RETE_NEURALE.MDB

Esempio di Multi-layer Perceptron

Realizzato con Visual Basic for Application (su base ACCESS)

da Cesare Brizio

Versione del 28/08/01

INDICE

Requisiti di sistema

- Microsoft Access 97 o 2000, trattandosi di file MDB
- Risoluzione video 1024 x 768 punti (consigliata) o 800 x 600 punti (minima): ciò non dipende da utilizzi di grafica ad elevata risoluzione, ma al fatto che la applicazione è stata sviluppata su un monitor 1024x768, e l'aspetto delle maschere video, nonché l'accessibilità dei pulsanti in esse contenuti, possono essere compromessi da una visualizzazione svolta con una risoluzione minore.
- A livello Microsoft Access, devono essere attivati i seguenti RIFERIMENTI:
  - ✓ Visual Basic for Applications
  - ✓ Microsoft Access 8.0 Object Library
  - ✓ Microsoft DAO 3.51 Object Library

Informazioni generali

L'unico obbiettivo di ALLINEAMENTI.MDB è stato di consentirmi di capire, e provare, gli algoritmi alla base delle RETI NEURALI, con particolare riferimento ai PERCETTRONI MULTISTRATO, dotati cioè di almeno un hidden layer.

- La sorgente originale di informazioni è stata una realizzazione in C++ di Piero Fariselli, più o meno correttamente tradotta in Basic e contestualizzata in Access. Relativamente a tale sorgente, trattandosi di codice criptico, non commentato e parametrico, la sua utilità (al di là della doverosa e grata citazione) è stata del tutto marginale.
- Molto più utili i consigli contenuti nel tutorial di Nikos Drakos su questo specifico. Tra i numerosissimi documenti disponibili su web che ho consultato, è l'unico che descriva il procedimento in modo abbastanza intuitivo.
- Risolutive, in ogni caso, le informazioni costantemente fornitemi a voce da Piero Fariselli, senza il quale non avrei portato a termine la cosa.
- La realizzazione è assolutamente artigianale

OBIETTIVI

Porre a disposizione degli utenti del codice sorgente in linguaggio "di alto livello" orientato all'utente per consentire loro di afferrare meglio gli algoritmi di rete neurale.

Come pretesto e test di funzionamento degli algoritmi Application Basic citati, il seguente problema: realizzare una rete neurale in grado di discernere quattro simboli grafici (X, +, O e □) proposti su di una matrice di 10 x 10 pixel. Per la rappresentazione del risultato è disponibile uno strato di output costituito da quattro celle, ognuna delle quali corrisponde ad una diversa soluzione.

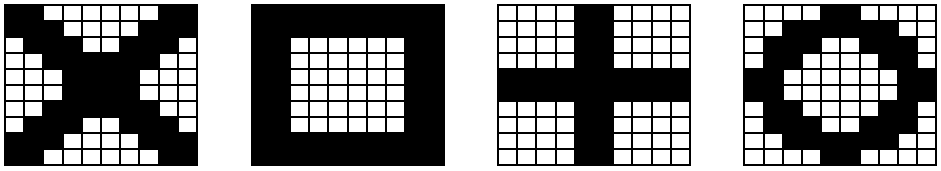
In termini assolutamente descrittivi, si desidera che quando venga memorizzato nelle celle della MATRICE DI INPUT (vedi sotto) un dato simbolo e si avvii la fase di "analisi" (Forward Propagation), la cella della MATRICE DI OUTPUT (vedi sotto) che in fase di addestramento ho "associato" a tale simbolo abbia una "attivazione" (contenga cioè un valore) significativamente più elevato di quello delle altre celle (si veda più avanti "descrizione colloquiale di una rete neurale").

Le reti neurali analizzano matrici ("matrici di input"), per cui è particolarmente facile creare un esempio didattico basato appunto su un riconoscimento di pattern bidimensionali di pixel, rappresentabili da caselle "a 1" e caselle "a 0". Moltissimi esempi didattici vertono su questo tema, che non ho scelto a caso.

Rappresentazione bidimensionale del problema

Il problema intrinsecamente bidimensionale che mi sono posto è quello classico di riconoscimento di simboli (patterns) rappresentati da aree discrete, alla stregua di tasselli di mosaico, che definiamo molto opportunamente PIXEL. Da notare che la rete neurale correttamente addestrata e funzionante riconoscerà anche patterns "sporcati" ad arte tramite il cambiamento dei valori di alcuni dei pixel ad essi associati.

Nella fattispecie, la mia rete neurale dovrà distinguere tra i seguenti patterns, rappresentati nella matrice di input 10 x 10 da valori 1 per le celle "nere" e 0 per le celle "bianche".



Questo problema è semplicissimo da rappresentare su di una matrice di input. Per chi ha già risolto il problema tecnico della rete neurale, si pone il problema di rappresentare su di una matrice bidimensionale ben altri problemi reali non intrinsecamente bidimensionali, ed è qui che entrano in gioco le vere competenze al riguardo. Questo è solo un esempio didattico, il più banale possibile.

Ma come può un problema così semplice mettere in luce le potenzialità delle reti neurali? Per capirlo, basta che proviate a risolvere il problema di distinguere tra di loro le quattro ipotesi su matrice 10 x 10 usando delle banali IF o CASE. La rete neurale proporrà una soluzione IMMEDIATA, al termine di un ciclo di Forward Propagation nell'ambito del quale NON VIENE ESEGUITO ALCUN RAFFRONTO, ma solo una serie di calcoli che hanno come protagonisti celle e collegamenti della rete neurale.

DESCRIZIONE COLLOQUIALE DI UNA RETE NEURALE

Nota: La trattazione teorica delle reti neurali mi è risultata pressoché incomprensibile fatte salve le linee del tutto generali della descrizione del problema. Tecnicamente sono affrontati temi complessi quali la funzione di attivazione del neurone, che (ben lungi dall'essere una discontinuità non lineare del tipo ON/OFF) deve graficamente esprimersi in forma sigmoide: questa funzione sigmoide è influenzata da una serie di

parametri che ne determinano ad esempio la "ripidità" del tratto ascendente, o il fatto se essa sia o meno centrata sullo zero del piano cartesiano. In altri punti il problema è descritto in termini matematici di RICERCA DI UN MINIMO ERRORE, attività nella quale non bisogna farsi fuorviare da minimi locali, ma cercare un minimo assoluto d'errore rappresentabile come un avvallamento o come "attrattore" in un metaforico campo gravitazionale. Utile tecnicamente, e sempre citata nei tutorials, è trattare la questione in termini di un "piano delle soluzioni" da dividere in regioni corrispondenti alle differenti soluzioni da ottenere.

Il grado di astrazione e di tecnicità al quale queste descrizioni si collocano non mi ha consentito (a causa più che altro delle mie specifiche incompetenze) di pervenire in modo immediato ad una sua corretta formalizzazione. Rimandando quindi ai numerosi testi tecnici disponibili per l'inquadramento teorico, preciso comunque che IL CODICE E' ESTENSIVAMENTE COMMENTATO (e comunque comprende le indicazioni originali di Nikos Drakos in particolare per ciò che attiene la fase di addestramento).

Intendo qui proporre una "trattazione", assolutamente informale, di una rete neurale, fatta proprio nel modo in cui avrei gradito vedermela proporre per pervenire rapidamente ad una sua realizzazione tramite un linguaggio di programmazione.

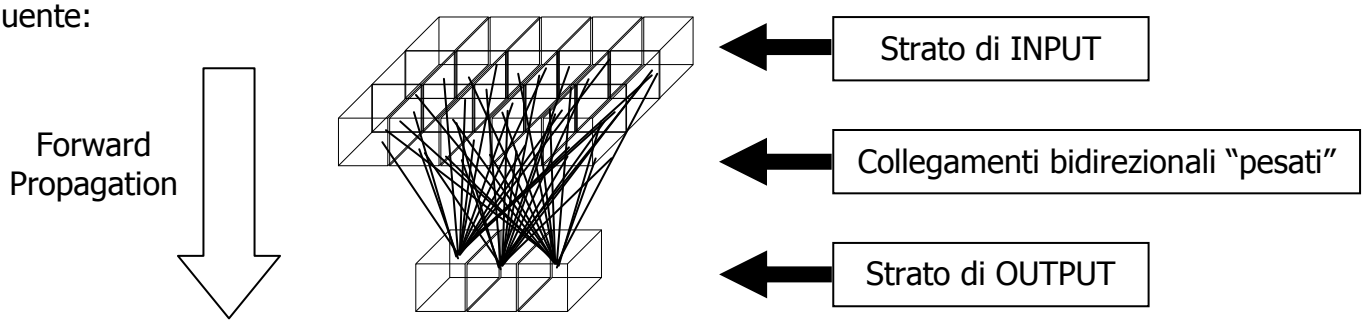
Definiamo "rete neurale" una tecnica di programmazione basata sull'applicazione di algoritmi ispirati da studi sul funzionamento del cervello umano. Tale tecnica ha grandi potenzialità, e NON HA UN CAMPO DI APPLICAZIONE ESCLUSIVO. Quindi, non esistono a priori "problemi da rete neurale" e "problemi irresolubili tramite rete neurale". Esistono semmai livelli di articolazione e di complessità dei problemi tali da rendere eccessivamente macchinosa e dispendiosa (quindi, non pratica) la realizzazione di una rete neurale adatta a risolverli. Vi sono quindi campi nei quali le reti neurali si sono affermate come strumento di soluzione dei problemi.

In realtà il problema deve essere posto in forma tale da potere essere trattato da una rete neurale. Tipicamente, esso deve quindi risultare rappresentabile, ad un qualche livello immediato o convenzionale, in forma di matrice n-dimensionale di valori.

Alcune possibili soluzioni del problema devono essere note a priori: la rete neurale è tipicamente uno strumento per risolvere rapidamente un problema noto, ad esempio la assegnazione di quanto proposto in input ad una categoria predeterminata. Perché ciò sia possibile è necessario disporre di un adeguato numero di esempi afferenti alle differenti categorie in cui il nostro output viene discretizzato (ovvero avere predefinito le possibili risposte e un certo numero di dati in input che corrispondano ALL'INTERO NOVERO DELLE RISPOSTE POSSIBILI). Sarà appunto tramite un "addestramento" basato su tali "esempi" che la rete neurale "apprenderà" come classificare l'input.

Ciò **non** implica che la rete neurale possa assegnare in modo corretto esclusivamente input esattamente corrispondenti ai soli esempi forniti. Anche nel rudimentale RETE_NEURALE.MDB qui fornito, sarà ad esempio evidente come le capacità di riconoscimento di un dato simbolo non sia influenzata dalla variazione di un certo numero di pixel rispetto al caso di riferimento.

Nella più semplice delle ipotesi, una rete neurale è rappresentabile graficamente nel modo seguente:



Come si vede, ogni cella dello strato superiore è collegata ad ogni cella dello strato inferiore. Come nota, si può anche concepire una rete neurale che abbia un UNICO NODO DI OUTPUT. In tal caso la Forward Propagation (vedi sotto) dei diversi input proposti provocherà la comparsa nel nodo di output di valori collocati in differenti ranges (ad esempio, un certo pattern di input sarà rappresentato da un risultato inferiore a 0.25, un secondo pattern sarà rappresentato da valori compresi tra .26 e .5, e così via).

E' importante notare come TUTTI GLI STRATI DELLA RETE NEURALE, COMPRESI GLI STRATI DI COLLEGAMENTI, POSSANO AGEVOLMENTE ESSERE RAPPRESENTATI IN MATRICI ("arrays", "vettori bidimensionali", chiamateli come volete), E QUINDI ESSERE AGEVOLMENTE INIZIALIZZATE, ANALIZZATE, VALORIZZATE TRAMITE ALGORITMI RELATIVAMENTE SEMPLICI IN QUALSIASI LINGUAGGIO DI PROGRAMMAZIONE CHE SUPPORTI LA DEFINIZIONE DI MATRICI.

Da ora in poi, teniamo presente che il "valore di una cella" o il "peso di un collegamento" sono semplici valori numerici memorizzati negli elementi di tali matrici. Sempre riferendomi al disegno, avrò quindi:

- una MATRICE DI INPUT (ad esempio, costituita da N elementi) i cui elementi si definiscono CELLE. Il valore assegnato alle celle dello strato di input rappresenta il problema da risolvere e viene chiamato "valore" o anche "ATTIVAZIONE".
- una MATRICE DI OUTPUT (ad esempio costituita da M elementi). i cui elementi si definiscono CELLE. Il valore assegnato alle celle dello strato di output rappresenta la soluzione proposta dalla rete neurale, ovvero la "classificazione" secondo la rete di quanto proposto in input, e viene chiamato "valore" o anche "ATTIVAZIONE".
- Interposta logicamente tra le due, avremo una MATRICE DI COLLEGAMENTI nell'ambito della quale ogni collegamento è rappresentato dal proprio "peso". Gli elementi della matrice di collegamenti (nell'esempio N * M) sono ovviamente più numerosi, visto che ogni cella di uno strato è collegata a tutte le celle dell'altro strato, e si definiscono COLLEGAMENTI. Il valore in essi memorizzato è il PESO DEL COLLEGAMENTO. Ovviamente il numero di dimensioni delle matrici di collegamento è la somma del numero di dimensioni delle matrici collegate. Ovvero, se ho una matrice bidimensionale di input M_Input(i,j) ed una monodimensionale di output M_Output(k) avrò una matrice tridimensionale di collegamenti M_Collegamenti(i,j,k) nella quale ogni elemento sarà identificato dalla coppia di indici della matrice di input e dal singolo indice della matrice di output.

Naturalmente, parlare di "sopra", "sotto" e "in mezzo" non ha alcun senso assoluto, ma solo descrittivo. Le matrici sin qui descritte possono essere dichiarate ed inizializzate in qualsiasi ordine e tecnicamente sono oggetti del tutto uniformi, salvo che nel dimensionamento e nel contenuto. Non esiste quindi il problema «Come faccio a dire che la tale matrice è "sopra" e la tal altra è "in mezzo" e la tal altra "sotto" ?». Ci si limita a creare delle matrici, a denominarle opportunamente, a dimensionarle secondo il problema da risolvere ed il tipo di rappresentazione scelto per il risultato (più celle per le diverse soluzioni, una sola cella …), e ad utilizzarle secondo quanto qui di seguito sommariamente illustrato.

La rete neurale "funziona" (risolve il problema) nell'ambito di un ciclo di "forward propagation", nel quale i valori dello strato di destinazione (nel disegno, lo strato di OUTPUT) vengono cella per cella calcolati come una funzione del valore memorizzato in tutte le celle dello strato superiore, moltiplicati per il "peso" della relativa connessione verso la cella di destinazione. In altre parole, per ogni cella dello strato di destinazione vengono "spazzate" una per una le celle dello strato superiore, e viene (sto semplificando!) calcolato il cumulo delle somme dei valori in ogni cella moltiplicata per il "peso" assegnato alla connessione tra essa e la mia cella di destinazione.

In questo modo vengono valorizzate le celle dello strato di output. Due cose appaiono ovvie:

1. Il valore risultante nelle celle di output varierà con il variare del contenuto dello strato di input: e difatti voglio appunto risposte specifiche per l'input proposto.
2. Il risultato sarà determinato dal valore ("peso") associato ad ogni singola connessione.

Sono io a determinare quale input proporre alla rete. **Ma come vengono determinati gli importantissimi valori di peso delle connessioni, che ovviamente debbono essere perfettamente calibrati per fornire il risultato desiderato?**

La risposta è semplice: è necessario che, prima di iniziare ad utilizzare la mia rete neurale, io predisponga i corretti valori di peso delle connessioni nella apposita matrice di collegamento, tipicamente caricandoli da un file dove essi erano stati memorizzati in precedenza. Solo una volta che i pesi siano stati inizializzati la mia rete potrà funzionare.

Ma da dove ho ricavato i valori "adatti" a risolvere il mio problema? I pesi "corretti" memorizzati su disco sono frutto di una o più SESSIONI DI ADDESTRAMENTO, svolte utilizzando l'algoritmo di ERROR BACKWARD PROPAGATION, sinteticamente "BACKPROP" (per così dire, il "complementare" di quello di Forward Propagation).

L'addestramento si svolge a partire da un adeguato numero di esempi (RETE_NEURALE.mdb è stato addestrato con di una sequenza di oltre 2500 esempi) da proporre uno dopo l'altro alla rete neurale.

Preliminarmente, memorizzo dei valori CASUALI (tipicamente, nell'intervallo che va da –0.001 a +0.001) come peso di tutte le connessioni. Poi avvio un ciclo di attività, da ripetersi per ogni esempio, che provocherà un continuo aggiustamento dei pesi delle connessioni:

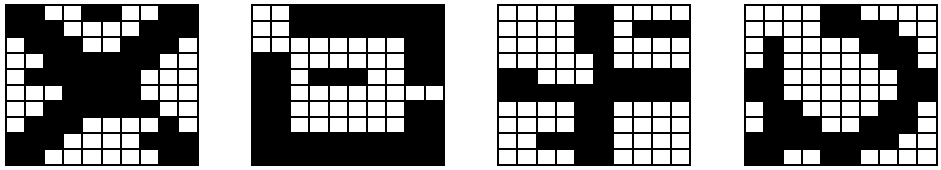
1. Carico l'esempio nella matrice di input, aspettandomi un ben determinato risultato (ad esempio, che una ben precisa cella dello strato di output, che per mia scelta ho predestinato a rappresentare proprio l'esempio che propongo in input, assuma valore 1 e le altre restino a 0)
2. Eseguo la forward propagation, ottenengo cioè la risposta della rete sotto forma di valorizzazione dello strato di output.
3. Calcolo (con appositi algoritmi) gli errori associati alle celle dello strato di output rispetto ai risultati attesi
4. Cella per cella, e connessione per connessione, correggo conseguentemente (con appositi algoritmi) il peso delle connessioni

5. Passo all'esempio successivo

Se la rete neurale è stata scritta correttamente, un monitoraggio degli errori associati alle celle di output rivelerà una progressiva diminuzione dell'errore, e dopo un adeguato numero di esempi la rete inizia a restituire risposte adeguate alle aspettative.

Ben difficilmente, comunque, uno dei nodi di output assumerà il valore "1". Si definisce quindi come "risposta della rete neurale" quella associata al NODO DI OUTPUT CON IL VALORE PIÙ ALTO (logica "winner takes all").

E' bene comunque ricordare che in una sessione di addestramento è opportuno alternare a simboli "puliti" anche simboli "sporchi", ovvero con alcune celle di input non correttamente valorizzate. Ciò è particolarmente importante nel caso in cui la mia rete neurale sia destinata ad un impiego pratico nel quale (ad esempio) i caratteri da riconoscere venissero letti via scanner da un cartellino che potrebbe anche non essere sempre perfettamente pulito. Ad esempio, con riferimento al caso di RETE_NEURALE.mdb, tra gli altri patterns in sede di addestramento sono stati proposti anche casi come i seguenti:



Il caso di RETE_NEURALE.mdb

Pur nella sua elementarità, quella di RETE_NEURALE.mdb rappresenta una architettura ben più complessa di quella sin qui sommariamente esemplificata. Tra l'altro nella descrizione colloquiale sono stati omessi altri elementi più tecnici che comunque è agevole evincere dal codice e – per la parte teorica – dalle numerose pubblicazioni disponibili su Internet.

Ai nostri fini basti sapere che una rete neurale del tipo "Multi-Layer Perceptron" (Percettrone multistrato) dotata di UN HIDDEN LAYER (strato nascosto – ce ne potrebbe essere anche più di uno) è fondamentalmente composta dai seguenti elementi:

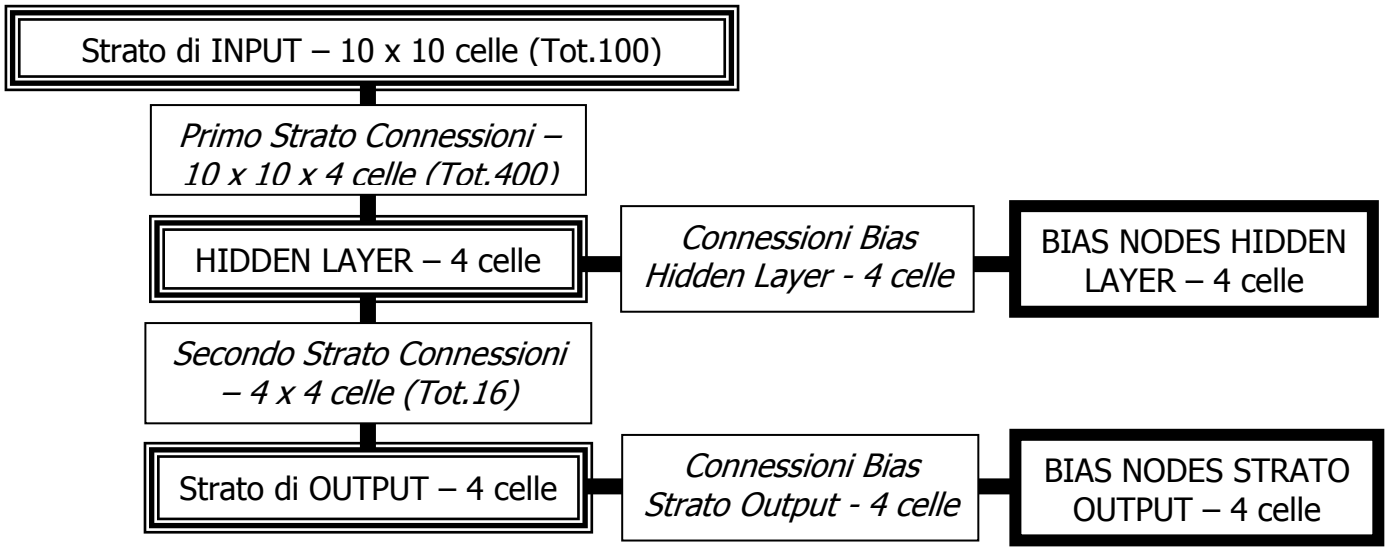
- Uno strato di INPUT (internamente al quale viene rappresentato il problema)
- Uno strato di COLLEGAMENTI tra strato di input e strato nascosto
- Uno STRATO NASCOSTO ("HIDDEN LAYER")
- Uno strato di COLLEGAMENTI tra strato nascosto e strato di output
- Uno strato di OUTPUT (che rappresenta la risposta della rete neurale)

Ad essi si aggiungono, per una maggiore affidabilità dell'assieme, due strati di NODI DI BIAS ("BIAS NODES") dimensionati esattamente come l'Hidden Layer e come lo STRATO DI OUTPUT. La loro funzione tecnica (rimando ai testi specifici) è quella di concorrere al corretto profilo sigmoide della funzione di attivazione del singolo nodo.

Sia durante la fase di addestramento / backward propagation, sia durante la fase di utilizzo standard / forward propagation, il valore di questi nodi sarà posto ad 1. Quindi, oltre a caricare

un pattern da riconoscere nella matrice di input, nell'uso pratico della mia rete neurale dovrò anche preliminarmente porre 1 in ogni "bias node" dei due strati citati.

Ogni Bias Node è connesso in modo diretto ad un solo nodo dello strato corrispettivo. Si può quindi affermare che ogni cella dell'hidden layer ed ogni cella dell'output layer ha il suo bias node specifico. Quindi anche lo strato di connessioni tra uno strato ed i suoi bias nodes ha la esatta dimensione dello strato. In sintesi, la topologia del "nocciolo" della rete neurale in RETE_NEURALE.mdb è come segue:



L'algoritmo di backward propagation richiede inoltre che:

- Siano disponibili i risultati attesi a fronte di ogni esempio: essi saranno ovviamente ospitati in una matrice delle stesse dimensioni dello strato di output
- prima si determinino gli errori associati alle singole celle degli strati hidden ed output, e poi si applichi la opportuna correzione dei pesi delle connessioni: ciò richiede di parcheggiare gli errori in una matrice di dimensionamento identico a quella dello strato di riferimento.
- prima di passare all'esempio successivo da esaminare, siano memorizzati tra un ciclo e l'altro le correzioni applicate ad ogni peso delle connessioni negli strati di collegamenti , perché le correzioni "del giro precedente" concorrono a determinare quelle del giro corrente. Ciò richiede quattro ulteriori matrici.

Riepilogando, in totale sono necessarie le seguenti 16 matrici (proposti i nomi usati in RETE_NEURALE.mdb):

1. M_INPUT, lo strato di input
2. M_Strato1, lo strato di connessioni tra input ed hidden layer
3. M_HIDDEN, l'hidden layer
4. M_Strato2, lo strato di connessioni tra hidden layer e output
5. M_OUTPUT, lo strato di output
6. M_BiasNodes_HIDDEN, la matrice con i BIAS NODES per le celle di HIDDEN LAYER
7. M_BiasNodes_OUTPUT, la matrice con i BIAS NODES per le celle di STRATO OUTPUT
8. M_Strato_HID_BIAS, lo strato di connessioni tra Bias Nodes e Nodi di Hidden Layer
9. M_Strato_OUT_BIAS, lo strato di connessioni tra Bias Nodes e Nodi di Strato Output

10. M_OUTPUT_DESIDERATI, cosa mi aspetto di ottenere in output
11. M_ERRORI_OUTPUT, errori in output rispetto alle mie aspettative
12. M_ERRORI_HIDDEN, errori in hidden layer come calcolati da backward propagation
13. M_ULTIMI_DELTAW_S1, ultimi delta applicati ai pesi strato 1
14. M_ULTIMI_DELTAW_S2, ultimi delta applicati ai pesi strato 1
15. M_ULTIMI_DELTAW_HB, ultimi delta applicati ai pesi tra HIDDEN ed i rispettivi BIAS NODES
16. M_ULTIMI_DELTAW_OB, ultimi delta applicati ai pesi tra OUTPUT ed i rispettivi BIAS NODES

COME SARÀ NOTATO PIÙ SOTTO, SAREBBE ABBASTANZA FACILE "CUMULARE" TUTTE LE MATRICI QUI CITATE IN UN NUMERO MINORE DI MATRICI DI DIMENSIONAMENTO MAGGIORE (al limite estremo, in una unica matrice multidimensionale). Si è scelto di "esplodere" il numero di matrici per la massima leggibilità.

Si rimanda al codice, commentatissimo, per ogni ulteriore chiarimento sull'utilizzo delle matrici citate.

POSIZIONE DEL CODICE

Il codice sorgente è situato in due diversi tipi di contenitore:

MODULI STANDARD

Trattasi di nudi contenitori di codice, semplici "librerie pubbliche di codice sorgente" che non si intende dislocare in un modulo locale per mantenerne visibilità ed utilizzo alla scala dell'intero set delle maschere, dei reports … insomma a livello dell'intero database Access.

MODULI DI CLASSE "LOCALI" DELLE MASCHERE E DEI REPORT

In essi giace il codice specifico delle singole maschere, che ha natura "locale". Nell'ambito di questi moduli locali, sono definibili funzioni associate ad eventi, come anche funzioni sganciate dagli eventi. La prassi è di posizionare nei moduli locali tutto quanto attenga i controlli posizionati sulla maschera video, o sul report.

Nel caso specifico, è stata dislocata nei moduli di classe la parte di algoritmi di allineamento strettamente detta, *ma è stata scritta internamente alla maschera video tutta la parte di backward tracking*.

OTTIMIZZAZIONE DEL CODICE

Tutte le matrici delle quali si è parlato, che anche all'interno del codice sono perfettamente separate e distinte per chiarezza espositiva, DAL PUNTO DI VISTA OPERATIVO POTREBBERO OVVIAMENTE ANCHE ESSERE "COLLASSATE" IN UN NUMERO MINORE DI MATRICI (al limite in una unica matrice di dimensionalità pari a quella del più complesso strato di collegamento), ED

ESSERE GENERATE IN MODO PARAMETRICO. Trattandosi di un esempio basico e didattico, si è sacrificata l'efficienza alla leggibilità.

Allo stesso modo, gli algoritmi di propagazione diretta e inversa sono stati frammentati ben al di là dell'indispensabile (con particolare riferimento al trattamento dei Bias Nodes).

Rispetto a quanto disponibile in letteratura, il codice è quindi stato "espanso", ed in certo modo de-modularizzato alla esclusiva ricerca della MASSIMA LEGGIBILITÀ, il tutto senza pretesa di oggettività ma solo secondo il parere di chi scrive.

La unica miglioria sostanziale rispetto a quanto disponibile in letteratura sta infatti nella pletora di commenti (tra l'altro, vale la pena che un esperto li verifichi per quanto attiene ai commenti interni agli algoritmi!).

Dal punto di vista del debug, tutte le routines e le funzioni sono dotate di error trapping con esplicitazione del codice e descrizione dell'errore.

INPUT/OUTPUT ED ALGORITMO

Una CONSISTENTE PARTE DEL CODICE è destinata a funzioni non connesse all'algoritmo di rete neurale, tra le quali:

- la generazione di patterns standard da proporre alla rete neurale
- il salvataggio ed i caricamento di esempi
- la memorizzazione ed il recupero dei pesi
- la rappresentazione grafica della matrice di input su di una maschera video
- la rappresentazione dei risultati

Nella "guida al codice" che segue, le parti di codice strettamente relative agli algoritmi di rete neurale saranno segnalati con un apposito marcatore nella intestazione e nel titolo.

GUIDA AL CODICE – MODULO STANDARD NEURAL

Modulo Standard	*Sezione*	*Attiene agli algoritmi di rete neurale!!!*
NEURAL	***DICHIARAZIONI***	

Dichiarazione commentata di tutte le matrici coinvolte. Il dimensionamento è effettuato sulla base di costanti qui dichiarate.

Modulo Standard	*Funzione*	*Attiene agli algoritmi di rete neurale!!!*
NEURAL	***AzzeraTrannePesiCollegamenti***	

Azzera tutti gli elementi di tutte le matrici, lasciando inalterati i pesi delle connessioni: necessario per analizzare in sequenza più input. I Bias Nodes restano inalterati ed inizializzati ad 1.

Modulo Standard	*Funzione*	*Attiene agli algoritmi di rete neurale!!!*
NEURAL	***AzzeraTutteLeMatrici***	

Azzera tutti gli elementi di tutte le matrici. Utilizzo tipico: preliminare ad una sessione di addestramento, prima della inizializzazione random dei pesi connessione.

Modulo Standard	*Funzione*	*SUPPORTO*
NEURAL	***FillMatriceInput***	

Pone in "matriceinput", un vettore bidimensionale usato dalle routines di rappresentazione dell'input sullo schermo, il contenuto della matrice passata come parametro.

Modulo Standard	*Funzione*	*USATA A SOLI FINI DI DEBUG*
NEURAL	***LeggiPesoCollegamento***	

Legge dalla tabella "PESO_COLLEGAMENTI" il valore del peso di un ben determinato collegamento, ricevendo come parametri il numero dello strato e gli indici posizionali dei nodi uniti dal collegamento.

Modulo Standard	*Funzione*	*SUPPORTO*
NEURAL	***LeggiTuttiPesiCollegamenti***	

Legge dalla tabella il cui nome viene indicato dall'utente i pesi di tutti i collegamenti esistenti nella rete neurale, e li memorizza nelle opportune posizioni dei corrispettivi vettori. Preliminare all'utilizzo di una rete neurale.

Modulo Standard	*Funzione*	*SUPPORTO*
NEURAL	***PredisponiTabellaPesoCollegamenti***	

- i pesi dei collegamenti vengono
- reinizializzati con valori random compresi nel range passato come parametro
- memorizzati in tabella PESO_COLLEGAMENTI secondo un idoneo sistema di chiavi

Preliminare all'addestramento della rete.

Modulo Standard	*Funzione*	*USATA A SOLI FINI DI DEBUG*
NEURAL	***ScriviPesoCollegamento***	

Scrive in tabella "PESO_COLLEGAMENTI" il valore del peso di un ben determinato collegamento, ricevendo come parametri il numero dello strato e gli indici posizionali dei nodi uniti dal collegamento.

Modulo Standard	*Funzione*	*SUPPORTO*
NEURAL	***ScriviTuttiIPesiCollegamenti***	

Scrive in tabella "PESO_COLLEGAMENTI" il valore del peso di un ben determinato collegamento, ricevendo come parametri il numero dello strato e gli indici posizionali dei nodi uniti dal collegamento. Viene così salvato l'esito di una sessione di addestramento.

GUIDA AL CODICE – MODULO "LOCALE" DI MASCHERA NEURAL

Modulo di Classe Locale	*Sezione*	*SUPPORTO*
NEURAL	***DICHIARAZIONI***	

Dichiarazione commentata di alcune matrici utilizzate per la rappresentazione a video del pattern da riconoscere.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***ColoraCella***	NESSUNO	

Colora una cella del display 10 x 10 su cui sono rappresentati i patterns in input.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***FillMatriceVideoconDisplay***	NESSUNO	

Trasferisce in "matricevideo" quanto rappresentato sul display 10 x 10 della maschera.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***FillMatriceVideoconRigheVideo***	NESSUNO	

Trasferisce in "matricevideo" quanto presente in matrice RIGEHVIDEO: quest'ultima matrice è utilizzata come "transito" dei valori memorizzati nelle tabelle destinate a contenere gli esempi per la rete.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***FillMatriceVideo0*** ***FillMatriceVideoPiu*** ***FillMatriceVideoQuad*** ***FillMatriceVideoX***	NESSUNO	

Trasferiscono in "matricevideo" uno dei quattro pattern standard. Preliminare alla fase di utilizzo della rete neurale

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***LeggiEsempio***	NESSUNO	

Viene letto da tabella ESEMPI un record il cui numero viene passato come parametro, contenente un pattern per il ciclo di addestramento. Invocata in fase di addestramento.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***MostraMatriceVideo***	NESSUNO	

Il contenuto della matrice "MatriceVideo" viene usato per colorare di bianco o di nero i quadratini costituenti il "display" sulla maschera.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***MostraMatriceVideo***	NESSUNO	

Il contenuto della matrice "MatriceVideo" viene usato per colorare di bianco o di nero i quadratini costituenti il "display" sulla maschera.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***PulisciDisplay***	NESSUNO	

Il contenuto della matrice "MatriceVideo" viene inizializzato con tutte caselle bianche. Per predisporre la rappresentazione di un nuovo pattern o esempio.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***SalvaEsempio***	NESSUNO	

Consente di salvare nell'apposita tabella un pattern rappresentato sul "display" 10 x 10 pixel della maschera.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***SvuotaEsiti***	NESSUNO	

Inizializza tutti i controlli destinati a visualizzare l'esito previsto (nel solo caso delle sessioni di addestramento) ed il risultato dell'analisi.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***SvuotaMatriceVideo***	NESSUNO	

Inizializza la matrice di transito "MatriceVideo"

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***Form_Open***	APERTURA MASCHERA	

Inizializza display 10 x 10 e risultati, e mostra avviso espandendo e riposizionando una etichetta di testo.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***Chiudi_Click***	Click su tasto CHIUDI	

Chiude la maschera.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*Attiene agli algoritmi di rete neurale!!!*
NEURAL	***Inizializza_Click***	Click su tasto "Predisposizione e caricamento pesi"	

Invoca le routines di azzeramento di tutte le matrici e caricamento dei pesi da file. Passo preliminare alla pressione del pulsante UTILIZZO.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***Presentazione_Click***	Click su etichetta "Presentazione"	

Fa sparire il testo iniziale di presentazione.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***R1C1_Click / R10C10_Click***	Click su etichetta "RnCn"	

Per ognuno dei "pixel" del "display" questa routine passa il pixel da bianco a nero e viceversa.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***SalvaE2_Click***	Click su pulsante "Salva Esempio "	

Salva all'interno del file ESEMPI, in formato di esempio, quanto visualizzato sul display.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*SUPPORTO*
NEURAL	***SelectPattern_Click***	Click su gruppo di opzioni "Carica pattern su Video"	

Viene invocata la routine di caricamento sul display del simbolo relativo alla opzione scelta .

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*Attiene agli algoritmi di rete neurale!!!*
NEURAL	***Training_Click***	Click su pulsante "ADDESTRAMENTO"	

Sessione di addestramento basata su set di esempi scelto dall'utente tra quelli memorizzati nelle apposite tabelle. Gli ultimi dieci cicli del set di training vengono fermati per dare il tempo di esaminare i valori ottenuti prima di passare al ciclo successivo. I cicli precedenti di addestramento si susseguono su video senza richiesta di input di verifica da parte dell'utente. Il messaggio di fine addestramento è ribadito più volte, e viene offerta la opzione di salvare il risultato dell'addestramento in una apposita tabella a scelta dell'utente tra quelle già esistenti.

Modulo di Classe Locale	*Funzione*	*Agganciato a evento …*	*Attiene agli algoritmi di rete neurale!!!*
NEURAL	***UTILIZZO_Click***	Click su pulsante "UTILIZZO"	

Forward Propagation singola: ricorda che ci si attende la avvenuta predisposizione di un pattern da riconoscere (tipicamente, tramite click su gruppo di opzioni "SelectPattern") ed il caricamento di opportuni pesi tramite click su pulsante PREDISPOSIZIONE E CARICAMENTO PESI.

Cesare Brizio, 28 Agosto 2001