Studio, implementazione e realizzazione di una rete neurale MLP riconfigurabile, processata tramite algoritmo di back propagation su scheda di sviluppo embedded Arduino Uno Rev.3

A cura di:
Andrea Valeriani
Federico Nassi
Matteo Maola

Indice generale

CAPITOLO 1
Rete Neurale

1.1 Modello generale di un Neurone

Un neurone è l'unità fondamentale di processamento dell'informazione ed è l'elemento cardine di una rete neurale.

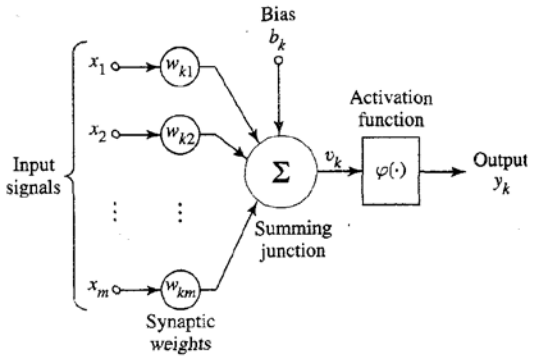


Fig.1

Il modello neurale include un ingresso esterno, denominato bias b_k che permette di incrementare o decrementare il valore di ingresso della funzione di attivazione. Matematicamente possiamo descrivere un neurone (k) con le due seguenti equazioni:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

$$y_k = \varphi(u_k + b_k)$$

dove gli x_i sono i segnali di ingresso, ω_{ki} sono i pesi sinaptici, u_k rappresenta la combinazione lineare in uscita dei segnali di ingresso,b_k rappresenta il bias,$\varphi()$ rappresenta la funzione di attivazione e y_k rappresenta il segnale di uscita del neurone.

Lo scopo di utilizzazione del bias è quello di applicare una trasformazione all'uscita u_k ed a seconda del suo segno quello di modificarne il valore come si vede nella seguente figura. $v_k=u_k + b_k$

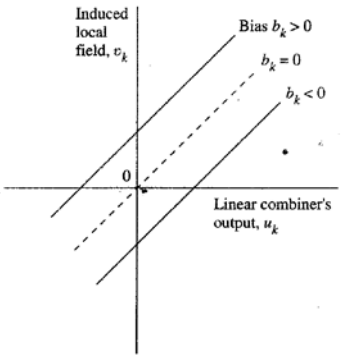


Fig.2

Quando $v_k > 0$ si attiva il neurone e gli ingressi opportunamente pesati superano il limite di attivazione indicato dal bias.
La funzione di attivazione, φ(.) è una funzione di tipo gradino o una funzione che ne approssima il comportamento e che sono descrittive del processo appena menzionato, utilizzata per limitare l'uscita tra 0 e 1. La funzione sigmoidale è la più comune forma di funzione di attivazione usata nella costruzione di reti neurali artificiali definita come una funzione strettamente crescente che mostra un bilanciamento tra un comportamento lineare ed uno non lineare ed è definita dalla seguente equazione:

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$

variando il parametro 'a' la funzione approssima curve differenti (fig.3) , all'infinito la funzione tende ad una semplice funzione di soglia ma a differenza di quest'ultima è differenziabile e assume valori continui tra 0 ed 1.

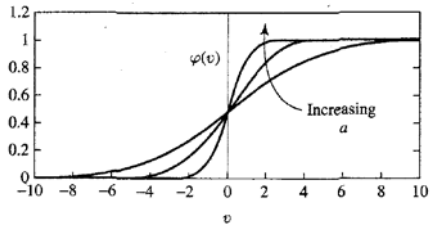


Fig.3

1.2 Multilayer perceptron (MLP)

Una rete neurale `e formata da una serie di unità elementari, cioè di singoli neuroni, collegati fra loro da connessioni.

Graficamente si rappresenta un neurone con un cerchio, mentre le connessioni fra i neuroni con frecce orientate, il cui verso indica la direzione del flusso di informazioni.

Una rete multistrato (Multilayer perceptron, MLP) `e costituita da uno strato di d neuroni di input e da uno o più strati nascosti (hidden layers), ognuno dei quali composti da un certo numero di neuroni e da uno strato di output.

Esistono varie architetture di rete tra cui quella che analizziamo qui , la rete" feed-forward", caratterizzata dal fatto che ogni neurone di uno strato è collegato ad ogni neurone dello strato successivo (fig 4), mentre non vi sono connessioni fra neuroni dello stesso strato, inoltre le relazioni sono unidirezionali.

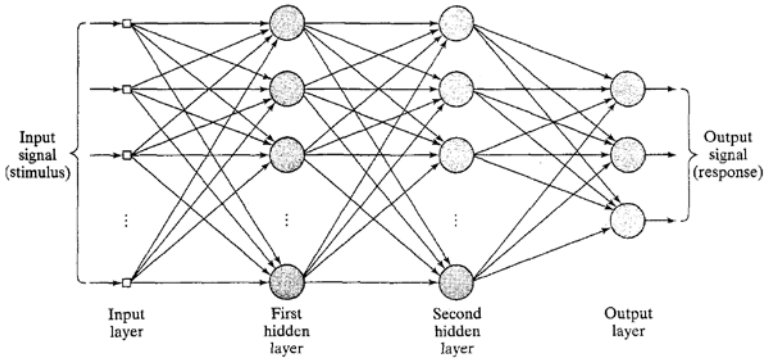


Fig.4

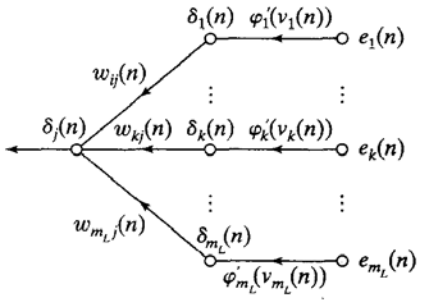
1.3 Algoritmo di Back-Propagation

L'algoritmo di back propagation è un algoritmo molto usato nell'ambito dell'addestramento delle reti neurali ed è lo stesso che è stato utilizzato qui per la seguente tesina.
Il funzionamento dell'algoritmo si basa sul settaggio dei pesi sinaptici ω in modo da minimizzare una funzione di errore (funzione di costo) descrivente la differenza tra i valori in uscita della nostra rete e quelli idealmente desiderati, forniti da *training set*. Inizialmente i valori dei pesi sono randomici e vengono aggiornati tramite il calcolo dell'errore.
L'algoritmo di Back Propagation si divide in due fasi la *forward pass* e la *backward pass*.

Nella prima fase i pesi sinaptici inseriti randomicamente rimangono invariati e vengono calcolati tutti i segnali provenienti dai neuroni [es. $y_j=\varphi(v_j(n))$], se il neurone si trova nel primo strato nascosto il pedice j si riferisce all'j-esimo ingresso della rete mentre se il neurone si trova nello strato di uscita della rete il pedice j riferisce al j-esimo terminale di uscita, questa uscita sarà confrontata con il valore desiderato $d_j(n)$ e di seguito verrà trovato l'errore $e_j(n)$.
A questo punto calcolato l'errore inzia la fase *backward pass,* che partendo dal layer di uscita attraverso il "passaggio" dell'errore indietro layer by layer (fig.5) muovendoci verso sinistra calcola ricorsivamente i δ (gradienti locali) aggiornando di conseguenza i pesi sinaptici in accordo con la *Deltarule*

$$\begin{pmatrix} \text{Weight} \\ \text{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{learning-} \\ \text{rate parameter} \\ \eta \end{pmatrix} \cdot \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \cdot \begin{pmatrix} \text{input signal} \\ \text{of neuron } j \\ y_i(n) \end{pmatrix}$$

Fig.5



CAPITOLO 2
Realizzazione

2.1 Arduino

Arduino è un progetto open source che permette la prototipazione rapida . È composto da una piattaforma hardware per il physical computing sviluppata presso l'Interaction Design Institute. Il nome della scheda deriva da quello di un bar di Ivrea (che richiama a sua volta il nome di Arduino d'Ivrea, Re d'Italia nel 1002) frequentato da alcuni dei fondatori del progetto.

Il sistema si basa su un circuito stampato che integra un microcontrollore con pin connessi alle porte I/O, un regolatore di tensione e quando necessario un'interfaccia USB che permette la comunicazione con il computer. A questo hardware viene affiancato un ambiente di sviluppo integrato per lo sviluppo di programmi con un linguaggio derivante da C e C++ chiamato Wiring.

Una scheda Arduino tipica consiste in un microcontroller a 8-bit AVR prodotto dalla Atmel, con l'aggiunta di componenti complementari per facilitarne l'incorporazione in altri circuiti. In queste schede sono usati chip della serie megaAVR - nello specifico i modelli ATmega8, ATmega168, ATmega328, ATmega1280 eATmega2560.
Nella figura sottostante il microcontrollore utilizzato nel nostro progetto

Fig.6

2.2 Obiettivo del progetto

Il nostro scopo è stato quello di implementare una rete neurale riconfigurabile in grado operare completamente in Arduino e quindi adattabile a qualsiasi problematica comune risolvibile esclusivamente con una rete neurale, cioè partendo da ingressi randomici arrivare ad uscite probabilisticamente esatte.
L'obiettivo principale è stato quello di addestrare la rete neurale con un algoritmo di *back propagation* che fosse contenuto completamente nella memoria di Arduino, senza dover ricorrere all'ausilio di hardware esterno quale quello di un pc.
A scapito di una ridotta velocità di calcolo si sono ottenuti risultati ottimi malgrado la semplicità stessa del microcontrollore.

2.3 Implementazione su microcontrollore

Come si vede in figura il primo passo è stato quello di assegnare gli ingressi alla rete neurale adeguati al tipo di rete che vogliamo strutturare (numero di ingressi, uscite, momentum constant,learning rate,pesi sinaptici ecc.)

Fig.7

Da un pin di arduino si sono ricavati i valori dei pesi sinaptici (random) a questo punto si è fatto un primo ciclo della rete e si è calcolato l'errore presente inizialmente. Tenendo conto di questo errore si è applicato l'algoritmo di back propagation e lo si è ripetuto finchè l'errore in uscita era sufficientemente piccolo o in alternativa dopo un determinato numero di cicli (nei nostri progetti abbiamo scelto un numero pari a 500).
L'ausilio di un terminale collegato alla porta COM ci ha permesso di mostrare a video i primi due cicli con i conseguenti errori e il ciclo finale con l'errore minimo con annessi i pesi sinaptici calcolati con l'algoritmo e la susseguente applicazione della rete addestrata ad un ingresso non presente nel training set.

CAPITOLO 3
Risultati ottenuti e sviluppi futuri

3.1 Risultati ottenuti

La rete neurale appena descritta è stata utilizzata per essere implementata all'interno di un dispositivo mobile per decidere autonomamente se avanzare oppure fermarsi.
Il prototipo da noi realizzato prevede l'inserimento della distanza a cui si trova il dispositivo mediante un array che simula un led a sette segmenti.
La forza del nostro prototipo sta nel fatto che riesce a trovare una logica di funzionamento in una serie di ingressi apparentemente casuali. Nella foto sottostante si vede il nostro dispositivo che simula un oggetto in movimento che si trova a 8 (cm) di distanza da un eventuale ostacolo e decide in maniera autonoma di proseguire (led verde) pur non avendo nel training set questo valore.

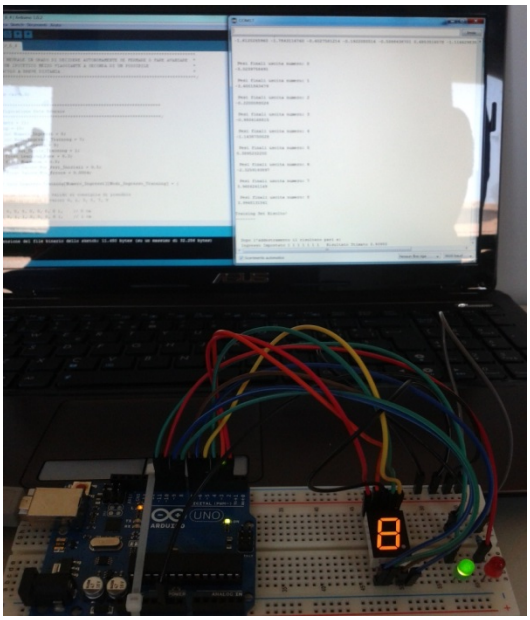


Fig.9

Un'altra applicazione da noi approntata è quella di implementare una rete neurale all'interno di una caldaia alimentata da tre diverse tipologie di combustibile aventi diverse temperature di combustione.

La nostra rete deciderà per quanto tempo accendere una pompa di raffreddamento nell'eventualità che la caldaia superi la temperatura massima consentita a seconda del tipo di combustibile. Immettendo all'interno della rete tramite training set che preveda l'accensione della pompa alla

temperatura massima e l’inattività della stessa ad una temperatura minima attraverso l’algoritmo di back propagation saprà, cosa fondamentale, affrontare situazioni intermedie.

3.2 Sviluppi futuri

Nel futuro prevederemo di implementare dei sensori che in tempo reale ci dicano in che situazioni ci troviamo in modo da fornire ingressi continui on line alla rete neurale la quale potrà cambiare uscita real time con tempi di risposta molto bassi, per questo un progetto futuro sarà quello di sviluppare questo algoritmo all’interno di un microprocessore con potenze di calcolo superiori quale il TI MSP430.

CAPITOLO 4
Codice Sorgente

4.1 Codice Sorgente

Ora riportiamo esplicitamente lo schetch riferito alla prima applicazione:

```
#include <math.h>

/*****************************************************************

 * Configurazione Rete Neurale

 *****************************************************************/

int Avanti = 11;

int Stop = 10;

const int Numero_Ingressi = 8;

const int Nodi_Ingresso_Training = 7;

const int Nodi_Interni = 9;

const int Nodi_Uscita_Training = 1;

const float Learning_Rate = 0.3;

const float Momentum = 0.9;

const float Valore_Max_Pesi_Iniziali = 0.5;

const float Valore_Min_Errore = 0.0004;

const byte Ingresso_Training[Numero_Ingressi][Nodi_Ingresso_Training] = {

  { 0, 0, 0, 0, 0, 0, 0 },   // 0 cm

  { 0, 1, 1, 0, 0, 0, 0 },   // 1 cm

  { 1, 1, 1, 1, 0, 0, 1 },   // 3 cm

  { 0, 1, 1, 0, 0, 1, 1 },   // 4 cm

  { 1, 0, 1, 1, 0, 1, 1 },   // 5 cm

  { 0, 0, 1, 1, 1, 1, 1 },   // 6 cm

  { 1, 1, 1, 0, 0, 0, 0 },   // 7 cm
```

```
  { 1, 1, 1, 0, 0, 1, 1 }     // 9 cm
};
const byte Obiettivo[Numero_Ingressi][Nodi_Uscita_Training] = {
 { 0 }, // Stop
 { 0 }, // Stop
 { 0 }, // Stop
 { 0 }, // Stop
 { 1 }, // Avanti
 { 1 }, // Avanti
 { 1 }, // Avanti
 { 1 }, // Avanti
};
const byte Ingresso[Numero_Ingressi][Nodi_Ingresso_Training] = {
 //{ 0, 0, 0, 0, 0, 0, 0 },    // 0 cm
 //{ 0, 1, 1, 0, 0, 0, 0 },    // 1 cm
 { 1, 1, 0, 1, 1, 0, 1 },    // 2 cm
 //{ 1, 1, 1, 1, 0, 0, 1 },    // 3 cm
 //{ 0, 1, 1, 0, 0, 1, 1 },    // 4 cm
 //{ 1, 0, 1, 1, 0, 1, 1 },    // 5 cm
 //{ 0, 0, 1, 1, 1, 1, 1 },    // 6 cm
 //{ 1, 1, 1, 0, 0, 0, 0 },    // 7 cm
 //{ 1, 1, 1, 1, 1, 1, 1 },    // 8 cm
 //{ 1, 1, 1, 0, 0, 1, 1 }     // 9 cm
};
/*****************************************************************
 * Fine Configurazione Rete Neurale inizio dichiarazioni variabili
 *****************************************************************/
int G=2;
```

```
int F=3;

int A=4;

int B=5;

int DP=6;

int C=7;

int D=8;

int E=9;

int i, j, p, q, r;

int ReportEvery1000;

int Indice_Randomizzato[Numero_Ingressi];

long  Cicli_Training;

float Rando;

float Error;

float Accum;

float Accumulatore;

float Hidden[Nodi_Interni];

float Uscita_Training[Nodi_Uscita_Training];

float Uscita[Nodi_Uscita_Training];

float Pesi_Livello_Interno[Nodi_Ingresso_Training+1][Nodi_Interni];

float Pesi_Uscita_Training[Nodi_Interni+1][Nodi_Uscita_Training];

float Variazione_Interna[Nodi_Interni];

float Variazione_Uscita_Training[Nodi_Uscita_Training];

float Cambio_Pesi_Livello_Interno[Nodi_Ingresso_Training+1][Nodi_Interni];

float Cambio_Pesi_Uscita_Training[Nodi_Interni+1][Nodi_Uscita_Training];


void setup(){

  pinMode(G, OUTPUT);

  pinMode(F, OUTPUT);
```

```
  pinMode(A, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(E, OUTPUT);
  pinMode(D, OUTPUT);
  pinMode(C, OUTPUT);
  pinMode(DP, OUTPUT);
  pinMode(Avanti, OUTPUT); // Set pin as OUTPUT
  pinMode(Stop, OUTPUT);
  Serial.begin(9600);
  randomSeed(analogRead(3));
  ReportEvery1000 = 1;
  for( p = 0 ; p < Numero_Ingressi ; p++ ) {
    Indice_Randomizzato[p] = p ;
  }
}
void loop (){
/*******************************************************
* Distanza scelta per testare la risposta della rete
*******************************************************/
  delay(2000);
   due();
/*************************************************************************
* Inizializzazione dei pesi del livello interno e variazione degli stessi
*************************************************************************/
  for( i = 0 ; i < Nodi_Interni ; i++ ) {
    for( j = 0 ; j <= Nodi_Ingresso_Training ; j++ ) {
      Cambio_Pesi_Livello_Interno[j][i] = 0.0 ;
      Rando = float(random(100))/100;
```

```
    Pesi_Livello_Interno[j][i] = 2.0 * ( Rando - 0.5 ) * Valore_Max_Pesi_Iniziali ;
   }
  }


/*****************************************************************************
* Inizializzazione dei pesi del livello esterno e variazione degli stessi
*****************************************************************************/
  for( i = 0 ; i < Nodi_Uscita_Training ; i ++ ) {
   for( j = 0 ; j <= Nodi_Interni ; j++ ) {
    Cambio_Pesi_Uscita_Training[j][i] = 0.0 ;
    Rando = float(random(100))/100;
    Pesi_Uscita_Training[j][i] = 2.0 * ( Rando - 0.5 ) * Valore_Max_Pesi_Iniziali ;
   }
  }
  Serial.println(" ");
  Serial.println("Stampa del TrainingSet, Validation Set e calcolo dell'uscita iniziale ");
  Serial.println(" ");
  Serial.println("senza l'applicazione dell'algoritmo di BackPropagation: ");
  Serial.println (" ");
  Vai_al_Terminale();
/*****************************************************************
* Inizio del training
*****************************************************************/
  for( Cicli_Training = 1 ; Cicli_Training < 500 ; Cicli_Training++) {
/*****************************************************************
* Inizializzazione casuale dei pesi sinaptici
*****************************************************************/
```

```
   for( p = 0 ; p < Numero_Ingressi ; p++) {
    q = random(Numero_Ingressi);
    r = Indice_Randomizzato[p] ;
    Indice_Randomizzato[p] = Indice_Randomizzato[q] ;
    Indice_Randomizzato[q] = r ;
   }
   Error = 0.0 ;
/*****************************************************************
* Inizializzaziaone degli indici randomici
*****************************************************************/
   for( q = 0 ; q < Numero_Ingressi ; q++ ) {
    p = Indice_Randomizzato[q];
/*****************************************************************
* Primo passaggio attraverso il layer interno (Feed Forward)
*****************************************************************/
    for( i = 0 ; i < Nodi_Interni ; i++ ) {
     Accum = Pesi_Livello_Interno[Nodi_Ingresso_Training][i] ;
     for( j = 0 ; j < Nodi_Ingresso_Training ; j++ ) {
      Accum += Ingresso_Training[p][j] * Pesi_Livello_Interno[j][i] ;
     }
     Hidden[i] = 1.0/(1.0 + exp(-Accum)) ;
    }
/*****************************************************************
* Secondo passaggio attraverso il layer esterno e calcolo dell'errore
*****************************************************************/
    for( i = 0 ; i < Nodi_Uscita_Training ; i++ ) {
     Accum = Pesi_Uscita_Training[Nodi_Interni][i] ;
     for( j = 0 ; j < Nodi_Interni ; j++ ) {
```

```
        Accum += Hidden[j] * Pesi_Uscita_Training[j][i] ;
      }
      Uscita_Training[i] = 1.0/(1.0 + exp(-Accum)) ;
      Variazione_Uscita_Training[i] = (Obiettivo[p][i] - Uscita_Training[i]) * Uscita_Training[i] * (1.0 -
Uscita_Training[i]) ;
      Error += 0.5 * (Obiettivo[p][i] - Uscita_Training[i]) * (Obiettivo[p][i] - Uscita_Training[i]) ;
    }
/*************************************************************************************
* Correzzione dell'errore attraverso l'algoritmo di Backpropagation nel layer interno
*************************************************************************************/
    for( i = 0 ; i < Nodi_Interni ; i++ ) {
      Accum = 0.0 ;
      for( j = 0 ; j < Nodi_Uscita_Training ; j++ ) {
        Accum += Pesi_Uscita_Training[i][j] * Variazione_Uscita_Training[j] ;
      }
      Variazione_Interna[i] = Accum * Hidden[i] * (1.0 - Hidden[i]) ;
    }
/*****************************************************************
* Cambio del valore dei pesi sinaptici nel layer interno
*****************************************************************/
    for( i = 0 ; i < Nodi_Interni ; i++ ) {
      Cambio_Pesi_Livello_Interno[Nodi_Ingresso_Training][i] = Learning_Rate * Variazione_Interna[i] +
Momentum * Cambio_Pesi_Livello_Interno[Nodi_Ingresso_Training][i] ;
      Pesi_Livello_Interno[Nodi_Ingresso_Training][i] +=
Cambio_Pesi_Livello_Interno[Nodi_Ingresso_Training][i] ;
      for( j = 0 ; j < Nodi_Ingresso_Training ; j++ ) {
        Cambio_Pesi_Livello_Interno[j][i] = Learning_Rate * Ingresso_Training[p][j] * Variazione_Interna[i] +
Momentum * Cambio_Pesi_Livello_Interno[j][i];
        Pesi_Livello_Interno[j][i] += Cambio_Pesi_Livello_Interno[j][i] ;
```

```
      }
     }

/*****************************************************************
* Cambio del valore dei pesi sinaptici nel layer esterno
*****************************************************************/
     for( i = 0 ; i < Nodi_Uscita_Training ; i ++ ) {
      Cambio_Pesi_Uscita_Training[Nodi_Interni][i] = Learning_Rate * Variazione_Uscita_Training[i] +
Momentum * Cambio_Pesi_Uscita_Training[Nodi_Interni][i] ;
      Pesi_Uscita_Training[Nodi_Interni][i] += Cambio_Pesi_Uscita_Training[Nodi_Interni][i] ;
      for( j = 0 ; j < Nodi_Interni ; j++ ) {
       Cambio_Pesi_Uscita_Training[j][i] = Learning_Rate * Hidden[j] * Variazione_Uscita_Training[i] +
Momentum * Cambio_Pesi_Uscita_Training[j][i] ;
        Pesi_Uscita_Training[j][i] += Cambio_Pesi_Uscita_Training[j][i] ;
      }
     }
   }
/*****************************************************************
*Ogni 1000 cicli stampa un Report sul PC
*****************************************************************/
   ReportEvery1000 = ReportEvery1000 - 1;
   if (ReportEvery1000 == 0)
   {
    Serial.println();
    Serial.println();
    Serial.print ("Dopo il primo ciclo di correzione l'errore pari a ");
    Serial.println (Error, 5);
    Serial.println();
```

```
    Serial.print ("Ciclo numero: ");
    Serial.print (Cicli_Training);

    Vai_al_Terminale();
    if (Cicli_Training==1)
    {
      ReportEvery1000 = 999;
    }
    else
    {
      ReportEvery1000 = 1000;
    }
   }
/****************************************************************
* Se l'errore è minore dell'errore minimo termina il training
****************************************************************/
   if( Error < Valore_Min_Errore ) break ;
  }
  Serial.println (" ");
  Serial.println();
  Serial.print ("Ciclo numero: ");
  Serial.print (Cicli_Training);
  Serial.print ("  Errore pari a = ");
  Serial.println (Error, 5);
  Vai_al_Terminale();
   for( p = 0 ; p < Numero_Ingressi ; p++ ) {
   Serial.println();
   Serial.println (" ");
```

```
  Serial.print ("  Pesi finali ingresso numero: ");
  Serial.println (p);
  Serial.print (" ");
  for( i = 0 ; i < Nodi_Interni ; i++ ) {
    Serial.print (Pesi_Livello_Interno[p][i], DEC);
    Serial.print (" ");
  }
}
  Serial.println (" ");
Serial.println (" ");
 for( p = 0 ; p < Nodi_Interni ; p++ ) {
  Serial.println();
  Serial.println (" ");
  Serial.print ("  Pesi finali uscita numero: ");
  Serial.println (p);
  Serial.print (" ");
  for( i = 0 ; i < Nodi_Uscita_Training ; i++ ) {
    Serial.print (Pesi_Uscita_Training[p][i], DEC);
    Serial.print (" ");
  }
}
Serial.println ();
Serial.println ();
Serial.println ("Training Set Risolto! ");
Serial.println ("--------");
Serial.println ();
Serial.println ();
ReportEvery1000 = 1;
```

```
  Inizio_Utilizzo_Rete_Neurale();
}


/*****************************************************************
* Parte di programma per la stampa su PC
*****************************************************************/
void Vai_al_Terminale()
{
  for( p = 0 ; p < Numero_Ingressi ; p++ ) {
    Serial.println();
    Serial.print ("  Ingresso numero: ");
    Serial.println (p);
    Serial.print ("  Ingresso Training ");
    for( i = 0 ; i < Nodi_Ingresso_Training ; i++ ) {
      Serial.print (Ingresso_Training[p][i], DEC);
      Serial.print (" ");
    }
    Serial.print ("  Obiettivo ");
    for( i = 0 ; i < Nodi_Uscita_Training ; i++ ) {
      Serial.print (Obiettivo[p][i], DEC);
      Serial.print (" ");
    }
/*****************************************************************
* Calcolo uscita del layer interno con i pesi sinaptici trovati
*****************************************************************/
    for( i = 0 ; i < Nodi_Interni ; i++ ) {
      Accum = Pesi_Livello_Interno[Nodi_Ingresso_Training][i] ;
```

```
    for( j = 0 ; j < Nodi_Ingresso_Training ; j++ ) {
      Accum += Ingresso_Training[p][j] * Pesi_Livello_Interno[j][i] ;
    }
    Hidden[i] = 1.0/(1.0 + exp(-Accum)) ;
   }
/******************************************************************************
* Calcolo dell'uscita del layer esterno con i pasi sinaptici trovati e calcolo dell'errore
******************************************************************************/
   for( i = 0 ; i < Nodi_Uscita_Training ; i++ ) {
    Accum = Pesi_Uscita_Training[Nodi_Interni][i] ;
    for( j = 0 ; j < Nodi_Interni ; j++ ) {
      Accum += Hidden[j] * Pesi_Uscita_Training[j][i] ;
    }
    Uscita_Training[i] = 1.0/(1.0 + exp(-Accum)) ;
   }
   Serial.print ("  Uscita Training ");
   for( i = 0 ; i < Nodi_Uscita_Training ; i++ ) {
    Serial.print (Uscita_Training[i], 5);
    Serial.print (" ");
   }
 }
}
/******************************************************************************
* Inizio dell'utilizzo della rete neurale addestrata con l'ingresso che gli vogliamo far valutare
******************************************************************************/
void Inizio_Utilizzo_Rete_Neurale()
{
   for( int np = 0 ; np < 1 ; np++ ) {   //numero di pattern
```

```
  Serial.println();
  Serial.println ("  Dopo l'addestramento il risultato pari a: ");
  Serial.print ("  Ingresso Impostato ");
  for( i = 0 ; i < Nodi_Ingresso_Training ; i++ ) {
   Serial.print (Ingresso[np][i], DEC);
   Serial.print (" ");
  }
/******************************************************************
* Calcolo uscita del layer interno con i pesi sinaptici trovati
******************************************************************/
  for( i = 0 ; i < Nodi_Interni ; i++ ) {
   Accum = Pesi_Livello_Interno[Nodi_Ingresso_Training][i] ;
   for( j = 0 ; j < Nodi_Ingresso_Training ; j++ ) {
    Accum += Ingresso[np][j] * Pesi_Livello_Interno[j][i] ;
   }
   Hidden[i] = 1.0/(1.0 + exp(-Accum)) ;
  }
/******************************************************************
* Calcolo dell'uscita del layer esterno con i pasi sinaptici trovati
******************************************************************/
  for( i = 0 ; i < Nodi_Uscita_Training ; i++ ) {
   Accum = Pesi_Uscita_Training[Nodi_Interni][i] ;
   for( j = 0 ; j < Nodi_Interni ; j++ ) {
    Accum += Hidden[j] * Pesi_Uscita_Training[j][i] ;
   }
   Uscita_Training[i] = 1.0/(1.0 + exp(-Accum)) ;
  }
  Serial.print ("  Risultato Stimato ");
```

```
   for( i = 0 ; i < Nodi_Uscita_Training ; i++ ) {
     Serial.print (Uscita_Training[i], 5);
     Serial.print (" ");
     float c = Uscita_Training[i];
     if (c < 0.5)
     {
       digitalWrite(Stop, HIGH); // turn the LED on
     }
     else
        digitalWrite(Avanti, HIGH);
   }
  // }
  }
delay(9999999);
}
void zero(){
  digitalWrite(G, LOW);
  digitalWrite(F, HIGH);
  digitalWrite(A, HIGH);
  digitalWrite(B, HIGH);
  digitalWrite(E, HIGH);
  digitalWrite(D, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(DP, LOW);
}
void uno(){
  digitalWrite(G, LOW);
  digitalWrite(F, LOW);
```

```
  digitalWrite(A, LOW);
  digitalWrite(B, HIGH);
  digitalWrite(E, LOW);
  digitalWrite(D, LOW);
  digitalWrite(C, HIGH);
  digitalWrite(DP, LOW);
}
void due(){
  digitalWrite(G, HIGH);
  digitalWrite(F, LOW);
  digitalWrite(A, HIGH);
  digitalWrite(B, HIGH);
  digitalWrite(E, HIGH);
  digitalWrite(D, HIGH);
  digitalWrite(C, LOW);
  digitalWrite(DP, LOW);
}
void tre(){
  digitalWrite(G, HIGH);
  digitalWrite(F, LOW);
  digitalWrite(A, HIGH);
  digitalWrite(B, HIGH);
  digitalWrite(E, LOW);
  digitalWrite(D, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(DP, LOW);
}
void quattro(){
```

```
  digitalWrite(G, HIGH);
  digitalWrite(F, HIGH);
  digitalWrite(A, LOW);
  digitalWrite(B, HIGH);
  digitalWrite(E, LOW);
  digitalWrite(D, LOW);
  digitalWrite(C, HIGH);
  digitalWrite(DP, LOW);
}
void cinque(){
  digitalWrite(G, HIGH);
  digitalWrite(F, HIGH);
  digitalWrite(A, HIGH);
  digitalWrite(B, LOW);
  digitalWrite(E, LOW);
  digitalWrite(D, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(DP, LOW);
}
void six(){
  digitalWrite(G, HIGH);
  digitalWrite(F, HIGH);
  digitalWrite(A, HIGH);
  digitalWrite(B, LOW);
  digitalWrite(E, HIGH);
  digitalWrite(D, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(DP, LOW);
```

```
}
void sette(){
  digitalWrite(G, LOW);
  digitalWrite(F, LOW);
  digitalWrite(A, HIGH);
  digitalWrite(B, HIGH);
  digitalWrite(E, LOW);
  digitalWrite(D, LOW);
  digitalWrite(C, HIGH);
  digitalWrite(DP, LOW);
}
void otto(){
  digitalWrite(G, HIGH);
  digitalWrite(F, HIGH);
  digitalWrite(A, HIGH);
  digitalWrite(B, HIGH);
  digitalWrite(E, HIGH);
  digitalWrite(D, HIGH);
  digitalWrite(C, HIGH);
  digitalWrite(DP, LOW);
}
void nove(){
  digitalWrite(G, HIGH);
  digitalWrite(F, HIGH);
  digitalWrite(A,HIGH);
  digitalWrite(B, HIGH);
  digitalWrite(E, LOW);
  digitalWrite(D, HIGH);
```

```
  digitalWrite(C, HIGH);
  digitalWrite(DP, LOW);
}
```