Relazione per l'esame di metodi computazionali per la fisica (C++): Riconoscimento di e-mail spam utilizzando reti neurali

Roberto Bondesan

11 febbraio 2008

1 Introduzione

Le e-mail sono al giorno d'oggi uno dei mezzi comunicativi piú comuni. Purtroppo, tra le e-mail che riceviamo quotidianamente vi é una considerevole quantitá di mail indesiderate, il cosidetto spam, il cui scopo va dalla pubblicitá a veri e propri tentativi di truffa.
É presente un certo numero di servizi e software che i server e-mail e gli utenti possono utilizzare per ridurre il carico di spam sui loro sistemi e caselle di posta. Alcuni di questi contano sul rifiuto dei messaggi provenienti dai server conosciuti come spammer (bloccaggio). Altri analizzano in modo automatico il contenuto dei messaggi e-mail ed eliminano quelli che somigliano a spam (filtraggio). Questo secondo approccio spesso si avvale di tecniche di apprendimento del software. In questo quadro si inserisce quindi la possibilitá di usare le reti neurali per un controllo intelligente della propria casella di posta.

In questa relazione sará illustrato come utilizzare le reti neurali per riconoscere lo spam e saranno riportati gli algoritmi usati e i risultati ottenuti. Il codice, ad eccezione del frontend grafico, é scritto in C++, utilizzando strumenti della Standard Template Library.

2 L'algoritmo

L'algoritmo qui presentato si basa sull'assunzione che guardando l'oggetto di una mail sia possibile distinguere lo spam dalle mail buone. Inizialmente si cercano nell'header di una e-mail le parole utilizzate dal mittente come

oggetto e si stila una statistica delle parole piú significative, in base alla probabilitá che una parola compaia in una mail spam o in una mail buona. Per parole qui si intende una qualunque sequenza contigua di lettere, e né l'ordine delle parole, né il significato semantico sono stati considerati. Dalle parole verrano poi estratte le informazioni su cui adattare i parametri nella fase di apprendimento della rete neurale. Il training della rete puó essere fatto in poco tempo e con poco sforzo da parte dell'utente.

2.1 Preparazione dei dati

Il primo problema che si é dovuto affrontare é quello di come estrarre da una serie di parole lette nel campo *Subject:* dell'header delle mail (per un esempio vedi Fig. 1), i dati di input della rete neurale.

```
Received: (qmail 26862 invoked by alias); 11 Jan 2008 17:15:47 -0000
Delivered-To: staff@pcteor1.mi.infn.it
Received: (qmail 26854 invoked from network); 11 Jan 2008 17:15:47 -0000
Received: from federation-hq.pcteor1.mi.infn.it (HELO pcteor1) (192.168.0.1)
  by xenon2.pcteor1.mi.infn.it with SMTP; 11 Jan 2008 17:15:47 -0000
Received: (qmail 13464 invoked from network); 11 Jan 2008 18:15:47 +0100
Received: from unknown (HELO smtp2.mi.infn.it) (192.84.138.106)
  by pcteor1.mi.infn.it with SMTP; 11 Jan 2008 18:15:47 +0100
Received: from maersk.com (cable-static-82-59.intergga.ch [157.161.82.59])
        by smtp2.mi.infn.it (8.14.1/8.11.2) with ESMTP id m0BHFjvp008387;
        Fri, 11 Jan 2008 18:15:47 +0100
Sender: <o.deckerpk@hammersmithresearch.com>
In-Reply-To: <912101c84f5e$b32818d8$e42ad560@camroz1>
X-Sender: <o.deckerpk@hammersmithresearch.com>
Reply-To: "Otis Decker" <o.deckerpk@hammersmithresearch.com>
Subject: Look Rich withRolexOmegaBreitling, 90% Off Original Price 4dxdyh3po7sre3k5lfy
Message-ID: <1200078194.6300@hammersmithresearch.com>
From: "Otis Decker" <o.deckerpk@hammersmithresearch.com>
Date: Fri, 11 Jan 2008 12:03:14 -0700
To: ktfr@pcteor1.mi.infn.it
```

Figura 1: Header di una e-mail spam. É evidenziato il *Subject:*, il campo da cui vengono raccolte le parole per la statistica.

Lo si é risolto considerando l'input della rete come una lista di parole presenti o meno in una mail; i dati di input si presenteranno dunque come una matrice booleana

$$A_{ij} = \begin{cases} 1 & \text{se nella mail } i \text{ vi é la parola } j \\ 0 & \text{altrimenti} \end{cases}$$

Per limitare la dimensione della rete, si é costruito un dizionario con le n parole piú significative, basato sulla statistica del set di mail di training. Inizialmente si creano un vettore di occorrenze delle parole per lo spam e per le mail buone. Poi da questo di calcola la probabilitá che una data parola w sia presente nello spam P_{junk}, o meno, P_{good} (vedi tabella 1). Si introduce

la misura $d = |P_{junk} - P_{good}|$, e le prime n parole ordinate in base a d, si considerano come le n parole piú rilevanti.

Parole	P_{good}	P_{junk}
Re:	0.0828331	0.00720288
your	0	0.027611
urgente	0.0168067	0
Xorg	0.0168067	0

Tabella 1: In questa tabella sono riportati esempi di parole con la relativa probabilitá che siano presenti in una e-mail spam P_{junk} o in una e-mail buona P_{good}.

Riassumendo, i punti principali dell'algoritmo della fase di preparazione dei dati di input della rete sono:

- `Creo un vettore di parole contenute nell'oggetto delle mail, distinguendo se una parola compare nello spam o no`
- `Calcolo la probabilita' che una parola sia in una mail spam e quella che sia in una mail buona`
- `Sullo spazio delle parole introduco la distanza` $d = |P_{junk} - P_{good}|$
- `Considero le n parole con distanza maggiore come le piu' significative e scrivo un file di input per le rete`

2.2 Apprendimento della rete neurale

Una rete neurale é una rete costituita da neuroni interconnessi. Oltre allo scopo biologico di modellizzare il funzionamento del cervello umano, esse vengono utilizzate in informatica come sistema adattivo che cambia la sua struttura in base a stimoli esterni o informazioni interne che viaggiano attraverso la rete. In pratica le reti neurali sono uno strumento efficace per parametrizzazioni non lineari di set di dati. Nella realizzazione del filtro anti-spam, queste verranno utilizzate per la classificazione delle mail, e una volta istruite, per predire il carattere di una mail.

Si é utilizzato il multilayer perceptron, una rete neurale artificiale senza feedback, che usa tre o piú layer di neuroni con funzione di attivazione

tipicamente sigmoide[1]:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Il valore che assume il neurone i–esimo appartenente al layer l, ξ_i^l, é dato dalla funzione di attivazione g valutata sulla somma con pesi ω_{ij} dei valori dei neuroni nel layer precedente:

$$\xi_i^l = g\left(\sum_{j=1}^{n_{l-1}+1} \omega_{ij}^{l-1}\xi_i^{l-1}\right) \tag{2}$$

La somma arriva fino al numero di nodi del layer precedente piú uno, poiché il nodo in piú é il threshold, che viene trattato come un neurone di valore 1.

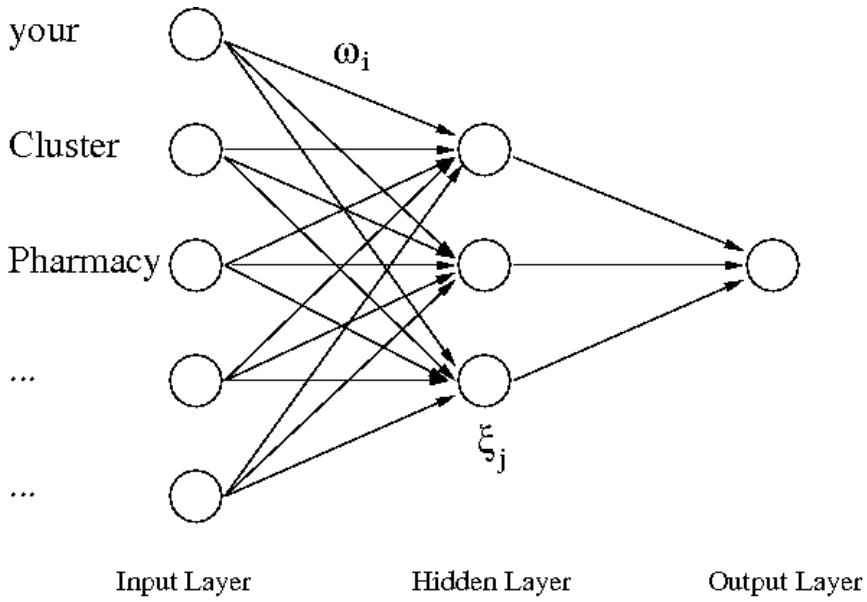

Figura 2: Schema della rete neurale utilizzata. Ogni nodo del layer di input corrisponde a una parola utilizzata per il training

In figura 2 si mostra la struttura della rete neurale usata. In input vi sono i dati relativi a una data mail: quando viene presentata una mail alla rete, tutti gli input corrispondenti a parole presenti nel messaggio di posta avranno valore 1, mentre tutti gli altri input sono posti a 0. La mail viene classificata come spam se il valore di output é maggiore di 0.5.

Durante la fase di apprendimento della rete l'output é messo uguale a 1 per le mail spam, a 0 per le mail buone. Si é controllato anche che il numero

[1]Si é comunque lasciata la possibilitá all'utente del programma di scegliere una funzione di attivazione lineare.

di mail buone e quello di spam scelti per l'apprendimento fossero circa uguali e in ordine sparso.

La capacitá di apprendimento delle reti neurali é una delle loro caratteristiche piú affascinanti. Dato un compito da svolgere e una classe di funzioni F, apprendere significa utilizzare una serie di osservazioni che permettano di trovare una funzione f^* che risolva il compito nel miglior modo. In pratica ció implica definire una funzione di errore $C : F \to \mathbb{R}$ che rappresenti il costo della scelta di una particolare funzione f, per cui valga $C(f^*) \leq C(f), \forall f \in F$. La funzione di errore per applicazioni che dipendono dai dati é solitamente il χ^2:

$$\chi^2 = \frac{1}{N} \sum_{i=0}^{N} \left(\frac{f(x_i) - y_i}{\sigma} \right)^2 \tag{3}$$

con (x_i, y_i) coppie di dati di input e di output, σ errore sperimentale o accuratezza numerica richiesta. Nel progetto si é utilizzata questa funzione di errore, con $\sigma = 0.4$.

Tra le tipologie di apprendimento si é scelto un apprendimento supervisionato, che utilizza un insieme di dati per l'addestramento comprendente esempi tipici d'ingressi con le relative uscite corrispondenti: in tal modo la rete puó imparare a inferire la relazione che li lega. La rete viene addestrata con un algoritmo genetico, che permette di modificare i pesi delle connessioni tra i neuroni in modo tale che si minimizzi la funzione di errore.

L'algoritmo genetico è un un algoritmo stocastico e puó essere riassunto in questo modo:

1 `Dall'array dei pesi iniziali della rete neurale, vengono creati dei cloni che costituiscono la popolazione iniziale.`

2 `Per ogni clone vengono generati dei mutanti, cloni che hanno alcuni elementi variati in modo random`

3 `Per ogni individuo della popolazione cosi' ottenuta, viene calcolata la funzione di errore, e si selezionano gli individui migliori, che saranno il nuovo set di cloni`

4 `Ripeto i punti 2 e 3 fino a che non viene soddisfatto un criterio di stopping`

I dati di input vengono partizionati in due set, uno di training e uno di validation. L'algoritmo genetico viene applicato al primo set, mentre il secondo viene utilizzato per confermare e validare l'analisi iniziale. Una maggiore frazione di dati nell'insieme di validation migliorerá la capacitá di generalizzazione, mentre un maggior numero in quello di training sará a vantaggio della rappresentazione della rete neurale. Se l'apprendimento é stato effettuato troppo a lungo o se vi é uno scarso numero di esempi di allenamento, il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del set di training, comportando un peggioramento sui dati non visionati (overfitting). Dividendo i dati in due set, per evitare l'overfitting basterá un controllo di stopping che controllerá la crescita della funzione di errore per il set di validation e la decrescita di quello di training. Viene anche usato un valore massimo per il set di training χ^2_{max}, sotto al quale viene fatto il controllo di stopping dinamico.

Questo metodo unito al controllo dell'errore sulle ultime iterazioni (smearing) permette di evitare l'overfitting e i casi non significativi in cui l'errore oscilla. In figura 3 é mostrato l'andamento dell'errore in funzione delle iterazioni.

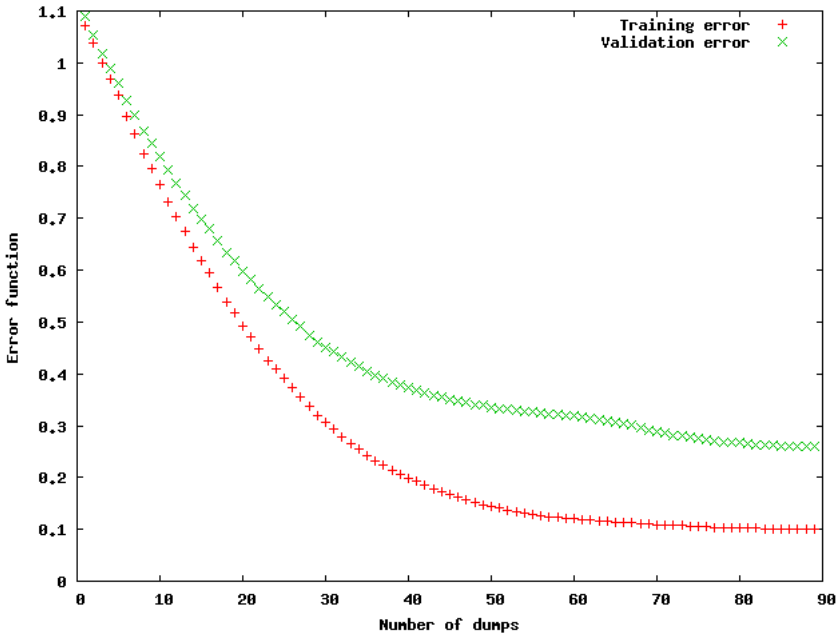

Figura 3: Andamento dell'errore per il set di training (rosso) e per quello di validation (verde) in funzione del numero di iterazioni.

Utilizzando l'algoritmo genetico si é notato che i parametri come il mutation rate o il numero di mutanti, si sarabbero potuti adattare all'andamento

del fit per ottenere migliori risultati. Infatti quando si é vicini al minimo dell'errore, il numero di mutanti dovrebbe aumentare e il mutation rate dovrebbe diminuire poiché serve piú precisione. Si é cosí implementato un metodo che nel caso l'errore saturi, agisce sul numero di mutanti moltiplicandolo per l'inverso dell'errore, in modo da aumentarlo, e rimpicciolisce il mutation rate. Effettivamente si é notato che questa scelta era buona e in molti casi faceva sí che il fit non saturasse.

2.3 Controllo delle nuove mail

Una volta che si sono fissati i parametri della rete neurale durante la fase di apprendimento, si puó creare una nuova rete con questi pesi e sottoporle nuovi dati di input per verificare la generalizzazione. Il controllo viene fatto solo su nuove mail per cui almeno una delle parole utilizzate per l'apprendimento é presente. Si é visto che questo atteggiamento non é riduttivo, come si puó notare dai risultati riportati in seguito.

3 Interfaccia grafica

Il software sviluppato puó essere compilato e eseguito dall'utente anche utilizzando un frontend grafico (figura 4).

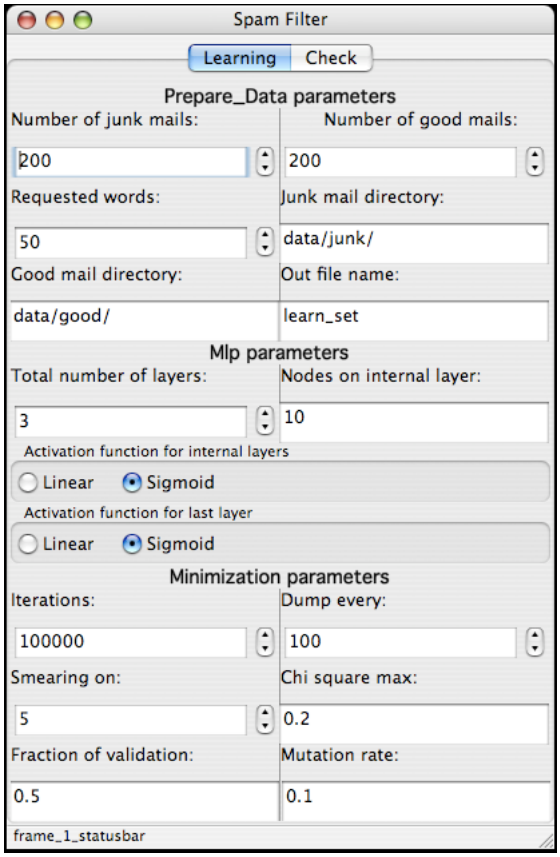

Figura 4: Screenshot dell'interfaccia grafica. É stata realizzata in python, usando le librerie grafiche wx.

L'interfaccia grafica é scritta in python e per il suo sviluppo si é utilizzato wxglade, un costruttore di GUI che usa le librerie grafiche wxPython. Come si vede in figura, viene data la possibilitá all'utente di modificare alcuni parametri sia della fase di apprendimento (tab 1, Learning), che di quella di controllo su nuove mail (tab 2, Check). La menubar permette di compilare (Make), eseguire il codice (Execute), fermare una esecuzione (Stop), monitorare l'andamento dell'errore (Plot), e consultare la documentazione del codice[2] (Contents).

[2]Generata con Doxygen

4 Debug e profiling del codice

Il codice é stato sviluppato su *linux*, e testato su *Mac OSX* con gli stessi risultati. Per testare e debuggare il software si sono usate degli strumenti standard in ambiente linux: gdb e gprof, il debugger e profiler della GNU, e valgrind, un tool per controllare l'uso della memoria. L'esecuzione della fase di training della rete neurale richiede tipicamente pochi minuti, e il controllo delle mail nuove, meno di un secondo. Ad esempio l'esecuzione[3] del programma con i dati del run 5 della tabella 4 presentata nel prossimo paragrafo, richiede 1 minuto e 2.936 secondi di tempo reale, mentre solo 0.581 secondi di CPU in system mode (spesi cioé quando il kernel esegue operazioni per conto del processo).

Piú sotto sono riporate le prime righe del profile[4] per la medesima esecuzione.

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 51.62     13.88    13.88 13488556     0.00     0.00  Mlp::output(
std::vector<double, std::allocator<double> >)
 28.97     21.67     7.79      630     0.01     0.04  Minimization
::genetic_algorithm(Data const&, Parametrization*&)
  8.54     23.97     2.30 134893567     0.00     0.00  std::vector
<double, std::allocator<double> >::_M_insert_aux(__gnu_cxx::
__normal_iterator<double*, std::vector<double, std::allocator<double>
 > >, double const&)
  7.77     26.06     2.09        1     2.09     2.09  Pr_Learn_Data::
sort_words()
```

Dal profile si nota che metá del tempo di esecuzione viene speso nella funzione *Mlp::output*, seguita da *Minimization::genetic_ algorithm*. Un utilizzo considerevole del tempo di CPU viene fatto anche da *_M_ insert_ aux* della classe *vector* e da *Pr_ Learn_ Data::sort_ words*, quest'ultima chiamata una sola volta[5]. Possibile ottimizzazioni delle prestazioni del codice potrebbero riguardare proprio l'algortimo di ordinamento delle parole e rinunciare alla comoditá di usare la classe *vector*, per un programma piú veloce.

[3] Il test é stato svolto su un computer con processore Intel Core 2 Duo

[4] La prima colonna indica la percentuale di tempo usata dalla routine, la seconda i secondi totali spesi nella funzione e in quelle superiori, la terza i secondi spesi nella sola funzione, la quarta il numero di chiamate, la quinta e la sesta il numero medio di millisecondi per chiamata spesi per la sola funzione (self) e anche per le discendenti (total) e infine l'ultima colonna é il nome della routine.

[5] Il profile é relativo a un set di dati ristretto, mediamente il tempo speso in *sort_ words* pur rimanendo rilevante, é meno significativo nel complesso.

Infine si é usato valgrind per controllare l'uso della memoria e questo non ha rivelato memory leak né errori.

5 Risultati

In questa sezione presento i risultati di alcuni test fatti sia per la fase di apprendimento della rete che per quella di controllo della capacitá di generalizzazione su nuove mail. In tutti i run riportati si é usata una rete neurale con uno strato nascosto composto da 7 nodi. Per la minizzazione si sono usate 100000 iterazioni massime, 10 iterazioni di dump e 10 di controllo dell'errore nel caso di saturazione, e 5 iterazioni di dump per lo smearing. La frazione di validation usata é 0.8, per avere una migliore capacitá di generalizzazione della rete.

Run	mutsize	mutrate	mutmul	stop	ratio tr	ratio val
1	100	0.1	2	1020	0.981	0.931
2	200	0.1	2	840	0.981	0.937
3	100	0.01	2	9960	0.981	0.931

Tabella 2: In questa tabella sono riportati i risultati relativi alla fase di training della rete con 50 input al variare dei parametri dell'algoritmo genetico (mutsize = numero di mutanti, mutrate = rate di mutazione, mutmul = numero di mutazioni in un mutante). Stop é l'iterazione a cui si é fermata la dinamica, ratio tr e ratio val, sono il numero di risposte giuste sulle totali per i set di training e di validation. Per istruire la rete si sono usate 250 mail spam e 250 mail buone tra cui se ne sono scelte 342 con le 50 parole piú significative. Il numero di parametri é 365. Si é usato $\chi^2_{max} = 0.1$.

Nella tabella 2 sono riportati i risultati della fase di apprendimento della rete neurale nel caso di 50 parole di input, per diversi valori dei parametri dell'algoritmo genetico. Si vede che in tutti e tre i casi la capacitá di rappresentazione della rete é molto buona. Utilizzare un numero di mutanti grande (run 2) riduce il numero di iterazioni per raggiungere l'errore desiderato, mentre incominciare la minimizzazione con un mutation rate piccolo (set 3) implica un numero elevato di iterazioni, poiché nella fase iniziale é molto probabile che si parta in un punto nello spazio dei parametri distante da quello di minimo, e utilizzando un passo piccolo, il tempo per raggiungere il minimo é elevato.

I risultati della fase di controllo relativi ai run con 50 parole mostrati in tabella 3 mostrano che su 200 mail, 112 risultano contenenti le 50 parole usate per il training, e in questi casi la percentuale di risposte positive é

Run	Mail controllate	Risposte positive	Ratio
1	112	98	0.875
2	112	98	0.875
3	112	98	0.875

Tabella 3: In questa tabella sono riportati i risultati relativi alla fase di controllo su nuove mail della rete con 50 input. Le mail di controllo sono scelte tra 100 mail buone e 100 spam, in base al numero di parole su cui é stata allenata la rete.

87.5%. La rete neurale é cosí in grado di generalizzare e predire il risultato nella maggior parte dei casi.

Run	mutsize	mutrate	mutmul	stop	ratio tr	ratio val
4	100	0.1	2	1440	0.990	0.889
5	200	0.1	10	660	0.990	0.899

Tabella 4: In questa tabella sono riportati i risultati relativi alla fase di training della rete con 80 input al variare dei parametri dell'algoritmo genetico (mutsize = numero di mutanti, mutrate = rate di mutazione, mutmul = numero di mutazioni in un mutante). Stop é l'iterazione a cui si é fermata la dinamica, ratio tr e ratio val, sono il numero di risposte giuste sulle totali per i set di training e di validation. Per istruire la rete si sono usate 350 mail spam e 350 mail buone tra cui se ne sono scelte 508 con le 80 parole piú significative. Il numero di parametri é 575. Si é usato $\chi^2_{max} = 0.1$.

Come si vede nella tabella 4, nel caso di 80 parole di input la capacitá di rappresentazione della rete é quasi assoluta. Come nel caso precedente utilizzare un numero di mutanti grande e un numero di mutazione multiple superiore (run 5) riduce il numero di iterazioni per raggingere l'errore desiderato. Rispetto al caso con 50 input, il numero di mail su cui la rete riesce

Run	Mail controllate	Risposte positive	Ratio
4	118	109	0.924
5	118	107	0.907

Tabella 5: In questa tabella sono riportati i risultati relativi alla fase di controllo su nuove mail della rete con 80 input. Le mail di controllo sono scelte tra 100 mail buone e 100 spam, in base al numero di parole su cui é stata allenata la rete.

a dare risposta é ovviamente aumentato, e la percentuale di risposte positive é aumentata, ed é leggermente superiore per il run 5.

Run	mutsize	mutrate	mutmul	stop	ratio tr	ratio val
6	200	0.1	10	830	0.985	0.891
7	200	0.1	10	1030	0.987	0.909

Tabella 6: In questa tabella sono riportati i risultati relativi alla fase di training della rete con 100 (run 6) e 150 (run 7) input, al variare dei parametri dell'algoritmo genetico (mutsize = numero di mutanti, mutrate = rate di mutazione, mutmul = numero di mutazioni in un mutante). Stop é l'iterazione a cui si é fermata la dinamica, ratio tr e ratio val, sono il numero di risposte giuste sulle totali per i set di training e di validation. Per il run 6 si sono usate 400 mail spam e 400 mail buone tra cui se ne sono scelte 605 con le 100 parole piú significative. Il numero di parametri é 715. Per il run 7 le mail per il training usate sono 1064 scelte tra 705 spam e 705 buone, e i parametri della rete sono 1065. Si é usato $\chi^2_{max} = 0.1$.

Nella tabella 6 sono riportati i risultati della fase di apprendimento della rete neurale nel caso di 100 parole di input (run 6) e 150 parole (run 7). Anche in questi casi la rete rappresenta bene i dati. I risultati della fase di

Run	Mail controllate	Risposte positive	Ratio
6	121	112	0.926
7	131	120	0.916

Tabella 7: In questa tabella sono riportati i risultati relativi alla fase di controllo su nuove mail della rete con 100 e 150 input. Le mail di controllo sono scelte tra 100 mail buone e 100 spam, in base al numero di parole su cui é stata allenata la rete.

controllo relativi al run 6 (tabella 7) mostrano che su 200 mail, 121 risultano contenenti le 100 parole usate per il training, e in questi casi la percentuale di risposte positive é intorno all'93%. Per il run 7 si hanno 120 risposte positive su 131.

6 Conclusioni

Il software sviluppato mostra come sia possibile utilizzare le reti neurali per separare le mail volute da quelle indesiderate. Le risposte della rete a questo compito sono soddisfacenti, raggiungendo il 99% in rappresentazione dei dati di training e il 95% del set di validation. Si é dimostrata poi l'effettiva possibilitá di utilizzare la reti istruita come un filtro antispam abbastanza

efficiente, che riesce a riconoscere all'incirca il 90% delle mail su cui esegue un controllo.

Sicuramente vi é spazio per miglioramenti e ottimizzazioni del software, come ad esempio una scansione di altri campi delle mail oltre al soggetto, o come l'utilizzo di altri metodi di apprendimento piú adatti a classificare dei dati, come le *Support Vector Machine*. Si potrebbe unire al training della rete una white–list ed una black–list di sender, con delle regole definite dall'utente, del tipo "considera non–spam tutte le mail dall'indirizzo pippo@gmail.com". Un possibile utilizzo del software potrebbe essere di eseguire la parte di apprendimento della rete su un server che risponda alle richieste dei client sulla natura delle loro mail.