Lorenzo Cioni

Introduzione al Neural Computing

(work in progress)

Introduzione.

Scopo di queste note è quello di presentare alcune nozioni di base sul Neural Computing, descrivere brevemente alcuni modelli di rete neurale e illustrare i principali campi applicativi, con particolare attenzione alla Linguistica. Viene inoltre presentata una breve descrizione delle modalità di implementazione di reti neurali e viene esaminata in dettaglio l'architettura sia hardware sia software di un neurocomputer, in relazione al quale vengono presentati alcuni progetti di ricerca che fanno uso di reti neurali.

Il Neural Computing rappresenta il tentativo di definire modelli di calcolo che, sia pure in modo molto semplificato, simulino le funzionalità del cervello biologico.

L'analogia con il cervello biologico è di basso livello e si basa su considerazioni architetturali e funzionali.

Il neurone è considerato, infatti, l'elemento base del sistema. Di tale elemento base, il nucleo costituisce l'elemento di elaborazione mentre dendriti ed assone sono, rispettivamente, i canali di ingresso e il canale di uscita. L'assone trasmette successioni di impulsi mentre le connessioni fra neuroni avvengono mediante sinapsi la cui forza sinaptica ne determina l'efficacia. Le connessioni possono essere sia di tipo eccitatorio sia di tipo inibitorio. Le sinapsi rappresentano le unità di memoria del sistema e l'apprendimento si traduce in una modifica della forza sinaptica.

L'architettura dei sistemi neurali, d'altra parte, si basa sulla definizione del concetto di neurone formale (vedi figura 1) caratterizzato da canali di ingresso e di uscita, dai pesi che simulano le sinapsi e la loro efficacia e da funzioni matematiche che ne modellizzano il comportamento.

Da un punto di vista funzionale, le caratteristiche del cervello biologico che si ritrovano in parte nei sistemi neurali sono l'elevata interconnettività di elementi semplici e specializzati, l'elevato parallelismo e la ridondanza

Le reti neurali rappresentano pertanto modelli di calcolo che derivano alcune delle loro proprietà salienti dalle teorie sul sistema nervoso centrale. Sebbene un feedback fra Neuroscienze e Neural Computing sia indubbiamente utile non è strettamente necessario nella misura in cui le reti neurali sono viste come modelli astratti di calcolo e non come modelli del cervello biologico e della mente.

Breve digressione sulle reti neurali ([KliGu88]).

Una rete neurale può essere vista come composta da un insieme di elementi di elaborazione, detti processing element o PE, interconnessi fra loro. I PE sono l'analogo dei neuroni del cervello biologico, nell'ambito del modello astratto che stiamo esaminando. Ogni PE è caratterizzato da canali di ingresso (dendriti), da un canale di uscita (assone) e, in generale, da una funzione degli ingressi e da una funzione di trasferimento.

La funzione degli ingressi net_j (vedi figura 1) si limita in genere a sommare, con una somma pesata, i contributi dei singoli canali di ingresso in base alla seguente relazione:

$$net_j = \sum_{i=0}^{n} x_i w_{i,j}$$

nella quale i termini x_i indicano lo stato di attivazione dei canali di ingresso al PE j-esimo mentre i termini $w_{i,j}$ (i=1,.., n) individuano i pesi sulle connessioni rappresentate da tali canali e $w_{0,j}$ rappresenta il bias (in genere variabile e addestrabile), ovvero la polarizzazione, posto su un canale sempre attivo ($x_0 = 1$) in modo da simulare il valore di soglia dei neuroni biologici.

La funzione di trasferimento è ottenuta dalla composizione di una funzione di attivazione e di una funzione di uscita. La funzione di attivazione determina il nuovo valore dello stato di attivazione del PE in funzione del vecchio valore dello stato di attivazione e del valore

corrente di net_j. Considerando gli istanti di attivazione come disposti lungo una scala temporale discreta, quanto asserito si traduce nella relazione seguente:

$$a_j(t) = F_j(a_j(t-1), net_j(t))$$

che in molti casi si semplifica in una dipendenza stretta fra stato di attivazione corrente e valore degli ingressi pesati e cioè in $a_j(t) = net_j$.

La funzione di uscita, in genere non lineare, traduce il valore corrente dello stato di attivazione in un valore che viene propagato sul canale di uscita (o_j) verso tutti i PE a quali il PE in questione è connesso. Usando una terminologia data-flow se $\mathtt{I}_k$ è l'insieme di ingresso del PE k-esimo, il PE j-esimo invia il proprio token di output a tutti i PE tali che $o_j \in \mathtt{I}_k$.

La suddetta traduzione avviene in base alla seguente relazione:

$$x_j = f_j(a_j(t)) = f_j(\, F_j(a_j(t-1), net_j(t)))$$

Le funzioni non lineari più usate sono una funzione di tipo lineare a soglia, una funzione bistabile (valori -1/1) o binaria (valori 0/1) e la cosiddetta logistica o sigmoide.

Nel caso più semplice, in cui $a_j(t) = net_j$, la funzione di uscita trasforma la somma pesata degli ingressi in un segnale sul canale di uscita che si propaga agli altri PE.

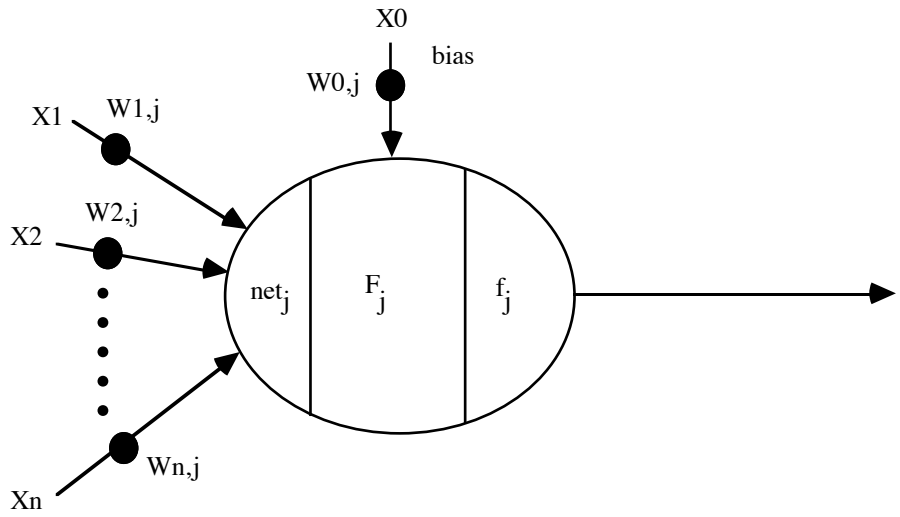

Figura 1. Modello di neurone formale (spiegazione nel testo)

La funzione di uscita pertanto è in genere non lineare, dato che una sua non linearità rende il modello più potente e considerato che, se la funzione di uscita fosse lineare, non avrebbe senso costruire reti neurali di più di due strati.

Le connessioni fra PE sono pesate con pesi positivi per connessioni eccitatorie e negativi per connessioni inibitorie e tali pesi rappresentano la forza sinaptica delle connessioni fra neuroni, forza che è oggetto di modifica durante le procedure di addestramento.

Le reti neurali sono di solito organizzate in layer o strati interconnessi fra di loro (vedi figura 2).

Di tali strati, quello che riceve dati dal mondo esterno si dice strato o buffer di ingresso o di input, quello che presenta i dati al mondo esterno si dice buffer di uscita o di output o di presentazione mentre gli altri si dicono strati interni o nascosti: compito del buffer di ingresso è quello di distribuire i pattern di ingresso[1] a tutti gli elementi degli strati interni, in funzione della topologia scelta. Gli strati sono numerati consecutivamente da quello di ingresso a quello di uscita.

Gli strati in una rete possono essere collegati senza o con feedback, nel primo caso si parla di reti feedforward nel secondo di reti riverberanti o con feedback di un PE su se stesso, su

1 Un pattern ha di solito la forma di un vettore o di una matrice i cui elementi assumono singolarmente valori in un qualche dominio $\mathbb{D}$. Gli insiemi $\mathbb{D}^n$ e $\mathbb{D}^{n \times m}$ rappresentano l'universo di tutti i pattern possibili. La funzione detta di pattern recognition permette di suddividere i pattern possibili in classi significative in modo che, dato un certo pattern, sia possibile assegnargli una classe di appartenenza.

elementi dello stesso strato o di strati precedenti. Nel caso di reti feedforward, inoltre, i PE di uno strato possono essere connessi solo con quelli dello strato successivo (feedforward puro) oppure no: in genere le reti più usate sono quelle del primo tipo ma ci sono topologie di rete (fra cui le reti ad accrescimento) che violano tale principio. La rimanente parte del paragrafo fa riferimento essenzialmente al caso di reti a topologia fissa.

Il funzionamento di una rete prevede due modalità: addestramento e recupero o produzione. La fase di addestramento permette di adattare i pesi presenti sulle connessioni in risposta a stimoli che vengono presentati sul buffer di input e sul buffer di output. Si hanno coppie di pattern corrispondenti, input e output desiderato, e con l'ausilio di tali coppie la rete viene addestrata. Si parla in questo caso di addestramento supervisionato o con teacher esterno.

Le coppie di pattern sono presentate alla rete ripetutamente in cicli detti epoche e l'addestramento viene interrotto quando l'errore che la rete commette sui pattern è sceso al di sotto di un certo valore oppure dopo un certo numero di epoche, caso questo in cui, se l'errore residuo è troppo alto, la rete non converge e cioè non riesce ad apprendere. In genere se i pattern di input e di output coincidono si parla di rete autoassociativa mentre, se sono diversi, si parla di rete eteroassociativa.

Nel caso in cui alla rete vengano presentati solo i pattern di input si parla di addestramento non supervisionato. Un altro esempio di addestramento è quello detto per rinforzo, in cui il teacher esterno non presenta i pattern desiderati sullo strato in uscita ma si limita a segnalare se la risposta è corretta o no.

In ognuno dei casi suddetti, l'addestramento si traduce nella modifica dei pesi sulle connessioni in risposta ad esempi di addestramento. Una differenza esistente fra l'addestramento supervisionato e quello non supervisionato è che il secondo, in genere, è incrementale nel senso che, una volta addestrata la rete su un set di pattern, la si può addestrare su una serie di pattern aggiuntivi senza ripetere l'addestramento dall'inizio, come si deve fare nel caso di addestramento supervisionato, e ciò senza che la rete rischi di dimenticare l'associazione di pattern già appresi.

L'apprendimento è guidato da regole che la rete implementa nelle proprie connessioni. I parametri che governano una regola di apprendimento possono variare nel tempo man mano che l'addestramento procede. Un parametro che tipicamente viene modificato in molte regole di apprendimento è quello che regola la cosiddetta velocità di apprendimento e che deve essere calibrato in modo da garantire procedure rapide ma stabili.

Durante la fase di richiamo, recupero o produzione la rete elabora un pattern presente al suo ingresso per produrre un corrispondente pattern in uscita. La fase di richiamo è usata anche durante l'addestramento supervisionato per confrontare l'output reale con quello atteso. In questo modo è possibile definire una funzione errore che (nel caso del metodo detto della backpropagation), propagata ricorsivamente su tutti gli strati fino a quello di ingresso, consente di modificare i pesi sulle connessioni in modo da minimizzare l'errore.

La forma di richiamo più semplice è presente nelle reti feedforward: in tali reti i pattern transitano dallo strato di input a quello di output attraverso gli strati nascosti e sono elaborati in base ai pesi sulle connessioni ed alle caratteristiche (funzione di ingresso e di trasferimento) dei singoli nodi, fino a produrre il pattern in uscita sullo strato di output.

Nel caso siano presenti connessioni in feedback, i pattern riverberano nella rete ripassando attraverso gli strati fino a che un predefinito criterio di convergenza non risulta verificato e il pattern di uscita viene prodotto.

Tale modalità richiede che si sia definita una strategia che controlla la successione in cui gli strati vengono elaborati. Una strategia possibile è quella di definire una funzione energia associata alla rete in modo che ogni stato abbia associato un valore di energia: durante la fase di addestramento vengono associati ai pattern da riconoscere valori di minimo dell'energia in cui la rete si deve portare durante la fase di produzione per operare riconoscimenti corretti.

Altre caratteristiche interessanti delle reti neurali riguardano sia il modo di funzionamento sia alcune operazioni che sono eseguite sui singoli strati.

Il modo di funzionamento può essere sia sincrono sia asincrono: nel primo caso tutti i PE della rete producono i valori di output allo stesso istante mentre nel caso asincrono (con o senza una linea di abilitazione/disabilitazione per ogni PE) i PE producono il loro output in modo random e indipendentemente uno dagli altri.

Fra le operazioni che coinvolgono gli strati sono degne di nota la normalizzazione e la competizione.

La normalizzazione è una operazione di scalatura del vettore che rappresenta l'output di uno strato della rete in modo tale che una funzione di tale vettore (ad esempio somma degli elementi) assuma un ben preciso valore. In questo modo i PE di uno strato aggiustano i propri valori in uscita in modo da realizzare un prefissato livello di attività.

Perché un generico strato possa eseguire tale operazione ogni PE dello strato deve essere connesso con tutti gli altri in modo che l'output di ogni PE sia calibrato sul valore corretto.

La competizione è una modalità di interazione di un PE con gli altri dello stesso strato: in caso di competizione fra PE, in genere, solo uno vince e produce un valore in output inibendo contemporaneamente gli altri PE dello stesso strato. La competizione è infatti ottenibile mediante l'inibizione laterale fra PE (uso di connessioni inibitorie fra PE di uno stesso strato).

Tipologie di reti neurali.

Quale che sia la topologia di una rete neurale, numerando in modo univoco in PE da 1 a N, indipendentemente dal layer (strato) in cui si trovano, è possibile descrivere la topologia con una matrice dei pesi W. Un esempio di matrice W è riportato nella figura che segue per il caso di una rete a tre layer (vedi anche figura 2). Nella figura sottostante i numeri individuano i layer che costituiscono la struttura di base della rete e le frecce la direzione delle connessioni fra i layer. La matrice è di ordine NxN e gli indici di riga individuano i PE sorgente mentre quelli di colonna i PE destinazione[2] in modo che $w_{i,j}$ individua il peso sulla connessione fra il PE i-esimo e il PE j-esimo.

La matrice è suddivisibile in blocchi $W_{i,j}$, dove i e j individuano rispettivamente il layer sorgente e quello destinazione.

$1 \rightarrow 1$	$1 \rightarrow 2$	$1 \rightarrow 3$
$2 \rightarrow 1$	$2 \rightarrow 2$	$2 \rightarrow 3$
$3 \rightarrow 1$	$3 \rightarrow 2$	$3 \rightarrow 3$

Di solito la matrice W è caratterizzata solo dai blocchi al di sopra della diagonale principale, del tipo $W_{i,i+1}$, mentre gli altri sono tutti a 0.

Un altro approccio di tipo astratto è quello proposto nell'ottimo lavoro del gruppo PDP ([RuMcCle91]) al quale si rimanda. Un aspetto essenziale dei modelli proposti dal gruppo PDP è che i vari PE rappresentano oggetti o concetti astratti e sono visti come elementi di modelli la cui caratterizzazione in termini di matrice di connettività[3], caratteristiche dei nodi e leggi di apprendimento permette di adattare il particolare modello alle varie classi di problemi.

Le topologie più semplici, ottenibili anche dalla caratterizzazione dei suddetti modelli, sono, in genere, quelle delle reti neurali usate per implementare memorie associative, sia eteroassociative sia autoassociative ([TA94]).

Una memoria autoassociativa è progettata ed addestrata per trasformare un pattern in ingresso in se stesso in modo che, in presenza di una versione degradata o incompleta di un pattern di ingresso, sia possibile ottenere il pattern originale. Le memorie eteroassociative, d'altra parte, realizzano mapping fra insiemi di pattern distinti.

In modo più formale, si ha che, nel caso di una memoria autoassociativa, in fase di addestramento si associano i pattern P_k a se stessi in modo che, in fase di produzione, sia possibile ricavare P_k anche in presenza di P_k^* tale che $d_H(P_k, P_k^*) > 0$, in cui d_H è la distanza di Hamming fra gli stimoli[4]. Nel caso di una memoria eteroassociativa, in fase di addestramento si associano i pattern I_k ai pattern O_k in modo che, in fase di produzione, sia possibile ricavare O_k anche in presenza di I_k^* tale che $d_H(I_k, I_k^*) > 0$.

Tali reti memorizzano i pattern in modo distribuito sulle connessioni fra i PE per cui ogni singolo PE interviene nella memorizzazione di molti dei pattern che la rete globalmente

---

2 In letteratura la notazione non è uniforme ed è possibile trovare usata la convenzione opposta per cui la matrice dei pesi coincide con la trasposta di quella da noi definita.

3 Il modello di connettività, concretizzato dalla matrice di connettività, rappresenta, mediante i valori correnti dei pesi, ciò che il sistema conosce e permette di determinare la sua risposta a determinati pattern di ingresso.

4 La distanza di Hamming misura il numero di componenti diverse in due pattern.

memorizza: il numero di pattern memorizzabili varia in genere proporzionalmente alle dimensioni di una rete e, parallelamente diminuisce il numero di pattern cui il singolo PE è sensibile.

Le memorie eteroassociative che verranno esaminate brevemente nel seguito sono pensate associare agli elementi della matrice X, le cui righe sono i pattern di ingresso $X_1..X_N$ e le cui colonne sono tante quanti i PE dello strato di ingresso, la matrice Y, le cui righe sono i pattern di uscita $Y_1..Y_N$ e le cui colonne sono tante quanti i PE dello strato di uscita, mediante la matrice dei pesi W[5], associata alle connessioni fra i due strati. Se la memoria è di tipo autoassociativo i vettori X e Y coincidono. La matrice W viene calcolata di solito con tecniche numeriche ed in modo che sia Y = XW. La valutazione della W comporta in genere la necessità di calcolare la pseudoinversa della X[6], per maggiori dettagli si rimanda a [Ta94].

Un modello classico di memoria eteroassociativa è dato dal modello di Kohonen. In questo modello sono note le matrici X e Y e il calcolo della matrice W lo si fa minimizzando l'errore E definito come E = Y - XW. In pratica, la quantità da minimizzare è la radice quadrata della traccia euclidea della matrice Q definita come $Q = EE^T$, dove la traccia euclidea è data dalla somma dei quadrati degli elementi presenti sulla diagonale principale della matrice Q. In tal modo si ottiene un sistema di equazioni alle derivate parziali che si separano a coppie e la cui soluzione permette di determinare gli elementi incogniti della matrice W.

Nota W si ha che X_i richiama Y_i . Il metodo suddetto ha validità generale ma mostra la sua utilità nel caso di matrici rettangolari, dato che se le matrici X e Y sono quadrate, la matrice W può essere facilmente calcolata come $X^{-1}Y$.

Un altro modello classico di memoria eteroassociativa è dato dalle Bidirectional Associative Memories o BAM ([Ta94] e [FreSka91]).

Una Bam è una rete a due strati connessi fra di loro nel modo descritto dalla matrice dei pesi W. Punto di partenza sono due matrici A e B composte dai pattern rispettivamente di input $A_1..A_N$ e di output $B_1..B_N$ a valori nell'insieme {0, 1}. Da tali matrici, mediante opportune trasformazioni, si ottengono le matrici X e Y a valori nell'insieme{-1, 1} composte dai vettori X_k e Y_k. La matrice W è calcolata come sommatoria delle matrici $W_k = X_k^T Y_k$.

La presentazione di uno degli A_k da luogo in uscita alla comparsa del vettore corrispondente B_k (ovvero ad un vettore B_k^* a valori in $\mathbb{Z}$ mappato mediante una trasformazione opportuna su B_k). A_k richiama B_k mediante W e, in modo analogo, B_k richiama A_k mediante W^T. Se i pattern di ingresso e di uscita coincidono la memoria è autoassociativa. Dato un ingresso spurio $A_i \notin \{A_k\}$ la rete implementa un algoritmo in base al quale A_i richiama un pattern B_i, che può coincidere o meno con uno dei B_k. Nel primo caso mediante W^T si ha il richiamo di A_k per cui la rete si stabilizza sulla coppia A_k, B_k, nell'altro caso B_i viene rielaborato mediante W^T in modo da produrre (fatta salva l'applicazione di mapping ad hoc) un nuovo A_i sul quale il ciclo viene ripetuto fino a che la rete cade nella situazione precedente oppure viene superato un numero massimo di iterazioni, nel qual caso il pattern di ingresso non è stato riconosciuto dalla rete. L'obiettivo è quello di far cadere un pattern A_i nel cosiddetto bacino di attrazione di uno degli A_k in modo che la rete gli associ il corrispondente B_k.

Un modello interessante di memoria autoassociativa è dato dal modello sviluppato da Hopfield ([Ta94]). I principali contributi di Hopfield nell'ambito del Neural Computing riguardano la descrizione delle reti neurali in termini di una funzione energia, la dimostrazione del fatto che le reti neurali sono isomorfe ai vetri di spin e la dimostrazione della convergenza di una rete in stati stabili di energia.

Nel modello di Hopfield, ogni PE esegue una somma pesata degli ingressi e applica su tale somma una funzione a soglia in modo da produrre un'uscita binaria che viene rinviata in ingresso a tutti gli altri PE. Sui pesi, infatti, vigono i seguenti vincoli: $w_{i,i} = 0$ e $w_{i,j} = w_{j,i}$.

La rete ha un buffer di ingresso, uno di uscita e uno strato detto di Hopfield. Gli elementi dello strato di ingresso sono collegati ciascuno con un solo elemento dello strato di Hopfield che è collegato a sua volta con un solo elemento del buffer di uscita. L'uscita del PE j-esimo dello strato di Hopfield, inoltre, è riportata in ingresso a tutti gli altri PE dello strato di Hopfield mediante dei pesi $w_{i,j}$, secondo la notazione usata da Hopfied. La rete, pertanto, vista come definita dalla matrice W ha un solo strato di PE connessi fra di loro.

---

5 A rigore la matrice W di cui si parla coincide con il blocco $W_{1,2}$ della tabella precedentemente illustrata.

6 La pseudoinversa coincide con l'inversa nel caso di matrici quadrate, altrimenti può essere ottenuta combinando in modo opportuno la matrice data con la sua trasposta ([Ta94]).

In fase di addestramento, i pesi sono rinforzati (resi più eccitatori) se l'output di un PE coincide con l'input mentre sono indeboliti (resi più inibitori) in caso inverso. Una rete di Hopfield a fine apprendimento presenta stati stabili spuri dovuti alla presenza incidentale di fattori comuni a pattern distinti che danno luogo alla comparsa di stati compositi che non hanno rilevanza ai fini dell'apprendimento. Alcuni di tali stati spuri possono essere rimossi con delle vere e proprie tecniche di rimozione.

Durante la fase di produzione un vettore viene presentato sul buffer di ingresso che lo trasferisce allo strato di Hopfield dando inizio ad una sua ricircolazione attraverso la matrice W fino a che un qualche criterio di convergenza è soddisfatto oppure per un numero massimo di cicli prefissato. Alla fine delle iterazioni, il risultato è accessibile sul buffer di uscita. In termini più formali, si ha X(i+1) = u(WX(i)), dove u(t) è la unzione gradino unitario applicata ai singoli elementi del vettore X.

Lo stato della rete può essere descritto anche da una funzione energia del tipo

$$E = -\frac{1}{2}\sum_i \sum_j w_{i,j}x_i x_j$$

con

$$E_i = -\frac{1}{2}\sum_j w_{i,j}x_i x_j$$

porzione di E influenzata dallo stato x_i del PE i-esimo. È possibile dimostrare[7] che la variazione di energia ΔE_i è proporzionale a Δx_i ed è sempre negativa per cui l'energia decresce fino a che la rete non si posiziona in stati stabili di minimo locale dell'energia: si può affermare che un PE modifica il proprio stato se e solo se la modifica contribuisce al decremento dell'energia globale della rete.

Nel caso delle tipologie di rete viste finora come pure nelle reti multistrato per le quali è stato concepito l'algoritmo di backpropagation, la topologia della rete è fissata a priori e allo stesso modo è fissato a priori il numero di PE di ciascuno strato: per gli strati di ingresso e di uscita tali quantità dipendono fortemente dalla classe di problemi che si vuole risolvere mentre il numero di unità negli strati nascosti viene fissato di solito per tentativi o sulla base di analogie con reti sviluppate per problemi simili. Allo stesso modo sono fissate a priori le funzioni caratteristiche dei singoli PE.

Un altro approccio prevede la possibilità di definire reti neurali la cui topologia cresce e si espande (come numero e tipo dei PE e di connessioni fra PE) in funzione delle esigenze della classe di problemi da risolvere.

Un primo modello che viene succintamente esaminato è quello proposto da Fahlman e Lebrere ([FaLe]) e detto "Cascade Correlation Learning Architecture".

Punto di partenza del modello è la definizione di una rete neurale minima cui vengono aggiunte man mano nuove unità nascoste, addestrate singolarmente, in modo da ottenere una struttura multi-layer, addestrabile senza far ricorso alla backpropagation.

Le unità nascoste sono aggiunte singolarmente, funzionano da rilevatori di feature e sono disponibili per la produzione di pattern di uscita o come rilevatori di feature più complessi.

L'aggiunta di una unità nascosta, che riceve segnali da tutte le unità di input della rete originale e da tutte le unità nascoste aggiunte in precedenza, prevede un addestramento separato di tale unità in modo che i pesi sulle sue connessioni di ingresso siano congelati una volta per tutte. Obiettivo dell'addestramento della singola unità nascosta è quello di massimizzare la covarianza fra l'uscita di tale unità e l'errore residuo in uscita, per un generico pattern di addestramento.

Una volta che l'unità nascosta è stata addestrata viene addestrata la rete nel suo complesso e i pesi soggetti a modifica sono quelli fra lo strato di ingresso e quello di uscita e fra quest'ultimo e le uscite delle unità nascoste. Se le unità nascoste sono in numeo sufficiente, la rete risolverà il problema per il quale viene progettata con l'errore atteso, altrimenti nuove unità nascoste dovranno essere aggiunte.

I vantaggi di tale topologia sono la velocità di apprendimento e il fatto che la rete sia in grado di autodeterminare la propria dimensione e la propria topologia e che l'apprendimento

---

7 Ad esempio, se x_i passa da 0 a 1 la sua variazione positiva è resa possibile dal fatto che la somma pesata in ingresso è a sua volta positiva per cui la ΔE_i è negativa e la somma di contributi negativi non può che essere negativa, lo stesso vale per la transizione da 1 a 0.

sia di tipo incrementale, cioè nuovi dati possono essere memorizzati in una rete già addestrata senza perdita di informazioni.

Il tempo di addestramento necessario, in numero di epoche[8], è proporzionale a NlogN, dove N è il numero di unità nascoste necessarie per risolvere una certa classe di problemi.

L'altro modello di rete neurale ad accrescimento che viene esaminato in queste note è quello sviluppato da Frean ([Fre89]) che va sotto il nome di "Upstart Algorithm".

Il metodo proposto da Frean permette di costruire ed addestrare reti a più strati caratterizzati da PE lineari a soglia. La regola su cui si basa la costruzione è una semplice regola ricorsiva che permette di costruire la struttura della rete neurale aggiungendo delle unità man mano che queste si rivelano necessarie. L'algoritmo di addestramento usato è stato derivato da quello per il Perceptrone, non trattato nelle presenti note. L'idea di base è quella di creare, per ciascuna unità che compie errori di classificazione, due unità figlie che operano le necessarie correzioni e di inserire tali unità fra l'unità che "sbaglia" e lo strato di unità di ingresso. In breve, l'algoritmo funziona come segue: data una unità Z (a soglia) questa può classificare bene oppure può compiere errori e cioè può essere ad 1 quando deve essere a 0 o viceversa. Per ovviare al problema l'idea è quella di aggiungere due unità figlie X e Y, attive solo per i pattern per i quali Z commette errori, ciascuna delle quali interviene in una delle due suddette situazioni in modo da inibire o eccitare fortemente l'unità Z impedendogli di sbagliare. Il ragionamento svolto per l'unità Z può essere svolto ricorsivamente per le unità X e Y in modo da arrivare a definire una rete "error free".

Lo scopo dichiarato è quello di sfruttare le potenzialità delle reti di tipo Perceptrone in modo da addestrarle con un algoritmo alternativo alla backpropagation i cui difetti principali, secondo Frean, sono legati alla presenza di minimi locali della funzione energia cui possono essere associati stati non significativi e il fatto che spesso non è chiaro il modo per stimare a priori il numero necessario di unità negli strati nascosti.

Altri algoritmi per la definizione di reti a topologia "incrementale" sono l'algoritmo tower e l'algoritmo tiling ([Sta94]).

L'algoritmo tower permette di costruire reti a struttura gerarchica di tipo Perceptrone con uno strato di ingresso o livello 0, uno strato di livello 1 composto da una cella completamente connessa con lo strato di ingresso, uno strato i-esimo (i = 2, ..N) di una cella completamente connessa con lo strato di ingresso e la cella dello strato (i-1)-esimo.

La struttura di partenza è data dallo strato di livello 0 e dalla cella dello strato di livello 1 (Perceptrone semplice). Tale rete viene addestrata su un set di pattern di ingresso e può classificarli bene o commettere errori: nel primo caso l'algoritmo termina, nell'altro viene aggiunta una cella al livello 2 che viene di nuovo addestrata. Il procedimento viene iterato fino a che la rete risultante non classifica i pattern di ingresso senza commettere errori, oppure fino a che l'errore percentuale non scende al di sotto di un valore prefissato.

L'addestramento riguarda solo la cella aggiunta al livello i-esimo e cioè i pesi sulle connessioni fra tale cella e lo strato di ingresso e fra tale cella e quella di livello i-1, per cui si può applicare la delta rule semplice (vedi oltre) oppure un algoritmo di Pocket[9].

L'algoritmo tiling, infine, permette di costruire reti aggiungendo nuovi strati oppure nuove unità all'interno di uno strato in modo da consentire la definizione di reti multilayer con un numero di strati nascosti e unità nascoste variabile. Il numero L degli strati è finito e questo è una garanzia per la convergenza degli algoritmi di apprendimento.

Durante la costruzione ognuno degli strati, le cui unità vedono solo quelle dello strato precedente, ha una unità master che fornisce una mappatura dei pattern in ingresso su quelli in uscita migliore di quella fornita dalla unità master dello strato precedente.

Punto di partenza dell'algoritmo di costruzione è una rete con due strati, uno di ingresso con N unità connesse alla sola unità master sullo strato di uscita. L'unità master viene addestrata (ad esempio con l'algoritmo Pocket) su un insieme di esempi e può accadere che li classifichi correttamente oppure no. Nel primo caso l'algoritmo termina mentre nell'altro caso alla rete è aggiunto un nuovo strato con una unità master mentre nel vecchio strato di uscita si aggiungono alcune unità dette ancelle. Il compito delle ancelle è quello di classificare correttamente il maggior numero possibile di esempi su cui l'unità master dello stesso strato

---

8 Si ricorda che un'epoca è, per definizione, una presentazione di tutti i pattern di ingresso alla rete durante il suo addestramento.

9 L'algoritmo Pocket è una variante dell'algoritmo del Perceptrone e permette di trovare un vettore di pesi ottimale W (nel senso che classifica correttamente il maggior numero possibile di esempi, data la topologia della rete) conservando, ad ogni passo, una copia del vettore W e aggiornandola solo se il nuovo vettore, calcolato dall'algoritmo, classifica in modo corretto un maggior numero di pattern.

commetteva errori: tali celle sono addestrate singolarmente su sottoinsiemi di pattern di addestramento.

L'algoritmo viene iterato fino a che la rete ottenuta non è in grado di classificare correttamente i pattern di addestramento.

Metodologie di addestramento.

L'addestramento di una rete neurale ([Sta94] e [RuMcCle91]) mira a far si che questa apprenda a svolgere un determinato compito con le migliori prestazioni possibili, data la topologia e il metodo di addestramento scelti. Il criterio di valutazione è spesso un criterio di ottimo che tende a minimizzare una funzione errore.

L'apprendimento da parte della rete si traduce nella modifica del valore dei pesi sulle connessioni in funzione dell'esperienza e in accordo a un ben preciso modello di apprendimento. La modifica dei pesi riesce a simulare sia la comparsa di nuove connessioni (un peso nullo assume un valore diverso da 0, positivo o negativo a seconda della natura della connessione) sia la scomparsa di connessioni esistenti (ad un peso viene assegnato un valore nullo) sebbene siano attualmente allo studio topologie ad accrescimento in cui, durante la fase di addestramento, la rete cresce per aggiunta di nuovi nodi e delle relative connessioni. Volendo agire solamente sui valori dei pesi è ovvio che la rete di partenza deve essere completamente connessa.

Scopo del modello di apprendimento è quello di far si che la rete apprenda a far corrispondere a un pattern di ingresso un pattern in uscita oppure ad un insieme di pattern di ingresso un insieme di pattern di uscita, insieme che può essere composto anche da un solo elemento (classificatori).

L' apprendimento è in genere di due tipi, supervisionato e non, ed entrambi sono caratterizzati da una pluralità di tecniche e teorie che, in questa sede, ci è possibile esaminare solo molto superficialmente.

L'apprendimento può essere visto come un autoadattamento a livello dei singoli PE in base al quale le connessioni pesate fra PE subiscono modifiche. Le modifiche permettono il conseguimento di determinati risultati senza che sia necessario scrivere un programma particolareggiato per ogni singolo problema.

Nel caso dell'apprendimento supervisionato, la rete è addestrata da un teacher esterno che, per ogni pattern di ingresso, mostra alla rete il pattern di uscita desiderato oppure segnala la correttezza o meno della risposta (a rigore si parla di apprendimento per rafforzamento).

Nel caso dell'apprendimento non supervisionato non si ha alcun teacher esterno e in genere non sono disponibili le risposte corrette agli insiemi di pattern di ingresso. La rete si deve autoorganizzare solo sulla base dei pattern di ingresso che riceve.

In entrambi i casi, i parametri in base ai quali avviene l'apprendimento possono variare in base a leggi ben precise.

Esempi di apprendimento non supervisionato sono la Adaptive Resonance Theory (ART) e le tecniche di apprendimento competitivo.

La ART è una teoria che sfrutta la capacità di autoorganizzazione di una rete in modo che questa sia utilizzabile per compiti di pattern recognition in assenza di teacher esterno.

Nel caso dell'apprendimento competitivo i vari PE competono fra di loro per rispondere ad un dato pattern di ingresso in modo che il vincitore sia l'unico a rispondere a tale pattern.

Esempi di apprendimento supervisionato sono l'apprendimento hebbiano, la delta-rule e la backpropagation.

L'apprendimento hebbiano ha molte varianti riconducibili alla regola in base alla quale un cammino neurale viene rafforzato ogni volta che viene usato.

La delta-rule rappresenta una modalità di apprendimento nella quale i pesi sulle connessioni sono modificati in modo da minimizzare la differenza fra l'uscita voluta e quella corrente dei PE dello strato di output[10].

Un esempio di generalizzazione della delta-rule è la backpropagation che permette di addestrare una rete multi-layer mediante una applicazione ricorsiva della delta-rule a partire dalle connessioni fra lo strato di output e lo strato nascosto immediatamente adiacente fino a quelle fra il primo strato nascosto e lo strato di ingresso della rete. Obiettivo della strategia di

10 Nel testo sono ovvie le corrispondenze fra termini inglesi quali input, output, hidden e altri e quelli italiani ingresso, uscita e nascosto e altri.

apprendimento è quello di addestrare una rete a svolgere compiti di pattern recognition e classificazione sotto la guida di un teacher esterno.

Date due unità i e j connesse fra di loro, si può affermare, come regola generale, che l'apprendimento, se coinvolge le due unità, si traduce in una modifica del peso della connessione fra le due unità, indicata con $\Delta W_{i,j}$, per cui la legge di variazione del peso sulla connessione dal PE i-esimo al PE j-esimo ha la seguente espressione:

$$W_{i,j}(t+1) = W_{i,j}(t) + \Delta W_{i,j}(t)$$

La quantità $\Delta W_{i,j}(t)$ è in genere espressa come prodotto di due funzioni:

$$\Delta W_{i,j}(t) = g(a_j(t), t_j(t))h(o_i(t), W_{i,j})$$

in cui $t_j(t)$ è la funzione teacher per l'unità j-esima, $a_j(t)$ il suo stato di attivazione e $o_i(t)$ lo stato di uscita dell'unità i-esima. Indicata con = una grandezza detta velocità di apprendimento a valori nell'intervallo [0, 1], la versione più semplice della suddetta regola è la seguente:

$$\Delta W_{i,j} = = a_j\, o_i$$

mentre una versione più versatile detta delta rule, la si ottiene con le seguenti scelte:

$h(o_i(t), W_{i,j}) = o_i$
$g(a_j(t), t_j(t)) = = (t_j - a_j)$

che danno luogo alla seguente relazione

$$\Delta W_{i,j} = = (t_j - a_j)\, o_i$$

Gli algoritmi di apprendimento caratterizzano le reti come sistemi che apprendono. Un sistema che apprende è, in generale, un sistema in grado di prendere decisioni sulla base di esperienze fatte su casi risolti con successo: un sistema di tale tipo estrae criteri di decisione da campioni di casi che gli vengono presentati e dai quali deve essere in grado di derivare delle regole che poi applicherà ai dati nuovi. In questi casi il compito principale è la classificazione, anche con sistemi ibridi che fanno uso di alberi di decisione ([SiNa90]).

I classificatori possono essere di tipo lineare (della famiglia del Perceptrone o sue varianti) oppure di tipo adattivo o competitivo.

Un esempio di classificatore del secondo tipo è dato dall'algoritmo di Carpenter & Grossberg ([Sta94]) detto algoritmo del leader. Tale algoritmo è implementato da una rete a due strati, X e Y, completamente connessi fra loro con connessioni pesate da X a Y, da Y a X e fra gli elementi di Y, per cui ognuno degli elementi ha una visione della computazione globale.

Tale rete, partendo da un insieme di pattern di ingresso, li suddivide in cluster senza l'ausilio di un teacher esterno.

Per ciascun pattern di ingresso l'algoritmo lo associa al cluster il cui leader dista dal pattern corrente per meno di un certo valore di soglia oppure, se non trova nessun leader che soddisfa tale requisito o in assenza di cluster, definisce tale pattern come leader di un nuovo cluster. In questo caso il numero di cluster tende a crescere con il tempo e dipende sia dal valore della soglia sia dal tipo di metrica usata per definire la distanza (ad esempio distanza di hamming o distanza Euclidea). Una volta definiti i cluster, un pattern è sempre associabile ad uno di essi oppure ad uno nuovo, di cui costituisce il leader, sfruttando, a livello di architettura di rete, l'inibizione laterale. Una cautela da tenere, in questo come in altri casi simili, è di evitare che la codifica dei dati in ingresso suddivida artificiosamente i dati in cluster.

L'algoritmo detto del leader è un classico esempio di apprendimento non supervisionato nel quale sono noti i pattern da usare nell'addestramento ma non i patten corrispondenti per cui l'unica soluzione possibile è quella di raggruppare i dati in cluster, cioè in insiemi caratterizzati da una qualche similitudine, tipo la distanza fra pattern. Vediamo ora molto brevemente altri algoritmi di clustering prima di passare all'apprendimento competitivo.

Il primo algoritmo di clustering che viene esaminato è detto di clustering di k-medie. Tale algoritmo si basa sul concetto di centroide, definito[11] come il vettore media di N vettori E^k, ed è caratterizzato dai passi seguenti.

1) Si partizionano i dati di addestramento E^i (i = 1...N) in k cluster in base al seguente procedimento:

- si prendono i primi k elementi per definire k cluster ciascuno con un solo elemento;
- si valutano i k centroidi e si assegna ciascuno degli N - k elementi rimanenti al cluster il cui centriode è più vicino all'elemento in questione. Dopo ogni assegnamento si ricalcola il centroide del cluster a cui si è aggiunto l'elemento.

Scopo di questo passo è quello di ottenere k cluster, ciascuno con un centroide C^j, in cui sono stati partizionati gli N elementi E^i. I centroidi fungono da centri di attrazione per gli elementi E^i .

2) A questo punto per ciascun E^i calcolo la distanza da ogni centroide C^j. Se E^i appartiene già al cluster caratterizzato dal centroide avente distanza minima da E^i non faccio nulla altrimenti sposto E^i da un cluster all'altro aggiornando i due centroidi. Il secondo passo viene ripetuto fino a che nessuno degli E^i deve essere spostato da un cluster ad un altro.

Il punto critico del presente algoritmo è rappresentato dalla necessità di definire a priori un valore per k con l'unico vincolo che sia più piccolo di N. Ovviamente tanto minore è k tanto più grossolana è la clusterizzazione dei dati. Algoritmi derivati da quello di clustering di k-medie e basati sulla Adaptive Resonance Theory sono ART1 e ART2. La differenza principale fra i due è che il secondo lavora anche su esempi E^i caratterizzati da componenti reali mentre ART1 classifica, raggruppandoli, esempi E^i i cui elementi assumono valori nell'insieme {0, 1}.

Scopo di ART1 è la clusterizzazione dei dati ottenuta, in questo caso, mediante la definizione di prototipi che combinano le caratteristiche di più esempi e che evitano la definizione a priori del numero di cluster. Dato un esempio si hanno due casi: lo si assegna ad un cluster esistente, se esiste un prototipo cui l'esempio è sufficientemente simile[12], oppure lo si assegna ad un cluster nuovo di cui l'elemento costituirà il prototipo. La seconda eventualità si verifica, ovviamente, in assenza di prototipi. L'assegnazione comporta una nuova valutazione del prototipo del cluster cui l'elemento viene assegnato per cui, in genere, si traduce in una modifica del prototipo: l'algoritmo stesso viene pertanto eseguito fino a che l'esame degli esempi provoca modifiche nell'insieme dei vettori prototipo. Durante la fase di produzione la rete è in grado di associare un pattern ad un preciso cluster mediante un confronto con i vari prototipi.

Per quanto riguarda l'apprendimento competitivo, questo viene realizzato mediante strutture gerarchiche caratterizzate da più strati di unità in cui ogni strato è connesso, mediante connessioni di tipo eccitatorio, con il successivo. Le unità sono fra loro completamente connesse per cui ogni unità vede tutte quelle dello strato precedente ed è vista da tutte quelle del successivo. La rete ha uno strato di ingresso o primo strato con unità attive o inattive in funzione del pattern corrente. Gli altri strati sono caratterizzati da cluster di unità mutuamente inibitorie in modo che solo una unità per cluster può essere attiva: ogni cluster ha una sola unità il cui stato vale 1 (e che apprende, vedi oltre) mentre lo stato delle altre vale 0. L'insieme degli stati delle unità rappresenta il pattern di ingresso allo strato successivo.

L'apprendimento si basa sul fatto che ogni unità ha un peso costante (ad esempio 1) distribuito sulle linee in ingresso e si traduce, per le unità attive per un dato pattern, nello spostamento del peso dalle linee in ingresso inattive a quelle attive in modo che la somma totale dei pesi rimanga costante. Se con $w_{i,j}$ si indica il peso della connessione dal PE i-esimo al PE j-esimo e il PE j-esimo vince all'interno del suo cluster per quel pattern allora si ha che la variazione dei pesi è governata dalla seguente legge:

---

11 La definizione formale è la seguente. $C = \frac{1}{N_k}\sum E^k$ in cui C indica il centroide di componenti $c_i = \frac{1}{N_k}\sum E_i^k$

12 Il grado di somiglianza può essere dato dal numero di 1 a comune.

$$\Delta w_{i,j} = g\frac{c_{i,k}}{n_k} - gw_{i,j}$$

in cui g rappresenta il guadagno, k individua il pattern k-esimo, $c_{i,k}$ vale 1 se l'unità i-esima dello strato inferiore è attiva e 0 altrimenti e n_k è il numero totale di unità attive per il pattern k-esimo sullo strato inferiore.

Se i pattern E^k sono visti come vettori di pari lunghezza (e ciò è possibile se hanno grosso modo lo stesso numero di componenti attive) e cioè come punti su una ipersfera a N dimensioni e se la stessa operazione viene fatta per i vettori dei pesi (eventualmente scalandoli, considerando che la somma delle loro componenti è costante) si ha che per ciascun pattern l'unità che vince è quella il cui vettore dei pesi è più prossimo al vettore rappresentativo del pattern corrente. L'apprendimento è semplicemente il processo di spostamento del vettore peso della unità risultata vincente in direzione del vettore del pattern E^k. Dati M pattern che sono "naturalmente" raggruppabili in ingresso a cluster di M unità, a fine addestramento uno dei vettori peso delle M unità si trova al centro di ciascun raggruppamento per cui è in grado di riconoscere i pattern che ad esso appartengono, per maggiori dettagli si rimanda a [Sta94].

Un altro esempio di sistema ad apprendimento competitivo è rappresentato dalle SOM o Self Organizing Map sviluppate da Kohonen ([Sta94]).

Le SOM traggono ispirazione dalla struttura della corteccia celebrale e conservano l'organizzazione spaziale dell'informazione, come si pensa succeda nella corteccia dove si suppone esista una corrispondenza fra il luogo in cui l'informazione è memorizzata e le caratteristiche possedute dall'informazione stessa.

Una SOM è caratterizzata da due strati completamente connessi fra loro: uno di ingresso con p unità e uno di uscita strutturato come una matrice di NxN unità. Lo strato di uscita è visto come una mappa, ossia una struttura organizzata spazialmente, per cui é possibile parlare di una unità e di un suo intorno, definito in base ad una distanza di Hamming. Lo scopo è quello di far si che una unità si comporti in modo consistente con quello delle unità del suo intorno: dato un pattern E^k si vuole che almeno una unità dello strato di uscita abbia il vettore dei pesi W prossimo, come distanza Euclidea, a E^k e che le unità del suo intorno si comportino in modo analogo in modo da preservare la topologia. L'algoritmo di modifica dei pesi è iterativo e, ad ogni iterazione, il passo di modifica dei pesi si riduce, tendendo a zero al crescere del numero di iterazione.

Peculiarità del Neural Computing.

Il Neural Computing rappresenta un approccio alternativo e concorrente agli approcci dell'Intelligenza Artificiale e del Calcolo tradizionale. In questa sede non ci è possibile entrare nel merito delle polemiche fra Connessionisti e Cognitivisti per cui ci si limita ad elencare i tratti salienti del Neural Computing, tratti che lo rendono un approccio interessante e promettente per la soluzione di molti problemi che, ad esempio, l'Intelligenza Artificiale non riesce ancora ad affrontare con la sperata incisività.

Nel caso del Neural Computing, la conoscenza non è esplicita sotto forma di regole come accade ad esempio nel caso dei Sistemi Esperti ma le reti generano le proprie "regole" mediante un'apprendimento realizzato durante la fase di addestramento ossia mediante una strategia di modifica dei pesi sulle connessioni in risposta a pattern presentati sul buffer di ingresso.

L'apprendimento può essere non supervisionato oppure supervisionato.

Nel primo caso alla rete vengono presentati solamente pattern di input e la rete si auto organizza internamente in modo che ogni PE degli strati nascosti risponda soltanto a gruppi di pattern di stimoli che rappresentano cluster nello spazio degli ingressi, cluster cui sono associati concetti distinti del mondo reale.

Nel secondo caso, alla rete vengono presentate coppie correlate di pattern in ingresso ed in uscita e la rete si auto organizza in modo che l'output prodotto coincida con quello presentato e corretto per l'input corrente.

Le regole utilizzate per l'apprendimento in base alle quali vengono modificati i pesi sulle connessioni sinaptiche sono l'apprendimento hebbiano, basato sulla legge di Hebb, la regola delta semplice e generalizzata e l'addestramento competitivo con o senza inibizione laterale.

L'apprendimento è in ogni caso un processo che richiede tempo per essere completato e che può beneficiare dell'utilizzo di architetture specializzate come i neurocomputer (vedi oltre).

Le reti neurali hanno, inoltre, la caratteristica di essere memorie distribuite e associative. I pesi sulle connessioni rappresentano le unità di memoria e il loro valore rappresenta lo stato corrente della conoscenza della rete. Le coppie di pattern oppure i cluster sono memorizzati in modo distribuito sulle unità di memoria e ciascuna coppia o elemento di un cluster condivide le unità di memoria con tutte le altre coppie o tutti gli atri elementi memorizzati dalla rete.

L'associatività, cioè la possibilità di accedere per contenuto e non per indirizzo, si traduce nella capacità delle reti neurali di riconoscere pattern "rumorosi" in ingresso associando loro il pattern in uscita corretto[13] o il pattern più prossimo (memorie eteroassociative) oppure producendo lo stesso pattern "filtrato" o completato (memorie autoassociative). La capacità di lavorare su pattern di ingresso sconosciuti e/o disturbati può essere vista come una capacità delle reti di generalizzare.

Con il termine generalizzazione si definisce, infatti, la capacità di una rete di derivare da coppie pattern-in-ingresso/pattern-in-uscita, usate in fase di addestramento un insieme di caratteristiche astratte che permettono alla rete di produrre un output significativo anche in presenza di pattern nuovi, incompleti o rumorosi. Considerazioni analoghe valgono anche nel caso di reti con addestramento non supervisionato, dove la capacità di generalizzazione è legata alla definizione di cluster a partire dai dati in ingresso. Ovviamente la qualità e significatività della capacità di generalizzazione dipende sia dalla applicazione sia dal grado di sofisticazione della rete. Le reti neurali possono generalizzare mediante interpolazioni, mappando un pattern di input su un pattern di output in modo da ricreare una coppia input/output appresa oppure (caso di reti multistrato con backpropagation) apprendendo negli strati nascosti un insieme di caratteristiche importanti del dominio in esame e usando tale conoscenza "nascosta" per eseguire generalizzazioni non banali.

Altre caratteristiche interessanti delle reti neurali sono il loro essere fault tolerant e la loro versatilità nel pattern recognition.

I sistemi per il Neural Computing sono caratterizzati da una elevata fault tolerance grazie alla loro natura intrinsecamente distribuita: tali sistemi, nella maggior parte dei casi, sono in grado di funzionare con prestazioni più o meno degradate, anche nel caso che alcuni PE siano guasti o disattivati oppure i pesi sulle loro connessioni siano stati modificati in modi non previsti dalla strategia di addestramento utilizzata. Il fatto che la conoscenza sia distribuita nella rete fa si che le prestazioni della rete peggiorino e il suo comportamento degradi in presenza di malfunzionamenti di entità crescente sebbene la rete continui a mostrare funzionalità residue.

Oltre ad essere fault tolerant, i sistemi per il Neural Computing sono portati a compiti di pattern recognition. Il pattern recognition richiede che un sistema ad esso dedicato abbia la capacità di eseguire operazioni di matching su grosse quantità di dati in ingresso per poi produrre o una generalizzazione o una categorizzazione su tali dati. I sistemi per il Neural Computing sono adatti a tali compiti dato che sono in grado di apprendere e costruire strutture univoche per un dato problema e sono composte da unità interconnesse che lavorano su tutti i dati contemporaneamente e con elaborazioni ad elevato parallelismo.

Cenni ai principali campi di applicazione del Neural Computing.

Il Neural Computing può essere utilizzato proficuamente in molti settori applicativi e può fornire un modello alternativo a quelli proposti dal Cognitivismo per la mente e il cervello e le relazioni mente-cervello. Da un punto di vista applicativo, il Neural Computing fornisce un approccio più vicino dei sistemi di calcolo tradizionali alle capacità percettive e di riconoscimento degli esseri umani.

---

13 Nel caso dell'apprendimento supervisionato, la rete memorizza corrispondenze fra un insieme di pattern di ingresso con i pattern corrispondenti in uscita per cui, durante la fase di produzione, in presenza di un pattern di ingresso appreso ricava il pattern corrispondente. Se il pattern di ingresso è degradato la rete lo associa a quello dei pattern appresi che gli è più vicino (ad esempio in base alla distanza di hamming) in modo da produrre l'output corrispondente. Il procedimento può richiedere una elaborazione ciclica in base alla quale un input produce un output che viene elaborato come input dalla rete percorsa in senso inverso e così via fino a che la rete non trova un matching. Tale procedimento può essere visualizzato come un cammino di avvicinamento a un minimo significativo della funzione energia associata alla rete.

Nei paragrafi seguenti vengono esaminati brevemente alcuni settori applicativi che hanno beneficiato dell'uso di reti neurali, con particolare attenzione alla Linguistica. I settori applicativi presi in esame non sono appannaggio esclusivo del Neural Computing e pertanto esistono soluzioni sia con Sistemi Esperti o altri paradigmi di Intelligenza Artificiale sia con approcci software più o meno tradizionali alle problematiche esaminate , soluzioni la cui trattazione esula completamente dagli scopi del presente articolo.

Nell'ambito dell'elaborazione del linguaggio le reti neurali hanno trovato applicazione nella conversione da testo scritto a parlato (text to speech) e nella elaborazione del Linguaggio Naturale.

Nel primo caso, il sistema viene addestrato in modo da apprendere a tradurre un testo scritto in parlato secondo modalità simili a quelle di un bambino che sta imparando a leggere. Il primo passo consiste nella traduzione di un particolare testo scritto in fonemi, mediante un particolare codifica. Al sistema neurale (la rete in questione si chiama NETtalk) viene presentato il testo suddetto e il sistema ne opera la traduzione nei fonemi corrispondenti con l'ausilio di un sintetizzatore vocale (DECTalk) che traduce codici di fonemi in fonemi. All'inizio dell'addestramento i pesi sulle connessioni hanno valori casuali per cui la rete considera le parole come strutturate in un flusso continuo. Durante l'addestramento supervisionato, alla rete sono presentati i fonemi corretti mediante le opportune codifiche ed il sistema opera gli opportuni confronti, in modo da minimizzare gli errori mediante una correzione dei pesi. Dopo un certo numero di sessioni di addestramento, il sistema si è dimostrato capace di inferire regole di lettura del testo inglese, quali la differenza fra vocali e consonanti e fra le diverse modalità di produzione di un simbolo (ad esempio c dura e dolce e simili), e dopo un addestramento di una notte ha mostrato le stesse prestazioni di un bambino di sei anni. In questo caso l'uso di una rete neurale permette di non dover esplicitare complessi insiemi di regole di pronuncia dato che NETtalk è in grado di autoistruirsi derivando le regole necessarie alla sintesi del parlato dagli esempi che gli vengono presentati durante l'addestramento.

Un esempio di applicazione per l'elaborazione del Linguaggio Naturale è costituito dall'ormai classico lavoro di Rumelhart e Mc Clelland per l'utilizzo di reti neurali per la simulazione dell'apprendimento del passato dei verbi regolari e irregolari dell'Inglese. In questo caso gli autori hanno sfruttato la capacità di una rete neurale di operare generalizzazioni sulla base di dati incompleti e di autoorganizzarsi in modo da assolvere con buone prestazioni al compito in questione.

In questo caso si sfrutta abilmente la capacità delle reti neurali di operare delle generalizzazioni sulla base di dati incompleti partendo dai quali autoorganizzarsi per generare nuove forme verbali o congetturare la miglior soluzione in presenza di verbi nuovi o sconosciuti.

La possibilità di usare con successo di reti neurali in questo caso ed in altri analoghi rappresenta un argomento contro la critica rivolta a tali sistemi di lavorare solo in modo quantitativo e non qualitativo e a livelli subsimbolici, in pratica di non poter astrarre regole, capacità posseduta invece gli esseri umani.

Altri settori applicativi delle reti neurali comprendono la compressione di dati relativi ad esempio ad immagini e il riconoscimento di caratteri.

La compressione dei dati prevede l'uso di reti neurali per operare una traduzione dei dati da una rappresentazione ad un altra allo scopo di ridurre la banda di memoria necessaria preservando la qualità dell'immagine, ricostruibile mediante un procedimento inverso.

Per quanto riguarda il riconoscimento dei caratteri si tratta di un procedimento di interpretazione visiva e riconoscimento di simboli scritti. L'uso di reti neurali è giustificato dalla loro abilità a trattare simultaneamente grandi quantità di dati da cui estrarre certi pattern.

Una applicazione interessante si riferisce al riconoscimento di caratteri scritti a mano. In questo settore, ad esempio, la Nestor Inc ha sviluppato un sistema in grado di riconoscere caratteri immessi mediante una tavoletta digitalizzatrice. La fase di addestramento prevede che il sistema sia addestrato a riconoscere un ampio insieme di caratteri scritti a mano con stili diversi cui associa i caratteri corretti. Al termine di una fase di addestramento non necessariamente lunga, il sistema è in grado di riconoscere con una buona accuratezza caratteri scritti con calligrafie mai viste prima e di fornire buone approssimazioni in caso di caratteri "rumorosi", ossia in presenza di macchie o tratti spuri.

Una possibile architettura della rete utilizzabile per tale compito prevede tre strati completamente connessi fra loro: un buffer di input, con tante unità quanti sono gli elementi della matrice di input usata per la digitalizzazione dei caratteri, uno strato nascosto, con un

numero di unità che dipende dall'accuratezza richiesta alla rete, e uno strato di output, con tante unità quanti sono i caratteri distinti che la rete deve saper riconoscere.

Un altro sistema sviluppato per consentire il riconoscimento di caratteri scritti a mano è il Neocognitron. Il Neocognitron è un sistema multistrato per l'esecuzione di compiti di pattern recognition e di estrazione di feature (caratteristiche) in grado di simulare il flusso di informazioni nella corteccia del cervello umano. Il Neocognitron fa uso di strati successivi per il riconoscimento di pattern indipendentemente dal loro orientamento e da una eventuale loro distorsione in modo da ricondurli ad una delle possibili categorie mediante elaborazioni in cascata lungo gli strati.

Oltre che nei settori suddetti, i sistemi per il Neural Computing trovano applicazione in problemi combinatori (ad esempio il problema del commesso viaggiatore), nel riconoscimento di pattern in immagini (ad esempio nella classificazione di bersagli sonar), in verifiche industriali (applicazioni di machine vision), nell'elaborazione dei segnali. (predizione, modellizzazione di sistemi e filtraggio del rumore), nella modellizzazione e predizione in economia e finanza, nel controllo di complessi servomeccanismi in robotica e nella sintesi funzionale.

Nell'ambito della sintesi funzionale, una possibile applicazione è nella sintesi di superfici multidimensionali complesse a partire da valori campione grazie alle capacità di interpolazione delle reti neurali.

Nel caso delle verifiche industriali, D Glover ha sviluppato una rete con 32 elementi di input, uno strato nascosto con un numero di elementi compreso fra 20 e 40 e uno strato di output con un numero di elementi pari al numero delle classi da individuare e ha usato la backpropagation per addestrarla a svolgere compiti quali l'esame di etichette e il sorting di pezzi.

Modalità di implementazione dei sistemi di calcolo neurale.

I sistemi di calcolo neurale rappresentano modelli astratti che devono essere implementati su sistemi reali in modo da poter essere utilizzati effettivamente in applicazioni concrete.

Perché ciò sia possibile, le strade sono essenzialmente due: fare uso di simulatori software con o senza acceleratori hardware oppure ricorrere ad hardwre specializzato quale quello dei Neurocomputer.

Un modo di implementare un sistema di Neural Computing è quello di simulare sia l'architettura delle reti neurali sia gli algoritmi di apprendimento via software su architetture tradizionali con o senza acceleratori hardware.

L'uso di simulatori prevede l'implementazione di ambienti che permettono la definizione di topologie di rete e delle relative regole di addestramento, in modo che sia possibile salvare in strutture permanenti le reti così definite. Il problema più grosso legato ai simulatori è che le loro prestazioni sono limitate da quelle del sistema ospite per cui l'addestramento di reti di dimensioni medio-grosse su compiti non banali può richiedere tempi anche molto lunghi con conseguenti ripercussioni sui tempi di sviluppo di una applicazione neurale. Un soluzione ormai classica è quella di fare uso di acceleratori hardware, un'altra soluzione è quella di collocare il simulatore su un host, demandando l'effettiva esecuzione delle operazioni cosiddette compute-intensive (vedi oltre) ad un hardware specializzato, quale può essere un Neurocomputer.

L'adozione di acceleratori hardware permette di elevare le prestazioni dei sistemi in modo da rendere più agevole l'uso del Neural Computing per lo sviluppo di applicazioni anche real-time e permette inoltre di implementare in modo più diretto molti dei tratti caratteristici delle reti neurali. Un Neurocomputer può essere visto, in questa ottica, come un acceleratore hardware di "ultima generazione", dotato di un proprio apparato software per lo scheduling dei processi e la comunicazione con l'host. In quanto segue non ci si propone, come è ovvio che sia, un esame approfondito di tutti gli acceleratori presenti sul mercato. Lo scopo è infatti quello di presentare la filosofia che sta alla base dell'uso di tali strumenti, visti come mezzi per ovviare alle basse prestazioni dei PC e delle Workstation tradizionali. Nelle presenti note, essenzialmente per motivi di spazio, non si farà cenno alle possibilità offerte dall'uso di transputer, di supercalcolatori quali il Cray o di macchine puramente SIMD quali la Connection Machine (peraltro utilizzabili ed utilizzate allo scopo) considerato anche che nessuno di essi rappresenta un approccio metodologico radicalmente diverso rispetto ai sistemi tradizionali con o senza acceleratori hardware oppure rispetto all'adozione di un hardware specializzato quale un neurocomputer.

Fra gli acceleratori, i più diffusi sono Mark II e IV, le schede ANZA e ANZA-plus e il Nep ([KliGu88]). Mark III è un processore per il Neural Computing di tipo general purpose che può essere usato per implementare un ampio spettro di applicazioni. In questo caso viene fatto uso di processori neurali virtuali che multiplexano l'uso dei processori e delle interconnessioni fisiche su un gran numero di processori e connessioni virtuali ([KliGu88]).

MarkIII contiene 8100 PE e 417000 connessioni pesate e può essere usato come coprocessore su un sistema Vax.

MarkIV è una versione migliorata del precedente e fa uso di una tecnologia che consente l'implementazione di un maggior numero di tipologie di rete neurale.

Le schede ANZA e ANZA-plus sono schede progettate esplicitamente per la simulazione di reti neurali. La prima usa un processore Motorola 60820 con coprocessore numerico 60821 e permette di arrivare fino a 45000 interconnessioni/sec mentre la versione plus presenta una unità aritmetica ad alte prestazioni e un cip in virgola mobile che gli consentono di arrivare a 1500000 interconnessioni/sec. Si fa notare per inciso che Sinapse-1 è documentato arrivare a 10^9 interconnessioni/sec.

Nep (Network Emulator Processor), infine, è un dispositivo molto potente (può aggiornare una rete di dimensioni massime 20 o 30 volte al secondo) che fornisce un supporto hardware per l'emulazione di grosse reti neurali di tipo associativo. Nep è caratterizzato da interfacce molto veloci che rendono possibile il collegamento di più Nep in cascata in modo da ottenere buone prestazioni ed implementare reti comunque grosse, delle quali ogni Nep emula una parte.

Altre tecniche per l'implementazione dei sistemi per il Neural Computing prevedono l'uso di chip di silicio e di processori ottici, la cui trattazione esulala dall'ambito delle presenti note. In relazione ai processori ottici si fanno solamente notare i notevoli parallelismi con i sistemi per il Neural Computing dato che i processori ottici fanno uso di grossi array di elementi di elaborazione semplici che funzionano in modo intrinsecamente parallelo.

I simulatori, con o senza accelleratori hardware, rappresentano in molti casi l'unico strumento disponibile agli utilizzatori per compiere esperienze su architetture neurali. Un simulatore rappresenta essenzialmente uno strumento ad alto livello la cui efficienza, come detto in precedenza, dipende da quella dei supporti hardware e software e la cui flessibilità è in genere embedded nel prodotto. L'uso di un neurocomputer, non confrontabile come filosofia con gli altri strumenti hardware suddetti, permette di accedere ad uno strumento che non è, ovviamente, alternativo al simulatore ma con il quale un simulatore si può interfacciaare e che permette di operare a basso livello sulla struttura algoritmica delle reti con una efficienza ed una flessibilità notevoli.

Un esempio di simulatore è dato dallo Stuttgart Neural Network Simulator (SNNS, vedi [Stu94]). Lo SNNS rappresenta un ambiente di simulazione efficiente e flessibile per la ricerca sulle reti neurali e lo sviluppo di applicazioni in tale settore ed è essenzialmente composto da due grossi moduli interagenti: il Kernel e l'interfaccia grafica, XGUI.

Il Kernel lavora sulle strutture dati interne che rappresentano le singole reti neurali mentre la XGUI fornisce una rappresentazione grafica delle reti e controlla il Kernel durante la simulazione.

La XGUI svolge compiti di alto livello molto complessi dato che:

- permette la creazione e la modifica in modo rapido ed efficiente di reti complesse;
- è una interfaccia orientata anche all'utente inesperto grazie ad un help in linea in parte sensibile al contesto;
- permette la selezione solo degli aspetti interessanti della rappresentazione visiva di una rete in modo da evidenziare solo certe sue parti o certe sue caratteristiche mediante windows multiple e da nascondere certe informazioni indesiderate con una tecnica di layering.

Il Kernel, a sua volta, permette agli utenti la definizione sia di funzioni di attivazione sia di procedure di apprendimento custom ed inoltre gestisce le rappresentazioni interne delle singole reti neurali. Visto come modulo software, il Kernel ha due interfacce: una verso la XGUI che consente una manipolazione diretta delle reti e l'altra verso un compilatore di reti, Nessus, che consente la traduzione di descrizioni ad alto livello in descrizioni dette di livello intermedio gestibili direttamente dal Kernel stesso.

Il neurocomputer Sinapse-1.[14]

Introduzione.

Le reti neurali eseguono elaborazioni ad elevato parallelismo, in genere caratterizzate dall'uso di funzioni non lineari, facendo uso di molti processori elementari corrispondenti ai neuroni dei sistemi biologici. I sistemi di calcolo caratterizzati da hardware tradizionale, anche con l'ausilio di acceleratori, permettono di ottenere simulazioni delle architetture di rete e delle strategie di apprendimento che sono pesantemente time-consuming e pertanto comportano lunghi tempi di sviluppo.

L'adozione di hardware specializzato, in questo caso un Neurocomputer, permette di ridurre notevolmente i tempi e i costi necessari allo sviluppo di applicazioni basate su reti neurali. L'uso di hardware dedicato dovrebbe supportare reti neurali di topologia arbitraria e consentire la progettazione di algoritmi di addestramento qualunque. Da un Neurocomputer si dovrebbe pretendere ([Ra et al. 94]) una potenza di calcolo almeno pari a tre volte quella dei sistemi attuali, in modo da limitare i tempi di sviluppo (condizionati dai tempi richiesti per addestrare la rete su compiti non banali), la capacità di supportare topologie arbitrarie e reti di dimensioni comunque grandi ed infine una interfaccia utente user-friendly per rendere il sistema facile da usare.

Il Neurocomputer Synapse-1 (NS1, vedi figura 3) è in grado di soddisfare tali requisiti dato che è caratterizzato da ([Ra et al. 94]):

- processori custom, dedicati all'esecuzione di operazioni, quali prodotto di matrici, che richiedono elevate potenze di calcolo (operazioni compute-intensive);
- processori tradizionali per l'esecuzione di operazioni neurali non compute-intensive, di controllo e di comunicazione con l'host;
- memorie dedicate per l'implementazione della topologia delle reti mediante la memorizzazione della matrice dei pesi;
- una architettura multiprocessor per l'implementazione di topologie arbitrarie e
- un linguaggio, detto nAPL (neural Algorithms Programming Language), ad alto livello che prevede la definizione delle operazioni base dei processori dedicati come tipi di dati separati.

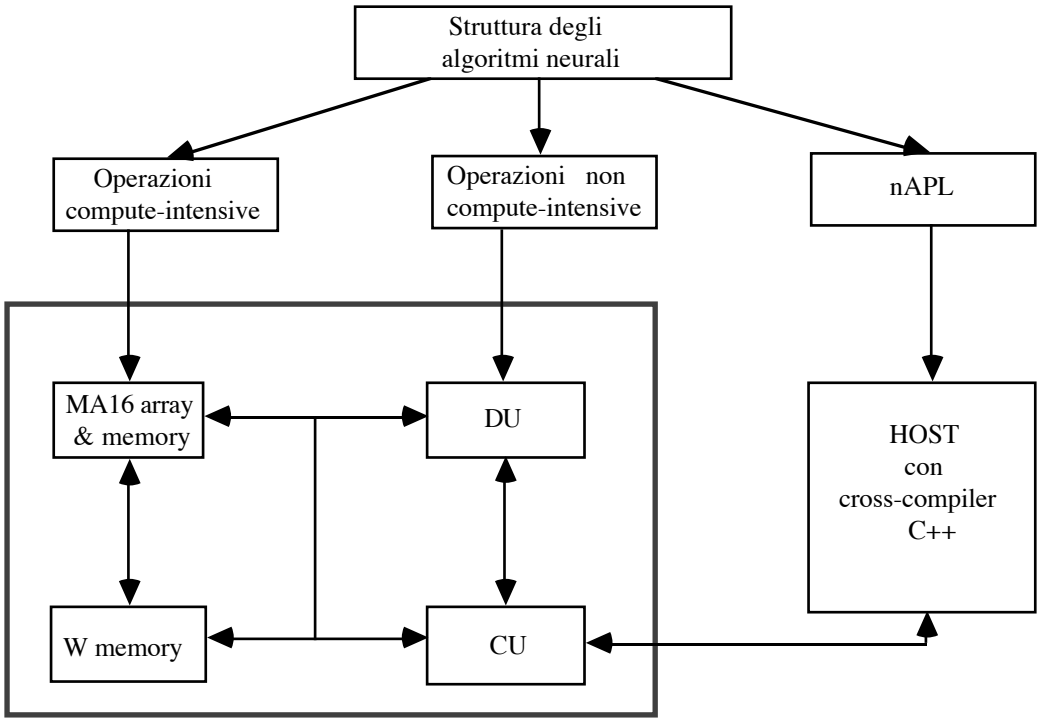

Figura 3. Visione sistemistica del Neurocomputer Synapse-1 e dell'host connesso (da [Ra et al. 94])

14 Il contenuto del presente paragrafo (e dei relativi sottoparagrafi) proviene dal testo di Ramacher U., W. Raab, J. Anlauf, U. Hacmann & M. Weßeling citato in bibliografia.

Da un prospettiva di tipo sistemistico il punto di partenza è la considerazione della struttura degli algoritmi neurali, caratterizzati da operazioni compute-intensive e operazioni non compute-intensive accessibili attraverso il linguaggio nAPL, implementato in C++ su un host di cui il Neurocomputer può essere visto come un coprocessore.

Le operazioni compute-intensive sono eseguite su chip custom, detti MA16 Neural Signal Processor (MA16NSP), montati a gruppi di otto su schede, dette MA16array, che contengono anche le memorie W su cui vengono memorizzate le matrici dei pesi che caratterizzano la rete a livello algoritmico.

Le operazioni non compute-intensive sono eseguite dalla Data Unit (DU) mentre un'altra unità , la Control Unit (CU) ha il compito di intercalare l'esecuzione di operazioni non compute-intensive e compute-intensive e di colloquiare con l'host.

Le schede di memoria W e ti tipo MA16array sono presenti in quantità dipendenti dall'applicazione.

Il linguaggio nAPL, caratterizzato da una sintassi tipica di un linguaggio ad alto livello, permette il mapping sull'architettura multiprocessor e multimemory della topologia di rete. Gli utenti hanno a disposizione una descrizione algoritmica della rete su una architettura che consente lo sviluppo di applicazioni che lavorano in tempo reale. Una delle caratteristiche dell'nAPL è quella di distinguere per tipo e posizione le operazioni compute-intensive da quelle non compute-intensive in modo da schedularle sui processori adatti, il tutto in modo trasparente agli utenti.

Cenni all'architettura hardware di NS1.

L'architettura di NS1 è di tipo multiprocessor e multimemory con memorie modulari in cui ogni elemento contribuisce alla sintesi degli algoritmi neurali mediante processi che evolvono in parallelo.

Gli elementi base dell'architettura (vedi figura 4) sono le schede MA16array, ciascuna con 8 processori special purpose ad architettura modulare e sistolica, due unità di tipo general purpose basate su processore Motorola MC 68040 dette Control Unit e Data Unit e un certo numero di tipi di memorie, fra cui la memoria dei pesi o W memory (Wm), la Z memory (Zm) e la Y memory (Ym).

Ognuna delle MA16array ha otto processori MA16NSP collegati in modo da formare un array bidimensionale, ognuno dei quali ha a sua disposizione una Zm locale mentre la scheda nel suo complesso ha accesso ad una scheda Wm che contiene quattro banchi di memoria, ciascuno di taglio multiplo di 16Mbyte. La Z memory di figura 4 in realtà e localizzata sulle schede MA16array in banchi di memoria che ognuno dei processori MA16NSP può usare come buffer locale per i propri dati.

Ogni scheda MA16array ha 32Mbyte di RAM (8 banchi di 4Mbyte ciascuno, uno per ogni MA16NSP) e offre prestazioni molto elevate: è in grado di eseguire $3.2*10^9$ prodotti di coppie di dati su 16 bit al secondo, di trasferire dati con velocità dell'ordine del Gbyte/sec e informazioni di controllo con velocità dell'ordine delle decine o centinaia di Mbyte/sec, a seconda del tipo di bus interessato al trasferimento.

Le schede Wm hanno ciascuna quattro banchi di memoria, uno per ogni coppia di MA16NSP sulla scheda MA16ARRAY cui la memoria è connessa.

Tali memorie presentano due modalità di trasferimento: nella prima (front bus mode) i dati sono trasferiti nella Wm dalla CU e tutte le celle di memoria sono accedute in modo uniforme mentre nella seconda (back-plane mode) la Wm è allocata ai processori MA12NSP sebbene i dati possano essere trasferiti contemporaneamente alla CU per un'operazione di monitoraggio.

La DU, caratterizzata da una CPU convenzionale Motorola 68040 e da chip CMOS, ha il compito di eseguire le operazioni non compute-intensive, fornisce dati in ingresso alle schede MA16array e riceve da queste i risultati.

Lo scambio di dati fra le schede MA16array e la DU avviene attraverso un anello che comprende la Ym in cui vengono memorizzati sia i pattern di input sia i valori di stato dei neuroni.

I risultati possono, infatti, essere post-elaborati sia da un hardware dedicato presente sulla DU (la Integer Unit, IU) sia dalla CPU locale alla DU stessa prima di essere salvati in memoria.

La DU, a tale scopo, ha accesso sia ad una memoria general purpose (C memory) sia ad una memoria locale (Memoria DU) che ha il compito di memorizzare le procedure per l'esecuzione delle operazioni non compute-intensive.

La C memory è la memoria principale della DU ed è una memoria dual port, accessibile anche dal bus VME[15] senza che sia necessario mettere la CPU locale in stato di attesa. Pertanto tale memoria è utilizzabile per la comunicazione fra schede.

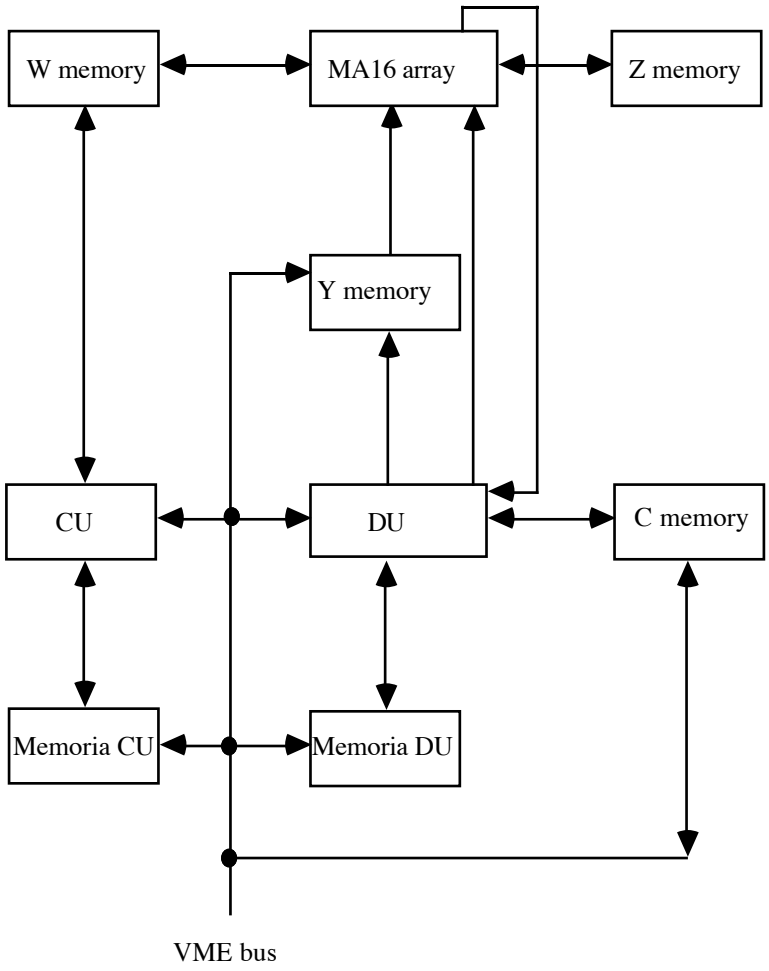

Figura 4. Architettura hardware (da [Ra et al. 94]): visione funzionale con i soli bus dati

La Ym di solito contiene i dati da elaborare sotto forma di pixel di immagini o di campioni di segnali oppure dati relativi alla rete neurale in corso di simulazione quali lo stato di attività dei vari neuroni.

La Ym ha una capacità massima di 16 Mbyte ed è composta di due banchi accessibili uno indipendentemente dall'altro. Tale caratteristica permette un trasferimento dati dal bus VME o dalla Integer Unit mentre, contemporaneamente, altri dati sono letti e inviati attraverso un buffer circolare FIFO alle schede MA16array in blocchi dati fino a 2*1024 word accessibili ripetutamente da tali schede.

Gli indirizzi per il trasferimento dei dati in DMA sono generati dalla CU (vedi). La Ym è responsabile della capacità delle schede MA16array di elaborare i pattern di ingresso in parallelo dato che i dati in ingresso possono essere spostati nella Ym dal bus VME un pattern alla volta mentre dalla Ym sono caricati su un buffer circolare FIFO in modo da essere intervallati nel modo richiesto dalle schede MA16array.

15 La DU ha due interfacce sul bus VME verso altre schede, una master e una slave: la prima è accessibile sia dalla CPU locale sia dalla IU per accedere ad altre schede mentre la seconda consente l'accesso alle memorie C e Y ed alla IU senza che la CPU locale si debba mettere in stato di attesa.

Sulla DU risiede la IU il cui compito è quello di post-elaborare (previa una conversione da floating point a formato intero) i dati X_i prodotti dalle schede MA16array prima di ripassarli nella Ym.

La IU è bufferizzata (in modo da poter elaborare un flusso continuo di dati) ed è in grado di eseguire direttamente e senza intervento della CPU locale operazioni della forma $f(c(X_i+s))$ in cui c ed s sono delle costanti oppure del tipo $f(c_i(X_i+s_i))$ in cui c_i e s_i sono parametri variabili precaricati dalla CPU locale in code FIFO dedicate agli operandi e ciascun parametro può essere usato una volta oppure otto volte (una per ognuno dei possibili pattern che possono essere elaborati contemporaneamente). Le funzioni f() sono implementate in hardware su SRAM[16] con tabelle di ricerca (Look-up table o LUT) che devono essere inizializzate dalla CPU locale e che permettono di stabilire corrispondenze arbitrarie fra coppie di valori su 16 bit. Nel caso la risoluzione richiesta sia minore è possibile memorizzare più LUT allo stesso tempo.

La CU ha il compito di controllare il flusso dei dati e di intercalare, secondo quanto specificato dall'algoritmo in corso di esecuzione, le operazioni non compute-intensive con quelle compute-intensive utilizzando una memoria locale (Memoria CU) per memorizzare le procedure di controllo. Le memorie Memoria CU e Memoria DU hanno, da questo punto di vista, il compito di memorizzare le routine del Sistema Operativo residente per l'inizializzazione, il controllo e la gestione di processi paralleli.

La CU è caratterizzata da una CPU locale come la DU, da un sequenzializzatore di microprogrammi, un'interfaccia verso la Wm e un generatore di indirizzi per il controller in DMA della Ym presente sulla DU.

Il generatore di indirizzi per la Ym definisce gli indirizzi usando informazioni fornitigli dalla CPU locale attraverso una coda. Sono possibili due modalità operative dette direct mode e loop mode. Secondo la prima la CPU fornisce un indirizzo esplicito che il generatore suddivide in un indirizzo di riga ed in uno di colonna prima di passarlo al controller DMA della Ym. Nel caso della seconda il generatore gode di una certa autonomia dato che è in grado di produrre una successione di indirizzi a partire da un indirizzo di inizio, dal numero di cicli da eseguire e dall'incremento da applicare per ciascun ciclo, valori fornitigli dalla CPU locale. É possibile avere più cicli intervallati, ciascuno con i suoi valori caratteristici, in modo che una porzione di immagine possa essere trasferita in memoria in modo molto efficiente a partire da una immagine globale memorizzata per righe o colonne in Ym.

La generazione di indirizzi avviene sulla base di poche informazioni fornite al generatore dalla CPU consentendo una elevata velocità di trasferimento con un basso carico per la CPU (gestione in fast page mode).

Il sequenzializzatore di microprogrammi ha il compito di controllare le schede MA16array e gli elementi che devono essere sincronizzati con queste, fra cui Wm, Zm e coda FIFO dei dati sulla DU.

Le informazioni di controllo necessarie vengono prelevate dalla memoria di microprogramma. Il caricamento dell'indirizzo di partenza di un microprogramma per l'esecuzione di una operazione elementare (ad esempio un prodotto di matrici) nel program counter del sequenzializzatore fa si che questi inizi ad eseguire il microprogramma opportuno la cui terminazione è comunicata alla CPU con una apposita richiesta di interrupt. Durante l'esecuzione di un microprogramma è possibile che si abbia sincronizzazione fra CPU e sequenzializzatore mediante uno scambio di segnali di interrupt. Il sequenzializzatore, inoltre, controlla che la coda FIFO dei dati sulla DU né si vuoti né rischi di dare luogo ad un overflow, eventualmente ritardando l'esecuzione delle istruzioni del microprogramma.

I microprogrammi che il sequenzializzatore esegue e che implementano le varie operazioni elementari vengono caricati nella memoria di microprogramma dall'hard disk dell'host caratterizzati da valori di default che possono essere modificati a run-time in modo da adattare il microcodice alle esigenze correnti.

L'interfaccia con la Wm, infine, consente lo scambio di dati attraverso code FIFO accessibili in lettura e in scrittura e driver di bus secondo due possibili modalità:
- direct mode ovvero accesso diretto della CPU alla Wm con scambio dati attraverso code FIFO in modo da rendere accessibili le singole celle di memoria oppure

16 Il termine SRAM individua le RAM statiche, cioè memorie statiche asincrone non pilotate da un clock.

- block mode ossia lettura o scrittura di blocchi di dati che richiede l'esecuzione di un microprogramma per la gestione del trasferimento mediante code FIFO e la preventiva preparazione della dimensione del blocco e dell'indirizzo di partenza.

I componenti software.

Gli elementi software dell'architettura (vedi figura 5) sono responsabili dell'esecuzione sia delle operazioni associate alla struttura delle reti neurali sia delle operazioni di scheduling, trasferimento dati (pattern in ingresso e in uscita, matrici dei pesi e elementi necessari al controllo) e di comunicazione con l'host. Le operazioni associate alla struttura delle reti neurali sono responsabili degli elevati requisiti di tempo di elaborazione necessari per la simulazione di algoritmi neurali per cui l'architettura è finalizzata alla loro esecuzione nel modo più efficiente possibile.

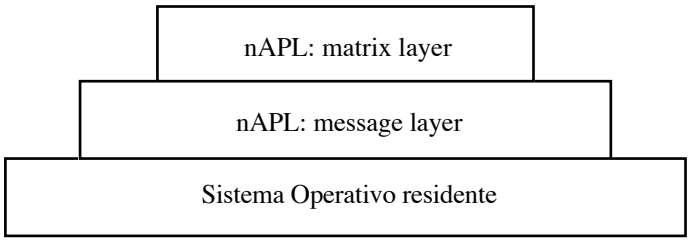

Figura 5. Architettura software di Synapse-1 (da [Ra et al. 94])

La struttura base del linguaggio di programmazione per l'NS1 si fonda sulla decomposizione di qualunque algoritmo neurale in operazioni compute-intensive e non compute-intensive. Una siffatta suddivisione riflette l'architettura hardware dell'NS1 per cui si può affermare che la sintassi del linguaggio di programmazione si modella sull'architettura. Ad esempio la dichiarazione di variabili di tipo matriciale o di funzioni vedremo che prevede la specificazione della memoria o del processore cui vanno assegnate attraverso l'uso di classi di appartenenza.

Il linguaggio nAPL, il cui potere espressivo è legato alla disponibilità di una estesa libreria di operazioni matriciali ed al fatto di essere implementato in C++ come una libreria di classi, può essere pensato some suddiviso in due strati (vedi figura 5) dei quali uno si interfaccia con il cosiddetto Sistema Operativo residente.

L'interfaccia cui si accede con programmi scritti in C++ è il cosiddetto matrix layer che fornisce tipi di dati astratti associati all'hardware sottostante quali matrici a blocchi in virgola mobile e tabelle di ricerca (look-up table o LUT) che consentono l'implementazione di funzioni arbitrarie direttamente in hardware, tipi che però nascondono al programmatore molta della complessità dell'hardware, compresa la necessità di una gestione esplicita dell'allocazione delle quantità di memoria.

Il matrix layer mette a disposizione una classe di matrici per ogni tipo di memoria (Ym, Wm, Zm e Cm) e ciascuna classe è caratterizzata dalle operazioni elementari (metodi) eseguibili su matrici di quel tipo e quindi sul corrispondente tipo di memoria. Le operazioni elementari sono eseguite mediante messaggi che creano processi paralleli sulla DU e sulla CU in modo che questi co-operino in modo corretto.

Il message layer rappresenta l'interfaccia object-oriented verso il coprocessore, interfaccia caratterizzata da due tipi di messaggi:
- chiamate di procedura remota (RPC) e cioè messaggi bloccanti che causano l'attesa da parte dell'host della terminazione della procedura sull'NS1, sono usate per operazioni non parallelizzabili che richiedono tempi di elaborazione molto brevi;
- chiamate di job e cioè messaggi non bloccanti che danno luogo ad un task sull'NS1 che inizia un job che evolve in parallelo alla computazione sull'host fino alla propria terminazione. É così possibile avviare più job in parallelo per elaborare una istruzione del linguaggio nAPL. Il compito di controllare e gestire tali job spetta al Sistema Operativo residente sulla DU e sulla CU (vedi), il cui compito è quello di implementare i job in hardware mediante microprogrammi.

Al di sopra del livello matrix layer, che può essere visto come una Application Programming Interface, è ovviamente possibile implementare ulteriori livelli di software applicativo hardware independent quali GUI e simulatori di reti neurali.

Pur essendo configurabile come un coprocessore collegato tramite un bus VME ad un host Sun Sparc workstation, l'NS1 è caratterizzato da un suo proprio Sistema Operativo

localizzato sulla CU e sulla DU con il compito di controllare l'NS1 e detto Sistema Operativo residente.

Il Sistema Operativo residente, caratterizzato da un kernel multitasking guidato da eventi per la gestione di task paralleli, si compone di una parte comune alla DU e alla CU e dedicata al colloquio con l'host e di porzioni di codice dipendenti dall'hardware, aventi il compito di controllare gli elementi caratteristici delle singole schede.

Le parti del Sistema Operativo residente dipendenti dalla particolare scheda sono implementate mediante un insieme di task, ognuno dei quali gestisce un certo numero di subroutines, o funzioni di job, accessibili dal software residente sull'host mediante uno scambio di messaggi. Si ricorda che, a livello utente, l'interfaccia è di tipo Object Oriented per cui si parla di metodi associati ad oggetti appartenenti a certe classi e a cui si accede mediante richieste di servizi. L'ambiente di sviluppo per le CPU sulla DU e sulla CU è costituito dal compilatore C, da un assemblatore e dalle librerie del linguaggio C.

Il Sistema Operativo residente sulla CU è caratterizzato da tre task detti sequencer task, ring buffer (coda circolare) task e task ausiliario, il quale svolge compiti ausiliari quali quelli di debugging.

Il primo, durante la fase di inizializzazione, carica nella memoria di microprogramma i microcodici residenti sull'hard disk dell'host[17] mentre, durante il funzionamento normale, gestisce il sequenzializzatore (nel senso che gli da il via e attende la fine del microprogramma) ed i relativi interrupt (fornendo e istallando le routine di gestione opportune) e, mediante semafori o scambio di messaggi, si sincronizza con i task che si occupano del trasferimento dati colloquiando con le schede MA16array.

Il secondo fornisce le informazioni di controllo al generatore di indirizzi per la Ym per mezzo di una coda FIFO in modo che sia possibile iniziare il trasferimento dati in DMA fra la Ym e la coda circolare.

Il Sistema Operativo residente sulla DU è caratterizzato da quattro task detti integer unit task, preload task, I/O task e task ausiliario, quest'ultimo responsabile di operazioni ausiliarie.

L'integer unit task gestisce l'unità aritmetica intera dato che supporta le operazioni da essa eseguite fornendo le informazioni di controllo e precaricando la LUT richiesta oppure le code del moltiplicatore e dell'addizionatore.

Il preload task si occupa del trasferimento dati dalla Cm alle code FIFO sulle schede MA16 mentre l'I/O task gestisce il trasferimento dei dati fra le memorie Y e C.

La parte comune del Sistema Operativo residente, infine, è caratterizzata da un kernel e da un insieme di routine di libreria. Le routine di libreria permettono l'implementazione di moduli per l'invio dei messaggi e di moduli per la gestione delle RPC inviate dall'host che traducono messaggi di richiesta di servizio.

Il kernel ha anch'esso una struttura modulare ed è caratterizzato da moduli per la gestione delle richieste di interrupt e dei messaggi in arrivo che interagiscono con un modulo scheduler che si occupa della schedulazione dei task.

Lo scheduler distribuisce il controllo fra i task eseguibili e cioè fra quelli che hanno ricevuto a quell'istante una richiesta di esecuzione di una funzione di job. Ai task disponibili sono assegnate priorità diverse in base alle quali lo scheduler sceglie il task da mandare in esecuzione. I task potenzialmente eseguibili sono noti a priori e le loro priorità sono fissate a valori di default durante la fase di bootstrap, sebbene siano modificabili a run-time. Oltre ad una scelta su base prioritaria lo scheduler può scegliere un task ben preciso, fra quelli pronti, sulla base di una RPC forzante e ciò anche in presenza di task pronti con priorità più alta.

Il task prescelto rimane in esecuzione fino a che non si sospende oppure viene sospeso da una richiesta di interrupt: in entrambi i casi lo scheduler sceglie un nuovo task fra quelli in attesa di essere eseguiti. Le richieste di interrupt significative in questo contesto sono generate da eventi a livello hardware che rendono necessaria una commutazione fra i task.

Lo scheduler può intervenire anche in seguito alla ricezione di un messaggio che si può tradurre nella esecuzione del task che ha il controllo sulla subroutine invocata dal messaggio ricevuto oppure nel rendere semplicemente eseguibile un ben preciso task dopo avergli messo a disposizione il contenuto del messaggio e cioè i necessari parametri.

---

17 Il Neurocomputer non possiede memorie di massa per cui la fase di bootstrap avviene durante un colloquio con l'host, colloquio gestito da un task sull'NS1.

L'nAPL è il componente software funzionalmente localizzato al di sopra del Sistema Operativo residente e a cui gli utenti hanno accesso diretto. L'nAPL è suddiviso a sua volta in due strati: message layer e matrix layer.

Il message layer rappresenta l'interfaccia con il Sistema Operativo residente e l'hardware dell'NS1: a questo livello si ha una elevata flessibilità e la possibilità di sfruttare al meglio le caratteristiche dell'hardware.

I comandi in arrivo dall'host sono trasferiti al coprocessore in due modi e cioè o come chiamate di procedura remota (RPC) o come chiamate di job (vedi).

Le RPC possono essere eseguite sia dalla DU sia dalla CU per cui il linguaggio mette a disposizione due tipi di oggetti di classe RPC, uno per ciascuna di tali schede.

Le chiamate di job sono richieste non bloccanti (vedi) il cui scopo è quello di avviare, mediante invocazioni di metodi in programmi C++, più job in parallelo che possano cooperare per l'esecuzione di operazioni elementari: tale effetto è ottenuto fornendo a task distinti più job da eseguirsi contemporaneamente[18]. In relazione alle chiamate di job sono resi disponibili oggetti propri e della CU e della DU e associati ai diversi task eseguibili da tali unità (vedi paragrafo "Alcuni esempi di utilizzo").

Il message layer fornisce il supporto per lo strato matrix layer sovrastante (e può fornirlo per tutte le applicazioni che accedono a tale livello all'nAPL). Oltre a ciò richiede conoscenze precise dell'hardware e del software dell'NS1 in quanto a tale livello l'utente/programmatore deve gestire esplicitamente la memoria dell'NS1, deve decomporre eventuali matrici di grosse dimensioni in sottomatrici, deve esplicitamente programmare operazioni di shift per scalare i dati e, last but not least, non può contare, per motivi di efficienza, su alcun check automatico dei parametri.

Il matrix layer mette a disposizione degli utenti due nuovi tipi di dati e cioè le matrici a blocchi in virgola mobile e le funzioni definibili dall'utente sotto forma di LUT.

Tali classi permettono di nascondere all'utente i problemi legati alla gestione delle memorie del coprocessore dato che ricercano automaticamente l'area libera di memoria richiesta, la riservano e la deallocano mediante un metodo detto distruttore di classe al momento in cui l'area riservata non è più utilizzata. Oltre a ciò nei programmi non compaiono più indirizzamenti espliciti dell'hardware e la gestione dell'elaborazione di matrici molto grosse è del tutto trasparente all'utente. Il parallelismo esistente a livello hardware è invisibile all'utente ed è accessibile solo in modo indiretto mediante l'invocazione di operazioni elementari incapsulate nei metodi forniti dal linguaggio.

Vediamo più in dettaglio i tipi di dati matrice e LUT. Le matrici a blocchi sono caratterizzate da elementi con una mantissa su 16 bit, compreso il bit di segno, e un esponente comune (base 2) su 32 bit. Le mantisse permettono di esprimere numeri in complemento a due nell'intervallo [-1, 1) che vengono scalati in base al valore dell'esponente comune.

Le matrici di interi sono ottenute con un valore dell'esponente pari a 15. Sulle matrici di interi non viene eseguita alcuna operazione di normalizzazione a seguito di operazioni aritmetiche, come invece accade nel caso delle matrici in virgola mobile.

Oltre a tali classi di matrici esiste la classe speciale WF15TagMatrix in cui le mantisse sono definite in modo simile al caso del tipo in memoria W e sono caratterizzate da un bit di segno, 14 bit per il valore numerico e un bit di tag. Se il tag vale 0 il valore contenuto negli altri bit è una costante mentre se vale 1 il valore contenuto negli altri bit è una variabile.

In tal modo si individuano i valori non aggiornabili rispetto a quelli aggiornabili e la distinzione può essere utile nel caso di pesi da aggiornare per strati non completamente connessi e strategie "onerose" come la backpropagation. L'architettura hardware prevede il posizionamento delle matrici che può essere affrontato mediante la definizione di classi di matrici, associando ciascuna classe ad un tipo di memoria (Wm, Ym, Cm e Zm). Vediamo in proposito alcuni esempi. L'istruzione che segue

```
WF16Matrix  W(1024, 512, 3)
```

definisce una matrice di nome W come elemento della classe W (e cioè memorizzata in memoria W) di 1024 righe e 512 colonne con un valore dell'esponente pari a 3, per cui i

18 Si ricorda che l'architettura è di tipo multiprocessor e multimemory.

suoi elementi assumono valori nell'intervallo [-8, 8) e presentano una risoluzione di 16 bit mentre l'istruzione

YF16Matrix Y1(512, 8, 0)

definisce la matrice di nome Y1 come elemento della classe Y di 512 righe, 8 colonne e con valore dell'esponente comune 0.

La necessità di posizionare le matrici nella memoria adatta discende dal fatto che molte operazioni elementari sono eseguibili solo se le matrici interessate sono localizzate nelle memorie appropriate e le prestazioni ottenibili sono migliori se si opera un accorto posizionamento delle matrici.

Una LUT hardware è implementata sul coprocessore con 2^{16} entry di 16 bit ciascuna e può essere usata per stabilire una corrispondenza qualunque fra coppie di dati su 16 bit. Il linguaggio mette a disposizione una classe F16lut che permette di convertire una qualunque funzione nella corrispondente rappresentazione "cablata" in hardware.

A tale scopo una LUT è caratterizzata da una tabella di mantisse e due esponenti, uno per l'interpretazione corretta dei bit di ingresso e l'altro per i bit di uscita memorizzati in tabella: tali esponenti sono usati per il controllo degli shifter hardware.

Vediamo un esempio. L'istruzione

F17lutTanh f(yExp, xExp, m)

genera una LUT di nome f che calcola la funzione y = f (x) = tanh (mx), in cui yExp è il valore dell'esponente (shift) delle y, e xExp è l'analogo valore per le x. Dopo tale definizione, il nome della funzione può essere usato come un qualsiasi argomento in altre istruzioni.

Alcuni esempi di utilizzo.

A illustrazione delle potenzialità di NS1 si riportano brevemente alcuni esempi di utilizzo di NS1 sia a livello di message layer sia a livello di matrix layer.

Message layer.

A livello di message layer il programmatore ha visibilità di molte delle caratteristiche dell'hardware e può quindi sfruttarle al meglio, ma ha anche l'onere (fra le altre cose) di gestire le allocazioni di memoria e le sincronizzazioni fra i task.

Ad esempio la seguente istruzione:

ringBufferTask.transfer(1024,512,3,0x1234);

da luogo ad un task che esegue un job consistente nel trasferimento di una matrice di 1024 righe e 512 colonne dalla locazione 0x1234 in Ym sui chip MA16NSP 3 volte attraverso il ring buffer (buffer circolare fra la Ym e le schede MA16array). Tale task può essere sincronizzato esplicitamente con un task sulla IU con una chiamata del tipo

integerUnitTask.wait();

per cui se ho

integerUnitTask.wait();
ringBufferTask.transfer(1024,512,3,0x1234);

condiziono il trasferimento alla terminazione di un task sulla IU, operazione necessaria se anche il primo dei due task prevede l'uso del ring buffer.

Un altro esempio è il prodotto di matrici Y2 = W * Y1. A livello di message layer occorre specificare che:

- W si trova in Wm all'indirizzo WAdr ed è di m righe e n colonne;
- Y1 in Ym nel banco1 indirizzo Y1Adr di n righe e q colonne e

- Y2 in Ym nel banco2 indirizzo Y2Adr di m righe e q colonne.

Fatto questo, il programma ha il seguente aspetto (da [Ra et al. 94], pag. 37):

```
dataUnitRPC.ymembank(1);
ringBufferTask.tranfer(n, q, nRep, Y1Adr);
sequencerTask.ywy_mult(m, n, WAdr);
integerUnitTask.pipeline(m*q, Y2Adr, LUT_IDENTITY, MULT_ONE, ADD_ZERO, 1);
```

Si parte con la scelta del banco 1 di memoria Ym, si forniscono poi i dati per Y1, si esegue il prodotto con i valori della matrice W e attraverso la IU si forma la matrice risultato. Le costanti utilizzate hanno i seguenti significati:

LUT_IDENTITY	scelta di una LUT;
MULT_ONE	esegui il prodotto degli elementi della matrice risultato con 1 nel moltiplicatore;
ADD_ZERO	somma 0 agli stessi dati nel sommatore;
1	shift di un bit a destra del risultato degli MA16NSP.

A livello di matrix layer lo stesso risultato (sebbene forse con minore efficienza ma privilegiando la semplicità) lo si otterrebbe con la seguente istruzione:

```
Y2.mult(W, Y1, DataUnit(LUT_IDENTITY, MULT_ONE, ADD_ZERO, 1));
```

il cui maggiore potere espressivo è evidente.

Matrix layer.

Il primo esempio, tratto di nuovo da [Ra et al. 94], è quello di una operazione elementare in cui la matrice risultante viene memorizzata in Ym mentre gli operandi necessari ad una post-elaborazione nella DU sono specificati insieme a quelli necessari per l'esecuzione dell'operazione elementare. In assenza degli operandi di post-elaborazione, il risultato sarebbe trasferito direttamente dalle schede MA16array nella Ym.

L'esempio presentato è quello di un prodotto fra matrici seguito da una somma fra matrici (una delle matrici è composta da elementi costanti), da un prodotto di tutti gli elementi della matrice risultante per uno scalare costante e dall'applicazione di una funzione arbitraria f agli elementi della matrice risultato. Si vuole cioè realizzare la seguente operazione:

```
Y2 = f(s * (W * Y1 + C))
```

e ciò è possibile con la seguente semplice istruzione

```
Y2.mult(W, Y1, DataUnit(f, s, C));
```

L'architettura di NS1 è stata progettata in modo che la computazione sull'host e quella sull'NS1 possano evolvere in parallelo tutte le volte che ciò è possibile: l'host genera i messaggi che richiedono l'esecuzione di operazioni elementari sull'NS1. Se questo è occupato, i messaggi sono inseriti in una coda e inviati effettivamente solo quando l'NS1 ha finito l'operazione associata al messaggio precedente. Le condizioni di errore di qualunque tipo, a questo livello, sono gestibili attraverso una classe speciale che permette all'utente di definire classi di errori particolari in modo da poter gestire le diverse situazioni di eccezione quali overflow di code, condizioni di memoria piena e così via.

Vediamo ora un programma che implementa la fase di produzione di una rete multistrato con un buffer di input, uno strato nascosto e un buffer di output.

La rete è caratterizzata da un buffer di input con 1024 neuroni, uno strato nascosto (hidden layer) con 256 neuroni e un buffer di output con 64 neuroni. I tre strati sono fra loro completamente connessi, come illustrato in figura 6. Nell'esempio proposto si esamina

la fase di produzione e il calcolo viene eseguito simultaneamente per otto pattern di ingresso.

Le strutture dati coinvolte nel calcolo sono le seguenti:

- matrice dei pesi per le connessioni fra buffer di input e hidden layer, W1 (256*1024);
- matrice dei pesi per le connessioni fra hidden layer e buffer di output, W2 (64*256);
- matrice degli otto pattern in ingresso, Y0 (1024*8);
- matrice descrittiva dello stato dei neuroni nello strato nascosto, Y1 (256*8)
- matrice dello stato dei neuroni nello strato di output (64*8).

Le matrici W1 e W2 sono state determinate durante una opportuna fase di addestramento e a questo punto sono note, la matrice Y0 è formata da dati noti, la Y1 è calcolata come W1*Y0 mentre la Y2 come W2*Y1, a meno delle funzioni di trasferimento eseguite dai singoli neuroni per cui a rigore si ha Y1 = f1 (W1*Y0) e Y2 = f2 (W2*Y1).

Le funzioni f1 e f2 sono realizzate con delle LUT e sono applicate ai singoli elementi delle matrici interessate. La prima ha la seguente definizione:

Y1 = tanh (0.01 * W1 * Y0)

mentre la seconda ha la seguente definizione:

Y2 = sign (W2 * Y1)

Il programma in questione si compone dei seguenti passi ([Ra et al. 94]):
- dichiarazione delle matrici dei pesi W1 e W2 come appartenenti alla classe W16 e con esponente comune pari a 3;
- dichiarazione delle matrici degli stati Y1 e Y2 e di ingresso Y0 come appartenenti alla classe Y16 e con valore dell'esponente comune pari a 0;
- dichiarazione delle LUT per la valutazione delle funzioni caratteristiche dei neuroni sugli strati della rete, esponente comune pari a 13 per le x e a 0 per le y;
- lettura dei dati dai file dei pesi e dei pattern di input;-
- calcolo di Y2 in due passi e
- scrittura in un file dei risultati.

Riportiamo qui di seguito il codice del programma che eseguirebbe il calcolo (da [Ra et al. 94] pag. 43):

```
WF16Matrix      W1(256,1024,3);
WF16Matrix      W2(64, 256, 3);
YF16Matrix      Y0(1024, 8, 0);
YF16Matrix      Y1(256, 8, 0);
YF16Matrix      Y2(64, 8, 0);

F16LutTanh      f1(0, 13, 0.01);
F16LutSign      f2(0, 13);

W1.read ("w1.dat");
W2.read ("w2.dat");
Y0.read ("y0.dat");

Y1.mult (W1, Y0, DataUnit (f1, MULT_ONE, ADD_ZERO));
Y2.mult (W2, Y1, DataUnit (f2, MULT_ONE, ADD_ZERO));

Y2.write ("y2.dat");
```

La struttura di tale breve programma ne mostra immediatamente la maggiore semplicità rispetto all'esempio visto a livello di message layer: in questo caso si fa uso di classi di memoria e il caricamento dei dati con relativa successione di operazioni di più basso livello è del tutto mascherata.

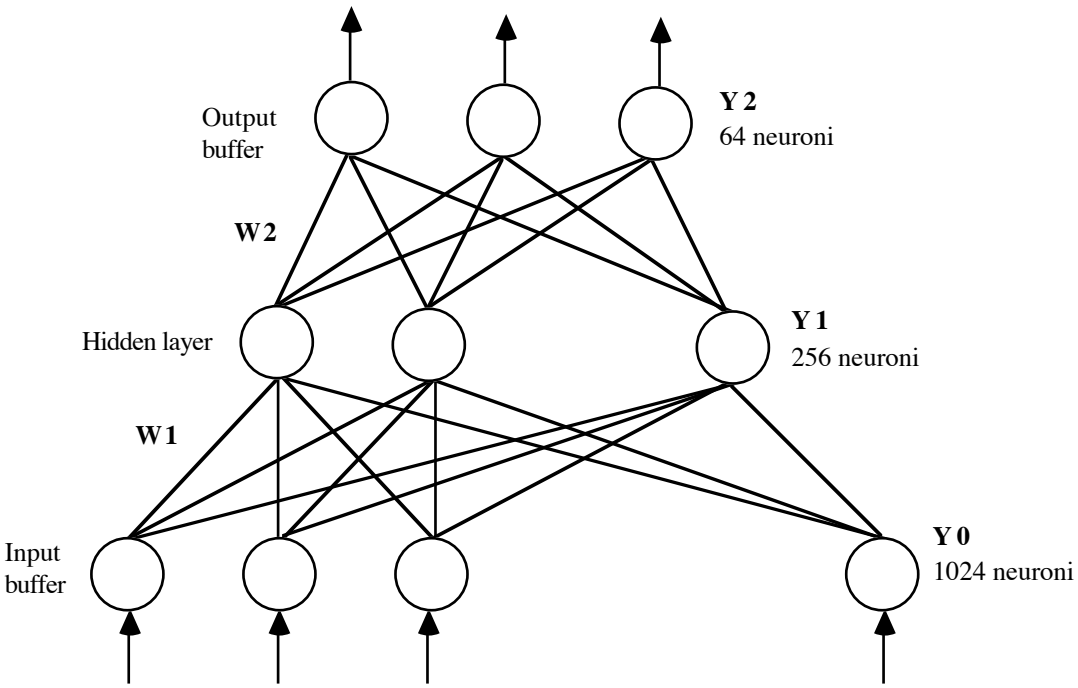

Figura 6. Esempio di applicazione sviluppata su NS1 (fase di produzione) (da [Ra et al. 94])

Progetti di ricerca sul neurocomputer Sinapse-1.

Presso il Laboratorio di Linguistica vengono svolti studi di Fonetica, Psicolinguistica, Semantica e Sintassi. In relazione agli studi di Fonetica numerose sono le esperienze compiute su hardware e software tradizionale che sarebbe interessante confrontare con soluzioni basate su un approccio completamente diverso come quello messo a disposizione da una architettura neurale.

Tale interesse riguarda sia una valutazione comparativa delle prestazioni ottenibili con metodi algoritmici standard su hardware tradizionale, sia un confronto con le soluzioni ottenibili mediante l'uso di simulatori di reti neurali sia su PC (NeuralWorks) sia su Workstation (Stuttgart Neural Network Simulator) sia, infine, la possibilità di implementare architetture di rete di vario tipo e svariate strategie di apprendimento, sia di tipo supervisionato sia di tipo non supervisionato. I settori nei quali ci si propone di effettuare studi ed esperienze sono l'analisi del segnale vocale e l'analisi delle regolarità e il soddisfacimento di vincoli multipli.

Analisi del segnale vocale.

In questo ambito i possibili settori di indagine comprendono il filtraggio, la segmentazione (riconoscimento dei confini di parola e di sillaba, riconoscimento di fonemi) e l'estrazione di parametri di frequenza (F0 e formanti) da flussi di parlato continuo, spontaneo e radiofonico ed anche l'estrazione di parametri caratteristici del parlante (età, sesso, stato emotivo e simili).

Analisi delle regolarità e soddisfacimento di vincoli multipli.

Nella fonologia di lingue quali il Bulgaro e il Francese sono presenti sequenze "anomale", nel senso che non sono facilmente generabili mediante regole, che in passato sono state studiate da diversi punti di vista (diacronico, sincronico, analisi fonologica e analisi acustica) senza tuttavia che i vari punti di vista abbiano mai profondamente interagito fra di loro.

I suddetti punti di vista permettono singolarmente di stabilire regole e vincoli in base a cui tali sequenze possono essere interpretate. L'uso di una rete neurale, in questo ambito, si pensa possa consentire la definizione di un sistema caratterizzato da vincoli multipli che devono essere soddisfatti e dei quali sia possibile verificare il grado di interdipendenza.

In entrambi i settori presenta notevole interesse la possibilità di utilizzare reti neurali, con particolare riferimento alle reti ricorrenti, su cui applicare algoritmi di apprendimento non supervisionato, quali gli algoritmi di clusterizzazione dei dati.

Conclusioni.

Il presente lavoro rappresenta una rielaborazione ed un ampliamento di [Cio95] e contiene molto materiale in forma ancora grezza, testimonianza dell'interesse dell'autore per le reti neurali e le loro applicazioni come modelli computazionali.

Nei programmi futuri dell'autore vi sono sia un'analisi ulteriore dei settori applicativi del Neural Computing, con particolare attenzione alla Linguistica, sia un'analisi approfondita delle topologie di rete e degli algoritmi di apprendimento. Inoltre l'autore si ripropone un esame approfondito e il conseguente utilizzo dei simulatori disponibili presso il Laboratorio di Linguistica (NeuralWorks e Stuttgart Neural Network Simulator) e l'uso dell'NS1, non appena questi sarà disponibile, come ambiente di sviluppo di applicazioni di Neural Computing nei settori applicativi elencati nel paragrafo precedente.

Bibliografia.

[Cio95] Cioni L., "Hardware neuronale: prospettive di utilizzo in linguistica", lavoro presentato alle Quinte Giornate di Studio del Gruppo di Fonetica Sperimentale, Roma 23-24 Novembre 1995.

[FaLe] Fahlman S. E. & C. Lebrere, "The cascade-correlaton learning architecture" in Advansces in Neural Informtion Processing Systems, altri dati mancanti.

[Fre89] Frean M., "The Upstart Algorithm: a method for construct and training feed-forward networks", Edimburgh Physics Department Preprints, 1989.

[FreSka91] Freeman J. A. & D. M. Skapura, *Neural Networks, Algorithms, Applications And Programming Tecniques*, Addison-Wesley, 1991.

[KliGu88] Klimasaukas C. C. & J. P. Guvier, *NeuralWorks, Networks I, II & III, Manuali d'uso di NeuralWorks*, Neural Ware Incorporated, 1988.

[MoSco91] Morgan D. P. & C. L. Scofield, *Neural Networks and Speech Processing*, Kluwen Academic Publisher, 1991.

[Ra et al. 94] Ramacher U., W. Raab, J. Anlauf, U. Hacmann & M. Weßeling, *Sinapse-1, A General Purpose Neurocomputer*, Technical documentation, Siemens Nixdorf, 1994.

[RuMcCle91] Rumelhart D. E. & J. L. Mc Clelland, editors, *PDP Microstruttura dei processi cognitivi*, Edizione parziale in italiano, Serie di Informatica, Il Mulino, 1991.

[SiNa90] Sirat J. A. & J.-P. Nadal, "Neural trees: a new tool for classification", Network 1, 1990.

[Sta94] Starita A., *Computazione Neurale*, Appunti per il corso di Teoria dei Sistemi, Università degli Studi di Pisa, Corso di Laurea in Scienze dell'Informazione, documento manoscritto, Anno Accademico 1994-95.

[Stu94] AAVV, *Stuttgart Neural Network Simulator, User Manual*, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems, 1994.

[Ta94] Tamburini E., *Memorie Associative*, Appunti per il corso di Teoria dei Sistemi, Università degli Studi di Pisa, Corso di Laurea in Scienze dell'Informazione, documento manoscritto, Anno Accademico 1994-95.