POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Laurea in
Ingegneria Informatica

UNA RETE NEURONALE ADATTATIVA SUPERVISIONATA PER IL MIGLIORAMENTO DELLA STIMA DELLA POTENZA PRODUCIBILE DA UN PANNELLO FOTOVOLTAICO

Relatore: **Prof. Sergio Cesare Vittorio BROFFERIO**

Tesi di Laurea di

Marco TRIESTE
Matr. 682238

Anno Accademico 2009 – 2010

Indice generale

Indice delle figure

Capitolo 1. Le reti neurali artificiali

Nonostante i computer moderni siano sempre più potenti e veloci, è ancora molto difficile utilizzarli per risolvere alcuni problemi che per gli esseri umani possono sembrare relativamente banali. Il riconoscimento di oggetti in situazioni quotidiane, la coordinazione motoria necessaria per spostarsi da una stanza ad un'altra e la valutazione contemporanea di un insieme di circostanze per poter prendere una rapida decisione sono tutti esempi di abilità che possediamo e che non comportano sforzi particolari. Tuttavia non esiste alcun programma per computer che sia in grado di ottenere prestazioni simili a quelle umane in questi compiti apparentemente semplici. Una delle possibili cause si basa sul fatto che il modo in cui i programmi elaborano le informazioni è radicalmente diverso dal modo in cui funzionano i sistemi biologici.

Il problema è che i sistemi di elaborazione tradizionali sono molto rapidi ed efficienti per la risoluzione di compiti in cui gli esseri umani trovano normalmente difficoltà (come ad esempio risolvere complessi calcoli matematici o memorizzare enormi quantità di dati) ma si rivelano particolarmente inefficienti e lenti nell'affrontare compiti più semplici dal punto di vista computazionale. Se osserviamo in maggior dettaglio la natura di questi due tipi di compiti, notiamo che i primi (quelli propri del mondo dei calcolatori) sono descrivibili da una serie di regole o procedure e possiedono una soluzione analitica, mentre gli altri sono difficili da descrivere attraverso regole esplicite, non sempre è possibile ricavarne una soluzione analitica e, anche se fosse possibile suggerire delle strategie, esse non necessariamente corrisponderebbero ai processi che di fatto vengono impiegati.

Questa differenza di rendimento trova riflesso anche nella struttura dei due sistemi di elaborazione sottostanti. Per esempio, al contrario del calcolatore seriale, il sistema nervoso centrale umano contiene circa 10^{11} elementi di elaborazione (i neuroni) ciascuno dei quali comunica in media con altri 10^4 elementi. Malgrado alcune differenze fisiologiche, è ragionevole ipotizzare che i neuroni funzionino pressappoco in modo simile: ciascuno di essi emette una

risposta in funzione del segnale globale ricevuto e della propria soglia di attivazione. Le analogie e le differenze potrebbero essere portate avanti in molti altri aspetti; l'importante è notare che il funzionamento di un sistema nervoso è decisamente diverso dal funzionamento di un sistema di elaborazione seriale dell'informazione. Le differenze principali riguardano i seguenti aspetti:

- l'elaborazione dell'informazione nei sistemi nervosi avviene in parallelo mentre nei calcolatori tradizionali ciascun dato viene elaborato individualmente e in successione. Malgrado il fatto che ogni singolo neurone sia relativamente lento, il parallelismo massivo comporta una maggior velocità del cervello nell'eseguire compiti che richiedono l'elaborazione contemporanea di un elevato numero di dati, come ad esempio il riconoscimento visivo di oggetti;
- l'elaborazione nei sistemi nervosi è distribuita su molti elementi, ovvero vi sono molti neuroni che si occupano della stessa operazione. L'osservazione di un sistema nervoso durante lo svolgimento di semplici compiti evidenzia l'attivazione contemporanea di molti neuroni, a volte organizzati in gruppetti locali, altre volte distribuiti "a macchie" in zone diverse del cervello. Inoltre un singolo neurone può prender parte in diversi tipi di operazioni eseguibili sia contemporaneamente che in tempi diversi;
- ogni dato nella memoria dei calcolatori è identificato da un indirizzo (in pratica un numero) che viene utilizzato dal processore centrale per recuperare le conoscenze necessarie allo svolgimento di un certo compito. Gli esseri umani invece accedono alle proprie memorie in base al contenuto: noi siamo in grado di recuperare un ricordo semplicemente in base a qualche indizio parziale o a un attributo (un profumo, una voce, una situazione simile);
- i sistemi nervosi, al contrario dei calcolatori, non devono essere programmati per svolgere un compito, bensì imparano autonomamente in base all'esperienza o con l'aiuto di dati esterni. Questo tipo di apprendimento consiste soprattutto nella modifica della forza delle connessioni attraverso cui i neuroni comunicano: quanto più una connessione (sinapsi) è forte, tanto maggiore sarà l'effetto del segnale

che vi passa sul neurone ricevente. Memorizzare un nuovo vocabolo, ricordare il viso di una persona o imparare come funziona una rete neurale artificiale consisterebbe quindi nel gioco di rafforzamento e indebolimento di un gran numero di sinapsi. Un calcolatore necessita invece di un programma che contiene tutte le istruzioni necessarie per portare a termine il compito correttamente, precisamente e infallibilmente.

In conclusione i computer seriali e i relativi programmi tradizionali sono degli strumenti molto potenti per svolgere dei compiti che richiedono la ripetizione di una serie di operazioni ben definite ove l'accuratezza, l'affidabilità e la velocità sono le caratteristiche importanti. Questi sistemi di elaborazione dell'informazione sono dunque molto utili, ma non certo intelligenti: l'unico elemento di intelligenza nell'intero processo è il programmatore che ha analizzato il compito e ha sviluppato il programma. Per questo motivo vale la pena studiare le modalità di elaborazione dell'informazione proprie dei sistemi nervosi biologici e cercare di analizzarle per capirne i principi di funzionamento.

Reti neurali artificiali (ANN: Artificial Neural Networks)

Le reti neurali (o neuronali) artificiali sono dei sistemi di elaborazione dell'informazione il cui funzionamento trae ispirazione dai sistemi nervosi biologici. Una rete neurale artificiale possiede molte semplici unità di elaborazione variamente connesse tra di loro: i neuroni. Alcune di queste unità ricevono informazioni dall'ambiente, altre emettono risposte nell'ambiente e altre ancora, se presenti, comunicano solamente con le unità all'interno della rete: esse sono definite rispettivamente unità di ingresso (input), unità di uscita (output) e unità nascoste (hidden). Ciascuna unità intende simulare il ruolo di un neurone o di un gruppo di neuroni nelle reti neurali biologiche: per questo motivo esse vengono anche definite impropriamente neuroni. Ciascuna unità svolge un'operazione molto semplice che consiste nel diventare attiva se la

quantità totale di segnale che riceve supera la soglia di attivazione. Se un'unità diventa attiva, essa emette un segnale che viene trasmesso lungo i canali di comunicazione fino alle altre unità a cui essa è connessa; ciascun punto di connessione agisce come un filtro che trasforma il dato ricevuto in un segnale inibitorio o eccitatorio aumentandone o diminuendone l'intensità a seconda delle proprie caratteristiche. Questi punti di connessione simulano le sinapsi biologiche, da cui prendono il nome; inoltre poiché il loro ruolo consiste nel "pesare" l'intensità dei segnali trasmessi, essi vengono comunemente definiti pesi sinaptici o semplicemente pesi.

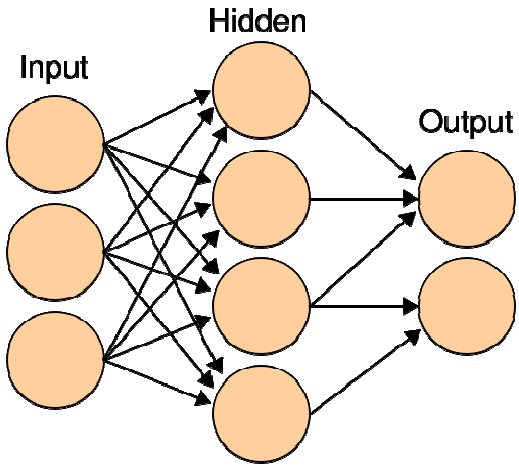

Figura 1 - Rete neurale con tre input, quattro unità hidden e due output.

Siccome ciascun nodo può ricevere in ingresso segnali da un gran numero di altri nodi o recettori ambientali ciascuno dei quali viene pesato dalla corrispondente connessione sinaptica, anche un semplice nodo è in grado di modellizzare comportamenti complessi. Un insieme di tali elementi connessi in rete può, in linea di principio, svolgere qualsiasi tipo di calcolo aritmetico e funzione. Quando uno stimolo (vettore o pattern di input) viene applicato ai neuroni d'ingresso della rete, i segnali viaggiano in parallelo lungo le connessioni attraverso i nodi interni fino ai nodi di uscita la cui attivazione rappresenta la risposta della rete neurale. Ciascun nodo elabora solo l'informazione locale: questo significa che esso si attiva solo in funzione

dell’informazione che riceve attraverso le proprie connessioni di ingresso, ma non sa né qual è lo scopo globale dell’elaborazione (a livello dell’intera rete) né quali operazioni vengono svolte dagli altri nodi ai quali non è collegato. La configurazione delle connessioni (architettura) e i valori delle sinapsi artificiali determinano in gran parte il comportamento e la risposta della rete.

Come detto precedentemente, una rete neurale impara a fornire le risposte appropriate per ciascun stimolo d’ingresso modificando i valori delle proprie connessioni sinaptiche (pesi) in base a delle regole di *apprendimento*. Solitamente la fase di apprendimento è graduale e richiede molte presentazioni di uno stesso esempio, anche se vi sono alcuni modelli di rete neurale che imparano in base a una singola presentazione di uno stimolo d’input. Vi sono tre grandi paradigmi di apprendimento, ciascuno corrispondente ad un particolare compito astratto di apprendimento: apprendimento supervisionato, apprendimento non supervisionato e apprendimento per rinforzo.

Apprendimento supervisionato (*supervised learning*)

Si utilizza quando si dispone di un insieme di dati per l'addestramento (o *training set*) comprendente esempi tipici d'ingressi con le relative uscite corrispondenti: in tal modo la rete può imparare ad inferire la relazione che li lega. La rete è addestrata mediante un opportuno algoritmo (tipicamente, la *backpropagation* che è appunto un algoritmo d'apprendimento supervisionato), il quale usa tali dati allo scopo di modificare i pesi ed altri parametri della rete stessa in modo tale da minimizzare l'errore di previsione relativo all'insieme d'addestramento. Se l'addestramento ha successo, la rete impara a riconoscere la relazione incognita che lega le variabili d'ingresso a quelle d'uscita, ed è quindi in grado di fare previsioni anche laddove l'uscita non è nota a priori; in altri termini, l'obiettivo finale dell'apprendimento supervisionato è la previsione del valore dell'uscita per ogni valore valido dell'ingresso, basandosi soltanto su un numero limitato di esempi di corrispondenza (vale a dire, coppie di valori *input-output*). Per fare ciò, la rete deve essere inoltre dotata di un'adeguata capacità di

generalizzazione, con riferimento a casi ad essa ignoti. Ciò consente di risolvere problemi di regressione o classificazione.

Apprendimento non supervisionato (*unsupervised learning*)

E' basato su algoritmi di addestramento che modificano i pesi della rete facendo esclusivamente riferimento ad un insieme di dati che include le sole variabili d'ingresso. Tali algoritmi tentano di raggruppare i dati d'ingresso e di individuare pertanto degli opportuni *cluster* rappresentativi dei dati stessi, facendo uso tipicamente di metodi topologici o probabilistici. L'apprendimento non supervisionato è anche impiegato per sviluppare tecniche di compressione dei dati.
In questa categoria di apprendimento ricadono le reti SOM (Self-Organizing Maps). Una mappa o rete SOM è basata essenzialmente su un reticolo o griglia di neuroni artificiali i cui pesi sono continuamente adattati ai vettori presentati in ingresso nel relativo insieme di addestramento. L'algoritmo può essere agevolmente descritto in un insieme di neuroni artificiali, ciascuno con una precisa collocazione sulla mappa rappresentativa degli output, che prendono parte ad un processo noto come *winner takes all*, al termine del quale il nodo avente un vettore di pesi più vicino ad un certo input è dichiarato vincitore, mentre i pesi stessi sono aggiornati in modo da avvicinarli al vettore in ingresso. Quando un nodo vince una competizione, anche i pesi dei nodi adiacenti sono modificati, secondo la regola generale che più un nodo è lontano dal nodo vincitore, meno marcata deve essere la variazione dei suoi pesi.

Apprendimento per rinforzo (*reinforcement learning, RL*)

Un opportuno algoritmo si prefigge lo scopo di individuare un certo modus operandi a partire da un processo d'osservazione dell'ambiente esterno; ogni azione ha un impatto sull'ambiente il quale produce una retroazione che guida l'algoritmo stesso nel processo d'apprendimento. Tale classe di problemi

ipotizza la presenza di un agente, dotato di capacità di percezione, che esplora un ambiente nel quale intraprende una serie di azioni; l'ambiente stesso fornisce in risposta un incentivo o un disincentivo, secondo i casi. Gli algoritmi per il reinforcement learning tentano quindi di determinare una politica tesa a massimizzare gli incentivi cumulati ricevuti dall'agente nel corso della sua esplorazione del problema.

Caratteristiche e impiego delle reti neurali artificiali.

Le reti neurali artificiali presentano alcune caratteristiche che si rivelano interessanti in molti campi di ricerca e domini di applicazione. Benché molte di queste caratteristiche varino da modello a modello, ve ne sono alcune sufficientemente generali.

Robustezza

Una rete neurale è resistente al rumore, ovvero è in grado di continuare a dare una risposta corretta anche se alcune delle sue connessioni vengono eliminate o lesionate o se viene aggiunto del rumore al segnale d'ingresso, ai canali di trasmissione o alla funzione di attivazione dei nodi. Questa proprietà è comune anche ai sistemi nervosi biologici dove la capacità di apprendere e ricordare non viene alterata in modo sostanziale dalla perdita continua di neuroni. In caso, le reti lesionate possono essere addestrate nuovamente per riacquistare le abilità perse.

Flessibilità

Un modello neurale può essere impiegato per un grande numero di finalità diverse: esso non ha bisogno di conoscere le proprietà del dominio specifico di applicazione perché le apprende in base all'esperienza. Questo non significa che un qualsiasi modello neurale possa essere utilizzato per tutti i tipi di compiti, ma implica che l'utente non deve necessariamente conoscere le soluzioni dettagliate e analitiche che caratterizzano il problema sotto indagine. In

generale l’utente di una rete neurale deve essere in grado di individuare precisamente le finalità del progetto, il tipo di compito e una serie di vincoli al fine di valutare qual è il modello neurale che risulta più appropriato.

Generalizzazione

Una rete neurale che è stata addestrata su un numero limitato di esempi è in grado di produrre una risposta adeguata a dei pattern d’ingresso che non ha mai visto in precedenza, ma che presentano tuttavia qualche somiglianza con gli esempi presentati durante la fase di addestramento. Questa proprietà deriva in parte dal fatto che molti modelli neurali rappresentano internamente un numero di associazioni stimolo-risposta più grande del numero di sinapsi disponibili; nel far questo la rete neurale tende a estrarre le caratteristiche invarianti dei pattern d'ingresso piuttosto che memorizzare ciascun singolo pattern.
La capacità di generalizzare a nuovi stimoli è una caratteristica molto apprezzata nei tipici campi di applicazione delle reti neurali dove spesso è impossibile ottenere una collezione esaustiva di tutti i dati su cui la rete neurale dovrà operare.

Per questi motivi, le ANN vengono impiegate in molti settori di ricerca ed in molti campi di applicazione, dall’informatica alla meccanica, dalla biomedica alla finanza, dalla matematica alla chimica. E’ possibile utilizzare una rete neurale artificiale per l’approssimazione di funzioni non lineari, regressione e predizione, per la classificazione in categorie (clustering), per il data mining, nei sistemi di controllo nel campo della robotica e nel controllo numerico, nel riconoscimento di sequenze e di modelli (immagini, caratteri o oggetti in generale).

Capitolo 2. Adaptive Resonance Theory

La risonanza adattiva (ART) è una teoria sviluppata da Stephen Grossberg e Gail Carpenter (datata 1976) basata su alcuni aspetti dell'elaborazione delle informazioni da parte del cervello umano. Essa descrive una serie di modelli di reti neurali che utilizzano metodi di apprendimento supervisionato e non supervisionato utilizzabili per problemi come il riconoscimento di oggetti (*pattern recognition*) e problemi di predizione e previsione.
Uno dei motivi principali dello sviluppo di questa teoria è il superamento del dilemma stabilità-plasticità delle reti neuronali: (1)

- plasticità: esprime la capacità di apprendere nuovi concetti o associazioni in seguito ai cambiamenti del mondo esterno o alla disponibilità di nuovi dati, diversi da quelli utilizzati nelle precedenti fasi di apprendimento;
- stabilità: significa non dimenticare quanto si è già appreso in passato, in particolare non dover addestrare nuovamente la rete sia con i nuovi dati sia con quelli vecchi ma ancora validi.

Grossberg (1) ha osservato che l'apprendimento competitivo e la maggior parte dei modelli neuronali non sono in grado di sviluppare una rappresentazione stabile quando esposti ad un ambiente in continua evoluzione. Se le caratteristiche degli ingressi subiscono delle modifiche nel corso del tempo, la risposta della rete a un determinato pattern sempre identico a sé stesso non resta costante; nel caso limite, se la distribuzione continua a mutare, la rete non raggiungerà mai un punto di stabilità. La plasticità dei pesi sinaptici fa sì che le esperienze di apprendimento più recenti cancellino drasticamente le conoscenze acquisite in precedenza: Carpenter e Grossberg hanno mostrato che in alcune situazioni questo fenomeno di instabilità può accadere anche con soli quattro pattern (2).

Contrariamente a questa limitazione dei modelli neurali artificiali, noi siamo in grado di riconoscere situazioni, persone e oggetti in ambienti che mutano costantemente e in modo imprevedibile e l'apprendimento di nuove conoscenze non cancella le acquisizioni precedenti. La definizione del dilemma della stabilità-plasticità è legata esattamente a questa ultima affermazione: vengono identificati quali sono i principi che permettono a un sistema di rimanere plastico in risposta a nuovi eventi significativi trascurando nel contempo variazioni irrilevanti e continuando a mantenere una rappresentazione stabile delle conoscenze già acquisite.

Tipi di reti ART

La teoria della risonanza adattiva rappresenta quindi una proposta formale per la soluzione del dilemma della stabilità-plasticità all'interno di un modello di apprendimento competitivo. Essa si basa su due principi fondamentali: l'esistenza di connessioni di feedback dall'output verso l'input e la presenza di un meccanismo di confronto fra l'attivazione dell'input sensoriale e l'informazione proveniente dalle unità di output. Le reti ART consentono un apprendimento continuo, on-line e con o senza supervisione, aggiungendo eventualmente nuovi nodi di riconoscimento (prototipi), quando quelli esistenti non riescono a classificare nuovi tipi di input, perché questi sono non familiari o nuovi. Nasce qui un nuovo dilemma tra un comportamento "conservatore", che filtra le novità ma è utile per categorizzare input inquinati da rumore, e un comportamento "innovativo" che tende ad aggiungere nuove categorie per input non del tutto familiari. Tale compromesso viene raggiunto generalmente con la scelta di un parametro denominato "paramento di vigilanza" (definito con la lettera dell'alfabeto greco ρ) il quale è fondamentale nella fase di matching durante la ricerca parallela all'interno delle reti ART.

Grossberg e Carpenter hanno posto grande attenzione alla plausibilità biologica e psicologica dei loro modelli: parte dei formalismi matematici si preoccupa proprio di descrivere le regole di transizione che avrebbero luogo nei circuiti

biologici e le proprietà del modello vengono sempre accuratamente confrontate con i dati sperimentali raccolti su soggetti. Sulla base di ciò sono stati proposti diversi algoritmi neurali, tra cui i principali ART-1, ART-2 e ART-3, per l'apprendimento e il riconoscimento di pattern: essi funzionano in modo simile, ma si distinguono per il crescente grado di generalità e dettaglio matematico. Ciascuno di essi è composto di una serie di moduli funzionali e unità altamente specializzate che intendono rappresentare determinati circuiti biologici. (1)

Le differenze sostanziali tra le diverse topologie di rete ART sono le seguenti:

- ART-1: è l'architettura più semplice e accetta solo input di tipo binario;
- ART-2: estende le caratteristiche della precedente consentendo di lavorare su ingressi di tipo analogico e continuo;
- ART-3: basata sull'architettura ART2, è provvista di un modulo aggiuntivo per il controllo della modifica dei pesi sinaptici e per la coordinazione tra i prototipi durante la ricerca parallela;
- fuzzy-ART: utilizza la logica fuzzy durante la fase di riconoscimento per aumentare la generalizzazione;
- ART-MAP: conosciuta sotto il nome di *Predictive ART*, è costituita dalla combinazione di due blocchi ART1 e ART2 con lo scopo supplementare di riuscire a modellare un parametro di vigilanza selettiva per effettuare una classificazione più accurata;
- fuzzy ART-MAP: è semplicemente la combinazione delle ultime due architetture descritte e ha come obiettivo l'aumento dell'efficienza.

ART-1: architettura e funzionamento

L'ART-1 è il modello più semplice ed è in grado di funzionare solo con input di tipo binario. Ciononostante la sua architettura ed i suoi principi di funzionamento rispecchiano completamente i concetti base della teoria della risonanza adattiva.

La rete si compone di due unità fondamentali. Il primo strato riceve l'input dall'ambiente esterno ed esegue il processo di comparazione mentre il secondo effettua il riconoscimento e rappresenta l'output del sistema. In figura si riporta lo schema basilare di tale architettura:

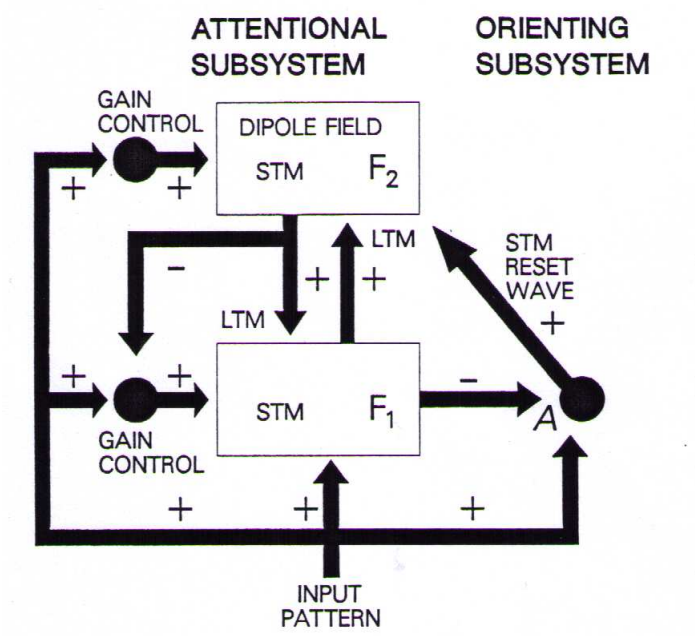

Figura 2 - Architettura ART-1

La rete è formata essenzialmente da due sottosistemi: *attentional* e *orienting subsystems*.

Il primo è composto da due stadi F1 e F2 in cui sono presenti i neuroni della rete, rispettivamente i neuroni di ingresso ed i neuroni target.

Viene effettuata la ricerca parallela dei pattern grazie a due tipi di memoria:

- STM - *Short Term Memory* che ha il compito di modulare il segnale di ingresso e di tenere traccia dell'input esaminato per verificare successivamente (attraverso LTM) un'eventuale corrispondenza con una base di conoscenza già acquisita;
- LTM - *Long Term Memory* che include l'insieme di tratti sinaptici tra F1 e F2 ed ha la funzione di inibire o attivare i neuroni appartenenti a quest'ultimo stadio.

Il secondo sottosistema genera segnali inibitori od eccitatori a seconda dei casi, diretti a F1 e F2. Per esempio può generare un segnale di reset per F2 per disattivare lo stadio nel caso in cui la ricerca non ha prodotto nessun risultato e si rende necessario creare una nuova base di conoscenza per quell'ingresso.

La ricerca parallela nella rete ART-1

Come detto, l'architettura ART-1 rispecchia interamente la teoria della risonanza adattiva e come tale anche la fase di ricerca di un pattern. Qui di seguito viene illustrata in modo semplice e conciso la ricerca di una base di conoscenza presente all'interno della rete, in modo da comprendere successivamente il modo di funzionamento della tipologia ART-2 che non si differenzia di molto.

La ricerca si compone di quattro fasi principali:

a. l'input *I* genera un modello *X* nello stadio F1 che a sua volta inibisce l'*orienting subsystem* e genera un segnale di output *S*. Attraverso le connessioni LTM tra F1 e F2, *S* viene trasformato in un segnale di ingresso *T* per F2 che a sua volta genera un modello *Y* all'interno dello stadio stesso;
b. il modello *Y* prodotto all'interno di F2 genera un segnale di tipo top-down *U* che simmetricamente alla fase a) viene trasformato in ingresso per F1 (segnale *V*). All'interno dello stadio F1 viene effettuato il matching tra *V* e l'input *I*: se non combaciano viene generato un nuovo modello *X** e viene riattivato l'*orienting subsystem*;
c. questa riattivazione genera un segnale di reset per F2 che cancella il segnale *U* e inibisce il modello *Y*;
d. successivamente all'inibizione di F2, il segnale originario *X* viene ripristinato in F1 con la conseguente generazione dei segnali *S* e *T*, come nella fase a); con il modello *Y* inibito, lo stadio F2 genera un diverso pattern *Y**, continuando con i segnali *U* e *V* come nella fase b)

ed effettuando ancora il matching nello stadio F1. Se i due modelli *X* e *Y** non corrispondono nuovamente, si continua la ricerca con altri modelli.

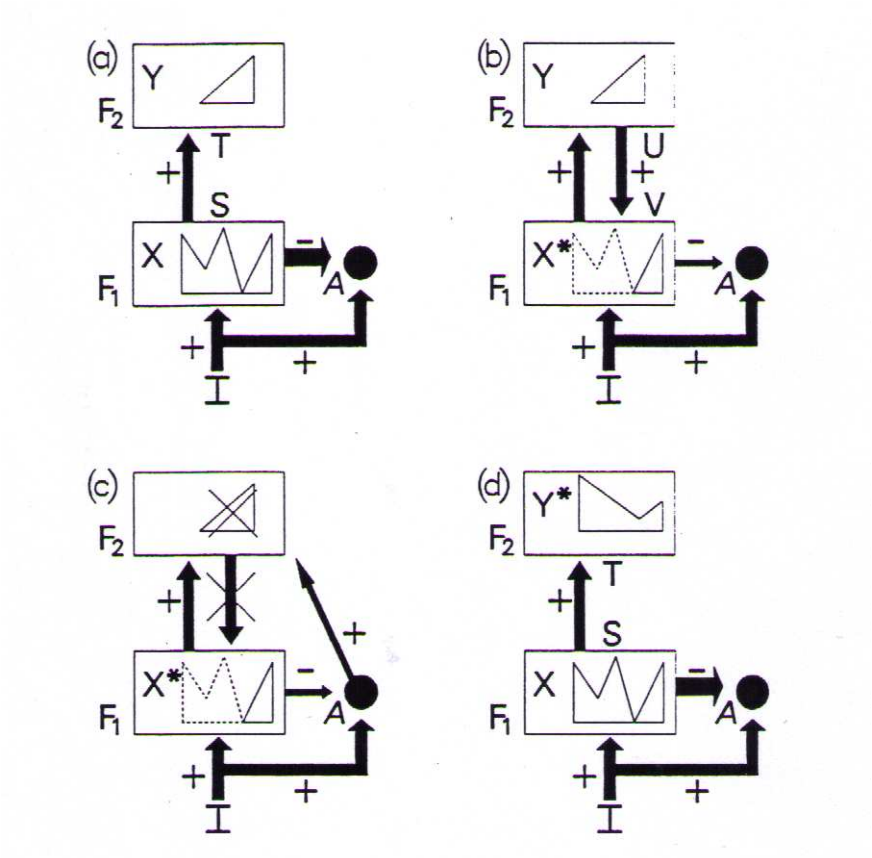

Figura 3 - Ricerca di pattern nelle reti ART-1

Dal modo in cui è descritta, questa ricerca sembra essere sequenziale, considerando un pattern alla volta all'interno di F2. Ciò è vero in parte perché sebbene l'inibizione dello stadio F2 è di tipo sequenziale e comandato dal sottosistema di *orienting*, la creazione dei segnali all'interno della LTM (quindi i segnali *S*, *T*, *U* e *V*) avvengono parallelamente per ogni pattern in F2.

Il livello di accuratezza nella fase di matching (alla fine della fase b della ricerca) dipende dal parametro di vigilanza selettiva ρ: in relazione alle formule matematiche utilizzate per lo stato di risonanza, questo parametro è di

fondamentale importanza durante la verifica tra i modelli perché il suo superamento implica la creazione di una nuova base di conoscenza. Infatti se la ricerca parallela non va a buon fine perché non è mai verificata la condizione di matching, la rete si trova davanti al caso in cui deve imparare un nuovo modello: nello stadio F2 quindi viene aggiunto un nuovo nodo che rappresenta questo nuovo pattern.

Una caratteristica molto importante della teoria della risonanza adattiva è che i pattern familiari, cioè quelli già classificati, vengono riconosciuti più rapidamente dei pattern nuovi a prescindere dalla loro dimensione e complessità perché vengono subito identificati dal nodo corretto; i pattern nuovi invece devono passare attraverso un processo di ricerca che si protrae più a lungo se il pattern presenta poche caratteristiche familiari o se è completamente nuovo. Questa proprietà rispecchia le modalità di riconoscimento degli esseri umani.
Infine, da un punto di vista tecnico, ART-1 richiede un numero di cicli di addestramento molto minore rispetto agli altri algoritmi competitivi; questa velocità viene però controbilanciata dal processo di risonanza che può essere relativamente lungo specialmente nei primi cicli di addestramento. (3)

ART-2: architettura e funzionamento

La rete ART-2 è concepita in modo tale da permettere di lavorare sia con input binari che con input continui e analogici. Lo schema della sua architettura è il seguente:

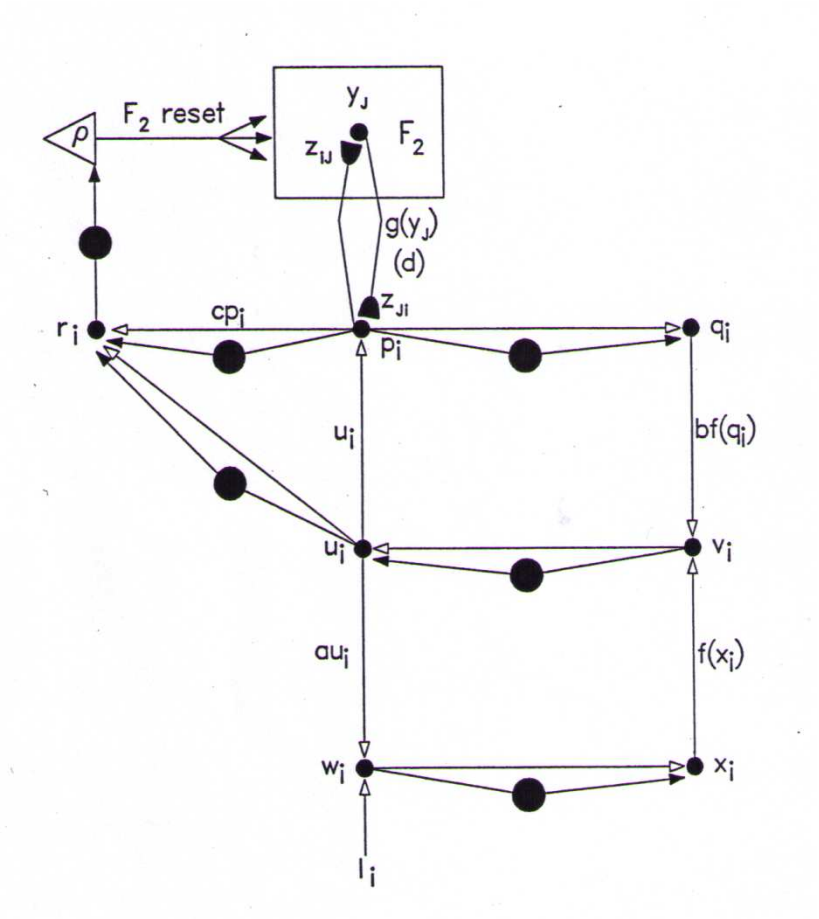

Figura 4 - Architettura ART-2

Teoricamente è simile all'architettura ART-1, con i due sottosistemi *orienting* (a sinistra) e *attentional* e con i due stadi F1 e F2, ma presenta molte diversità a livello di connessioni fra i vari moduli. La principale differenza consta nella maggiore complessità dello stadio F1: infatti per trattare input di tipo analogico, sia i segnali di ingresso che i segnali top-down provenienti dallo stadio F2 sono elaborati in diversi punti da diversi organi di controllo. Inoltre i segnali di feedback all'interno di F1 consentono il miglioramento della fase di matching e l'eliminazione del rumore.

Il funzionamento è concettualmente lo stesso della precedente versione. Sebbene sia architetturalmente più complessa, la rete ART-2 utilizza delle equazioni molto più semplici:

$$z_j = x_j + au \qquad (1)$$

$$q_j = \frac{z_j}{e + \| Z \|} \qquad (2)$$

$$v_j = f(q_j) + bf(s_j) \qquad (3)$$

$$u_j = \frac{v_j}{e + \| V \|} \qquad (4)$$

$$p_j = u_j + \sum_{j=0}^{m-1} g(y_j) w'_{ji} \qquad (5)$$

$$s_j = \frac{p_j}{e + \| P \|} \qquad (6)$$

$$f(x) = \begin{cases} x & x \geq \vartheta \\ 0 & x < \vartheta \end{cases} \qquad (7)$$

$$g(y_j) = \begin{cases} d & j = I \\ 0 & j \neq I \end{cases} \qquad (8)$$

Figura 5 - Equazioni di funzionamento della rete ART-2

Le espressioni *(2)*, *(4)* e *(6)* normalizzano l'ingresso al nodo (il valore *e*, infinitesimale, permette di normalizzare vettori di modulo molto piccolo). Ciò fa perdere completamente l'informazione riguardante il modulo del segnale da riconoscere ed utilizzare esclusivamente valori normalizzati.

Il vettore ***p*** viene inviato allo strato F2 mediante i pesi w_{ij}, cioè prototipi già riconosciuti precedentemente dalla rete. Questo stadio riconosce il prototipo che più si avvicina al vettore ***p*** in base alla massima correlazione con i prototipi già memorizzati e il prototipo vincente viene comunicato allo strato F1 attraverso i pesi in retroazione, i classici segnali top-down.

A questo punto il vettore filtrato *u* e il vettore vincente *c**p*** vengono passati all'*orienting subsystem* che verifica l'eventuale presenza della condizione di risonanza. In caso positivo, tutti i pesi *w* sono aggiornati al nuovo vettore di ingresso, altrimenti il pattern vincitore viene escluso e si passerà alla verifica del secondo vincitore. Nel caso in cui nessun prototipo riesca a superare il test di vigilanza, si va a creare un nuovo nodo nello strato F2, realizzando quindi l'adattamento della rete neuronale all'ambiente. (4)

Nelle sezioni seguenti viene descritto il tipo di rete utilizzato per il lavoro: si tratta di una rete SART, una versione semplificata dell'architettura ART-2. Vengono inoltre illustrati gli obiettivi del progetto e le scelte progettuali per l'implementazione.

Rete SART

Come detto in precedenza, è possibile impiegare una rete neurale per l'approssimazione di funzioni e per la classificazione dei dati, il clustering.
Per realizzare questo progetto è stata utilizzata una variante della rete neuronale ART-2: la rete SART, una variante più semplice da implementare e soggetta ad equazioni di tipo lineare.

Questo progetto nasce inizialmente dall'elaborato di altri due studenti del Politecnico (5), che nel loro corso di studi hanno implementato questa tipologia di rete nel linguaggio di programmazione C e l'hanno addestrata con dati provenienti da simulazioni e misurazioni. Lo scopo di questo lavoro è stato principalmente riprendere il suddetto elaborato, sviluppandolo nuovamente usando un altro linguaggio di programmazione, più dinamico e più adatto, estendendo caratteristiche e capacità ed aumentando la sua efficienza e le sue prestazioni.

Obiettivo e descrizione del progetto

Il progetto consiste quindi nell'implementare una rete adattativa supervisionata per migliorare la stima della potenza producibile da un pannello fotovoltaico. La potenza massima P_m prodotta da un pannello fotovoltaico dipende dalla radiazione solare G [W/m^2] e dalla temperatura ambiente T [°C] ed è una funzione non lineare dei due parametri: $P_m=f(G,T)$. In particolare la radiazione solare G può essere ulteriormente divisa in due parti distinte:

- la radiazione diretta $G_{diretta}$ che arriva direttamente sulla superficie del pannello sottoforma di raggi solari;
- la radiazione diffusa (detta anche indiretta) $G_{diffusa}$ che proviene dall'ambiente circostante attraverso riflessione e rifrazione degli stessi raggi solari da parte degli ostacoli.

L’obiettivo è determinare un modello lineare a tratti nello spazio vettoriale 4D nelle variabili $G_{diretta}$, $G_{diffusa}$, T e P_m utilizzando appunto una rete neurale adattativa del tipo ART2 addestrata con dati ricavati da simulazioni e misurazioni.

Lo schema a blocchi dell’intera catena di misurazione, di stima e correzione della potenza è riportata in figura:

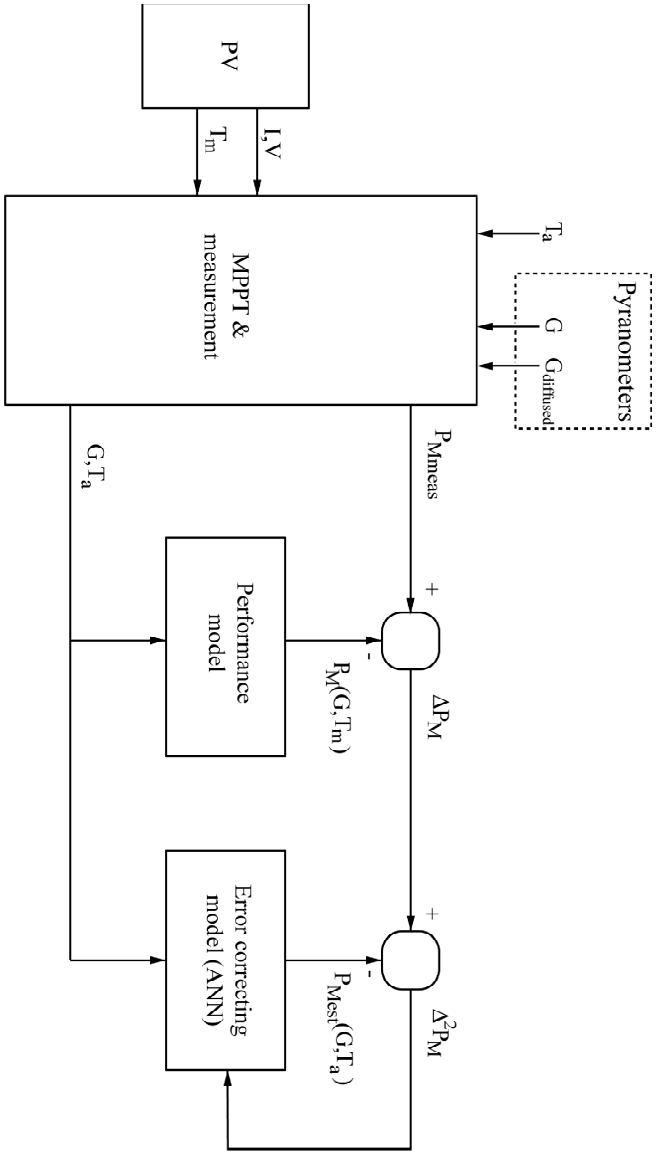

Figura 6 - Schema a blocchi del modello di misurazione e stima

La rete neuronale SART implementata, che si trova a valle del blocco di misurazione, ha lo scopo di classificare e correggere la stima della potenza del pannello attraverso un feedback calcolato come differenza tra potenza misurata e quella esterna. La rete accetta in ingresso come parametri la temperatura *T* e la radiazione *G* (che viene scissa nelle due tipologie diretta e diffusa) e la rete viene addestrata con il segnale $\Delta^2 P_M$.

Architettura della rete SART

Una versione semplificata dell'architettura della rete utilizzata è schematizzata nella seguente figura:

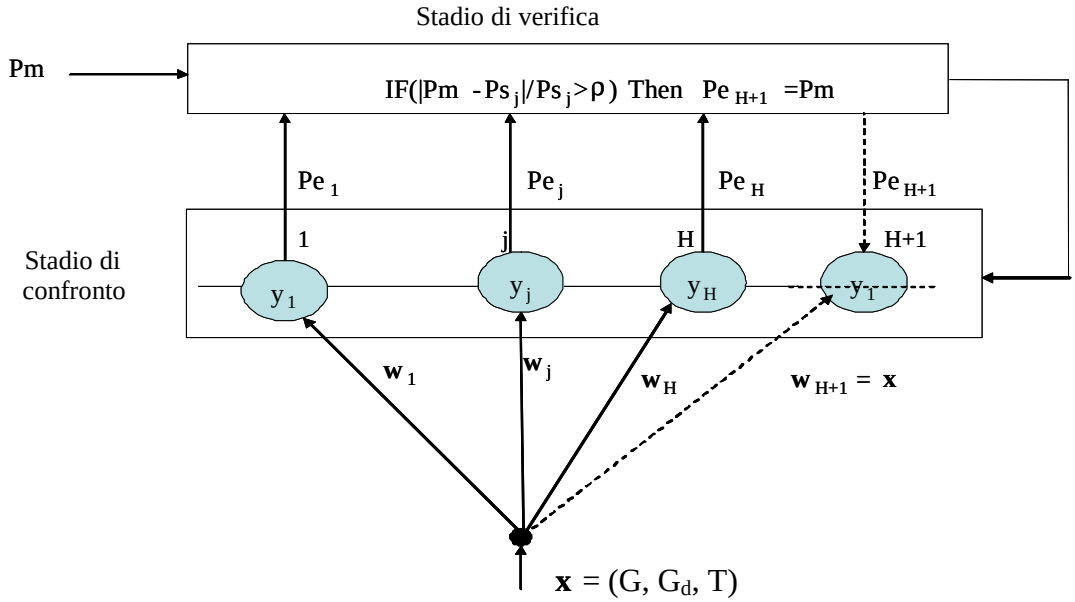

Figura 7 - Architettura della rete SART

Rispetto al modello generale dell'architettura ART-2, gli stadi F1 ed F2 sono leggermente modificati: nel primo viene effettuato solo il confronto tra i pattern mentre il secondo viene adibito esclusivamente alla verifica della risonanza ed

eventualmente all'invio di un segnale top-down per la generazione di una nuova base di conoscenza.

Il funzionamento del modello di rete considerato è piuttosto semplice ed è possibile dividerlo in fasi:

- fase di inizializzazione;
- fase di competizione;
- fase di calcolo;
- fase di matching;
- fase di pruning.

Vi è la fase iniziale di set-up della rete (inizializzazione) nella quale vengono creati i primi nodi con dei valori caratteristici degli ingressi. Successivamente con la fase di competizione vengono considerati gli ingressi in ordine casuale e si identifica l'insieme dei nodi più attivi, cioè più "vicini" nell'iperpiano della potenza. Quindi vi è la fase di calcolo del valore obiettivo che permette di determinare il valore della potenza (o del delta ΔP della potenza, a seconda dei casi e del training set) dell'ingresso considerato partendo dalle informazioni dei nodi più vicini; vedremo successivamente che questa operazione verrà effettuata quantificando il determinante di una matrice di ordine 5 e risolvendo un sistema di equazioni lineari. La fase consecutiva è la fase di matching, durante la quale la rete decide se l'input considerato può essere assimilato ad un prototipo già esistente oppure se è necessario creare un nuovo nodo perché l'input è nuovo o troppo diverso dai prototipi esistenti. Infine, con la fase di pruning, vengono eliminati i nodi ridondanti; vedremo che è possibile inserire quest'ultima fase a diversi livelli dell'algoritmo, con risultati più o meno differenti.

Capitolo 3. Descrizione dell'algoritmo

In questa parte viene illustrato in modo più approfondito l'algoritmo basato su rete neurale SART. Verranno fatte diverse simulazioni e verranno riportati i risultati dei test.

La piattaforma MatLab

Per implementare l'algoritmo è stato scelto l'ambiente di lavoro di MatLab, con l'uso dell'omonimo linguaggio di programmazione.
Originariamente il codice del funzionamento della rete neurale è stato sviluppato nel linguaggio di programmazione ANSI C: dopo una verifica dei requisiti e del tipo di sviluppo da realizzare è stato deciso di utilizzare MatLab per diversi motivi. Primo fra tutti è la semplicità con cui questa piattaforma consente di manipolare e gestire vettori e matrici e soprattutto variabili con dimensioni dinamiche. Per fare un esempio, nel codice C originale si deve ricorrere ad un ciclo *for* per il calcolo del valore minimo di un vettore; in MatLab si utilizza un solo comando per accedere a tale valore.
Nel programma originale (allegato) vengono infatti usati numerosi cicli per la lettura dei dati di input e per la successiva scrittura degli output, per la scansione e la lettura dei valori dei nodi della rete o per il calcolo delle distanze tra i nodi stessi. Tutto ciò viene fatto quasi in modo naturale nell'ambiente MatLab, considerato che i dati vengono trattati come matrici o vettori e il loro accesso e la loro lettura sono possibili in pochissime righe di codice. Ad esempio, nel codice C la rete vera e propria è rappresentata da un vettore di dimensione fissa di strutture prestabilite (*struct*) mentre con la nuova codifica la rete è trattata come una sola variabile matriciale.
Inoltre è possibile utilizzare variabili che crescono in modo dinamico molto più flessibilmente rispetto al C: si prenda come esempio la rete neurale che può

aumentare la sua dimensione perché aumentano le basi di conoscenza durante l'apprendimento.

Il codice MatLab implementato è costituito da diversi file *.m*:

- *genero*: per l'import e la generazione dei dati;
- *art2*: per la creazione e addestramento della rete;
- *pruning*: per applicare l'algoritmo di pruning;
- *errore_rete*: per il calcolo dell'errore.

Nella sezione seguente verranno spiegati in dettaglio il funzionamento di questi singoli script. Da notare che non sono stati omessi i commenti che in MatLab iniziano con il carattere speciale *%*.

Fasi principali del programma

Come detto precedentemente l'algoritmo è composto da cinque fasi principali:

- fase di inizializzazione;
- fase di competizione;
- fase di calcolo;
- fase di matching;
- fase di pruning.

Inizialmente è però presente la fase di generazione dei dati.

Per importare e generare il training set per la rete è stato realizzato un file MatLab chiamato *genero.m*: l'output di questo script porta alla creazione di un file di testo contenente i valori della radiazione diretta, della radiazione diffusa, della temperatura e della potenza. Per poter svolgere questo compito, inizialmente sono state utilizzate delle formule che legano P_m a $G_{diffusa}$, $G_{diretta}$ e T in funzione di altri parametri noti; in particolare:

$$P_m = (I_0 \cdot (G - G_d) + I_1 G_d) \cdot V_{oc} \cdot FF$$

$$V_{oc} = V_{oc0} + (V_{T0} \log G)$$

$$FF = \frac{\left(\frac{V_{oc0}}{V_{T0}}\right) - \log\left(\frac{V_{oc0}}{V_{T0}} + 0.72\right)}{\frac{V_{oc0}}{V_{T0}} + 1}$$

Solo successivamente sono stati ricavati dei dati sperimentali grazie ad un'altra unità che ha lavorato parallelamente ed il cui scopo principale è stato quello di fornire dati sempre più precisi e veritieri della temperatura *T* e della radiazione *G* in diverse condizioni di funzionamento.
Nelle ultime simulazioni infatti sono stati utilizzati questi ultimi valori utilizzando la differenza di potenza richiesta (ΔP) invece della potenza vera e propria, come precedentemente indicato nella sezione in cui viene descritto lo schema a blocchi del sistema di misurazione. Questi dati sono forniti in allegato.
Successivamente all'import e alla generazione del training set vi è la fase di inizializzazione della rete neurale. Vengono definiti alcuni parametri come la vigilanza selettiva ρ e il parametro di cambiamento dei pesi sinaptici η.

Inizializzazione

Dopo una prima verifica della consistenza dimensionale dei dati in ingresso (se la cardinalità del training set è troppo piccola il programma termina con un messaggio di errore), la rete viene creata. Vengono generati i primi quattro nodi prendendo valori significativi del set dei dati di ingresso in modo da poter poi iniziare a esaminare i rimanenti in modo casuale avendo già dei prototipi nella rete. Come nodi significativi sono stati considerati il valore massimo, il valore minimo e due valori intermedi nello spazio della P.
Per agevolare il calcolo dei quattro nodi iniziali viene effettuato l'ordinamento della matrice relativa al training set. Questa operazione non influisce sul successivo addestramento visto che gli ingressi verranno considerati in modo random.

Si riporta di seguito un estratto del codice MatLab relativo alla fase di inizializzazione della rete.

```
% verifica della consistenza dimensionale dei dati di
% ingresso
if(size(dati,1) < 4)
    fprintf('Dimensione dei dati insufficiente\n');
    return;
end

% inizializzo la rete con i 4 nodi caratteristici
dati = sortrows(dati,4);
medio1 = ceil(size(dati,1)*0.25);
medio2 = ceil(size(dati,1)*0.75);
rete = [dati(size(dati,1),:)' dati(1,:)' dati(medio1,:)'
dati(medio2,:)'];
% inizializzo Pms = Pm per i primi 4 nodi
rete = [rete; rete(4,:)];

% copio la matrice in un'altra per non perdere i dati
% originali ed elimino i 4 input già inseriti nella rete
dati_tmp = dati;
dati_tmp(size(dati,1),:) = [];
dati_tmp(medio2,:) = [];
dati_tmp(medio1,:) = [];
dati_tmp(1,:) = [];
```

Figura 8 - Codice MatLab per la fase di inizializzazione

Come detto, inizialmente è presente la verifica della dimensione dei dati in input. Successivamente la matrice dei dati viene ordinata in base alla P e vengono presi i quattro nodi caratteristici: il minimo (indice 1), i due mediani (indici medio1 e medio2) e il valore massimo (che si trova chiaramente alla fine, visto l'ordinamento crescente). Infine vengono eliminati questi quattro input dal training set dato che nella fase di competizione non sono più necessari.

Competizione

In questa fase vengono presi gli ingressi in modo casuale e si definiscono i quattro nodi più vicini (distanza euclidea), tra i quali compare il nodo più attivo cioè più vicino nell'iperpiano della potenza. Saranno questi i nodi che verranno poi utilizzati nella fase successiva per il calcolo della potenza stimata.
La relazione per il calcolo dei nodi attivi è la seguente:

$$J = argmin(|\boldsymbol{w}h - \boldsymbol{x}|) \;\; h = 1 \div H$$

L'insieme J sarà composto da quattro nodi appartenenti all'insieme H dei prototipi della rete.
Il motivo per il quale gli ingressi vengono presi singolarmente in modo random è presto spiegato: se dovessimo considerarli in ordine sequenziale la rete creata e addestrata sarebbe sempre la stessa. Inoltre viene usato il criterio della minima distanza Euclidea nello spazio 4D per valutare ed estrapolare i nodi più attivi al contrario del criterio della massima correlazione utilizzato nella rete ART-2 originale. (6)

La parte di codice relativa alla fase di competizione è la seguente:

```
% calcolo la distanza tra l'ingresso considerato e
% i nodi presenti nella rete in base alla Pm
distanza = abs(rete(4,:) - inputConsiderato(4,:));
% calcolo dei 4 nodi più vicini
k = 0;
indiciMinDistanza = [];
distanza_tmp = distanza;
while k < 4
    [minDist, indDist] = min(distanza_tmp);
    distanza_tmp(:,indDist) = Inf;
    indiciMinDistanza = [indiciMinDistanza indDist];
    k = k+1;
end
```

Figura 9 - Codice MatLab per la fase di competizione

Innanzitutto viene creato un vettore chiamato *distanza* in cui sono presenti i valori della distanza tra l'ingresso considerato e tutti i prototipi della rete; in seguito con un ciclo while questo vettore viene scandito per trovare i quattro nodi più attivi. Viene impiegata la variabile temporanea *indiciMinDistanza* per memorizzare gli indici di questi quattro nodi: alla fine del ciclo in questa variabile saranno presenti gli indici dei nodi della rete più vicini all'input considerato.

Calcolo

In questa fase viene calcolato il valore P_s per l'ingresso preso in esame: questo valore viene definito dall'interpolazione lineare dei quattro prototipi valutati nella tappa precedente. L'operazione viene eseguita costruendo la matrice dell'iperpiano relativo alla P_s stessa

$$\begin{vmatrix} G_{diretta} & G_{diffusa} & T & P_s & 1 \\ w_{11} & w_{12} & w_{13} & P_{e1} & 1 \\ w_{21} & w_{22} & w_{23} & P_{e2} & 1 \\ w_{31} & w_{32} & w_{33} & P_{e3} & 1 \\ w_{41} & w_{42} & w_{43} & P_{e4} & 1 \end{vmatrix} = 0$$

e ponendo il determinante pari a zero in modo da estrapolare il valore incognito della potenza.
Nella matrice ogni riga *j* rappresenta un nodo vicino a più attivo nell'iperpiano 4D (*j* = 1,2,3,4) e Pe_j è il relativo valore target, cioè il valore della potenza memorizzato nel prototipo.
Utilizzando il metodo di La Place per calcolare il determinante di questa matrice di ordine 5, l'incognita viene determinata attraverso una semplice equazione lineare.

Di seguito l'estratto di codice che implementa tale funzione.
La matrice (indicata con la variabile *m*) viene costruita riga per riga utilizzando la variabile *indiciMinDistanza* grazie alla quale è possibile accedere velocemente ai quattro nodi più vicini (determinati nella precedente fase di competizione). In seguito vengono calcolati i complementi algebrici c_{ij} dell'elemento P_s, che si trova nella posizione (1,4) della matrice, come previsto dal metodo di La Place.
Ponendo il determinante pari a zero ed utilizzando i complementi algebrici, è possibile ricavare l'incognita P_s grazie ad una semplice equazione.

```
% costruisco la matrice
uno = indiciMinDistanza(1,1);
due = indiciMinDistanza(1,2);
tre = indiciMinDistanza(1,3);
quattro = indiciMinDistanza(1,4);
riga1 = [inputConsiderato' 1];
riga2 = [rete(1:3,uno)' rete(5,uno) 1];
riga3 = [rete(1:3,due)' rete(5,due) 1];
riga4 = [rete(1:3,tre)' rete(5,tre) 1];
riga5 = [rete(1:3,quattro)' rete(5,quattro) 1];
m = [riga1; riga2; riga3; riga4; riga5];
% calcolo del deteminante con l'utilizzo dei
% complementi algebrici (metodo di LaPlace)
c11 = det(m(2:5,2:5));
c12 = det([m(2:5,1) m(2:5,3:5)]);
c13 = det([m(2:5,1:2) m(2:5,4:5)]);
c14 = det([m(2:5,1:3) m(2:5,5)]);
c15 = det(m(2:5,1:4));
% calcolo di Ps ponendo determinante = 0
Ps = 1/c14*(m(1,1)*c11-m(1,2)*c12 + m(1,3)*c13 + c15);
```

Figura 10 - Codice Matlab per la fase di calcolo

Matching

E' detta anche fase di risonanza visto che la rete persiste in questo stato fino a quando non viene trovato un pattern adeguato; in caso contrario viene creata una nuova base di conoscenza, cioè un nuovo prototipo all'interno della rete neurale. Inoltre in questo stadio viene preso in considerazione il parametro di vigilanza selettiva ρ per decidere se creare un nuovo prototipo o lasciare l'architettura della rete invariata e modificare solo i pesi sinaptici.
Fondamentalmente la fase di matching consiste in una verifica di una condizione matematica; dopo aver calcolato il valore interpolato P_s nella fase precedente, viene effettuato il seguente test:

$$\left|1 - \frac{P_m}{P_s}\right| < \rho$$

dove P_m è il valore della potenza dell'ingresso considerato.

Nel caso in cui la condizione fosse verificata non è necessario creare nessuna nuova base di conoscenza dato che l'input considerato può essere unificato al nodo più attivo: in pratica l'ingresso è simile ad un prototipo già esistente nella rete. Si aggiornano però i pesi sinaptici delle connessioni dello stadio di input e dello stadio di verifica in modo tale da avvicinare il prototipo all'ingresso considerato.
Le formule di aggiornamento dei pesi sono le seguenti:

$$\Delta \boldsymbol{w}_j = \eta_w(\boldsymbol{x} - \boldsymbol{w_j}), \text{ per i pesi di input}$$

$$\Delta P_{ej} = \eta_P(P - P_{ej}), \text{ per il pesi della classe target}$$

dove i valori η_w e η_P sono stati definiti nella fase di inizializzazione.

Se la condizione non fosse verificata è necessario modificare l'architettura della rete creando un nuovo prototipo, dato che i valori di ingresso si discostano troppo da qualsiasi nodo presente nella rete. Ci troviamo nel caso in cui è necessario classificare un nuovo input perché non familiare o completamente nuovo.

La generazione di un nuovo nodo (H+1) utilizza le seguenti formule:

$$\boldsymbol{w}_{H+1} = \boldsymbol{x}$$
$$P_{H+1} = P_m$$

La prima crea i pesi sinaptici per le connessioni dello stadio di input mentre la seconda formula crea il nodo vero e proprio ponendo il valore della potenza caratteristica dell'ingresso.

Come nelle fasi precedenti si riporta una sintesi del codice MatLab utilizzato.

```
% calcolo della precisione della stima
err = abs((input(4) - Ps)/Ps);

if (err <= RHO)
% se l'errore è minore di RHO la classe è già
% presente
    % aggiorno i pesi
    deltaW = ETAw*(input(1:3)-rete(1:3, uno));
    rete(1:3,uno) = rete(1:3,uno) + deltaW;
    % aggiorno il valore della P stimata
    deltaPe = ETAp * (Ps - rete(5,uno));
    rete(5,uno) = rete(5,uno) + deltaPe;
else
    % se l'errore è maggiore di RHO allora si crea
    % una nuova classe
    rete = [rete [input; input(4)]];
end
```

Figura 11 - Codice MatLab per la fase di matching

Si può notare immediatamente il modo agevole con cui vengono trattate le variabili matriciali e la facilità di una semplice condizione di *if/else* per la verifica della condizione di matching.

Pruning

La fase finale dell'intero algoritmo è il pruning: con questa operazione vengono eliminati i prototipi ridonanti o linearmente dipendenti.
Il valore target P_{eh} di ogni nodo *h* della rete è confrontato con il valore target dei quattro nodi più vicini dell'iperpiano 4D della potenza stimata. Questa rimozione viene effettuata in base alla relazione

$$\left|1 - \frac{P_{eh}}{P_{sh}}\right| < \rho$$

Il nodo *h* candidato al pruning viene eliminato:

- se per ogni nodo vicino la relazione è verificata;
- se nessuno dei suoi vicini è a sua volta candidato all'eliminazione.

La parte di codice che implementa questa funzione è stata inserita in un nuovo file nominato *pruning.m* e richiamato alla fine dello script di addestramento della rete.

```
k = 1;
while(k <= size(rete_pru,2))
    % nella variabile 'nodo' (indice k) è presente il nodo della
    % rete considerato per il calcolo dei nodi vicini
    nodo = rete(:,k);
    distanza = abs(rete_pru(5,:) - nodo(5));

    % ciclo while per individuare i 4 nodi vicini;
    % alla fine del ciclo, nella variabile 'indiciMinDistanza'
    % sono salvati gli indici dei 4 nodi più vicini
    i = 0;
    indiciMinDistanza = [];
    while i<5
        [minDist, indDist] = min(distanza);
        distanza(:,indDist) = Inf;
        indiciMinDistanza = [indiciMinDistanza indDist];
        i=i+1;
    end
    % elimino il primo che è il nodo considerato stesso
    indiciMinDistanza(1) = [];

    i = 1;
    while i <= 4
        Peh = nodo(5);
        Psh = rete_pru(5,indiciMinDistanza(i));
        errore(i) = abs((Peh - Psh)/Psh);
        i = i+1;
    end

    % elimino il nodo se la condizione di pruning è verificata
    if(errore < RHO)
        rete_pru(:,k) = [];
    end

k = k+1;
end
```

Figura 12 - Codice MatLab per la fase di pruning della rete

Pruning locale e globale

L'algoritmo di pruning sopra descritto viene messo in pratica solo alla fine della creazione della rete e dei suoi prototipi. Oltre a questo, sono stati realizzati altri tipi di pruning:

- per i dati iniziali
- durante fase di addestramento.

Il pruning dei dati iniziali consiste nel filtrare il training set per la rete e rimuovere i valori linearmente dipendenti affinché la rete possa mantenere un elevato grado di generalizzazione.
Per realizzare tale compito è stato utilizzato l'algoritmo di eliminazione di Gauss-Jordan che grazie ad alcune manipolazioni della matrice designa le linee linearmente dipendenti e pronte per una successiva eliminazione. La piattaforma MatLab non fornisce questo tipo di funzione, perciò il codice usato è stato prelevato dal sito degli sviluppatori indipendenti della MathWorks. In allegato è possibile trovare le poche righe di codice che compongono tale applicazione.

In un secondo momento invece è stato effettuato un pruning locale durante la fase di addestramento vera e propria: ad ogni iterazione è possibile che vengano generati prototipi linearmente dipendenti, sia nel caso in cui si crea una nuova base di conoscenza, sia nel caso in cui si aggiornano i pesi esistenti. Per ogni input considerato vengono determinati i quattro nodi più vicini, quindi viene utilizzata la relazione già nota

$$\left|1 - \frac{P_{eh}}{P_{sh}}\right| < \rho$$

Se un nodo risultasse dipendente dai vicini, l'algoritmo lo elimina reiterando il procedimento.

Analisi dei risultati

Sono state effettuate differenti simulazioni con il set di ingressi utilizzando diverse soglie di attenzione selettiva ρ, con l'obiettivo di verificare la relazione tra il numero di nodi creati dalla rete SART e il parametro stesso. Nella seguente tabella sono presenti alcuni valori caratteristici di ρ e il numero di nodi creati, durante una simulazione da 100 iterazioni:

ρ	numero nodi creati
0.01	27
0.05	18
0.1	14
0.2	12
0.5	8

Figura 13 - Tabella di confronto rho/numero nodi creati

Come si può notare, il numero di nodi della rete diminuisce con l'aumentare del parametro di attenzione selettiva: questo perché più questo parametro è elevato e più un nuovo ingresso verrà unito ad un prototipo già esistente nella rete.
Allo stesso modo, se ρ è piccolo il numero di nodi della rete aumenta. Il caso limite $\rho=0$ porta comprensibilmente alla creazione di una rete da 34 nodi, che è la cardinalità dell'insieme degli ingressi. L'altro caso limite è un valore di ρ molto grande che implica unicamente la creazione dei primi quattro nodi iniziali.

Nella seguente figura è riportato l'andamento grafico (realizzato con MatLab) del numero di nodi creati in funzione del parametro di attenzione selettiva variabile tra 0.01 e 0.9:

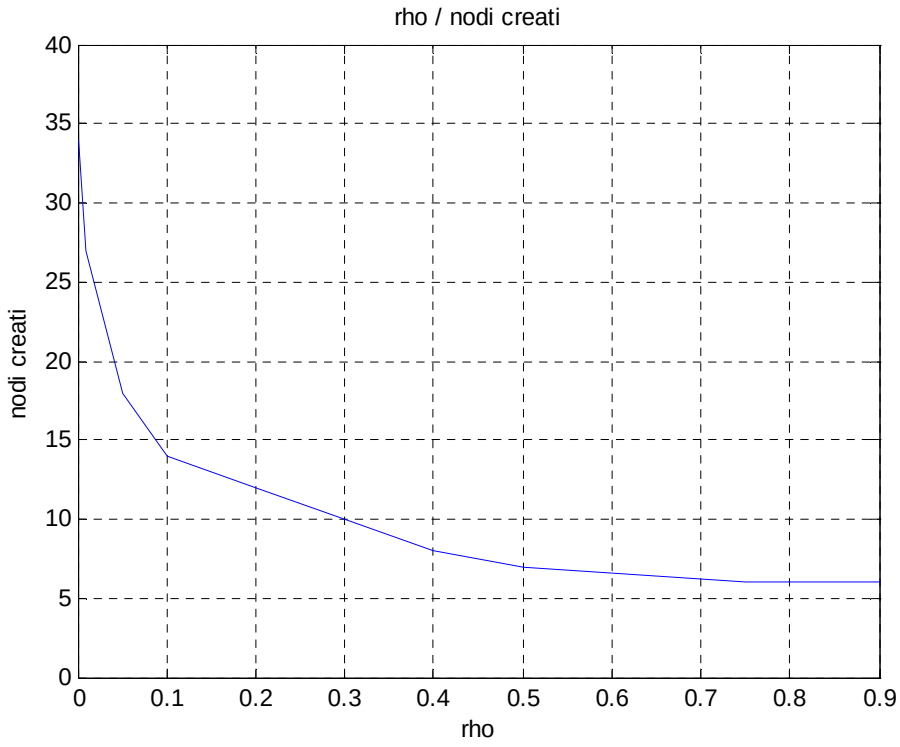

Figura 14 - Andamento grafico rho/numero nodi creati

Non sono state invece effettuate simulazioni modificando la dimensione del training set come nell'elaborato generale.

Analisi dell'errore

L'errore viene calcolato attraverso lo script *errore_rete.m* alla fine dell'intera computazione.
Di seguito si riportano le poche righe di codice MatLab che compongono tale funzione:

```
% calcolo dell'errore medio
errPotenza = abs(rete_pru(5,:) - rete_pru(4,:));
media = sum(errPotenza);
errMedio = media / size(rete_pru,2);
fprintf('L errore medio è: %f\n', errMedio);
```

Figura 15 - Codice MatLab per il calcolo dell'errore

In pratica viene calcolata la differenza tra i target dei prototipi e la potenza di input; successivamente questo risultato viene diviso per il numero di nodi della rete neurale generata per ottenere il valore dell'errore medio.

Nella tabella vengono riassunti i risultati delle simulazioni per il calcolo dell'errore, utilizzando sempre diversi valori di ρ ed effettuando 100 computazioni.

ρ	Errore (%)
0.01	0.035
0.05	0.20
0.1	0.34
0.2	0.53
0.5	0.88
0.9	1.85

Figura 16 - Tabella di confronto rho/errore medio

Come prevedibile l'errore si mantiene molto contenuto per valori molto bassi del parametro di attenzione selettiva ρ, dato che si vengono a creare molti nodi che possiedono una precisione maggiore. Al contrario, con un valore di ρ molto grande l'errore aumenta a causa della diminuzione della sensibilità durante la creazione dei prototipi.
Qui di seguito si riporta l'andamento grafico dell'errore medio considerando valori di ρ caratteristici, sempre compresi nell'intervallo 0.01-0.9:

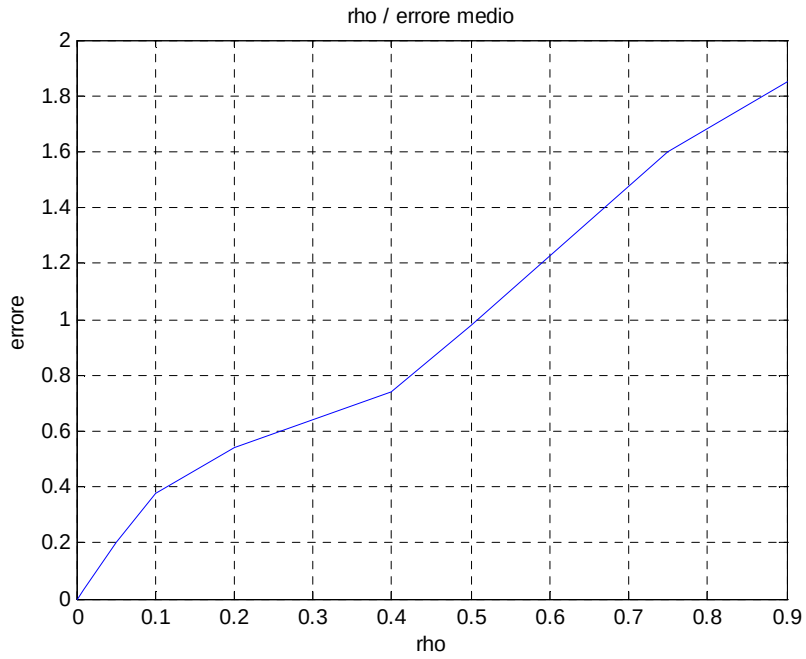

Figura 17 - Andamento grafico rho/errore medio

Posizione dei prototipi iniziali

Un'interessante variazione dell'andamento della rete e dei risultati ottenuti riguarda la scelta dei nodi iniziali nella fase di inizializzazione. (7)
Come descritto in precedenza durante l'algoritmo sviluppato la rete è stata inizializzata con dei valori caratteristici del training set: il valore massimo della potenza, il valore minimo e due valori intermedi. E' possibile scegliere questi primi nodi della rete in modo diverso:

- random
- selezionando i primi quattro del training set
- usando i quartili senza ordinamento del training set

Nonostante ci siano diversi modi per inizializzare la rete, i risultati non variano di molto. Anche se la rete viene addestrata con diversi set-up, la fase di pruning garantisce una convergenza quasi uniforme verso i valori calcolati in precedenza.

Nella tabella seguente vengono confrontati i vari metodi: (Nota: *s*: senza - *c*: con - *p*: pruning)

ρ	Standard	Random		Primi 4 dati		Quartili	
		s.p.	c.p.	s.p.	c.p.	s.p.	c.p.
0.01	27	31	29	30	27	27	27
0.05	18	20	20	25	21	19	19
0.1	14	15	14	17	14	15	15
0.2	12	14	13	13	13	12	12
0.5	8	9	8	10	9	10	10

Figura 18 - Tabella di confronto tra i metodi di inizializzazione

Graficamente si può osservare come con la fase di pruning i risultati hanno lo stesso andamento dei valori relativi al metodo standard:

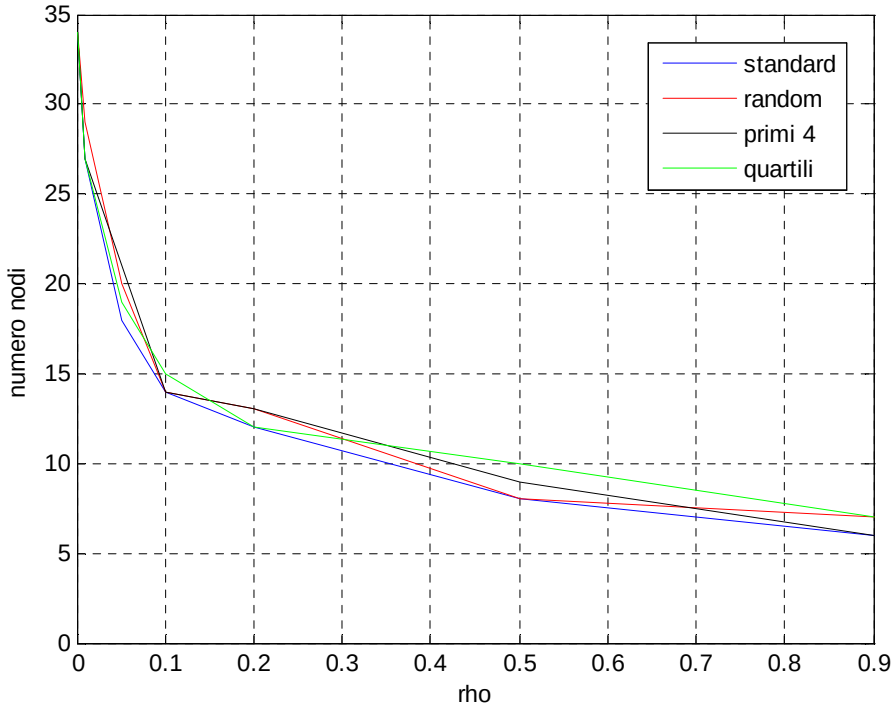

Figura 19 - Confronto grafico tra i metodi di inizializzazione

Performance

Come misura di prestazione è stato preso il tempo totale di esecuzione del programma nella sua interezza, considerando quindi il tempo di esecuzione di tutti i file MatLab in ordine:

- genero.m
- art2.m
- pruning.m
- errore_rete.m

Sono stati fatte delle simulazioni considerando sempre lo stesso training set di cardinalità 34 ma variando il numero di epoche di apprendimento della rete. Per numero di epoche si intende il numero di volte che la rete viene addestrata con il training set.
Nella seguente tabella vengono riportati i risultati:

ρ	Numero epoche	Tempo esecuzione [sec]
0.01	1	0.286
	10	0.433
	100	2.551
	1000	22.577
0.1	1	0.25
	10	0.409
	100	1.928
	1000	17.795
0.9	1	0.243
	10	0.373
	100	1.574
	1000	14.75

Figura 20 - Tempo di esecuzione del programma

Si può notare come all'aumento del parametro di attenzione selettiva il tempo di esecuzione diminuisce sensibilmente perché la ricerca di pattern familiari dà quasi sempre esito positivo e non è necessario creare nuovi nodi nella rete (che ha dimensione molto piccola, dell'ordine di 6/7 nodi).

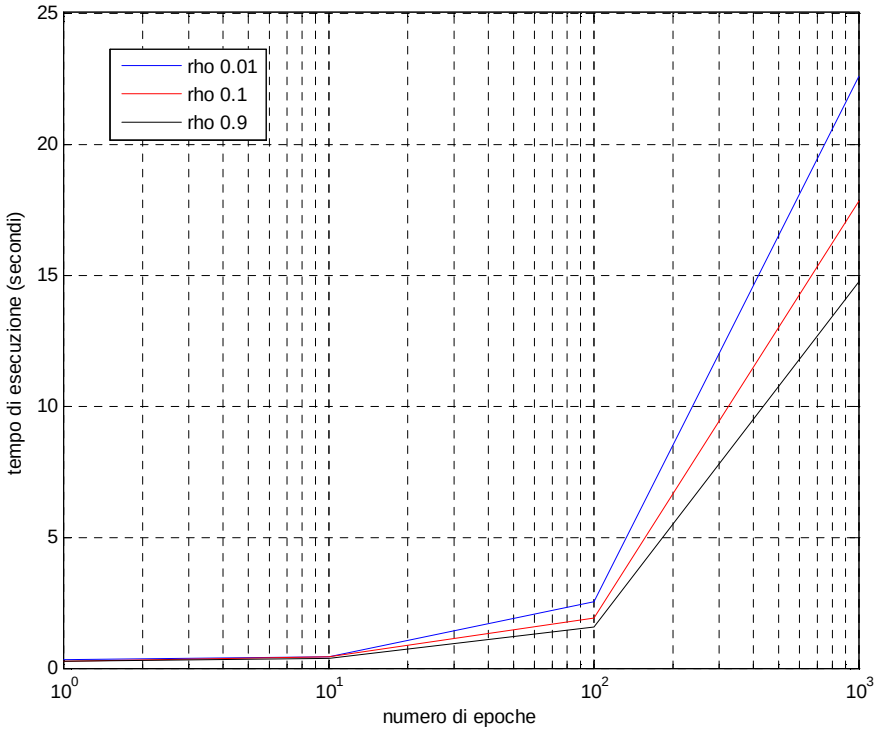

Figura 21 - Andamento grafico del tempo di esecuzione

Da un punto di vista della differenza di precisione, inoltre, è stato notato che quest'ultima varia di pochissimi centesimi di punto percentuale tra l'addestramento con 100 epoche e l'addestramento con 1000 epoche. Per questo motivo tutte le simulazioni nel progetto sono state eseguite utilizzando il valore di 100 epoche.

In ogni caso sono valori di tempo molto bassi confrontati con il tempo di esecuzione del codice C originale. Ciò è causato da due principali fattori: la gestione di MatLab di matrici e vettori e la gestione della presentazione degli ingressi alla rete da addestrare.
Come detto precedentemente la piattaforma MatLab gestisce variabili di tipo vettore e matrice in modo natio. A differenza del linguaggio C, con MatLab non è necessario effettuare nessun ciclo per scandire vettori o ricercare valori

all’interno delle matrici: tutte queste funzioni sono integrate nell’ambiente di lavoro, consentendo una maggior efficienza spaziale e temporale.
In secondo luogo è stata cambiata la gestione di addestramento della rete. Nel codice C gli input vengono presi in ordine casuale grazie all’ausilio di una variabile binaria dedicata “*visitato*” che indica se l’ingresso in questione è già stato presentato alla rete per il suo addestramento. Questo metodo non è molto efficiente perché per visitare tutto il training set può passare molto tempo, soprattutto quando rimangono pochi ingressi da considerare. La media del tempo di esecuzione del codice in C è di circa 9 minuti per una sola epoca. Usando il linguaggio MatLab è stato modificato questo collo di “bottiglia”. In pratica tutto il set di input viene copiato in una variabile temporanea (per non perdere la matrice originale ed evitare nuovamente la lettura da file) ed ogni ingresso che viene preso (sempre in modo casuale) viene a sua volta cancellato per evitare di considerarlo nuovamente. A fine ciclo la variabile temporanea utilizzata sarà una variabile vuota perché tutti gli ingressi sono stati presi in considerazione e utilizzati per l’addestramento. Questa procedura velocizza di molto il tempo di esecuzione dato che nel set di input temporaneo non ci sarà mai un ingresso già “visitato”.
Tra i due elementi descritti che influenzano le performance, sicuramente quest’ultimo incide molto di più sulle prestazioni globali.

Conclusioni e sviluppi futuri

Le reti neurali rappresentano un potentissimo modello matematico per realizzare algoritmi e funzioni troppo complesse da codificare in modo seriale. In questo progetto è stata usata una particolare tipologia di rete neurale (SART) per la generazione di un modello lineare a tratti nello spazio 4D della temperatura, della radiazione e della potenza associati ad un pannello fotovoltaico. Attraverso il suo addestramento è stato possibile classificare in prototipi un insieme di input provenienti da simulazioni sul campo.

I risultati ottenuti sono particolarmente positivi e accurati. Prima di tutto sono state migliorate le performance globali dell'intero programma, abbattendo di molto il tempo di esecuzione grazie a diversi accorgimenti durante lo sviluppo. Inoltre anche dal punto di vista della precisione, è stato raggiunto un traguardo non indifferente, visto che l'errore di computazione si è mantenuto molto basso, al di sotto dell'1% per valori di ρ fino a 0.5.

L'algoritmo proposto è composto da diversi file *.m*. Si è cercato di mantenere una certa modularità in corrispondenza delle varie fasi dell'addestramento, in particolare inizializzazione, calcolo e matching e infine pruning. Un utilizzatore può effettuare simulazioni modificando solo il file di pertinenza senza andare a leggere decine di righe di codice inutilmente. Inoltre vista la diffusa non familiarità con l'ambiente MatLab, si è deciso di scrivere una piccola guida per la comprensione e per poter utilizzare il codice con facilità effettuando simulazioni e test con diversi parametri o con diversi training set. Tale guida è annessa in allegato.

Possibili miglioramenti del codice e nuove funzionalità sono di seguito proposti:

- implementazione di un'interfaccia grafica per permettere l'inserimento più agevole dei parametri della rete e per visualizzare alla fine dell'addestramento i prototipi della rete stessa;
- la possibilità di effettuare un aggiornamento differente dei pesi sinaptici: invece di modificare solo il prototipo più vicino si può pensare di usare una funzione radiale in modo da modificare i pesi dei prototipi all'interno di un certo raggio prefissato.

In definitiva si è cercato di rendere il codice più pulito e semplice possibile in modo da poter continuare il lavoro nel caso di nuovi esperimenti o nuovi e diversi dati di ingresso o nel caso in cui l'utilizzatore finale non è un informatico o uno sviluppatore, figure che hanno una certa dimestichezza con questo linguaggio di programmazione.
Grazie alla sua modularità, il programma permette di usare la rete SART implementata con molti più input rispetto a quelli usati in questo elaborato. Questa funzionalità sarà molto utile nel caso in cui ci saranno da considerare altre variabili esterne oltre a temperatura, radiazione e potenza del pannello.
Sullo stesso piano, i valori target potranno essere valori di potenza oppure altre variabili. Il procedimento di addestramento della rete è sempre identico.

Bibliografia

1. **S. Grossberg.** *Competitive learning: from interactive activation to adaptive resonance.* Boston University, 1987.

2. **S.Grossberg.** *Pattern recognition by self-organizing neural networks.* Cambridge : MIT Press, 1991.

3. **G.A. Carpenter, S. Grossberg.** *ART2: Self-Organization of Stable Category Recognition Codes for Analog Input Pattern, Applied Optics.* 1987.

4. **D.Floreano.** *Manuale sulle reti neurali.* Il Mulino, 1996.

5. **S. Wei, S. Sun.** *A Magnitude-based ART classifier: Structure and Algorithms.* 2006. The Sixth World Congress on Intelligent Control and Automation.

6. **Locatelli, Mangioni.** *Una rete neuronale supervisionata adatattiva per la stima della potenza prodotta da un pannello fotovoltaico.* Politecnico di Milano, 2010.

7. **L.Barletta.** *Architettura ART2: riconoscimento in modulo e fase di segnali di tempo continui.* Politecnico di Milano, 2008.

8. **Presicce, Raimondi.** *Interpolazione lineare mediante rete SART e metodo di pruning.* Politecnico di Milano, 2009.

Allegato 1 - File *art2.m*

```
% Programma di addestramento della rete

% define del parametro di attenzione selettiva
RHO  = 0.1;
% define del parametro di cambiamento pesi ETAw
ETAw = 0.5;
% define del parametro di cambiamento target ETAp
ETAp = 0.5;

% richiamo il file genero.m per generare i dati di input
genero;


% FASE DI INIZIALIZZAZIONE

% verifica della consistenza dimensionale dei dati di ingresso
if(size(dati,1) < 4)
    fprintf('Dimensione dei dati insufficiente\n');
    return;
end

% inizializzo la rete con i 4 nodi caratteristici
dati = sortrows(dati,4);
medio1 = ceil(size(dati,1)*0.25);
medio2 = ceil(size(dati,1)*0.75);
rete = [dati(size(dati,1),:)' dati(1,:)' dati(medio1,:)'
dati(medio2,:)'];
% inizializzo Pms = Pm per i primi 4 nodi
rete = [rete; rete(4,:)];

% copio la matrice in un'altra per non perdere i dati originali
% ed elimino i 4 input già inseriti nella rete
dati_tmp = dati;
dati_tmp(size(dati,1),:) = [];
dati_tmp(medio2,:) = [];
dati_tmp(medio1,:) = [];
dati_tmp(1,:) = [];


% FASE DI COMPETIZIONE

% invece di usare una variabile dedicata per vedere se un nodo è
```

```
% già stato visitato (vedi codice C originale), elimino la riga
% corrispondente dalla matrice dei dati
while size(dati_tmp, 1) ~= 0
    % prendo random un ingresso tra i rimanenti
    i = randi(size(dati_tmp,1));
    inputConsiderato = dati_tmp(i,:)';
    % calcolo la distanza tra l'ingresso considerato e
    % i nodi presenti nella rete in base alla Pm
    distanza = abs(rete(4,:) - inputConsiderato(4,:));
    % calcolo dei 4 nodi più vicini
    k = 0;
    indiciMinDistanza = [];
    distanza_tmp = distanza;
    while k < 4
        [minDist, indDist] = min(distanza_tmp);
        distanza_tmp(:,indDist) = Inf;
        indiciMinDistanza = [indiciMinDistanza indDist];
        k = k+1;
    end
    clear distanza_tmp;


    % FASE DI CALCOLO

    % costruisco la matrice per il calcolo del determinante
    uno = indiciMinDistanza(1,1);
    due = indiciMinDistanza(1,2);
    tre = indiciMinDistanza(1,3);
    quattro = indiciMinDistanza(1,4);
    riga1 = [inputConsiderato' 1];
    riga2 = [rete(1:3,uno)' rete(5,uno) 1];
    riga3 = [rete(1:3,due)' rete(5,due) 1];
    riga4 = [rete(1:3,tre)' rete(5,tre) 1];
    riga5 = [rete(1:3,quattro)' rete(5,quattro) 1];
    m = [riga1; riga2; riga3; riga4; riga5];
    % calcolo del deteminante con l'utilizzo dei complementi
    % algebrici (metodo di LaPlace)
    c11 = det(m(2:5,2:5));
    c12 = det([m(2:5,1) m(2:5,3:5)]);
    c13 = det([m(2:5,1:2) m(2:5,4:5)]);
    c14 = det([m(2:5,1:3) m(2:5,5)]);
    c15 = det(m(2:5,1:4));
    % calcolo del target Ps ponendo determinante = 0
    Ps = 1/c14 * (m(1,1)*c11 - m(1,2)*c12 + m(1,3)*c13 + c15);

    fprintf('Pm: %f\t', inputConsiderato(4));
    fprintf('Ps: %f\t', Ps);


    % FASE DI MATCHING
```

```
    % calcolo della precisione della stima
    err = abs((inputConsiderato(4) - Ps)/Ps);
    fprintf('Errore: %f\t', err);

    if (err <= RHO)
        % se l'errore è < di RHO la classe è già presente
        fprintf('Nodo presente\n');
        % aggiorno i pesi
        deltaW = ETAw * (inputConsiderato(1:3)-rete(1:3, uno));
        rete(1:3,uno) = rete(1:3,uno) + deltaW;
        % aggiorno il valore della P stimata
        deltaPe = ETAp * (Ps - rete(5,uno));
        rete(5,uno) = rete(5,uno) + deltaPe;
    else
        % se l'errore è > di RHO allora si crea una nuova classe
        fprintf('Nuovo nodo\n');
        rete = [rete [inputConsiderato; inputConsiderato(4)]];
    end

    % elimino l'ingresso considerato dalla matrice
    dati_tmp(i,:) = [];
end

% scrittura su file
fprintf('\nScrittura della rete su file in corso...\n');
dlmwrite(NOME_FILE_RETE,rete','delimiter','\t','precision',...
                                                    '%.5f');
fprintf('Scrittura della rete su file compleatato.\n\n');

fprintf('Sono stati creati %d nodi nella rete\n', size(rete,2));

% richiamo il file pruning.m per l'algoritmo di pruning della
rete
pruning;

% richiamo il file errore_rete.m per il calcolo dell'errore
errore_rete;
```

Allegato 2 - File *genero.m*

```
% File per il caricamento dei dati dal file

% define per i nomi dei files
NOME_FILE_INPUT = 'dati_input.txt';
NOME_FILE_OUTPUT = 'dati_output.txt';
NOME_FILE_RETE = 'rete.txt';
NOME_FILE_PRUNING = 'pruning.txt';

% import dei dati dal file datiSim.txt
fprintf('\n\nImportazione dati per la simulazione in corso\n');
d = dlmread('datiSim.txt', '\t');
fprintf('Importazione dati per la simulazione completato.\n\n');

% costruisco la matrice dei dati
T = d(:,1);
Gdiretta = d(:,2)-d(:,3);
Gdiffusa = d(:,3);
P = d(:,4);

% nella variabile 'dati' sono presenti le quattro colonne:
% T, Gdiretta, Gdiffusa, P
dati = [T Gdiretta Gdiffusa P];
```

Allegato 3 - File *pruning.m*

```
% Script per eseguire il pruning della rete

fprintf('\nAlgoritmo di pruning in corso...\n');

% uso la variabile rete_pru come copia della variabile ‘rete’
% originale
rete_pru = rete;

k = 1;
while(k <= size(rete_pru,2))
    % nella variabile 'nodo' (indice k) è presente il nodo della
    % rete considerato per il calcolo dei nodi vicini
    nodo = rete(:,k);
    distanza = abs(rete_pru(5,:) - nodo(5));

    % ciclo while per individuare i 4 nodi vicini;
    % alla fine del ciclo, nella variabile 'indiciMinDistanza'
    % sono salvati gli indici dei 4 nodi più vicini
    i = 0;
    indiciMinDistanza = [];
    while i<5
        [minDist, indDist] = min(distanza);
        distanza(:,indDist) = Inf;
        indiciMinDistanza = [indiciMinDistanza indDist];
        i=i+1;
    end
    % elimino il primo che è il nodo considerato stesso
    indiciMinDistanza(1) = [];

    i = 1;
    while i <= 4
        Peh = nodo(5);
        Psh = rete_pru(5,indiciMinDistanza(i));
        errore(i) = abs((Peh - Psh)/Psh); %#ok<SAGROW>
        i = i+1;
    end
    % elimino il nodo se la condizione di pruning è verificata
    if(errore < RHO)
        rete_pru(:,k) = [];
        fprintf('Elimino il nodo %d\n',k);
    end

k = k+1;
end
```

```
% scrittura su file
dlmwrite(NOME_FILE_PRUNING,rete_pru','delimiter','\t',...
                  'precision','%.5f');

fprintf('Algoritmo di pruning completato\n\n');
fprintf('I nodi della rete dopo la fase di pruning sono:  ...
                  %d\n\n',size(rete_pru,2));
```

Allegato 4 - File *errore_rete.m*

```
% Script per il calcolo dell'errore della rete ART2

% calcolo l'errore medio sommando la differenza tra i valori
% della potenza e il valore target dei nodi e successivamente
% dividendo per il numero di nodi della rete
errPotenza = abs(rete_pru(5,:) - rete_pru(4,:));
media = sum(errPotenza);
errMedio = media / size(rete_pru,2);
fprintf('L errore medio è: %f\n', errMedio);
```

Allegato 5 - Schema di funzionamento

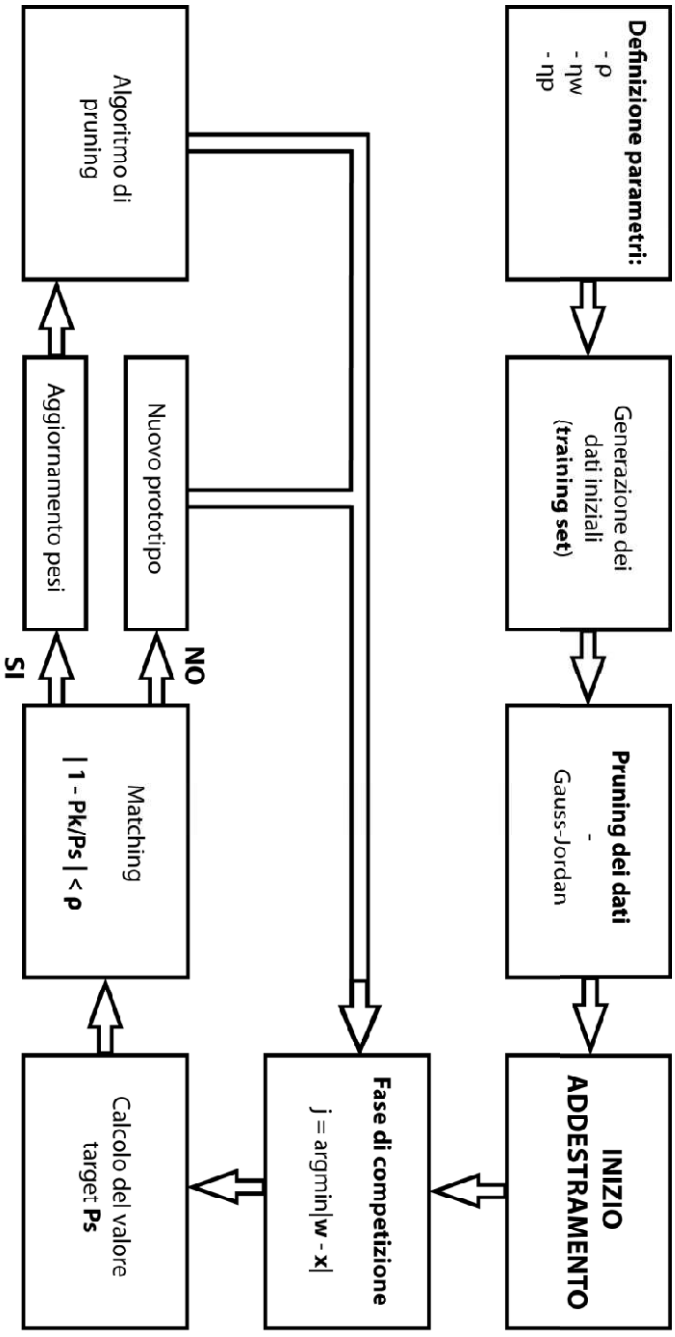

Figura 22 - Schema a blocchi dell'algoritmo

Allegato 6 - Dati utilizzati per la simulazione

N.	Ta [°C]	T mod [°C]	G [W/mq]	Gd [W/mq]		P mpp [W]	PM(G,Tm)		ΔPM
1	26,4	40,8	469	323		44	41,53545		2,464548
2	26	40,1	577	388		51,7	53,8191		-2,1191
3	25,9	41,2	529	380		48,2	48,15371		0,046289
4	25,7	39,1	342	308		33,3	28,14973		5,150273
5	26	39,1	422	306		38,7	36,63845		2,061546
6	26,3	39,2	416	340		38,6	35,97627		2,623727
7	26,5	34,5	376	296		34,1	32,23222		1,867779
8	23,4	45,7	868	428		65,2270242	86,94601		-21,719
9	23,7	43,7	850	160		65,94991303	85,37807		-19,4282
10	24,8	47,2	948	190		70,95126287	96,23783		-25,2866
11	24,9	46,2	912	188		72,60518892	92,16019		-19,555
12	24,4	47,1	1034	144		78,26176163	106,9459		-28,6842
13	25,1	50,4	1023	114		75,3070282	104,2909		-28,9839
14	25,2	49,3	1016	129		75,29959335	103,8517		-28,5521
15	25,5	48,2	1004	124		74,47666876	102,7868		-28,3101
16	25,2	48,6	1000	130		74,64515099	102,1403		-27,4952
17	25,4	48,1	1007	126		74,82896712	103,1966		-28,3676
18	25,5	47,6	997	124		74,70535512	102,1458		-27,4404
19	25,5	49,7	988	123		73,37700006	100,2491		-26,8721
20	25,5	47,8	1001	123		75,09666143	102,5664		-27,4697
21	25,6	48,6	990	123		73,49047921	100,9052		-27,4147
22	25,6	47	944	134		70,13067839	95,81614		-25,6855
23	25	49,1	949	133		70,32888836	95,68761		-25,3587
24	26,2	45,6	825	120		62,63281515	81,76332		-19,1305
25	26,4	44,5	800	117		61,30985438	79,07119		-17,7613
26	26,4	44,5	796	115		60,88224097	78,58944		-17,7072
27	26,3	46,3	779	105		59,48225074	76,04533		-16,5631
28	26	47,1	769	109		58,87421533	74,63615		-15,7619

29	26,1	44,7	768	109		58,3992007	75,17515		-16,776
30	26,3	44,8	758	110		57,70324772	73,95473		-16,2515
31	26,4	43,1	756	106		57,98875439	74,17331		-16,1846
32	26,4	43,8	723	103		55,0016758	70,05704		-15,0554
33	26,3	43,7	703	102		53,72357463	67,71724		-13,9937
34	26,3	42,7	699	100		53,89411737	67,48986		-13,5957