

01GPW - Gestione della conoscenza e intelligenza artificiale

Docente: Elio Piccolo



**Modello di rete neurale**  
**MULTY LAYER PERCEPTRON (MLP)**  
**con tecniche Feed-Forward e Back-Propagation**  
**per il riconoscimento ottico dei caratteri**

**Gianfranco Politano – matr.99420**



<b>Premesse</b> .....	<b>4</b>
Obiettivo .....	4
Modello Concettuale di Rete Neurale.....	4
Ulteriori modelli valutati .....	4
Progetto Software .....	4
<b>Il progetto</b> .....	<b>5</b>
<b>Apprendo.exe</b> .....	<b>6</b>
Flow Chart.....	6
Premessa – Gli input.....	7
Premessa – Gli output.....	8
Funzionamento .....	9
• Inizializzazione .....	9
• Iterazione per apprendimento.....	9
• Stampa .....	13
<b>Eseguo.exe</b> .....	<b>14</b>
Flow Chart.....	14
Premessa – Gli input.....	15
Premessa – Gli Output.....	15
Funzionamento .....	16
• Inizializzazione .....	16
• Propagazione alle uscite .....	16
• Stampa .....	16
<b>Tool.exe</b> .....	<b>17</b>
Flow Chart.....	17
Premessa – Gli input.....	18
Premessa – Gli Output.....	18
Funzionamento .....	18

# Premesse

## **Obiettivo**

Riconoscimento ottico dei caratteri

## **Modello Concettuale di Rete Neurale**

- **Rete Multi-Layer Perceptron - Feed-Forward con Back-Propagation**

Ho scelto questo modello di rete vista la capacita' di approssimazione universale di cui essa gode. Applicando all'MLP la tecnica di discesa a gradiente (chiamata anche regola di Back-Propagation) si riesce inoltre a minimizzare la funzione errore tra l'uscita effettiva e l'uscita desiderata, ottenendo così una supervisione selettiva e completa in fase di apprendimento.

## **Ulteriori modelli valutati**

Prima di operare la decisione sono stati valutati anche i seguenti modelli di rete:

- **Rete di Kohonen**

Rete in grado di apprendere in modo non supervisionato dati non etichettati e raggrupparli in celle di adiacenza. Ho scartato questo modello perché troppo approssimativo e soprattutto non supervisionato in fase di apprendimento

- **Rete di Hopfield**

Rete di neuroni completamente connessi in grado di funzionare come autoassociatori o in grado di affrontare problemi di ottimizzazione combinatoria. Questo modello di rete si prestava al riconoscimento dei caratteri, ma essendo un rigido autoassociatore avrebbe potuto creare dei problemi nella riconduzione di elementi disturbati agli originali.

## **Progetto Software**

La simulazione della rete neurale prevede 3 programmi scritti in linguaggio C

- **Apprendo.exe** – questo programma simula i processi di apprendimento di una rete neurale MLP con tecnica Back-Propagation, in uscita restituisce un file output.txt che contiene i Bias (dei nodi hidden e output) e i pesi (delle connessioni input-hidden e hidden-output).
- **Eseguo.exe** – questo programma sfruttando il file dei pesi e dei bias dato in output dopo l'apprendimento propaga alle uscite un pattern in ingresso associandolo all'output più simile.
- **Tool.exe** – Genera automaticamente un vettore con rumore definito dall'utente in base al pattern specificato. Salva il risultato nel file riconosco.txt

## **Il Progetto**

Verranno analizzati i tre programmi presentati a livello di codice e di funzionalità

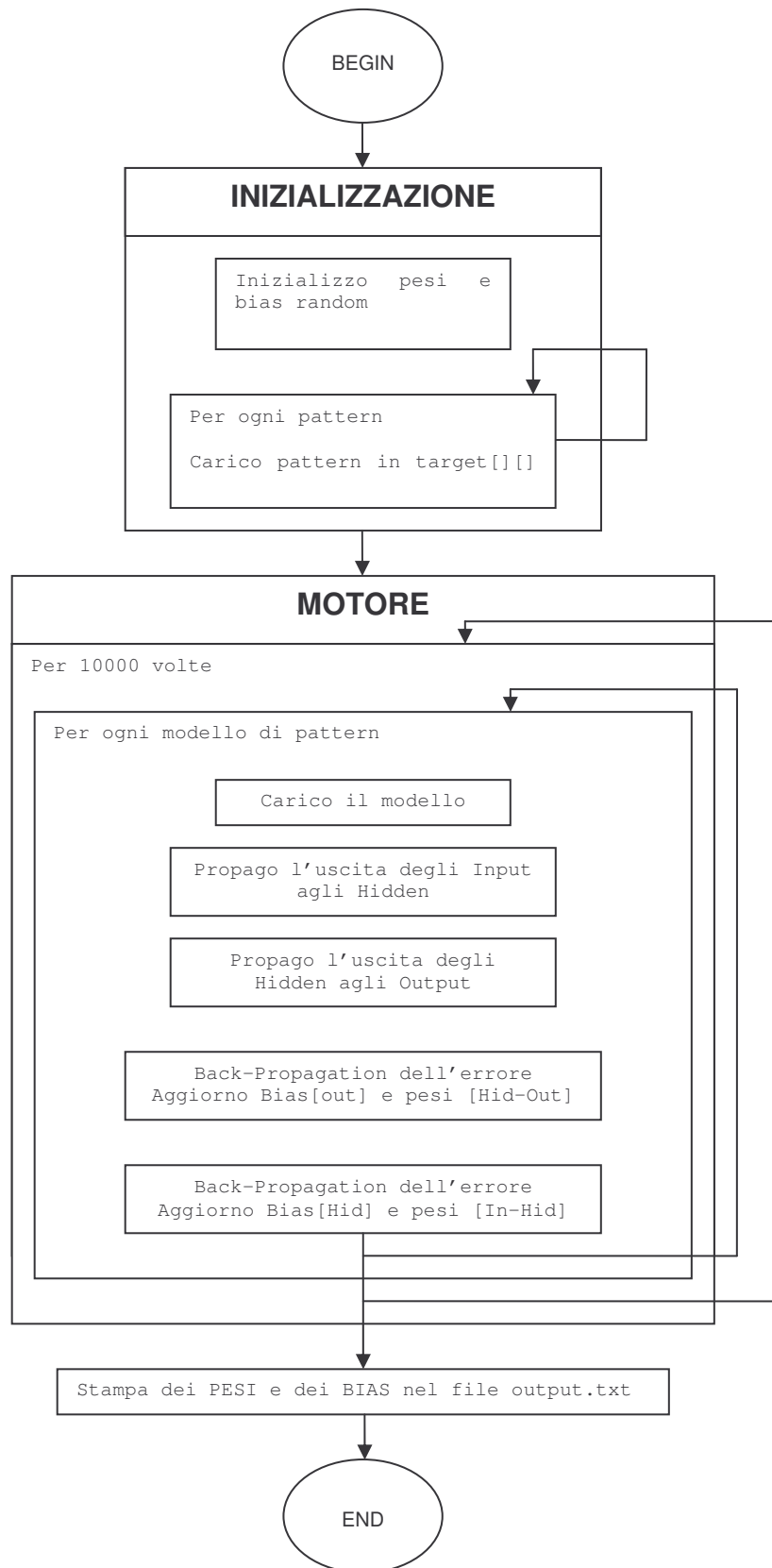
- **Apprendo.exe – Descrizione**
- **Eseguo.exe – Descrizione**
- **Tool.exe – Descrizione**

Il codice completo dei tre programmi viene inserito in allegato

- **Allegato 1 – Codice Apprendo.exe**
- **Allegato 2 – Codice Eseguo.exe**
- **Allegato 3 – Codice Tool.exe**

# Apprendo.exe

## Flow Chart



## Premessa – Gli input

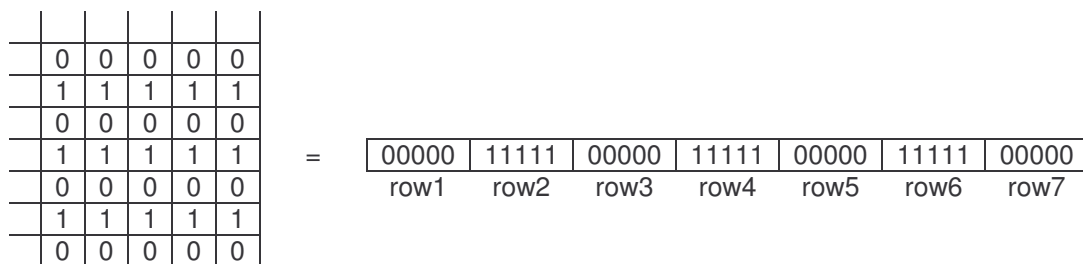
Si suppone che l'input da noi rilevato corrisponda ad una matrice di punti di dimensione 7x5 (righe,colonne).

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

Dopo l'acquisizione ottica (es. scanner) viene circoscritta la zona corrispondente al primo carattere e ad essa sovrapposta la matrice 7x5 che con una funzione OR, cella per cella, modifica il suo contenuto ad 1 laddove viene percepito il contorno del carattere



La matrice così ottenuta viene poi trasformata in un vettore affiancando le varie righe ed ottenendo quindi un vettore di 7x5 elementi



La simulazione prevede l'uso di n° 5 pattern predefiniti, nella forma

- Pattern\_1: 000000000000000111110010101011110000
- Pattern\_2: 000001111110000000000000000000000111
- Pattern\_3: 11111111111111111111000000000000000000
- Pattern\_4: 000000000111111111111010101111111111
- Pattern\_5: 111111111000000001111111111000000000

Questi pattern saranno associati ciascuno alla rispettiva uscita in caso di riconoscimento

L'input atteso dal programma Apprendo.exe consta di un file .txt, **vettori.txt**, contenente i 5 PATTERN DI PARTENZA sopra descritti.

## **Premessa – Gli output**

Il programma Apprendo.exe genera in uscita un file txt formattato per il successivo uso, **output.txt**, contenente l'apprendimento, ovvero tutti i PESI e i BIAS adattati ai pattern proposti in fase di apprendimento, in modo da minimizzare l'errore in uscita.

Il file Output.txt appare nella forma:

```
I BIAS output sono:
1.488223    -0.184742    -0.687037    -2.231103    1.839168    -1.003948

I BIAS hidden sono:
0.153173    -2.757319    -0.377542    1.836630    1.097314    2.282515
3.068789    0.384806    -0.718557    1.220544    1.786689    1.792840
0.490414    0.278190    -1.303098    -1.486624    1.823734    0.153052
0.145938    -2.270676    -1.108089    1.205218    -1.654399    -2.881255
2.470200

I Pesi dell'output numero X nei confronti degli hidden sono:
0.771481    -0.924876    -0.210810    1.076408    1.937539    -4.848616
2.029350    -0.848046    1.839643    0.452472    -0.850767    -1.013011
0.475741    -1.505333    -0.287280    -3.988389    -1.133060    -1.666473
-4.044342    0.644399    -2.665715    -1.156071    0.884532    -1.442835
1.319074

... eseguito per ogni output

I Pesi dell'hidden numero X nei confronti degli input sono:
-1.239142    0.545734    -0.026181    0.989453    2.104448    -2.226511
-0.426702    -0.814902    0.657247    -2.367252    -1.040688    -0.787808
-2.355096    -0.386662    0.727083    1.967576    0.705333    2.041921
1.943157    -1.104187    1.058092    -0.548285    -0.099492    0.546715
1.811062    -1.999773    0.324029    1.254278    -1.542472    -1.960576
2.300341    1.053195    -1.271624    -0.090633    0.583502

... eseguito per ogni hidden
```

## **Funzionamento**

- **Inizializzazione**

Il programma contiene una prima fase di inizializzazione

```
(1) init();
(2) for (goal=0;goal<PATTERN_TARGET;goal++)
(3) { target_init(); }
```

(1) In questa fase vengono inizializzati i pesi dei collegamenti Hidden-Output, i pesi dei collegamenti Input-Hidden, i Bias degli Output e i Bias degli Hidden. L'inizializzazione è completamente affidata ad una funzione random del tipo:

```
(a) srand ( time(NULL) );
(b) m1=(rand()%10000+1)/10000;
(c) m2=(rand()%2+1);
(d) if (m2>1) {m1=m2*(-1);}
```

Il cui scopo è restituire in m1 un valore random compreso fra 0 ed 1 con 4 cifre decimali ed in m2 un valore compreso fra 0 e 2 in modo da dare il 50% di possibilità (if m2>1) di rendere negativo il valore m1, estendendo il range di m1 fra -1 e +1.

(2,3) Per ognuno dei 5 pattern viene poi richiamata la funzione (3)target\_init che si occupa di caricare dal file vettori.txt i pattern precedentemente inseriti.

- **Iterazione per apprendimento**

L'apprendimento vero è proprio della rete è però affidato al loop delle funzioni di Feed-Forward e Back-Propagation, che hanno lo scopo di minimizzare la funzione di errore tra l'uscita effettiva e l'uscita desiderata.

```
(4) for (k=0;k<10000;k++){
(5)     for (targ=0;targ<PATTERN_TARGET;targ++){
(6)         carica(target[targ]);
(7)         propaga_hidden();
(8)         propaga_out();
(9)         errore_output(targ);
(10)        errore_hidden();     } }
```

Viene generato un ciclo di 10000 iterazioni con cui di volta in volta viene ottimizzata la propagazione di errore cercando di minimizzarla quanto più possibile.

```
(6) carica(target[targ]);
```

Carica in ognuno dei 35 nodi di input una cella del pattern di addestramento, ciclando per ognuna delle 10000 iterazioni tutti i possibili pattern di addestramento. In questo modo gli input e le uscite sono sempre diversi e si è rilevata una maggiore uniformità della propagazione d'errore ed una migliore riduzione del medesimo.

(7) `propaga_hidden()`

```
for (i=0; i<HIDDEN; i++) {
    sum=bias_h[i];
    for (j=0; j<INPUT; j++) {
        sum=sum+(pesi_h_i[i][j]*((float)input[j])); }
    hidden[i]=squash(sum); }
```

Che sviluppa la formula di propagazione delle uscite input ai nodi hidden:

$$Hidden_{Hid} = Squash \left( bias_{Hid} + \sum_{In} Peso_{In-Hid} * Uscita_{In} \right)$$

(8) `propaga_out()`

```
for (i=0; i<OUTPUT; i++) {
    sum=bias_o[i];
    for (j=0; j<HIDDEN; j++) {
        sum=sum+(pesi_o_h[i][j]*hidden[j]); }
    output[i]=squash(sum); }
```

Che sviluppa la formula di propagazione delle uscite degli hidden ai nodi output:

$$Output_{Out} = Squash \left( bias_{Out} + \sum_{Hid} Peso_{Hid-Out} * Uscita_{Hid} \right)$$

(Nota) `float squash (float X)`

```
result=1/(1+exp(X*(-1)));
return(result);
```

La funzione `squash` modella il segnale in uscita di una cella neurale a seguito della computazione degli ingressi.

Il segnale in uscita è una sigmode, con soglie comprese fra 0 ed 1, di funzione:

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

### (9) errore\_output(targ)

```
for (out=0;out<OUTPUT;out++) {
    delta_o[out]= (
        (voluto_out [targ] [out]-output [out])
        * (output [out])
        * (1-output [out])
    );
    for (hid=0;hid<HIDDEN;hid++) {
        temp_p=pesi_o_h[out] [hid];
        pesi_o_h[out] [hid]= (
            pesi_o_h[out] [hid]
            + (ETA*delta_o[out]*hidden[hid])
            + (ALFA*(pesi_o_h[out] [hid]-old_pesi_o_h[out] [hid]))
        );
        old_pesi_o_h[out] [hid]=temp_p;
    }
    temp_b=bias_o[out];
    bias_o[out]= (
        bias_o[out]
        + (ETA*delta_o[out])
        + (ALFA*(bias_o[out]-old_bias_o[out]))
    );
    old_bias_o[out]=temp_b;
}
```

Sviluppa la formula di propagazione all'indietro dell'errore rilevato tra gli output effettivi e quelli desiderati. Questo errore viene inserito nella delta\_o e va a modificare i pesi dei collegamenti Hidden-Output e i Bias degli Output:

$$\delta_{Out} = (Desiderato_{Out} - Output_{Out}) * Output_{Out} * (1 - Output_{Out})$$

$$\Delta Peso_{Out-Hid} = (\eta * \delta_{Out}) + (\alpha * (Peso(t)_{Out-Hid} - Peso(t-1)_{Out-Hid}))$$

$$\Delta Bias_{Out} = (\eta * \delta_{Out}) + (\alpha * (Bias(t)_{Out} - Bias(t-1)_{Out}))$$

$$Bias_{Out} = Bias_{Out} + \Delta Bias_{Out}$$

$$Peso_{Out-Hid} = Peso_{Out-Hid} + \Delta Peso_{Out-Hid}$$

```

(10) errore_hidden()

for (hid=0;hid<HIDDEN;hid++) {
  for (out=0;out<OUTPUT;out++) {
    h_sum=(h_sum+(delta_o[out]*pesi_o_h[out][hid]));
  }
  delta_h=(hidden[hid]*(1-hidden[hid])*h_sum);
  h_sum=0;
  for (in=0;in<INPUT;in++){
    h_temp_p=pesi_h_i[hid][in];
    pesi_h_i[hid][in]=(
      pesi_h_i[hid][in]
      +(ETA*delta_h*(float)input[in])
      +(ALFA*(pesi_h_i[hid][in]-old_pesi_h_i[hid][in]))
    );
    old_pesi_h_i[hid][in]=h_temp_p;
  }
  h_temp_b=bias_h[hid];
  bias_h[hid]=(
    bias_h[hid]
    +(ETA*delta_h)
    +(ALFA*(bias_h[hid]-old_bias_h[hid])));
  old_bias_h[hid]=h_temp_b;
}
}

```

Sviluppa la formula di propagazione all'indietro dell'errore rilevato tra gli output effettivi e quelli desiderati. Questo errore viene inserito nella  $\delta_h$  (dipendente dalla precedente  $\delta_o$ ) e va a modificare i pesi dei collegamenti Input-Hidden e i Bias degli Hidden:

$$h\_sum = \left( \sum_{Out} \delta_{Out} * \text{Pesi}_{Out-hid} \right)$$

$$\delta_h = h\_sum * \text{Hidden}_{Hid} * (1 - \text{Hidden}_{Hid})$$

$$\Delta \text{Peso}_{Hid-In} = (\eta * \delta_h * \text{Input}_{In}) + (\alpha * (\text{Peso}(t)_{Hid-In} - \text{Peso}(t-1)_{Hid-In}))$$

$$\Delta \text{Bias}_{Hid} = (\eta * \delta_h) + (\alpha * (\text{Bias}(t)_{Hid} - \text{Bias}(t-1)_{Hid}))$$

$$\text{Peso}_{Hid-In} = \text{Peso}_{Hid-In} + \Delta \text{Peso}_{Hid-In}$$

$$\text{Bias}_{Hid} = \text{Bias}_{Hid} + \Delta \text{Bias}_{Hid}$$

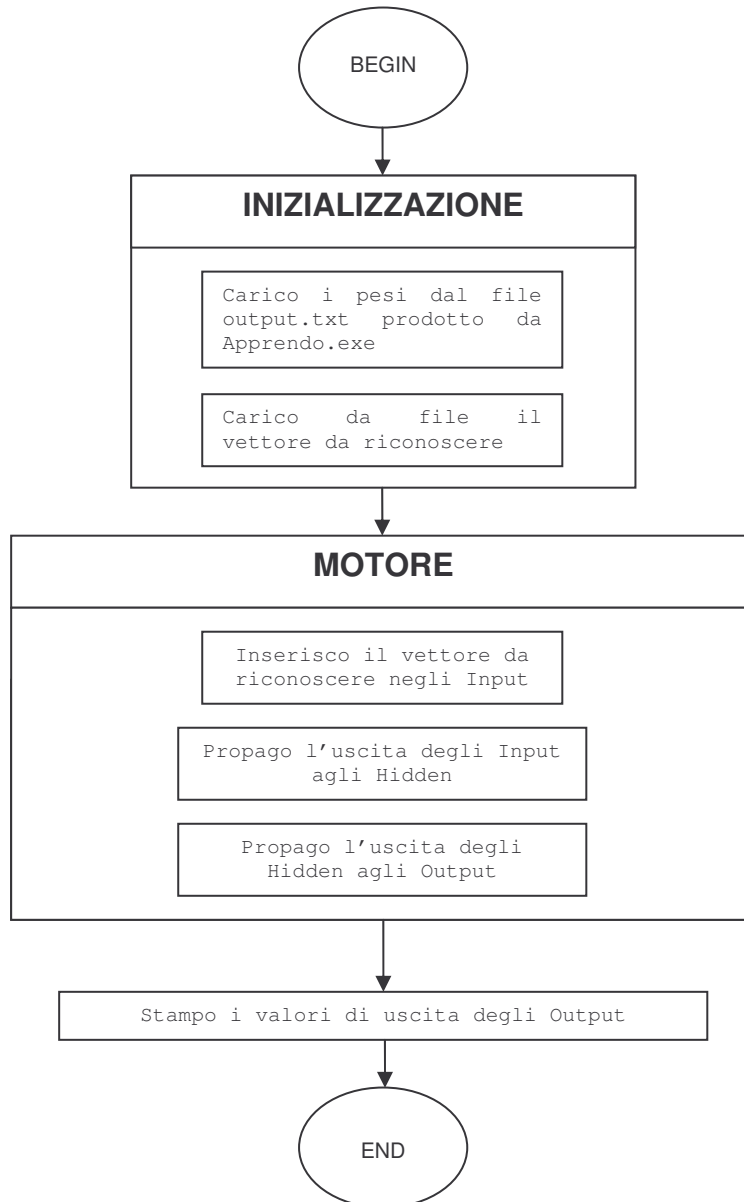
- **Stampa**

```
(1) stampa();
```

(1) La funzione `(1) stampa();` stampa sul file `output.txt` i BIAS ed i PESI risultanti dopo l'intero processo di apprendimento e quindi dopo il processo di minimizzazione dell'errore in uscita. La formattazione del file segue uno schema predefinito in modo da assicurarne la compatibilità con il programma `Eseguo.exe`, che a sua volta userà il file `output.txt` per utilizzare i valori di BIAS e PESI in esso contenuti.

# Eseguo.exe

## Flow Chart



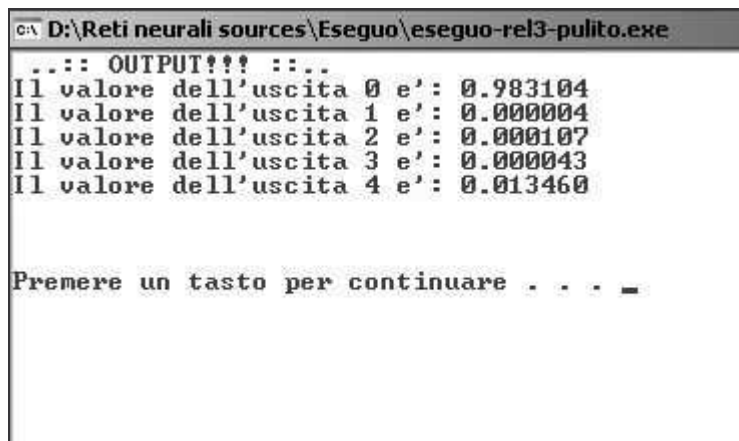
## **Premessa – Gli input**

Gli input attesi dal programma Eseguo.exe sono due file .txt

- il primo, **output.txt**, contiene i PESI e i BIAS risultanti dall'apprendimento della rete, la formattazione del file deve essere rigida per il corretto caricamento.
- il secondo, **riconosco.txt**, contiene il pattern da riconoscere che può essere un pattern "pulito" o un pattern degradato da un certo rumore, cosa che simula un caso più simile alla realtà e sfrutta appieno le capacità adattative dell'MLP.

## **Premessa – Gli Output**

L'output generato dall'esecuzione del programma è una schermata del tipo:



```

D:\Reti neurali sources\Eseguo\eseguo-rel3-pulito.exe
...:: OUTPUT!!! ::..
Il valore dell'uscita 0 e': 0.983104
Il valore dell'uscita 1 e': 0.000004
Il valore dell'uscita 2 e': 0.000107
Il valore dell'uscita 3 e': 0.000043
Il valore dell'uscita 4 e': 0.013460

Premere un tasto per continuare . . . _

```

dove, a seguito della propagazione del pattern di input, vengono visualizzati i valori in uscita da ogni cella di output,

I valori di uscita saranno compresi (come previsto dalla funzione squash) fra 0 e 1 e l'output col valore di uscita più alto corrisponderà alla risposta della rete. In caso di vettori molto disturbati la rete può essere indecisa e rispondere con soglie medie su più uscite, oppure effettuare un riconoscimento sbagliato; al contrario, se il riconoscimento avviene correttamente e senza "incertezze", si genera solitamente un output  $\sim=1$  per l'uscita corrispondente, e output  $\sim=0$  per le rimanenti uscite.

## **Funzionamento**

- **Inizializzazione**

Il programma contiene una prima fase di inizializzazione

```
(1) init();  
(2) carica_vet();
```

(1) Alla funzione (1) `init()`; è affidato il compito di caricare nelle variabili di programma i valori dei PESI e dei BIAS acquisiti dal file `output.txt`. Il file `output.txt` deve essere formattato secondo lo schema del programma `Apprendo.exe` affinché il caricamento dei valori sia eseguito correttamente.

(2) La funzione (2) `carica_vet()`; si occupa di caricare il pattern da riconoscere dal file `riconosco.txt` ed inserirlo nei nodi di Input della rete.

- **Propagazione alle uscite**

Dopo la fase iniziale di inizializzazione la rete propaga alle uscite i valori caricati nelle celle di Input con la metodologia Feed-Forward.

```
(3) propaga_hidden();  
(4) propaga_out();
```

(3, 4) Le funzioni (3) `propaga_hidden()`; (4) `propaga_out()`; sono identiche a quelle usate nel programma `Apprendo.exe` e svolgono la funzione di propagare con la metodologia Feed-Forward il pattern in ingresso prima ai nodi Hidden, poi ai nodi Output. Non sono presenti le funzioni di Back-Propagation poichè i BIAS e i PESI acquisiti precedentemente sono quelli risultanti dall'apprendimento e quindi si considera l'errore già minimizzato.

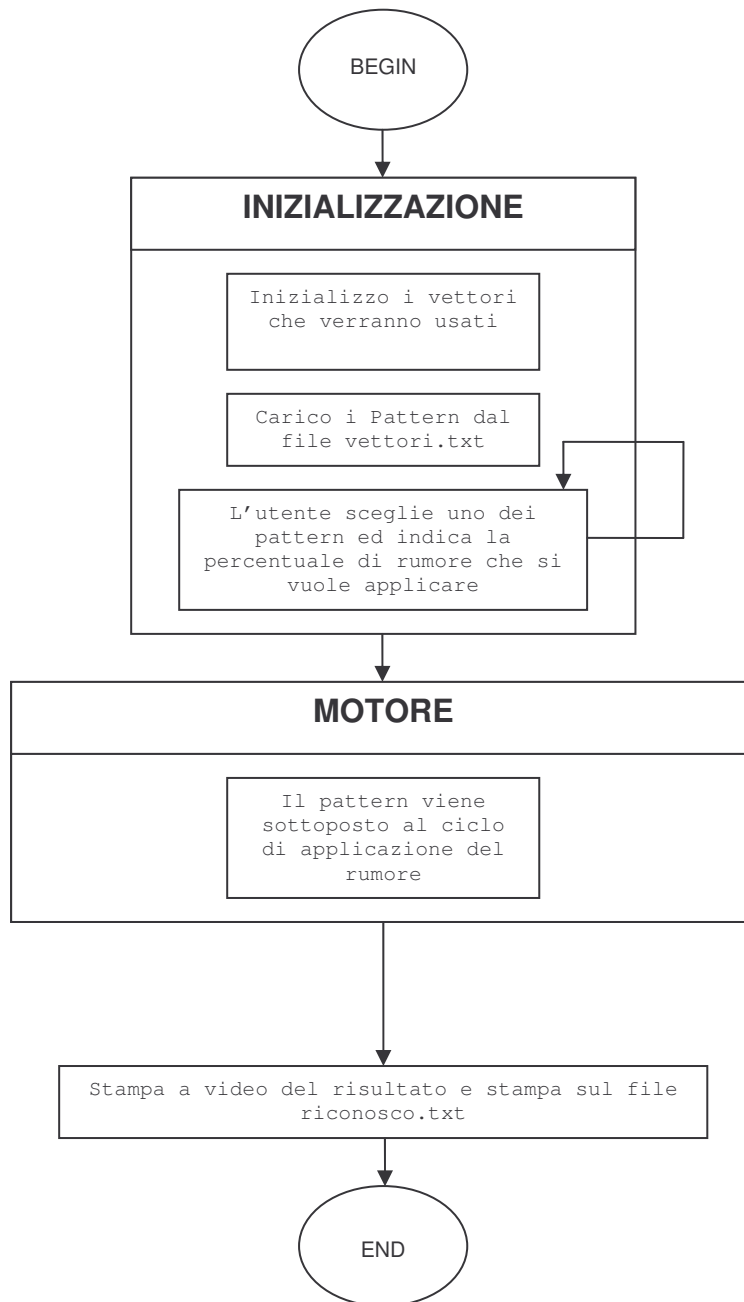
- **Stampa**

```
(5) stampa();
```

(5) La funzione (5) `stampa()`; stampa a schermo le uscite dei nodi Output

# Tool.exe

## Flow Chart



## **Premessa – Gli input**

L'input atteso dal programma Tool.exe è un file .txt contenente i pattern base con cui si è istruita la rete. Il file di nome vettori.txt è lo stesso che utilizza il programma Apprendo.exe

## **Premessa – Gli Output**

L'output generato dall'esecuzione del programma è una schermata contenente il risultato dell'applicazione del rumore al vettore ed una copia del vettore degradato nel file riconosco.txt. Il file riconosco.txt verrà poi utilizzato dal programma Eseguo.exe come input da analizzare.

## **Funzionamento**

L'unica funzione degna di nota del programma Tool.exe è il ciclo con cui si randomizza l'applicazione del rumore.

```
(1)   for (j=0; j<35; j++) {
(2)       if ((m=rand()%100+1)>RUMORE)
(3)       vettore_tmp[j]= vettore[numpat-1][j];
(4)       else{
(5)           if (vettore[numpat-1][j]=='1')
(6)           vettore_tmp[j]='0';
(7)           else
(8)           vettore_tmp[j]='1';
```

Il cuore del ciclo è la funzione (2) che, per ogni elemento del vettore (per ogni j (1)), carica in m un valore random compreso fra 1 e 100, questo valore viene successivamente confrontato con la soglia di rumore scelta dall'utente (RUMORE) e nel caso sia

- maggiore di RUMORE: non viene eseguito alcunché, mantenendo il bit originale uguale al sorgente (3)
- minore di RUMORE: viene invertito il bit sorgente, simulando l'errore di acquisizione (5, 8)