Reti neurali e implementazione

Davide Gessa

21 Giugno 2010

Indice

Elenco delle figure

Capitolo 1

Introduzione

Questa tesi nasce da un vecchio software da me iniziato verso l'inizio del 2009 (e mai terminato) che riassumeva il frutto di qualche mia curiosita' riguardo l'argomento delle reti neurali artificiali; qualche mese fa ho ritrovato i sorgenti e mi sono interessato nuovamente all'argomento, riscrivendo da zero una nuova liberia (che analizzero' in seguito) che implementa alcuni tipi di reti neurali artificiali e alcuni programmi che utilizzano questa libreria e presentano alcune possibilita' di utilizzo. Avevo gia' iniziato una breve trattazione da esporre su un altro mio progetto, ma ho deciso di iniziare da capo per dedicarmi ad un argomento che raccoglie in se' piu' materie, come statistica, matematica, informatica e per certe caratteristiche tecniche anche sistemi. Per vedere lo sviluppo dei miei progetti www.inventati.org/dak.

1.1 Licenza

Il proggetto e' rilasciato interamente sotto licenza GPLv2, presente integralmente nel file license.txt; e' riportato qui di seguito l'header presente in ogni file sorgente del progetto:

1.2 Strumenti software

Ecco una lista dei software principali utilizzati per realizzare il progetto e la tesi che state leggendo; da sottolineare che tutti sono free software e opensource, e che lo sviluppo e' avvenuto col sistema operativo linux gentoo ed in parte con freebsd, anch'essi free e opensource. Ecco una lista dei programmi utilizzati:

gcc compilatore per il linguaggio C

cmake sistema di compilazione

vim editor di testo

subversion controllo di revisione

gtk+ librerie per interfacce grafiche

gnuplot programma di plotting matematico

latex compilatore per il linguaggio LaTeX

1.3 LaTeX

Per scrivere la documentazione del sistema e' stato utilizzato il linguaggio di markup LaTeX, che permette di preparare dei testi basati su TeX, un linguaggio di composizione tipografica. Utilizzare LaTeX, permette di risparmiare un tempo notevole per quanto riguarda la formattazione delle pagine, la creazione degli indici, la visualizzazione di formule matematiche e molto altro; per questo motivo e' utilizzato da gran parte di accademici, scienziati, matematici e ingegneri. LaTeXe' distribuito come software libero ed e' disponibile su molte piattaforme.

Capitolo 2

Reti neurali

Questo capitolo mira ad esporre il funzionamento di una rete neurale artificiale e di alcuni algoritmi per l'apprendimento; e' comunque necessario fare una breve introduzione riguardo il funzionamento delle reti neurali in natura.

2.1 Reti neurali biologiche

Il funzionamento delle reti neurali artificiali, deriva da degli studi effettuati sulle reti neurali biologiche, presenti, seppur con diverse caratteristiche, nel cervello di ogni animale; i primi successi significativi riguardo lo studio del funzionamento delle reti neurali sono relativamente recenti, e alcuni aspetti sono ancora ignoti. Le reti neurali sono delle strutture costituite da neuroni; i neuroni sono classificabili secondo diverse caratteristiche, una di queste e' la loro funzione:

- Neuroni sensitivi o neuroni di input: si occupano di ricevere impulsi e stimoli dagli organi sensoriali.
- Neuroni motori o neuroni di output: generano impulsi di tipo motorio che vengono trasmessi agli organi periferici.
- Interneuroni o neuroni nascosti: elaborano le informazioni fornite dai neuroni sensitivi per trasmetterle ai neuroni motori.

Ogni neurone (che sia sensitivo, motorio o un interneurone) e' formato principalmente da tre parti:

- La soma: che comprende il corpo cellulare compreso il nucleo e altri apparati dedicati alla sopravvivenza della cellula.
- Gli assoni: conducono il segnale generato dal soma verso altre cellule neurali.
- I dentriti: son costituiti da diramazioni ad albero che trasportano segnali di altri neuroni, verso la soma; i dentriti hanno la caratteristica di non essere buoni conduttori di segnali nervosi, i segnali ricevuti tendono quindi a diminuire di intensita'.

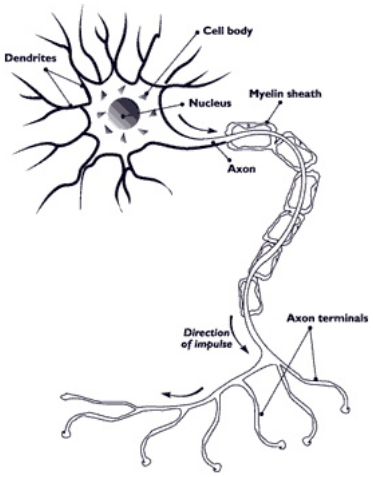

Figura 2.1: Struttura di un neurone

Gli assoni e i dentriti, comunicano con altre cellule neurali attraverso dei punti di connessione detti sinapsi. La trasmissione di un informazione da un neurone all'altro avviene attraverso l'emissione di sostanze chimiche dette neuro-trasmettitori stimolata da segnali chimici o da segnali elettrici generati dal neurone stesso.

Il neurone riceve molti segnali di input tramite i dentriti, e rilascia un solo segnale di output tramite l'assone. Il segnale di output e quindi il rilascio dei neuro-trasmettitori, avviene soltanto se la somma dei segnali di input supera una certa soglia; questa soglia puo' essere ridotta dall'assunzione di sostanze che provocano eccitazione, (come la caffeina) e puo' essere aumentata dall'assunzione di tranquillanti. Le connessioni tra i neuroni non sono immobili, durante la vita le connessioni si creano, si trasformano e si distruggono continuamente.

2.2 Reti neurali artificiali

Le reti neurali artificiali sono in sostanza, dei modelli matematici composti da neuroni artificiali (anche chiamati nodi; sono delle strutture matematiche con il compito di emulare in parte il funzionamento di un neurone biologico), e dalle loro interconnessioni che simulano il funzionamento di una rete neurale biologica; le reti neurali artificiali permettono di risolvere problemi (fase di esecuzione) piu' o meno complessi, avendo a disposizione dati non necessariamente esatti, e non utillizzando algoritmi specifici per il problema; cio' sara' possibile a seguito di un periodo nel quale la rete verra' istruita a svolgere i compiti desiderati (fase di apprendimento).

2.2.1 Neurone artificiale

Il neurone artificiale e' rappersentato come una struttura matematica, in cui sono memorizzate determinate informazioni:

- Stato x_i: lo stato in cui si trova il neurone; ad esempio puo' essere 0 o 1 se si tratta di un neurone binario o da 0 a 1 se si tratta di un neurone a valori continui.
- Pesi sinaptici ω_i^k: sono i valori delle interconnessioni con i neuroni dello strato precedente.
- Errore δ_i: errore atteso.

Per generare l'input netto del neurone bisogna fare la somma del prodotto dei pesi di ogni connessione dei neuroni dello strato precedente con i loro valori di output.

2.2.2 Tipi di rete

Multi Layered Perceptons

E' il tipo di rete piu' utilizzato; prevede la presenza di neuroni divisi in vari strati, generalmente uno di input, uno di output, ed uno o piu' strati intermedi. Ogni neurone utilizza l'output di tutti i neuroni dello strato precedente per generare l'output, che sara' utilizzato come input dallo strato successivo.

Self Organization Map

Le reti dette a mappa autoorganizzante prevedono una matrice di neuroni; per l'apprendimento e' utilizzato un innovativo algoritmo di autoapprendimento, che aggiorna continuamente i pesi sinaptici utilizzando dei vettori dati in ingresso, in genere molto grandi. Le uscite sono di solito 2 o 3, ed i valori d'uscita nella fase di esecuzione individuano un neurone della rete, detto vincitore. Le reti som insomma, permettono di associare ad un certo vettore di input, un neurone posizionato nella griglia della rete.

Hopfield

Le reti di Hopfield si basano sul concetto delle proprieta' collettive di un modello costituito da unita' elementari di elaborazione; questo tipo di rete supera il limite imposto dalle reti mlp che prevedono una sola direzione per il processo di esecuzione e apprendimento, utilizzando invece delle reti in cui tutti i neuroni comunicano con tutti gli altri. Le reti di Hopfield sono utilizzate specialmente per le memorie associative e per sistemi di ottimizzazione.

2.2.3 Apprendimento

Per poter utilizzare la rete per scopi pratici, bisogna innanzitutto addestrarla opportunamente per trovare (piu' precisamente imparare) la relazione tra dati di input e dati di output. L'addestramento puo' essere eseguito con svariati algoritmi, ma fondamentalmente si possono distinguere tre grandi tipologie di apprendimento:

- Supervisionato: consiste nel proporre alla rete dei set di dati di input e i corrispettivi dati di output (risultato degli input); tramite un algoritmo, la rete impara il legame che c'e' tra essi. Uno degli algoritmi chiave

dell'apprendimento supervisionato, e' l'algoritmo di propagazione inversa (meglio noto come backpropagation), che mediante un pattern di input ed uno di output, calcola l'errore per ogni strato della rete ricalcolando i pesi sinaptici dei singoli neuroni. L'algoritmo di backpropagation verra' analizzato nei dettagli nell'articolo dedicato alle reti MLP.

- Non supervisionato: non prevede nessun intervento esterno, bensi' utilizza metodi probabilistici per individuare dei possibili input e dei corrispondenti risultati di output; un esempio famoso e' l'algoritmo di autoapprendimento utilizzato nelle reti som.

- Per rinforzo: un algoritmo si occupa di generare azioni e di interpetare il risultato della retroazione dell'ambiente stesso, valutando se tale retroazione e' positiva o negativa per raggiungere lo scopo prefissato dalla rete.

Capitolo 3

Reti MLP e apprendimento supervisionato

La rete MLP e' il modello di rete neurale artificiale piu' utilizzato; prevede la presenza di neuroni artificiali divisi in vari strati, generalmente uno di input, uno di output, ed uno o piu' strati intermedi. Ogni neurone utilizza l'output di tutti i neuroni dello strato precedente per generare l'output, che sara' utilizzato come input dallo strato successivo. In questo capitolo analizzero' in breve gli algoritmi di esecuzione della rete, e l'algoritmo base di backpropagation per l'apprendimento supervisionato.

3.1 Elenco dei simboli

Ecco una tabella riassuntiva dei simboli che utilizzero' nella schematizzazione degli algoritmi:

δ_i	Errore del neurone i
η	Costante di apprendimento
ω_i^k	Peso sinaptico del neurone i rispetto al neurone k precedente
x_i	Valore del neurone di input i
y_i	Valore del neurone hidden i
z_i	Valore del neurone di output i

Tabella 3.1: Elenco dei simboli per le reti MLP

3.2 Esecuzione

La fase di esecuzione consiste nel fornire alla rete dei dati di input, elaborarli ed ottenere dei dati di output; l'esecuzione della rete consiste in 3 step:

- Inserimento i dati di input nei neuroni di input.
- Per ogni neurone di ogni strato nascosto e dello strato di output, calcolo dello stato del neurone tramite la formula:

$$y_i = f(\sum(\omega_i^k * x_k))$$

- Prelevamento degli output dai neuroni di output.

3.3 Back Propagation

Il back propagation (propagazione inversa) e' l'algoritmo piu' diffuso per l'addestramento delle reti neurali di tipo MLP; l'algoritmo puo' essere riassunto in 2 step:

- Si inseriscono i valori di input del pattern di addestramento nei neuroni dello strato di input. Si esegue la rete, e si calcola l'errore delta di output: per ogni neurone dello strato di output si calcola la differenza tra il valore desiderato (d_i) e quello ottenuto dall'esecuzione, poi il valore ottenuto viene moltiplicato per la derivata della funzione di trasferimento avente come argomento il valore di output ottenuto dalla rete nello stato in cui e' il neurone attualmente:

  $$\delta_i = (d_i - z_i) * f'(z_i)$$

  Successivamente si calcola l'errore delta di ognuno degli strati nascosti della rete sino ad arrivare al primo neurone nascosto:

  $$\delta_i = (\sum(\delta_k * \omega_k^i)) * f'(y_i))$$

- I valori cosi' ottenuti permetteranno di modificare i pesi delle connessioni di ogni strato per ogni neurone, per ottenere un valore di output piu' verosimile al valore desiderato Per ogni neurone di output e di hidden quindi:

  $$\omega_i^k = (\sum(\eta * y_k * \delta_i))$$

Il procedimento termina qui; si passa ad un altro pattern di addestramento e si riprende dal punto uno.

3.4 Pattern Recognition mediante rete neurali

Il pattern recognition (riconoscimento di pattern, in italiano), e' una delle possibili applicazioni delle reti neurali; il pattern recognition in se', e' un processo di riconoscimento e classificazione di pattern, partendo da un insieme di dati grezzi in input. Per l'addestramento vengono generati vari pattern per ogni carattere (se i pattern sono caratteri); questi pattern vengono passati alla rete neurale come dati di input, e come output viene specificato di che carattere si tratta; successivamente si esegue l'algoritmo di backpropagation per un cospicuo numero di epoche; la rete cosi' addestrata sara' in grado di riconoscere anche pattern diversi da quelli utilizzati per l'addestramento.

Capitolo 4

Neural Network Abstraction Layer

La mia liberia prende il nome di nnal, ovvero neural network abstraction layer (layer di astrazione di reti neurali); il motivo di questo nome deriva dal fatto che lo scopo e' quello di realizzare un framework che offre strumenti semplici per simulare reti neurali, risparmiando ai programmatori l'arduo compito di crearsi una libreria. Nnal mira a studiare nei dettagli problematiche relative al calcolo e alla memoria utilizzata, ottimizzando al meglio il codice e prevedendo l'utilizzo di cuda e opencl per il calcolo tramite gpu; attualmente i tipi di rete implementati sono due:

- Reti MLP, implementate completamente
- Reti SOM, ancora in fase di sviluppo

4.1 Simulare una rete MLP

La libreria nnal offre semplici funzioni per simulare le reti MLP e per eseguire l'apprendimento tramite backpropagation; supponiamo di voler creare una rete per svolgere la funzione logica or; per prima cosa decidiamo il numero di neuroni per ogni layer, in questo esempio 2 neuroni per l'input 3 per l'hidden e 1 per l'output:

```
int layers[] = { 2, 3 1 };
```

Si crea poi una rete neurale con 3 layer passando l'array precedentemente creato:

```
mlp_neural_network_t *n = mlp_new(layers, 3);
```

Decidiamo poi che funzioni matematiche vogliamo utilizzare per il trasferimento dei dati da un neurone all'altro (nella libreria sono gia' implementate alcune funzioni di trasferimento); ad esempio utilizziamo la sigmoide logistica in modo che ci sia possibile verificare l'approssimazione dei risultati:

```
n->transf_func = &sigmodial;
n->transf_func_derivate = &sigmodial_derivate;
```

Ora facciamo un semplice esempio di addestramento; l'unico algoritmo di addestramento implementato per le reti mlp e' l'algoritmo di backpropagation. Utilizziamo un ciclo for per far elaborare alla rete i pattern di addestramento da noi generati, per un certo numero di epoche; il numero delle epoche ed il numero di pattern devono essere adeguati in modo da garantire un errore delta trascurabile. I pattern di addestramento sono fondamentalmente due array, il primo, contente i dati di input ed il secondo contenente i dati di output:

```
for(i = 0; i < N_EPOCHE; i++)
{
in[0] = 1.0;
in[1] = 1.0;
out[0] = 0.0;
mlp_back_propagate(n, in, out);

in[0] = 1.0;
in[1] = 0.0;
out[0] = 1.0;
mlp_back_propagate(n, in, out);

in[0] = 0.0;
in[1] = 1.0;
out[0] = 1.0;
mlp_back_propagate(n, in, out);

in[0] = 0.0;
in[1] = 0.0;
out[0] = 0.0;
mlp_back_propagate(n, in, out);

printf("epoch: %d\n",i);
}
```

Sono inoltre presenti delle funzioni per salvare la rete in un file e per caricare una rete da file:

```
mlp_save(n, "saved/or.nn");

n = mlp_load("saved/or.nn");
```

Successivamente all'addestramento, e' possibile utilizzare la rete; facciamo un test passando la combinazione 0, 1 in input; il risultato verra' messo dalla funzione nell'array out e dovrebbe essere 1:

```
in[0] = 0.0;
in[1] = 1.0;
mlp_execute(n, in, out);
```

4.2 Visualizzazione reti con GtkWidget

Libnnal implementa anche un widget per gtk+ che permette di visualizzare una rete neurale graficamente, quindi i neuroni di ogni layer, le relative connessioni tra di essi ed i pesi sinaptici delle connessioni. I diversi valori delle connessioni e dei neuroni sono rappresentati tramite l'utilizzo di colori diversi.

L'integrazione in un applicazione gtk e' molto semplice; per la creazione e l'utilizzo della rete si fa riferimento al codice visto in precedenza, mentre le funzioni per utilizzare il widget sono principalmente due:

- La prima crea il widget:

  ```
  GtkWidget *mlp = gtk_mlp_new();
  ```

- La seconda assegna la rete neurale al widget (viene utilizzato un puntatore, quindi si puo' operare sulla rete net, ottenendo le modifiche anche sul widget:

  ```
  gtk_mlp_set_network(GTK_MLP(mlp), net);
  ```

4.3 Applicazioni software

Ho realizzato svariati programmi programmi in C, alcuni con un interfaccia grafica in gtk+, per mostrare varie applicazioni della libreria di astrazione:

- bptimetest: test sui tempi di apprendimento
- epochtest: test sul variare del numero di epoche di apprendimento
- gtkmlp: utilizzo di reti mlp con un interfaccia gtk
- gtkmlptest: test del widget mlp per gtk
- mlptest: vari esempi di utilizzo delle reti mlp
- patterngtk: riconoscimento di pattern con interfaccia gtk
- patterntest: test di pattern recognition

Di questa lista pero', ne analizzero' in specifico solo alcuni.

4.3.1 Neural network

MlpGtk permette di creare una rete neurale generica, decidendo il numero di neuroni per ogni layer, il coefficiente di apprendimento ed il tipo di funzione di trasferimento; dopo la creazione della rete e' possibile:

- Addestrare la rete tramite dati prelevati da file
- Osservare il cambiamento dei pesi sinaptici graficamente durante e dopo l'addestramento
- Impostare dei valori ai neuroni di input per farli elaborare alla rete
- Salvare e caricare la rete

Ecco uno screenshoot della schermata principale:

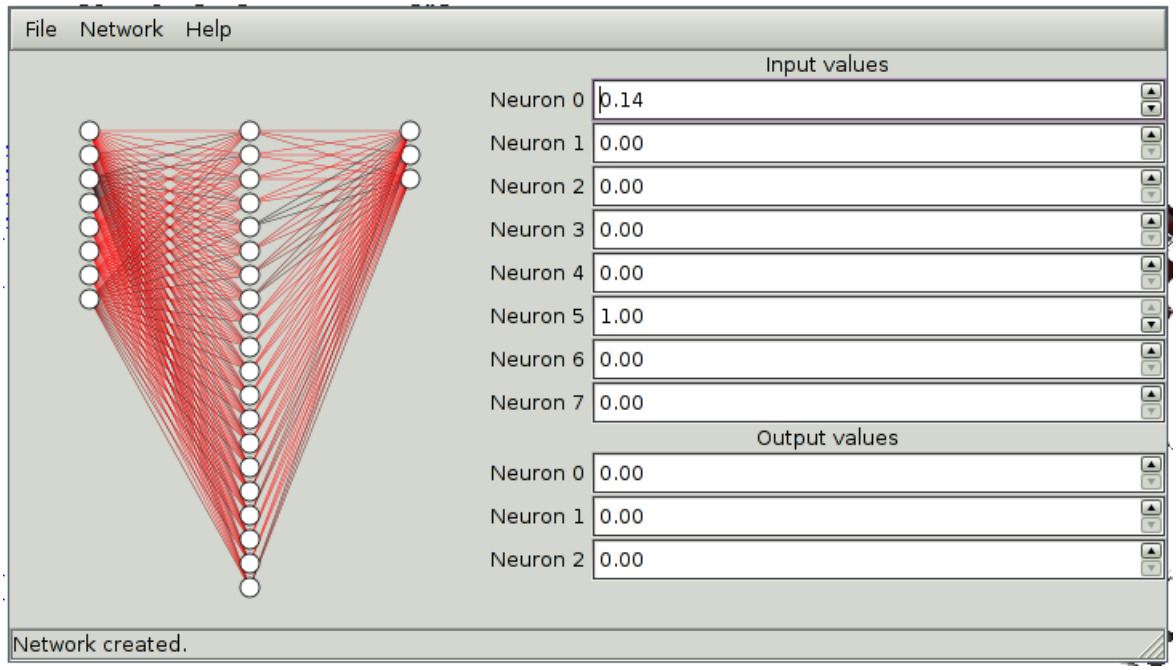

Figura 4.1: Neural Network in GTK+

4.3.2 Pattern Recognition

PatternGtk invece, e' un programma per il riconoscimento di pattern, fondamentalmente di lettere e numeri; con qualche piccola modifica e' possibile utilizzarlo anche per altri tipi di dati. Il programma permette di indicare una cartella contenente file png (per ora solo 64x64 in scala di grigi), selezionare il numero di epoche, e avviare il processo di apprendimento; e' possibile poi utilizzare un'altra immagine, con le stesse caratteristiche dei pattern di addestramento, per testare la rete ed ottenere in output il codicce ascii della lettera corrispondente all'immagine.

Per fare test diversi, ho creato vari set di pattern, in figura possiamo vedere un test con le lettere cirilliche (translitterate in lettere latine):

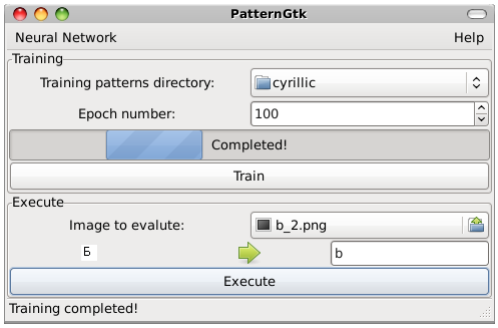

Figura 4.2: Pattern Recognition in GTK+ - Alfabeto Cirillico

E qui un esempio con l'alfabeto italiano:

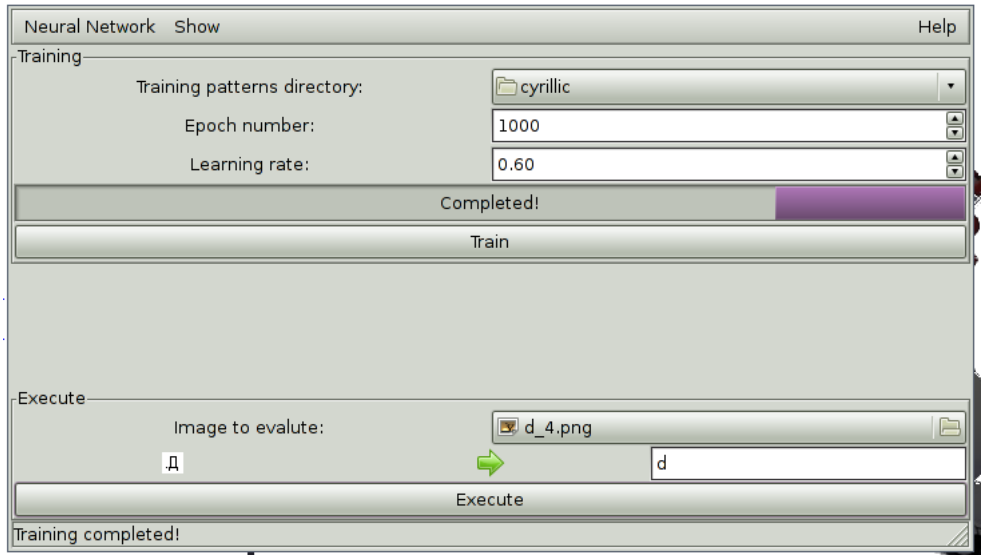

Figura 4.3: Pattern Recognition in GTK+ - Alfabeto italiano

Ho realizzato inoltre dei test sull'andamento dell'errore delta calcolato su un set di pattern di 22 caratteri per differenti numeri di epoche. Come si puo' notare, si raggiunge un valore di errore di circa 0.3 dopo circa 1000 epoche; aumentando il numero di epoche si ottiene un andamento asintotico sopra questo valore:

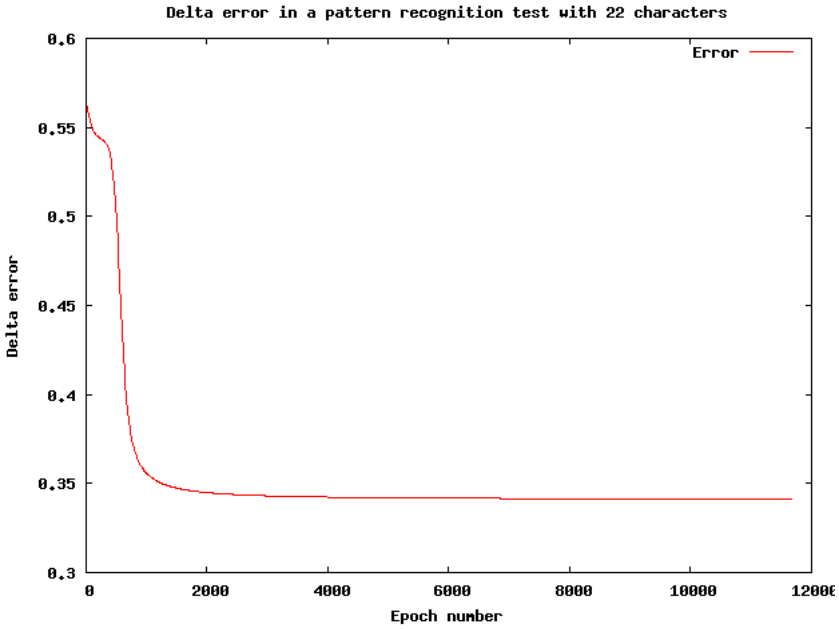

Figura 4.4: Pattern Recognition - Epoch test

4.3.3 Mlp Widget

In figura possiamo vedere l'utilizzo del widget gtk nel programma per il pattern recognition:

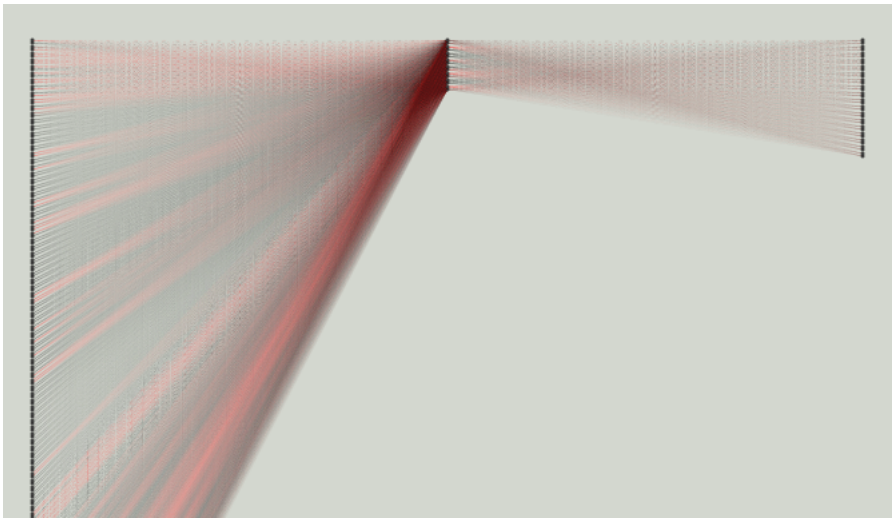

Figura 4.5: Pattern Recognition in GTK+ - Visualizzazione grafica

Ed un esempio di rete piu' piccola creata con MlpGtk:

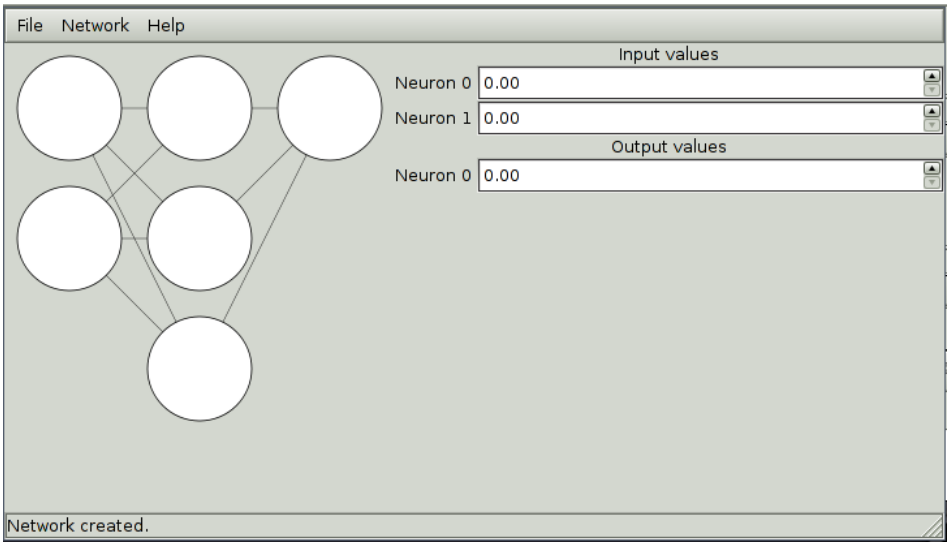

Figura 4.6: Reti MLP in GTK+ - Visualizzazione grafica

Capitolo 5

Bibliografia

Bibliografia

[1] Silvio Cammarata - Reti neurali, dal Percepton alle reti caotiche e neuro-fuzzy

[2] Christopher M. Bishop - Pattern Recognition and Machine Learning