizzazione, il quale poi sarà utilizzato proprio come termine di paragone per confrontare le diverse reti. La logica di base può essere quella di *growing* o quella di *pruning*. Si parte da una rete con un'unica unità nascosta per poi aumentare il numero delle unità verificando il comportamento della rete: in questo caso il metodo prende il nome di *forward selection*. Nel secondo caso si parte da un numero massimo di neuroni e si procede eliminando un neurone per volta dalla rete: questo procedimento viene chiamato *backward selection*. L'aggiunta di neuroni (nel caso di *forward selection*) si ferma quando l'errore di generalizzazione della rete inizia ad aumentare, o parimenti quando arriva per la prima volta al di sotto del goal (detto anche valore di *threshold*). I centri e i raggi delle unità della rete sono fissati durante il processo. Questi metodi presentano due vantaggi principali :

- Il numero di neuroni non deve essere fissato a priori.
- I requisiti computazionali sono bassi, per tale ragione l'addestramento è abbastanza veloce.

È possibile ottenere dei notevoli miglioramenti in termini di velocità dell'algoritmo combinandoli con l'OLS: in questo modo ogni nuova colonna aggiunta alla matrice di design della rete (Φ) durante il processo di *growing* risulta essere ortogonale a quelle già presenti. Ciò semplifica l'equazione per l'aggiornamento dell'errore quadratico medio di predizione. Le performace di questo algoritmo possono essere ancora migliorate introducendo la regolarizzazione di Riedge.

4.4.7 La regressione del 'kernel' per le RBF

In questo paragrafo si propone un approccio leggermente differente da quello utilizzato nella regolarizzazione di Tichonov per determinare l'architettura classica delle funzioni radiali basiche. Questo approccio si basa sull'inferenza statistica per il calcolo della funzione di approssimazione e conduce a dei risultati leggermente differenti rispetto a quelli visti nell'approccio precedente. Riprendendo quanto visto nel paragrafo sull'interpretazione statistica dell' apprendimento, ricordiamo l'espressione per esprimere la funzione di regressione

$$f(x) = E[y|x] \tag{4.53}$$

La regressione del Kernel è un modo che può essere utilizzato per determinare tale funzione di regressione sulla base di un set di input randomici nello spazio degli input. La tecnica è quella di esprimere la relazione tra la funzione di regressione $f(x)$ e le funzioni di densità di probabilità di x e del legame tra x e y ovvero $f_{x,y}(x,y)$ e $f_x(x)$. Utilizzando la formula del valore atteso di una variabile casuale si ottiene:

$$f(x) = \frac{\int y f_{x,y}(x,y)}{f_x(x)} \tag{4.54}$$

A questo punto si utilizza uno stimatore non lineare (lo stimatore di Partzen- Rosenblatt) che permetta di effettuare una stima delle PDF prima menzionate come combinazine lineare di funzioni kernel.

$$f(x) = \frac{\int y \hat{f}_{x,y}(x,y)}{\hat{f}_x(x)} \tag{4.55}$$

Integrando la 4.55 si ricava la funzione di regressione stimata :

$$F(x) = \hat{f}(x) = \frac{\sum_{i=1}^{N} y_i \; K(\frac{x-x_i}{h})}{\sum_{j=1}^{N} K(\frac{x-x_j}{h})} \tag{4.56}$$

In parole povere, possiamo calcolare i valori desiderati di output noto un set di input x e un set di output y utilizzando la stima approssimata della funzione di regressione. Dove il termine

$$W_{N,i}(x) = \frac{\sum_{i=1}^{N} y_i \; K(\frac{x-x_i}{h})}{\sum_{j=1}^{N} K(\frac{x-x_j}{h})} \tag{4.57}$$

si chiama l'estimatore di regressione di Nadayara-Watson (NWRE). Nel caso in cui si considerino delle funzioni di Kernel radiali basiche allora tale approccio conduce a delle funzioni radiali basiche normalizzate.

$$\psi\left(x, x_i\right) = \frac{\sum_{i=1}^{N} K(\frac{||x-x_i||}{h})}{\sum_{j=1}^{N} K(\frac{||x-x_j||}{h})} \tag{4.58}$$

E la funzione di regressione può essere stimata come

$$F\left(x\right) = \hat{f}\left(x\right) = \sum_{i=1}^{N} y_i \ \psi\left(x, x_i\right) \tag{4.59}$$

Ciò equivale a creare una rete neurale con architettura equivalente alle classiche reti neurali RBF che utilizza delle unità nascoste le cui funzioni basiche sono delle funzioni Kernel normalizzate (gaussiane normalizzate o NWRE) e che utilizza dei pesi, nel *summation layer*, pari proprio ai valori di output osservabili.

$$F\left(x\right) = \hat{f}\left(x\right) = \sum_{i=1}^{N} y_i \ \psi\left(x, x_i\right) = \sum_{i=1}^{N} w_i \ \psi\left(x, x_i\right) \tag{4.60}$$

Nel caso particolare di distribuzione gaussiana multivariata si ottiene

$$F\left(x\right) = \frac{\sum_{i=1}^{N} w_i exp\left(\frac{||x-x_i||}{2\sigma^2}\right)}{\sum_{j=1}^{N} exp\left(\frac{||x-x_j||}{2\sigma^2}\right)} \tag{4.61}$$

La sostanziale differenza con l'approccio di Tichonov consiste nelle basi utilizzate per il calcolo della funzione di approssimazione: in questo caso sono normalizzate rispetto ad un temine legato alla funzione densità di probabilità degli input. In più la loro somma ci restituisce l'unità, condizione non necessariamente soddisfatta per le funzioni di Green viste nell'approccio di Tichonov. Spesso le reti neurali costruite seguendo questa logica sono ancora più performanti delle reti RBF convenzionali purché siano unite a delle specifiche tecniche di regolarizzazione.

Capitolo 5

Progettazione di una rete neurale RBF per la stima di AoA

In questo capitolo verrà presentata una procedura generica per la progettazione di una rete neurale RBF adibita alla stima dell'angolo di attacco su un aeroplano. Il tutto nell'ottica dello sviluppo di un sensore sintetico che integri nella sua architettura proprio la suddetta rete. Come già detto nei capitoli iniziali, la progettazione di una rete efficiente per il calcolo di AoA è un tema centrale nello sviluppo di apparati sensoristici all'avanguardia, che mirano a sostituire i sistemi ridondanti ADS tradizionali con dei sistemi parzialmente basati su sensori sintetici, meno costosi e meno ingombranti

Si sfrutteranno, dunque, tutte le conoscenze teoriche acquisite nel capitolo precedente con la finalità di comprendere il funzionamento degli algoritmi commerciali che saranno utilizzati per l'apprendimento della rete che si sta progettando.

In particolare, la rete sarà addestrata tramite:

- L'algoritmo EMRAN (*Extended Minimal Resource Allocating Network*), che prevede un addestramento on line.
- Toolbox di Matlab, che prevede un addestramento in *batch*

Ulteriori dettagli sui suddetti algoritmi saranno forniti all'interno del capitolo.

5.1 Analisi dinamica dell'aeroplano e vettore di input

La qui presente analisi dinamica dell'aeroplano è inserita al fine di esplicitare le variabili da cui dipende l'angolo di attacco e determinare il vettore di input da inserire nella rete neurale da progettare. Le equazioni del moto si ottengono sulla base delle forze aerodinamiche, propulsive del vento e considerando i momenti che agiscono attorno all'aereo. Dallo studio della dinamica dell'aereo si ricavano le equazioni che governano la dinamica longitudinale e quella latero-direzionale dell'aereo. In condizione di piccoli disturbi il problema del controllo longitudinale può essere trattato in maniera disaccoppiata rispetto al problema del controllo latero-direzionale. Nei casi di applicazione reale, in presenza di disturbi rilevanti e manovre dell'aereo, non è possibile trattare indipendentemente i due problemi, dunque se i comandi dell'aereo (dati) sono fissi, tutte le forze i momenti dipendono dall'orientazione dell'aereo rispetto al flusso di aria cioè da tre quantità:

- La TAS
- L'AoA
- L'AoS

Gli angoli di attacco e di derapata sono comunemente indicati come angoli aerodinamici. Il vettore di stato che governa la dinamica dell'aereo è il seguente:

$$X^T = [u, v, w, \phi, \theta, \psi.p, q, r, p_N, p_E, H] \tag{5.1}$$

Dove H è la quota dell'aeromobile nel sistema di riferimento NED, mentre l'altezza si riferisce di solito al valore misurato dal baro-altimetro. È bene precisare che per sistema di riferimento NED si intende il sistema *north-east-down* con origine su un punto qualsiasi della superficie terrestre, per cui l'asse nord è diretto lungo il meridiano verso il polo nord, l'asse est è diretto lungo il parallelo e l'asse down è diretto ortogonalmente ai primi due, verso il centro della terra. Il sistema NED è, in teoria, un sistema di riferimento non inerziale perché ruota con velocità angolare pari a quella della terra. Tuttavia, limitatamente alle applicazioni di interesse in ambito aeronautico per le quali l'intervallo di tempo è molto ridotto rispetto al periodo di rotazione terrestre, il sistema NED può essere considerato inerziale.

Specificando le relazioni tra le velocità in assi body e gli angoli aerodinamici è possibile eliminare u,v,w dal vettore di stato ed esplicitare nelle grandezze sopracitate V_{TAS} (che si indicherà per semplicità con V), α e β.

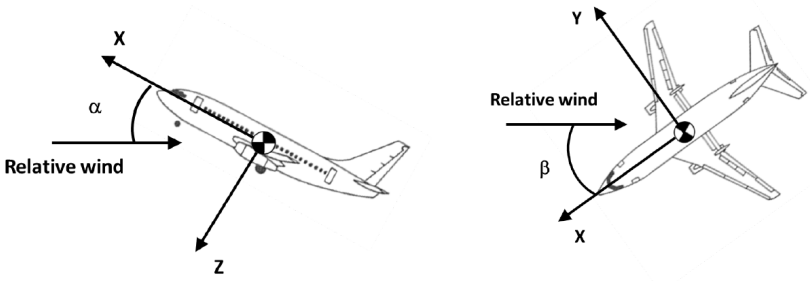
$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = V \begin{bmatrix} cos\alpha cos\beta \\ sin\beta \\ sin\alpha cos\beta \end{bmatrix} \tag{5.2}$$

o in alternativa:

$$V = \sqrt{u^2 + v^2 + w^2} \tag{5.3}$$

$$\alpha = \arctan\left(\frac{w}{u}\right) \tag{5.4}$$

$$\beta = \arctan\left(\frac{u}{\sqrt{u^2 + v^2}}\right) \tag{5.5}$$



(a) angle of attack (b) angle of sideslip

Figura 5.1: Visualizzazione grafica degli angoli aerodinamici.

Allora il vettore di stato può essere così modificato

$$X^T = [V, \alpha, \beta, \phi, \theta, \psi, p, q, r, p_N, p_E, H] \tag{5.6}$$

Il sistema che lega le forze aerodinamiche ai parametri dell'aeroplano può essere scritto sinteticamente come segue:

$$\begin{cases} \dot{X} = f(X, U, v, w) \\ Y = g(X, U, v, w) \end{cases} \tag{5.7}$$

dove U rappresenta un vettore contenente: un parametro legato al numero di giri del motore (δ_n); i comandi del pilota quali la deflessione dell'equilibratore (δ_e), la deflessione del timone (δ_r), la deflessione dell'alettone (δ_a) e la deflessione dell'ipersostentatore (δ_f).

$$U^T = [\delta_e, \delta_r, \delta_a,\ \delta_f, \delta_n] \tag{5.8}$$

Al sistema vanno aggiunte le equazioni che esprimono la variazione degli angoli aerodinamici in funzione delle componenti della velocità, perché risulti risolvibile in ogni istante di tempo. Per ricavarle bisogna differenziare le equazioni 5.4 e 5.5. In particolare, per l'angolo di attacco si ottiene:

$$\dot{\alpha} = \frac{u\ \dot{w} - w\ \dot{u}}{u^2 + w^2} \tag{5.9}$$

Dove inserendo le espressioni di u e w mostrate in precedenza e riarrangiando si ottiene:

$$\dot{\alpha} = \frac{\dot{w}\ cos\alpha - sin\alpha\ \dot{u}}{V cos\beta} \tag{5.10}$$

Il passaggio successivo consiste nel sostituire all'interno della 5.9 e della 5.10 le espressioni delle velocità angolari nel riferimento body in funzione della accelerazione assiale e dell'accelerazione di gravità 5.11.

$$\begin{aligned} \dot{u} &= rv - qw - gsin\theta + \tfrac{F_x}{m} \\ \dot{v} &= -ru + pw + gsin\phi cos\theta + \tfrac{F_y}{m} \\ \dot{w} &= qu - pv + gcos\phi cos\theta + \tfrac{F_z}{m} \end{aligned} \tag{5.11}$$

Gli ultimi termini nelle tre equazioni sono accelerazioni inerziali ($F = ma_i$) e possono essere espresse rispetto all'accelerazione gravitazionale, prendendo il nome di accelerazioni inerziali assolute

$$n_x = \frac{F_x}{mg} \tag{5.12}$$

Questo termine, misurato sull'aereo dagli accelerometri comprende le accelerazioni inerziali assolute relative agli effetti aerodinamici, alla gravità e agli effetti propulsivi. In questa trattazione si preferiscono trascurare gli effetti gravitazionali su questi termini allo scopo di evidenziare meglio il legame

con i parametri di assetto dell'aereo.

$$\begin{aligned}\dot{\alpha} = \frac{1}{Vcos\beta}\{&(qu - pv + g_0 cos\phi cos\theta + n_z\prime)\ cos\alpha \\ &-sin\alpha\,(rv - qw - g_o sin\theta + n'_x)\}\end{aligned} \tag{5.13}$$

Infine, sostituendo le relazioni 5.2 si ottiene:

$$\begin{aligned}\dot{\alpha} = \frac{1}{Vcos\beta}\{&[V\ \ (qcos\alpha cos\beta - psin\beta) + n'_z + g_0\ cos\phi cos\theta]\, cos\alpha \\ &-sin\alpha\,[V(rsin\beta - qsin\alpha cos\beta) - g_o sin\theta + n'_x]\}\end{aligned} \tag{5.14}$$

Lo stesso procedimento si può seguire per β il che porta al risultato seguente:

$$\begin{aligned}\dot{\beta} = \frac{1}{V}\{&(-n'_x + g_o sin\theta)\, cos\alpha sin\beta + \left(n'_y + g_o cos\theta sin\phi\right) cos\beta \\ &+ (-n'_z - g_o cos\theta cos\phi)\, sin\alpha sin\beta\} + psin\alpha - rcos\alpha\end{aligned} \tag{5.15}$$

Il prossimo passaggio consiste nell'esprimere le accelerazioni inerziali aerodinamiche e propulsive n'_i in funzione dei parametri aerodinamici, di propulsione e dei comandi. Queste relazioni dipendono dal particolare velivolo scelto ma sono ricavabili per ogni velivolo.

Le relazioni finali di AoA e AoS, che si ricavano combinando le equazioni 5.14 e 5.15 e le relazioni di n'_i, dipendono dai seguenti parametri:

$$\alpha = f_\alpha(q_c, n_x, n_y, n_z, \beta, \theta, \phi, p, q, r, \delta_e, \delta_r, \delta_a,\ \delta_f, \delta_n) \tag{5.16}$$

$$\beta = f_\beta(q_c, n_x, n_y, n_z, \alpha, \theta, \phi, p, q, r, \delta_e, \delta_r, \delta_a,\ \delta_f, \delta_n) \tag{5.17}$$

Dove è possibile eliminare la reciproca dipendenza tra i due parametri aerodinamici inserendo un'equazione all'interno dell'altra ed espicitando l'angolo aerodinamico di interesse. Per AoA si ottiene:

$$\alpha = f_\alpha(q_c, n_x, n_y, n_z, \theta, \phi, p, q, r, \delta_e, \delta_r, \delta_a,\ \delta_f, \delta_n) \tag{5.18}$$

Ciò è di fondamentale importanza per la progettazione della rete neurale in quanto la sua architettura sarà costruita considerando come vettore di input proprio un vettore che contiene tutte le variabili da cui α dipende. Dunque, il vettore di input della rete sarà

$$\mathbf{X}^T = [q_c, n_x, n_y, n_z, \theta, \phi, p, q, r, \delta_e, \delta_r, \delta_a, \ \delta_f, \delta_n] \tag{5.19}$$

Nel presente lavoro δ_n non verrà preso in considerazione.

$$\mathbf{X}^T = [q_c, n_x, n_y, n_z, \theta, \phi, p, q, r, \delta_e, \delta_r, \delta_a, \ \delta_f] \tag{5.20}$$

In alternativa la q_c può essere sostituita dalla TAS. L'interscambiabilità è possibile perché la q_c può essere confusa con la CAS a meno di una costante in quanto l'anemometro è calibrato proprio utilizzando la legge di Bernoulli($CAS = \sqrt{2q_c/\rho_o}$, con ρ_o densità dell'aria a SL). A sua volta la CAS può essere confusa con la TAS se le variazioni di densità sono trascurabili ($TAS = CAS/\sqrt{\rho/\rho_o}$), come nel caso in esame. Il vettore di input può in un secondo momento essere ridotto effettuando delle analisi di sensibilità della rete rispetto a ciascun input. Questo procedimento ha lo scopo di eliminare i pattern di input che hanno poca influenza sull'angolo d'attacco e di alleggerire il costo computazionale della rete [2]. Questa analisi, tuttavia, non sarà compresa nel corrente lavoro.

5.2 Criteri di scelta delle manovre: trattazione e campionamento dei dati di *training* e di test

Per creare una rete neurale che sia quanto più performante possibile è fondamentale trattare preliminarmente i dati in ingresso alla rete sia in fase di test che in fase di *training*.

La rete può essere addestrata e validata o su un set di dati provenienti da una ambiente reale, quindi catturati da sensori durante una serie di test in volo oppure su un set di dati provenienti da un ambiente simulato. I criteri per selezionare il giusto set di dati *training* e test in ambiente reale o simulato sono gli stessi, ma ovviamente i risultati sono differenti. Di solito le performance della rete peggiorano in ambiente reale a causa della presenza

di un elevato rumore di sottofondo, di fenomeni legati alla turbolenza e alla natura fortemente dinamica delle manovre effettuate in ambiente reale.

La prima modalità d trattazione dei dati passa dalla selezione delle manovre più adatte relativamente alla fase di *training* e di quelle più adatte alla fase di test. La strategia per la selezione delle manovre di *training* è quella di scegliere delle manovre che ricoprano bene o male tutto 'inviluppo di volo' del velivolo. I criteri utilizzati sono:

- Dinamica latero-direzionale e longitudinale disaccoppiate. Questa caratteristica può essere soddisfatta appieno solo se le manovre di *training* vengono create ad hoc attraverso l'utilizzo di un simulatore di volo. In problemi reali è più difficile disaccoppiare completamente le due dinamiche pur ammettendo che siano soddisfatte le condizioni di piccole perturbazioni.
- Velocità di poco variabile
- Contenute entro certi limiti dell'inviluppo che non vanno oltrepassati

Ovviamente per quanto riguarda le manovre reali queste condizioni sono più difficilmente realizzabili. Ad esempio, accade spesso che il set di dati fuoriesca dall'inviluppo ristretto di volo peggiorando le performance della rete [9].

È stato dimostrato una distribuzione uniforme dei dati su tutto l'inviluppo di volo riesce ad evitare il problema di minimo locale e a migliorare le performance della rete anche nel caso in cui il set di dati fuoriesca dai limiti dell'inviluppo ristretto. Perciò per migliorare le prestazioni la strategia è quella di infittire i dati di *training* nelle zone dell'iperspazio dove la densità è troppo bassa o troppo elevata. Per essere più precisi, si è notato che la densità di dati risulta essere molto più elevata durante le manovre dinamiche piuttosto che in quelle stazionarie. Questo porta ad un peggioramento dei risultati che può essere risolto mettendo appunto delle tecniche che riescano automaticamente a ridurre la popolazione di dati nelle zone relative a manovre dinamiche e ad aumentare la popolazione nelle porzioni di iperspazio relative a manovre quasi statiche [10].

Le tecniche di trattazione dei dati descritte si sono dimostrate molto efficaci per le MLP. L'applicazione di tali tecniche potrebbe non essere necessaria per le RBF: nel capitolo precedente si sono evidenziate le ottime

capacità delle reti RBF come interpolatori, grazie alla capacità di separabilità lineare dei pattern di input(come enunciato dal teorema di Cover) che le rende estremamente semplici dal punto di vista architettonico, e alla natura intrinseca di regolarità (Tichonov). Queste caratteristiche permettono alle RBF di ottenere delle ottime performance per l'AoA (attestando l'errore al di sotto di una soglia limite) anche per set di dati in ambiente reale non particolarmente trattati.

Per quanto riguarda, invece le manovre di test, queste sono selezionate senza seguire delle limitazioni particolari. Lo scopo è quello di testare la capacità della rete di generalizzare in condizioni più estreme e dinamiche delle condizioni in cui la rete è stata addestrata. Le caratteristiche principali delle manovre di test sono:

- Accoppiamento della dinamica latero-direzionale con quella longitudinale
- Velocità compresa in un range che varia tra la velocità di stallo, senza flap, alla massima velocità ottenibile in picchiata
- Massima spinta durante le manovre di *pushover* e *pullup*
- Volo in condizione *unsteady*

Un altro aspetto su cui vale la pena soffermarsi è il campionamento dei dati di manovra che siano essi reali o ottenuti da un simulatore. Spesso è necessario effettuare un campionamento sulle manovre per alleggerire il costo computazionale in fase di addestramento, vista l'enorme mole di dati con cui si ha a che fare durante queste applicazioni. Infatti, se si considera che gli input alla rete sono 13 e che ogni vettore di input contiene circa 200000 punti, ciò vorrebbe dire che la rete dovrebbe processare più di due milioni di dati con un costo computazionale molto elevato. Per alleggerire i dati si necessita dell'informazione sulla frequenza di campionamento dei vettori dei dati acquisiti in volo. I dati alleggeriti dovranno essere campionati in modo tale da risultare compatibili con frequenza di campionamento originale. Questo assicura che il risultato non peggiori eccessivamente riducendo la frequenza di campionamento.

5.3 Valutazione del rumore di sottofondo

In ambiente reale i dati misurati dai sensori sono corrotti da un rumore di sottofondo che è legato a quattro fondamentali fenomeni:

- Raffiche di vento
- Turbolenza
- rumore elettronico
- vibrazioni indotte dal motore

Il rumore di sottofondo può costituire un aspetto particolarmente critico per le performance di una rete neurale. Un eccessivo aumento del rumore può degradare oltremisura i risultati facendo in modo che essi fuoriescano dai limiti consentiti. Le misurazioni inerziali possono essere influenzate da fattori esterni che non sono legati alle sole velocità o accelerazioni angolari: innanzitutto bisogna considerare le vibrazioni strutturali dell'aeromobile le quali sono costituite da uno sfondo a banda larga con picchi a banda stretta sovrapposti. Lo spettro di fondo deriva principalmente dal motore, combinato con molti componenti periodici di livello inferiore, dovuto agli elementi rotanti (motori, cambi, alberi, ecc.) associati all'elica.

La vibrazione rilevata a bordo è strettamente correlata con il tipo di materiale che compone la struttura dell'aeromobile e anche col particolare metodo utilizzato per l'isolamento delle vibrazioni strutturali. Allo stesso modo un'altra fonte di corruzione dei dati potrebbe consistere nell'imperfezione del modello matematico e nella relativa discrepanza che si verifica con il modello reale del velivolo. Ciò è vero soprattutto quando la frequenza di campionamento del sensore è vicina alle prime frequenze di modo strutturale, come nei corpi snelli. Da quanto detto, risulta che la caratterizzazione del rumore è strettamente dipendente dalla tipologia di aeromobile considerato, pertanto una trattazione di validità generale che individui le più corrette tecniche per l'isolamento del rumore spesso non è possibile.

5.4 Architettura di base della rete RBF

In questa sezione si presenta l'architettura di base della rete RBF selezionata per stimare l'angolo di attacco AoA insieme alla logica di funzionamento della rete. Nel paragrafo 5.1 si è trovata la dipendenza tra l'angolo di attacco e le

principali variabili caratteristiche della dinamica dell'aereo. Da quell'analisi si è ricavato il vettore di input della rete :

$$\mathbf{X}^T = [q_c, n_x, n_y, n_z, \theta, \phi, p, q, r, \delta_e, \delta_r, \delta_a,\ \delta_f] \tag{5.21}$$

Dunque, i dati di input disponibili sono i seguenti:

- $q_c(t)$, pressione dinamica
- $n_x(t)$,accelerazione body longitudinale
- $n_y(t)$,accelerazione body laterale
- $n_z\,(t)$, accelerazione body verticale
- $\theta(t)$, angolo di beccheggio
- $\phi\,(t)$, angolo di rollio
- $p(t)$, velocità angolare di rollio
- $q(t)$ velocità angolare di beccheggio
- $r(t)$ velocità angolare di imbardata
- $\delta_e(t)$ deflessione dell'equilibratore
- $\delta_r(t)$ deflessione del timone
- $\delta_a(t)$ deflessione dell'alettone
- $\delta_f(t)$ deflessione del flap

Per quanto detto nel paragrafo 5.1, la q_c può essere confusa con la TAS. Al vettore di input bisogna però aggiungere un ulteriore informazione che è quella dell'angolo di attacco calcolato dalla teoria linearizzata $\hat{\alpha}$ che si calcola come

$$\hat{\alpha} = \theta - \gamma \tag{5.22}$$

dove θ è un dato del problema mentre γ si calcola come:

$$\gamma = \arctan\left(\frac{w}{V}\right) \tag{5.23}$$

Dunque, si ricava

$$\hat{\alpha} = \theta - \arctan\left(\frac{w}{V}\right) \tag{5.24}$$

L'angolo di attacco stimato dalla rete può essere calcolato come somma dell'angolo di attacco lineare più un certo incremento $\Delta\alpha$ che rappresenta di quanto il valore dell'AoA stimato dalla rete si discosta dal valore lineare.

$$\alpha_{NN} = \hat{\alpha} + \Delta\alpha_{NN} \tag{5.25}$$

Il valore dell'incremento rappresenta l'output della rete RBF da stimare calcolato a partire dal valore desiderato dell'angolo di attacco fornito alla rete in fase di *training*

$$\Delta\alpha = \alpha - \hat{\alpha} \tag{5.26}$$

In fase di addestramento si fornisce alla rete $\Delta\alpha$ come output desiderato e la rete stima il valore approssimato di $\Delta\alpha$ cioè $\Delta\alpha_{NN}$. Poi sfruttando la 5.25, si ottiene α_{NN}. In definitiva si può affermare che la rete scelta ha 14 valori di input in ingresso (in ogni istante temporale) e un valore di output pari proprio ad $\Delta\alpha_{NN}$. La rete è classificabile nella categoria MISO ed è una rete ad uno strato nascosto. L'architettura di base infatti riprende quella delle reti RBF generalizzate discusse nel capitolo precedente. La generalizzazione tramite metodo di Galerkin è necessaria visto che la rete deve funzionare off line dovendo incamerare una grande mole di dati in ingresso.

Tale rete sarà caratterizzata da uno strato di input, uno strato nascosto non lineare e un *summation layer* di output cioè lineare. Il numero delle unità nascoste della rete è ignoto inizialmente e verrà determinato, come si vedrà, in maniera integrata con l'algoritmo di apprendimento.

Di seguito uno schema della rete RBF nella forma più generale, dove in realtà il numero di neuroni non è uguale al numero di input ma è ancora sconosciuto in questa fase.

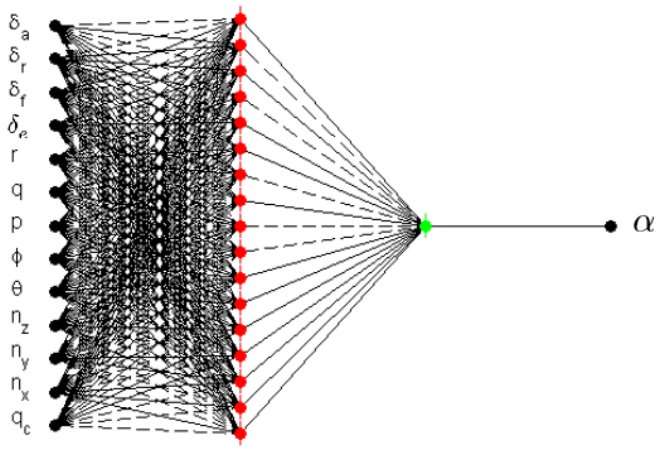


Figura 5.2: Visualizzazione schematica di una RBFnn MISO

Tale rete sarà poi da integrare nel sistema di controllo del velivolo. L'immagine 5.3 schematizza una possibile configurazione di un sensore virtuale integrato al FCS. In questo caso i valori di AoA e AoS sono misurati dalla rete neurale sfruttando i segnali di comando in uscita proprio dal FCS (relativi ad un istante $t-3$ rispetto all'istante corrente t e conservati all'interno di un'apposita memoria) e i dati inerziali (misurati all'istante t). Successivamente i valori degli angoli aerodinamici in uscita dalla rete vengono forniti in supporto alle leggi di controllo dell'FCS che determina il comando più adatto all'istante $t-1$ per il controllo del velivolo all'istante t.

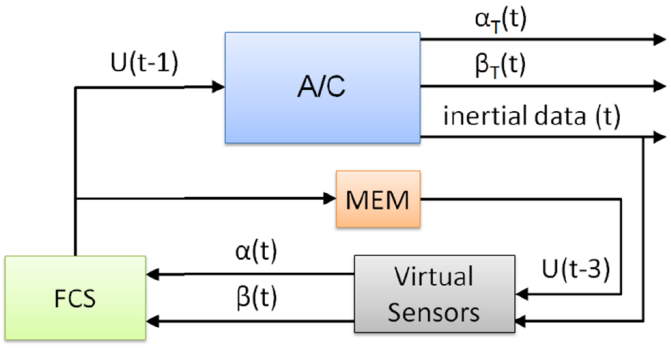


Figura 5.3: Integrazione del sensore virtuale con il FCS

5.5 Strategie di apprendimento per la RBFnn

5.5.1 Algoritmo EMRAN (*Extended Minimal Resource Allocating Network*)

Come accentato nel capitolo precedente, relativamente alla costruzione di architetture neurali RBF il problema era quello di trovare dei metodi che permettessero in modo automatico di ottimizzare l'architettura trovando il giusto numero di neuroni per la rete. Si pensò perciò di proporre una serie di algoritmi basati su una logica simile a quella *forward selection*. A tal proposito l'algoritmo RAN (*Resource Allocating Network*), proposto da Platt, ha consentito di risolvere il problema menzionato. Tale algoritmo aggiunge nuove unità basandosi sul grado di innovazione dei valori di input. I valori sono definiti 'innovativi' se si discostano molto dai centri già preesistenti e se l'errore della funzione approssimata è molto elevato. I pesi, i centri e le varianze sono calcolati con il LMS che viene applicato ricorsivamente ogni qual volta viene aggiunta un' unità. Successivamente un'altra soluzione fu proposta da Junge e Unbehauen. I due misero appunto un algoritmo conosciuto con il nome di *On-line Structural Adaptive Hybrid Learning*(ONSAHL) applicabile in fase di apprendimento on line. Tale algoritmo si basa su un procedimento ibrido che mira a trovare adattivamente (con un processo di *clustering* sulla base dell'errore calcolato) l'ampiezza e i centri delle RBF dei neuroni nascosti e a calcolare i pesi utilizzando l'algoritmo deterministico RLS (*Recoursive Least Square*). La principale differenza con l'algoritmo RAN precedentemente discusso, consiste nel fatto che solo i centri e le varianze delle RBF più vicine all'input sono aggiornate ad ogni step in modo da 'calzare' ai dati di *training*. Successivamente i pesi vengono aggiornati utilizzando, come detto con il RLS. È particolarmente utilizzato per risolvere problemi di identificazione non lineare di sistemi dinamici. Tale algoritmo fu successivamente migliorato da Kadirkamanathan e Niranjan sostituendo il metodo LMS con un metodo EKF (*Extended Kalman Filters*) che consentiva di ottenere una convergenza più veloce e di ridurre la complessità. La rete risultante prende il nome di RANEKF. Un ulteriore miglioramento dell'algoritmo RAN fu ottenuto introducendo delle tecniche di *pruning* per ottenere una rete neurale minimale e migliorare le prestazioni. Il suddetto algoritmo è conosciuto come MRAN (*Minimal Resource Allocating Network*). Infine l'applicazione di una logica di *growing* basata sul RMS (*Root Mean Square*) della rete ha con-

sentito di ridurre le oscillazioni correlate agli effetti del rumore di sottofondo misurato negli input data. Lo step successivo è stato quello di adattare l'algoritmo MRAN ad applicazioni in real time. Per qualsiasi applicazione pratica di algoritmi di identificazione di nuova concezione, è importante studiare l'implementazione in tempo reale dell'algoritmo. Nell'algoritmo MRAN, i parametri della rete, inclusi tutti i centri, le larghezze e i pesi nascosti del neurone, devono essere aggiornati in ogni fase. Ciò vuol dire che le dimensioni della matrice dei parametri della rete si accrescono all'aumentare dei neuroni nascosti e la struttura della rete RBF diventa più complessa dal punto di vista computazionale, il che rende MRAN poco adatto per applicazioni in tempo reale. Sulla base di questa analisi, è stata proposta un'estensione a MRAN denominata MRAN esteso (EMRAN) [11]. L'obiettivo è ridurre i costi computazionali dell'MRAN e realizzare uno schema per una rapida identificazione in linea. A tal fine, nell'algoritmo MRAN convenzionale è incorporata una strategia del "neurone vincitore". L'idea chiave dell'algoritmo EMRAN è che in ogni fase, solo i parametri correlati al neurone vincitore selezionato vengono aggiornati dall'EKF. EMRAN tenta di ridurre notevolmente i tempi di calcolo on-line ed evita problemi di memoria, mantenendo allo stesso tempo le buone caratteristiche di MRAN, ovvero minor numero di neuroni nascosti, minore errore di approssimazione. I risultati di numerose simulazioni mostrano che EMRAN è adatto per problemi di identificazione di sistemi non lineari con dinamiche mutevoli che si traducono in una rete con numerosi neuroni nascosti inattivi, poiché le dinamiche che hanno causato la loro creazione inizialmente diventano inesistenti. Per tali sistemi alcune unità nascoste, sebbene inizialmente attive, potrebbero successivamente contribuire con un piccolo contributo all'output di rete. Questo può essere il caso del volo di un aeroplano per cui le condizioni di volo sono soggette a variazioni dinamiche spesso non facilmente prevedibili.

Funzionamento dell'algoritmo

Si richiamano preliminarmente alcune generalità sulle reti GRBF (*Generalized Radial Basis Functions Neural Networks*).

Data una rete costituita da m unità nascoste, dato un vettore x di input costituito da $N_i * p$ punti (dove N_i si riferisce al numero di variabili input e p al numero di istanti temporali in cui ogni variabile di input è stata osservata),

il valore approssimato della rete dell'output desiderato è il seguente:

$$\hat{y} = \hat{f}(x) = F^{*}(x) = a_o + \sum_{i=1}^{m} w_i \phi(x - \mu_i) \tag{5.27}$$

dove w_i sono i pesi della rete, μ_i sono i centri della rete ed è stato introdotto un valore di bias a_0. Per evitare di fare confusione con gli indici utilizzati nella parte precedente del lavoro, in questo caso utilizzeremo l'indice k per indicare gli istanti di tempo. Gli indici relativi ai pattern di input e output rimarranno rispettivamente i e j. Si testerà una rete le cui RBF coincidono con delle gaussiane:

$$\phi_i(x) = exp\left(-\frac{||x - \mu_i||}{2\sigma_i^2}\right) \tag{5.28}$$

Dove σ è la varianza della gaussiana che si calcola come:

$$\sigma_i^2 = \frac{1}{p}\sum_{k=1}^{p}(x_k - x_i)^2 \tag{5.29}$$

Nell'algoritmo MRAN la rete comincia la rete comincia con zero unità nascoste per poi incrementarne il numero seguendo una specifica logica di *growing* che sarà descritta in seguito:

1. Per stabilire se un nuovo neurone deve essere aggiunto alla rete in presenza di un'osservazione $(x_k, f(x_k))$ è necessario che vengano soddisfatti i seguenti criteri:

$$e_k = \left|\left|f(x_k) - \hat{f}(x_k)\right|\right| > E_1 \tag{5.30}$$

$$RMSE = \sqrt{\sum_{j=k-(N_o-1)}^{k} \frac{e_j^2}{N_o}} > E_2 \tag{5.31}$$

$$\left|\left|f(x_k) - \mu_{nr}^k\right|\right| > E_3 \tag{5.32}$$

dove E_1, E_2 e E_3 sono dei valori prestabiliti e μ_{nr}^k è il centro dell'unità nascosta più vicina ad x_k. Il primo criterio è relativo all'errore di approssimazione che deve risultare al di sopra di una certa soglia perché si aggiunga un neurone; il secondo criterio invece è relativo alla

radice dell'errore quadratico medio di approssimazione; il terzo criterio assicura che il neurone da aggiungere sia abbastanza lontano dalle unità già esistenti. Questo step rappresenta la sostanziale differenza tra un algoritmo *foward selection* e l'algortimo MRAN : in questo caso, la strategia di aggiornamento dei pesi e di valutazione dell'errore non si basa un un *model selection critera* come il GCV, ma su 3 criteri che la rete utilizza contemporaneamente per capire se aggiungere una nuova unità o meno.

2. Se una nuova unità è aggiunta, i parametri per la nuova unità assumono i seguenti valori:

$$w_{k+1}^{i} = e_k \tag{5.33}$$

$$\mu_{k+1}^{i} = x_k \tag{5.34}$$

$$\sigma_{k+1}^{i} = \lambda \left\| x_k - \mu_{nr}^{k} \right\| \tag{5.35}$$

Con λ fattore di *overlap*, il cui significato verrà spiegato in seguito. Ogni qual volta una nuova unità viene aggiunta si passa allo step 4 per applicare una tecnica di *pruning* alla rete.

3. Se i criteri nel primo step non sono soddisfatti e nessuna unità è aggiunta allora bisogna procedere all'aggiornamento dei parametri della rete (centri, bias e pesi). Si passa prima per la determinazione della matrice del gradiente della funzione $f(x_k)$ cioè $\nabla_{\mathbf{W}} f(x_k)$, rispetto vettore dei parametri della rete misurato all'istante $k-1$. Successivamente si procede all'aggiornamento di tutti i parametri attraverso la regola del gradiente modificata oppure attraverso la regola del filtro di Kalman esteso (EKF). In riferimento alla prima delle due il vettore dei parametri della rete $\mathbf{W}$ è modificato secondo la seguente equazione:

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \eta \, \nabla_{\mathbf{W}} f(x_k) \; e_k + \eta \, \gamma \, \mathbf{W}_k \tag{5.36}$$

dove η è il rateo di apprendimento e γ è il fattore stabilizzante.

4. La tecnica di *pruning* applicata prevede che l'output normalizzato di ogni neurone sia osservato per un certo numero di osservazioni conse-

cutive ossia di input consecutivi. Se l'output normalizzato, dopo le N osservazioni consecutive, scende al di sotto di un valore soglia, allora il neurone è eliminato perché considerato 'inattivo'.

In questo modo si ottiene una rete ottimizzata in termini di numero di unità attive con notevoli capacità di approssimazione e grande velocità computazionale.

Per rendere tale algoritmo performante e adatto anche alle applicazioni on line si inserisce un'unica modifica che consiste nella definizione di un 'neurone vincitore' relativamente allo step di aggiornamento dei parametri della rete. Per 'neurone vincitore' si intende quel neurone per cui il centro della rispettiva gaussiana è più vicino all'input corrente x_k. Dunque, per meglio intendersi, solo i parametri relativi a tale neurone saranno aggiornati, risparmiando una quantità di memoria non indifferente: questa caratteristica rende EMRAN perfetto per applicazioni in real time [11] [12].

Schema della rete GRBF

Di seguito è mostrata una visione schematica della rete utilizzata. Come detto in precedenza il vettore dei parametri della rete che contiene pesi, bias e centri (inizializzati ad un valore nullo), viene aggiornato in ogni istante di tempo dopo che il processo di *growing* è concluso. Alla fine di ogni addestramento su tutti i dati di *training* (cioè su tutti gli istanti di tempo) si reitera il processo più volte inizializzando i parametri della rete a quelli relativi all' ultimo istante di tempo del ciclo precedente finché il numero di neuroni della rete si stabilizza. Si noti, inoltre, che i valori in ingresso alla rete sono normalizzati rispetto alla distanza tra valore massimo e minimo dei dati di input, per poi essere anormalizzati all'uscita.

Gli input della rete sono i seguenti:

- Il primo input è x (normalmente ma non necessariamente ridimensionato tra -1 e 1).
- Il secondo ingresso è il segnale di errore (cioè $e_k = \ f(x_k) - f(x_k)$).
- Il terzo input è l'abilitazione all'apprendimento: con $LE = 1$ l'apprendimento è abilitato, con $LE = 0$ l'apprendimento è disabilitato.

Gli output della rete invece sono:

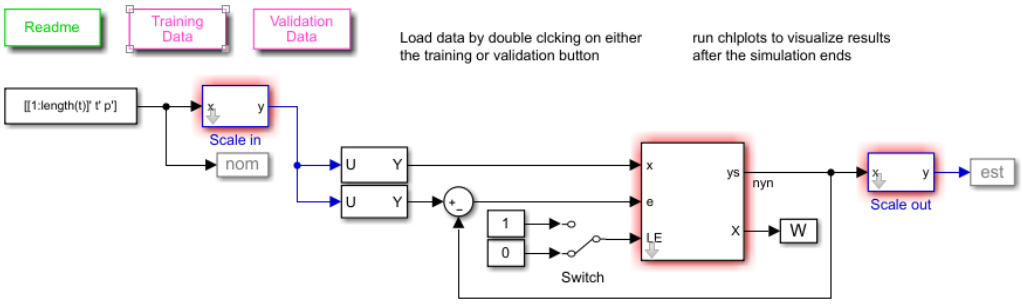


Figura 5.4: Modello a blocchi della GRBFnn implementata con EMRAN

- Il primo output è la funzione appresa $\hat{f}(x)$
- Il secondo output è la matrice degli stati rimodellata a colonne.

Si noti che la rete ha in totale :

$$((N_i + 2) * N_{max} + 2) * N_o \tag{5.37}$$

dove $Nmax$ è il numero massimo di neuroni della rete per ogni singolo output.

Parametri della rete

Di seguito si elencano i parametri fondamentali che governano il funzionamento e le prestazione della rete RBF in esame:

1. Il primo parametro è un vettore contenente N_i e N_o, ovvero le dimensioni (numero di elementi) di x e y.
2. Il secondo parametro è un vettore contenente:
   - N_{max}: il numero massimo di neuroni attivi per un singolo output. Il numero massimo totale di neuroni attivi nell'intera rete è questo valore moltiplicato per il numero di uscite, ovvero N_{max} * N_o. Si comprende che il numero massimo di neuroni non influenza le prestazioni della rete, ma solamente lo spazio di memoria preallocata nel codice per la matrice dei parametri della rete.

- Il Fattore di sovrapposizione o fattore di *overlap* λ che è indicatore del grado di sovrapposizione delle funzioni radiali basiche nella regione degli input. Infatti la varianza della nuova unità aggiunta dipende, attraverso tale parametro, dalla distanza tra il nuovo centro (il cui valore coincide proprio con il valore di input della osservazione x_k corrente) e il centro dell'unità più vicina a x_k, come si nota dalla 5.35. Il fattore di *overlap*, dunque, influenza la varianza delle nuove unità: più è grande, più c'è sovrapposizione tra le unità. Fissato N_{max}, un valore di *overlap* troppo piccolo vuol dire che le unità non coprono bene tutto lo spazio degli input, quindi l'errore sarà elevato perché si trascurano dei set di input; viceversa un valore troppo alto di *overlap* fa perdere la località delle funzioni di attivazione.
- Raggio, è il raggio dell'ipersfera all'interno del quale vengono aggiornati i neuroni. Si noti che il raggio influenza solo l'aggiornamento (apprendimento) ma non riguarda il calcolo dell'output.
- Pruning, quando 1, consente alla rete di potare il neurone peggiore per allocarne uno nuovo dopo che è stato raggiunto il numero massimo di neuroni.

3. Il terzo parametro contiene i 3 tassi di apprendimento per tutti gli stati dei neuroni: pesi, sigmi e centri.

4. Il quarto parametro contiene i 3 fattori stabilizzanti (modifiche Sigma) per pesi, sigmi e centri. Quando l'apprendimento è abilitato ($LE = 1$), questi parametri contrastano la deriva casuale guidando lentamente pesi, sigmi e centri verso lo zero con un guadagno di feedback negativo pari al tasso di apprendimento del fattore stabilizzante.

5. Il quinto parametro contiene 3 limiti per ciascun elemento delle matrici peso, varianze e centri. Fondamentalmente, all'interno del codice è implementato un meccanismo di limitazione in modo tale che le norme di ciascun elemento dei vettori peso, sigma e centro siano confinate all'interno del limite superiore inserito come parametro.

6. Il sesto parametro è la soglia dell'errore. Un nuovo neurone (correlato all'uscita j-esima) viene aggiunto nell'attuale posizione $x(t)$ solo se

l'errore dell'elemento h-esimo è maggiore di questa soglia, come detto in precedenza.

$$e_k = \left\| f(x_k) - \hat{f}(x_k) \right\| > E_1 \tag{5.38}$$

Dove E_1 rappresenta la soglia dell'errore.

7. Il settimo parametro è un vettore che contiene le soglie massima e minima per la distanza, nonché il fattore di decadimento. In ogni dato istante t il valore corrente del *center distance parameter* (CDP) è $dgen = \max(E_{max} * E_{gam}^t, \ E_{min})$ ovvero la soglia inizia da E_{max} e va a E_{min} con un decadimento esponenziale E_{gam}. Se voglio mantenere le distanze costanti durante tutto il *training* allora $E_{max} = E_{min}$ e $E_{gam} = 1$. Questa sarà la strategia utilizzata nel presente lavoro: mantenere le distanza tra i centri costanti lungo l'apprendimento in modo da creare una griglia equi-spaziata. Tuttavia, sebbene la distanza sia fissata, non possiamo dire lo stessi per le posizioni dei centri che vengono ottimizzate durante il *training* in modo da ricoprire le zone più densamente popolate dell'iperspazio degli input. Un nuovo neurone (per approssimare l'uscita h-esima) viene aggiunto nella posizione $x_i(t)$ corrente solo se la distanza dal neurone più vicino di centro μ_{nr}^i (correlata all'uscita h-esima) è maggiore di questa soglia.

$$\left\| x_i - \mu_{nr}^i \right\| > dgen \tag{5.39}$$

Minore è la distanza tra i centri, più l'approssimazione della funzione risulta migliore (funzione approssimata è meno *smooth* e più precisa) ma il numero di neuroni utilizzati per approssimare la funzione è maggiore perché ci vogliono più neuroni per coprire tutto lo spazio degli input. Ne si deduce che al di sotto di un certo valore di *dgen* non conviene scendere in quanto aumenterebbe la complessità della rete e i benefici in termini di riduzione di errore sono praticamente trascurabili. Con dei centri molto vicini, infatti, si favorisce la sovrapposizione e per ogni valore di input ci sarà almeno un contributo rilevante relativo alla funzione radiale più vicina (distanza euclidea tra l'input e il centro della funzione radiale basica).

8. L'ottavo parametro è un vettore contenente la soglia per l'errore filtrato e il polo del filtro lineare. Un nuovo neurone, correlato all'uscita h-esima, viene aggiunto nella posizione corrente $x(t)$ solo se l'errore filtrato dell'output h-esimo è maggiore di questa soglia.

9. L'ultimo parametro è tempo di campionamento.

Nella figura 5.5 sono riportati una serie di valori tipici di tali parametri.

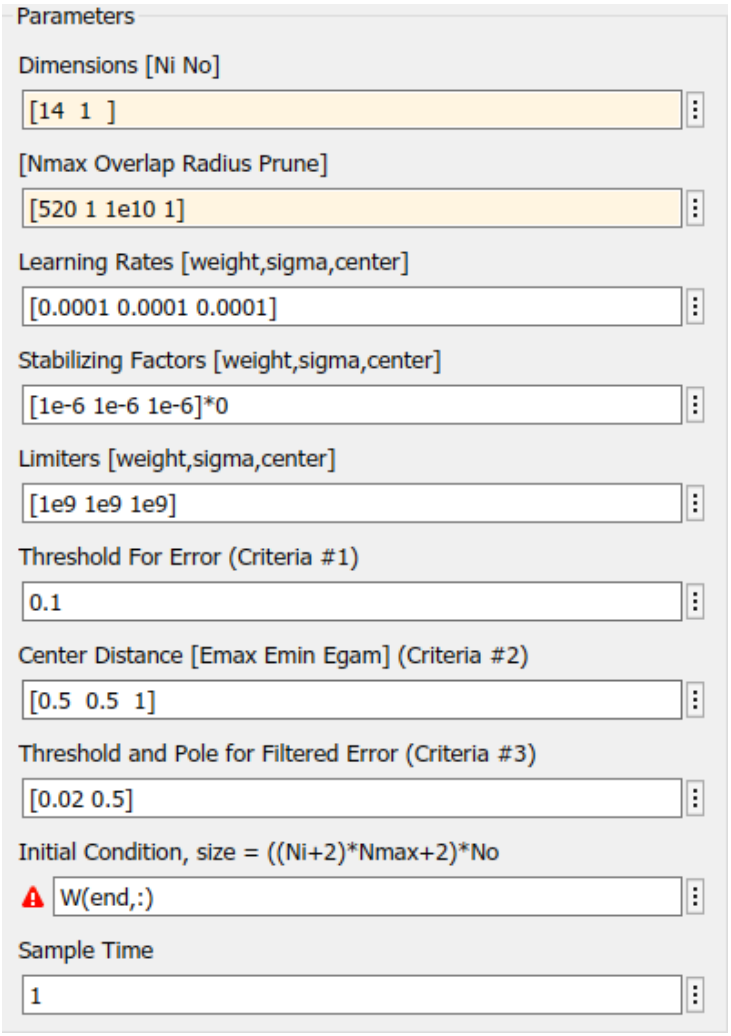


Figura 5.5: Tipici valori dei parametri per la GRBFnn con EMRAN

La condizione iniziale deve essere un vettore di dimensioni $((N_i+2)N_{max}+2)N_o$ e di valori nulli. Ogni output j è relazionato ad un vettore degli stati

di dimensione $(N_i + 2)N_{max} + 2$. Tali stati sono posizionati nella matrice dei parametri della rete W come in seguito:

- $ofY + [1..N_{max} * N_i]$: posizione dei neuroni
- $ofY + [N_{max} * N_i + 1..N_{max} * (N_i + 1)]$: varianza dei neuroni
- $ofY + [N_{max} * (N_i + 1) + 1..N_{max} * (N_i + 2)]$: pesi dei neuroni
- $ofY + N_{max} * (N_i + 2) + 1$: numero di neuroni attivi
- $ofY + N_{max} * (N_i + 2) + 2$: valore corrente dell'errore filtrato $e_f(k)$

dove $ofY = (j - 1) * ((N_i + 2) * N_{max} + 2)$ è l'offset relativo all'output h-esimo. Gli stati legati ad un certo output sono diversi e indipendenti dagli stati legati ad un altro output.

5.5.2 Toolbox di Matlab

Nella corrente sezione si farà riferimento al toolbox di Matlab per la costruzione della GRBFnn. In particolare, la rete è progettata utilizzando la funzione NEWRB. L'algoritmo di apprendimento è di tipo *batch*.

Funzionamento dell'algoritmo

Inizialmente la rete non ha neuroni. Il procedimento iterativo segue I passaggi sotto menzionati finché l'errore arriva al di sotto del *goal* stabilito che è rappresentato dall'errore quadratico medio:

- La rete è simulata
- Il vettore di input (relativo ad un certo istante) che restituisce l'errore maggiore durante la corrente iterazione è selezionato. Questo passaggio rende l'algoritmo di tipo *batch*.
- Si aggiunge un'unità alla rete i cui parametri sono proprio uguali a tale vettore di input selezionato nel passaggio precedente (come visto per EMRAN).
- I pesi sono ottimizzati per ottenere il minimo errore con la configurazione corrente.

Schema della rete

L'architettura generale è quella delle classiche GRBF descritta nel capitolo precedente. L'input pesato di ogni neurone è la distanza tra il vettore di input e il suo vettore di peso, calcolata come segue

$$||\mathbf{W} - \mathbf{X}|| = \sqrt{\sum (x_i - w_i)^2} \tag{5.40}$$

con $X = \{x_1 x_2 \ldots\ldots x_n\}$ e $W = \{w_1 w_2 \ldots\ldots w_n\}$.
L'input 'netto' di ogni neurone invece è uguale a:

$$||\mathbf{W} - \mathbf{X}|| \; b = \left(\sqrt{\sum (x_i - w_i)^2}\right) \; b \tag{5.41}$$

Dove b è il valore di bias dello strato *hidden*. L'output di ogni neurone è il suo input netto passato attraverso la funzione radiale basica(gaussiana). Se il vettore dei pesi di un neurone è uguale al vettore di input (trasposto), il suo input pesato sarà 0, il suo input netto sarà 0 e il suo output sarà 1. Se la distanza tra il vettore dei pesi del neurone e il vettore input ha un valore pari al valore dello *spread* (si veda il seguito), il suo input ponderato sarà distribuito, il suo input netto sarà sqrt $(-log(.5))$ (o 0.8326), quindi il suo output sarà $0,5$. Nela figura 5.6 si riporta una visione schematica di quanto affermato:

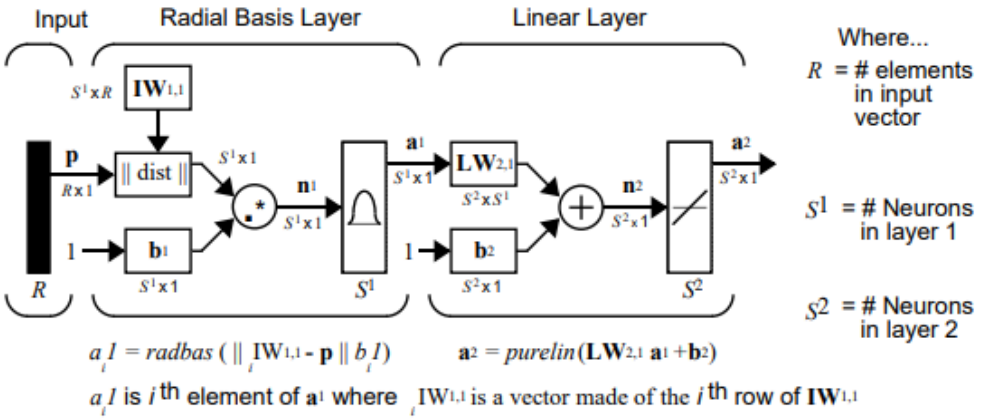


Figura 5.6: Modello di GRBFnn implementato da Matlab. Immagine tratta da `http://128.174.199.77/matlab_pdf/nnet.pdf`

L'architettura è identica a quella proposta nell'algoritmo EMRAN se non

per il fatto che l'introduzione del bias avviene non solo nello strato lineare ma anche in quello nascosto.

Parametri della rete

I parametri in ingresso sono:

1. N_{max} neuroni
2. *Error goal*
3. L'intervallo di neuroni da aggiungere ogni passo
4. Lo *spread*. Il parametro di spread rappresenta il valore del *weighted input* dei neuroni (cioè la distanza euclidea tra vettore input e pesi) tale per cui la rispettiva funzione RBF assume un valore pari a 0.5. Ogni bias nel primo livello è impostato su $0,8326/spread$. Ciò fornisce funzioni di base radiale che attraversano 0,5 agli ingressi ponderati di +/- *spread*: si fissa cioè un punto per cui tutte le funzioni radiali devono passare influenzando la varianza delle stesse. Si è scelto uno *spread* unico per la rete, uguale per tutti i neuroni.

   Dunque, per essere più chiari, se la distanza tra un vettore di input e di pesi relativa ad un neurone assume il valore dello *spread* allora l'input pesato è pari proprio allo *spread*, l'input alla rete è uguale a sqrt $(-log(0.5))$ o 0.8326, e l'output sarà uguale proprio a 0.5. Il bias della rete è calcolato come $0.8326/spread$. Tale parametro, dunque, delimita una regione radiale (di raggio pari allo *spread*) all'interno della quale i neuroni rispondono con un output maggiore o uguale di 0.5 e, di conseguenza, sono da considerarsi attivi. Ad esempio, se lo *spread* è 60, allora ogni neurone RBF, per un vettore di input ad una distanza dal vettore dei pesi inferiore a 60, risponderà con un valore di uscita maggiore o uguale di 0.5.

   In sintesi, lo *spread* influenza la varianza delle funzioni RBF. Più lo *spread* è alto, più le regioni di *overlap* (sovrapposizione) si allargano perché le funzioni RBF sono più *smooth*. È importante che il valore di *spread* sia abbastanza grande da consentire che tutto lo spazio degli input dia una risposta abbastanza rilevante per ogni neurone della rete. Ciò si traduce in una migliore capacità di generalizzazione della rete in fase di test. Tuttavia, un valore di *spread* troppo elevato, vuol dire

funzioni radiali troppo piatte e rischio che tutti i neuroni siano attivi e rispondano allo stesso modo nello spazio degli input. Di solito il valore di *spread* deve essere più grande della distanza tra due vettori input adiacenti, ma più piccolo della distanza misurata su tutto lo spazio degli input. È possibile in un certo modo affermare che lo *spread* ha una funzione simile al parametro di *overlap* λ visto nell'algoritmo EMRAN.

5.6 Analisi parametrica

Dopo aver ottenuto una rete ottimizzata dal punto di vista dell'architettura integrando questo processo, come visto, nella logica di apprendimento è bene effettuare un'analisi parametrica al variare dei parametri principali che compaiono all'interno dell'equazioni di stato delle reti costruite. Si escludono, ovviamente, da questo studio quei parametri che sono ottimizzati all'interno dell'algoritmo stesso cioè i pesi, i bias e i centri.

Dunque, per la rete GRBF costruita con l'algoritmo EMRAN i parametri da far variare durante quest' analisi sono:

- *dgen* (*center distance parameter*)
- λ (*overlap*)

Mentre per la rete costruita tramite il Toolbox di Matlab il parametro che si fa variare è lo *spread*. Si noti che, al fine di abbassare i costi computazionali, queste analisi saranno effettuate considerando una frequenza di campionamento leggermente più bassa rispetto alle analisi effettuate una volta definiti i parametri ottimali. Ciò è comprensibile in quanto siamo interessati a comprendere il trend dei risultati piuttosto che ai risultati stessi. Una volta determinati i parametri ottimali, è possibile procedere alla presentazione dei risultati definitivi aumentando la frequenza di campionamento quanto basta ad ottenere delle buone performance senza penalizzare eccessivamente i costi computazionali.

5.7 AoA: modalità di presentazione e criteri di valutazione dei risultati

In questo paragrafo si riportano dei risultati generici ottenuti con una rete RBF relativamente all'AoA. Il formato presentato per i seguenti risultasti

sarà ripreso nel corso di tutto il lavoro. Sia per il *training* che per il test saranno mostrati 3 grafici:

- Nel primo grafico è rappresentato gli andamenti dell'AoA calcolato dalla rete neurale, l'AoA desiderato e l'AoA lineare, tutti indicati con diversi colori come specificato nella leggenda.
- Nel secondo grafico si rappresenta l'andamento dell'errore tra l'AoA desiderato è quello computato dalla RBFnn
- Nel terzo grafico si mostra la PDF del suddetto errore e i parametri statistici che la caratterizzano. La curva in rosso rappresenta la gaussiana che meglio approssima la distribuzione dell'errore.

Per valutare la bontà dei risultati potrebbero essere scelti numerosi criteri. Molto dipende dalla funzione che il sensore progettato deve svolgere sul velivolo. Ad esempio, se il sensore sintetico ha la funzione di controllo dello stallo è ovvio che le sue prestazioni dovrebbero essere valutate esclusivamente nei tratti di manovra in cui lo stallo potrebbe incorrere e che il *training* avvenga nei suddetti tratti. Scegliere, dunque, l'errore medio e la deviazione standard dell'errore (valutati sull'intera manovra) come parametri discriminanti per la misura delle performance del sensore, potrebbe non avere molto senso. Si potrebbe affermare il contrario se i dati venissero selezionati, considerando esclusivamente i tratti di manovra interessanti per le valutazioni delle performance.

Nel presente lavoro il parametro discriminante per valutare la bontà dei risultati sarà l'errore massimo raggiunto in fase di test, valutato sulla manovra nella sua interezza. Questo per evitare di doversi soffermare su considerazioni come quelle fatte in precedenza, perché l'errore massimo prescinde dal tipo di manovra che utilizziamo e dal punto scelto.

Inoltre, questa strategia è stata richiesta per evitare qualsiasi picco maggiore dei limiti di accettazione (specificati nel paragrafo 2.5), poiché il sensore virtuale può essere utilizzato come discriminatore in tempo reale, mentre la radice quadrata media è calcolata su un intero volo.

Capitolo 6

AoA *training* e *testing* di una GRBFnn su un G-70 ultraleggero

Il G-70 è un aereo ultraleggero dotato di equipaggio e totalemente made in Italy, progettato dalla Nando Groppo Srl.

Figura 6.1: G-70 della Nando Groppo Srl

Di seguito si elencano alcune delle principali caratteristiche del velivolo in questione:

- Mtow: kg 600
- Apertura alare: mm 8900
- Superficie alare: mq $10,68$
- Apertura del piano orizzontale: mm 2740

- Lunghezza: mm 6220
- Massima larghezza della cabina: mm 1120
- Serbatoi: 50 $ltx2$
- Fattore di carico massimo: $g\ +4-2$
- Fattore di carico ultimo: $g\ +6-4$
- Velocità di crociera : km/h 190
- VNE: km/h 210
- Velocità di stallo: km/h 60
- Take off: m 160
- Landing: m 160

Per il G-70 Nando Groppo Srl appena presentato, vale quanto precedentemente affermato in merito all'utilità che un sensore sintetico potrebbe acquisire come sensore ridondante analitico. Il crescente interesse nei confronti dei velivoli ultraleggeri si riflette, dunque, anche sulle tecnologie che possono rendere tali velivoli più sicuri e appetibili sul mercato (come, appunto, un sensore virtuale). Sensori sintetici fondati su tecniche neurali sono l'unica valida alternativa ai sistemi sensoristici convenzionali in grado di offrire un ottimo compromesso tra peso, ingombro e costi. Tuttavia, un brevetto vero e proprio per la tecnologia innovativa qui proposta non esiste ancora, ma la strada è sicuramente quella giusta. La corretta progettazione di un sensore sintetico passa inevitabilmente attraverso una efficace progettazione della rete neurale integrata al suo interno. Il capitolo corrente si concentra proprio su questo aspetto, e cioè sulla concreta progettazione di una rete neurale GRBF (costruita nel capitolo precedente utilizzando l'algoritmo EMRAN e il Toolbox di Matlab) che possa stimare, con discreta accuratezza, l'angolo di attacco AoA durante una serie di manovre di *testing* condotte in ambiente reale dove i dati sono corrotti da un elevato rumore di sottofondo. I risultati ottenuti nella qui presente trattazione possono essere considerati validi con un certo margine di errore legato all'integrazione della rete neurale all'interno del FCS del velivolo. Infatti, per la competa progettazione di un sensore virtuale e per un eventuale successiva certificazione c'è bisogno che la rete testata sull'aeromobile. Quest'analisi, compete ad una fase più avanzata del progetto e non sarà oggetto del presente elaborato di tesi.

6.1 Manovre di *training* e testing

Le manovre utilizzate per allenare e validare la rete neurale e quindi i dati di input e output della rete sono stati raccolti in ambiente reale durante una campagna di test in volo condotta con il suddetto G-70 e durata alcuni mesi.

C'è una sostanziale differenza nell'allenare la rete con dati reali piuttosto che con dati simulati: le manovre condotte in ambiente reale presentano delle caratteristiche dinamiche molto più spiccate di quelle condotte in ambiente simulato. Anche nel caso in cui si parli di manovre di *training*, in ambiente reale, non è possibile disaccoppiare totalmente la dinamica longitudinale da quella latero-direzionale. In più, trattandosi di manovre in ambiente reale è sempre difficile, per motivi di sicurezza, coprire tutto l'inviluppo di volo in modo tale che la rete sia addestrata in tutte le condizioni di volo possibili. A maggior ragione non è consentito uscire al di fuori dei limiti dell'inviluppo in fase di validazione e ciò non consente di valutare le reali capacità della rete in condizioni particolarmente estreme, che potrebbero anche non essere mai raggiunte durante la vita operativa del velivolo. In particolare, quest'ultimo aspetto, e cioè quello di testare il velivolo in condizioni di assoluta criticità, è discriminante per l'ottenimento di una certificazione aeronautica del sensore.

Considerando un altro punto di vista, tuttavia, il *training* e il *testing* della rete in ambiente reale permette di apprendere in contesti altamente turbolenti, rumorosi e altamente dinamici che non sempre sono riproducibili attraverso un volo simulato. Inoltre, non bisogna trascurare che l'aereo in volo in ambiente simulato rappresenta in ogni caso un'approssimazione del sistema reale per cui la dinamica descritta dal simulatore non rispecchierà mai del tutto quella ottenuta sull'aereo vero e proprio. Per tali ragioni è facile comprendere il perché spesso le reti vengano addestrate e testate sia in ambiente reale che simulato. I criteri descritti per la selezione delle manovre di addestramento e validazione sono già stati discussi nel capitolo precedente. Per tale ragione ci si limiterà a descrivere le manovre particolari utilizzate per addestrare e testare le reti GRBF precedentemente costruita. Le manovre utilizzate per il *training* sono principalmente:

- *Manovra di stallo* e relativa rimessa dallo stallo; si mantiene la quota aumentando l'angolo d'attacco e riducendo la velocità con potenza del motore al minimo: questo finché non si raggiungono le condizioni di stallo. A questo punto il muso dell'aeromobile di abbasserà bruscamente e l'aereo inizierà a picchiare. Ciò che si fa per ritornare nelle

condizioni iniziali è accettare la caduta scendendo di quota in modo che l'ala inizi a portare nuovamente. Seguono aumento di potenza del motore e aumento di velocità

- *Manovra sawthoot-glide*, che rappresenta la cosiddetta manovra a dente di coltello.
- *Manovra fugoide*, che consiste nel mantenere quota e velocità costante per beccheggiare attorno alla direzione di traslazione.

Per i test le manovre utilizzate sono le seguenti:

- Manovra *sawthoot-glide*
- Manovra di stallo

È bene far notare che i dati utilizzati sono relativi a voli completi, per cui al loro interno si troveranno informazioni relative anche ad altre manovre (come ad esempio manovre di transizione).Per tutte le manovre descritte è stata effettuata una selezione dei dati al fine di escludere le porzioni di dati relativi a decollo e atterraggio.

6.2 Analisi parametrica GRBnn con algoritmo EMRAN

La seguente analisi parametrica della rete è effettuata con la finalità di determinare i valori ottimali di tutti quei parametri della rete che non sono ottimizzati nel processo iterativo. In questo studio si faranno variare combinatamente:

- Il parametro di distanza dei centri che, come già detto in precedenza, fissa la distanza tra tutti i centri durante il *training*. In particolare, questo parametro si farà variare tra 0.1 e 0.6 ad intervalli di 0.1.
- Il parametro di *overlap* che regola la varianza delle unità aggiunte. L'*overlap* si farà variare da 1 a 4 ad intervalli unitari

Lo scopo è quello di trovare la coppia di *dgen* e λ che ottimizza la rete. Per ulteriori dettagli sul funzionamento del codice si rimanda al paragrafo 5.5.1.

La frequenza di campionamento utilizzata sia per il test che per l'addestramento è più bassa rispetto a quella che si utilizzerà nell'analisi a parametri ottimizzati (circa 1/300). Per meglio comprendere il trend dei parametri statistici si propone anche una visualizzazione attraverso diagrammi a barre 3D dell'errore medio in valore assoluto, della deviazione standard e dell'errore massimo in valore assoluto. Gli stessi risultati saranno poi presentati sottoforma di tabelle.

Relativamente al test 1, facendo riferimento alla figura 6.2 e alla tabella 6.1, è possibile affermare che per il ME non c'è un trend specifico. Tuttavia, si noti che esso risulta più basso per valori bassi di *dgen* combinati a valori bassi di *overlap* oppure per valori elevati di *dgen* combinati a valori elevati di *overlap*. L'errore massimo tende a scendere al diminuire di *dgen* (figura 6.3 e tabella 6.2) , il che coincide con un aumento di neuroni nella rete. Lo stesso non si può dire per l'*overlap* che non influenza in maniera lineare l'errore massimo: tale grandezza risulta minimizzata per un valore di *overlap* pari a 2. La deviazione standard per il test 1 (figura 6.4 e tabella 6.3) diminuisce al decrescere di *dgen* ma non presenta un particolare trend al variare di λ. Si noti che c'è un brusco aumento del valore di 2σ per piccoli valori di *overlap* (1) e per grandi valori di dgen (0.6).

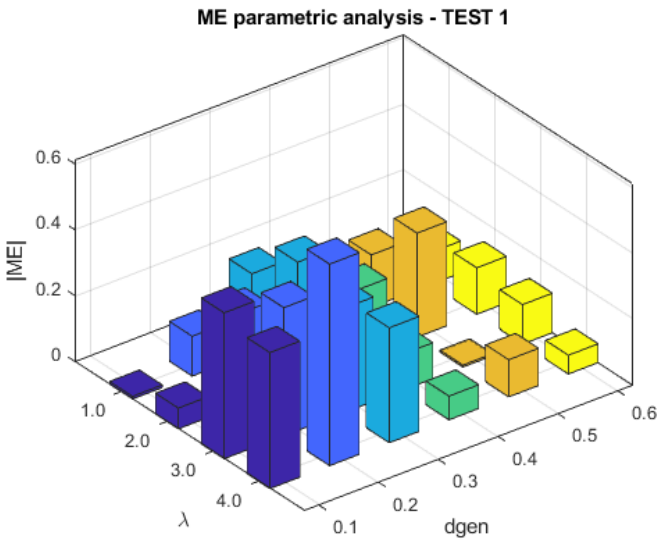


Figura 6.2: Analisi parametrica di ME in valore assoluto al variare di λ e *dgen* con diagrammi a blocchi tridimensionali per il TEST 1 (parametri statistici misurati in gradi)

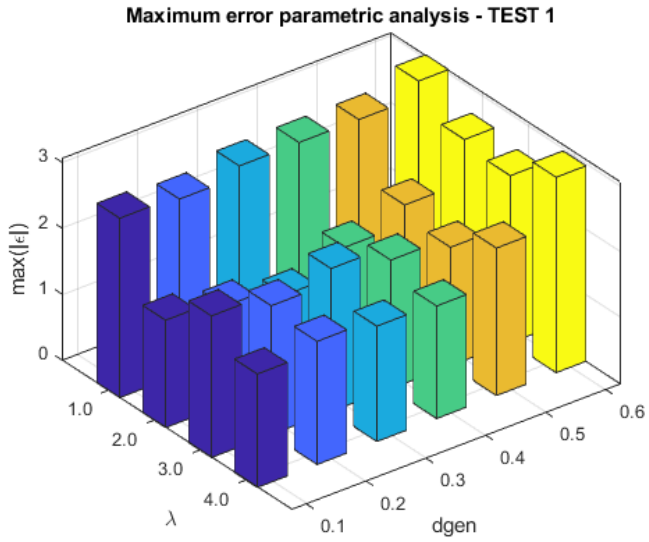


Figura 6.3: Analisi parametrica dell' errore massimo in valore assoluto al variare di λ e *dgen* con diagrammi a blocchi tridimensionali per il TEST 1 (parametri statistici misurati in gradi)

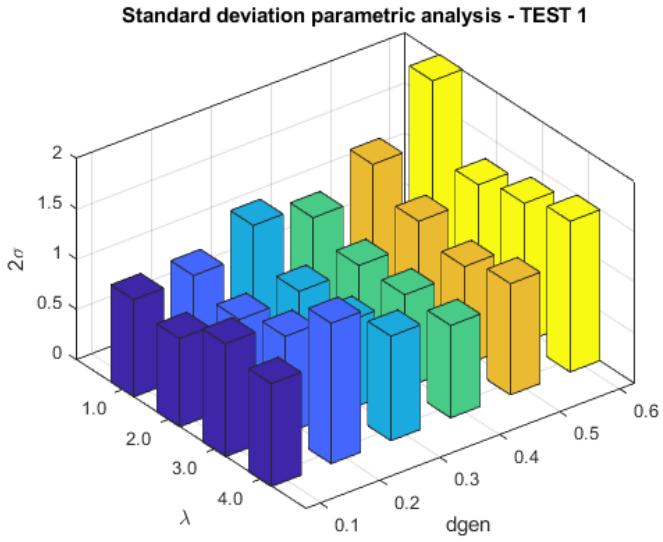


Figura 6.4: Analisi parametrica della deviazione standard al variare di λ e *dgen* con diagrammi a blocchi tridimensionali per il TEST 1 (parametri statistici misurati in gradi)

	$D = 0.1$	$D = 0.2$	$D = 0.3$	$D = 0.4$	$D = 0.5$	$D = 0.6$
$\lambda = 1$	-0.008°	0.121°	0.241°	0.164°	-0.159°	0.097°
$\lambda = 2$	0.063°	0.279°	0.367°	0.213°	0.316°	0.141°
$\lambda = 3$	0.443°	0.389°	0.317°	0.109°	0.006°	0.12°
$\lambda = 4$	0.413°	0.612°	0.349°	0.072°	-0.113°	0.057°

Tabella 6.1: Analisi parametrica di ME in valore assoluto al variare di λ e *dgen* per il test 1. Si è indicato il *distance center parameter* con la lettera D

	$D = 0.1$	$D = 0.2$	$D = 0.3$	$D = 0.4$	$D = 0.5$	$D = 0.6$
$\lambda = 1$	2.721°	2.663°	2.818°	2.825°	2.825°	3.061°
$\lambda = 2$	1.628°	1.493°	1.315°	1.691°	1.983°	2.628°
$\lambda = 3$	2.147°	1.953°	2.169°	1.949°	1.795°	2.533°
$\lambda = 4$	1.72°	1.867°	1.749°	1.707°	2.237°	2.963°

Tabella 6.2: Analisi parametrica dell' errore massimo in valore assoluto al variare di λ e *dgen* per il test 1. Si è indicato il *distance center parameter* con la lettera D

	$D = 0.1$	$D = 0.2$	$D = 0.3$	$D = 0.4$	$D = 0.5$	$D = 0.6$
$\lambda = 1$	0.967°	0.979°	1.249°	1.098°	1.4°	2.003°
$\lambda = 2$	0.878°	0.839°	0.891°	0.92°	1.132°	1.266°
$\lambda = 3$	1.123°	0.968°	0.879°	0.926°	0.979°	1.379°
$\lambda = 4$	1.023°	1.395°	1.039°	0.918°	1.105°	1.498°

Tabella 6.3: Analisi parametrica del 2σ al variare di λ e *dgen* per il test 1. Si è indicato il *distance center parameter* con la lettera D

Per il test 2 è possibile fare le stesse considerazioni per il ME. Osservando le figure 6.5, 6.6, 6.7 e le tabelle 6.4, 6.5, 6.6 si noti che l'errore massimo diminuisce linearmente al diminuire sia di λ che di *dgen* mentre la deviazione standard non varia di molto.

Da quanto visto un buon compromesso nelle prestazioni si ottiene per:

- $\lambda = 2$
- $dgen = 0.1$

Il valore molto basso di *dgen* indica che la configurazione ottimale ottimale ha una rete molto fitta di unità utile per ricoprire al meglio tutto l'iperspazio degli input. Contemporaneamente, però, è ovvio che riducendo *dgen* si aumenta il numero di neuroni necessari per ricoprire l'iperspazio degli input nella sua interezza e ciò si traduce in un costo computazionale più elevato.

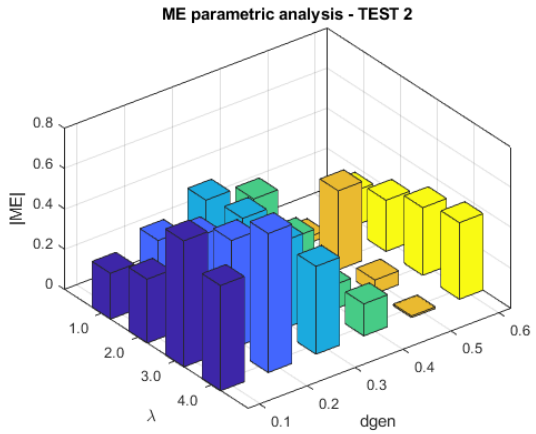


Figura 6.5: Analisi parametrica di ME in valore assoluto al variare di λ e *dgen* con diagrammi a blocchi tridimensionali per il TEST 2 (parametri statistici misurati in gradi)

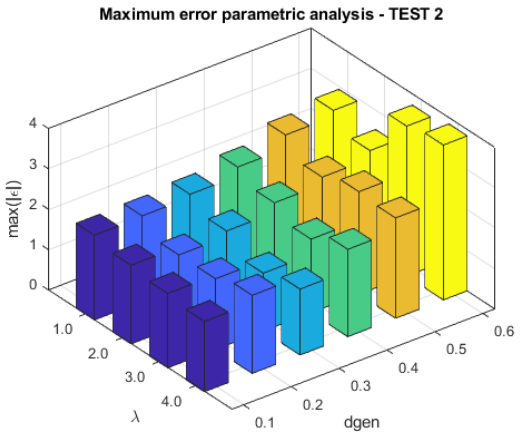


Figura 6.6: Analisi parametrica dell' errore massimo in valore assoluto al variare di λ e *dgen* con diagrammi a blocchi tridimensionali per il TEST 2 (parametri statistici misurati in gradi)

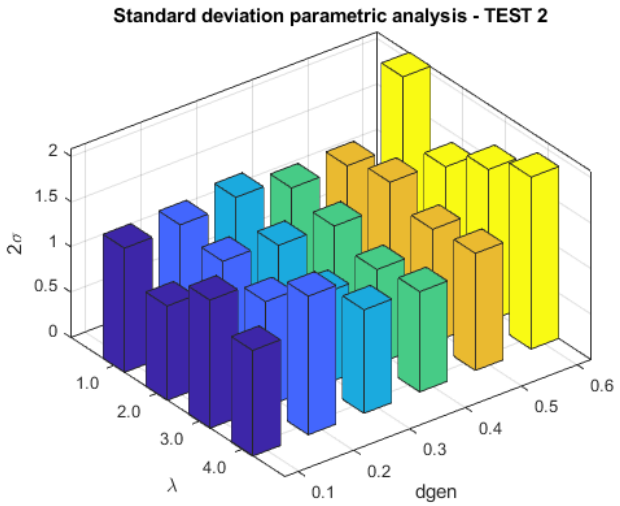


Figura 6.7: Analisi parametrica della deviazione standard al variare di λ e *dgen* con diagrammi a blocchi tridimensionali per il TEST 2 (parametri statistici misurati in gradi)

	$D = 0.1$	$D = 0.2$	$D = 0.3$	$D = 0.4$	$D = 0.5$	$D = 0.6$
$\lambda = 1$	0.228°	0.3°	0.409°	0.324°	-0.031°	-0.171°
$\lambda = 2$	0.317°	0.432°	0.437°	0.246°	0.392°	0.254°
$\lambda = 3$	0.626°	0.537°	0.435°	0.121°	0.068°	0.336°
$\lambda = 4$	0.523°	0.69°	0.436°	0.159°	-0.008°	0.38°

Tabella 6.4: Analisi parametrica del ME in valore assoluto al variare di λ e *dgen* per il test 2. Si è indicato il *distance center parameter* con la lettera D

	$D = 0.1$	$D = 0.2$	$D = 0.3$	$D = 0.4$	$D = 0.5$	$D = 0.6$
$\lambda = 1$	2.098°	2.123°	2.202°	2.423°	2.763°	2.916°
$\lambda = 2$	1.949°	1.748°	1.888°	2.13°	2.311°	2.513°
$\lambda = 3$	1.843°	1.751°	1.434°	1.822°	2.543°	3.712°
$\lambda = 4$	1.741°	1.934°	1.633°	2.163°	2.486°	3.837°

Tabella 6.5: Analisi parametrica dell'errore massimo in valore assoluto al variare di λ e *dgen* per il test 2. Si è indicato il *distance center parameter* con la lettera D

	$D = 0.1$	$D = 0.2$	$D = 0.3$	$D = 0.4$	$D = 0.5$	$D = 0.6$
$\lambda = 1$	1.371°	1.392°	1.458°	1.326°	1.334°	2.079°
$\lambda = 2$	1.041°	1.298°	1.237°	1.213°	1.461°	1.394°
$\lambda = 3$	1.414°	1.161°	1.004°	1.042°	1.25°	1.673°
$\lambda = 4$	1.167°	1.529°	1.142°	1.106°	1.292°	1.897°

Tabella 6.6: Analisi parametrica del 2σ al variare di λ e *dgen* per il test 2. Si è indicato il *distance center parameter* con la lettera D

6.3 Analisi parametrica GRBnn con Toolbox di Matlab

La stessa analisi parametrica è condotta per la rete creata tramite l'utilizzo del Toolbox di Matlab. Le considerazioni generali fatte nel paragrafo precedente possono essere riprese anche in questo caso.

Il parametro che si fa variare in quest'analisi è lo *spread*, che governa la varianza delle unità aggiunte alla rete. Tale parametro si farà variare tra 6000 e 12000 ad intervalli di 500 con un *mean error goal* di 0.2. Valori troppo bassi di *spread* non sono stati considerati perché si è visto che la rete raggiunge con estrema lentezza l'*error goal*.

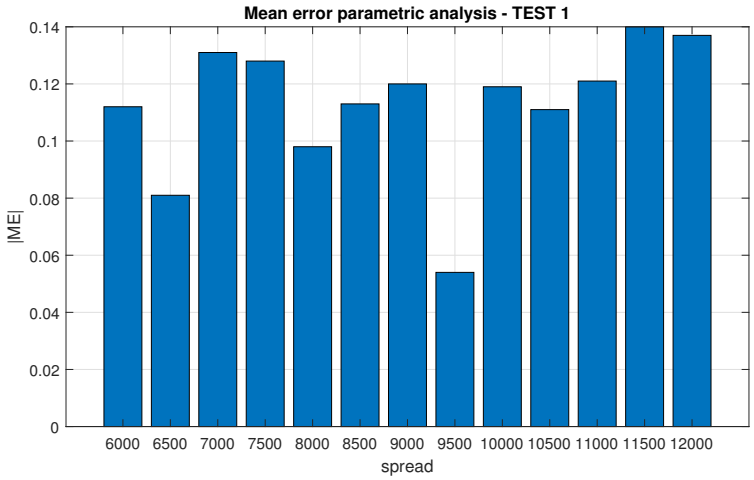


Figura 6.8: Analisi parametrica del ME in valore assoluto al variare dello *spread* con diagrammi a blocchi 2D per il TEST 1 (parametri statistici misurati in gradi)

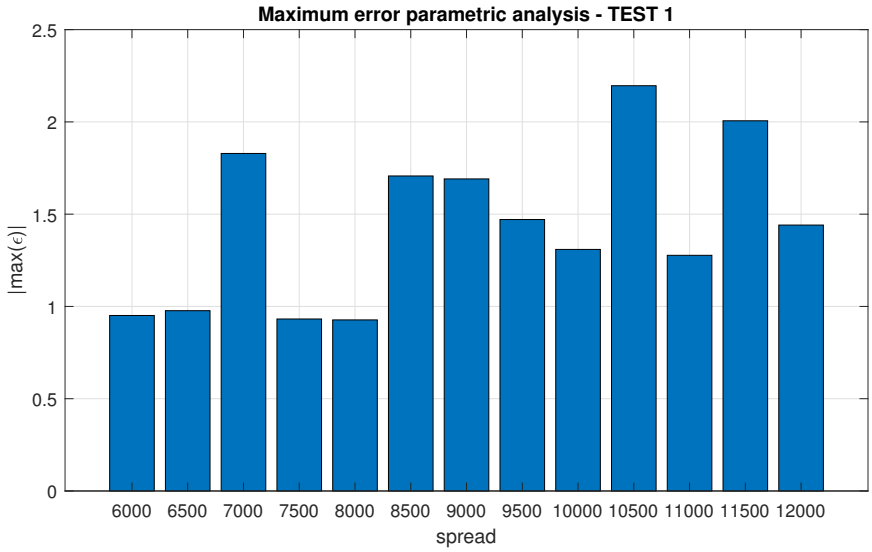


Figura 6.9: Analisi parametrica dell' errore massimo in valore assoluto al variare dello *spread* con diagrammi a blocchi 2D per il TEST 1 (parametri statistici misurati in gradi)

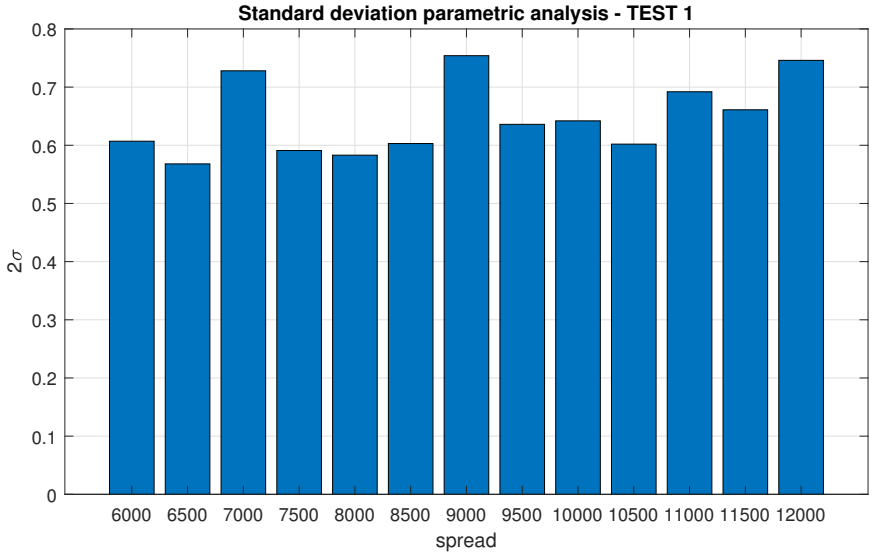


Figura 6.10: Analisi parametrica di 2σ al variare dello *spread* con diagrammi a blocchi 2D per il TEST 1 (parametri statistici misurati in gradi)

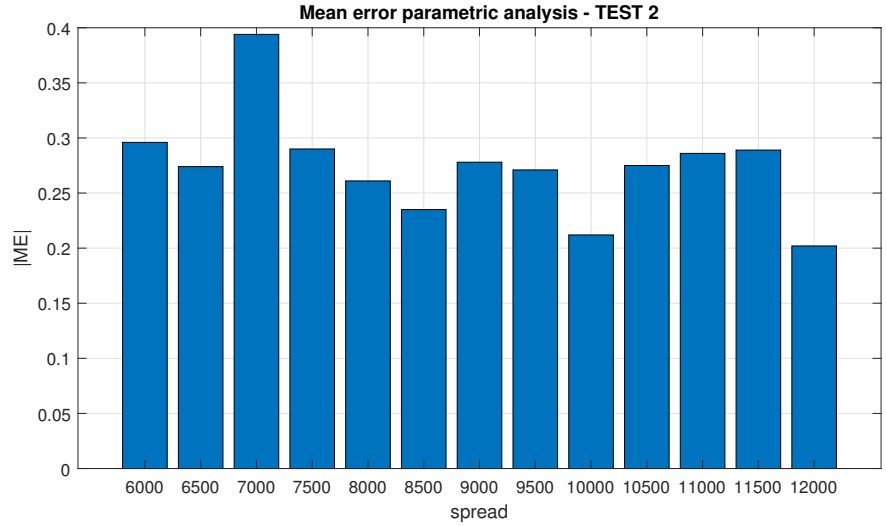


Figura 6.11: Analisi parametrica del ME in valore assoluto al variare dello *spread* con diagrammi a blocchi 2D per il TEST 2 (parametri statistici misurati in gradi)

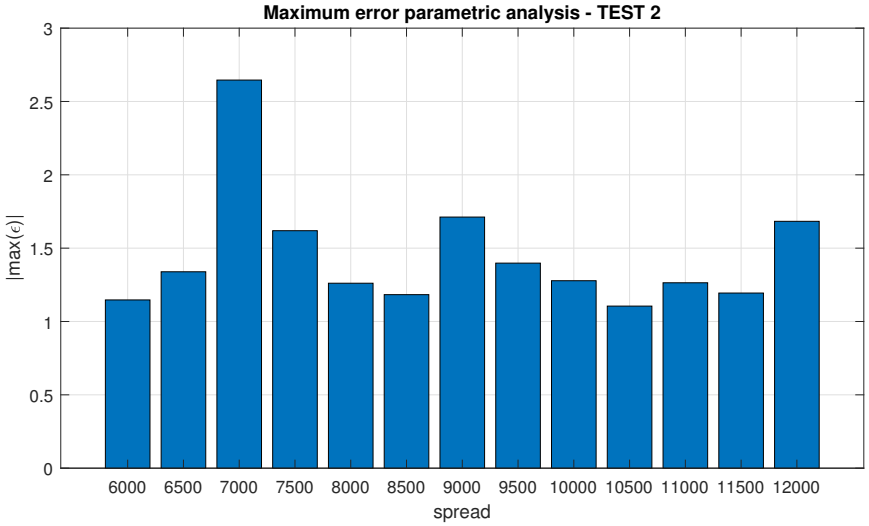


Figura 6.12: Analisi parametrica dell' errore massimo in valore assoluto al variare dello *spread* con diagrammi a blocchi 2D per il TEST 2 (parametri statistici misurati in gradi)

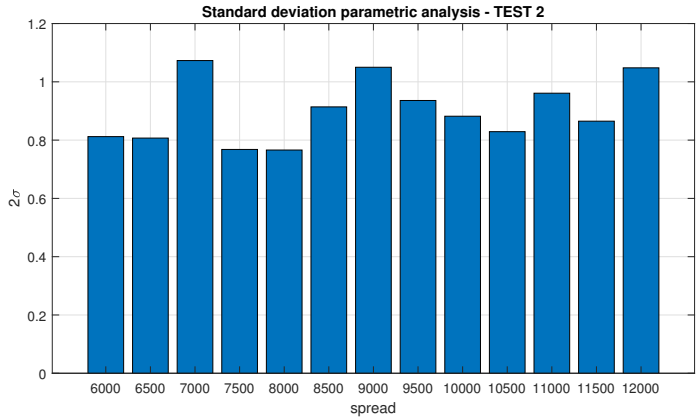


Figura 6.13: Analisi parametrica di 2σ al variare dello *spread* con diagrammi a blocchi 2D per il TEST 2 (parametri statistici misurati in gradi)

Dai diagrammi a barre, relativi rispettivamente al test 1 e al test 2, è possibile scegliere il valore più adatto di *spread* in modo che i parametri statistici risultino ottimizzati. Il parametro discriminante per la scelta è l'errore massimo ma si cerca di ottenere un buon compromesso anche per quanto riguarda il valore della deviazione standard e dell'errore medio. Il valore di *spread* scelto per condurre l'analisi ottimale è pari ad 8000.

$Spread$	ME'	ME''	$2\sigma'$	$2\sigma''$	$max(\epsilon)'$	$max(\epsilon)''$
6000	0.112°	0.296°	0.607°	0.812°	0.951°	1.147°
6500	0.081°	0.274°	0.568°	0.807°	0.977°	1.339°
7000	0.131°	0.394°	0.728°	1.073°	1.829°	2.646°
7500	0.128°	0.29°	0.591°	0.768°	0.932°	1.619°
8000	0.098°	0.261°	0.583°	0.766°	0.927°	1.261°
8500	0.113°	0.235°	0.603°	0.914°	1.707°	1.183°
9000	0.12°	0.278°	0.754°	1.05°	1.691°	1.712°
9500	0.054°	0.271°	0.636°	0.936°°	1.471°	1.398°
10000	0.119°	0.212°	0.642°	0.882°	1.309°	1.278°
10500	0.111°	0.275°	0.602°	0.829°	2.196°	1.105°
11000	0.121°	0.286°	0.692°	0.961°	1.277°	1.264°
11500	0.14°	0.289°	0.661°	0.865°	2.006°	1.194°
12000	0.137°	0.202°	0.746°	1.048°	1.441°	1.683°

Tabella 6.7: Analisi parametrica delle misure statistiche al variare dello *spread*. Le grandezze relative al test 1 sono state indicate con $'$ mentre quelle realtive al test 2 con $''$.

6.4 Analisi ottimale

Una volta definite le architetture ottimali per entrambe le reti progettate è possibile procedere ad un'analisi dei risultati per l'AoA a parametri ottimizzati.

Di seguito sono riportate alcune considerazioni di carattere generale valide per entrambe le reti:

- In questo caso si è interessati anche all'andamento dell'errore nei dati relativi al *training*.
- La frequenza di campionamento utilizzata per dati di *training* è maggiore rispetto all'analisi parametrica effettuata (circa 1/50).
- Per il test si è utilizzata una frequenza di campionamento di 1/50, non si è scesi al di sotto di tale valore perché la memoria allocata per processare il codice è troppo grande.
- Si sceglie come parametro discriminante per la valutazione della bontà dei risultati ottenuti l'errore massimo in quanto prescinde dal tipo di manovra che utilizziamo e dal punto scelto. In particolare, perché i risultati siano ritenuti 'buoni', si stabilisce un intervallo limite al di fuori della quale è preferibile che l'errore non fuoriesca e cioè [-2°, 2°]. Questo intervallo è scelto sulla base del fatto che la rete lavora in ambiente reale. Nel caso in cui l'ambiente di lavoro fosse simulato l'intervallo di accettabilità si restringerebbe (paragrafo 2.5.

Nelle figure da 6.14 a 6.19 sono mostrati i risultati finali relativi all'angolo d'attacco per la GRBFnn progettata tramite EMRAN rispettivamente per le manovre di *training*, la prima manovra di test e la seconda manovra di test. Ogni figura è costituita da tre grafici: nel primo grafico è plottato l'andamento dell'angolo d'attacco desiderato, l'andamento dell'angolo d'attacco misurato dalla rete neurale e l'andamento dell'angolo di attacco calcolato dalla teoria linearizzata. Nel secondo grafico, invece, è plottato il profilo dell'errore che la rete commette nella stima dell'angolo d'attacco. Nel terzo grafico è raffigurata la funzione densità di probabilità dell'errore con i relativi parametri statistici che la caratterizzano in sovraimpressione. Il numero massimo delle iterazioni (che è un parametro solo nell'algoritmo EMRAN) è fissato pari a 10 in quanto si osserva che oltre tale valore il numero di neuroni della rete si stabilizza. In particolare, il numero finale di unità della rete si attesta a 2688 neuroni.

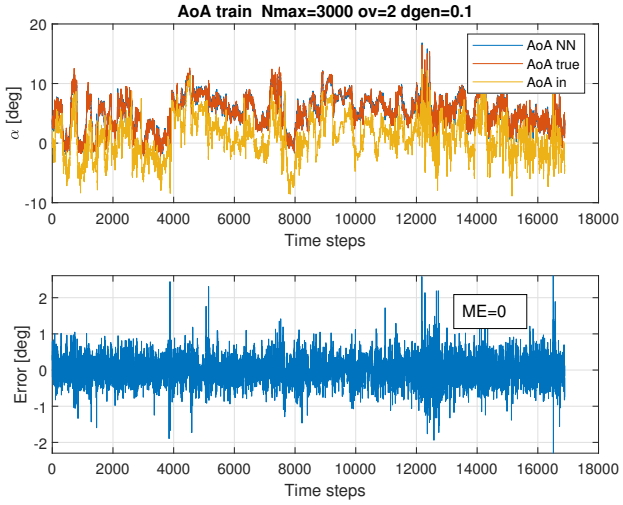


Figura 6.14: Addestramento AoA ottimale per la rete EMRAN GRBF. Il primo grafico mostra l'andamento dell'AoA stimato dalla rete neurale (in blu), l'andamento dell' AoA reale (in arancio), e l'andamento dell' AoA stimato con la teoria lineare (in giallo). Il secondo grafico mostra l'andamento dell'errore tra la stima della rete neurale e l'andamento reale. I parametri statistici sono da intendersi in gradi.

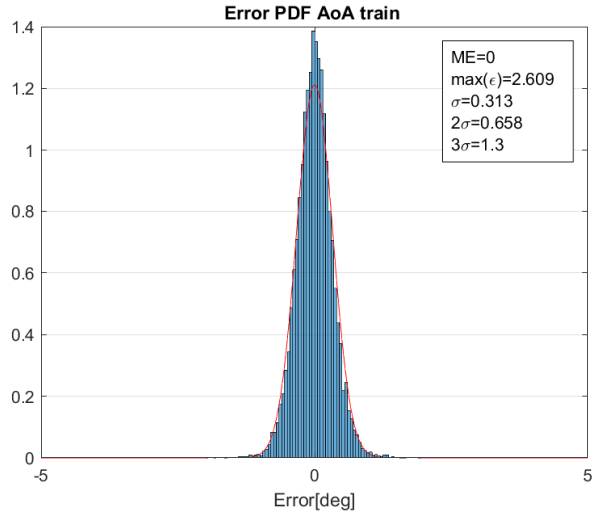


Figura 6.15: Funzione densità di probabilità dell' errore relativo ad AoA per il *training* della rete EMRAN GRBF. I parametri statistici sono da intendersi in gradi. La curva rossa è la gaussiana che meglio interpola la distribuzione dell'errore ottenuta.

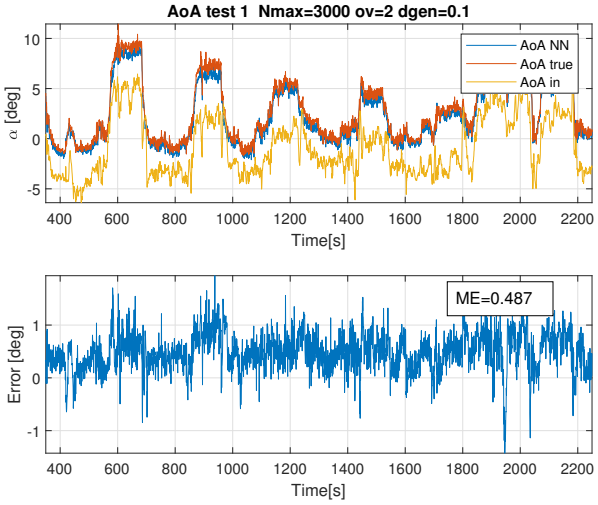


Figura 6.16: Test 1 AoA ottimale per la rete EMRAN GRBF. Il primo grafico mostra l'andamento dell'AoA stimato dalla rete neurale (in blu), l'andamento dell' AoA reale (in arancio), e l'andamento dell' AoA stimato con la teoria lineare (in giallo). Il secondo grafico mostra l'andamento dell'errore tra la stima della rete neurale e l'andamento reale. I parametri statistici sono da intendersi in gradi.

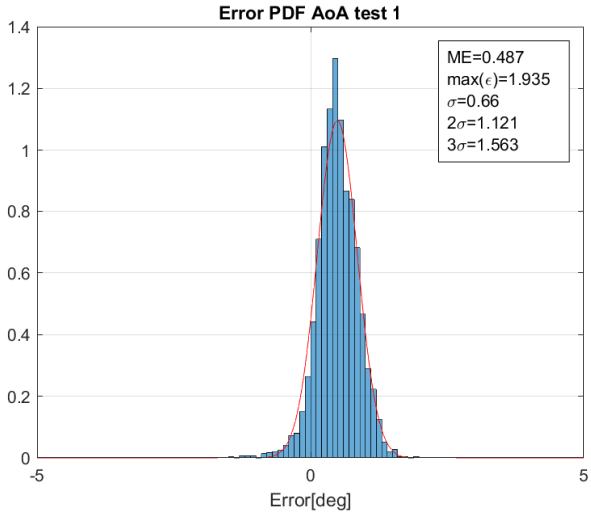


Figura 6.17: Funzione densità di probabilità dell' errore relativo ad AoA per il test 1 della rete EMRAN GRBF. I parametri statistici sono da intendersi in gradi. La curva rossa è la gaussiana che meglio interpola la distribuzione dell'errore ottenuta.

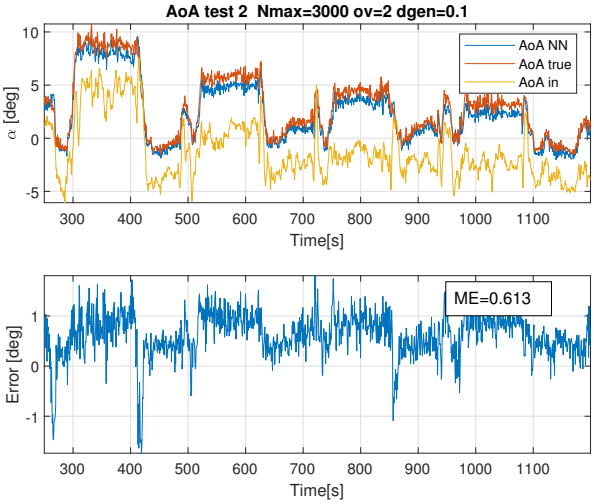


Figura 6.18: Test 2 AoA ottimale per la rete EMRAN GRBF. Il primo grafico mostra l'andamento dell'AoA stimato dalla rete neurale (in blu), l'andamento dell' AoA reale (in arancio), e l'andamento dell' AoA stimato con la teoria lineare (in giallo). Il secondo grafico mostra l'andamento dell'errore tra la stima della rete neurale e l'andamento reale. I parametri statistici sono da intendersi in gradi.

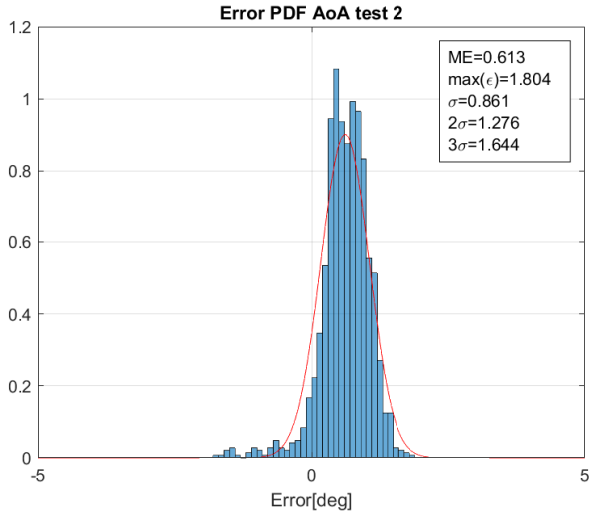


Figura 6.19: Funzione densità di probabilità dell' errore relativo ad AoA per il test 2 della rete EMRAN GRBF. I parametri statistici sono da intendersi in gradi. La curva rossa è la gaussiana che meglio interpola la distribuzione dell'errore ottenuta.

Analizzando i risultati si evince che la rete progettata riesce a prevedere con accuratezza più che accettabile l'andamento dell'angolo d'attacco in quanto l'errore massimo per i due test rientra all'interno della banda di accettabilità prefissata cioè tra -2 gradi e 2 gradi. L'errore medio è leggermente spostato dallo 0 (tra 0.5 e 0.6 gradi), mentre la varianza si attesta attorno a 1.2 gradi sia per il primo che per il secondo test. Il *training* presenta qualche picco d'errore più accentuato legati probabilmente alla natura dinamica delle manovre di addestramento, ma ciò non è preoccupante alla luce dei risultati ottimi ottenuti nei test.

Le figure da 6.21 a 6.26 raffigurano i risultati per la GRBFnn progettata tramite Toolbox di Matlab. In particolare, la figura 6.20 rappresenta la curva delle performance della rete in termini di errore medio all'aumentare del numero di iterazioni cioè del numero di neuroni, visto che per ogni iterazione si aggiunge un unico neurone. La linea orizzontale in nero rappresenta il goal che la rete tenta di raggiungere. Il numero di neuroni finale della rete si attesta a 200. È possibile notare che il *goal* impostato inizialmente non è mai raggiunto, ma la rete approssima comunque in maniera ottimale le funzioni desiderate. Per le altre figure si ripetono le stesse considerazioni effettuate per i precedenti risultati: si nota che, anche in questo caso i risultati sono accettabili per entrambe le reti. Infatti, il massimo valore dell'errore è contenuto all'interno dell'intervallo [-2°, 2°].

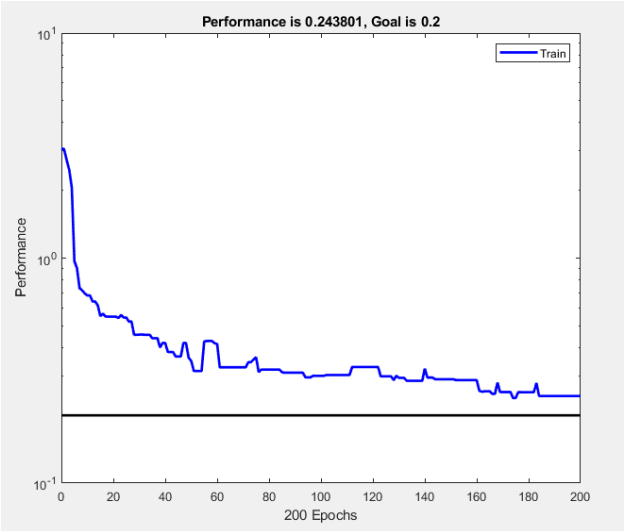


Figura 6.20: Performance della rete GRBF implementata con Matlab al variare del numero di iterazioni, cioè dei neuroni aggiunti.

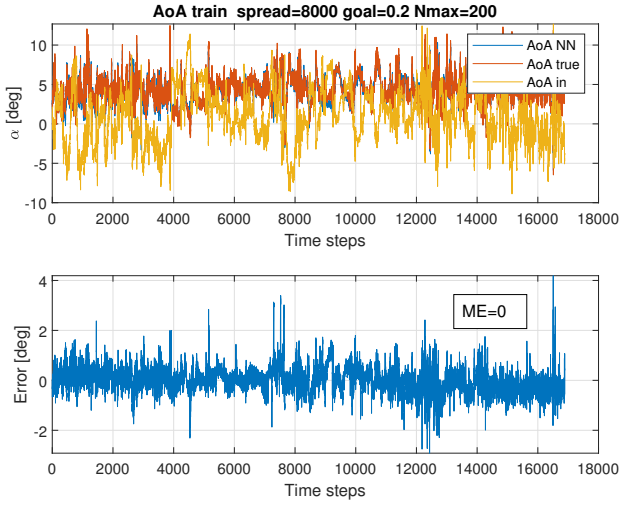


Figura 6.21: Addestramento AoA ottimale per la rete Matlab GRBF. Il primo grafico mostra l'andamento dell'AoA stimato dalla rete neurale (in blu), l'andamento dell' AoA reale (in arancio), e l'andamento dell' AoA stimato con la teoria lineare (in giallo). Il secondo grafico mostra l'andamento dell'errore tra la stima della rete neurale e l'andamento reale. I parametri statistici sono da intendersi in gradi.

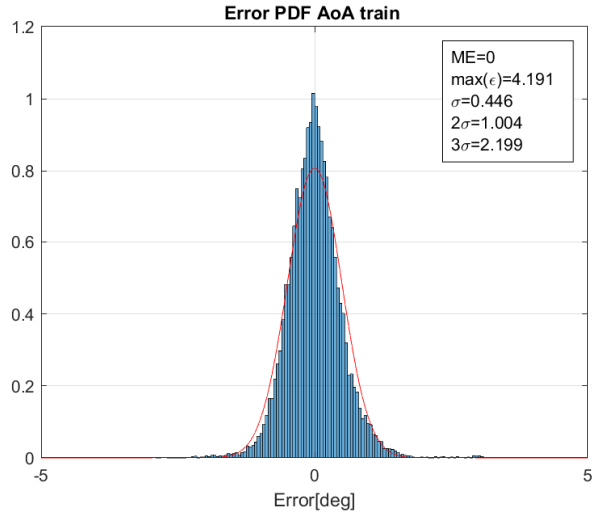


Figura 6.22: Funzione densità di probabilità dell' errore relativo ad AoA per il *training* della rete Matlab GRBF. I parametri statistici sono da intendersi in gradi. La curva rossa è la gaussiana che meglio interpola la distribuzione dell'errore ottenuta.

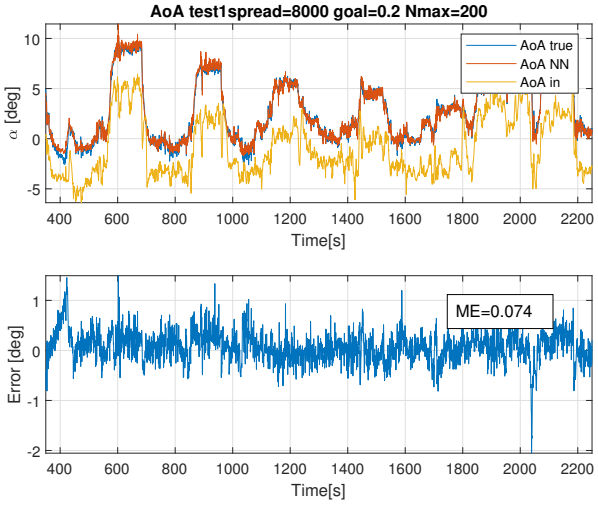


Figura 6.23: Test 1 AoA ottimale per la rete Matlab GRBF. Il primo grafico mostra l'andamento dell'AoA stimato dalla rete neurale (in blu), l'andamento dell' AoA reale (in arancio), e l'andamento dell' AoA stimato con la teoria lineare (in giallo). Il secondo grafico mostra l'andamento dell'errore tra la stima della rete neurale e l'andamento reale. I parametri statistici sono da intendersi in gradi.

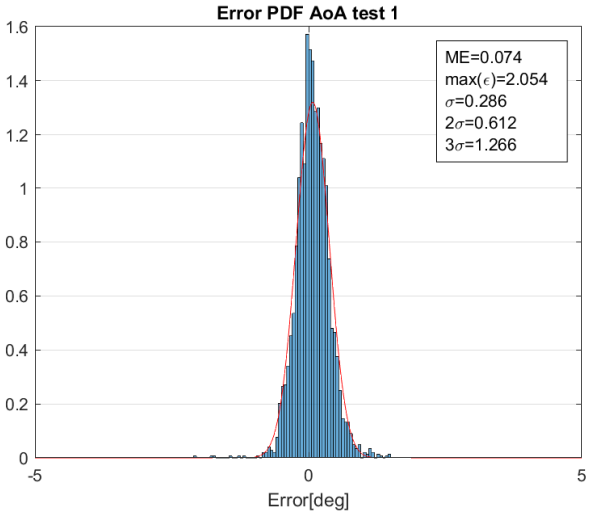


Figura 6.24: Funzione densità di probabilità dell' errore relativo ad AoA per il test 1 della rete Matlab GRBF. I parametri statistici sono da intendersi in gradi. La curva rossa è la gaussiana che meglio interpola la distribuzione dell'errore ottenuta.

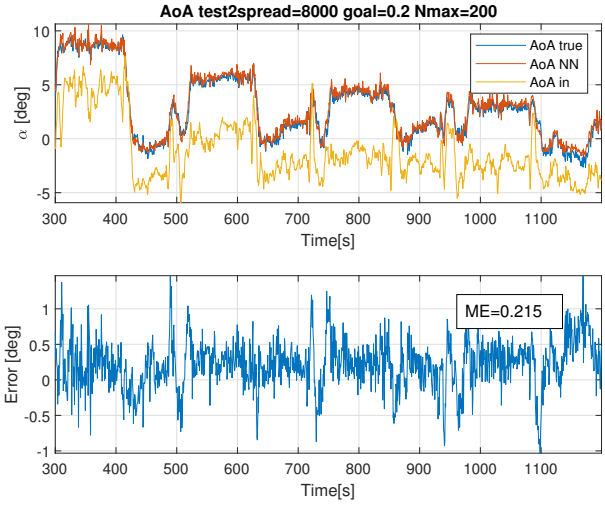


Figura 6.25: Test 2 AoA ottimale per la rete Matlab GRBF. Il primo grafico mostra l'andamento dell'AoA stimato dalla rete neurale (in blu), l'andamento dell' AoA reale (in arancio), e l'andamento dell' AoA stimato con la teoria lineare (in giallo). Il secondo grafico mostra l'andamento dell'errore tra la stima della rete neurale e l'andamento reale. I parametri statistici sono da intendersi in gradi.

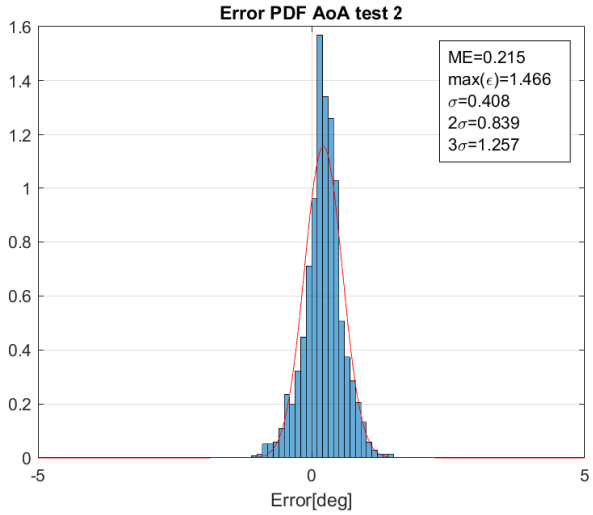


Figura 6.26: Funzione densità di probabilità dell' errore relativo ad AoA per il test 2 della rete Matlab GRBF. I parametri statistici sono da intendersi in gradi. La curva rossa è la gaussiana che meglio interpola la distribuzione dell'errore ottenuta.

Volendo confrontare i risultati, quelli ottenuti con il toolbox di Matlab sembrerebbero preferibili rispetto a quelli ottenuti con EMRAN in quanto sia l'errore massimo che l'errore medio sono minori per entrambi i test, come si può notare dalle tabelle 7.1 e 6.9. Si tenga in considerazione che la rete EMRAN è basata su un algoritmo di apprendimento on line per cui la sua configurazione cambia in ogni istante di tempo e i suoi comportamenti sono più imprevedibili. Il vantaggio è che risulta computazionalmente meno dispendiosa perchè non conserva tutti i pattern di *training* in memoria, ma li scarta dopo averli utilizzati. l'algoritmo sfruttato dal toolbox di Matlab, che si basa su un *batch learning*, rende la rete immutabile una volta addestrata. Pertanto il suo comportamento è più semplice da predire e meno instabile. Inoltre, si nota che la rete progettata con EMRAN ha in media molti più neuroni della rete progettata con Matlab, il che rende la prima più complessa.

	$\|ME\|$	$max(\|\epsilon\|)$	2σ
Train	0°	2.609°	0.658°
Test1	0.487°	1.935°	1.121°
Test2	0.613°	1.804°	1.276°

Tabella 6.8: Risultati per la rete GRBF progettata con EMRAN

	$\|ME\|$	$max(\|\epsilon\|)$	2σ
Train	0°	4.191°	1.004°
Test1	0.074°	2.054°	0.612°
Test2	0.215°	1.466°	0.839°

Tabella 6.9: Risultati per la rete GRBF progettata con Matlab

Capitolo 7

Confronto tra reti neurali MLP e GRBF

Lo scopo di questo capitolo è quello di confrontare le reti neurali MLP e le reti neurali RBF per comprenderne le sostanziali differenze e cercare di delineare un quadro più chiaro sulle possibili applicazioni in cui una rete potrebbe essere più performante dell'altra. Come per molti problemi che si affrontano quotidianamente, il criterio del 'migliore' non è universale ma è strettamente relazionato al contesto e le condizioni in cui tale valutazione avviene. È difficile, dunque, dire quale delle due reti sia migliore dell'altra, perciò ci si limita ad evidenziare i punti salienti che differenziano le due tipologie di rete i quali sono da ricercarsi in un approccio progettuale a priori totalmente discordante. Sebbene una scelta discriminante sia difficile è tuttavia possibile affermare che entrambe sono degli ottimi 'approssimatori universali' che riescono a risolvere problemi di interpolazione con elevata accuratezza. Di seguito si elencano le sostanziali differenze tra le due reti, dedotte dalle conoscenze teoriche acquisite e da precedenti lavori che hanno evidenziato pro e contro delle due architetture mostrando le tipologie di applicazioni più adatte per entrambe:

1. L'**architettura** delle RBF è molto semplice ed è sempre univoca cioè costituita da un unico strato *hidden*, uno strato input e uno strato output. L'architettura delle MLP al contrario può presentare delle configurazioni molto complesse con più strati *hidden fully connected*. Questo rende l'algoritmo di addestramento e il calcolo dei pesi molto più veloce nelle RBF piuttosto che nelle MLP.

2. Il **numero di neuroni** per le reti RBF, per il teorema di Cover risulta essere in media molto più elevato di quello delle MLP in quanto la capacità di approssimazione delle RBFnn dipende dalla capacità di rendere altamente probabile la separabilità lineare degli input nell'iperspazio dello strato nascosto e ciò accade se questi ulitmi sono proiettati in uno spazio ad elevata dimensionalità. Questo aspetto bilancia in un certo senso la maggiore complessità architettonica e computazionale delle MLP rendendo le due strutture abbastanza simili in merito alla velocità di addestramento.

3. Lo **strato nascosto** per le RBF è non lineare e lo **strato di output** è lineare. Per le MLP accade lo stesso nel caso di problemi di regressione non lineare ma non nel caso di problemi di classificazione dove entrambi gli strati sono non lineari. In particolare, le funzioni di attivazione per le MLP sono sigmoidali mentre per le RBF sono funzioni radiali, come indicato dal nome.

4. Il modello e le **funzionalità dei neuroni** dello strato nascosto e dello strato di uscita sono equivalenti per le MLP, mentre ciò non è vero per le RBF per le quali solo i neuroni dello strato nascosto hanno una funzione computazionale

5. **Il valore in ingresso per lo strato nascosto** e quindi per ogni funzione di attivazione risulta essere la distanza euclidea tra il vettore input e i centri delle funzioni radiali per le RBF mentre per le MLP corrisponde al prodotto tra il vettore di input e il vettore dei pesi.

6. La differente **natura delle funzioni di attivazione** delle due reti e della struttura rende le reti neurali RBF degli ottimi **approssimatori locali** in quanto solo i neuroni che vengono attivati dagli input in una certa porzione di iperspazio contribuiscono alla soluzione e all'aggiornamento dei parametri della rete. Le MLP sono degli **approssimatori globali** in quanto tutte le unità contribuiscono sempre al calcolo della soluzione e all'aggiornamento dei parametri della rete.

7. La **strategia di classificazione** degli input nell'iperspazio è diversa per le due reti: le RBF classificano i dati attraverso **ipersfere**, mentre le MLP classificano i dati attraverso **ipersuperfici**. Ciò è legato alla diversa natura delle funzioni di attivazione.

8. Le reti RBF soffrono più difficilmente i **problemi di minimo locale**: infatti la natura auto-regolarizzata di queste reti (come mostrato dall'approccio di Tichonov), e cioè intrinseca nelle funzioni radiali stesse, conferisce una natura più *smooth* alla funzione di costo evitando problemi di minimo locale. Ciò grazie all'azione del termine di regolarizzazione.

9. Per le RBF c'è bisogno di conoscere preliminarmente molti più **parametri** rispetto alle MLP (centri e varianze) per la disposizione delle unità nascoste nell'iperspazio.

10. I **pesi** per le RBF possono assumere un significato fisico aiutando alla comprensione del modello. Per le MLP ciò non accade: i pesi non hanno alcun significato fisico.

11. Numerosi studi hanno dimostrato che le RBF sono molto adatte per approssimare funzioni altamente irregolari forse proprio per la natura non lineare delle funzioni di attivazione di queste reti, mentre le MLP hanno un'accuratezza più elevata per l'approssimazione di funzioni più *smooth*; per i problemi di classificazione le MLP risultano più performanti. Infine, si è visto che le RBFnn sono molto più robuste e tolleranti in presenza di rumore di sottofondo che corrompe i dati di input, mentre le MLP sono più instabili e quindi devono essere trattate con opportune tecniche di regolarizzazione [14].

7.1 Stima di AoA con MLPnn e GRBFnn: confronto dei risultati su un G-70

Di seguito si propone un confronto dei risultati ottenuti in merito alla stima di AoA attraverso una rete neurale GRBF e una rete neurale MLP allo scopo di comprendere la sostanziale differenza nelle performance delle reti in ambiente reale.

La rete MLP scelta è costituita da 13 neuroni con un unico strato nascosto; come per la RBFnn è una MISO.I risultati per la rete MLP sono confrontati con quelli ottenuti nel capitolo precedente relativi alle due reti neurali RBF.

Tutte le considerazioni effettuate nel capitolo 6 in merito alle modalità di scelta delle manovre, trattazione dei dati, frequenza di campionamento,

presentazione dei risultati e scelta dei parametri discriminanti per la loro valutazione sono valide anche per i presenti risultati. Nelle figure da 7.1 a 7.6 sono raffigurati i risultati per la MLP.

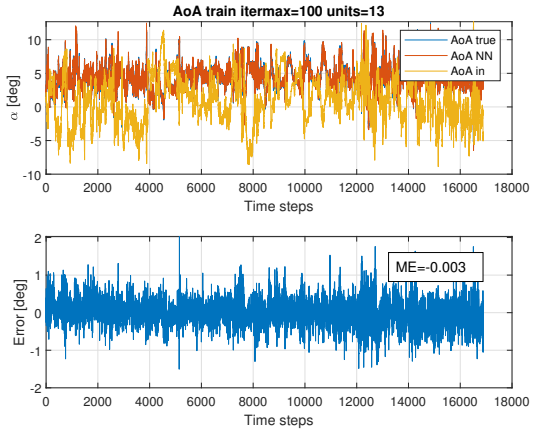


Figura 7.1: Addestramento AoA ottimale per la rete MLP. Il primo grafico mostra l'andamento dell'AoA stimato dalla rete neurale (in arancio), l'andamento dell' AoA reale (in blu), e l'andamento dell' AoA stimato con la teoria lineare (in giallo). Il secondo grafico mostra l'andamento dell'errore tra la stima della rete neurale e l'andamento reale. I parametri statistici sono da intendersi in gradi.

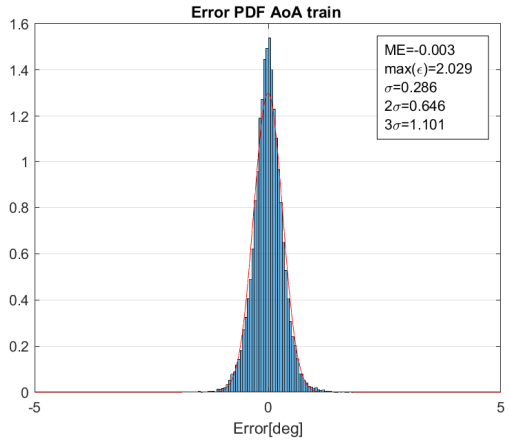


Figura 7.2: Funzione densità di probabilità dell' errore relativo ad AoA per il *training* della rete MLP. I parametri statistici sono da intendersi in gradi. La curva rossa è la gaussiana che meglio interpola la distribuzione dell'errore ottenuta.

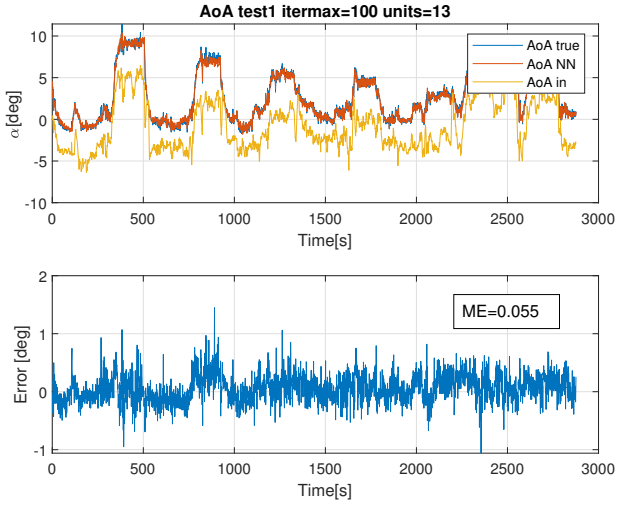


Figura 7.3: Test 1 AoA ottimale per la rete MLP. Il primo grafico mostra l'andamento dell'AoA stimato dalla rete neurale (in arancio), l'andamento dell' AoA reale (in blu), e l'andamento dell' AoA stimato con la teoria lineare (in giallo). Il secondo grafico mostra l'andamento dell'errore tra la stima della rete neurale e l'andamento reale. I parametri statistici sono da intendersi in gradi.

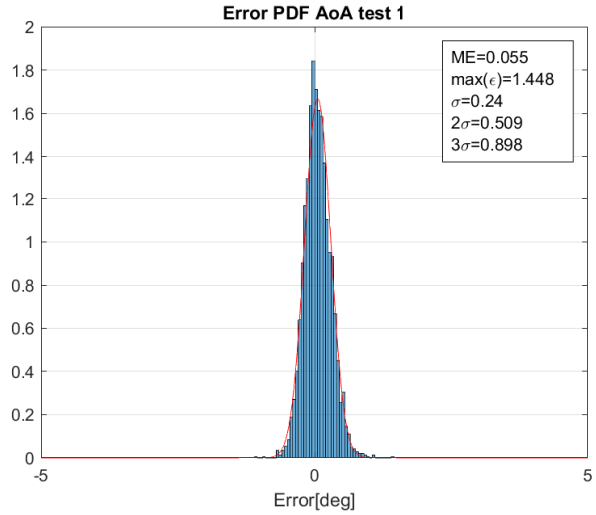


Figura 7.4: Funzione densità di probabilità dell' errore relativo ad AoA per il test 1 della rete MLP. I parametri statistici sono da intendersi in gradi. La curva rossa è la gaussiana che meglio interpola la distribuzione dell'errore ottenuta.

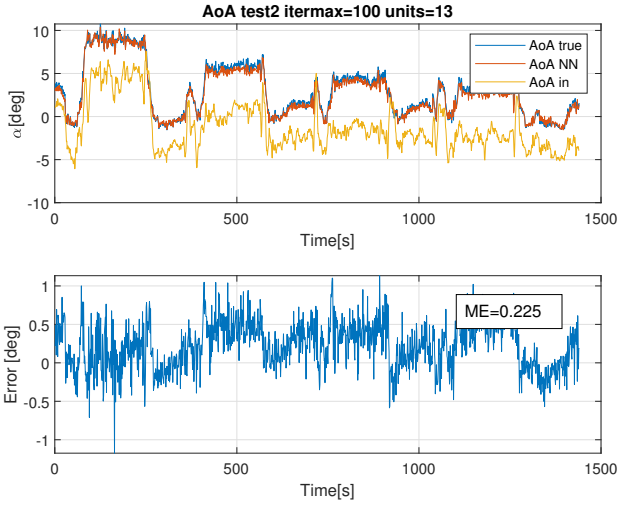


Figura 7.5: Test 2 AoA ottimale per la rete MLP. Il primo grafico mostra l'andamento dell'AoA stimato dalla rete neurale (in arancio), l'andamento dell' AoA reale (in blu), e l'andamento dell' AoA stimato con la teoria lineare (in giallo). Il secondo grafico mostra l'andamento dell'errore tra la stima della rete neurale e l'andamento reale. I parametri statistici sono da intendersi in gradi.

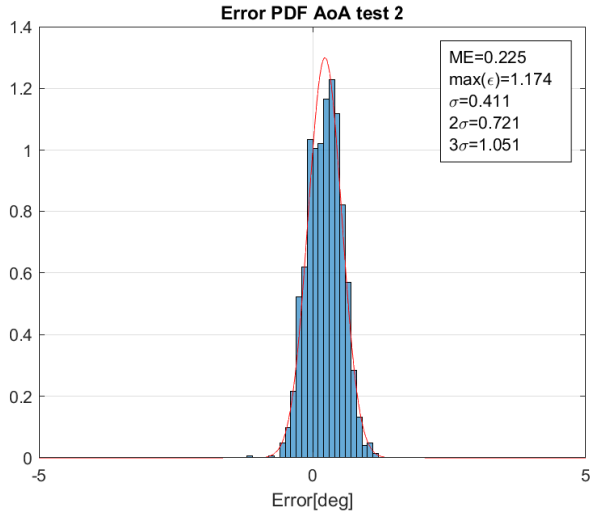


Figura 7.6: Funzione densità di probabilità dell' errore relativo ad AoA per il test 2 della rete MLP. I parametri statistici sono da intendersi in gradi. La curva rossa è la gaussiana che meglio interpola la distribuzione dell'errore ottenuta.

7.1. Stima di AoA con MLPnn e GRBFnn: confronto dei risultati su un G-70

Come già preannunciato, i risultati delle MLP sono ottimi, alla pari delle RBF, se non ancora migliori in questo caso. Ovviamente, è difficile stabilire univocamente quale delle due tipologie di reti sia migliore dell'altra: l'efficacia di una rete nella stima di una grandezza dipende dal tipo di problema considerato, dalla distribuzione dei dati di input nell'iperspazio, dal grado di corruzione a cui sono sottoposti per effetto del rumore e, infine, dal tipo di manovre utilizzate per l'addestramento e la validazione. Un confronto più approfondito, che premetterebbe di avere un quadro più chiaro delle performance di MLP e GRBF, dovrebbe essere condotto in condizioni più estreme di volo (tramite simulatore) per testare la capacità delle due reti di generalizzare i risultati anche al di fuori dell'inviluppo di volo.

		$\lvert ME \rvert$	$max(\lvert \epsilon \rvert)$	2σ
EMRAN GRBFnn	**Train**	0°	2.609°	0.658°
	Test1	0.487°	1.935°	1.121°
	Test2	0.613°	1.804°	1.276°
Matlab GRBFnn	**Train**	0°	4.191°	1.004°
	Test1	0.074°	2.054°	0.612°
	Test2	0.215°	1.466°	0.839°
MLPnn	**Train**	0.003°	2.029°	0.646°
	Test1	0.055°	1.448°	0.509°
	Test2	0.225°	1.174°	0.721°

Tabella 7.1: Risultati delle reti GRBF a confronto con risultati della MLP

Conclusioni

Questo documento riguarda lo sviluppo e la dimostrazione pratica di un potente approccio innovativo per la stima dell'angolo aerodinamico su velivoli ultraleggeri utilizzando sensori di dati aerei virtuali basati su tecniche predittive neurali. Sono state sviluppate due reti neurali *Generalized Radial Basis Functions* con l'obiettivo di prevedere l'angolo di attacco, elaborando dati inerziali, posizioni delle superfici di comando e pressione dinamica. Il target di accuratezza di più o meno 2 gradi per l'errore massimo (con riferimento alle condizioni di volo in ambiente reale) è stato considerato come criterio di *fail-pass*.

Nell'insieme del processo progettuale descritto in questo elaborato, i primi due capitoli sono stati utili a dare una visione generale del problema delineando le tutte le configurazioni possibili di ADS convenzionale e successivamente fornendo una panoramica sul mondo dei sensori sintetici con la finalità di comprendere come integrare un sensore sintetico all'interno dell'architettura ADS del velivolo. Il tutto mostrando i benefici di tale soluzione, rispetto a quelle convenzionalmente utilizzate.

Nel capitolo successivo si è fornita una panoramica sul funzionamento delle reti neurali nella loro generalità descrivendone la struttura, le principali tecniche di apprendimento, il problema dell'*overfitting* e il processo di generalizzazione.

Il capitolo 4 è il centro del lavoro e contiene tutte le considerazioni teoriche sulle reti RBF: in questo capitolo si costruisce una conoscenza capillare, sotto tutti i punti di vista, delle suddette reti che consente di individuare punti forti e punti deboli. Si parte dal teorema di Cover che descrive la loro capacità intrinseca di separare linearmente i pattern di input proiettandoli in uno spazio ad alta dimensionalità; si passa per la definizione dell'approccio di Tichonov che consente di definire tali reti auto-regolarizzate per poi descrivere la procedura che le rende reti 'generalizzate' permettendo di ac-

cettare in ingresso un numero di input diverso dal numero di unità nascoste e di scegliere una serie di centri arbitrari per il posizionamento delle unità stesse. Si continua elencando le tecniche di selezione dei centri e di ottimizzazione delle reti per poi concludere con l'approccio probabilistico al problema dell'interpolazione e il metodo della regressione del Kernel.

Nel capitolo 5 si individua la procedura generale per la costruzione di una rete GRBF volta alla determinazione dell'AoA utilizzando l'algoritmo EMRAN e il Toolbox di Matlab.

Infine, il capitolo 6 e 7 si concentrano sulla presentazione dei risultati e sul relativo confronto tra reti RBF e reti MLP.

I sensori virtuali sviluppati hanno dimostrato di essere accurati per l'intero range di valori in cui sono stati testati. Sulla base di quanto visto sia per via teorica che per via pratica è possibile confermare quanto preannunciato nell'introduzione e cioè che, alla pari delle MLPnn, le GRBFnn risultano essere degli ottimi 'approssimatori universali', particolarmente adatti per l'approssimazione di funzione multivariate di notevole complessità. Anche le reti RBF sono dunque delle perfette candidate per la progettazione di sensori virtuali che mirano a sostituire o a supportare (come banca dati a cui attingere per un confronto) i classici sistemi sensoristici costosi e ingombranti, soprattutto relativamente a velivoli ultraleggeri.

In conclusione, è possibile affermare che i risultati ottenuti sono estremamente promettenti e suggeriscono la necessità di indagare maggiormente in questa direzione per mettere a nudo tutti i pregi e difetti delle RBFnn applicate alla misura degli angoli aerodinamici. Si potrebbe ad esempio, suggerire di riproporre il seguente lavoro utilizzando dati di volo simulati utili a testare una rete di questo tipo in condizioni più estreme con lo scopo di comprendere le reali potenzialità di generalizzazione di codeste reti.

Bibliografia

[1] S. Haykin, Neural networks - a comprehensive foundation, 2005.

[2] A. Lerro, development and evaluation of neural network-based virtual air data sensor for estimation of aerodynamic angles, Politecninco di Torino, 2010-2012.

[3] L. Zadeh e J. Kacprzyk, Fuzzy Logic for the Management of Uncertainty, 1992-07-01.

[4] T. Kohonen, Competition and self organisation: Kohonen nets, part of Kevin Gurney's web-book on Neural Nets.

[5] T. Cover, «Geometrical and Statistical properties of systems of linear inequalities with applications in pattern recognition,» IEEE Transactions on Electronic Computers, Vol. 1 di 2EC-14, n. 3, p. 326–334, 1965.

[6] T. Girosi e F. Poggio, «Networks for Approximation and learning».

[7] A. N. Tychonoff, «On the stability of inverse problems in,» Doklady Akademii Nauk SSSR, vol. vol. 39, n. n. 5, p. pp. 195–198, 1943.

[8] M. J. L. Orr, Introduction to Radial Basis Fuctions Networks, Centre for Cognitive Science, Univesity of Edinburgh, 1996.

[9] A. Lerro, P. Gili, M. Battipede e A. Brandl, «Aerodynamic Angle Estimation: Comparison Between Numerical Results and Operative Environment Data,» CEAS Aeronautical Journal.

[10] P. Gili, A. Lerro, A. Brandl e M. Battipede, «A Data-Driven Approach to Identify Flight Test Data Suitable to Design Angle of Attack Synthetic Sensor for Flight Control Systems,» MDPI Aerospace Open Access Journal, 2020.

[11] G. Campa, M. L. Fravolini, B. Seanor, M. R. Napolitano, D. Del Gobbo, G. Yu e S. Gururajan, «On-line learning neural networks for sensor validation for the flight control system of a b777 research scale model,» International Journal of Robust and Nonlinear Control.

[12] L. Yan, N. Sundararajan e P. Saratchandran, «Analysis of Minimal Radial Basis Function Network Algorithm for Real-Time Identification of Nonlinear Dynamic Systems,» [Online]. Available: citeseerx.ist.psu.edu.

[13] A. Lerro, M. Battipede, P. Gili e A. Brandl, «Preliminary Design of a Model-Free Synthetic Sensor for Aerodynamic Angle Estimation for Commercial Aviation,» MDPI Aerospace Open Access Journal, 2019.

[14] Tiantian Xie, Hao Yu and Bogdan Wilamowski, Auburn University :«Comparison between Traditional Neural Networks and Radial Basis Function Networks» , conference paper, July 2011

[15] Neural Network Toolbox User's Guide, Matlab (`http://128.174.199.77/matlab_pdf/nnet.pdf`)

[16] Lu Ying Wei, N. Sundararajan e P. Saratchandran, «Radial Basis Function Neural Networks with Sequential Learning: MRAN and Its applications»