Tesi di Diploma di Laurea in Informatica di

Pietro Mele

matricola 524304

Analisi e Sviluppo di una Rete Neurale Modulare basata su “Mixture of Experts”, e Confronto con Algoritmi di Boosting

Relatore: Prof. Alberto Bertoni

Correlatori: Prof. Nicolò Cesa-Bianchi

Prof.ssa Paola Campadelli

Facoltà di Scienze dell'Informazione

Università degli Studi di Milano

a.a. 2000/2001

Indice

Appendici:

A. Software utilizzato.

B. Introduzione a UML.

C. Note riguardo lo Standard C++.

Bibliografia.

Ringraziamenti

La persona che più devo ringraziare, non solo per questo lavoro, è il Dottor Albino Bricolo (Ospedale Civile Maggiore, Verona), senza il cui inconsapevole contributo non mi sarei mai interessato a questi argomenti, e senza le cui capacità da tempo non potrei più camminare, sognare, vivere. E' a lui che dedico questa mia tesi.

In caso voleste contattarmi in futuro, sarò rintracciabile al seguente indirizzo di posta elettronica:

pietromele@yahoo.com

1. Introduzione

In questa tesi descriverò un tipo particolare di rete neurale, noto con il nome di «Mixture of Experts», traducibile in italiano con l'espressione «mistura di esperti»; d'ora in poi utilizzerò il termine inglese poiché è quello diffuso in letteratura.
Tratterò questo modello di rete dal punto di vista teorico, ne descriverò una implementazione reale, e farò un confronto con altri algoritmi, in un certo senso simili, che si pongono gli stessi obbiettivi.
In particolare, nel capitolo 2 farò una introduzione al concetto di algoritmo di apprendimento da un punto di vista dapprima generico e successivamente focalizzato sulle reti neurali. Nel capitolo 3 descriverò i concetti alla base di una rete neurale modulare. Nel capitolo 4 farò un confronto con un algoritmo specifico, detto di "Boosting". Nel capitolo 5 farò, per quanto possibile, un parallelismo tra il modello che tratto ed il sistema nervoso centrale. Nel capitolo 6 descriverò in modo analitico il funzionamento di un tipo particolare di rete modulare, la cosiddetta architettura "Mixture of Experts". Nel capitolo 7 farò una descrizione di una possibile implementazione tramite diagrammi UML, che ne permetteranno una visione tramite un linguaggio grafico. Nel capitolo 8 tratterò l'implementazione che sto realizzando tramite il linguaggio di programmazione Standard C++. Nei due capitoli successivi tratterò di due possibili estensioni all'algoritmo base che permetterebbero una efficace pre-elaborazione dell'ingresso tramite wavelets, e uno sviluppo autonomo dell'architettura della rete utilizzando algoritmi genetici.

2. Introduzione agli algoritmi di apprendimento e alle reti neurali

Un algoritmo o, più in generale, un sistema che apprende può essere definito come una struttura che si auto modifica in funzione di un obbiettivo da raggiungere e di un insieme di condizioni al contorno, cioè dell'ambiente che lo circonda.
Per realizzare un tale algoritmo si possono seguire approcci molto diversi, che però, almeno finora, si sono sempre ispirati al modo in cui noi esseri umani ragioniamo. La differenza fondamentale tra queste diverse strade per affrontare il problema sta nel punto di vista che adottiamo per osservare il funzionamento della mente umana. Questi punti di vista si possono raggruppare in due categorie: una visione dall'alto verso il basso, ed una visione dal basso verso l'alto.
Nella visione dall'alto ci ispiriamo al modo in cui ragioniamo quotidianamente, o per lo meno a ciò che possiamo percepire di tale ragionamento, e cerchiamo di ricostruirlo attraverso un insieme di regole. In questo caso il termine "regole" va inteso nel senso più ampio: possono essere regole di tipo logico, statistico, "fuzzy", influenzate parzialmente dal caso, …
Nella visione del problema dal basso, l'ispirazione viene dal funzionamento del sistema nervoso, dal modo in cui i neuroni si sviluppano e si scambiano informazioni, e dalle modalità seguite dalle specie viventi per evolversi, cioè dalla genetica.
Mentre seguendo il primo approccio le regole da apprendere per raggiungere l'obbiettivo vengono "manipolate" direttamente, esplicitamente o implicitamente, nel secondo caso esse "emergono" come una conseguenza di ciò che avviene su una scala molto ridotta per un gran numero di elementi.

Teoricamente, entrambe queste tecniche dovrebbero poter raggiungere gli stessi risultati, anche se da punti di partenza opposti. In realtà ogni metodologia si esprime meglio per alcune classi di problemi e peggio per altre. Ad esempio i metodi del primo tipo sono adatti a risolvere problemi esprimibili "secondo una logica" ben definita (ad esempio la dimostrazione di un

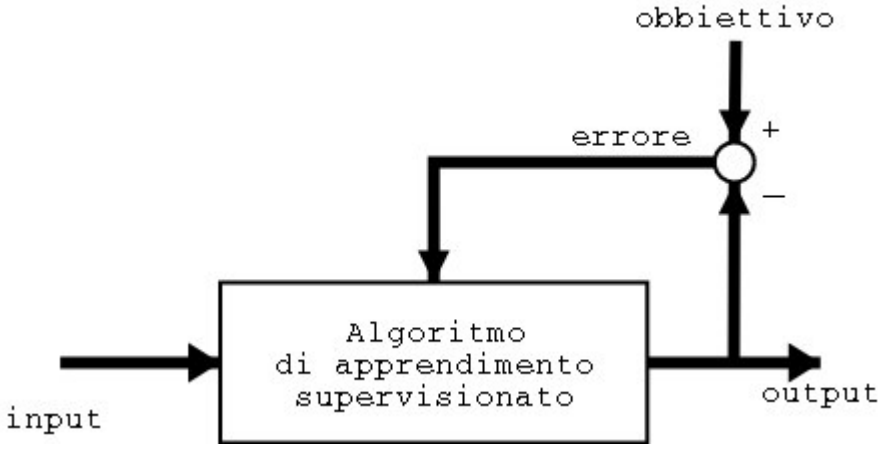
teorema), mentre i secondi si prestano a trattare situazioni dove noi useremmo "l'istinto", poiché non inquadrabili in uno schema rigido (ad esempio l'interpretazione di un'immagine o di un segnale in genere). Si potrebbe dire che l'approccio che parte dall'alto simula meglio l'emisfero sinistro del cervello, mentre l'approccio che parte dal basso simulerebbe meglio l'emisfero destro [Kandel et al., 1991]. Questa differenziazione, tuttavia, è solo approssimativa, in quanto alcune funzioni (prime tra tutte l'elaborazione delle immagini visive) vengono svolte in maniera simmetrica, o quasi, dai due emisferi.

Un generico algoritmo di apprendimento può essere schematizzato come un modulo che riceve delle informazioni in ingresso e ne fornisce delle altre in uscita, e che deve poter stabilire una qualche relazione tra le prime e le seconde sulla base di un qualche criterio.

In particolare i tipi di apprendimento che vengono realizzati possono essere suddivisi in tre gruppi che si distinguono in base alle informazioni che vengono fornite riguardo l'errore commesso: apprendimento supervisionato, apprendimento con rinforzo e apprendimento non supervisionato.

Apprendimento supervisionato (*Supervised Learning*)

In questa situazione l'algoritmo, oltre a ricevere un ingresso e a fornire un'uscita, riceve anche l'uscita corretta (obbiettivo) che avrebbe dovuto generare, e di conseguenza l'errore commesso da ogni elemento del vettore di uscita.



Lo scopo è quello di ricavare una funzione *F* che permetta di associare un vettore di uscita **y** ad un vettore di ingresso **x**:

$$\mathbf{y} = F(\mathbf{x})$$

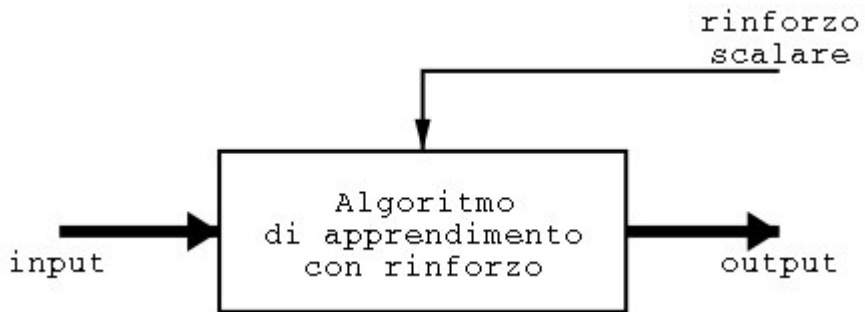
sulla base di un insieme di esempi $\{(\mathbf{x}_1;\mathbf{y}_1),...,(\mathbf{x}_n;\mathbf{y}_n)\}$, in modo da minimizzare una qualche funzione di errore dei valori di uscita, ad esempio:

$$E = \sum_t \left\| \mathbf{y} - \mathbf{y}_t \right\|^2$$

Quindi si può dire che l'apprendimento supervisionato modifica dei parametri interni sulla base di una correlazione tra l'errore in uscita e l'ingresso presinaptico.

Apprendimento con rinforzo (*Reinforcement Learning*)

In questo caso l'unica informazione che viene fornita all'algoritmo in risposta alla sua uscita è uno scalare che indica la "distanza" dell'output generato dall'output corretto.



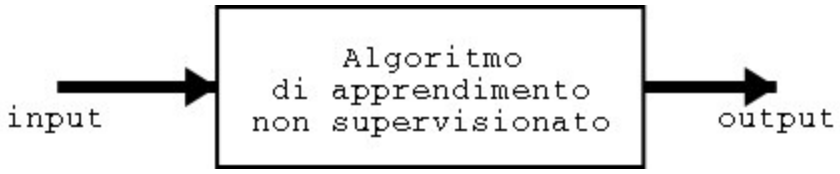
L'obbiettivo, anche in questo caso, è quello di ricavare una funzione *F* che permetta di associare un vettore di uscita **y** ad un vettore di ingresso **x**:

$$\mathbf{y} = F(\mathbf{x})$$

avendo come segnale di risposta uno scalare *r* che indica la correttezza della risposta data. La funzione *F* viene trovata facendo in modo che, per gli ingressi dati, la somma dei valori restituiti venga massimizzata.

Apprendimento non supervisionato (*Unsupervised* o *Self-organized Learning*)

Con l'apprendimento non supervisionato non si ha nessuna informazione riguardo la correttezza dell'uscita prodotta. Un algoritmo di questo tipo deve raggiungere un obbiettivo predeterminato e indipendente dagli ingressi forniti, utilizzando il quale può modificare i suoi parametri interni. Un simile tipo di apprendimento può essere utilizzato nella compressione di segnali, nell'estrazione di particolari caratteristiche da insiemi di dati, nel *clustering*, ...



Come descritto più avanti, nel cervello umano vengono realizzate tutte e tre queste forme di apprendimento in regioni specializzate. Anche una rete neurale artificiale può fare altrettanto, a seconda degli algoritmi impiegati nella fase di apprendimento; in particolare, in una rete neurale modulare questi diversi modelli di apprendimento coesistono e si influenzano l'un l'altro.

3. Descrizione generale di una rete neurale modulare

Generalmente con il termine "modulo" si intende una parte di un sistema più complesso che può essere replicata un numero indefinito di volte e che, di solito, si presenta come una "scatola nera" della quale si conoscono gli ingressi e le uscite, ma non il suo funzionamento interno. Quando la complessità aumenta, questi moduli possono essere di tipo diverso, possono coesistere su uno stesso "livello logico" o possono essere nidificati gli uni dentro gli altri formando una struttura gerarchica.

Le reti neurali viste finora possono essere considerate come modulari a livello dei loro neuroni e dei loro strati. Un neurone, ad esempio, può essere assimilato ad un modulo in quanto può essere visto come un'entità che riceve dei segnali in ingresso dalle sue sinapsi e trasmette un segnale in uscita attraverso il suo assone. Il meccanismo di generazione di questo output sulla base degli input ricevuti può essere nascosto all'interno del neurone stesso, può cambiare in funzione del tempo ed in funzione del contesto, può essere diverso per tipi differenti di neuroni, e così via. In una determinata situazione, quindi, può essere utile considerare un neurone come un'entità astratta di cui non interessano i dettagli di funzionamento, che magari risultano fondamentali partendo da un altro punto di vista.
Lo stesso discorso può essere fatto per gli strati di neuroni che, ad esempio, possono avere connessioni intra-strato con topologie differenti.

Per una rete modulare, questo concetto di modulo viene ulteriormente portato avanti (ed è da qui che ne deriva il nome). Ognuna di queste unità racchiude una o più reti complete ed "autosufficienti", che possono essere viste come scatole nere di livello più alto, con una struttura interna qualsiasi e che interagiscono con ciò che è a loro esterno tramite una interfaccia ben definita. In questo contesto il modulo viene definito come un "esperto" ("expert") che da la sua risposta all'ingresso corrente.

Generalizzando questo discorso, cosa che verrà fatta più avanti, si può svincolare un modulo dal contenere una rete neurale, e lo si può rendere un "contenitore" generico al cui interno vi può essere un algoritmo qualsiasi. Vi sarà poi un meccanismo, ad esempio di "competizione" tra i moduli, a determinare una selezione degli algoritmi che meglio si prestano alla soluzione del problema in esame; quindi il problema iniziale verrà suddiviso in sotto-problemi, ciascuno dei quali sarà assegnato al sottoinsieme dei moduli che si sono dimostrati migliori durante la fase di addestramento.

Una definizione che è stata data di rete neurale modulare è la seguente [Osherson et al., 1990]:
"Una rete neurale è detta modulare se l'elaborazione svolta dalla rete può essere suddivisa tra due o più moduli (sottosistemi) che operano senza comunicare l'un l'altro. Le uscite dei moduli sono combinate da una unità di integrazione alla quale non è possibile mandare informazioni di ritorno ai moduli stessi. In particolare, l'unità di integrazione (1) decide come le uscite dei moduli devono essere combinate per formare l'uscita finale del sistema, e (2) decide quali moduli devono apprendere quale esempio del training set."

Prendendo questa definizione come punto di partenza, è possibile sviluppare strategie differenti per ottenere un algoritmo che realizzi un simile sistema.
Una prima fondamentale distinzione può essere fatta in base alla modalità con cui vengono combinate le uscite dei singoli moduli. Un meccanismo possibile è quello di calcolare dei valori che vadano a pesarne opportunamente le uscite. Questi valori potranno essere derivati in modi diversi, dando luogo alle due seguenti categorie di algoritmi:

- *Algoritmi statici*: le uscite dei moduli vengono combinate in maniera *indipendente* dagli ingressi.
- *Algoritmi dinamici*: le uscite dei moduli vengono combinate in maniera *dipendente* dagli ingressi.

Gli algoritmi statici possono, a loro volta, essere suddivisi in più tipologie differenti, due delle quali, trattate in questa tesi, sono:

- *Ensemble averaging*: le uscite dei moduli sono combinate linearmente.

- *Boosting*: un algoritmo di apprendimento debole viene convertito in uno che raggiunge un arbitrario grado di accuratezza (descritto in un capitolo successivo).

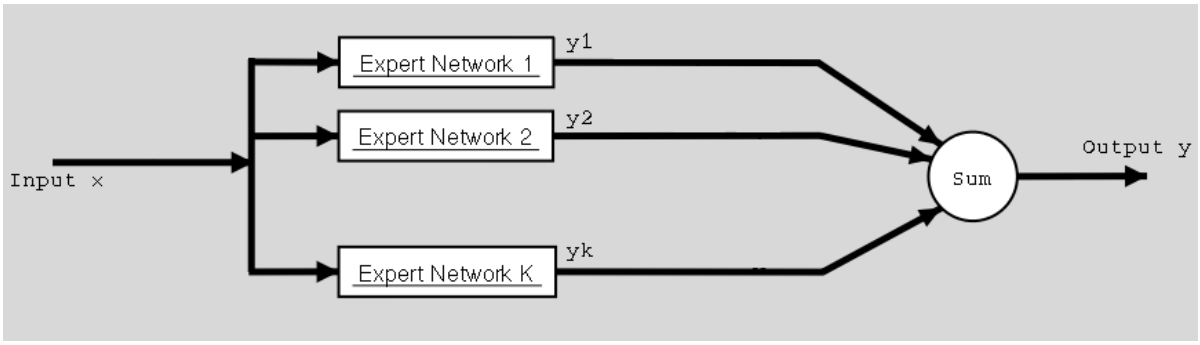
Gli algoritmi dinamici presi qui in considerazione sono le cosiddette "*mixtures of experts*".

Ensemble Averaging

Questo algoritmo è il più semplice tra quelli che realizzano una rete neurale modulare; può essere tuttavia considerato come un primo approccio dal quale si sviluppano i successivi tramite l'aggiunta di dettagli specifici.

Come si vede dalla figura, abbiamo un certo numero di moduli, chiamati ***expert networks***, che hanno un ingresso comune, che non si scambiano informazioni tra loro e che, tramite un meccanismo interno che può essere ignoto, generano un'uscita. Successivamente, le uscite dei singoli moduli vengono combinate secondo una certa modalità per generare l'uscita della rete complessiva.

L'ingresso e l'uscita sono dei vettori in spazi n_i e n_o dimensionali, rispettivamente.



Il motivo per cui può risultare conveniente l'utilizzo di più esperti anziché di un'unica rete neurale dipende dal fatto che (1) il numero di parametri (pesi sinaptici, ad esempio) da trattare può essere inferiore e (2) ogni esperto convergerà, in genere, in minimi locali differenti sulla superficie che rappresenta l'errore, migliorando l'uscita complessiva come una qualche combinazione delle uscite dei singoli esperti.

Il punto (1) può essere mostrato come segue. Supponiamo di avere una rete con n ingressi ed m uscite. In una rete neurale tradizionale, ad esempio del tipo a retropropagazione dell'errore totalmente connessa, avremmo, considerando la presenza di uno strato nascosto di p unità, un numero di pesi sinaptici pari a:

$$N = np + pm = p(n + m)$$

Supponendo il numero di neuroni nello strato nascosto uguale alla media di quelli presenti negli strati di ingresso e di uscita, si ha:

$$N = n\frac{n+m}{2} + \frac{n+m}{2}m = \frac{n+m}{2}(n+m) = \frac{1}{2}(n+m)^2 = \frac{1}{2}(n^2 + 2nm + m^2)$$

da cui si vede come il numero di sinapsi sia proporzionale al quadrato della somma del numero di neuroni negli strati di ingresso e di uscita.

Per fare un paragone, consideriamo ora una rete modulare costituita da K esperti, ognuno contenente una rete neurale senza strati nascosti. Il numero di sinapsi sarà ora pari a:

$$N' = Knm$$

Ora avremo che $N' \leq N$ quando:

$$Knm \leq \frac{1}{2}(n^2 + 2nm + m^2)$$

e quindi per valori di K che soddisfano la condizione:

$$K \leq \frac{n^2 + 2nm + m^2}{2nm} = \frac{n}{2m} + \frac{m}{2n} + 1$$

Da questa disequazione si ha, considerando che stiamo trattando con numeri interi, che per $n=m$ otteniamo il valore minimo del limite superiore $K \leq 2$. Per valori di n ed m diversi, e portando questa differenza all'estremo ponendo $m=1$, si ha:

$$K \leq \frac{n}{2} + \frac{1}{2n} + 1 = \frac{n}{2} + 1$$

Quindi, a seconda del numero degli ingressi e delle uscite presenti, possiamo utilizzare un numero di moduli K compreso nell'intervallo $[2, n/2+1]$ se vogliamo avere un numero di pesi sinaptici inferiore rispetto ad un modello a retropropagazione dell'errore.

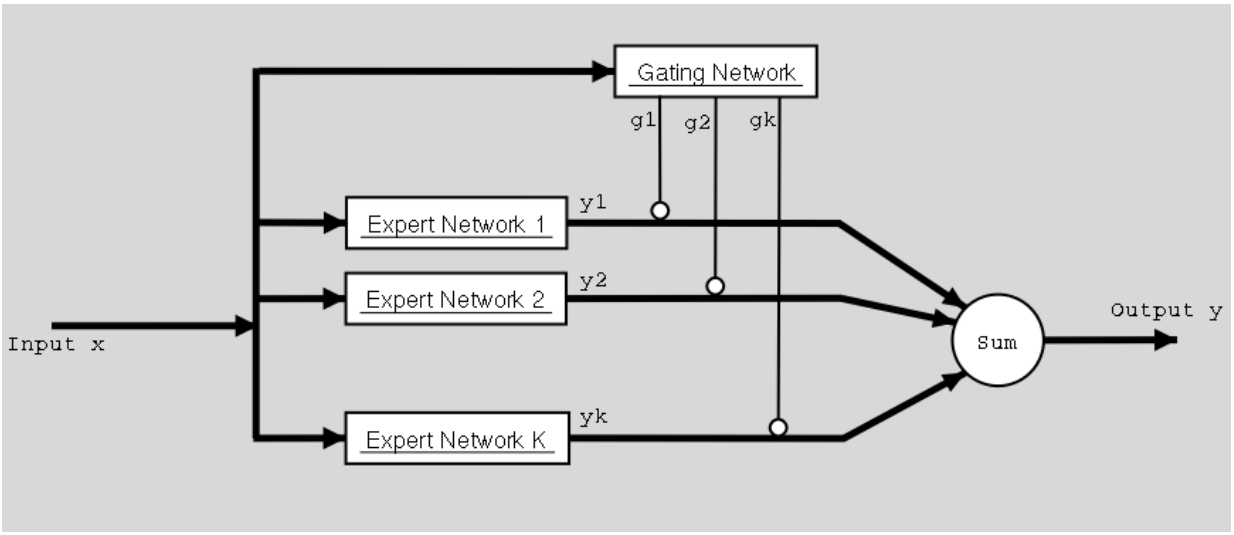
Considerando che K è relativamente piccolo, si può notare che $N'<N$. Più avanti si vedrà come una rete con una simile configurazione sia in grado di apprendere ciò che apprende una rete a retropropagazione dell'errore con strati nascosti.

Il principale svantaggio di questo modello è che ogni modulo che compone la rete non ha la possibilità di differenziarsi dagli altri in nessun modo, in quanto non gli viene fornita nessuna informazione riguardo il comportamento delle altre expert networks. Ciò non vuol dire che dopo la fase di addestramento tutti i moduli saranno identici; piuttosto ogni modulo modificherà i suoi parametri senza essere a conoscenza del contesto in cui opera. Così facendo si avrà ridondanza nell'apprendimento di particolari esempi del training set, e per altri esempi potrà non esserci alcun modulo che converga. Inoltre, poiché l'uscita della rete non è che la media aritmetica delle uscite dei moduli, non si ha la possibilità di selezionare queste ultime in base alla loro correttezza, ma anzi le uscite più lontane da quella voluta andranno a "contaminare" quelle che vi si avvicinano di più.

Di seguito vedremo come risolvere questo problema con l'introduzione di un ulteriore modulo che si occuperà di mediare secondo un qualche criterio le uscite delle expert networks.

Mixture of Experts

Come scritto in precedenza, questo modello può essere fatto derivare dall'*ensemble averaging* in quanto la struttura base della rete è sostanzialmente la stessa, con l'aggiunta di un ulteriore modulo, chiamato ***gating network*** (o *unità di integrazione* secondo la definizione data da Osherson), che controlla dall'esterno le uscite delle expert networks. Anche questo modulo, al pari degli altri, può essere considerato come una scatola nera, che ha come ingresso lo stesso vettore a cui accedono gli altri moduli della rete, e come uscita un vettore i cui elementi possono essere visti come coefficienti che vanno a pesare le uscite delle expert networks.



La gating network può operare in modalità differenti, a seconda di ciò che si vuole ottenere. Volendo considerare gli estremi opposti del suo comportamento, può generare un vettore di uscita i cui elementi siano sempre uguali ad un valore costante uguale a 1/*k*, dove *k* è il numero di expert networks da controllare. In questa modalità la rete si comporta esattamente come il modello ensemble averaging. All'estremo opposto, la gating network può, in funzione dell'ingresso corrente, selezionare una sola expert network e considerare la sua uscita come l'uscita della rete complessiva. In questo caso otteniamo un algoritmo del tipo "winner-takes-all".

Un comportamento intermedio può essere quello di ottenere l'uscita della rete tramite una somma pesata delle uscite di tutte le expert networks, dove i pesi dipendono dall'ingresso corrente.

In entrambi questi due ultimi casi, la gating network non solo varia le proprie uscite a seconda dell'ingresso corrente, ma modifica anch'essa il suo comportamento attraverso una fase di addestramento, tramite il quale può apprendere il criterio secondo cui controllare le uscite degli altri moduli.

Un altro compito fondamentale della gating network è quello di controllare i tassi di apprendimento delle expert networks, sempre relativamente all'ingresso corrente (continuo a ripetere questa nota riguardo l'ingresso corrente, in quanto è di fondamentale importanza per questo genere di algoritmo; naturalmente non è una condizione necessaria, ma in sua assenza si ottiene un algoritmo completamente diverso).

Si può quindi dire che la gating network *apprende* la modalità secondo la quale le expert networks devono *apprendere*, che in ambedue le fasi si utilizza lo stesso training set, e che questi processi possono/devono essere eseguiti in parallelo.
Analogamente a quanto avviene nel modello *ensemble averaging*, anche in questo caso le expert networks non si scambiano informazioni tra loro, e l'unico modo in cui possono indirettamente interagire è tramite la funzione svolta dalla gating network.
Inoltre, grazie a questo particolare modulo, vengono superati i problemi connessi al precedente tipo di rete neurale. Ora le expert networks hanno una qualche informazione che permette loro di differenziarsi le une dalle altre, quindi la ridondanza sarà inferiore e ogni modulo potrà specializzarsi su particolari esempi del training set.

Analizzando la fase di apprendimento da un altro punto di vista, si hanno le due seguenti modalità:

- apprendimento supervisionato: possono essere fornite le uscite corrette alle expert networks;

- apprendimento non supervisionato: le expert networks competono nel fornire una risposta e nell'apprendere il corrispondente esempio, attraverso l'influenza della gating network. Quindi, terminata la fase di addestramento della rete, si ha uno spazio di ingresso partizionato in modo tale che ogni expert network modelli bene il sottospazio assegnatole.

Questo partizionamento permette di rappresentare relazioni ingresso-uscita discontinue in modo naturale; se si vuole approssimare una "superficie" di grado n, si possono utilizzare più superfici di grado m, dove $m<n$. Le expert networks possono quindi essere relativamente semplici, in quanto devono apprendere solo "pezzi" della funzione obbiettivo.

4. Raffronto con algoritmi di "Boosting"

Il “boosting” è un metodo statico, che però si differenzia notevolmente dall’ensemble averaging che pure si colloca nella stessa classe di algoritmi. Infatti, mentre nella seconda categoria tutti gli esperti vengono addestrati sullo stesso insieme di esempi, nel boosting gli esperti sono addestrati su insiemi di dati che possono essere completamente distinti.
Il boosting può essere anche visto come una metodologia generale per migliorare le prestazioni di un *qualunque* algoritmo di apprendimento.

Esistono fondamentalmente tre modi per implementare il boosting:

1. *Boosting by filtering (Boosting attraverso filtraggio)*: gli esempi per l’addestramento vengono filtrati da differenti versioni di un algoritmo di apprendimento debole; viene assunta la disponibilità di un numero di esempi tendente all’infinito, che vengono scartati o meno durante l’addestramento.
2. *Boosting by subsampling (Boosting attraverso “ripescaggio”)*: viene utilizzato un numero di esempi fissato in partenza; questi vengono “riesaminati” durante l’addestramento secondo una certa distribuzione di probabilità.
3. *Boosting by reweighting (Boosting attraverso ricalcolo dei pesi)*: analogo al precedente; in questo caso però l’algoritmo di apprendimento debole riceve degli esempi *pesati*, e l’errore viene calcolato tenendo conto di questo peso.

Di seguito verrà trattato il boosting by filtering [Schapire, 1990].

Accenniamo innanzi tutto al concetto di apprendimento *PAC* (*Probably Approximately Correct*). Viene in questo contesto definita la nozione di concetto (*concept*) come una funzione Booleana che ha come variabile indipendente una codifica di un oggetto. Nell’apprendimento PAC una entità prova ad identificare un concetto binario sconosciuto sulla base di una serie di

esempi scelti a caso riguardo quel concetto.
L'obbiettivo di un sistema che apprende è quello di trovare una ipotesi con un tasso di errore minore o uguale ad ε, dove ε è un valore positivo arbitrariamente piccolo, in modo che l'ipotesi sia valida per tutti gli ingressi possibili con una probabilità pari a 1-δ, dove δ è un piccolo numero positivo. Queste caratteristiche fanno dell'apprendimento PAC un "modello di apprendimento forte" (*strong learning model*).
Esiste una variante del modello PAC, chiamata "modello di apprendimento debole" (*weak learning model*), dove i requisiti sono molto meno stringenti. Il sistema deve ora trovare un'ipotesi con un tasso di errore solo leggermente inferiore ad 1/2, che corrisponde ad una scelta casuale. Schapire ha dimostrato che se è possibile apprendere un concetto in modo debole, allora è possibile apprendere lo stesso concetto in modo forte, rendendo equivalenti i due tipi di apprendimento.

Nel boosting by filtering sono presenti tre expert networks, addestrate, appunto, da un algoritmo di boosting. Chiamando questi esperti "A", "B" e "C", questo è l'algoritmo che li controlla:

1. L'esperto A viene addestrato su un primo insieme di N_1 esempi.
2. L'esperto A, ora addestrato, viene usato per filtrare un secondo insieme di esempi nel seguente modo:
   - Scegli in modo *casuale* una delle due seguenti alternative:
     i. Passa i nuovi esempi attraverso l'esperto A e scarta gli esempi classificati correttamente finché un esempio non viene classificato in modo errato. Aggiungi quest'ultimo esempio al training set dell'esperto B.
     ii. Fai l'opposto. Passa i nuovi esempi attraverso l'esperto A e scarta gli esempi classificati in modo errato finché un esempio non viene classificato correttamente. Aggiungi quest'ultimo esempio al training set dell'esperto B.

   Continua questo processo finché un totale di N_1 esempi sono stati filtrati dall'esperto A. Questo insieme di esempi selezionati costituisce il training set dell'esperto B.

   Seguendo questa procedura, si ottiene che, se all'esperto A viene passato il secondo

insieme di esempi, si otterrebbe un errore pari ad 1/2. Quindi il secondo insieme di N_1 esempi usato per addestrare l'esperto B ha una distribuzione completamente differente da quella del primo insieme di N_1 esempi utilizzato per l'esperto A.

3. Ora che anche l'esperto B è stato addestrato, un terzo insieme di esempi viene prodotto per l'esperto C nel modo seguente:
   - Passa un nuovo esempio attraverso gli esperti A e B; se questi danno la stessa risposta, scarta l'esempio. Altrimenti aggiungi l'esempio al training set dell'esperto C.
   - Continua finché nel training set dell'esperto C non si trovano N_1 esempi.

L'addestramento del sistema è ora terminato.

Una caratteristica da notare di questo algoritmo è che, pur necessitando di un gran numero di esempi per poter operare, in realtà utilizza solo un sottoinsieme di questi dati per l'addestramento. Inoltre, grazie al meccanismo di selezione degli esempi da apprendere, l'esperto B si limita ad apprendere i "dettagli" che sfuggono all'esperto A, mentre l'esperto C apprende ciò che sfugge contemporaneamente ad A e a B.

Supponendo che i tre esperti abbiano un tasso di errore $\varepsilon<1/2$ relativamente alla distribuzione per la quale sono stati addestrati, si ha che l'errore globale g è pari al massimo a:

$$g(\varepsilon) = 3\varepsilon^2 - 2\varepsilon^3$$

Considerando che l'algoritmo di boosting può essere applicato ricorsivamente, si ha che l'errore globale può essere ridotto arbitrariamente. Da ciò deriva che un algoritmo di apprendimento debole, che si comporta solo leggermente meglio di una scelta casuale, viene convertito in un algoritmo di apprendimento forte.

5. Parallelismi con il sistema nervoso centrale

Il sistema nervoso centrale, da cui si traggono spesso spunti in questo settore della ricerca, può essere visto come una struttura altamente modulare, a cui si "sovrappone" una organizzazione di tipo gerarchico: di conseguenza, a diverse scale di osservazione, si notano tipologie specifiche di moduli, "inseriti" in moduli di livello più alto, e costituiti da sotto-moduli più semplici.

Considerando il sistema nervoso nel suo complesso, possiamo individuare i seguenti componenti principali:

- I due ***emisferi cerebrali***, che presiedono alle più alte funzioni cognitive, percettive e motorie. Sono rivestiti dalla *corteccia*, una struttura a 6 strati caratterizzati da un gran numero di connessioni ricorrenti. Qui sono identificabili tipologie differenti di moduli, a seconda della scala di osservazione che adottiamo: si va da zone relativamente estese di corteccia fino alle *colonne corticali*. I neuroni qui presenti sono generalmente sensibili ad alcune caratteristiche dei segnali di ingresso, come ad esempio ad alcune specifiche bande di frequenza sonore per quanto riguarda i segnali acustici, o ad alcuni colori relativamente alla corteccia visiva (questo tipo di comportamento è simulabile in modo relativamente efficiente tramite *wavelets*). La risposta di questi neuroni alle caratteristiche dei segnali in ingresso dipende fortemente dall'esperienza e dall'apprendimento.
  Le sinapsi corticali generalmente seguono una regola di tipo Hebbiano per quanto riguarda la modifica dei loro pesi: vengono potenziate quando vi è coincidenza tra segnale presinaptico e risposta postsinaptica, e vengono depotenziate quando questo sincronismo non è presente.

- Il ***cervelletto***, che regola la forza e la precisione dei movimenti, controlla il loro apprendimento, l'equilibrio e i movimenti oculari. Ha un'organizzazione anatomica

estremamente uniforme, ed una modalità di trasmissione interna praticamente di tipo *feed-forward*. Se ne può identificare una struttura modulare nelle cosiddette "*microzone*", composte da circa 3000 cellule di Purkinje ciascuna. Queste, a loro volta, sono particolari tipi di neuroni che hanno due diversi tipi di ingressi: un primo gruppo di fibre (*parallel fibers*) trasmette un determinato segnale (ad esempio la codifica di un movimento da eseguire), mentre una fibra distinta (*climbing fiber*) codifica gli errori commessi dal neurone. In questo modo, l'attivazione coincidente di una parallel fiber e della climbing fiber induce una depressione a lungo termine (LTD) della parallel fiber.

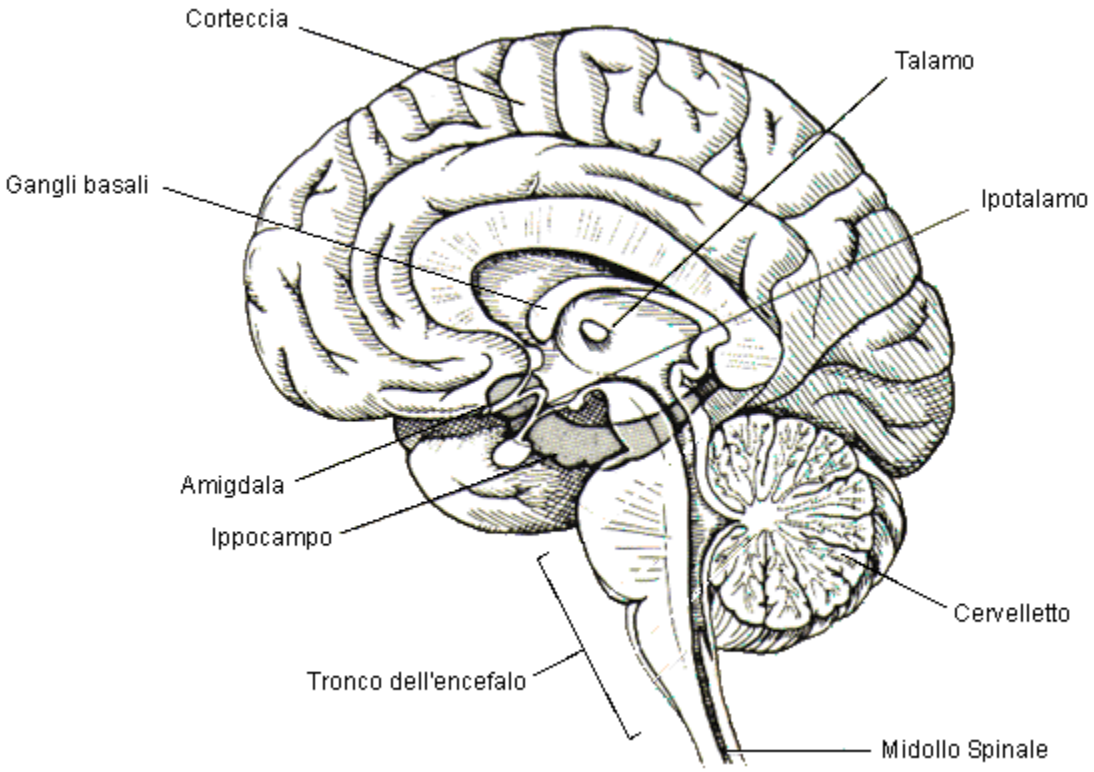
- I ***gangli basali***, coinvolti in aspetti di alto livello dei movimenti, e nei movimenti involontari; sono caratterizzati da una struttura "circuitale" parallela con molteplici vie inibitorie. Ricevono segnali dalla corteccia e trasmettono i loro segnali al talamo, che, essendo a sua volta collegato con la corteccia, permette di realizzare un "circuito chiuso". La struttura modulare che è qui identificabile è costituita dai *canali di uscita* (*output channels*). Dati sperimentali hanno messo in evidenza che i gangli basali sono fondamentali nell'apprendimento e nell'esecuzione di azioni che hanno un fine preciso e di tipo sequenziale, come ad esempio tutte le attività che hanno come conseguenza l'ottenimento di una ricompensa.

- Il ***diencefalo***, che fa da struttura di raccordo tra varie parti del sistema nervoso.

- Il ***tronco dell'encefalo***, che riceve informazioni dagli organi di senso dell'udito, dell'equilibrio e del gusto e controlla i muscoli facciali e i bulbi oculari.

- Il ***midollo spinale***, che è sia il canale di comunicazione con la periferia, sia una regione dove vengono elaborati segnali in modo riflesso e quindi involontario.

Per un approfondimento dal punto di vista anatomico e fisiologico del sistema nervoso, si veda [Kandel et al., 1991].



Secondo le teorie più diffuse riguardo il funzionamento del cervello, ogni sua parte sarebbe destinata a compiti specifici, come il controllo motorio, l'elaborazione dei segnali sensoriali, o il ragionamento. In questi ultimi anni, però, si sta facendo strada un'altra interpretazione del ruolo svolto da diverse regioni del sistema nervoso centrale [Doya, 1999]. Uno dei motivi che hanno spinto in questa direzione è il fatto che una serie di compiti sembrano essere svolti contemporaneamente da diverse aree del cervello, che inoltre sono tra loro collegate e sembrano attivarsi reciprocamente.

L'ipotesi, in parte confermata sperimentalmente, su cui si basa questa teoria consiste in quanto segue:

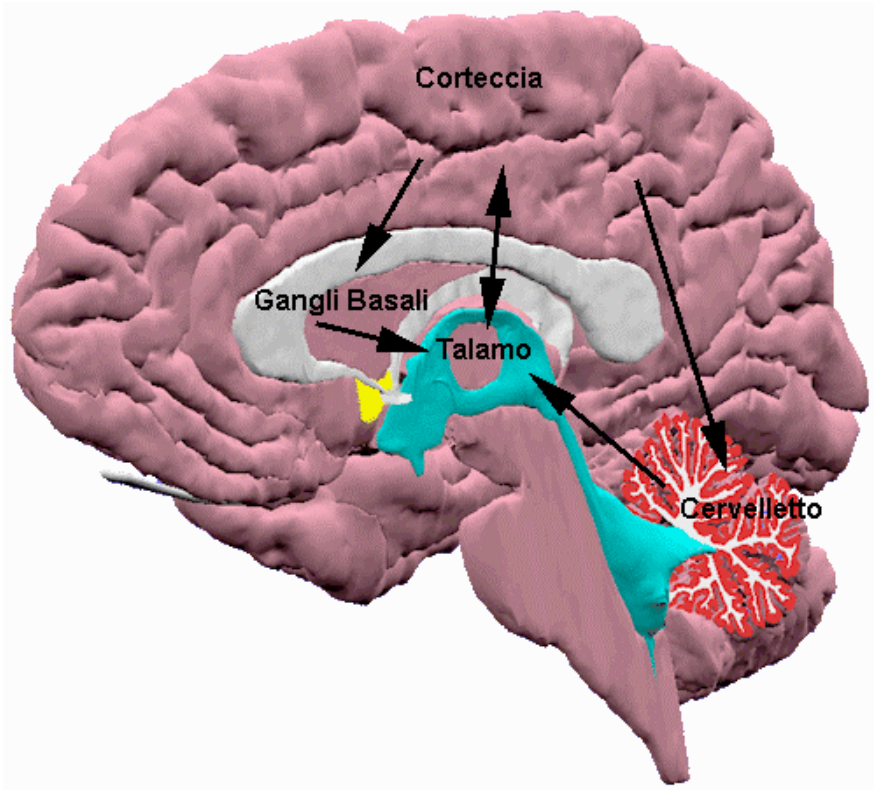
- Il cervelletto sarebbe specializzato nel realizzare una forma di apprendimento di tipo *supervisionato* (tramite l'interazione tra parallel fibers e climbing fiber). Una delle sue funzioni sarebbe quella di ricostruire un modello interno dell'ambiente.

- I gangli basali utilizzerebbero un apprendimento con *rinforzo*, per valutare determinate situazioni e selezionare una azione conseguente.

- La corteccia cerebrale, invece, funzionerebbe in modo *non supervisionato*, rappresentando lo stato dell'ambiente esterno e del contesto interno. Inoltre, non essendo direttamente collegati, mette in comunicazione cervelletto e gangli basali attraverso il talamo.

Di conseguenza, un determinato compito può essere portato a termine tramite la cooperazione di più parti del cervello che operano con "algoritmi" differenti, e allo stesso tempo una singola parte del cervello può essere utilizzata per compiti diversi a seconda della strada seguita al suo interno dalle informazioni e dalla loro successiva "area obbiettivo".

Nella seguente figura sono rappresentati i percorsi seguiti dai segnali all'interno del sistema nervoso centrale.



Si può notare come alcune aree siano collegate tra loro in modo bidirezionale (la corteccia con il talamo) mentre altre permettano il flusso dei segnali lungo una sola direzione (corteccia → gangli basali, gangli basali → talamo, corteccia → cervelletto, cervelletto → talamo). D'altra parte tutti i canali di trasmissione presenti (gli assoni, i dendriti e le sinapsi) sono unidirezionali, quindi per realizzare un canale bidirezionale è necessario avere un insieme di linee che trasmettano informazioni in una direzione, e un altro insieme distinto dal precedente che le trasmetta nella direzione opposta.

Dalla direzione delle frecce nella figura precedente si può notare come non esistano punti stazionari; di conseguenza, in linea teorica, un segnale può attraversare un numero indefinito di

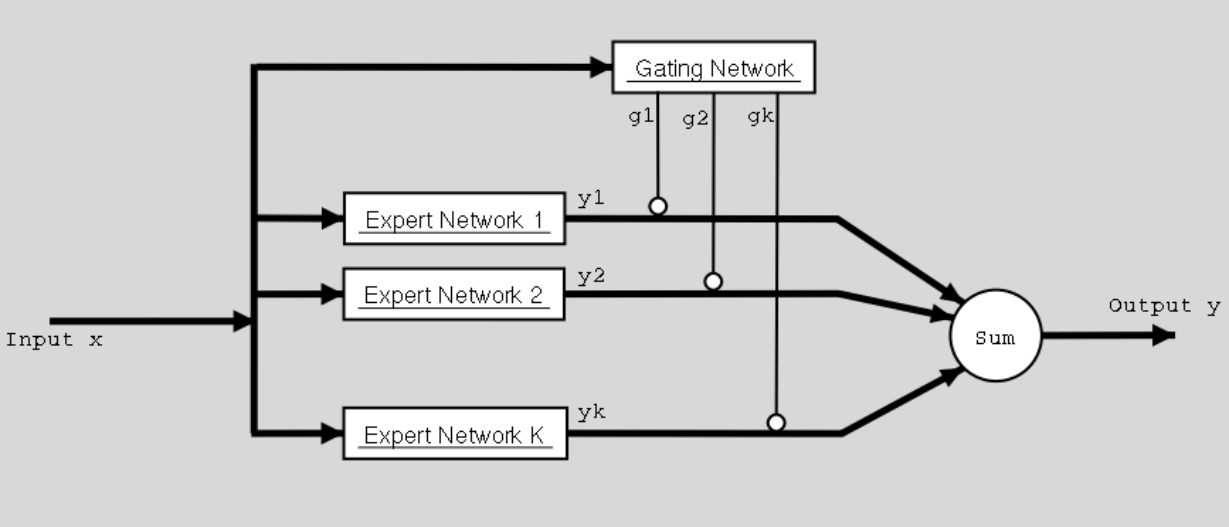
volte le stesse zone del cervello.

***

Tutte queste unità sono a loro volta costituite da strutture di livello inferiore, che variano a seconda della funzione svolta. Al livello più basso troviamo i neuroni, i dendriti e le sinapsi. In quanto segue verrà fatta l'ipotesi che i neuroni utilizzino una funzione di attivazione applicata alla loro uscita. Non dovrebbero esserci problemi, però, nel generalizzare la rete modulare per farle impiegare modelli più realistici come ad esempio gli *spiking neurons*, che, pur comportando un costo computazionale superiore, permettono di realizzare funzioni fuori dalla portata dei neuroni "classici".

6. Descrizione analitica di una rete con architettura "Mixture of Experts"

Riproponiamo il diagramma della rete neurale complessiva ed adottiamo una notazione per rappresentarla in modo analitico:



Notazione:

- **x** vettore di ingresso di dimensione p;
- **y** vettore di uscita di dimensione q;
- **d** uscita corretta di dimensione q;
- K numero di expert networks;
- $\mathbf{y}_i$ vettore di uscita della i-esima expert network di dimensione q;
- **g** vettore di uscita della gating network di dimensione K;
- h_i probabilità a posteriori che la i-esima expert network generi l'uscita attuale;
- **w** vettore dei pesi sinaptici delle expert networks;
- **a** vettore dei pesi sinaptici della gating network;
- e errore.

Vista come una “scatola nera”, la rete è analoga ad una tipica rete neurale supervisionata, sia per quanto riguarda la sua comunicazione con l'esterno, sia per quanto riguarda il modo in cui il suo funzionamento è percepito.

Denotiamo l'ingresso della rete con il vettore **x** (di dimensione *p*), l'uscita con il vettore **y** e l'uscita corretta con **d** (entrambi di dimensione *q*); quest’ultima verrà utilizzata durante la fase di addestramento per valutare l’errore commesso. Sia *K* il numero di expert networks presenti. Indichiamo l'uscita della *i*-esima expert network con $\mathbf{y}_i$ (sempre di dimensione *q*), dove *i* varia da 1 a *K*, e l'uscita della gating network con **g**, vettore di *K* componenti.
Il vettore di uscita della rete modulare sarà funzione dei vettori di uscita delle expert networks e del vettore **g**; in particolare è comodo utilizzare una combinazione lineare dei vettori pesati dai valori g_i:

$$\mathbf{y} = \sum_{i=1}^{K} g_i \cdot \mathbf{y}_i$$

Consideriamo adesso il comportamento della gating network: come scritto in precedenza, questa può operare in diverse modalità, a seconda dell’algoritmo che si intende ottenere. Di seguito verrà trattato il caso in cui viene fatta una somma pesata delle uscite delle expert networks, con l’ulteriore vincolo di forzare la somma delle uscite g_i ad 1, $g_i \geq 0 \ \forall i$.
In questo modo sarà possibile interpretare i valori forniti come delle probabilità associate alle singole expert networks. Più precisamente il valore assunto da una uscita della gating network rappresenta la *probabilità a priori* (cioè la probabilità basata solo sullo stato degli ingressi, con l'uscita corretta incognita) che la corrispondente expert network stia fornendo una risposta corretta.

Indicando con u_i i valori di uscita della gating network non ancora normalizzati, con $G(\cdot,\cdot)$ la funzione che rappresenta il funzionamento di questo modulo, e con $\mathbf{a}_i$ un insieme di parametri di controllo, si ha:

$$u_i = G(\mathbf{x}, \mathbf{a}_i)$$

Per rispettare il vincolo precedente che impone $\sum_i g_i = 1$, $g_i \geq 0 \ \forall i$, si può utilizzare la funzione *softmax*:

$$g_i = \frac{\exp(u_i)}{\sum_j \exp(u_j)}$$

da cui si deduce che le seguenti condizioni devono essere soddisfatte:

$$0 \leq g_i \leq 1 \quad \forall i, \qquad \sum_{i=1}^{K} g_i = 1$$

Volendo implementare la gating network tramite una classica rete neurale senza strati nascosti e con funzione di attivazione lineare, si ottiene la seguente espressione per $G(\cdot,\cdot)$:

$$u_i = \mathbf{x}^T \cdot \mathbf{a}_i$$

dove il vettore **a** rappresenta i pesi sinaptici della rete in questione, e $\mathbf{x}^T$ è la matrice trasposta di **x**.

Per quanto riguarda le expert networks, queste possono avere una struttura e un modo di funzionamento molto diversi tra loro, anche all'interno della stessa rete modulare. Una lista (incompleta) degli algoritmi che possono essere utilizzati per realizzarle è la seguente:

- reti neurali supervisionate o non supervisionate;
- reti feed-forward o ricorrenti;
- algoritmi fuzzy;
- algoritmi genetici;
- metodi statistici;
- metodi simbolici;
- sistemi esperti;
- ……

Indicando con $\mathbf{y}_i$ il vettore di uscita della *i*-esima expert network, rappresentata in modo generico con $E_i(\cdot,\cdot)$, e con $\mathbf{w}_i$ i relativi parametri di controllo, si ha:

$$\mathbf{y}_i = E_i(\mathbf{x}, \mathbf{w}_i)$$

Da questa equazione si può notare che i parametri presenti non sono solo i vettori $\mathbf{w}_i$, ma c'è anche una dipendenza dal genere di moduli utilizzati, che possono essere di tipo diverso all'interno dello stesso sistema (indice *i* in E_i).

Come si vedrà più avanti, questi competeranno tra loro per fornire la soluzione e per apprendere l'esempio corrispondente, e saranno quindi soggetti ad una sorta di "selezione naturale" che comporterà una "evoluzione" della struttura. Inoltre, a differenza di quanto avviene per altri algoritmi (tra cui quello di *boosting*), la competizione tra i moduli non avviene in modo globale su tutto il training set, cercando l'elemento migliore in senso assoluto. Piuttosto le expert networks si confrontano di volta in volta sul singolo esempio in esame, e relativamente a questo avviene la competizione. Di conseguenza, con il procedere dell'addestramento, alcuni moduli si specializzeranno su un sottoinsieme dei possibili ingressi, mentre altri moduli si faranno carico degli altri.
In altri termini, il compito della gating network è quello di *partizionare* in modo lineare o non lineare lo spazio di ingresso, e di assegnare i sottospazi corrispondenti a specifiche expert networks. Questo partizionamento, realizzato durante la fase di addestramento, può essere fatto in modo "rigido" o "sfumato". Nel primo caso ad un certo insieme di valori di ingresso può corrispondere una ed una sola expert network (ed in questo caso il sistema si comporta come una rete "*winner takes all*") o un insieme definito di expert networks (se la gating network fornisce uscite di tipo binario). Nel secondo caso ogni vettore di ingresso viene elaborato da tutte le expert network presenti, che contribuiranno alla formazione dell'uscita in modo variabile ed in funzione del comportamento della gating network. Quindi tra le partizioni in cui

viene suddiviso lo spazio di ingresso non si possono individuare confini netti come nel primo caso, ma piuttosto si avranno delle aree delimitate da perimetri *fuzzy* (dove le aree e i perimetri si intendono in uno spazio *n* dimensionale).

Questa forma di competizione/cooperazione permette di ottenere buoni risultati con algoritmi estremamente semplici: il motivo è che essendo lo spazio dei valori di ingresso partizionato dalla gating network, ogni expert network si trova a dover risolvere solo un sotto-problema meno complesso di quello di partenza.

Di seguito, quindi, considererò solo expert networks formate da reti neurali supervisionate, feed-forward, costituite di un unico strato di neuroni. Si tenga presente, comunque, che la discussione che segue può essere immediatamente generalizzata a qualsiasi tipo di modulo si voglia utilizzare.

* * *

Per ottenere un algoritmo di qualunque tipo, è necessario innanzi tutto stabilire quali informazioni iniziali si hanno a disposizione, e quale obbiettivo si vuole raggiungere. Nel caso degli algoritmi di apprendimento supervisionato, l'obbiettivo è quello di modellare la distribuzione di probabilità del training set, rappresentabile tramite un insieme di coppie ingresso-uscita desiderata $\{\mathbf{x},\mathbf{d}\}$, che costituiscono le informazioni iniziali.
D'altra parte, per modellare una distribuzione di probabilità si può massimizzare la funzione di verosimiglianza (*likelihood*). Tenendo presente che, nel caso in esame, le expert networks e la gating network vengono addestrate contemporaneamente, consideriamo il funzionamento del primo tipo di rete. Data la *i*-esima expert network, questa è la sua funzione di verosimiglianza:

$$f(\mathbf{d} \mid \mathbf{x}, i) = \frac{1}{(2\pi)^{q/2}} \exp\left(-\frac{1}{2} \left\| \mathbf{d} - \mathbf{y}_i \right\|^2 \right)$$

E' immediato verificare come questa funzione cresca con il tendere di $\mathbf{y}_i$ a **d**, e tenda a zero in caso contrario. Quindi la verosimiglianza cresce con l'avvicinarsi della risposta fornita alla risposta corretta. Lo scopo della sotto-rete in questione è allora quello di massimizzare il valore di *f*.

Prendendo ora in considerazione la rete complessiva, possiamo scrivere una equazione che leghi più expert networks tramite la gating network:

$$f(\mathbf{d} \mid \mathbf{x}) = \sum_{i=1}^{K} g_i f(\mathbf{d} \mid \mathbf{x}, i) = \frac{1}{(2\pi)^{q/2}} \sum_{i=1}^{K} g_i \exp\left(-\frac{1}{2} \left\| \mathbf{d} - \mathbf{y}_i \right\|^2 \right)$$

Questa equazione rappresenta una distribuzione di probabilità chiamata *"modello di mistura gaussiana associativa"* (una brutta traduzione di *"associative Gaussian mixture model"*), dove il termine "associativa" indica che l'apprendimento è basato sull'associazione tra un ingresso e la corrispondente uscita corretta. I parametri da calcolare per massimizzare questa funzione di verosimiglianza sono i pesi sinaptici **w** delle expert networks e i pesi sinaptici **a** della gating network (si noti che anche se i vettori **w** ed **a** non compaiono esplicitamente nella precedente equazione, i valori di $\mathbf{y}_i$ e **g** ne dipendono direttamente).

Poiché sarà più semplice lavorare con il logaritmo naturale di $f(\mathbf{d}|\mathbf{x})$, definiamo la seguente funzione che utilizzeremo in seguito:

$$l(\mathbf{w}, \mathbf{g}) = \ln f(\mathbf{d} \mid \mathbf{x})$$

Precedentemente abbiamo considerato la probabilità a priori g_i che la *i*-esima expert network generasse l'uscita. Questa probabilità è stata definita "a priori" proprio perché l'uscita stessa non era nota, ma lo erano solo gli ingressi. Ora, conoscendo l'uscita corretta **d**, possiamo utilizzare la funzione di verosimiglianza trovata per modificare i parametri che influenzano il funzionamento della rete neurale: i pesi sinaptici delle expert networks e della gating network.

Definiamo la probabilità a posteriori h_i che la i-esima expert network generi l'uscita attuale:

$$h_i = \frac{g_i \exp\left(-\frac{1}{2}\left\|\mathbf{d} - \mathbf{y}_i\right\|^2\right)}{\sum_{j=1}^{K} g_j \exp\left(-\frac{1}{2}\left\|\mathbf{d} - \mathbf{y}_j\right\|^2\right)} \qquad i = 1,\ldots,K$$

Questo è il rapporto tra la funzione di verosimiglianza della expert network che stiamo considerando e la somma delle funzioni di verosimiglianza di tutte le expert networks, dove tutti i termini sono pesati dai corrispondenti elementi del vettore **g**. Dunque la probabilità a posteriori h_i dipende sia dalla correttezza della risposta data, sia dalla probabilità a priori g_i che la i-esima expert network ha di rispondere al particolare ingresso presentato. Quindi, qualunque sia l'algoritmo di apprendimento utilizzato dalle expert networks, la modifica dei loro parametri deve essere pesata dai valori h_i. Così facendo è possibile addestrare principalmente le expert networks che per l'ingresso in questione si mostrano "migliori" delle altre, lasciando a queste ultime la possibilità di specializzarsi per altri elementi del training set. Inoltre, in questo contesto, il fattore g_i può essere visto come una sorta di "inerzia", che agevola l'apprendimento dei moduli che si erano dimostrati migliori nelle iterazioni precedenti per gli stessi (o per simili) valori di ingresso. E' attraverso questo meccanismo, dunque, che si realizza la competizione tra i diversi moduli della rete.
Come per le uscite della gating network, anche per i valori h_i valgono le seguenti condizioni:

$$0 \le h_i \le 1 \quad \forall i, \qquad \sum_{i=1}^{K} h_i = 1$$

Vediamo ora come modificare i pesi delle sinapsi contenute nelle expert networks per risolvere un problema di regressione, considerando neuroni con funzione di attivazione lineare regolati dalla relazione:

$$y_i^{(m)} = \mathbf{x}^T \mathbf{w}_i^{(m)} \qquad \begin{cases} i = 1,\ldots,K \\ m = 1,\ldots,q \end{cases}$$

Poiché vogliamo che l'uscita della expert network in questione si avvicini all'uscita corretta, dobbiamo massimizzare il logaritmo della funzione di verosimiglianza definito precedentemente. Deriviamo dunque questa funzione rispetto ai parametri da modificare, cioè i pesi sinaptici:

$$\frac{\partial l}{\partial \mathbf{w}_i^{(m)}} = \frac{\partial l}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial \mathbf{w}_i^{(m)}}$$

Da quanto appena ipotizzato riguardo il funzionamento dei neuroni, si ottiene che i fattori a secondo membro della precedente equazione sono pari a:

$$\frac{\partial y_i^{(m)}}{\partial \mathbf{w}_i^{(m)}} = \mathbf{x}$$

$$\frac{\partial l}{\partial y_i^{(m)}} = \frac{\partial}{\partial y_i^{(m)}} \ln f(\mathbf{d} \mid \mathbf{x}) = \frac{\partial}{\partial y_i^{(m)}} \ln \sum_{j=1}^{K} g_j f(\mathbf{d} \mid \mathbf{x}, j) =$$

$$= \frac{\partial}{\partial y_i^{(m)}} \ln \left[\frac{1}{(2\pi)^{q/2}} \sum_{j=1}^{K} g_j \exp\left(-\tfrac{1}{2} \left\| \mathbf{d} - \mathbf{y}_j \right\|^2 \right) \right] =$$

$$= \frac{g_i \exp\left(-\tfrac{1}{2} \left\| \mathbf{d} - \mathbf{y}_i \right\|^2 \right)}{\sum_{j=1}^{K} g_j \exp\left(-\tfrac{1}{2} \left\| \mathbf{d} - \mathbf{y}_j \right\|^2 \right)} \left(d^{(m)} - y_i^{(m)} \right) = h_i \left(d^{(m)} - y_i^{(m)} \right)$$

Combinando le tre precedenti equazioni si ottiene:

$$\frac{\partial l}{\partial \mathbf{w}_i^{(m)}} = h_i \left(d^{(m)} - y_i^{(m)} \right) \mathbf{x} = h_i e_i^{(m)} \mathbf{x}$$

Ora dobbiamo spostarci nella direzione di massima crescita del gradiente di l nello spazio dei pesi; quindi modifichiamo $\mathbf{w}_i$ nel seguente modo:

$$\Delta \mathbf{w}_i^{(m)} = \eta \frac{\partial l}{\partial \mathbf{w}_i^{(m)}} = \eta h_i e_i^{(m)} \mathbf{x}$$

dove il parametro η è un piccolo tasso di apprendimento.

Utilizzando come algoritmo di risoluzione il metodo di *Eulero in avanti*, indicando con n l'attuale passo di calcolo, si ottiene la seguente regola di aggiornamento per $\mathbf{w}$:

$$\mathbf{w}_i^{(m)}(n+1) = \mathbf{w}_i^{(m)}(n) + \Delta \mathbf{w}_i^{(m)}(n)$$

Pur essendo il metodo di Eulero in avanti (esplicito, del primo ordine) il più semplice ed immediato per risolvere questo tipo di equazioni, ha il difetto di essere molto instabile e, di conseguenza, richiede valori di η molto piccoli per non divergere. Non ha questi difetti il metodo di *Eulero all'indietro* (implicito, ancora del primo ordine), che è sempre stabile, e che permette di utilizzare valori di η molto più grandi (e grazie a ciò, pur avendo un costo computazionale superiore per singolo passo di calcolo, è più efficiente del metodo di Eulero in avanti). La corrispondente regola di aggiornamento è:

$$\mathbf{w}_i^{(m)}(n+1) = \mathbf{w}_i^{(m)}(n) + \Delta \mathbf{w}_i^{(m)}(n+1)$$

Consideriamo adesso il funzionamento della gating network, che assumerò composta da un unico strato di neuroni come fatto in precedenza. Questo tipo di rete ha un numero di unità di uscita uguale al numero di expert networks presenti, poiché proprio a queste sono diretti i segnali da lei generati. Inoltre, come abbiamo visto precedentemente, le uscite vengono normalizzate nel seguente modo per poterne dare un'interpretazione probabilistica:

$$g_i = \frac{\exp(u_i)}{\sum_j \exp(u_j)}$$

L'obbiettivo della gating network, durante la fase di addestramento, è quello di modificare i propri parametri (pesi sinaptici) contenuti nel vettore **a** affinché le probabilità a priori **g** tendano alle probabilità a posteriori **h**.

Sostituiamo allora la precedente espressione nella funzione che da il logaritmo della verosimiglianza $l(\mathbf{w},\mathbf{g})$:

$$l(\mathbf{w},\mathbf{g}) = \ln f(\mathbf{d} \mid \mathbf{x}) = \ln \sum_{i=1}^{K} g_i f(\mathbf{d} \mid \mathbf{x}, i) =$$

$$= \ln\left[\frac{1}{(2\pi)^{q/2}} \sum_{i=1}^{K} g_i \exp\left(-\frac{1}{2} \left\| \mathbf{d} - \mathbf{y}_i \right\|^2 \right) \right] =$$

$$= \ln\left[\frac{1}{(2\pi)^{q/2}} \sum_{i=1}^{K} \frac{\exp(u_i)}{\sum_j \exp(u_j)} \exp\left(-\frac{1}{2} \left\| \mathbf{d} - \mathbf{y}_i \right\|^2 \right) \right] =$$

$$= \ln \frac{1}{(2\pi)^{q/2}} + \ln \sum_{i=1}^{K} \exp(u_i) \cdot \exp\left(-\frac{1}{2} \left\| \mathbf{d} - \mathbf{y}_i \right\|^2 \right) - \ln \sum_{j=1}^{K} \exp(u_j)$$

Ora, per trovare la direzione in cui modificare i pesi sinaptici contenuti nel vettore **a**, deriviamo il logaritmo della verosimiglianza rispetto a questi parametri:

$$\frac{\partial l}{\partial \mathbf{a}_i} = \frac{\partial l}{\partial u_i} \cdot \frac{\partial u_i}{\partial \mathbf{a}_i}$$

Da quanto ricavato precedentemente, si ha:

$$\frac{\partial u_i}{\partial \mathbf{a}_i} = \frac{\partial (\mathbf{x}^T \mathbf{a}_i)}{\partial \mathbf{a}_i} = \mathbf{x}$$

$$\frac{\partial l}{\partial u_i} = \frac{\partial}{\partial u_i} \ln f(\mathbf{d} \mid \mathbf{x}) = \frac{\partial}{\partial u_i} \ln \sum_{i=1}^{K} g_i f(\mathbf{d} \mid \mathbf{x}, i) =$$
$$= \frac{\partial}{\partial u_i} \ln \left[\frac{1}{(2\pi)^{q/2}} \sum_{i=1}^{K} g_i \exp\left(-\frac{1}{2} \left\| \mathbf{d} - \mathbf{y}_i \right\|^2 \right) \right] =$$
$$= \frac{\partial}{\partial u_i} \ln \sum_{j=1}^{K} g_j \exp\left(-\frac{1}{2} \left\| \mathbf{d} - \mathbf{y}_j \right\|^2 \right) = h_i - g_i$$

da cui:

$$\frac{\partial l}{\partial \mathbf{a}_i} = \frac{\partial l}{\partial u_i} \cdot \frac{\partial u_i}{\partial \mathbf{a}_i} = (h_i - g_i)\,\mathbf{x}$$

Per ottenere un massimo dalla funzione di verosimiglianza poniamo $\partial l / \partial \mathbf{a}_i$ uguale a zero, il che vuol dire che la probabilità a priori **g** deve tendere alla probabilità a posteriori **h**:

$$\frac{\partial l}{\partial \mathbf{a}_i} = (h_i - g_i)\,\mathbf{x} = 0 \qquad \Rightarrow \qquad h_i = g_i$$

La variazione dei pesi **a** è allora:

$$\Delta \mathbf{a}_i = \eta \frac{\partial l}{\partial \mathbf{a}_i} = \eta (h_i - g_i)\mathbf{x}$$

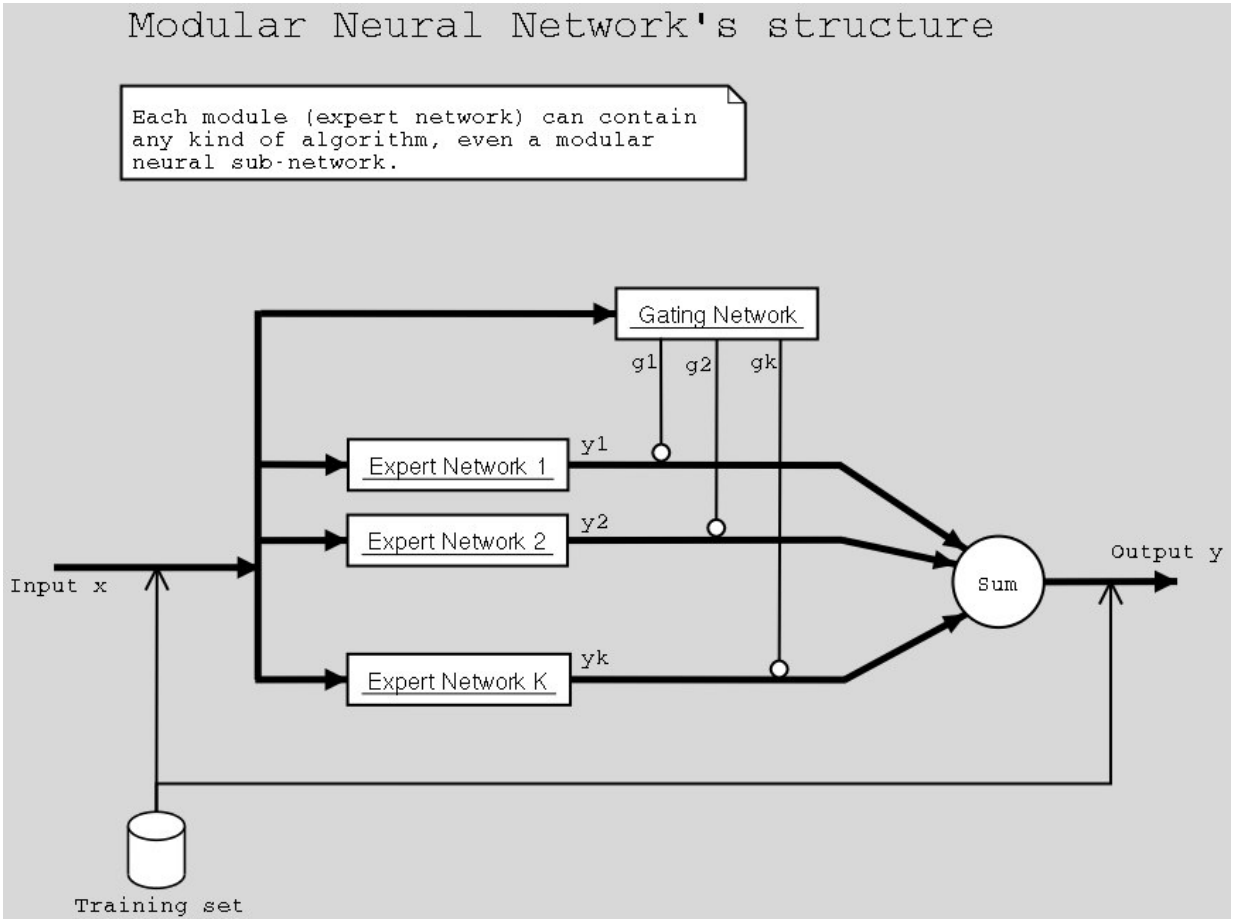
che, appunto, si stabilizza quando g_i tende a h_i.

Come si può notare non c'è retro-propagazione dell'errore in quanto tutti i moduli sono composti da reti con un singolo strato di neuroni.

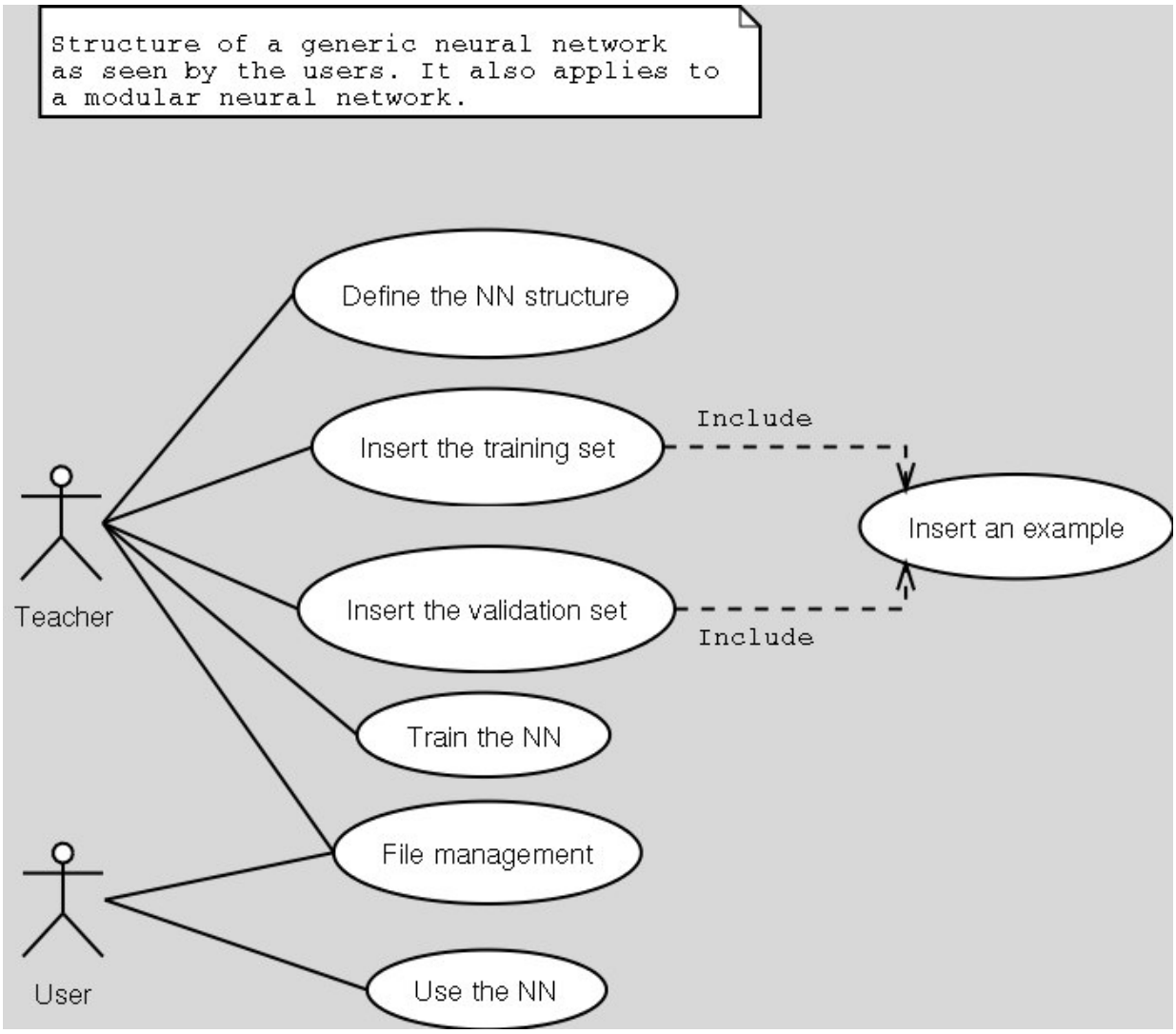
7. Descrizione di una possibile implementazione tramite il linguaggio di modellazione UML

Di seguito presenterò la struttura del programma che sto sviluppando per la simulazione di una rete neurale modulare. Utilizzerò il linguaggio di modellazione UML (Unified Modeling Language), che da qualche anno è divenuto uno standard nella rappresentazione grafica di algoritmi, sistemi, processi, basi di dati, e di tutto ciò che è riconducibile ad una struttura ad oggetti. Per una introduzione a questo argomento si veda l'appendice B e [Fowler, 2000].

Innanzitutto ripropongo lo schema mostrato in precedenza che rappresenta la struttura di un modello basato su "Mixture of Experts", che permetterà di associare i componenti descritti successivamente con la funzione da loro svolta:



Seguendo lo “stile” UML, vediamo il sistema dal punto di vista dell’utilizzatore, umano o non umano, reale o astratto, con cui il programma dovrà interagire.
Ecco allora il diagramma *use-case* corrispondente:

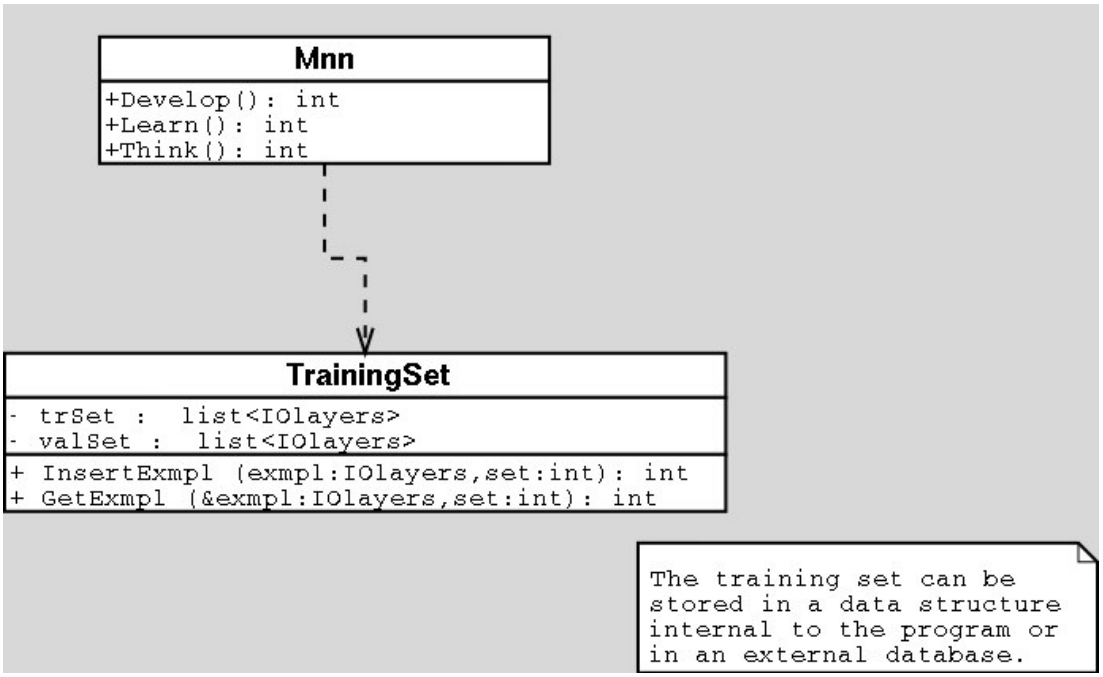


In questo schema è possibile notare le due modalità che si hanno a disposizione per interagire con il sistema: svolgendo il ruolo di utente (user) o di insegnante (teacher). Mentre il primo ruolo può essere svolto da un utente reale, il secondo sarà più convenientemente svolto da un

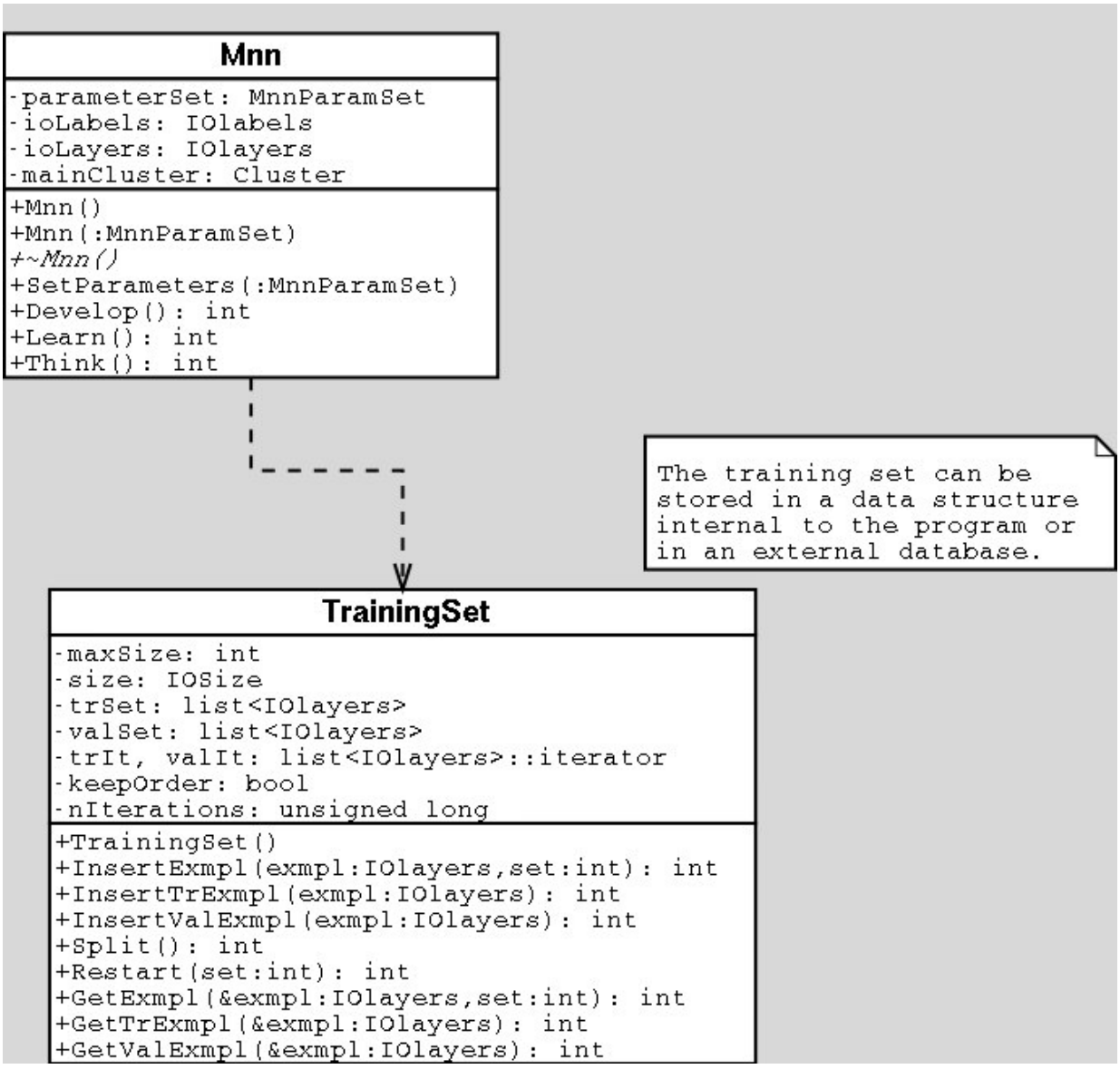
file o da una base di dati contenente l'insieme degli esempi per l'addestramento e la validazione della rete. All'interno degli ovali si trovano le varie funzioni svolte dal programma, così come sono percepite dall'utilizzatore: come primo passo, dopo aver inquadrato il problema da affrontare, si deve definire la rete in termini di dimensione dei vettori di ingresso e di uscita, oltre che di dimensionamento dei componenti interni (numero e tipologia di Expert Networks, livello di nidificazione delle strutture dati, ecc.). Successivamente si dovrà provvedere all'inserimento del training set e del validation set. A questo punto si potrà iniziare l'addestramento, per poi passare all'utilizzo del sistema una volta scesi sotto una soglia di errore prefissata.
Il diagramma use-case utilizzato, rappresentando la visione che si ha del sistema dall'esterno, è molto generico e può essere applicato ad una rete neurale o addirittura ad un algoritmo di apprendimento qualsiasi.

Come passo successivo consideriamo la struttura interna del programma tramite una serie di diagrammi di classe (*class diagrams*), che adesso sono legati all'implementazione specifica.

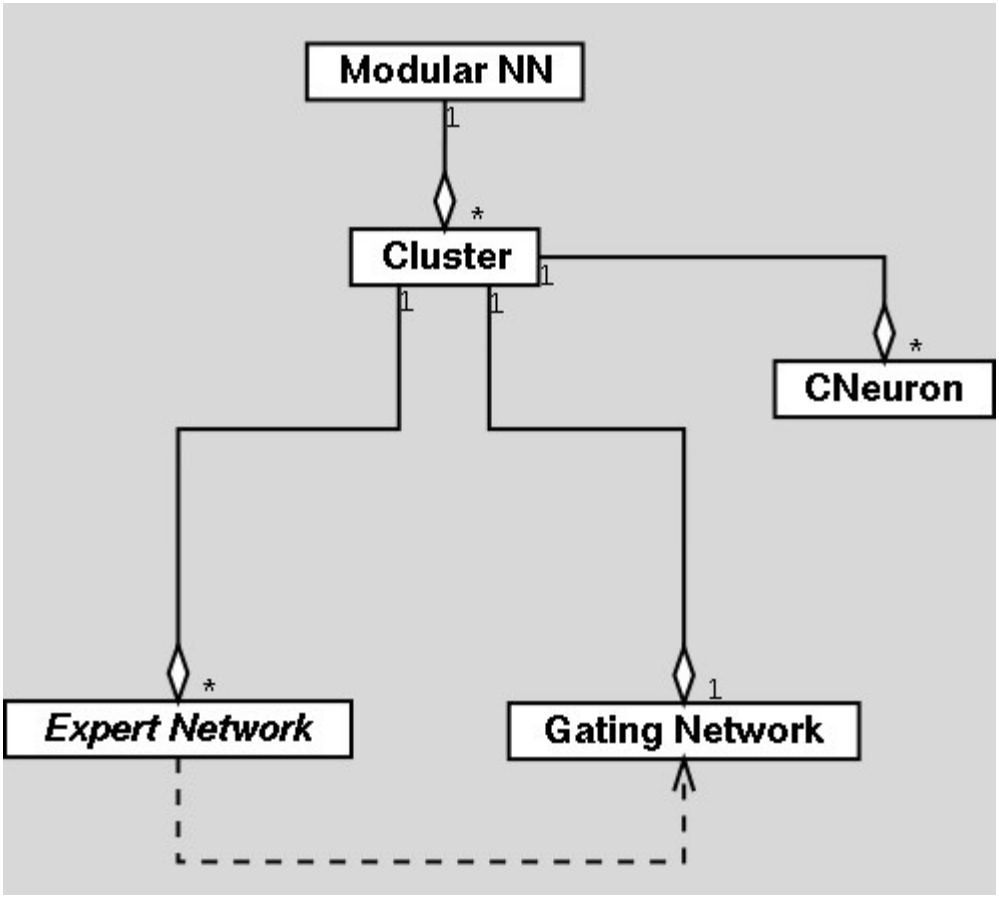
Seguiamo un approccio *top-down*, che ci permetterà di esaminare il sistema a partire dai blocchi gerarchicamente a più alto livello:



Da questo punto di vista, il programma può essere scomposto in due classi: **`Mnn`** che rappresenta la rete vera e propria, e **`TrainingSet`** che rappresenta l'insieme di esempi per l'addestramento e la validazione (raccolti nella stessa classe e gestiti in modo omogeneo).
Di seguito abbiamo un diagramma equivalente al precedente, ma più dettagliato:



Ora “esplodiamo” la classe `Mnn` ed esaminiamone la struttura. Scendendo di un livello nella gerarchia, troviamo che la rete modulare complessiva è costituita da uno o più oggetti di classe **`Cluster`**, che possono essere visti come “contenitori” che racchiudono altri sotto-moduli. Questi sotto-moduli, le *Expert Networks*, sono quelli che ho realizzato con delle reti neurali, ma che in realtà possono funzionare con algoritmi qualsiasi, anche diversi tra loro all’interno dello stesso cluster e che possono competere o collaborare nel fornire l’uscita corrispondente all’ingresso corrente.



Da quanto scritto durante la descrizione analitica della rete, abbiamo che ogni cluster può contenere più expert networks e una sola gating network.

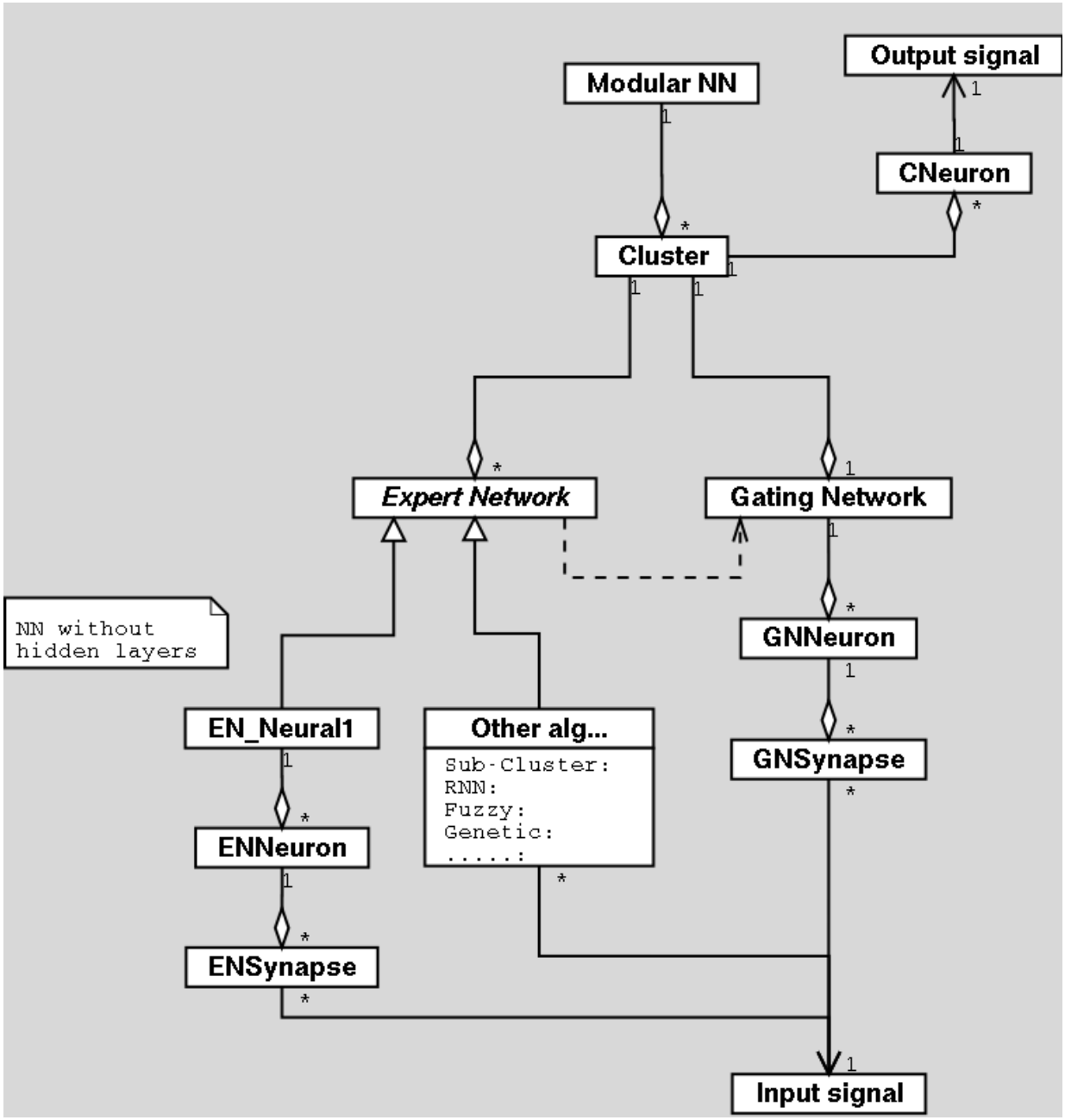
Nel precedente diagramma UML, il simbolo di rombo presente ad una estremità delle linee che collegano le classi esprime un relazione di classe nidificata (*inner class* o *nested class*). La freccia tratteggiata invece indica che le classi corrispondenti sono in qualche modo collegate e che possono scambiarsi informazioni. I numeri e gli asterischi indicano la molteplicità esistente nella relazione, in modo analogo a quanto avviene nei diagrammi Entità-Relazione utilizzati per modellare le basi di dati.

Scendiamo ora più nel dettaglio, andando ad esaminare la relazione gerarchica esistente tra tutti i moduli dal livello più alto al più basso.

Nel diagramma che segue, si può notare che la classe che rappresenta le Expert Networks è una classe astratta (il suo nome è scritto in corsivo); in questo modo il concetto di Expert Network viene reso generico, permettendone una manipolazione uniforme indipendentemente da come verranno effettivamente implementate le sue classi derivate. In questo caso le classi derivate mostrate rappresentano una rete neurale di tipo classico senza strati nascosti e una lista dei possibili algoritmi alternativi utilizzabili. Tra questi si noti che vi può essere anche un cluster di livello inferiore, in modo da poter realizzare una struttura che si nidifica ricorsivamente. Le altre possibilità indicate riguardano l'utilizzo di reti neurali ricorrenti, sistemi fuzzy e algoritmi genetici. Questi ultimi, nel contesto del modello che sto presentando, possono essere di due tipi:

- algoritmi genetici racchiusi in moduli che ne mascherano il funzionamento, che si interfacciano con l'esterno solo tramite i loro ingressi e uscite;
- un algoritmo genetico che fa evolvere la *struttura* della rete complessiva (si veda cap. 13).

Un discorso analogo si può fare per la gating network, qui rappresentata come un rete neurale generica.


Output signal
Modular NN
CNeuron
Cluster
Expert Network
Gating Network
NN without
hidden layers
GNNeuron
EN_Neural1
Other alg...
Sub-Cluster:
RNN:
Fuzzy:
Genetic:
.....:
GNSynapse
ENNeuron
ENSynapse
Input signal

I diagrammi di classe fin qui mostrati rappresentano la struttura del programma da un punto di vista statico: ne mostrano, come dice il nome, la gerarchia delle classi e il modo in cui sono collegate tra loro. Non ne viene messo in evidenza, però, il comportamento in funzione del tempo ed in funzione di ciò che accade nel contesto in cui operano.

8. Descrizione di una possibile implementazione tramite il linguaggio Standard C++

In questo capitolo descriverò a grandi linee la struttura di un programma che sto sviluppando per la simulazione di una rete neurale modulare.
Ho scelto il linguaggio Standard C++ per la combinazione di strutture ad alto livello che fornisce e per le sue prestazioni, oltre al fatto di essere uno standard multi piattaforma.

Presenterò le classi principali che costituiscono lo "scheletro" precedentemente mostrato in forma grafica tramite UML.

Partendo anche qui dal livello più alto, abbiamo la classe `cluster` che contiene al suo interno gli altri oggetti secondo la gerarchia già mostrata (verranno omesse le istruzioni del precompilatore, i costruttori e le parti meno significative):

```
// cluster.h:  interface for the Cluster class.
//
//////////////////////////////////////////////////////////////////////

class  Cluster
   {
      MnnParamSet parameterSet;     // set of parameters for the part of
                                    //  the nn contained in this cluster.

      float  h;         // a posteriori probaility that this cluster
                        //  generates a particular desired response.
      float  sqrErr;          // cluster's squared error
      float  clusterDen;      // used to calculate h

      int  level;       // cluster's level (0 for the mainCluster)

      vector<CNeuron>      cNeu;       // set of cluster neurons

      GatingNetwork        gn;         // GN inside the cluster

      vector<ExpertNetwork*>  en;            // set of Expert Networks
      vector<Cluster>         subCluster;    // set of sub-clusters
```

```
   public:
      Cluster (const MnnParamSet &parSet)  { parameterSet = parSet; }

      virtual int  Develop (IOlayers &ioLayers);
      virtual int  Develop (IOlayers &ioLayers,
                            const MnnParamSet &parameterSet);
      virtual int  Learn (void);
      virtual int  Think (void);

   };



// expertnetwork.h:  interface for the ExpertNetwork abstract class.
//
//////////////////////////////////////////////////////////////////////

/**
   ExpertNetwork is an abstract class that provides the general
   concept of a learning module. Its derived classes can use any
   kind of algorithm to reach the result (e.g. neural networks,
   fuzzy logic, genetic algorithms, simbolic reasoning, ...).
*/

   class  ExpertNetwork
   {
   protected:
      float  h;      // a posteriori probability that this expert network
                     //  generates a particular desired response.

      float  sqrErr;             // EN's squared error

      vector< vector<float> >  output;    // output layer of this module

   public:
      virtual void  Develop (IOlayers &ioLayers)  {}
      virtual void  CalcError (const IOlayers&);
      virtual float Error (void)  { return sqrErr; }
      virtual void  Learn (void)  {}
      const vector< vector<float> >&  Output (void)  { return  output; }
   };
```

```
// gatingnetwork.h:  interface for the GatingNetwork class.
//
//////////////////////////////////////////////////////////////////////

   class GatingNetwork
   {
      vector<GNNeuron>  gnNeuron;      // 1-D layer of neurons

      vector<float>  output;        // output layer of this module
      float  normFactor;            // used for output normalization

   public:
      virtual void  Develop (IOlayers &ioLayers, int nEN);
      virtual void  Develop (IOlayers &ioLayers);
      virtual void  Think (void);
   };



// enneuron.h:  interface for the ENNeuron and ENSynapse classes.
//
//////////////////////////////////////////////////////////////////////

   class  ENNeuron : public Neuron
   {
      class  ENSynapse;
      friend class  ENSynapse;
      vector< vector<ENSynapse> >  synapse;

      inline void  LearnBias (float h);

   public:
      void  Develop (IOlayers &ioLayers);          // neuron's development
      float Fire (void);
      void  Learn (float h, float correctResponse);
   };



   class  ENNeuron::ENSynapse : public Synapse
   {
      float  *pInput;      // point to an input layer element

   public:
      bool  Connect (IOlayers&, InputLayElem&);       // set pInput
      float Signal (void);                            // signal transmission
      void  Learn (const ENNeuron *father, float h);  // weight modification
   };
```

9. Utilizzo di wavelets per la pre-elaborazione dell'ingresso

Una rete neurale può fornire risultati molto diversi in termini di prestazioni e di generalizzazione a seconda del fatto che i suoi ingressi vengano manipolati in qualche modo prima di essere introdotti nel sistema.

Fino a qualche anno fa il metodo più comune era la serie di Fourier, ma ultimamente ha preso piede una sua alternativa: le wavelets.
Entrambi questi metodi hanno la caratteristica di ricevere un generico segnale in ingresso, e di fornire un altro segnale trasformato in uscita.

La serie di Fourier, in particolare, esprime un segnale come una somma di una serie (anche infinita) di seni e di coseni. Il suo grosso svantaggio consiste nel fatto che ha risoluzione in frequenza, ma non spaziale; quindi, anche se ci fornisce tutte le frequenze presenti nel segnale, non ci può dire dove sono collocate temporalmente.

Le wavelets invece adottano un approccio diverso: una finestra di dimensioni variabili esamina il segnale in posizioni diverse fornendone una rappresentazione “multipla”. Un’immagine, ad esempio, può essere trasformata tramite wavelets in un’altra immagine dove i dettagli sono distribuiti in modo “ordinato” sulla sua superficie; partendo da una matrice di pixel che rappresentano la luminosità di una scena, se ne può ricavare un’altra dove il pixel più in alto a sinistra rappresenta la luminosità media dei pixel originali, e man mano che ci allontana da questo gli altri pixel rappresentano dei dettagli sempre maggiori. Dopo una tale trasformazione, diventa banale effettuare una compressione dell’immagine: è sufficiente “tagliare” i bordi destro ed inferiore che contengono informazioni di dettaglio che possono essere trascurabili.

Un altro motivo che giustifica l’interesse sorto per questo argomento dipende dal fatto che

anche l'occhio umano elabora i segnali secondo questa modalità, anche se attraverso una rete neuronale. Questo fatto è riscontrabile osservando le proprietà che hanno questi algoritmi, che risultano simili a quelle della retina, del nervo ottico e della corteccia visiva: invarianza traslazionale, invarianza di scala, e invarianza rispetto a modeste distorsioni dell'immagine.

10. Utilizzo di algoritmi genetici per lo sviluppo della struttura

Il modello di rete neurale presentato finora ha una struttura relativamente complessa, che però ha due difetti:

1) E' una struttura che va progettata fin dall'inizio in relazione al problema da risolvere.
2) E' una struttura statica.

Poiché non si conosce a priori il contesto nel quale si verrà ad operare, è difficile prendere delle decisioni a riguardo, poiché non esistono metodi analitici che permettano, dato un problema, di ricavarne la migliore topologia per poterlo risolvere. Inoltre, mentre con una classica rete del tipo a retropropagazione dell'errore, l'unico aspetto di cui tener conto è la dimensione dello strato nascosto, con una rete modulare si hanno molti più parametri da considerare: il numero e il tipo di expert networks, la loro eventuale nidificazione, la presenza di meccanismi di pre e post elaborazione dei segnali come le wavelets, e così via.

Da queste considerazioni si deduce che una sua estensione potrebbe introdurre dei meccanismi che rendano adattiva la rete, in modo che a partire da una configurazione iniziale molto semplice, questa possa crescere e auto-modificarsi a seconda del contesto in cui opera.

Data la generalità delle situazioni in cui si può operare, una delle soluzioni più promettenti è quella di guidare lo sviluppo della rete tramite un algoritmo genetico. Bisogna però fare attenzione a quali parametri si decide di far evolvere, e quali si preferisce assegnare a metodi più "classici". L'osservazione di ciò che avviene in natura dà degli utili suggerimenti. I pesi sinaptici, ad esempio, non vengono modificati da un meccanismo genetico per due principali motivi: 1) il loro numero rende impossibile una qualsiasi forma di "registrazione" nel DNA, che ha una capacità limitata; 2) deve essere possibile modificare i pesi in tempo reale per

adattarsi all'ambiente circostante. Alla prima di queste cause è dovuto il fatto che, soprattutto nella fase dell'infanzia, dobbiamo apprendere in base all'esperienza.

Caratteristiche che invece si prestano a modifiche di tipo evolutivo possono essere le funzioni di attivazione dei neuroni, la modalità secondo la quale si sviluppano gli assoni (ad esempio il tipo di marcatori che ne dirigono la crescita), il livello di nidificazione dei moduli e la loro interconnessione per quanto riguarda una rete del tipo qui proposto, ecc... Quindi tutte quelle caratteristiche che dipendono da un numero relativamente limitato di parametri, anche perché si deve considerare che con questa metodologia non si ha più a che fare con un "individuo" che si sviluppa, ma si deve gestire una "popolazione" di individui che evolve, e quindi un carico computazionale notevolmente superiore.

Appendice A: **Software utilizzato**

Nello sviluppo di questo software non mi sono legato ad ambienti o librerie specifiche. Ho utilizzato il linguaggio di programmazione Standard C++ (appendice C).
Ho iniziato la stesura di una prima parte del codice sotto il sistema operativo Microsoft Windows NT 4, utilizzando l'ambiente di sviluppo Microsoft Visual C++ 6. Ho poi proseguito la scrittura del programma sotto il sistema operativo *open-source* Linux, dove, oltre al linguaggio di programmazione GNU-GCC 3.0, ho utilizzato l'ambiente di sviluppo K-Develop. Per disegnare i diagrammi UML ho impiegato il programma Dia (sempre sotto Linux).

Appendice B: **Introduzione a UML**

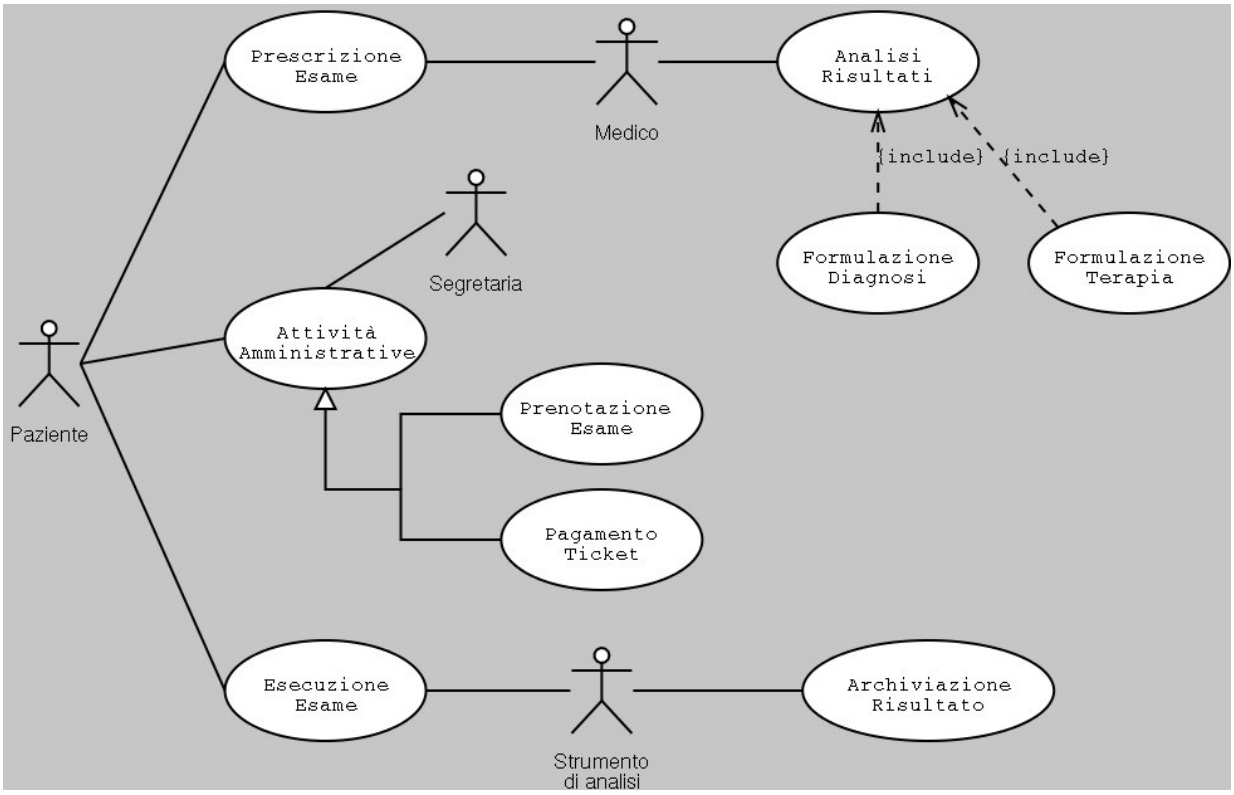
Il linguaggio "Unified Modeling Language" (UML) è un linguaggio di modellazione grafico definito nel 1997 che si ispira ed integra le metodologie precedenti creando una notazione standard adottata dall'organizzazione OMG (Object Management Group).
Lo scopo di UML è quello di descrivere un insieme di algoritmi, sistemi, processi, basi di dati,... rendendone possibile una visione da punti di vista differenti. Si hanno quindi diversi tipi di diagrammi, ognuno dei quali si presta ad una rappresentazione particolare, che mette in evidenza dettagli che in altri contesti possono essere nascosti.

Verrà adesso mostrata una sequenza di diagrammi per illustrare sinteticamente la filosofia di questo linguaggio; per un approfondimento si veda [Fowler, 2000]. L'ordine che seguirò rispecchia la sequenza di passi svolti durante il processo di sviluppo del software, anche se per progetti di media/alta complessità ci saranno dei cicli per un raffinamento iterativo di ciò che viene prodotto.

Diagrammi *Use-Case*

I diagrammi *use-case* (letteralmente "casi d'uso") rappresentano un insieme di *scenari*, cioè di sequenze di passi che descrivono l'interazione tra uno o più utenti ed un sistema. Questi utenti sono da considerarsi come entità astratte; ad esempio un utente può essere una persona così come può essere un database o un qualunque altro sistema esterno; in UML vengono chiamati "attori" (*actors*) e vengono rappresentati con la figura stilizzata di un uomo.

Di seguito mostrerò un esempio legato ad un ambiente ospedaliero, che quindi non ha niente a che vedere con le reti neurali, che ho sviluppato per un corso interno all'azienda in cui lavoro che, appunto, produce software per il settore medico.

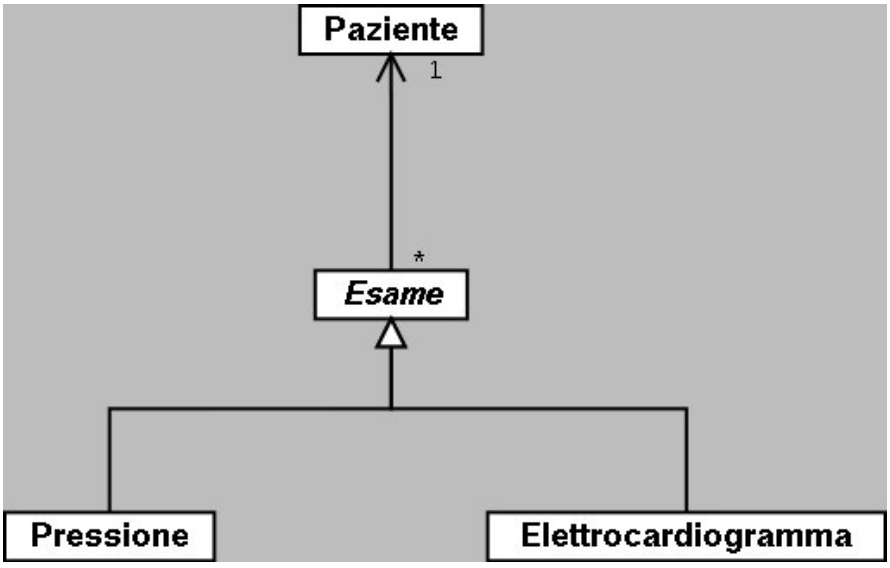


Gli ovali rappresentano delle attività svolte dal sistema. Le linee continue rappresentano delle interazioni tra gli oggetti che si trovano ai loro estremi. Le frecce tratteggiate rappresentano delle relazioni di "inclusione", mentre la freccia triangolare esprime una generalizzazione.

Quando si disegna un diagramma *use-case*, è importante tenere presente che si deve rappresentare la visione del sistema che si ha dall'esterno, e che quindi, di solito, non c'è una correlazione con la sua struttura interna (ad esempio con le classi utilizzate).

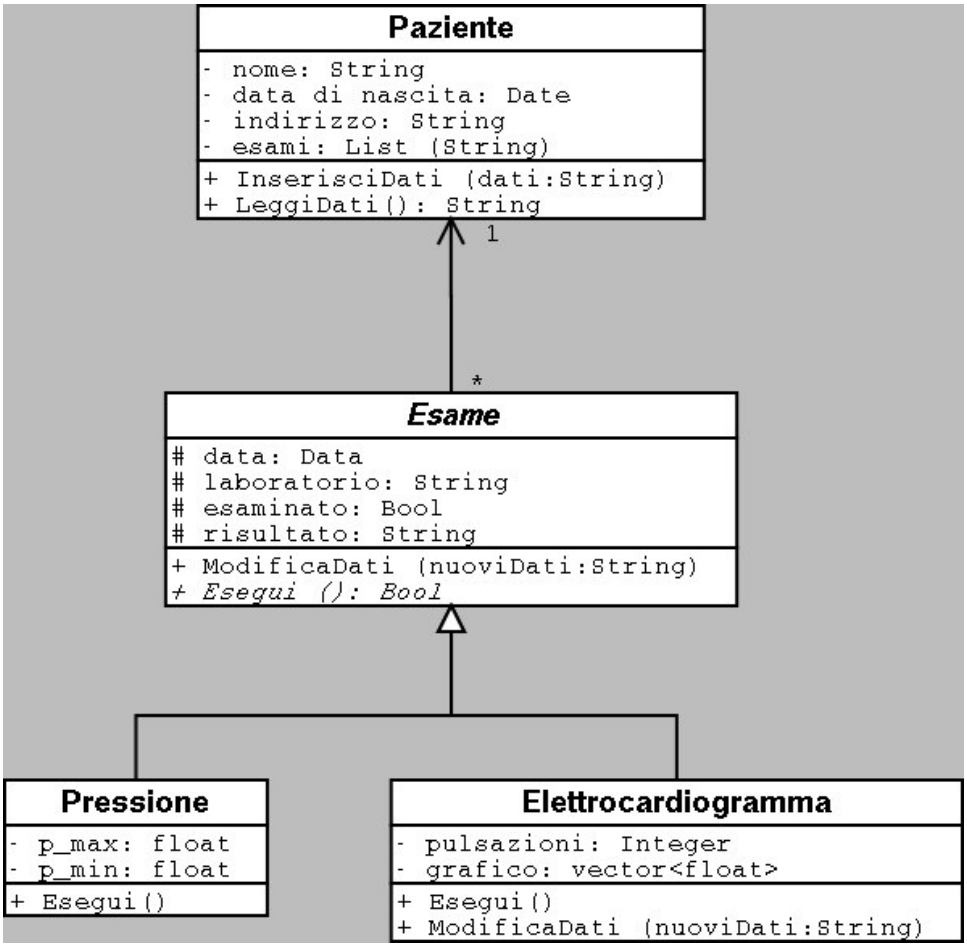
Diagrammi di classe

I diagrammi che considereremo d'ora in poi, a differenza dei diagrammi *use-case*, hanno lo scopo di rappresentare il funzionamento interno del sistema in modo diretto. I diagrammi di classe, in particolare, si rifanno al concetto di classe che si trova nei linguaggi di programmazione orientati agli oggetti.
Nel diagramma seguente, abbiamo le classi rappresentate con dei rettangoli che sono messe in relazione tra loro con delle frecce.



In questo esempio, l'1 e l'asterisco indicano che ad un paziente si possono associare più esami, e che ad un esame si può associare un unico paziente (una notazione che ricorda quella dei diagrammi entità-relazione usata nei data base). La freccia triangolare, invece, esprime la nozione di generalizzazione, implementabile ad esempio con delle classi derivate.

Una classe ha una sua struttura interna che nel diagramma precedente non è stata mostrata; se si vuole avere una visione globale, e di conseguenza approssimativa, questo è un modo di nascondere dettagli che possono essere temporaneamente trascurati. Di seguito abbiamo lo stesso diagramma rappresentato in forma completa:

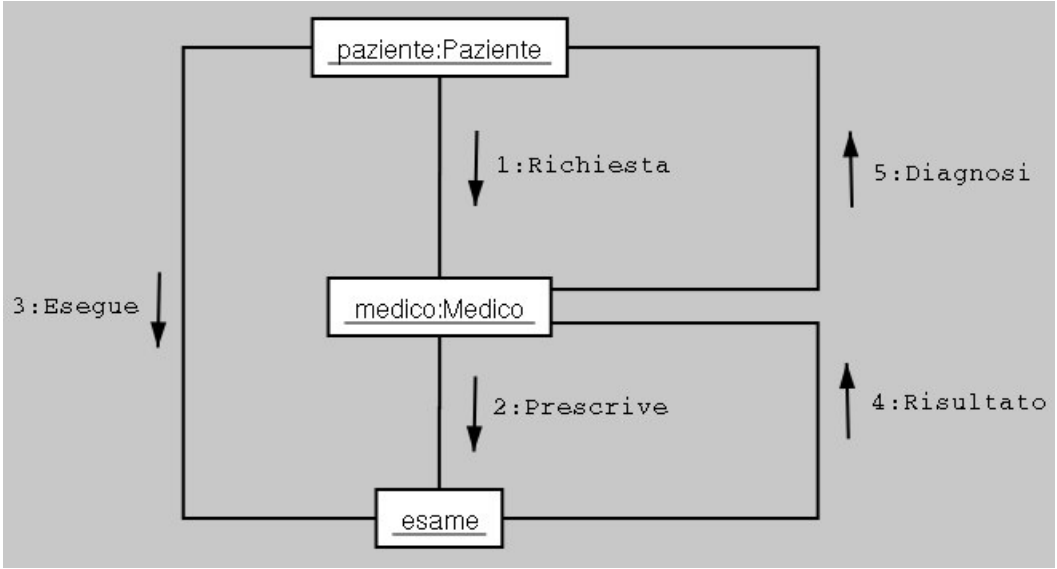


Una classe ha membri (nella parte superiore) e funzioni membro (nella parte inferiore) pubbliche (precedute dal segno +), protette (#) e private (-). Il nome della classe scritto in *corsivo* sta ad indicare che la classe e astratta, e le funzioni membro scritte in *corsivo* sono i corrispondenti metodi virtuali.

I diagrammi di classe rappresentano la situazione in modo statico; le due tipologie di diagrammi seguenti, invece, rappresentano la stessa situazione rispetto allo svolgere del tempo.

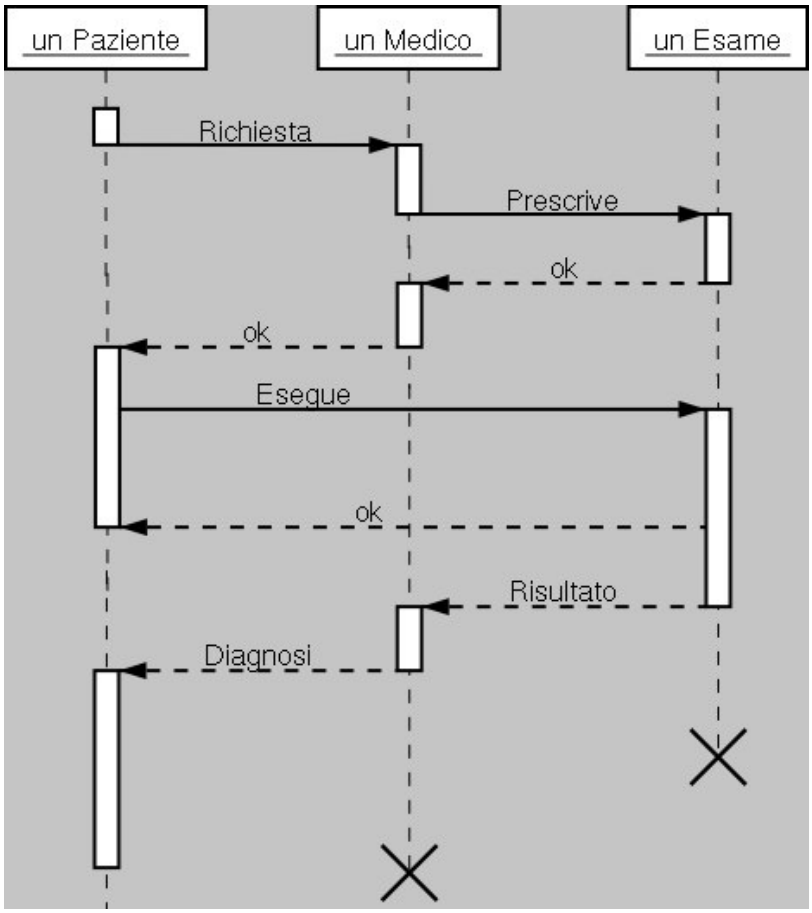
Diagrammi di collaborazione

I diagrammi di collaborazione (*collaboration diagrams*), così come quelli presentati di seguito, non mostrano le classi di cui è composto il programma, ma i loro oggetti che interagiscono nel tempo. Questi si scambiano dei messaggi attraverso le linee di comunicazione disegnate, e il numero scritto prima della loro etichetta ne permette un ordinamento.



Diagrammi di sequenza

I diagrammi di sequenza (*sequence diagrams*) rappresentano l'evoluzione nel tempo del sistema da un punto di vista più esplicito rispetto ai diagrammi del tipo precedente:



Gli oggetti sono indicati nella parte alta del diagramma e le linee tratteggiate verticali ne rappresentano la vita, che può terminare con la loro distruzione (indicata dalla X). Gli oggetti sono attivi (in esecuzione) quando la loro linea "della vita" è coperta dai rettangoli bianchi (si può notare che il sistema descritto lavora in multitasking, poiché l'oggetto Paziente e l'oggetto Esame sono attivi contemporaneamente per un certo intervallo di tempo).

Appendice C: **Note riguardo lo Standard C++**

(ISO/IEC 14882)

Lo Standard C++, definito e standardizzato in ambito ISO nel 1997, rappresenta una estensione del linguaggio di programmazione ad oggetti C++. La novità che ho trovato più rilevante risiede nelle librerie che fanno parte dello standard. Queste permettono uno stile di programmazione ad un livello ben più alto di quanto non fosse possibile fare in precedenza, e al tempo stesso mantengono delle buone prestazioni, in quanto il carico di lavoro aggiuntivo ricade principalmente sulla fase di compilazione.

Per un testo completo sull'argomento si veda [Stroustrup, 1997], che ha costituito un riferimento per la definizione del nuovo standard.

Tra le novità introdotte, quella che ho maggiormente utilizzato è stata quella dei *containers*, in particolare le classi **vector** e **list**. Queste permettono di fare in modo molto più elegante ciò che una volta richiedeva l'uso esplicito dei puntatori.
Un oggetto di classe vector ha una dimensione dinamica; quando si eccede lo spazio inizialmente allocato, tutta la struttura dati viene trasferita in un'altra area di memoria libera sufficientemente grande; poiché una tale operazione è costosa in termini di tempo, per evitare che venga ripetuta troppo spesso è possibile specificare quanto spazio extra si desidera venga allocato. D'altra parte un oggetto di classe list, pur avendo molti punti in comune con la classe vector, sfrutta in modo differente la memoria: in questo caso gli elementi della lista non sono vincolati a risiedere in un'area di memoria contigua, quindi non c'è riallocazione.
L'efficienza in termini di velocità di esecuzione e di memoria occupata varia a seconda dell'algoritmo che si sta implementando e delle classi utilizzate.

L'esempio nella pagina seguente mostra un tipico utilizzo di una simile classe:

```
#include <vector>
#include <iostream>

using namespace std;          // per l'utilizzo della Standard Library

class  MyClass
{
public:
   int a;
};

void  main (void)
{
   vector<MyClass>  vec;
   vector<MyClass>::iterator  it;

   MyClass  mc;

   for (int i=0; i<10; i++)         // inserisce 10 elementi in vec
   {
      mc.a = i;
      vec.push_back (mc);           // vec cresce dinamicamente
   }

   for (it = vec.begin(); it != vec.end(); it++)
   {
      cout << it->a << endl;     // uso un iteratore per accedere
                                 //  agli elementi.
   }

   cout << endl << "Dimensione:   " << vec.size() << endl;
   cout << "Capacita':   " << vec.capacity() << endl;
   cout << "Dimensione massima:  " << vec.max_size() << endl << endl;

}
```

Bibliografia

Chen, K., Xu, L., Chi, H. (1999). "Improved learning algorithms for mixture of experts in multiclass classification", *Neural Networks, 12*, 1229-1252.

Doya, K. (1999). "What are the computations of the cerebellum, the basal ganglia and the cerebral cortex?", *Neural Networks, 12*, 961-974.

Fowler, Martin (2000). "UML Distilled" 2nd ed., Addison-Wesley.

Haykin, Simon (1999). "Neural Networks, a Comprehensive Foundation", Prentice Hall.

Hinton, G., Sejnowski, T., ed. (1999). "Unsupervised Learning", MIT Press.

Hughes, Cameron, et al. (1999). "Standard C++ Classes", Wiley.

Jacobs, R.A., Jordan, M.I., Nowlan, S. J., Hinton, G. E. (1991). "Adaptive Mixtures of Local Experts", *Neural Computation, 3*, 79-87.

Jordan, M.I., Jacobs, R.A. (1994). "Hierarchical Mixtures of Experts and the EM Algorithm", *Neural Computation, 6*, 181-214.

Jordan, M.I., Xu, L. (1995). "Convergence results for the EM approach to Mixture of Experts architectures", *Neural Networks, 8*, 1409-1431.

Kandel, Schwartz, Jessell (1991). "Principles of Neural Science", 3rd ed., Prentice Hall.

Koch, C., Segev, I., eds., "Methods in Neuronal Modeling, from Ions to Networks", 2nd ed., 1998 MIT Press.

Mele, Pietro (1997). "Reti Neurali Modulari", *seminario tenuto presso il C.N.R.*

Osherson, D.N., Weinstein, S., Stoli, M., (1990). "Modular Learning", *Computational Neuroscience*, MIT Press.

Quarteroni, A. (1994). "Elementi di Calcolo Numerico", Esculapio.

Sato, M., Ishii, S. (2000). "On-line EM Algorithm for the Normalized Gaussian Network", *Neural Computation 12,* 407-432.

Schapire, R.E. (1990). "The strength of weak learnability", *Machine Learning*, *5*, 197-227.

Stroustrup, Bjarne (1997). "The C++ Programming Language", 3rd ed., Addison-Wesley.

Ueda, N., Hinton, G., et al (2000). "SMEM Algorithm for Mixture Models", *Neural Computation, 12*, 2109-2128.

Xu, L., Jordan, M.I. (1996). "On convergence properties of the EM algorithm for Gaussian Mixtures",

Neural Computation, 8, 129-151.