

Relazione progetto

“Riconoscitore di caratteri”

Corso di “Sistemi Intelligenti”

Laurea specialistica in Ingegneria Informatica

Anno Accademico: 2008/2009

Mario Bambagini, 291410

Università degli studi di Pisa

Indice generale

Relazione progetto.....	1
OBIETTIVO.....	4
PIANIFICAZIONE.....	4
REALIZZAZIONE.....	4
1. Dimensionamento della rete.....	4
2. Scelta e realizzazione del data set.....	6
2.1 Lettera A.....	6
2.2 Lettera B.....	6
2.3 Lettera C.....	6
2.4 Lettera D.....	6
2.5 Lettera E.....	7
2.6 Lettera F.....	7
2.7 Lettera G.....	7
2.8 Lettera H.....	7
2.9 Lettera I.....	7
2.10 Lettera J.....	8
2.11 Lettera K.....	8
2.12 Lettera L.....	8
2.13 Lettera M.....	8
2.14 Lettera N.....	8
2.15 Lettera O.....	8
2.16 Lettera P.....	9
2.17 Lettera Q.....	9
2.18 Lettera R.....	9
2.19 Lettera S.....	9
2.20 Lettere T.....	9
2.21 Lettere U.....	9
2.22 Lettera V.....	10
2.23 Lettera W.....	10
2.24 Lettera X.....	10
2.25 Lettere Y.....	10
2.26 Lettera Z.....	10
3. Addestramento della rete neurale backpropagation.....	11
4. Addestramento con disturbo.....	16
5. Addestramento della rete neurale RBF.....	20
6. Addestramento con disturbo.....	22
7. Confronto dei risultati.....	25
8. Realizzazione della gui.....	26
APPENDICE.....	27
Appendice A: Risultati della prima fase di test.....	27
Appendice B: File del progetto.....	29
Appendice C: ErrorFun.m.....	31
Appendice D: Calcolo spread minimo.....	32
Appendice E: Dati addestramento backpropagation con rumore.....	33

OBIETTIVO

L'obiettivo del progetto consiste nel realizzare, in Matlab, un riconoscitore di caratteri disegnati a mano libera.

Questo software si deve basare su reti neurali backpropagation e RBFN (Radial Basis Function Network).

Oltre al riconoscimento normale deve prevedere il riconoscimento con disturbo.

Realizzare una gui che permetta di utilizzare le reti addestrate.

PIANIFICAZIONE

Ho diviso il problema in otto fasi consecutive:

1. Dimensionamento della rete;
2. Scelta e realizzazione del data set;
3. Addestramento della rete neurale backpropagation;
4. Addestramento con disturbo;
5. Addestramento della rete neurale RBF;
6. Addestramento con disturbo;
7. Confronto dei risultati;
8. Realizzazione della gui.

REALIZZAZIONE

1. Dimensionamento della rete

La rete accetta in ingresso un array di 35 elementi con valori uguali a 0 o a 1 (nero o bianco).

Questa struttura dati rappresenta la vettorizzazione di una matrice 7x5, ottenuta ridimensionando e convertendo in bianco e nero, l'immagine originale.

Durante il dimensionamento ho cercato di:

- ridurre il più possibile i neuroni dello strato di ingresso;
- individuare tutto ciò che non è intrinsecamente legato al carattere per inserirlo nella fase di preprocessing dell'immagine;
- utilizzare prevalentemente funzioni Matlab per scrivere meno codice.

La conversione in bianco e nero è utile perché evita alla rete di imparare anche i colori, che sostanzialmente non introducono informazione utile sul carattere.

L'unico vincolo che nasce è che la lettera deve essere scritta con colori scuri (possibilmente nero) su sfondo chiaro (possibilmente bianco).

Questo vincolo potrebbe essere superato nella fase di preprocessing dell'immagine, ma non ho realizzato questa funzione dato che va al di là degli obiettivi.

Ho fatto la scelta di un fattore di scala 7x5 perché, in genere, è quello usato per rappresentare le lettere e permette una corretta visualizzazione.

Questa scelta ha come conseguenza di disegnare i caratteri con dimensioni multiple di 7x5.

Una codifica su 35 pixel è quella che mi ha dato il trade off migliore tra numero di ingressi e rappresentatività.

Un numero minore di pixel comporta una perdita di informazione notevole.

Un numero superiore aggiunge informazioni sostanzialmente superflue.

Una codifica con dimensioni doppie, 14x10, comporterebbe 140 neuroni nello strato di ingresso, un numero quattro volte superiore alla mia scelta.

Uno svantaggio di un notevole rimpicciolimento è che le linee, nella figura originale, devono avere uno spessore minimo.

Questo particolare viene integrato nel vincolo che le linee devono avere le stesse dimensioni di quelle del data set.

Su *gimp* ho usato *circle 13* con scala 0.8 per immagini 70x50.

Questo vincolo potrebbe essere superato nella fase di preprocessing dell'immagine, ma non ho realizzato questa funzione dato che va al di là degli obiettivi (vedere *imdilate*).

Effettuo il ridimensionamento esclusivamente con funzioni Matlab perché ho riscontrato che, utilizzando un fattore di scala intero, ottiene risultati molto soddisfacenti.

Un ulteriore vincolo è quello di disegnare la lettera nell'intera area di lavoro, senza lasciare righe e/o colonne completamente bianche.

Ho utilizzato alcune funzioni di Matlab per automatizzare questa operazione.

Come ulteriore vincolo impongo che, nel limite del possibile, le lettere non devono avere segmenti eccessivamente mozzati.

Di seguito è riportata un esempio di 'K' non lecito, perché le barre oblique sono sproporzionate.



Per quanto riguarda l'uscita, la codifico su 26 bit, ognuno dei quali è associato ad una lettera. Il neurone con valore maggiore è considerato il vincitore.

2. Scelta e realizzazione del data set

Dall'analisi precedente deduco che in una rete feed forward, ogni neurone nello strato nascosto comporta 61 parametri in più (35 in ingresso e 26 in uscita).

Supponendo di lavorare con 20 neuroni nascosti, si avranno 1220 pesi, che se dovessi rispettare la regola empirica comporterebbe un training set di 6100 elementi.

Al termine del progetto, il data set è composto da un totale di 824 caratteri.

Per ogni carattere sono stati scelti uno, due o tre varianti.

Nel caso di una sola variante, ho previsto un livello di libertà maggiore sull'inclinazione dei segmenti.

2.1 Lettera A

Con linee rette (12 esempi)



Con arco superiore (10 esempi)



Con tratti inferiori (8 esempi)



2.2 Lettera B

Archi uguali (13 esempi)



Arco superiore minore (10 esempi)



2.3 Lettera C

Lettera normale (21 esempi)



Lettera con tratto in basso a sinistra (9 esempi)



Gli esempi spaziano da figure con arco squadrato a totalmente circolare.

2.4 Lettera D

Normale (19 esempi)



Con tratti sul lato sinistro (13 esempi)



2.5 Lettera E

Tratto centrale uguale agli altri due (13 esempi)



Tratto centrale più corto (14 esempi)



2.6 Lettera F

Tratti orizzontali uguali (22 esempi)



Tratto orizzontale inferiore più corto (9 esempi)



2.7 Lettera G

Tratto centrale



Tratto interno, arco squadrato



Tratto interno, arco curvilineo



In totale ci sono 32 esempi.

2.8 Lettera H

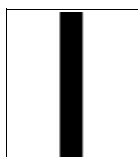
Come tipologia di H ho previsto solo quella standard, aggiungendo come grado di libertà la curvatura dei segmenti.

In totale ho inserito 34 esempi.

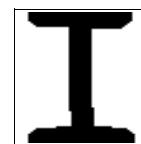


2.9 Lettera I

Solo segmento verticale (14 esempi)



Con tratti (12 esempi)



Sugli esempi di questa lettera non ho mai eseguito un controllo sui bordi bianchi laterali, perché altrimenti il risultato del ridimensionamento sarebbe stata una figura interamente di colore nero,

quindi ho preferito lasciarla così (ho eliminato solo le strisce bianche sopra e sotto).

2.10 Lettera J

Tradizionale (15 esempi)



Con tratto superiore (13 esempi)



Ho inserito negli esempi angolature molto diverse.

2.11 Lettera K

Ho inserito solo il tipo base di K, giocando sull'inclinazione dei segmenti.

In totale ho inserito 24 esempi.

2.12 Lettera L

Normale (21 esempi)



Con tratto (12 esempi)



2.13 Lettera M

Ho inserito solo il tipo base di M, giocando sull'inclinazione dei segmenti e sulla profondità della parte centrale. In totale ho inserito 31 esempi.



2.14 Lettera N

Ho inserito solo il tipo base di N, giocando sull'inclinazione dei segmenti.

In totale ho inserito 36 esempi.

2.15 Lettera O

O circolare



O intermedia



O tendente ad un rettangolo



In totale ho inserito 34 esempi.

2.16 Lettera P

Squadrata



Rotonda



In totale ci sono 34 esempi.

2.17 Lettera Q

Q più rotonda



Q più squadrata



In totale ho inserito 35 esempi.

2.18 Lettera R

Stacco del segmento obliquo dall'inizio



Stacco del segmento obliquo dal centro



In totale ho inserito 28 esempi.

2.19 Lettera S

Squadrata



Circolare



In totale ho inserito 40 esempi.

2.20 Lettere T

Senza tratti laterali



Con tratti laterali



In totale ho inserito 33 esempi.

2.21 Lettere U

Base circolare

Base squadrata



Nel data set ci sono 35 esempi.

2.22 Lettera V

Angolo stretto



Angolo intermedio



Angolo largo



Ho inserito 40 esempi in tutto.

2.23 Lettera W

Tratto centrale alto



Tratto centrale basso



Ci sono 28 esempi nel data set.

2.24 Lettera X

Ho inserito solo il tipo base di X, giocando sull'inclinazione dei segmenti.
Esistono 31 esempi.

2.25 Lettere Y

Ho inserito 30 esempi solo del tipo base di Y, giocando sull'inclinazione dei segmenti.

2.26 Lettera Z

Senza tratti



Con tratti



Nel data set ci sono 36 esempi.

Dopo la realizzazione del data set ho eseguito sull'insieme di immagini una funzione di analisi per eliminare eventuali righe e colonne completamente bianche ai bordi della figura.

Successivamente sono passato alla ricerca e rimozione di eventuali esempi, di una stessa lettera, con codifica identica.

La loro presenza non provoca danno, ma evita di avere una rappresentanza più vasta.

3. Addestramento della rete neurale backpropagation

Questa fase ha impiegato la maggior parte del tempo.

Non sapendo come scegliere la funzione di addestramento e di performance da usare sono andato avanti per tentativi.

Per quanto riguarda i neuroni, intuitivamente ho pensato ad un minimo di 15 e ad un massimo di 30.

I primi tentativi di addestramento (anche su versioni parziali del data set) hanno mostrato un buon comportamento di *trainlm/mse*, *trainlm/sse*, *traingda/mse* e *traingdx/mse*.

Anche se *trainlm* offre risultati soddisfacenti, ha come svantaggio un tempo di elaborazione abbastanza lungo, anche se converge dopo poche epoche.

Questo spiega perché *trainlm* è adatta per reti con al più qualche centinaio di parametri e non è consigliata per problemi di riconoscimento di pattern.

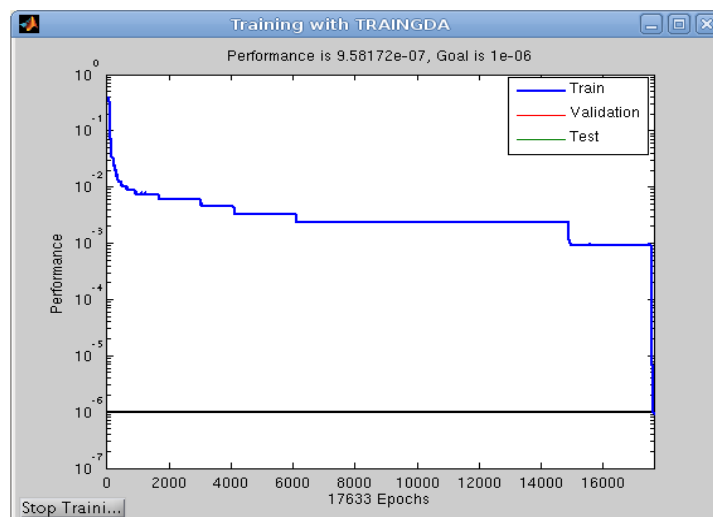
Per prova ho tentato di realizzare una funzione di performance (*./errorFun.m*) che, dato in ingresso la matrice degli errori ritorna l'errore relativo (vedere Appendice C per una sua descrizione). Essa permette di avere risultati di poco inferiori a *mse* per questo specifico problema.

Le combinazioni *traingda* e *traingdx* con *msereg* hanno dato risultati non convincenti.

Una bozza dei dati della prima fase di test è presentata nell'Appendice A.

In base al tempo richiesto e ai risultati ottenuti scelgo di utilizzare *traingda/mse*.

Dalle prove ho osservato che è quella che mi permette di superare più facilmente lo scoglio dei minimi locali.

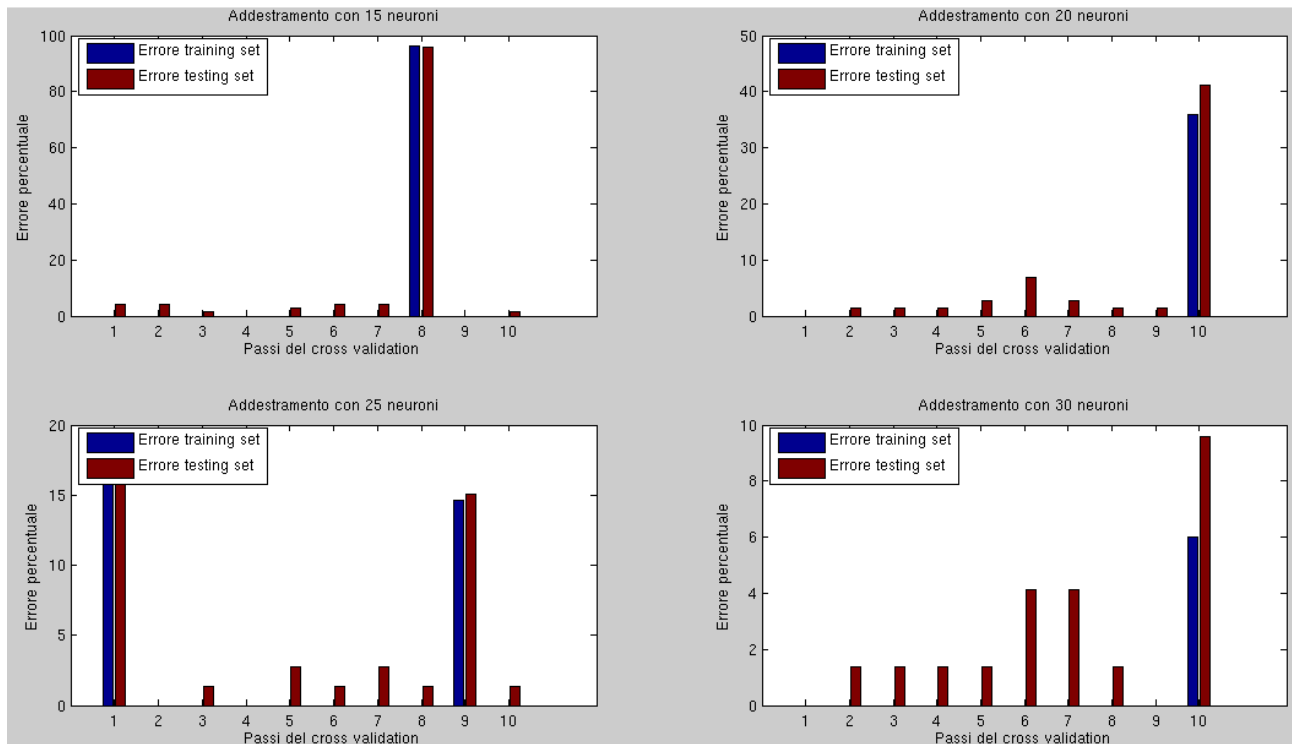


Il test finale che ho eseguito si basa su quattro sezioni di addestramento, con stesso data set, utilizzando cross validation con 10 folder.

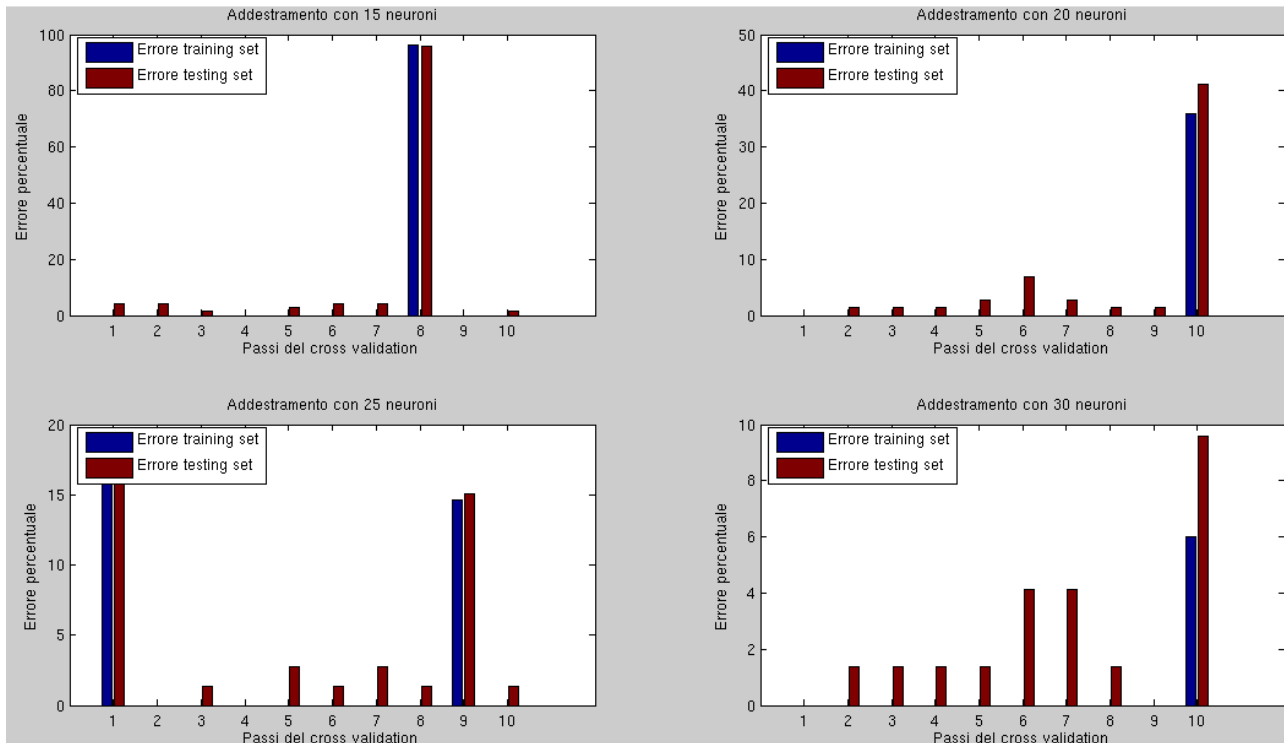
L'addestramento è stato studiato anche al variare del numero di neuroni nello strato nascosto (15, 20, 25 e 30).

Di seguito vengono riportati i grafici dell'errore di ogni sezione e successivamente l'errore medio.

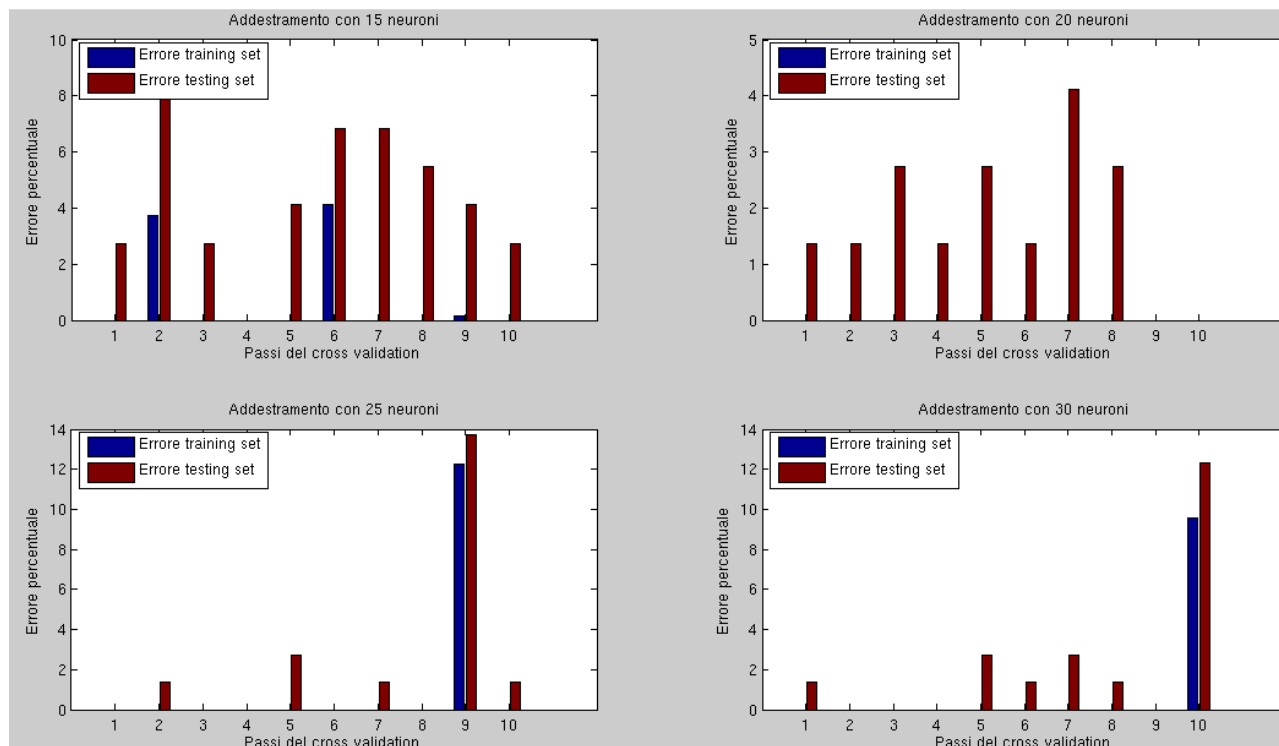
Primo addestramento



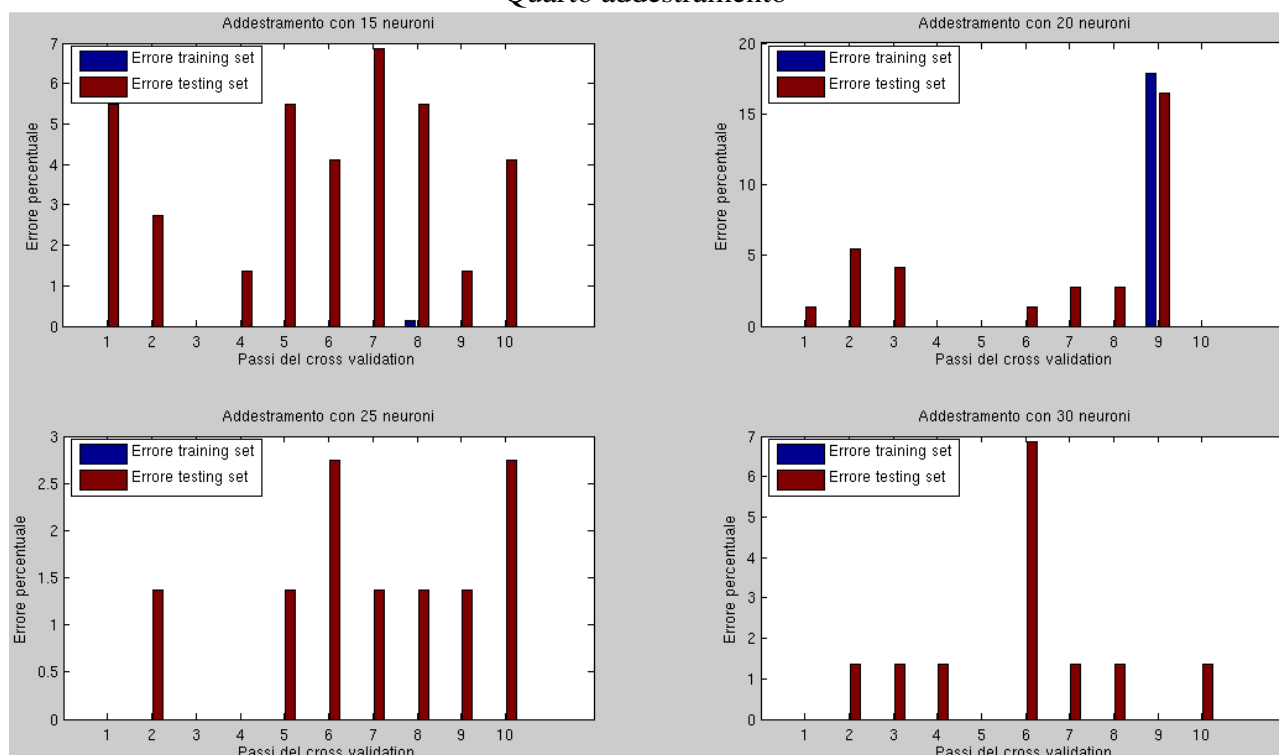
Secondo addestramento



Terzo addestramento

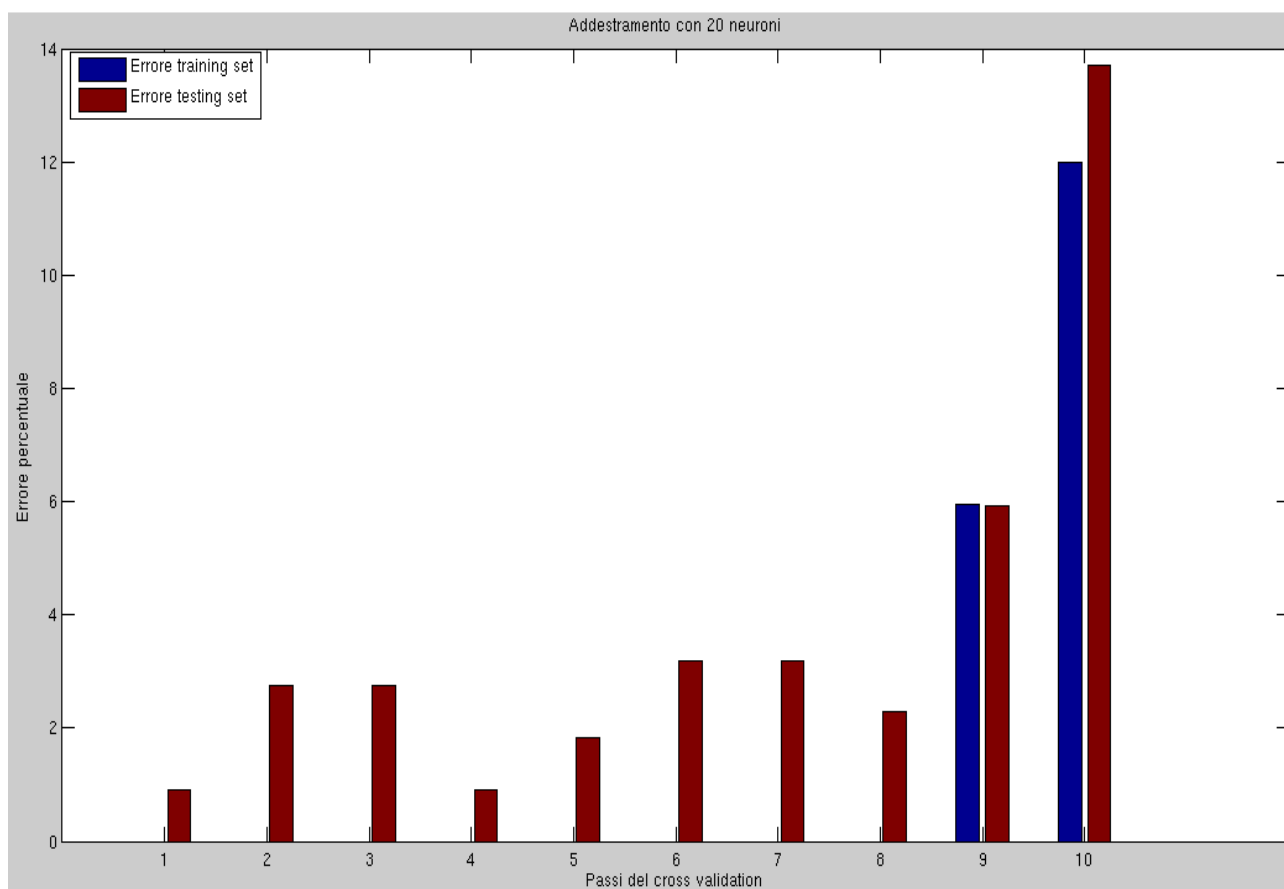
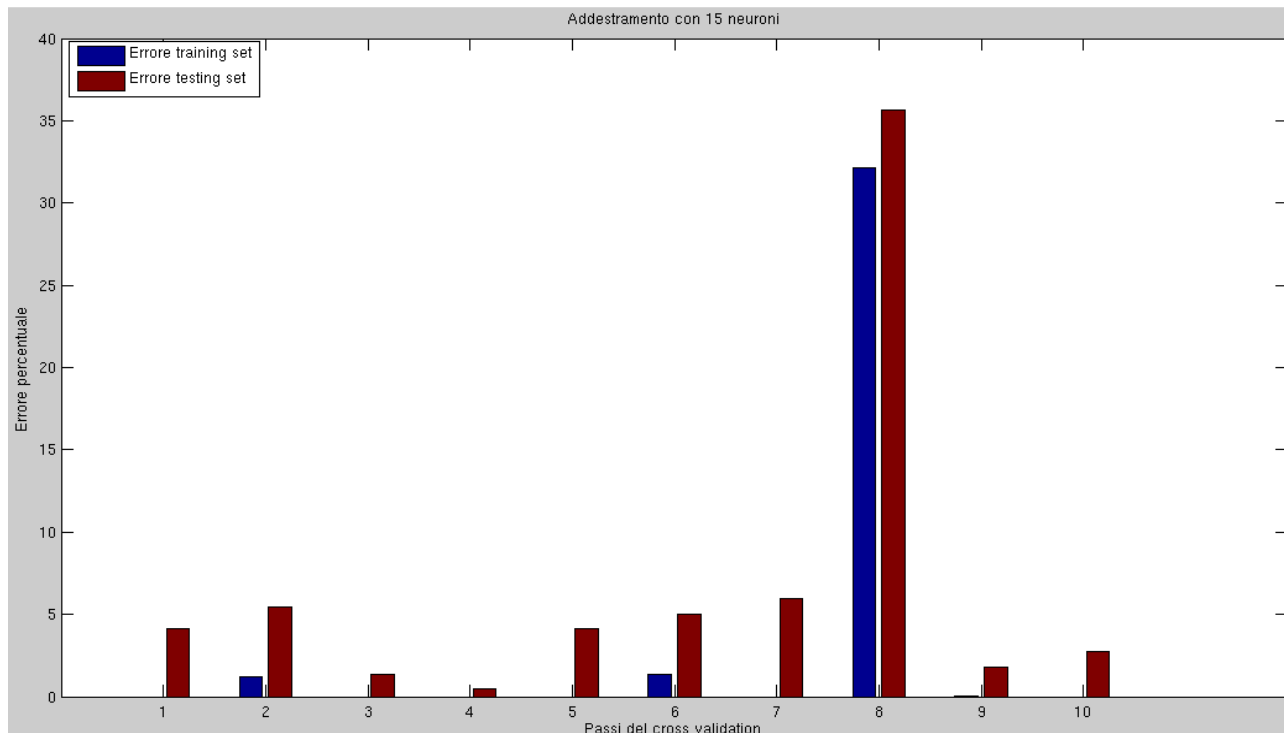


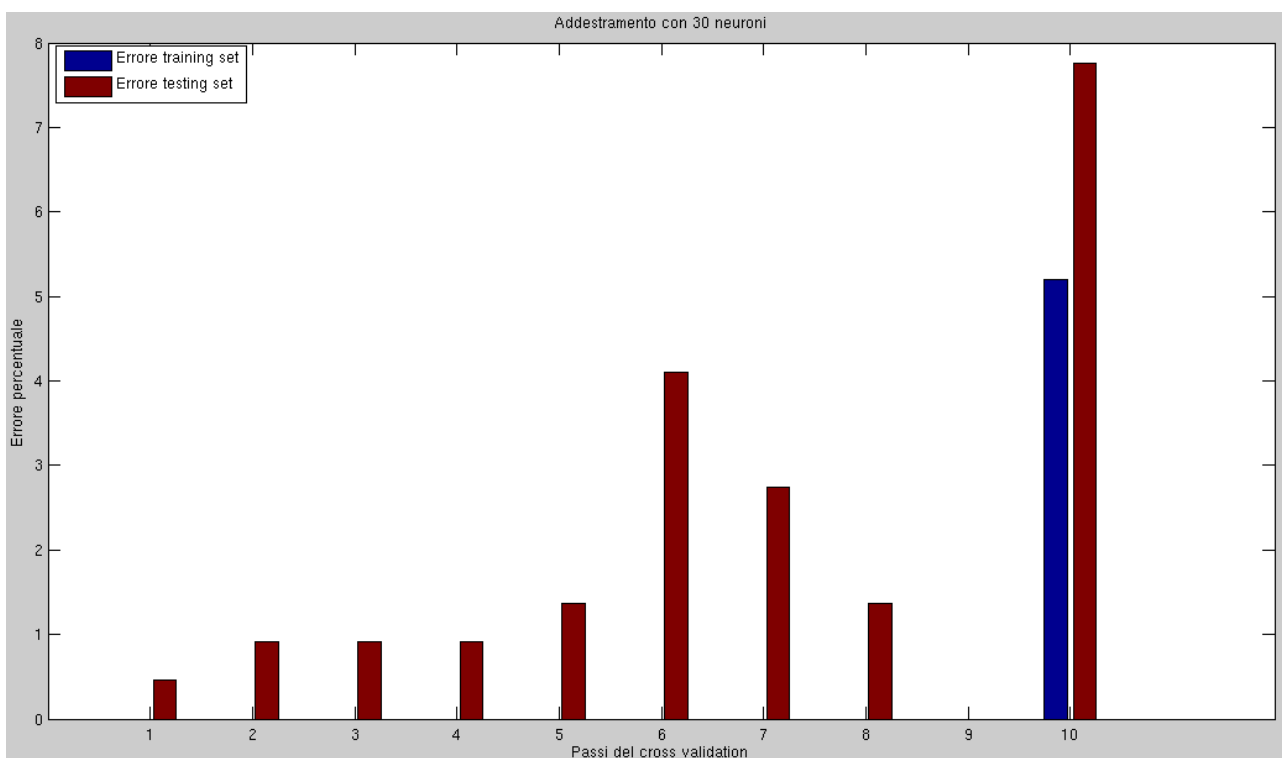
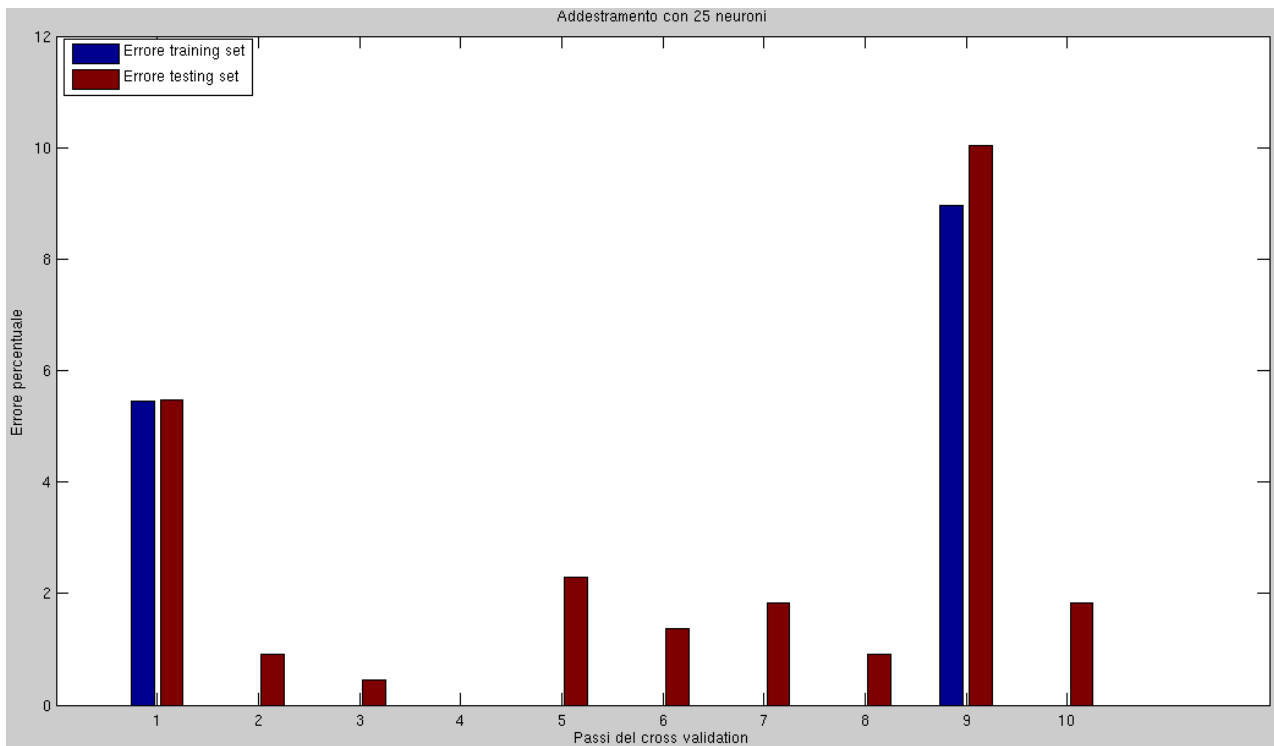
Quarto addestramento



Dove è presente errore sul training set, significa che l'addestramento si è bloccato in minimi locali.

Errori medi ottenuti:





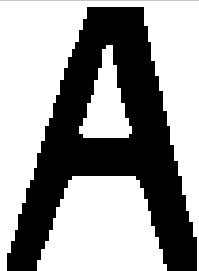
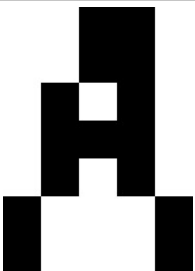
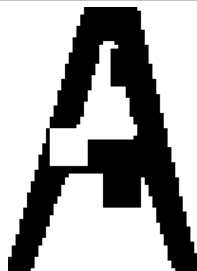
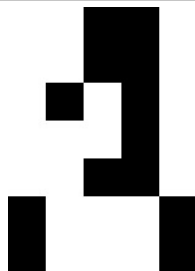
Come si può vedere esistono simulazioni con errore medio nullo, o inferiore al 3% (che su 73 elementi di test corrispondono a due errori).

A questo punto mi basta salvare su file le reti con risultato migliore, in modo da renderle disponibili all'interfaccia grafica.

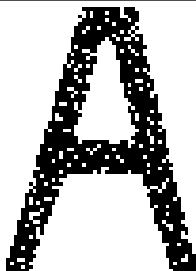
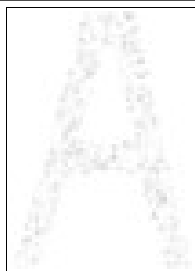
4. Addestramento con disturbo

La generazione del disturbo avviene mediante una versione adattata dell'algoritmo *'salt & pepper'*. Il disturbo viene applicato all'immagine originale, cioè quella ancora non elaborata (ridimensionata e convertita in bianco e nero).

Le variazioni all'algoritmo consistono nell'inserire solo disturbi con colore bianco o nero (monocromatici) e di dimensioni tali da corrispondere ad un pixel nella codifica 7x5 (quella data in ingresso alla rete).

A senza errore	A elaborata	A con disturbo di 0.1	A elaborata con disturbo
			

L'effetto di una variazione di un singolo pixel, o di un numero ridotto, nell'immagine originale risulta ininfluente quando si va a ridimensionare, mentre, un disturbo di dimensioni appropriate (proporzionale ad un pixel in un'immagine 7x5) ha effetti.

Disturbo Gaussiano con media 0.3	Disturbo Gaussiano con media 1
	
La codifica è identica a quella ottenuta dall'immagine senza disturbo	La codifica è interamente bianca (il bordo è stato evidenziato per mettere meglio a fuoco la figura)

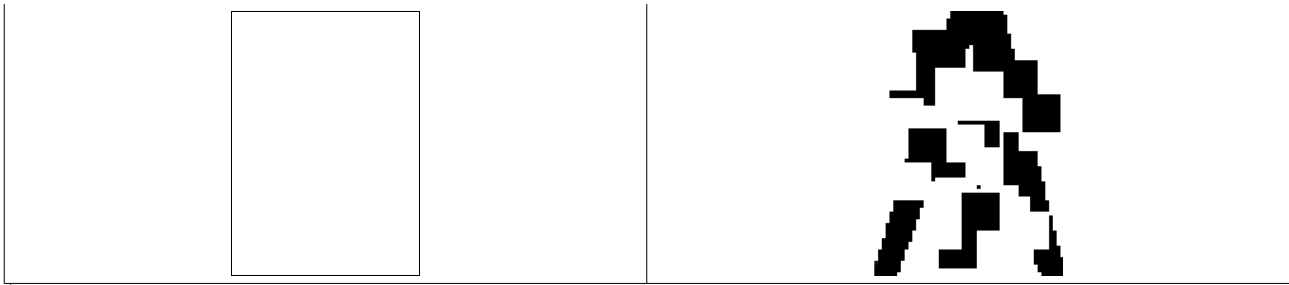
Il nuovo colore viene scelto casualmente e non è relazionato al colore precedente dell'area.

L'unico parametro manipolabile è l'intensità del disturbo.

Tale valore indica quanti disturbi inserire, per esempio, un valore di 0.05 indica di aggiungere un numero di disturbi pari al 5% di 35, che arrotondato all'intero inferiore, fa 1.

Un effetto positivo di questo algoritmo è che, a differenza di altri, con un fattore di disturbo molto elevato e applicandolo più volte, non si arriva mai ad avere un'immagine completamente bianca (più correttamente, la probabilità è estremamente bassa).

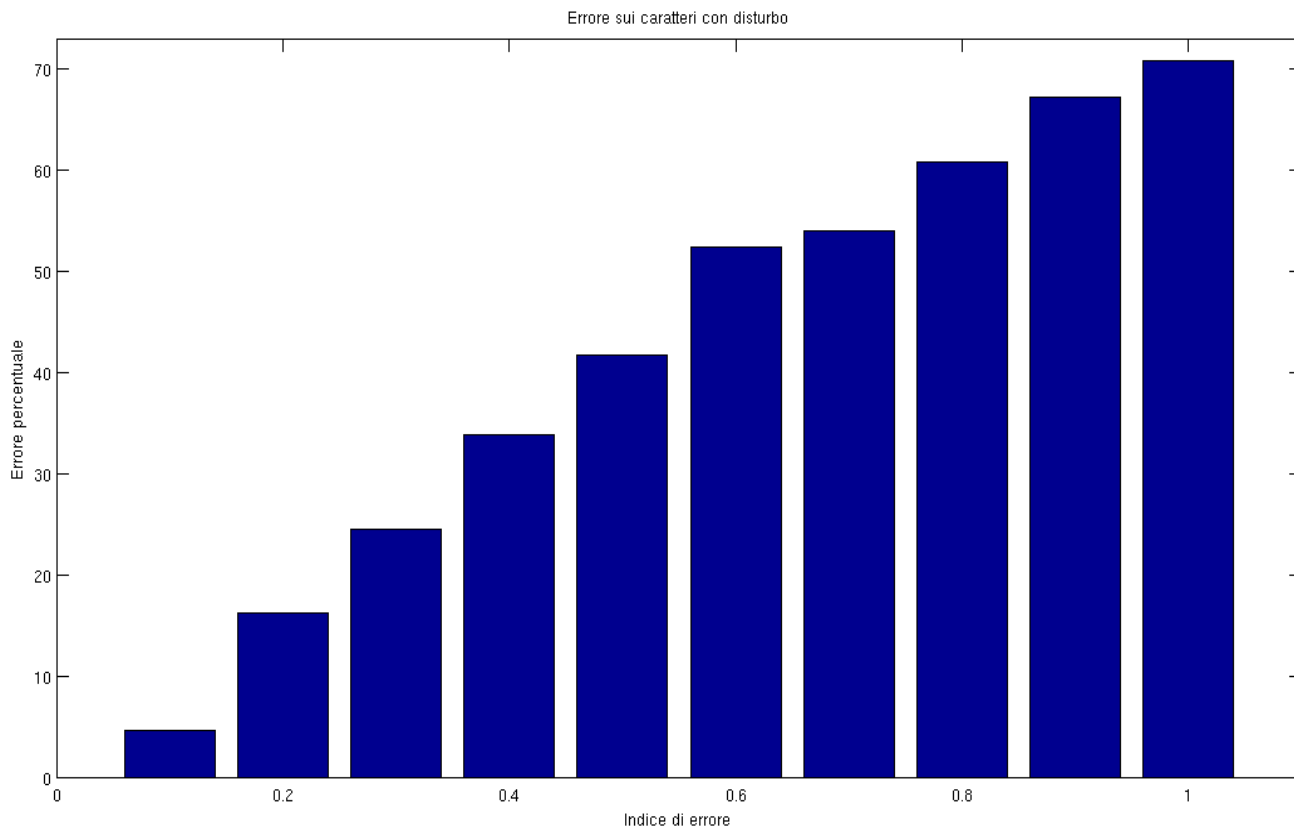
Applicando due volte un disturbo Gaussiano con media 1 (il bordo è stato evidenziato per mettere meglio a fuoco la figura)	Applicando una volta un disturbo unitario con l'algoritmo da me utilizzato
--	--



È intuibile che nel secondo caso un tentativo di addestramento è molto arduo dato che associa valori a simboli che non hanno più una geometria.

Prima di procedere con l'addestramento con disturbo, ho esaminato la risposta, relativa ad ingressi con rumore, di una rete allenata con training set non corrotto.

La rete possiede 25 neuroni nello strato nascosto e durante la fase di addestramento ha ottenuto errore nullo con *traingda* e *mse*.



Per un indice di rumore pari a 0.1, l'errore rimane sotto il 10%, ma poi la crescita è lineare e consistente.

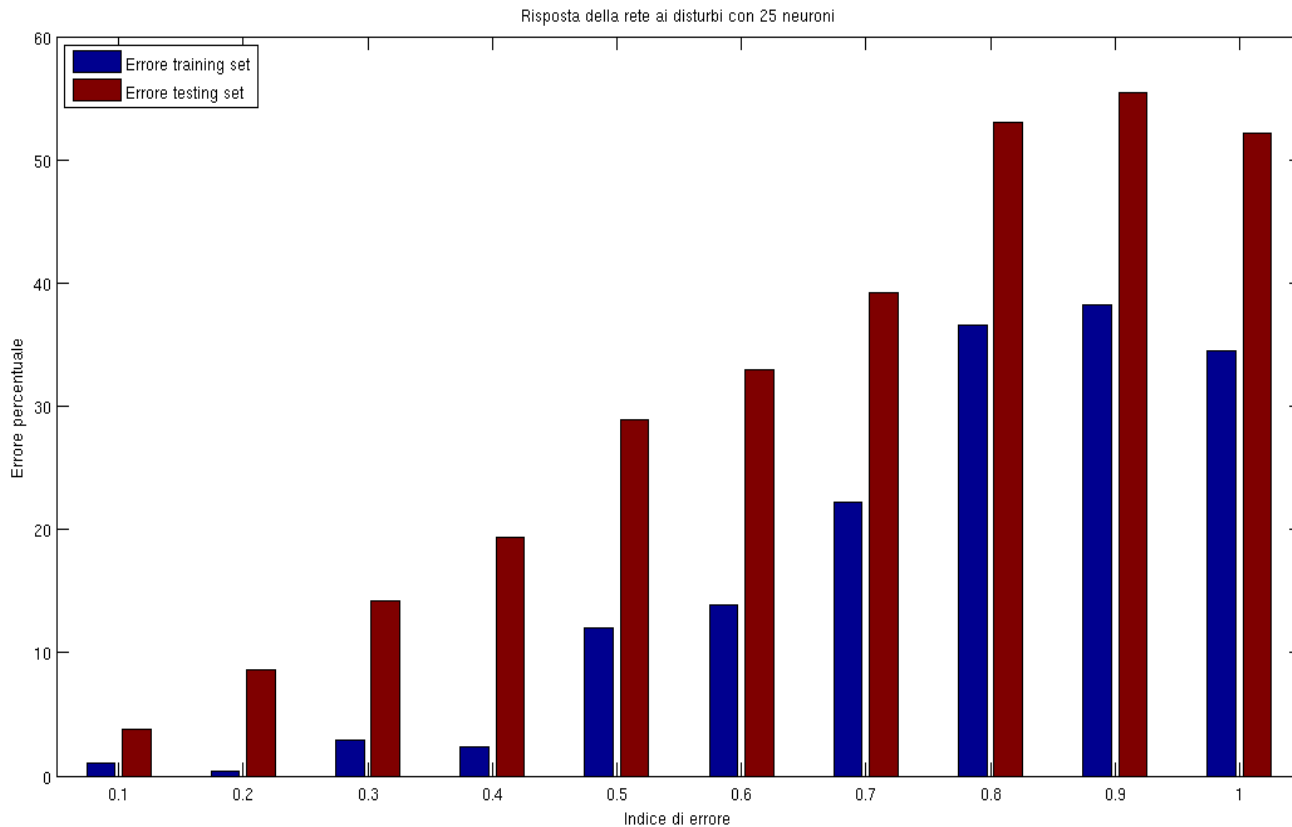
Di seguito viene riportato il grafico con le medie degli errori compiuti, al variare dell'indice di disturbo da 0.1 a 1, con passo 0.1.

La funzione di addestramento è *traingda* e quella di performance è *mse*.

La rete è stata addestrata con training set normale e disturbato contemporaneamente, ed è stata testata sul testing set normale e disturbato contemporaneamente.

Per ogni indice di disturbo è stata calcolata una sezione di addestramento (intero cross validation) e

nel grafico è riportato il valore medio dell'errore.
Nell'Appendice E sono riportati i singoli grafici di errore.



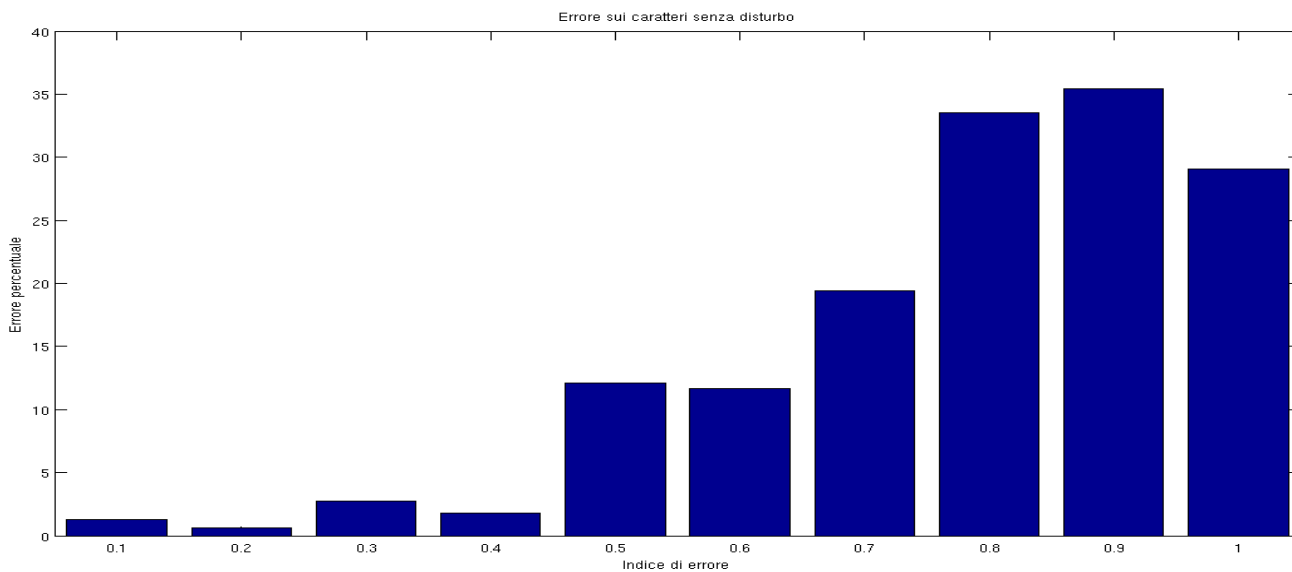
Per prima cosa devo segnalare che nel cross validation, alcuni folder presentano un errore notevolmente superiore agli altri, e ciò comporta un innalzamento della media non trascurabile, per rimediare a questo problema avrei dovuto ripetere molte più volte lo stesso test.

Come si può notare, fino ad un coefficiente inferiore a 0.5, l'errore sul training set è molto basso, se non quasi nullo.

Per valori superiori a 0.4, l'errore commesso cresce rapidamente, fino a far superare il 50% al testing set e 40% al training set.

Il fatto di aver avuto errori minori per indice uguale a 1 rispetto a 0.9, è puramente casuale.

Ulteriore osservazione, il grafico seguente riporta l'errore medio compiuto su tutte le lettere senza rumore (intero data set) dalle reti addestrate, con differente indice di disturbo.

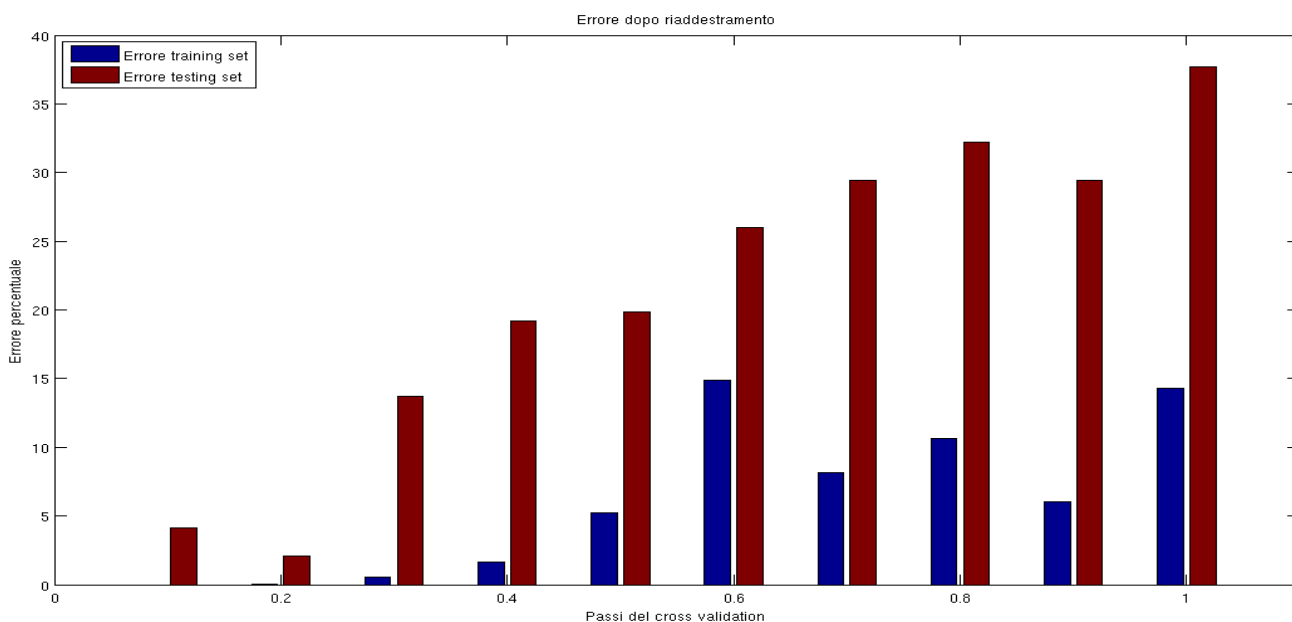


Siccome questo errore non è trascurabile, si può concludere che le reti smettono anche di imparare bene dopo una certa soglia di rumore.

Abbiamo dimostrato che addestrare le reti con del disturbo serve, infatti l'errore è notevolmente diminuito rispetto al primo grafico del paragrafo.

Da notare che il primo grafico si riferisce solo all'errore su dati disturbati, mentre gli altri all'errore su dati normali e disturbati.

Riaddestrando la rete con dati non corrotti da disturbo ho ottenuto i seguenti risultati.



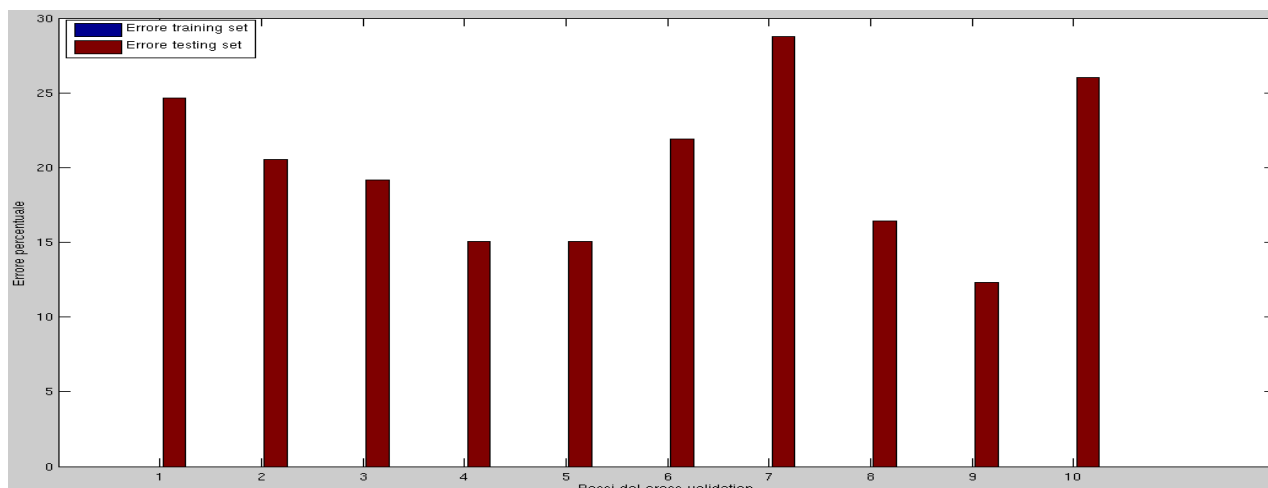
Si può vedere che l'errore sul training set è notevolmente ridotto e anche sul testing set c'è stato un ridimensionamento.

Quindi, riaddestrando la rete con gli esempi non disturbati, ha ridotto l'errore, migliorando l'apprendimento.

5. Addestramento della rete neurale RBF

Per prima cosa ho provato l'addestramento con newrbe che crea un neurone nello strato nascosto per ogni esempio, quindi ritorna una matrice che sul training set ha errore nullo.

Di seguito riporto i dati relativi all'addestramento, utilizzando cross validation e spread uguale al valore di default, cioè 1.



Come previsto l'errore sul training set è nullo, ma quello sul testing set non lo è, quindi ho calcolato un valore di spread appropriato.

Ho individuato lo spread massimo, come la massima distanza tra tutte le possibili coppie di esempi, ed è 5.5678.

Avendo utilizzato la funzione *dist* (distanza euclidea), significa che si ha una variazione massima di 31 bit, che su 35 è l'88%, quindi è un valore molto alto.

Questo risultato è stato ottenuto con le seguenti codifiche (in rosso gli unici pixel in comune):

Codifica di una X

0	1	1	1	0
1	0	1	0	1
1	0	0	0	1
1	1	0	1	1
1	0	0	0	1
1	0	1	0	1
0	1	1	1	0

Codifica di una O

1	0	0	0	1
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
1	0	0	0	1

Fatto ciò sono passato ad esaminare il valore minimo dello spread.

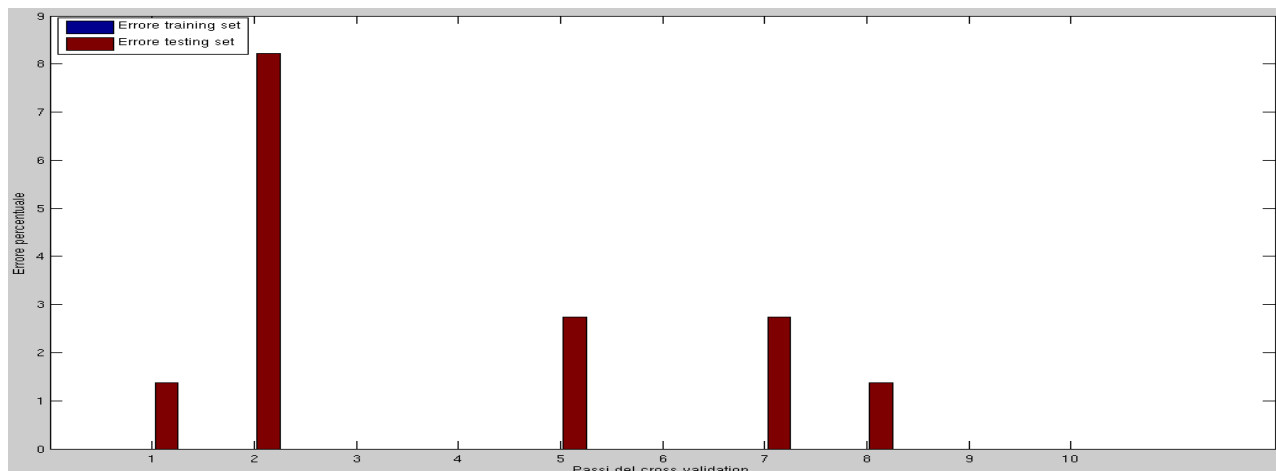
Tale valore non si può basare sull'intero insieme degli esempi, ma deve essere calcolato sui vari gruppi di lettere simili.

Per prima cosa ho calcolato la distanza massima e media tra tutti gli esempi di una stessa categoria del data set (vedere l'Appendice D per i risultati dettagliati).

Lo spread medio, tra i singoli gruppi, è 1.777 (3.25 bit).

Ho scelto un valore di spread pari a 2.3 che risulta maggiore delle medie delle distanze dei singoli gruppi e minore della distanza massima.

Sempre usando newrbe, ho ottenuto i seguenti errori.



L'errore sul testing set è significativo solo in un caso.

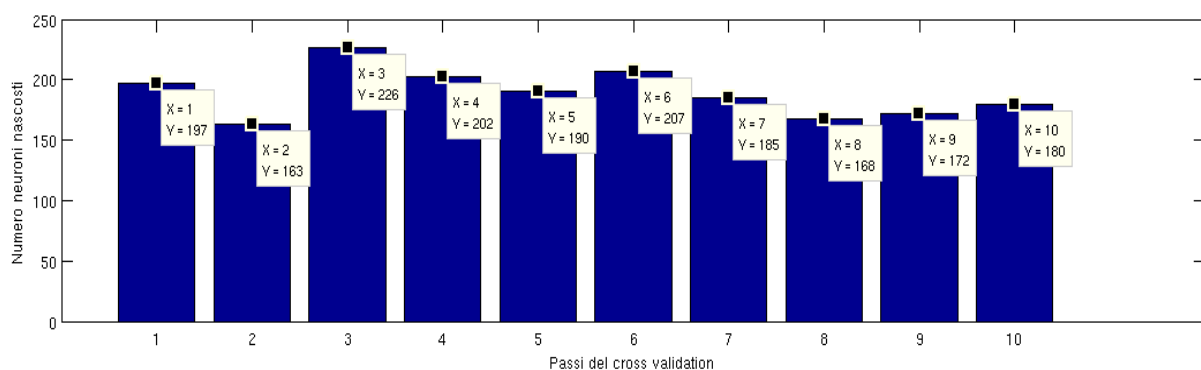
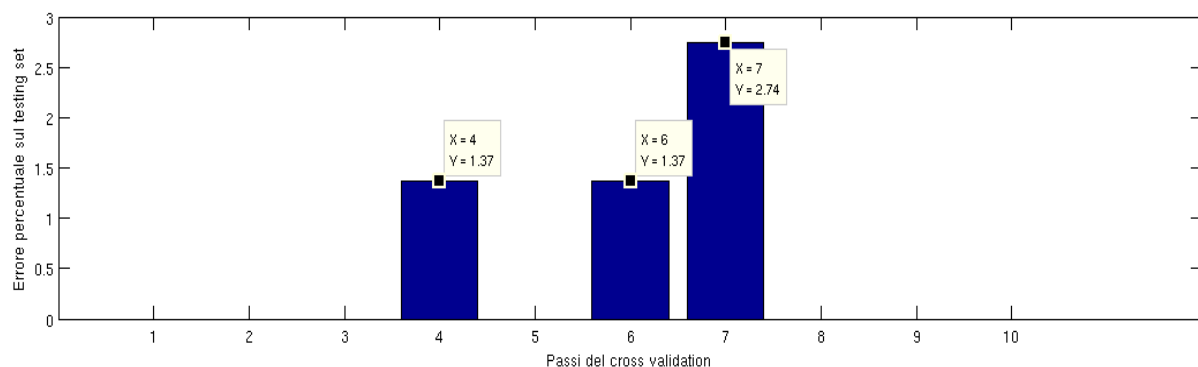
L'inconveniente di questa tecnica è che, dati 751 esempi come addestramento, crea 751 neuroni.

Per ridurre il numero di neuroni ho usato la variante della rete precedente, *newrb*.

Dopo varie simulazioni ho ottenuto dei buoni risultati con goal uguale a 30 (utilizza *sse* come funzione di performance).

Usare un goal uguale a 0 non avrebbe avuto senso, perché si sarebbe raggiunto con 751 neuroni.

Di seguito sono riportati i risultati ottenuti.



Non ho riportato sui grafici l'errore sul training set perché è sempre nullo.

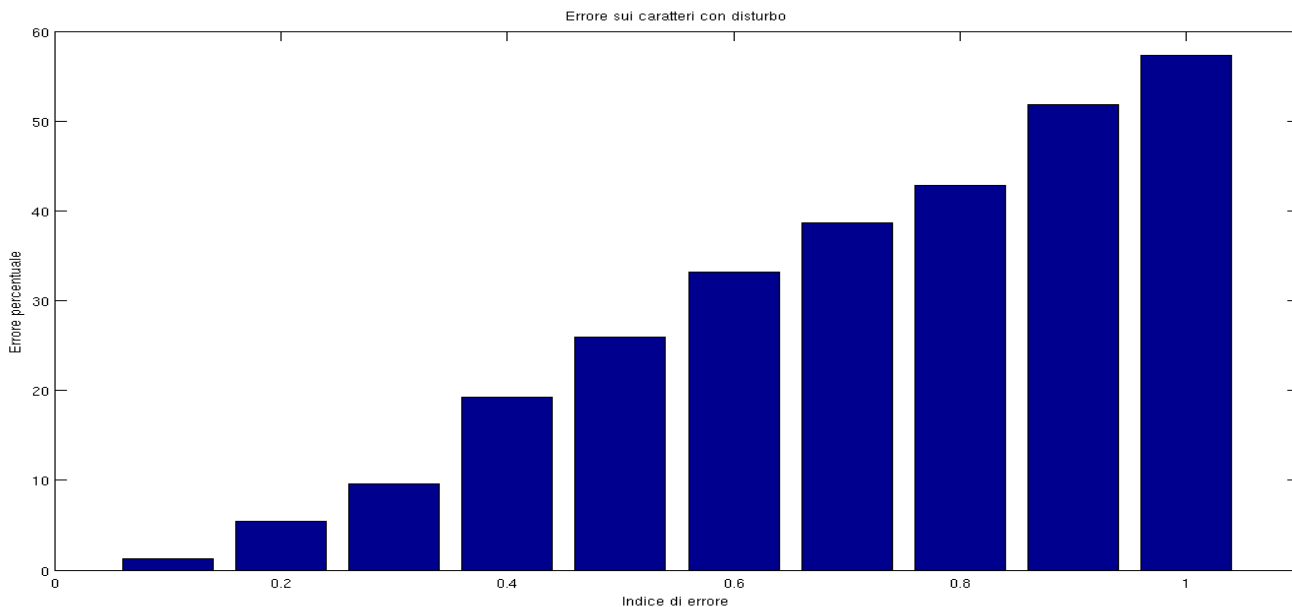
Il numero medio di neuroni nello strato nascosto è 189.

Ho riportato una sola simulazione perché tutte le esecuzioni, sullo stesso training set, ritornano lo stesso risultato.

6. Addestramento con disturbo

Come per il caso precedente, eseguo un test per verificare il comportamento di una rete, addestrata con dati corretti, con input corrotto da disturbo.

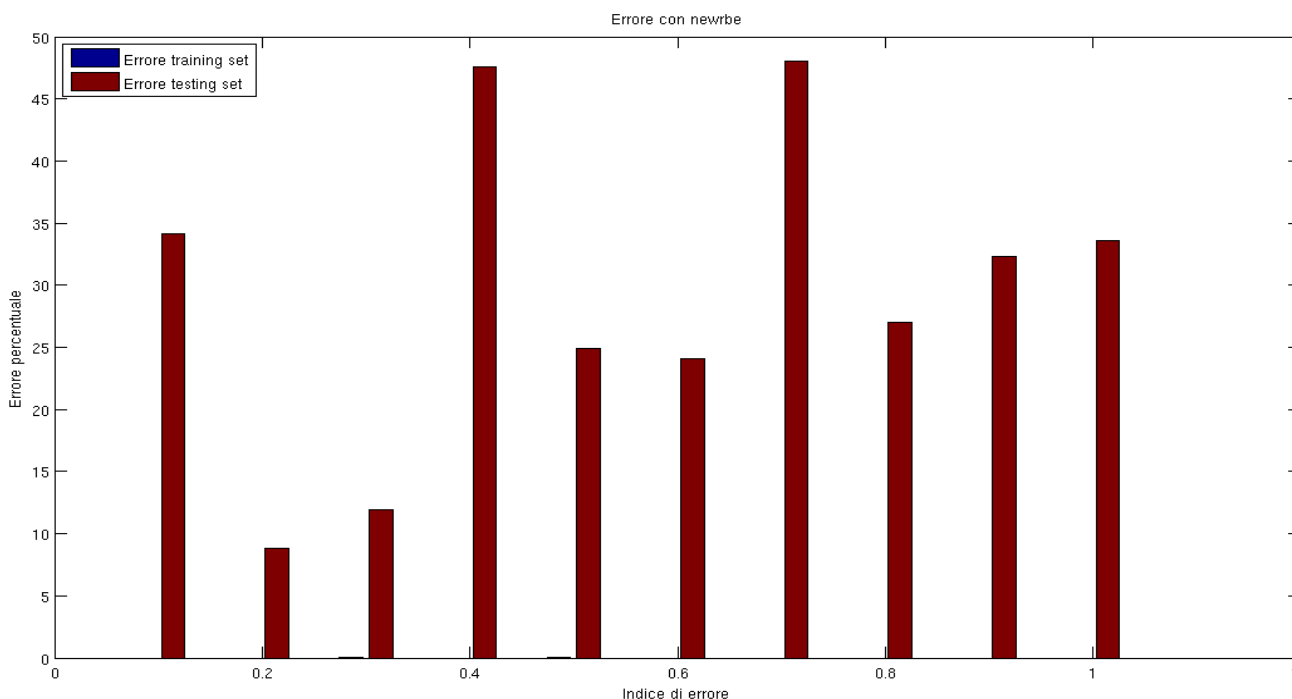
La rete ha 161 neuroni nello stato nascosto e durante l'allenamento ha riportato errore nullo.



L'errore rimane sotto il 10% fino a un disturbo pari a 0.3.

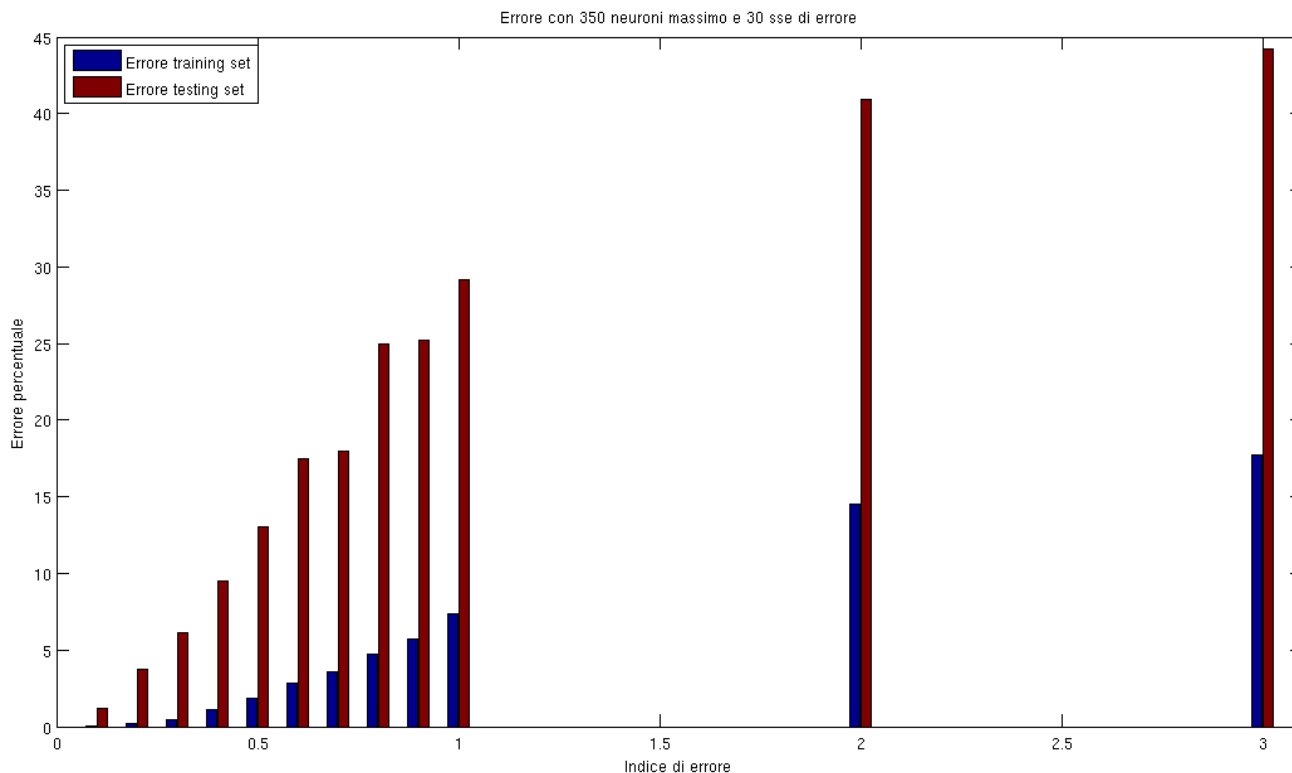
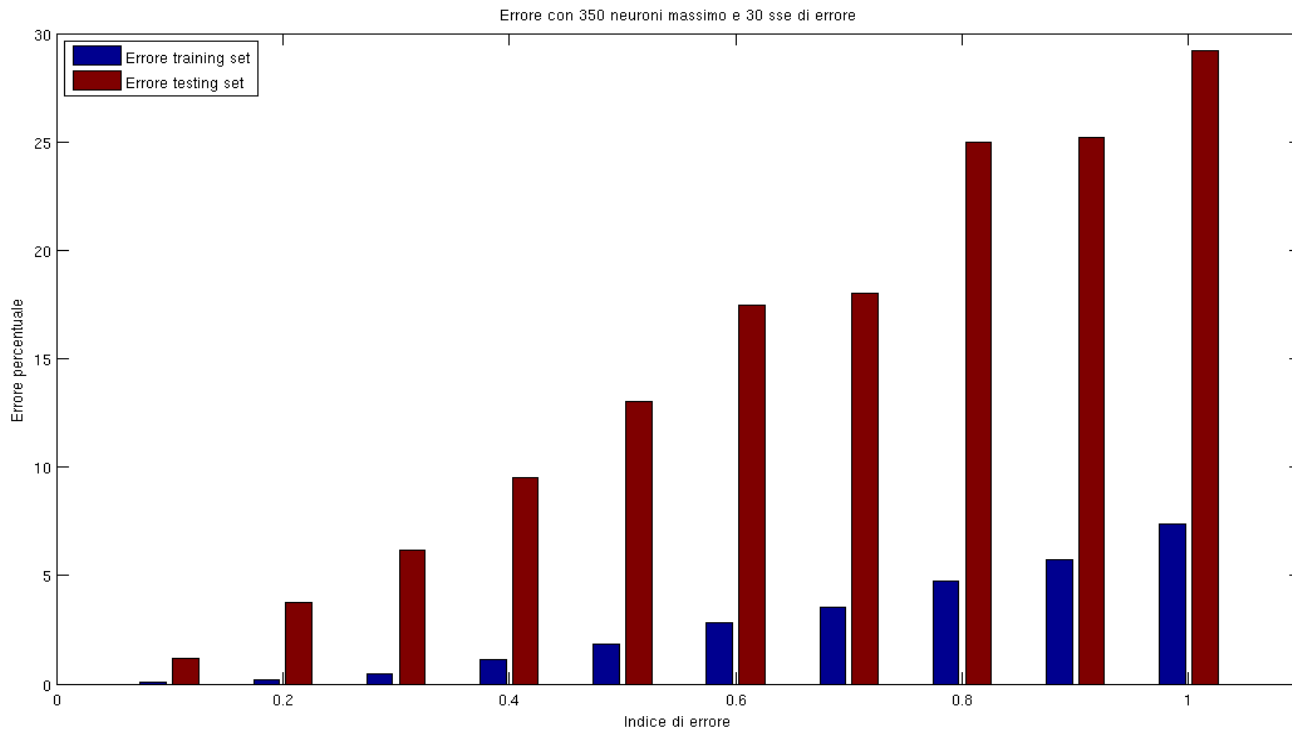
Continuo ad usare uno spread uguale a 2.3.

Come primo esempio ho provato ad addestrare una rete mediante *newrbe*, con errore variabile (da 0.1 a 1) e fornendo come esempi gli insiemi con e senza disturbo.



Noto che l'errore sul training set è nullo come previsto, mentre l'errore sul testing set è quasi sempre significativo.

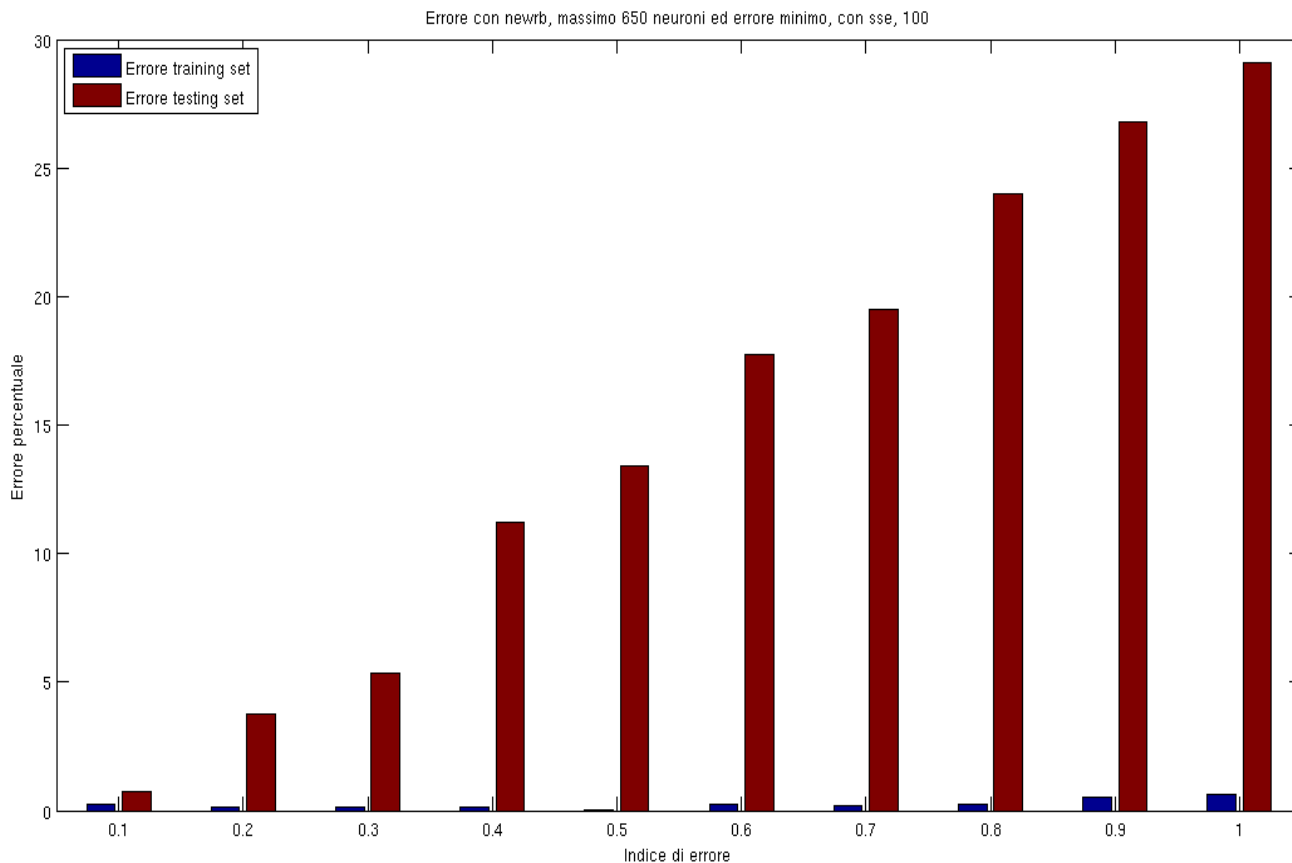
Come passo successivo eseguo un test con *newrb*, imponendo un numero massimo di 350 neuroni e un goal di 30 (con sse).



Nel secondo grafico ho mostrato i risultati del precedente, in relazione ad addestramenti con indice di disturbo 2 e 3.

Si nota subito che, rispetto alle reti backpropagation, l'errore presenta una crescita più lineare.

Le sezioni di addestramento sono quasi tutte terminate a causa del raggiungimento del numero massimo di neuroni, quindi ho rieseguito tutto il test impostando 650 come nuovo limite e 100 come minimo errore.



Avendo permesso di usare più neuroni, l'errore sul training set si è notevolmente ridotto.

Ciò vuol dire che almeno il training lo ha imparato.

Per quanto riguarda il testing set, l'errore è stato solo un poco limato, quindi si continua ad avere un errore sotto il 15% per indici di errore inferiori a 0.6.

7. Confronto dei risultati

Per quanto riguarda l'addestramento senza disturbo, le reti backpropagation permettono di avere un numero di neuroni molto inferiore nello strato nascosto, ne ho usati 25, rispetto alle reti RBF che ne hanno richiesti circa 190.

Come aspetto negativo le reti backpropagation necessitano di un addestramento più lungo.

Entrambe le reti permettono di avere errori nulli sul training set e testing set.

L'addestramento con rumore ha mostrato un minor errore con le reti RBF.

Il problema delle reti backpropagation è che, dopo un certo indice di errore, smettono di imparare bene, mentre le reti RBF continuano a conservare correttamente ciò che apprendono.

Questo è spiegabile dalla struttura stessa delle reti, le prima hanno neuroni con una rappresentatività maggiore e quindi è naturale che vengano influenzati maggiormente da ingressi distorti.

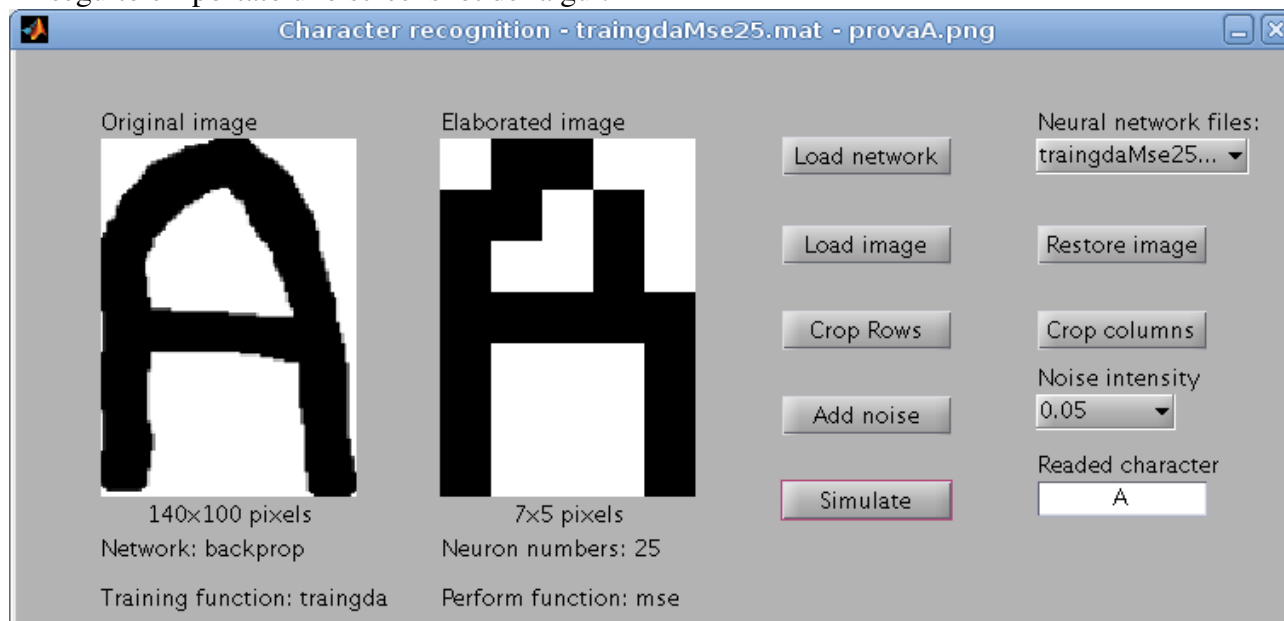
Per rendere competitive le reti backpropagation è necessario un nuovo addestramento con gli esempi senza disturbo, per rafforzare ciò che avevano appreso, richiedendo ulteriore tempo (molto di più delle RBFN).

Per quanto riguarda il numero di neuroni, le reti backpropagation continuano ad usarne 25, mentre le reti RBF ne richiedono più di 350 per un buon risultato.

8. Realizzazione della gui

L'obiettivo è stato quello di realizzare una interfaccia il più intuitiva possibile e che faccia notare quale codifica vada effettivamente in ingresso alla rete.

Di seguito è riportato uno screenshot della gui.



Nella barra del titolo viene inserito il nome dell'interfaccia '*Character recognition*', seguito dal nome del file della rete neurale e dell'immagine caricata.

La prima figura rappresenta l'immagine originale caricata, con sotto indicata la dimensione in pixel. La seconda invece rappresenta ciò che andrà in ingresso alla rete neurale e altro non è che la prima immagine ridimensionata e convertita in bianco e nero.

Il menù popup '*Neural network files*' elenca tutte le reti, già addestrate, disponibili e con il pulsante '*Load network*' è possibile caricarle.

Sotto le figure vengono riportati alcuni dati sulla rete neurale caricata, e sono:

- *Network*: tipo di rete, *backprop* o *RBF*;
- *Neuron number*: numero di neuroni dello strato nascosto;
- *Training function*: funzione usata per l'addestramento (non ha valore per RBFN);
- *Perform function*: funzione usata per la valutazione delle performance.

Il pulsante '*Load image*' permette di carica un'immagine.

I pulsante '*Crop rows*' e '*Crop columns*' elimina eventuali righe e colonne bianche ai bordi.

'*Add noise*' permette di aggiungere del rumore all'immagine originale.

Il livello di intensità del disturbo è selezionabile dall'apposito popup menù.

Infine, il pulsante '*Restore image*' annulla tutte le modifiche di cropping e rumore effettuate sull'immagine, ripristinando la versione caricata da file.

'*Simulate*' effettua una simulazione, ritornando il carattere riconosciuto nel campo di testo adiacente.

Per rendere utilizzabile dalla gui una rete neurale addestrata, basta digitare:

```
>> net = <rete>; save <path>/gui/nns/<nome>.mat net
```

APPENDICE

Appendice A: Risultati della prima fase di test

In questa fase utilizzo un data set ancora in fase di analisi, che contiene vari dopplioni di lettere, quindi sono dati abbastanza indicativi.

Numero di elementi del training set: 778

Numero di elementi del test set: 46

Algoritmo di addestramento: *trainlm*

neuroni	terminazione	funzione	errore	errore_training	errore_test
10	Max mu	ERRORFUN	0.00128535	0.12853%	6.5217%
12	Goal	ERRORFUN	0	0%	6.5217%
15	Max mu	ERRORFUN	0.0437018	4.3702%	10.8696%
20	Goal	ERRORFUN	0	0%	4.3478%
25	Goal	ERRORFUN	0	0%	6.5217%
10	Goal	MSE	2.36003e-07	0%	2.1739%
12	Max mu	MSE	4.94364e-05	0.12853%	4.3478%
15	Goal	MSE	1.70568e-07	0%	2.1739%
20	Goal	MSE	4.20505e-07	0%	2.1739%
25	Goal	MSE	4.27461e-07	0%	6.5217%
10	Goal	SSE	7.27195e-07	0%	4.3478%
12	Goal	SSE	4.10644e-07	0%	6.5217%
15	Max mu	SSE	32	4.1131%	6.5217%
20	Goal	SSE	2.28713e-07	0%	4.3478%

Numero di elementi del training set: 457

Numero di elementi del test set: 42

Algoritmo di addestramento: *traingda*

neuroni	epoche	funzione	errore	errore_training	errore_test
12	1441	MSE	0.00446663	11.5974%	11.9048%
12	50000	MSEREG	0.0381107	95.186%	95.2381%
12	15403	sse	2.00001	0.43764%	0 %
13	2104	MSE	0.00211098	5.4705%	7.1429%
13	50000	MSEREG	0.0368169	95.186%	95.2381%
13	12411	sse	2.00001	0.43764%	0 %
14	1291	MSE	0.010357	26.9147%	28.5714%
14	50000	MSEREG	0.036954	94.5295%	95.2381%
14	6549	SSE	9.99905e-05	0%	0 %
15	1389	MSE	0.00497117	12.9103%	11.9048%
15	50000	MSEREG	0.0372408	94.5295%	95.2381%
15	9355	SSE	19	4.1575%	2.381%
16	1063	MSE	0.00497051	12.9103%	16.6667%
16	50000	MSEREG	0.0368068	94.7484%	95.2381%
16	5153	SSE	9.67501e-05	0	0
17	1554	MSE	0.00328669	8.5339%	9.5238%
17	50000	MSEREG	0.0361251	95.186%	97.619%
17	7180	SSE	183	40.0438%	42.8571%
18	1577	MSE	0.00488481	12.6915%	14.2857%
18	50000	MSEREG	0.0370677	95.4048%	95.2381%

18	3456	SSE	9.91492e-05	0%	0 %
19	1427	MSE	0.0030346	7.8775%	7.1429%
19	15001	MSEREG	0.0368848	94.7484%	95.2381%
19	6421	SSE	95	20.7877%	19.0476%
20	1683	MSE	0.00202429	5.2516%	7.1429%
20	50000	MSEREG	0.0368444	94.5295%	95.2381%
20	802	SSE	457	95.186%	95.2381%
21	1561	MSE	0.00261269	6.7834%	9.5238%
21	50000	MSEREG	0.0365833	95.6236%	95.2381%
21	2757	SSE	9.97075e-05	0%	0 %
22	1104	MSE	0.00572697	14.8796%	14.2857%
22	50000	MSEREG	0.0335079	15.7549%	14.2857%
22	3599	SSE	9.96207e-05	0%	0 %

Algoritmo di addestramento: *traingdx*

neuroni	epoch	funzione	errore	errore_training	errore_test
12	4866	MSE	0.00177532	4.5952%	2.381%
12	4796	MSEREG	0.0367098	94.5295%	95.2381%
12	1683	SSE	457	94.9672%	95.2381%
13	687	MSE	0.00152223	3.9387%	2.381%
13	50000	MSEREG	0.0363406	94.7484%	95.2381%
13	393	SSE	457	95.6236%	95.2381%
14	1126	MSE	0.00513837	13.3479%	14.2857%
14	50000	MSEREG	0.0360952	94.5295%	95.2381%
14	951	SSE	457	98.2495%	100%
15	1320	MSE	0.0100197	26.0394%	30.9524%
15	46053	MSEREG	0.0357696	94.5295%	95.2381%
15	454	SSE	457	95.186%	95.2381%
16	641	MSE	0.0148995	38.7309%	38.0952%
16	50000	MSEREG	0.0355515	94.7484%	95.2381%
16	312	SSE	457	97.8118%	97.619%
17	612	MSE	0.008841	22.9759%	28.5714%
17	50000	MSEREG	0.0353523	94.5295%	95.2381%
17	220	SSE	457	95.186%	95.2381%
18	811	MSE	0.0124584	32.3851%	26.1905%
18	50000	MSEREG	0.0351832	94.5295%	95.2381%
18	640	SSE	457	98.2495%	95.2381%
19	959	MSE	0.0102703	26.6958%	30.9524%
19	50000	MSEREG	0.0350666	94.9672%	95.2381%
19	866	SSE	433	94.7484%	95.2381%
20	698	MSE	0.0130482	33.9168%	35.7143%
20	50000	MSEREG	0.0349551	94.7484%	95.2381%
20	295	SSE	457	95.186%	95.2381%
21	744	MSE	0.00488559	12.6915%	16.6667%
21	50000	MSEREG	0.0322349	21.663%	23.8095%
21	999	SSE	457	95.6236%	95.2381%
22	602	MSE	0.0047173	12.2538%	14.2857%
22	13757	MSEREG	0.0315772	16.6302%	19.0476%
22	764	SSE	457	98.2495%	100%

Appendice B: File del progetto

Di seguito viene riportata una lista dei file e delle cartelle usate:

- `./addNoise.m`
 - Aggiunge dei quadrati bianchi o neri all'immagine (la dimensione dipende da quella dell'immagine);
 - Argomenti:
 - immagine originale
 - percentuale di intensità del disturbo [0, 1];
 - Ritorno: immagine modificata
- `./errorFun.m`
 - Utilizzata come funzione di performance con l'algoritmo *trainlm*;
 - Argomenti: equivalenti a *mse*;
- `./evaluateNetwork.m`
 - Data una rete, calcola l'errore percentuale commesso su un set di dati;
 - Argomenti:
 - rete neurale da valutare;
 - ingressi per la rete;
 - uscite desiderate;
 - Ritorno: errore percentuale commesso;
- `./getFileList.m`
 - Ritorna la lista dei file in una cartella;
 - Argomenti:
 - il path della cartella da analizzare;
 - Ritorno: lista dei nomi dei file contenuti e il loro numero totale;
- `./getImage.m`
 - Carica e ritorna un'immagine;
 - Argomenti:
 - path dell'immagine;
 - nuova altezza;
 - nuova larghezza;
 - eventuale coefficiente di disturbo;
 - Ritorno: immagine ridimensionata e in bianco e nero;
- `./getImageSet.m`
 - Carica tutte le immagini di una cartella;
 - Argomenti:
 - path della cartella da esaminare;
 - nuova altezza per le immagini;
 - nuova larghezza per le immagini;
 - eventuale disturbo da applicare;
 - Ritorno: immagini elaborate come ingressi e uscite per la rete neurale;
- `./getImageSetCV.m`
 - Carica tutte le immagini di una cartella, creando le matrici da usare con il cross

- validation;
- Argomenti:
 - path della cartella da esaminare;
 - nuova altezza per le immagini;
 - nuova larghezza per le immagini;
 - indice da usare per il cross validation [1, folder];
 - eventuale coefficiente di disturbo da applicare ad ogni immagine;
- Ritorno: ingressi e uscite per l'addestramento e per il test;
- ./getOutput.m
 - Dato un indice di lettere ritorna l'uscita che dovrebbe dare la rete;
 - Argomenti:
 - carattere [0, 64];
 - Ritorno: output desiderato;
- ./imageAdjust.m
 - Applica crop a tutte le immagini in una cartella;
 - Argomenti:
 - path della cartella sorgente da cui prelevare le immagini;
 - path della cartella destinazione in cui salvare le immagini;
- ./getMaxSpread.m
 - Ritorna la massima distanza tra tutte le immagini;
 - Argomenti:
 - path della cartella principale da cui leggere le immagini;
 - Ritorno: spread massimo;
- ./getMinSpread.m
 - Ritorna, per ogni cartella di immagini, la distanza minima, massima e media e informazioni relative alla somiglianza delle codifiche;
 - Argomenti:
 - path della cartella principale delle immagini da leggere;
 - Ritorno: statistiche;
- ./testFun.m
 - Crea, addestra e ritorna una rete feed forward backpropagation;
- ./testSup.m
 - Genera i piani di test per reti backpropagation e rbfn;
- ./gui/*
 - File relativi alla gui;
- ./gui/nns/*
 - File contenenti le reti neurali che la gui può caricare;
- ./dataSet/*
 - Cartelle delle immagini, divise per lettera e indice.

Appendice C: ErrorFun.m

Inserisco per completezza una funzione di performance, utilizzabile con *trainlm*, da me realizzata e usata nella fase di test.

Non fa altro che ritornare il rapporto tra numero di simulazioni errate e numero totale di simulazioni effettuate.

```
function errValue = errorFun( errPar, x, y, z )
    %gestisco la richiesta di informazioni sul tipo di parametri da passare
    if strcmp(errPar, 'pdefaults')
        errValue=struct;
        return;
    end
    %definizione strutture dati
    errPar=errPar{1,1};
    n_rows=size(errPar,1);
    n_cols=size(errPar,2);
    errValue=0;
    %esecuzione
    for i=1:n_cols
        %errPar=t -y
        %y ha valori nell'intervallo [1,0] quindi errPar avrà tutti valori negativi eccetto quello con
        indice
        %compet(t) che ha valore nell'intervallo [1,0], che è il massimo
        app=errPar(:,i);
        [val, I]=max( app );
        %ricostruisco t
        t=zeros(n_rows,1);
        t(I)=1;
        y=t-errPar;
        pos=find( compet( y ) ==1 );
        %I è l'uscita desiderata mentre pos è l'uscita della rete
        if(pos~=I)
            errValue=errValue+1;
        end
    end
    errValue=errValue/n_cols;
```

Siccome *mse* e *sse* calcolano una funzione sull'errore compiuto su ogni neurone di uscita, ho pensato che se avessi avuto una funzione che si focalizzasse solo sul numero di errori compiuto, e quindi scegliesse in base all'obiettivo di minimizzare tale errore, si potrebbe raggiungere più velocemente un buon addestramento.

La pratica ha mostrato che le prestazioni sono più basse rispetto a *mse* e *sse*.

Penso che sia dovuto al fatto che, scegliendo solo in base al neurone corretto, si trascura la condizione che gli altri devono essere addestrati ad essere zero.

Appendice D: Calcolo spread minimo

Ho memorizzato ogni tipologia di lettera, o quasi, in una cartella distinta.

Per ogni cartella ho riportato la distanza massima e media tra tutte le sue possibili coppie di esempi.

A1 max:3.1623 mean:2.1305	B2 max:3.4641 mean:1.9646	D1 max:3.6056 mean:2.0137	E2 max:2.4495 mean:1.5779	G1 max:3.1623 mean:1.9982	O1 max:2.8284 mean:2.0605	P2 max:2.6458 mean:1.5764
A2 max:3.6056 mean:2.5116	Q3 max:3.873 mean:2.8247	G2 max:3.1623 mean:2.2516	H3 max:2 mean:1.32	J2 max:2.2361 mean:1.4943	L2 max:2.4495 mean:1.4071	M3 max:2.6458 mean:1.5857
A3 max:3.7417 mean:2.2542	S2 max:3 mean:2.3338	T2 max:2.4495 mean:1.4322	U3 max:2.4495 mean:1.4903	V4 max:3.3166 mean:2.2055	X2 max:3 mean:1.7677	Y3 max:2.2361 mean:1.5841
C2 max:3.873 mean:2.4524	D3 max:4.4721 mean:2.6882	F2 max:2.8284 mean:1.6705	G3 max:3.873 mean:2.7025	I1 max:1.7321 mean:0.4663	K1 max:3.1623 mean:1.7146	L3 max:2.8284 mean:1.6467
V1 max:2.2361 mean:1.3889	N1 max:3 mean:1.9292	O2 max:2.4495 mean:1.4675	P3 max:2.8284 mean:1.5182	R1 max:2.6458 mean:1.779	S3 max:3 mean:1.8802	T3 max:2.2361 mean:1.5827
W1 max:2.8284 mean:2.069	Z1 max:3 mean:1.8398	B1 max:2.6458 mean:1.7305	C3 max:3 mean:1.8659	E1 max:2.8284 mean:1.7638	I2 max:1.4142 mean:0.2357	K2 max:3.7417 mean:2.204
S4 max:3 mean:2.0932	F3 max:2.4495 mean:1.5995	H1 max:3.3166 mean:1.6448	M1 max:2.4495 mean:1.6624	N2 max:3.1623 mean:1.8684	O3 max:2.8284 mean:1.6999	Q1 max:3.873 mean:2.6147
Z3 max:2.6458 mean:1.8232	X3 max:2.4495 mean:1.6666	R2 max:3.4641 mean:2.3504	Z2 max:2.6458 mean:1.6814	Y1 max:2 mean:1.2796	W2 max:3.6056 mean:1.8188	V2 max:2.8284 mean:1.956
U2 max:2 mean:1.2934	V3 max:3.873 mean:1.7251	X1 max:2.4495 mean:1.5657	Y2 max:2.2361 mean:1.4415	D2 max:3 mean:2.0676	U1 max:2.2361 mean:1.3828	F1 max:3.1623 mean:1.9358
H2 max:1.7321 mean:1.1671	L1 max:2.4495 mean:1.5998	M2 max:2.8284 mean:1.9354	N3 max:3.3166 mean:1.9598	P1 max:2.4495 mean:1.6099	Q2 max:3.3166 mean:2.1495	S1 max:2.6458 mean:1.6331
J1 max:2.2361 mean:1.3385	T1 max:2.8284 mean:1.622	C1 max:1.7321 mean:1.1546				

Appendice E: Dati addestramento backpropagation con rumore

Di seguito sono riportati i grafici dell'errore sul training set e testing set compiuti dalla rete backpropagation in presenza di rumore, con l'indice che varia da 0.1 a 1.

La rete è stata addestrata con training set con e senza disturbo e testata sul testing set con e senza disturbo.

