



UNIVERSITÁ DEGLI STUDI DI NAPOLI
FEDERICO II

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

**Progetto Machine Learning e
Applicazioni mod.B**

Gruppo 1

Gianluca Panzuto N97/217
Luigi Iacuaniello N97/234

Anno accademico 2015-16

Indice

1	Introduzione e traccia progetto	2
1.1	Traccia Progetto parte A	2
1.2	Traccia Progetto parte B	2
1.3	Introduzione	2
2	Addestramento Rete Neurale	3
2.1	Back Propagation	3
2.2	Resilient Back-Propagation	6
3	Sviluppo Progetto	7
3.1	Main	7
3.2	Altre funzioni	7
3.2.1	new_net	7
3.2.2	feedForward	8
3.2.3	backPropagation	8
3.2.4	simulationNetwork	8
3.2.5	RProp	8
3.2.6	loadAndShowBananaDataSet	9
3.2.7	plotDataDS	9
3.2.8	derivativeFunction	9
4	Test	10
4.1	Tabelle comparative degli errori	10
4.1.1	Test per 1 strato interno	10
4.1.2	Test per 5 strati interni	11
4.1.3	Test per 10 strati interni	11
4.1.4	Test per 20 strati interni	11
4.2	Grafici dei test effettuati	12
4.2.1	Grafici per 1 strato interno	12
4.2.2	Grafici per 5 strati interni	20
4.2.3	Grafici per 10 strati interni	28
4.2.4	Grafici per 20 strati interni	36
4.3	Risultati ottenuti ed osservazioni	44
4.3.1	1 strato interno	44
4.3.2	5 strati interni	44
4.3.3	10,20 strati interni	44
4.3.4	Conclusioni e una possibile soluzione	45
5	Codice Sorgente e Riferimenti	46

1 Introduzione e traccia progetto

1.1 Traccia Progetto parte A

- Progettazione ed implementazione di funzioni per la propagazione in avanti di una rete neurale multi-strato con qualsiasi funzione di output per ciascun strato.
- Progettazione ed implementazione di funzioni per la realizzazione della back-propagation per reti neurali multistrato con qualsiasi funzione di output per ciascun strato e con qualsiasi funzione di errore derivabile rispetto all'output.

1.2 Traccia Progetto parte B

Dato il banana dataset, si fissi la resilient backpropagation (Rprop) come algoritmo di aggiornamento dei pesi. Si studi l'apprendimento della rete neurale al variare dei nodi interni. Scegliere e mantenere invariati tutti gli altri parametri come dataset, funzioni di output e funzioni di errore.

1.3 Introduzione

Nello sviluppo del seguente progetto, sono state implementate tutte le funzionalità richieste, comprese quelle *facoltative*.

Per la **parte B** del progetto, il dataset utilizzato nei test rappresenta un problema di classificazione a due classi. Il banana dataset contiene 5300 punti del piano reale, quindi la dimensionalità delle caratteristiche è 2.

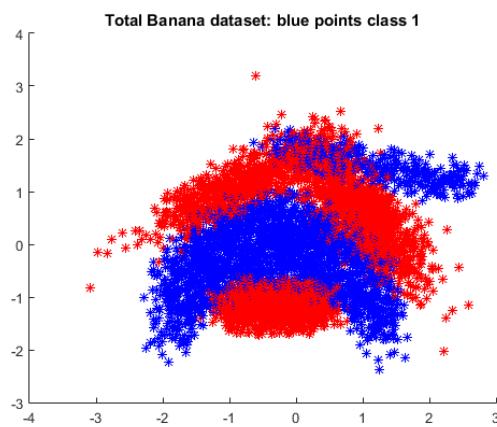


Figure 1: *Banana Dataset*

2 Addestramento Rete Neurale

2.1 Back Propagation

L'algoritmo della *Back Propagation* valuta le derivate della funzione d'errore rispetto ai pesi; ovvero, una propagazione all'indietro degli errori della rete neurale.

In una rete feed-forward, ogni unitá calcola una somma pesata dei suoi input:

$$a_j = \sum_i w_{ji} z_i \quad (1)$$

z_i rappresenta l'attivazione di un' unitá, o input, che invia una connessione all'unitá j, e w_{ji} é il peso associato a tale connessione. La somma viene effettuata su tutte le unitá connesse a j. I bias possono essere inglobati nella sommatoria introducendo un ulteriore unitá con attivazione fissa ad 1. La sommatoria in (1) é trasformata da una funzione di attivazione non lineare $f()$ in modo da ottenere l'attivazione z_j dell'unità j nella seguente forma:

$$z_j = f(a_j) \quad (2)$$

In (1) le variabili z_i possono essere input della rete, in questo caso vengono denotate con x_i . In modo analogo, l'unità j (2) puó essere un output, quindi la sua attivazione é denotata con y_k . L'obiettivo é quello di individuare dei valori appropriati per i pesi della rete tramite la minimizzazione di una funzione d'errore appropriata. Supponiamo che la funzione d'errore é esprimibile come la somma degli errori su tutti i pattern del training set:

$$E = \sum_n E^n \quad (3)$$

Inoltre, l'errore E^n puó essere espresso come funzione differenziabile degli output della rete:

$$E^n = E^n(y_1, \dots, y_c) \quad (4)$$

Innanzitutto, si definisce una procedura per valutare le derivate della funzione d'errore E rispetto ai pesi e bias della rete neurale. Utilizzando la (3) si possono esprimere queste derivate come somma delle derivate dell'errore rispetto ai singoli pattern del training set. Per ogni pattern, sono state calcolate le attivazioni delle unitá nascoste e d'output tramite l'applicazione successive di (1) e (2). Questo processo é denominato Feed-Forward o propagazione in avanti, dato che rappresenta un flusso di informazione propagato attraverso la rete neurale. Si consideri il calcolo delle derivate di E^n rispetto ad un qualche peso w_{ji} . Gli output delle varie unitá dipendono dal pattern n-esimo. E^n

dipende dal peso w_{ji} tramite la somma di a_j in ingresso all'unità j . Quindi si può applicare la regola della catena per le derivate parziali ottendendo come segue:

$$\frac{\delta E^n}{\delta w_{ji}} = \frac{\delta E^n}{\delta a_j} \frac{\delta a_j}{\delta w_{ji}} \quad (5)$$

Nella seguente notazione:

$$\frac{\delta E^n}{\delta a_j} \equiv \delta_j \quad (6)$$

δ_j indica l'errore. Utilizzando la (1) scriviamo:

$$\frac{\delta a_j}{\delta w_{ji}} = z_j \quad (7)$$

Sostituendo la (6) e (7) nella (5), otteniamo:

$$\frac{\delta E^n}{\delta w_{ji}} = \delta_j z_j \quad (8)$$

Nell'ultima equazione, la derivata richiesta è ottenuta moltiplicando il valore di δ_j per il valore di z_j . L'unità j è il capo terminale della connessione, mentre l'unità i è il capo iniziale della connessione. In definitiva, per calcolare la derivata, bisogna calcolare il valore di δ per i nodi interni e nodi d'output della rete e successivamente applicare la (8). Per le unità d'output il calcolo dei δ è immediata, in quanto dalla definizione di (6) si ha:

$$\frac{\delta E^n}{\delta a_k} \equiv g'(a_k) \frac{\delta E^n}{\delta y_k} \quad (9)$$

Dove si è utilizzato la (2) con z_k denotata con y_k .

Per i δ dei nodi nascosti:

$$\delta_j \equiv \frac{\delta E^n}{\delta a_j} = \sum_k \frac{\delta E^n}{\delta a_k} \frac{\delta a_k}{\delta a_j} \quad (10)$$

Dove la sommatoria avviene per tutte le unità k alle quali j invia le connessioni. Le unità etichettate con k possono essere sia altre unità nascoste che d'output. Il procedimento della back-propagation è illustrata in figura 2. Se si sostituisce la definizione di δ data in (6) nella (10) e facendo uso di (1) e (2) si ottiene la seguente formula di back-propagation:

$$\delta_j = g'(a_j) \sum_k w_{kj} \delta_k \quad (11)$$

La quale suggerisce che il valore di δ per una data unità nascosta può essere ottenuto dalla propagazione all'indietro (back-propagation) dei δ delle unità negli strati superiori (strato d'output nel caso in cui lo strato superiore è composto dai nodi d'output) come mostrato in figura 2. Dato che si conosce già i valori per δ dei nodi d'output, segue che applicando ricorsivamente la (11) si può valutare i δ per tutte le unità nascoste in una rete feed-forward, indipendentemente dalla sua topologia. In sintesi, la procedura di back-propagation per il calcolo delle derivate dell'errore E^n rispetto ai pesi è suddivisa in 4 passi:

1. Dare in input alla rete un vettore (detto pattern) x_n e propagarlo in avanti usando la (1) e (2) per trovare le attivazioni di tutti i nodi interni ed di output (feed-forward);
2. Valutare i δ_k per tutte le unità di output usando la (9);
3. Propagare all'indietro i δ usando (11) per ottenere δ_j per ogni nodo interno della rete;
4. Usare la (8) per valutare le derivate richieste.

La derivata dell'errore E può essere ottenuta ripetendo i passi sopra citati per ogni pattern del training set, e successivamente sommare:

$$\frac{\delta E}{\delta w_{ji}} = \sum_n \frac{\delta E^n}{\delta w_{ji}} \quad (12)$$

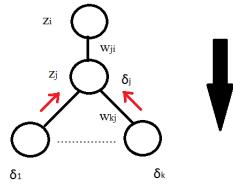


Figure 2: *Calcolo dei δ_j per i nodi interni j tramite la back-propagation dei δ_k dalle unità k alle quali j invia le connessioni*

Riguardo l'efficienza se indichiamo con W il numero totale di pesi e bias della rete neurale, la back-propagation ha complessità $O(W)$ perché sia la feed-forward che la back-propagation hanno complessità $O(W)$ ed il calcolo delle derivate usando la (8) ha ancora costo $O(W)$.

2.2 Resilient Back-Propagation

La Resilient Back Propagation detta *Rprop* é una tecnica batch di apprendimento adattativo locale che garantisce una rapida convergenza. Essa esegue una modifica del valore dei pesi basandosi sulle informazioni fornite dal gradiente.

Il principio é quello di eliminare la grandezza della derivata parziale dell'errore rispetto al valore dei pesi, poiché ciò che interessa nel processo di minimizzazione é il segno della derivata (la direzione dell'aggiornamento dei pesi) e non l'effettivo valore. Ogni peso é aggiornato esclusivamente in base ad un specifico parametro, il margine di aggiornamento $\Delta_{ij}^{(t)}$:

$$\Delta w_{ij}^{(t)} \begin{cases} -\Delta_{ij}^{(t)} & se \frac{\delta E^{(t)}}{\delta w_{ij}} > 0 \\ +\Delta_{ij}^{(t)} & se \frac{\delta E^{(t)}}{\delta w_{ij}} < 0 \\ 0 & altrimenti \end{cases} \quad (13)$$

dove $\frac{\delta E^{(t)}}{\delta w_{ij}}$ denota la (12) al passo t-esimo. Il margine di aggiornamento é il seguente:

$$\begin{cases} \eta^+ * \Delta_{ij}^{(t-1)} & se \frac{\delta E^{(t-1)}}{\delta w_{ij}} * \frac{\delta E^{(t)}}{\delta w_{ij}} > 0 \\ \eta^- * \Delta_{ij}^{(t-1)} & se \frac{\delta E^{(t-1)}}{\delta w_{ij}} * \frac{\delta E^{(t)}}{\delta w_{ij}} < 0 \\ 0 & altrimenti \end{cases} \quad (14)$$

dove $\eta^+ > 1 > \eta^- > 0$.

Il funzionamento dell'algoritmo é il seguente: ogni volta che la derivata parziale dell'errore su w_{ij} cambia segno (cioé l'ultimo aggiornamento é stato troppo grande) e si é superato il minimo locale, il margine di aggiornamento $\Delta_{ij}^{(t)}$ viene diminuito di un fattore η^- . Se la derivata conserva il segno, il margine viene leggermente aumentato in modo da accelerare la convergenza in regioni piatte. Se il segno cambia, allora nel passo di apprendimento successivo non ci dovrá essere un adattamento. Si ottiene ciò impostando $\frac{\delta E^{(t)}}{\delta w_{ij}} > 0$ nella (13).

I valori che si danno, in genere, sono $\eta^+ = 1.1$ ed $\eta^- = 0.5$. Da come si evince, la Rprop necessita di due parametri: il margine di aggiornamento iniziale Δ_0 e un limite massimo per l'aggiornamento Δ_{max} . Inizialmente tutti i margini venono inizializzati ad Δ_0 e questo parametro é impostato ad 0.1 per default. Il parametro Δ_{max} limita superiormente la dimensione del margine di aggiornamento. Il suo valore di default é 50.0. Inoltre, viene fissato un limite minimo affinché l'algoritmo non si blocchi troppo velocemente in qualche limite ottimale sub-locale. Di default $\Delta_{min} = 1e-6$.

3 Sviluppo Progetto

Per lo sviluppo di questo progetto, sono stati utilizzati diversi file. In seguito sono riportati le istruzioni dell'uso del main e una breve descrizione dei vari file del progetto.

N.B. Parleremo dei file sviluppati della parte B, visto che includono anche gli stessi file della parte A.

3.1 Main

Il file principale si chiama "main.m". Qui viene effettuato:

- caricamento e visualizzazione (tramite plot) del banana dataset;
- estrazione di un numero N di punti dal dataset caricato;
- creazione di una rete neurale con due nodi di input e uno di output (per quanto riguarda gli strati interni, l'utente deciderà il numero di strati interni e il numero di nodi per ogni strato interno, modificando le variabili m e n nel file main.m);
- visualizzazione (tramite plot) del training set e del validation set;
- addestramento della rete (esso viene ripetuto per un certo numero di epoche);
- visualizzazione degli errori calcolati e dei risultati ottenuti sia sul training che sul validation.

3.2 Altre funzioni

3.2.1 new_net

- Input: numero nodi dello strato di input (d), numero nodi degli strati interni (m) e numero nodi dello strato di output (c).
- Output: Rete Neurale
- Descrizione: Viene creata una struttura che contiene un array di pesi per ogni strato della rete (W), un array di bias per ogni strato della rete (b), i delta dei pesi e dei bias che verranno utilizzati nella Rprop (rispettivamente deltaW e deltaB).

3.2.2 feedForward

- Input: Rete Neurale (net), input dello strato precedente (x) e funzione di output scelta (funzione_output).
- Output: Risultato nodo di output (y).
- Descrizione: Implementazione della feed forward (propagazione in avanti).

3.2.3 backPropagation

- Input: Rete Neurale (net), input training set (XT), target training set (TT), funzione di output scelta (function_output) e funzione di errore scelta (function_error).
- Output: Delta Pesi (DW) e Delta Bias (DB) di ciascun livello della rete neurale.
- Descrizione: Implementazione della back-propagation. Richiama la feedForward e poi la funzione generateDelta.

3.2.4 simulationNetwork

- Input:
- Output: Risultato della funzione di output (scelta dall'utente).
- Descrizione: Essa contiene tre funzioni di output: *computeSig* che implementa la sigmoide; *computeTanh* che implementa la funzione tanH; infine *computeIdentity* che implementa la funzione identità.

3.2.5 RProp

- Input: Rete Neurale (net), delta dei pesi e bias della rete calcolati tramite backPropagation (rispettivamente DW e DB).
- Output: Rete Neurale con i pesi e i bias aggiornati.
- Descrizione: Implementazione della Resilient Back Propagation.

3.2.6 loadAndShowBananaDataSet

- Input: input (bananaInput) e target (bananaTarget) del banana dataset.
- Output:
- Descrizione: Caricamento del banana dataset in matlab e visualizzazione tramite plot del dataset caricato.

3.2.7 plotDataDS

- Input: ascissa e ordinata.
- Output:
- Descrizione: plot in 2D dei valori di ascissa e ordinata dati in input.

3.2.8 derivativeFunction

- Input: input dello strato dove si sta calcolando il delta.
- Output: calcolo derivata sull'input;
- Descrizione: dato l'input dello strato della rete dove si sta calcolando il delta, si calcola la derivata della funzione di output rispetto all'input a_j .

4 Test

Abbiamo effettuato diverse simulazioni al fine di poter studiare l'apprendimento della rete neurale implementata, in differenti casi. Fissato un numero di strati interni, si è scelto di aumentare il numero di nodi ad ogni test, aumentandone la complessità per comprenderne il comportamento.

- i Il numero degli strati interni viene fissato ai valori 1, 5, 10 e 20.
- ii Per ogni strato interno, abbiamo inserito 1, 2, 4, 8, 16, 32, 64 e 128 nodi.

I seguenti parametri restano costanti durante il processo di apprendimento:

1. I nodi di input, che abbiamo fissato a 2;
2. Abbiamo 1 nodo di output;
3. Funzione di output: **Sigmoide**;
4. Funzione di errore: **Somma di Quadrati**;
5. Numero di **epoch**: 1000;
6. Numero totale di punti presi dal Banana data set: 400.

4.1 Tabelle comparative degli errori

Di seguito sono riportate le tabelle delle simulazioni effettuate, riportando gli andamenti degli errori del training set e del validation set.
Le tabelle riportano gli errori alla fine del ciclo di addestramento, ovvero quando il numero di epoch è uguale a 1000.

4.1.1 Test per 1 strato interno

Strati	Nodi	E.Training	E.Validation
1	1	21.3953	25.254
1	2	12.0323	16.7459
1	4	12.7411	18.2226
1	8	6.2706	7.9771
1	16	5.9833	9.7764
1	32	7.1242	12.1342
1	64	5.8172	6.2356
1	128	7.6883	13.179

4.1.2 Test per 5 strati interni

Strati	Nodi	E.Training	E.Validation
5	1	23.0113	25.8365
5	2	22.3492	23.8572
5	4	15.0519	19.6945
5	8	11.8023	13.2224
5	16	6.3454	12.9608
5	32	13.3868	15.417
5	64	5.6538	17.2592
5	128	9.8643	14.6942

4.1.3 Test per 10 strati interni

Strati	Nodi	E.Training	E.Validation
10	1	22.3954	25.0434
10	2	22.7868	22.3181
10	4	21.821	22.2275
10	8	21.2859	22.809
10	16	11.7057	15.8675
10	32	8.1845	15.8428
10	64	13.5957	14.3397
10	128	22.9992	21.4357

4.1.4 Test per 20 strati interni

Strati	Nodi	E.Training	E.Validation
20	1	22.35	23.91
20	2	21.0188	21.3729
20	4	18.6721	21.811
20	8	16.9331	21.5531
20	16	20.7824	24.039
20	32	21.105	21.8014
20	64	25.3642	24.6344
20	128	24.96	24.7994

4.2 Grafici dei test effettuati

4.2.1 Grafici per 1 strato interno

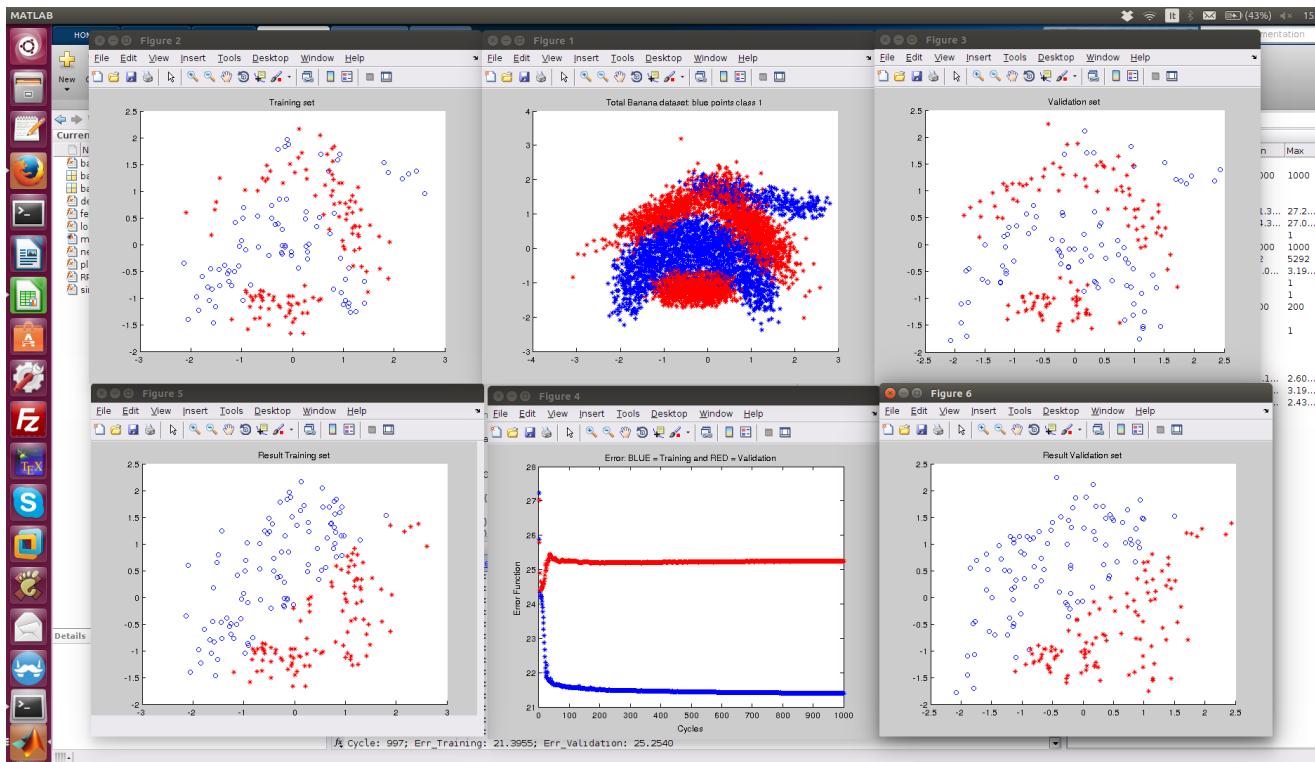


Figure 3: 1 strato interno - 1 nodo per strato

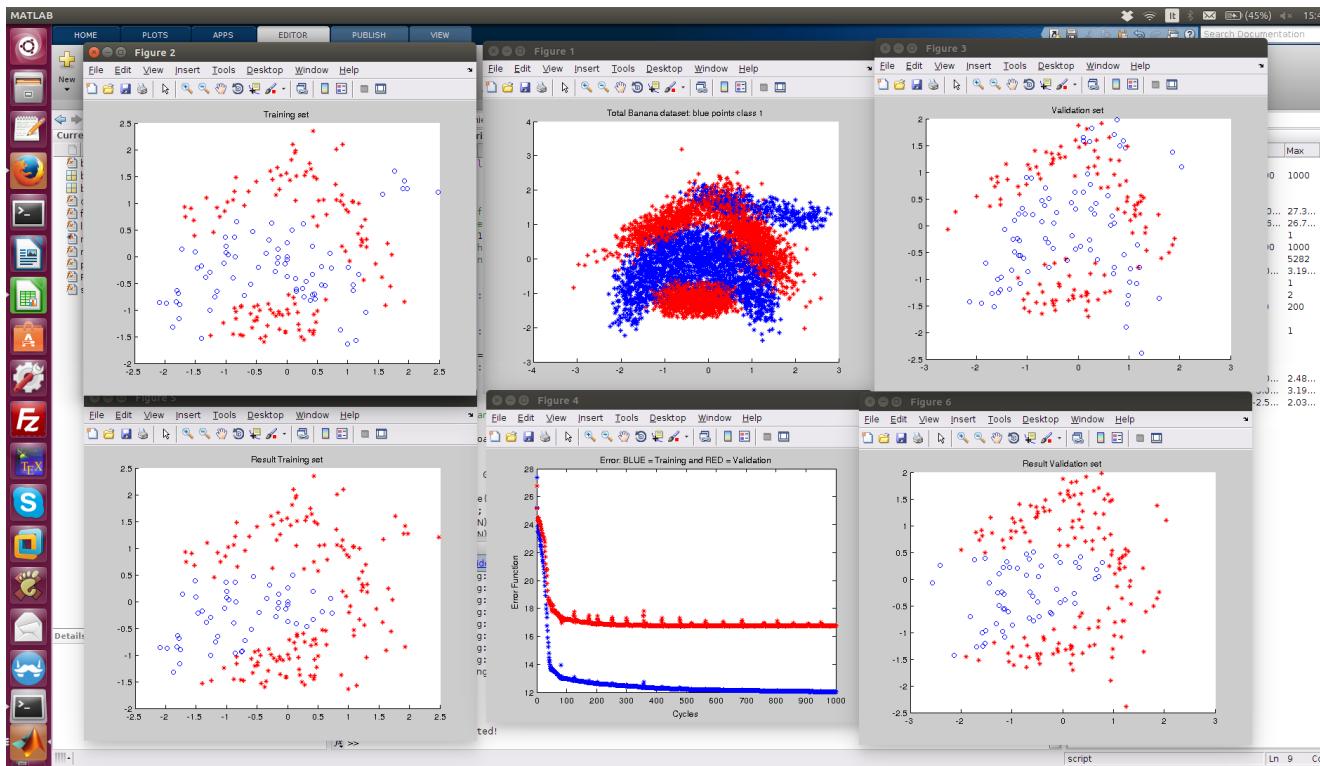


Figure 4: 1 strato interno - 2 nodi per strato

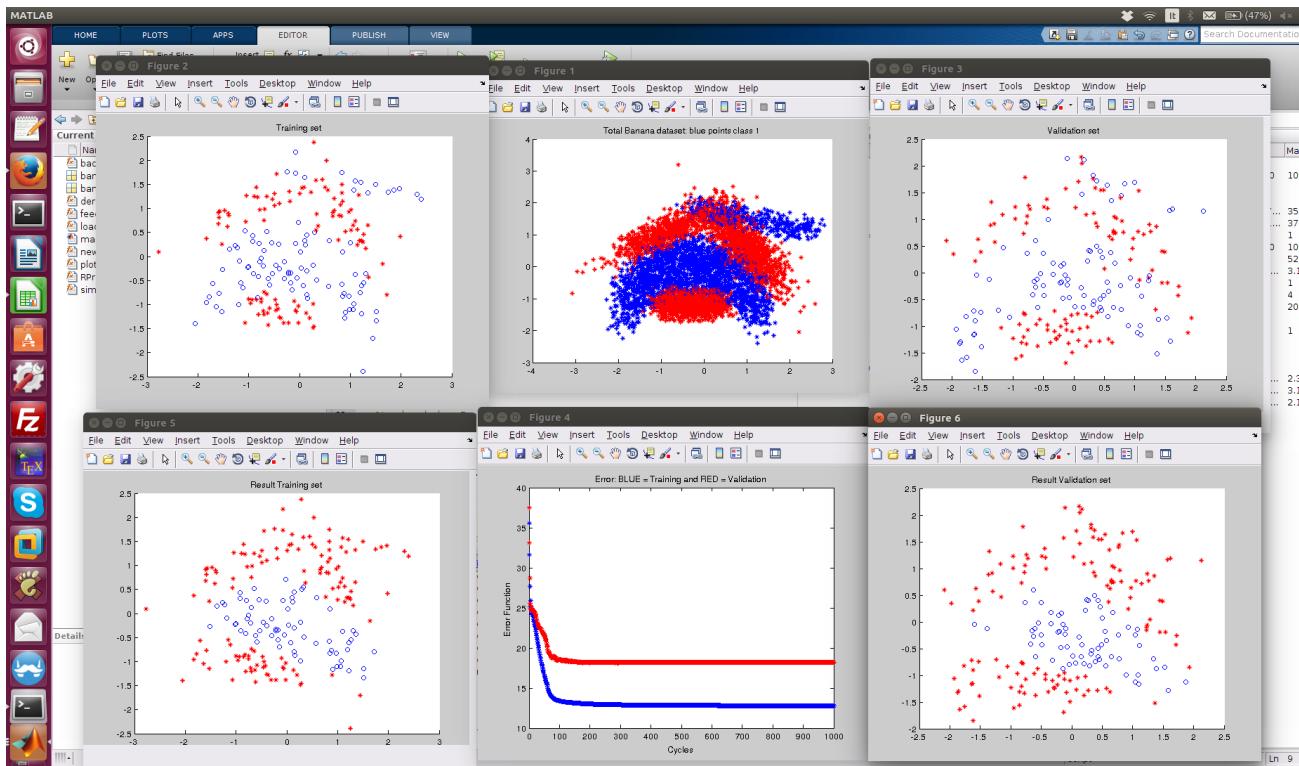


Figure 5: 1 strato interno - 4 nodi per strato

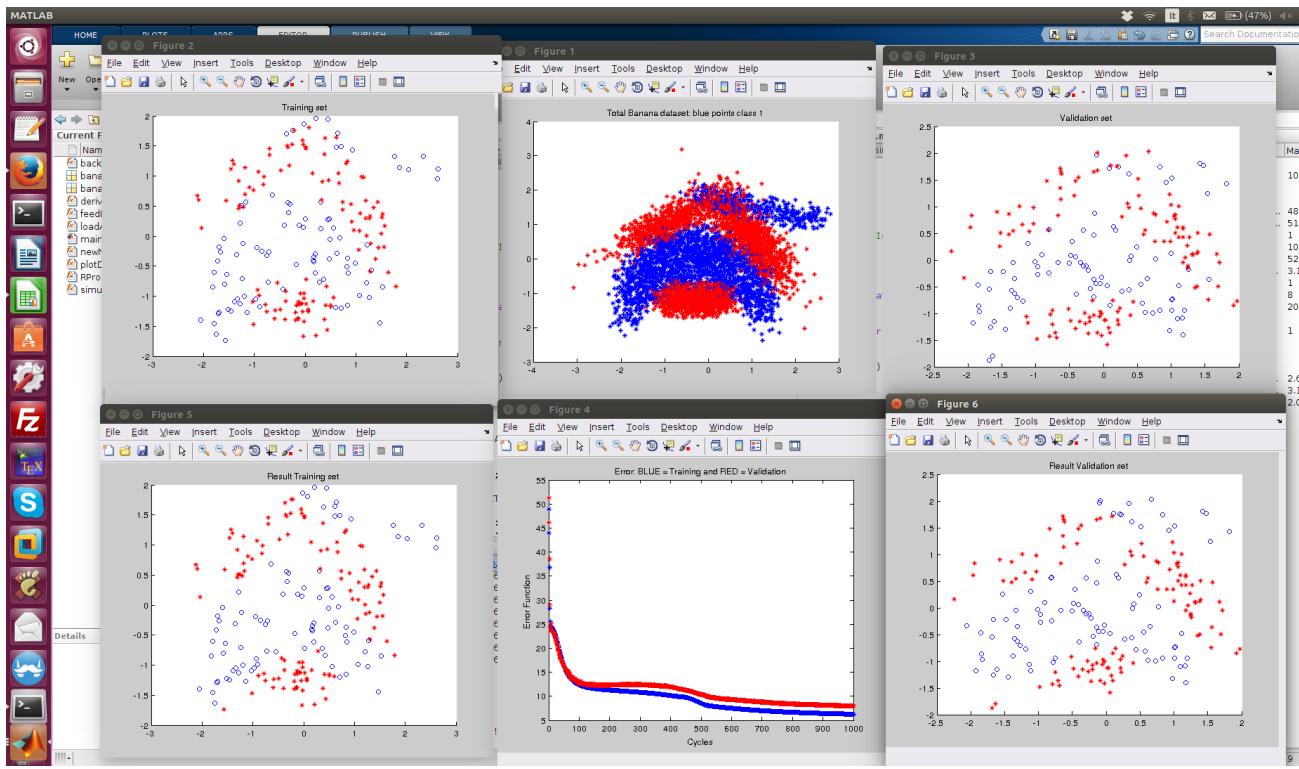


Figure 6: 1 strato interno - 8 nodi per strato

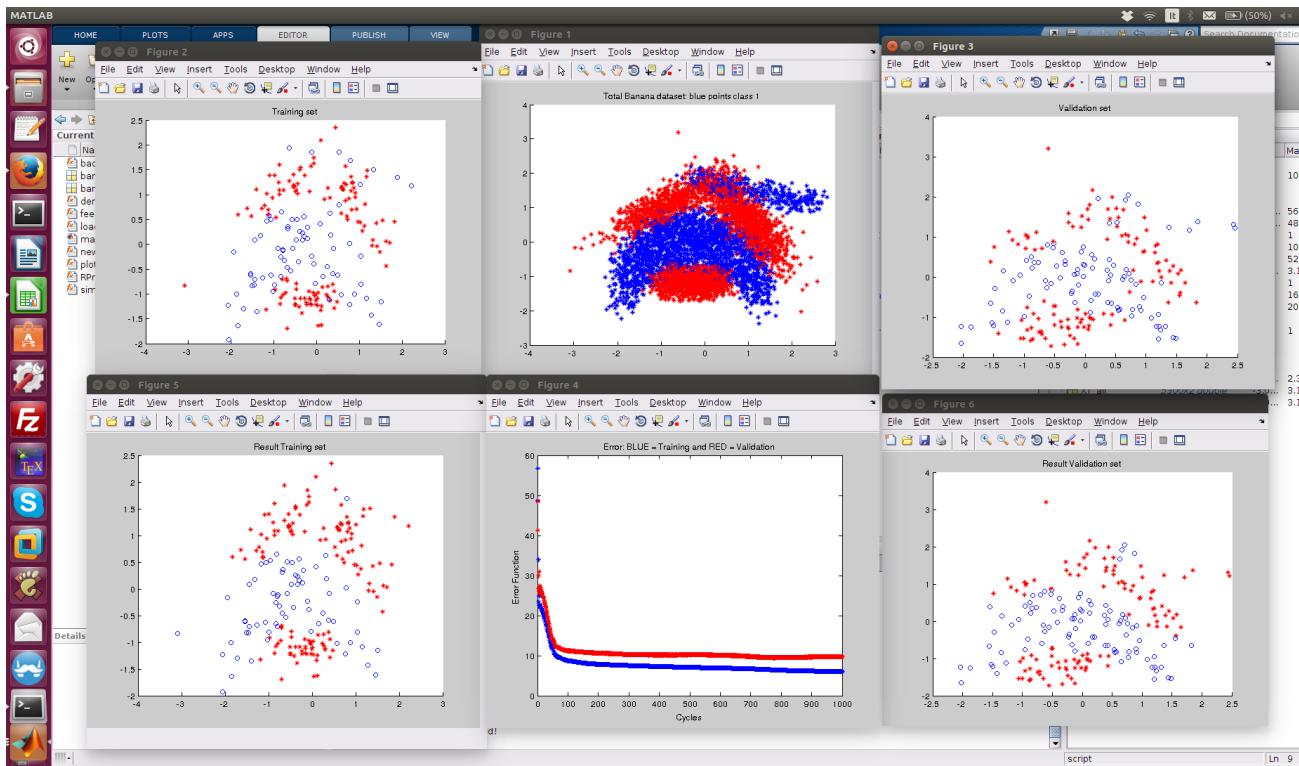


Figure 7: 1 strato interno - 16 nodi per strato

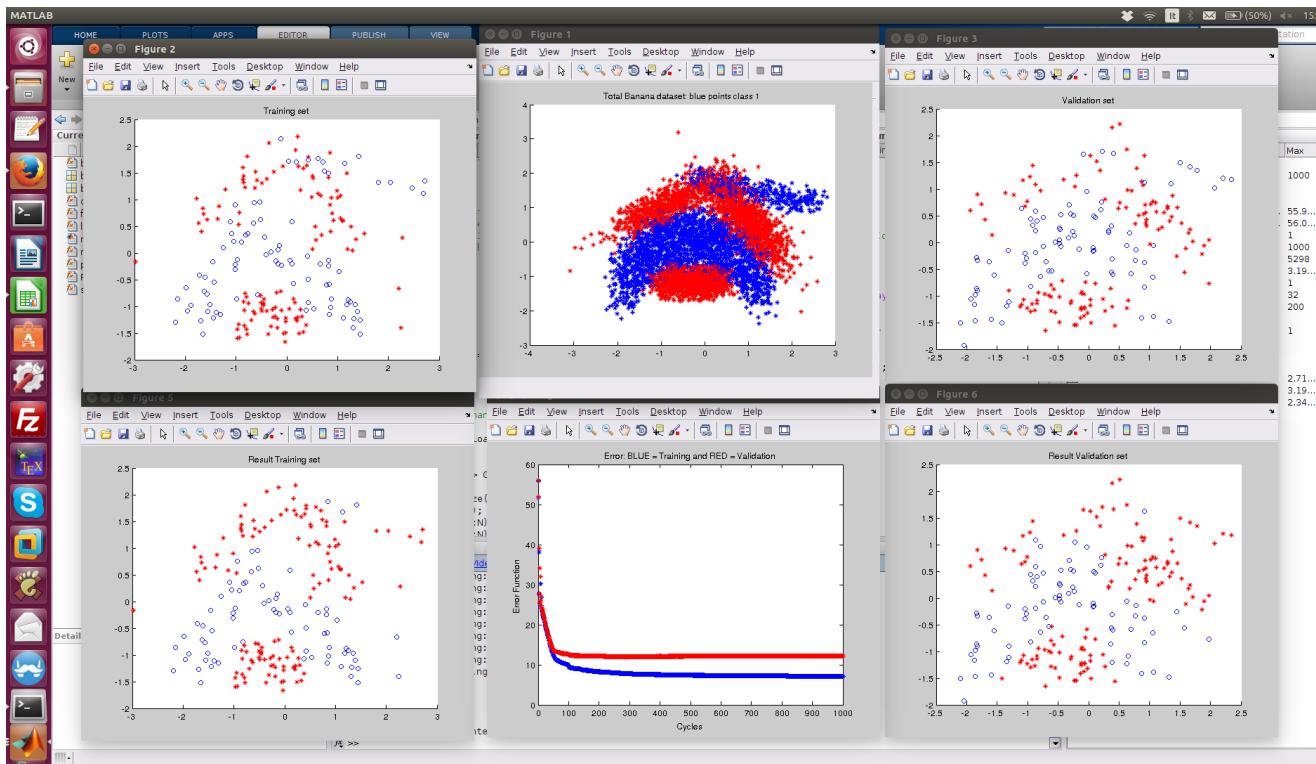


Figure 8: 1 strato interno - 32 nodi per strato

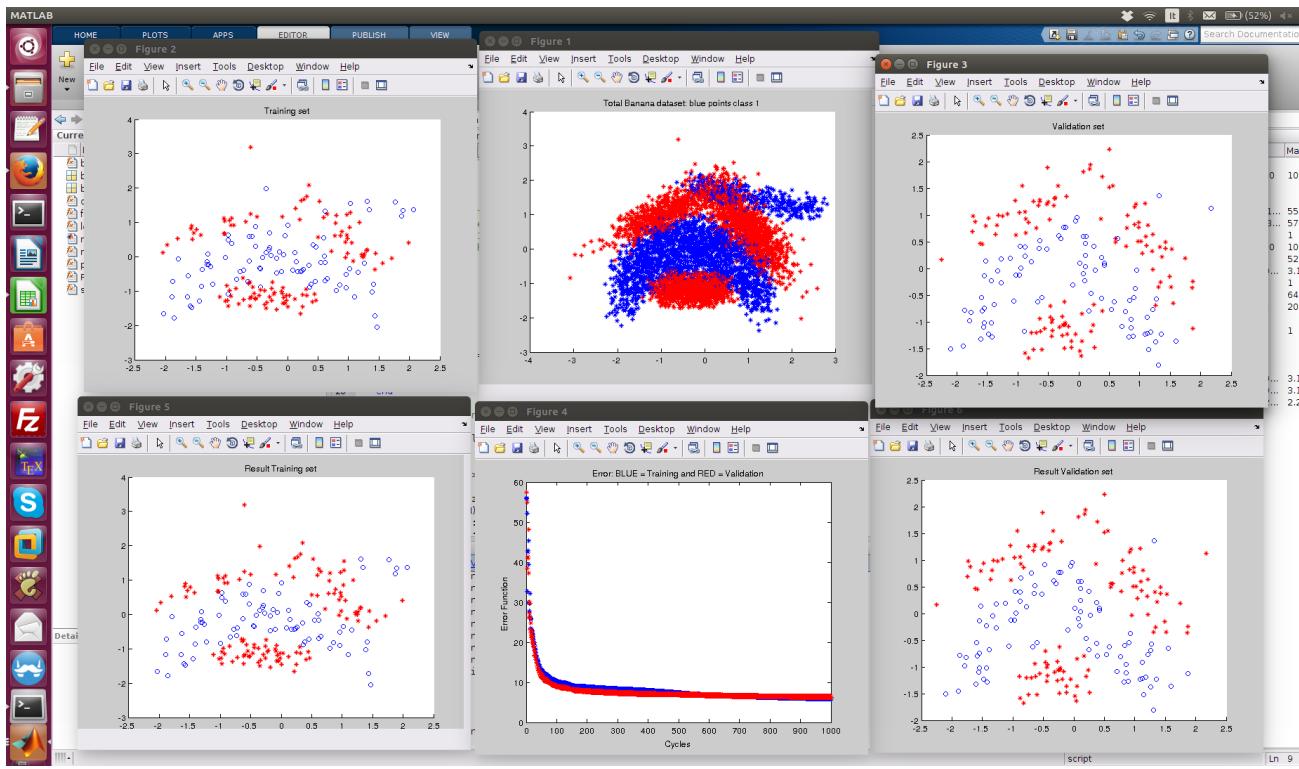


Figure 9: 1 strato interno - 64 nodi per strato

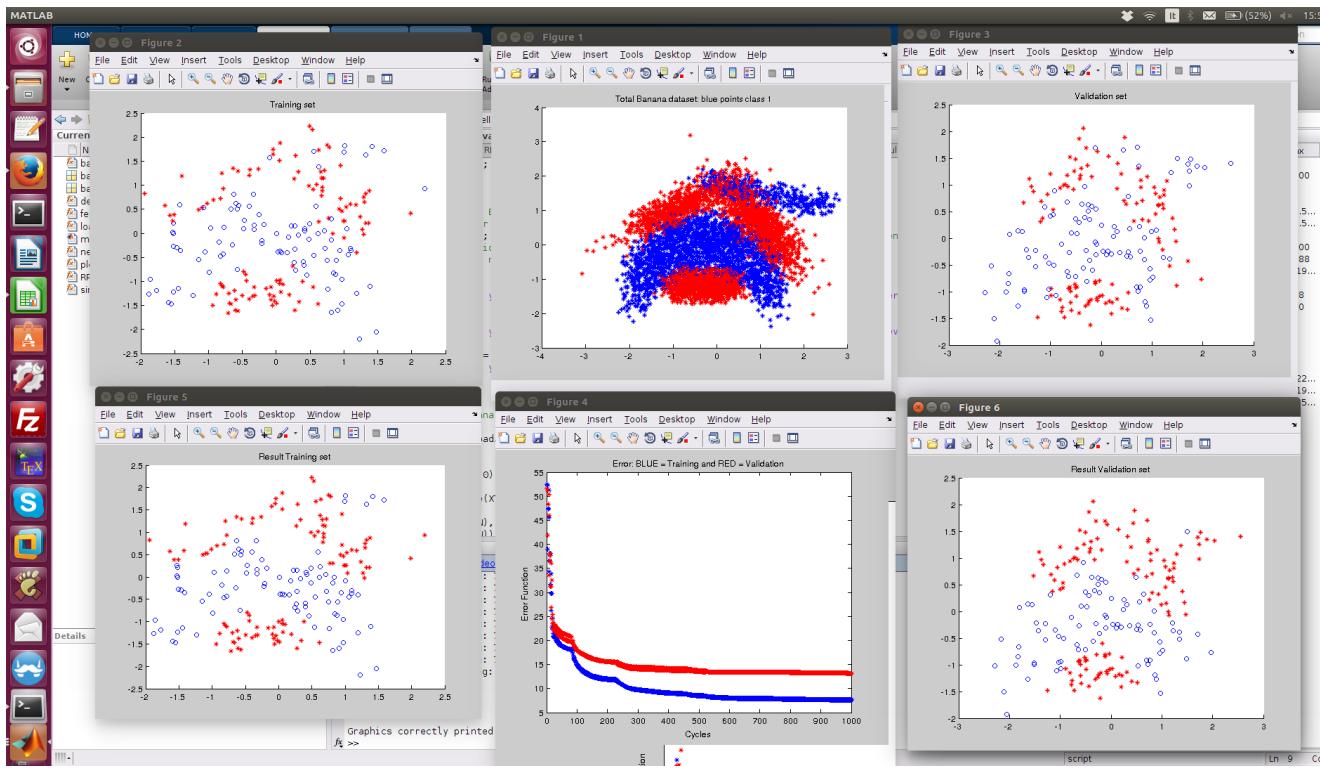


Figure 10: 1 strato interno - 128 nodi per strato

4.2.2 Grafici per 5 strati interni

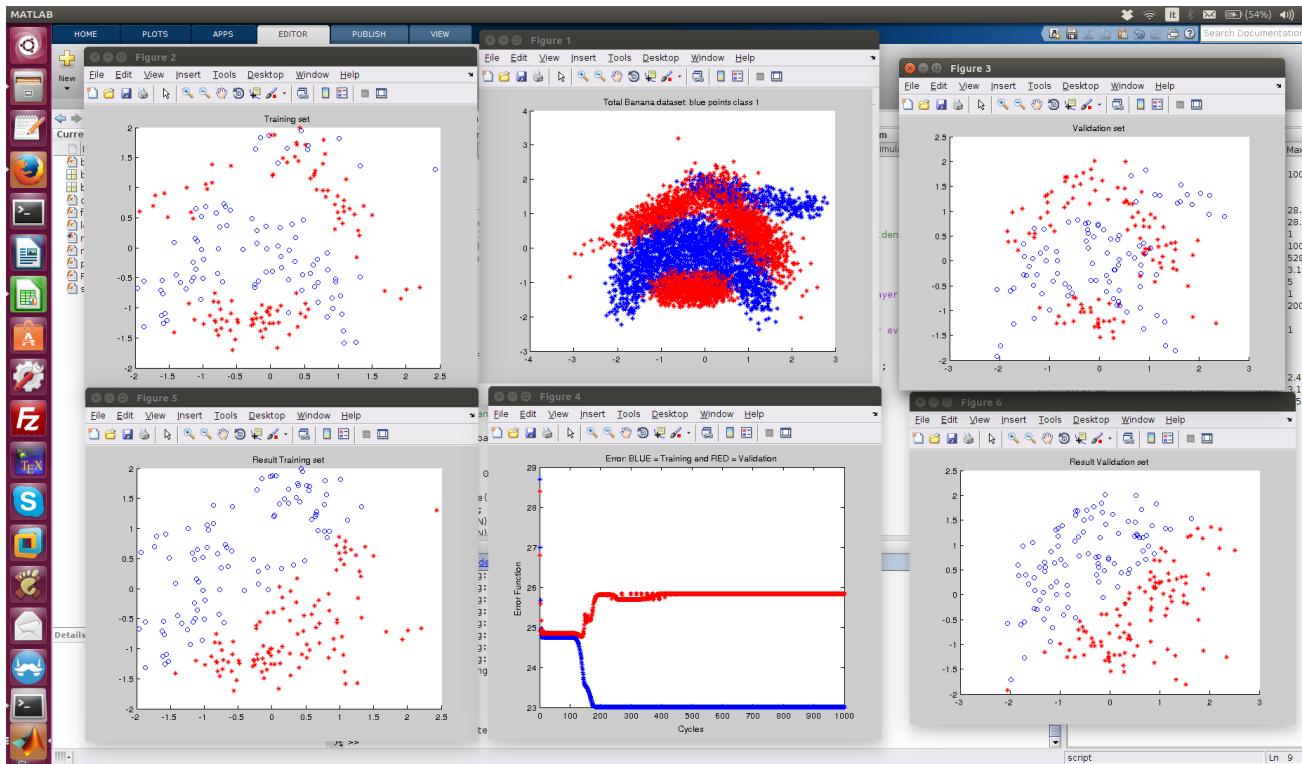


Figure 11: 5 strati interni - 1 nodo per strato

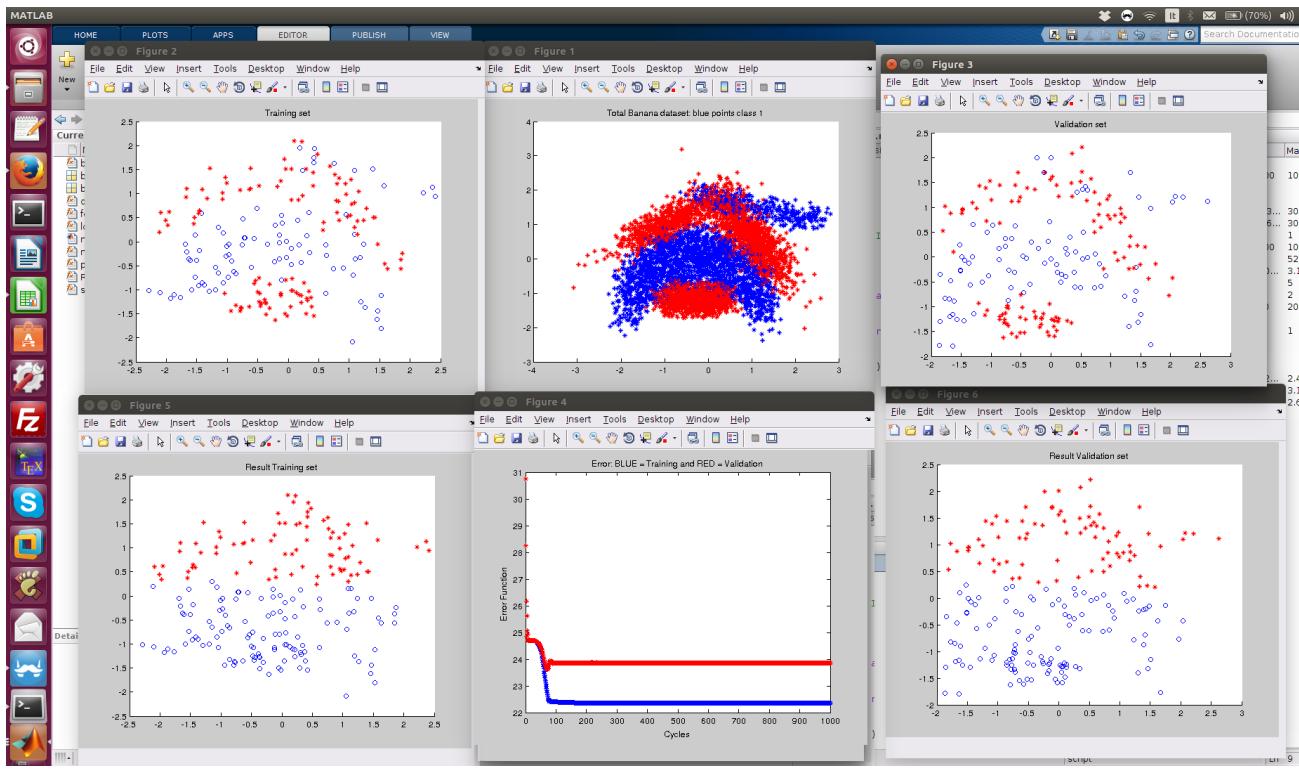


Figure 12: 5 strati interni - 2 nodi per strato

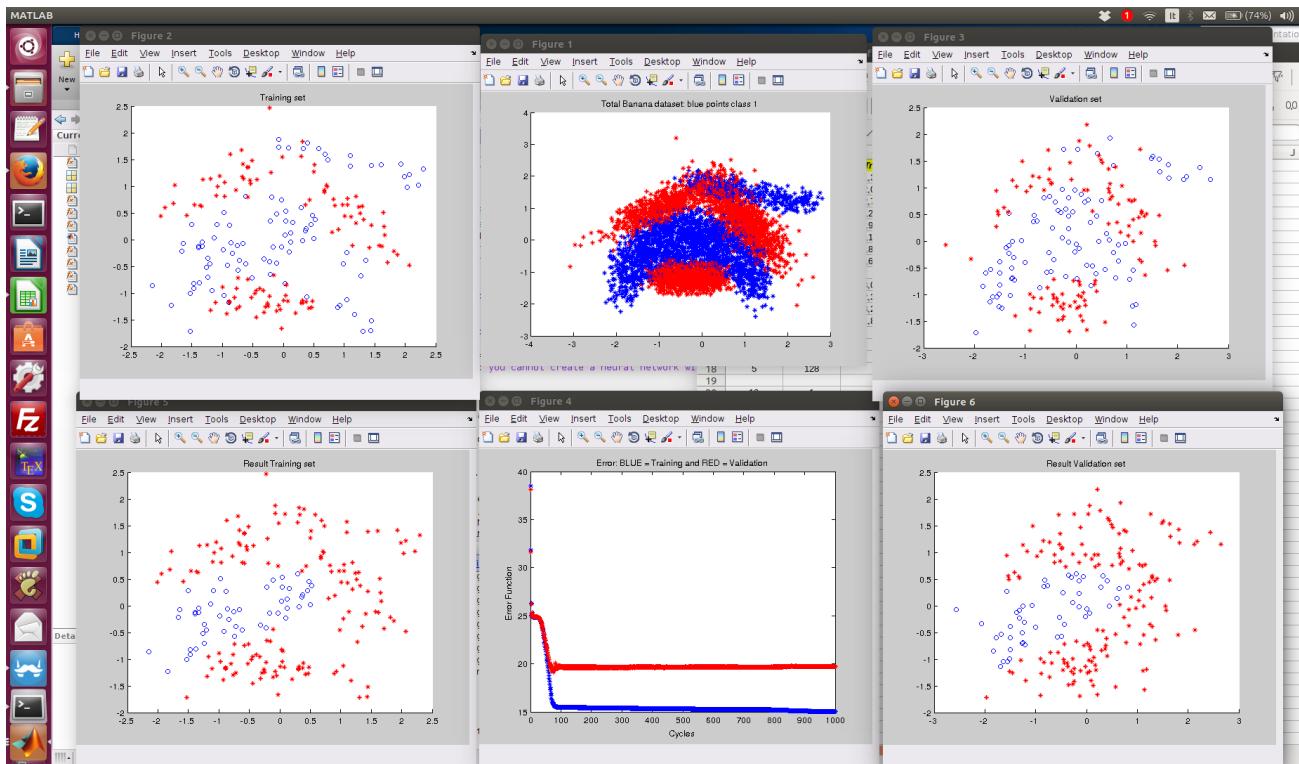


Figure 13: 5 strati interni - 4 nodi per strato

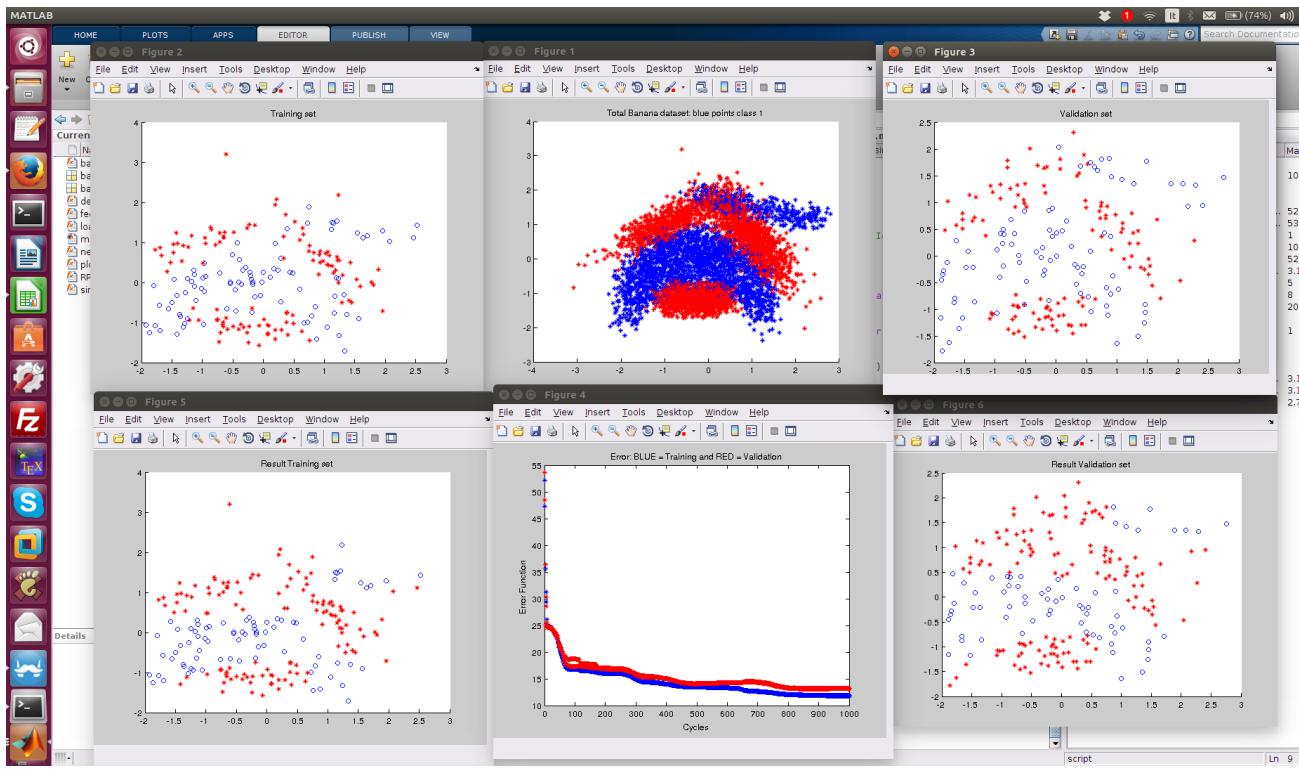


Figure 14: 5 strati interni - 8 nodi per strato

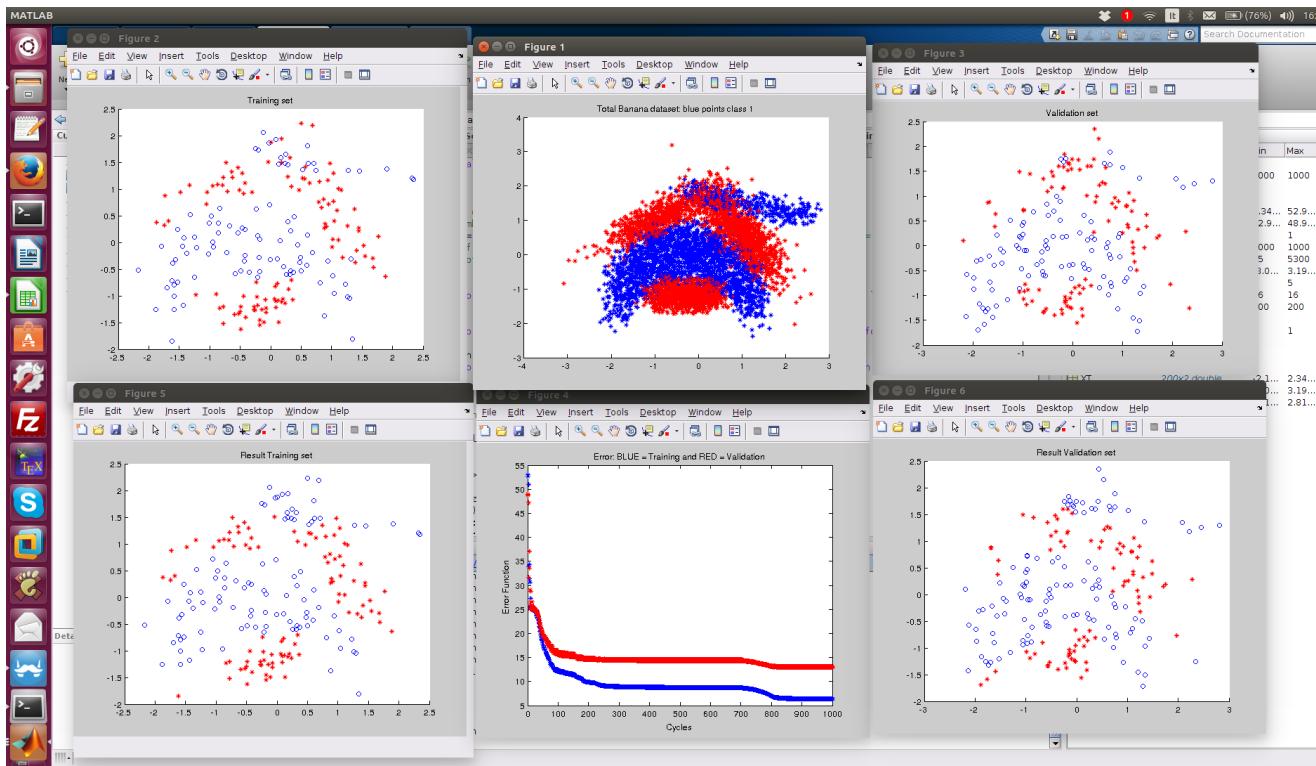


Figure 15: 5 strati interni - 16 nodi per strato

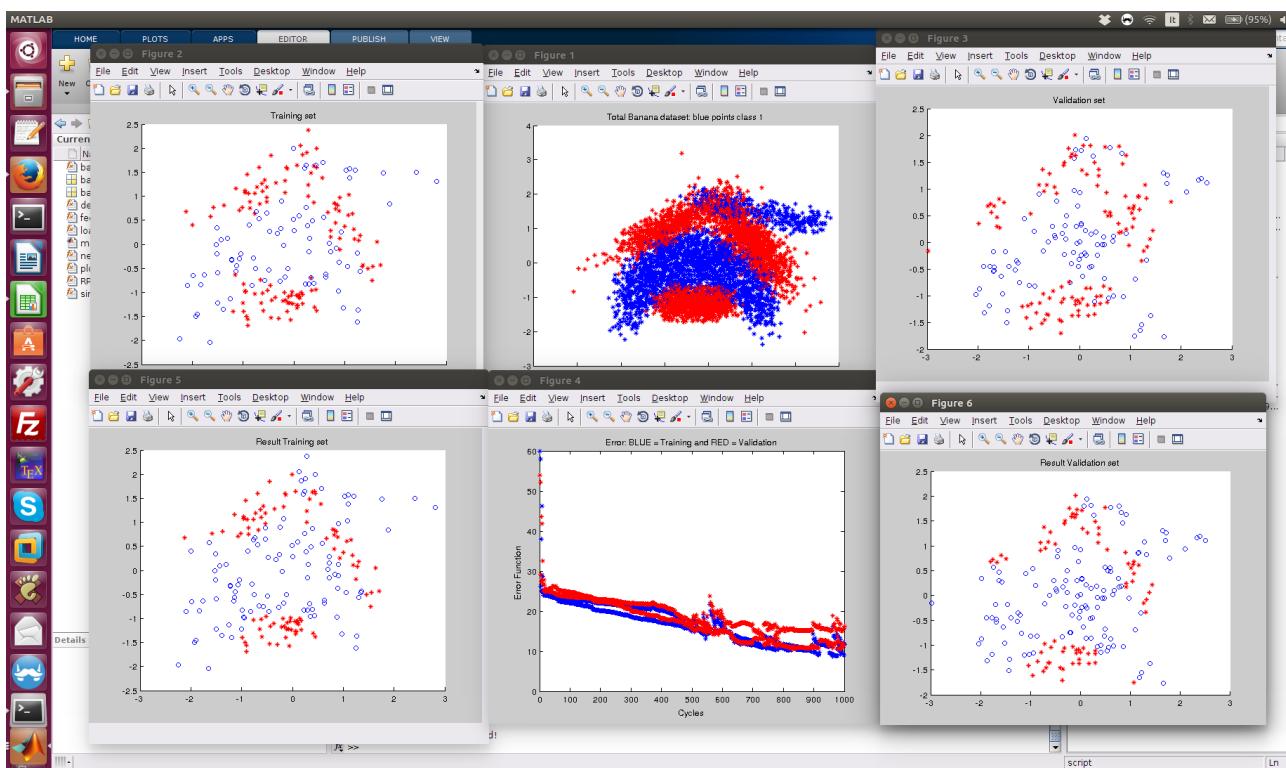


Figure 16: 5 strati interni - 32 nodi per strato

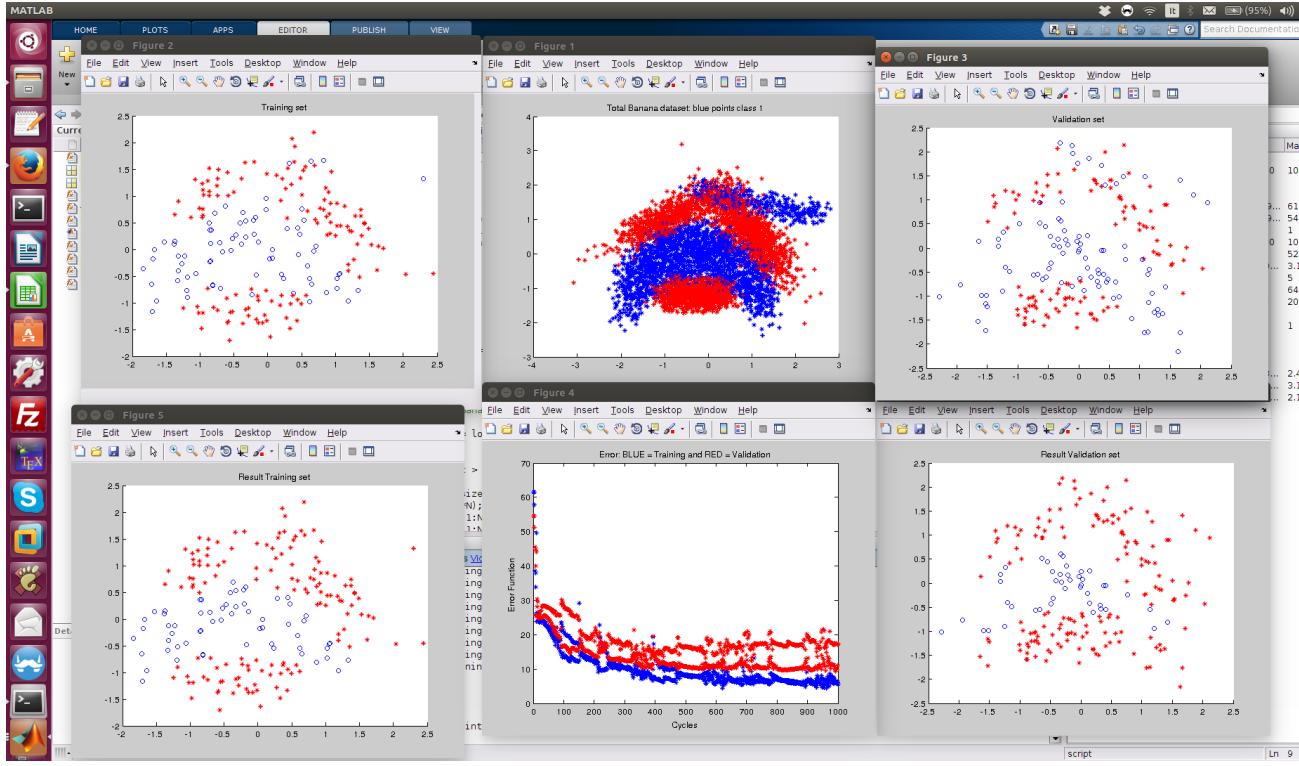


Figure 17: 5 strati interni - 64 nodi per strato

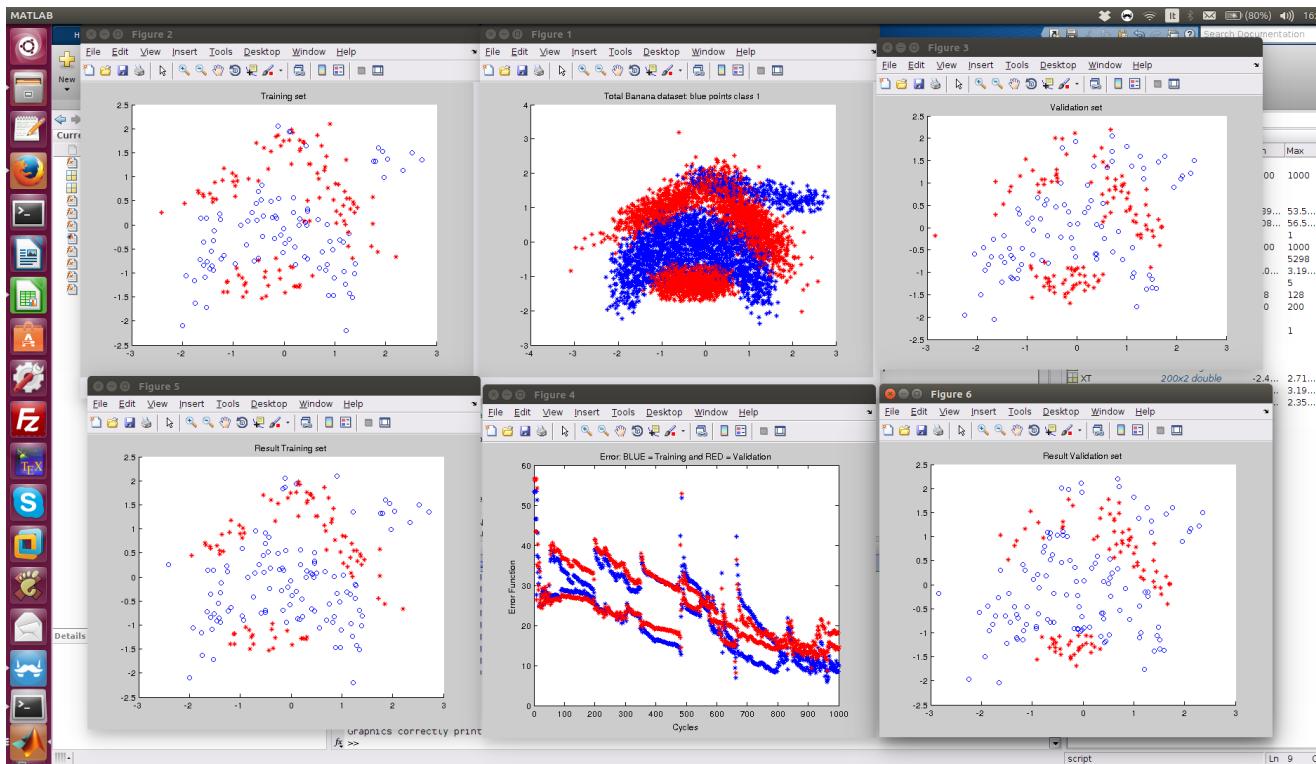


Figure 18: 5 strati interni - 128 nodi per strato

4.2.3 Grafici per 10 strati interni

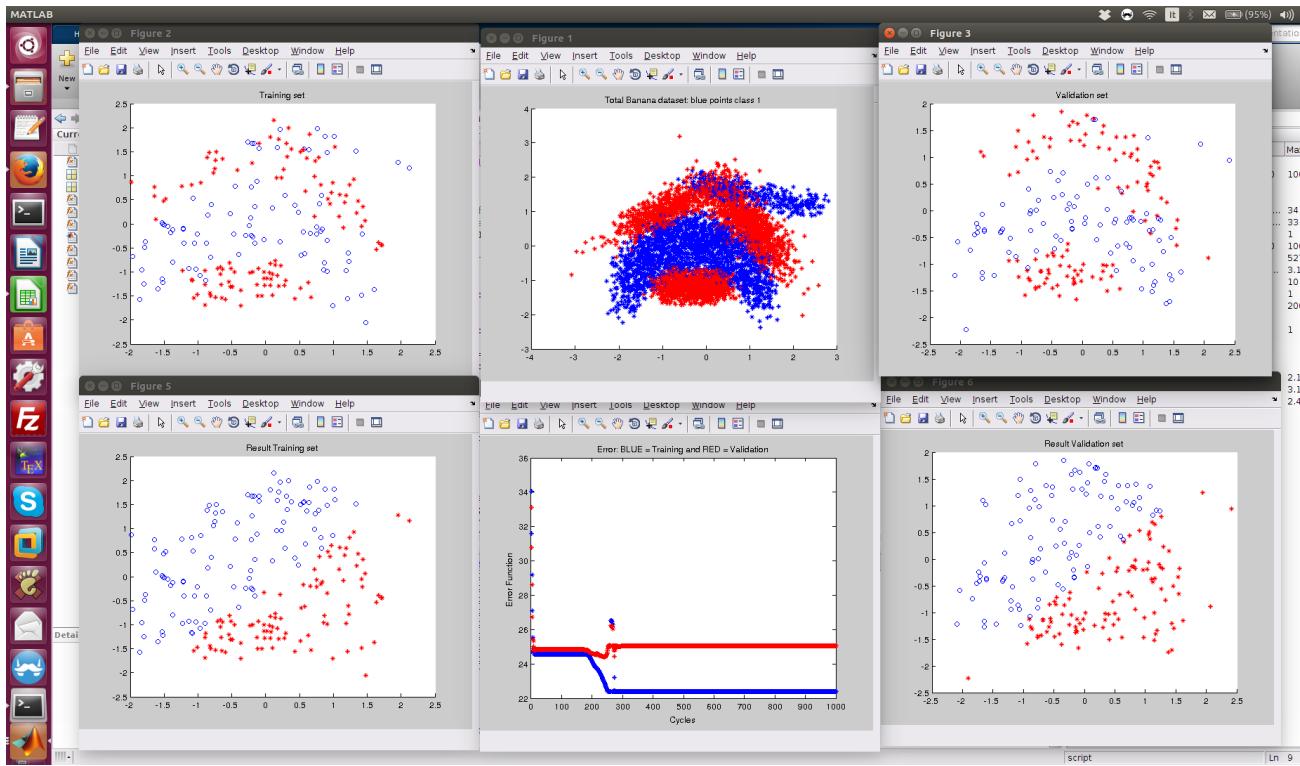


Figure 19: 10 strati interni - 1 nodo per strato

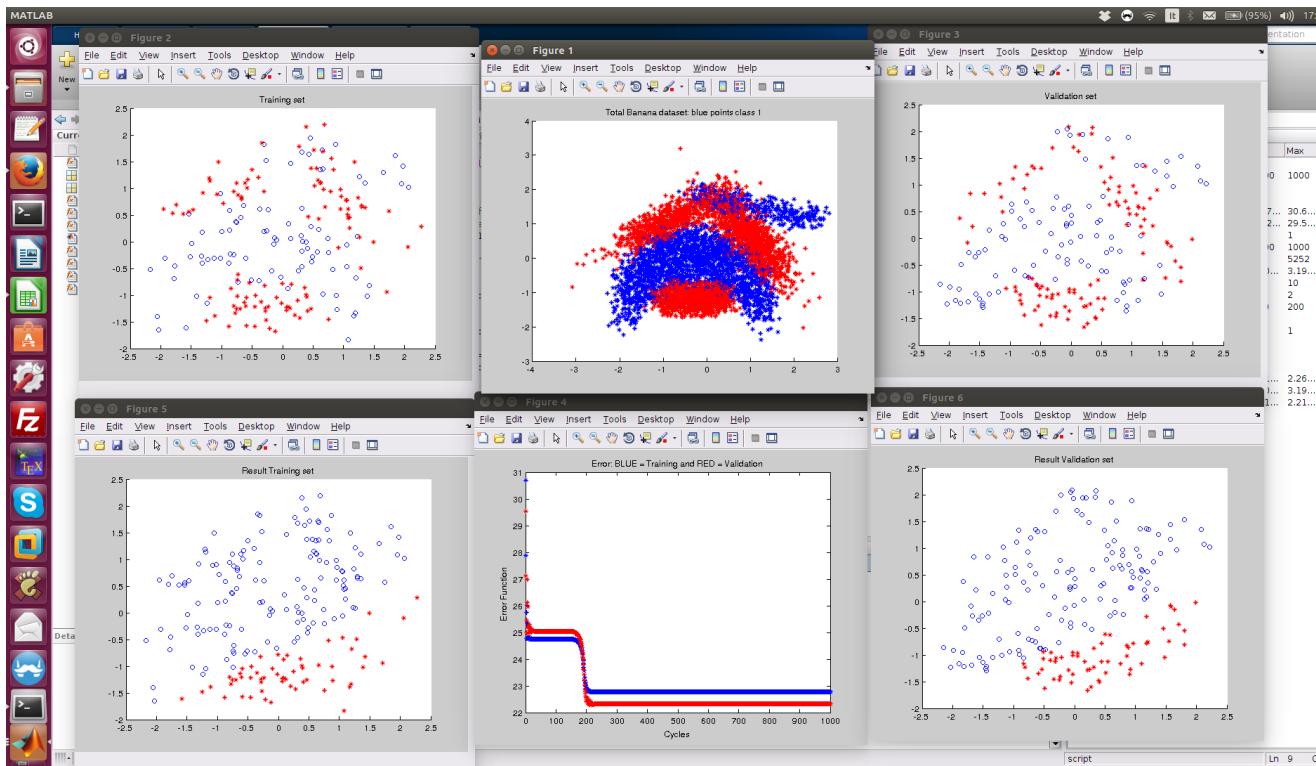


Figure 20: 10 strati interni - 2 nodi per strato

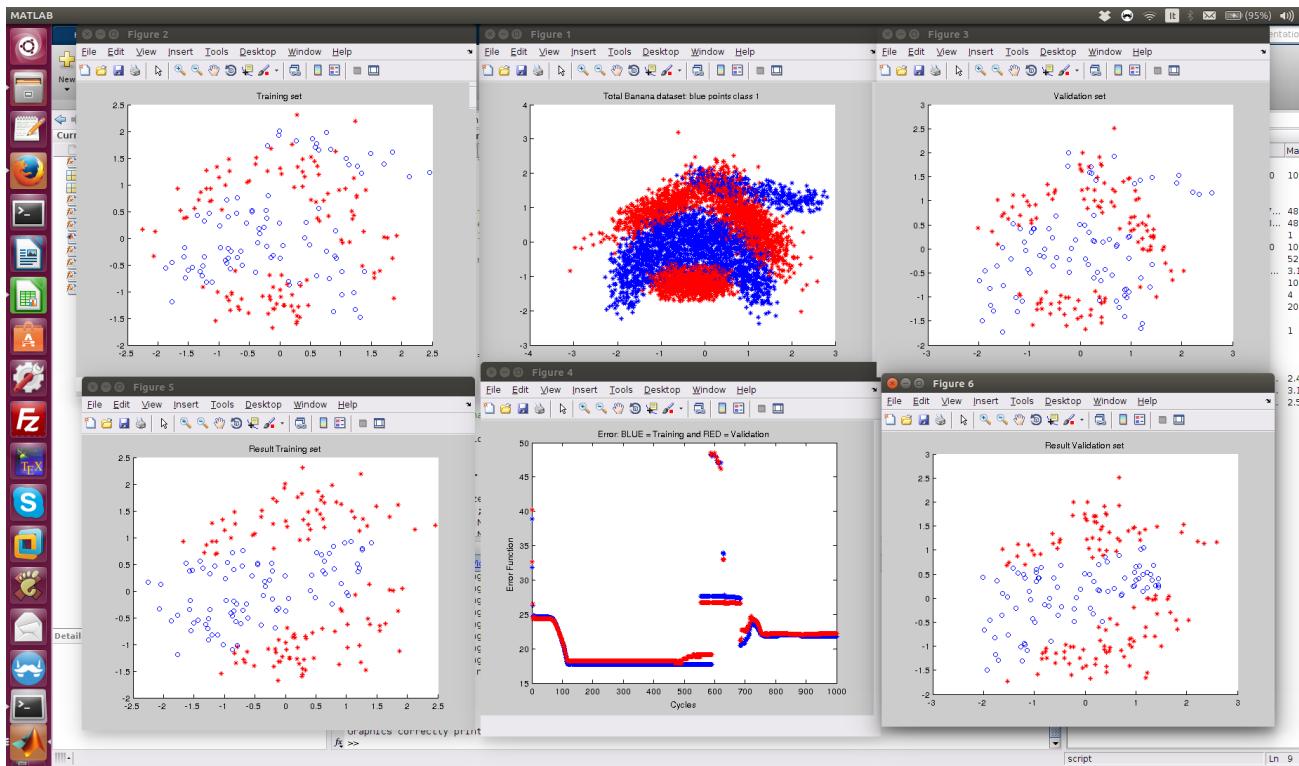


Figure 21: 10 strati interni - 4 nodi per strato

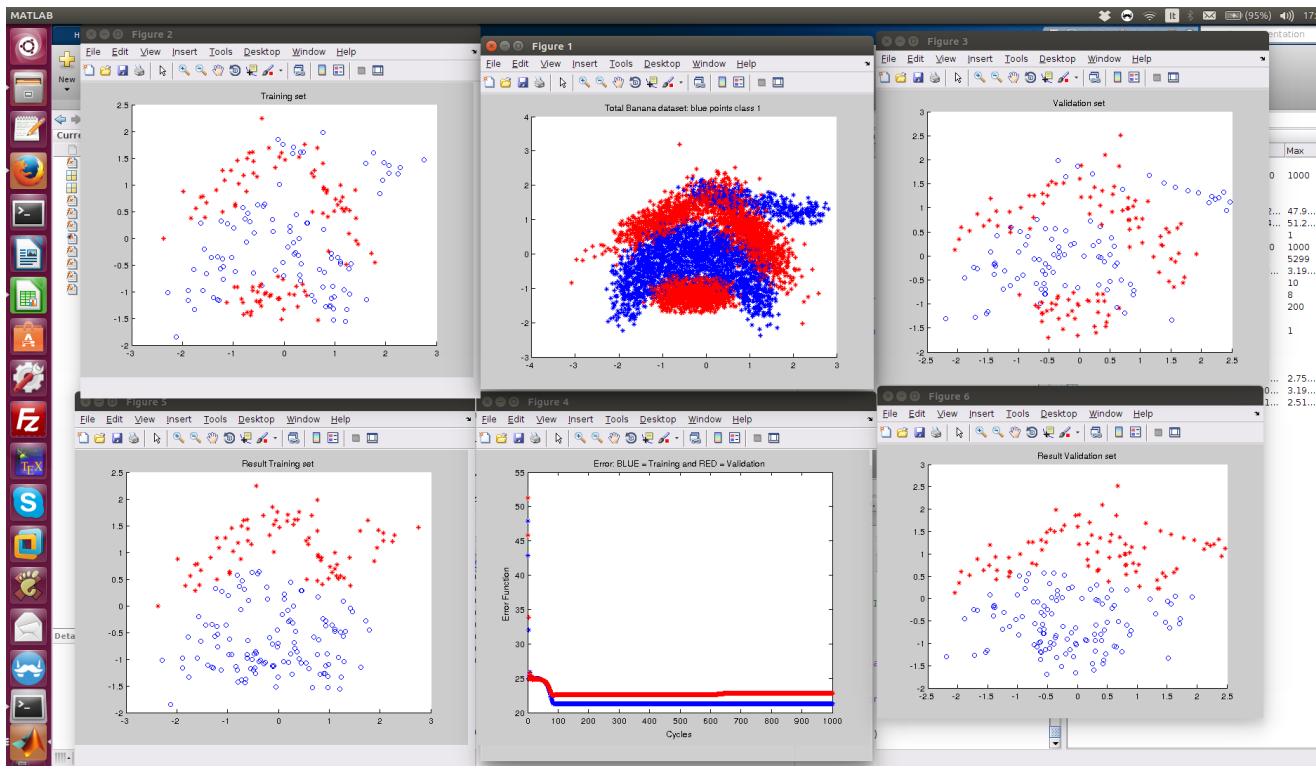


Figure 22: 10 strati interni - 8 nodi per strato

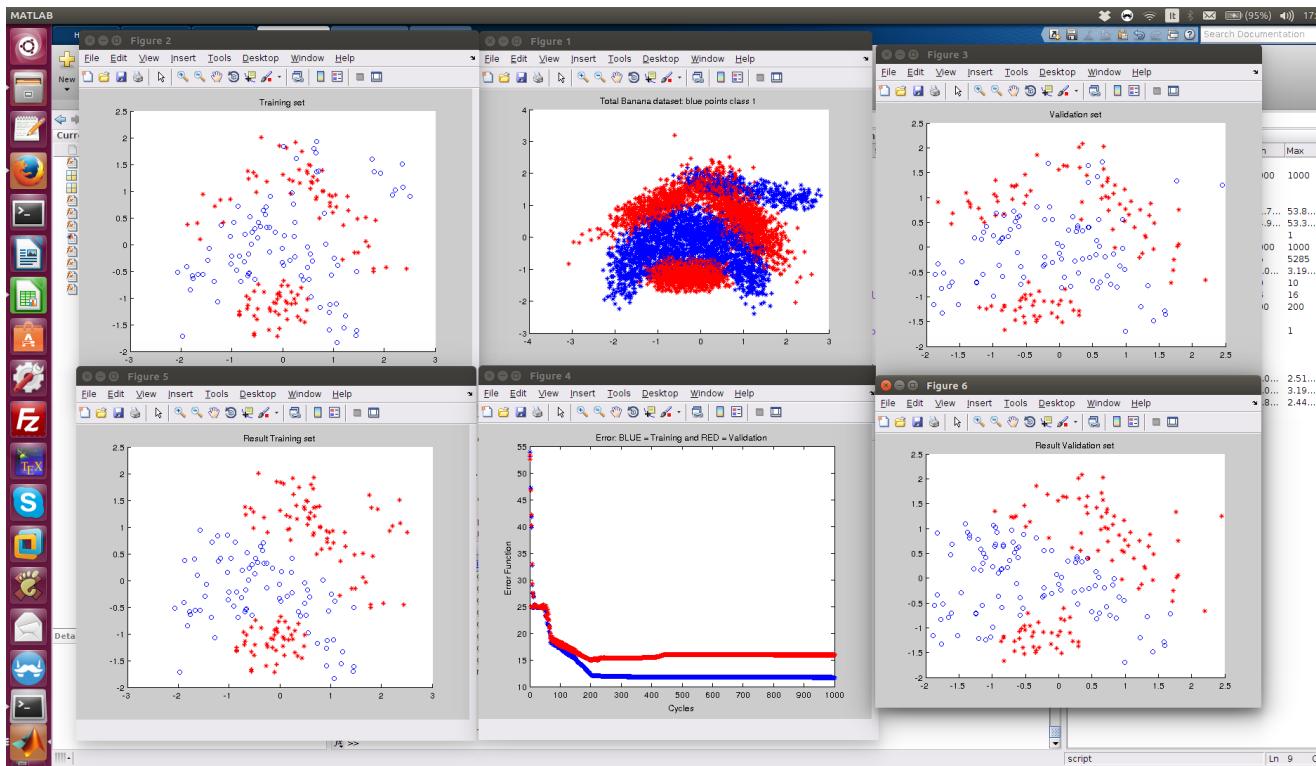


Figure 23: 10 strati interni - 16 nodi per strato

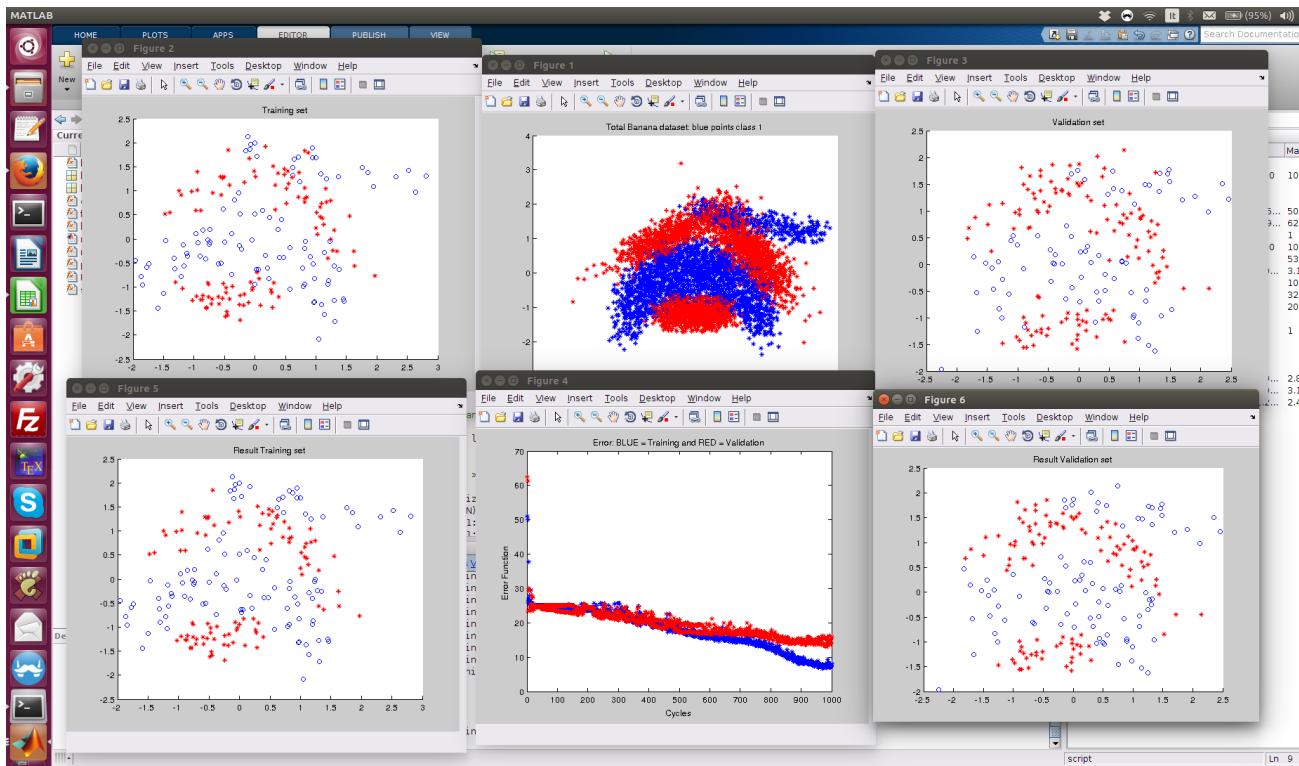


Figure 24: 10 strati interni - 32 nodi per strato

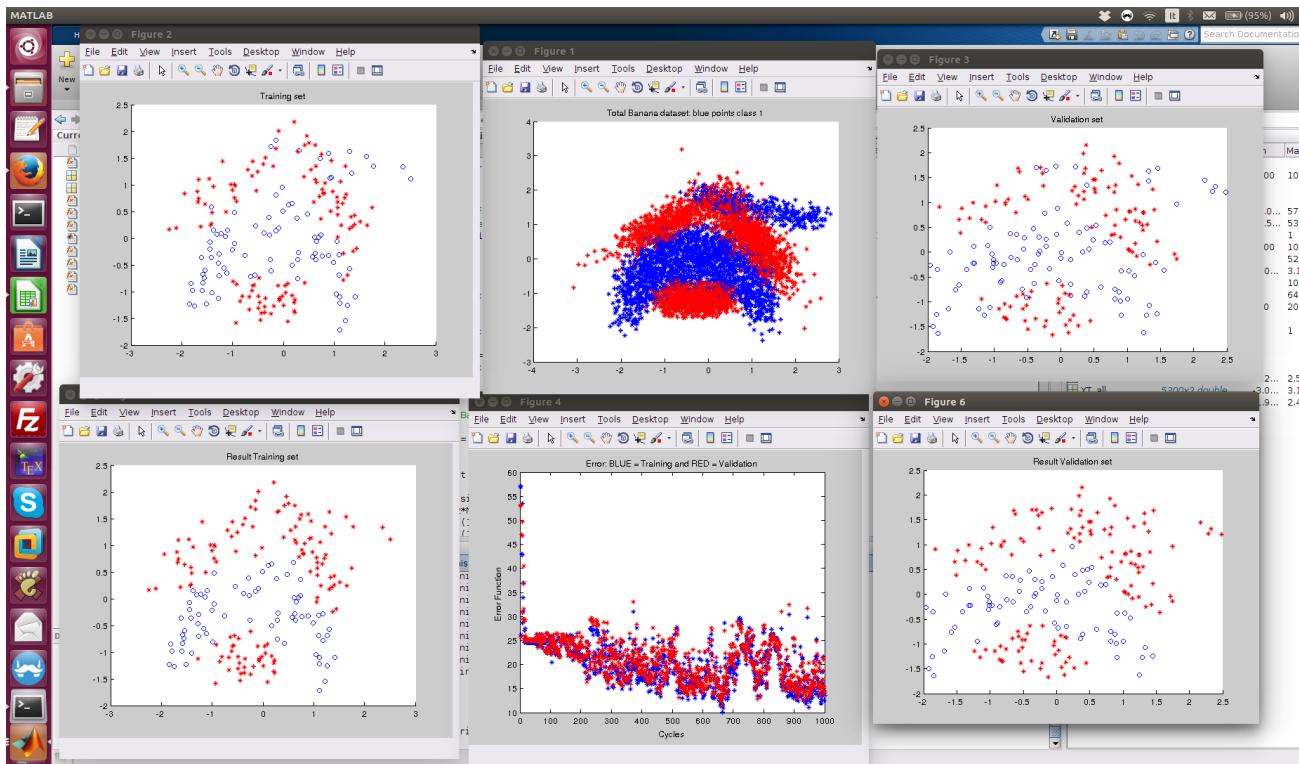


Figure 25: 10 strati interni - 64 nodi per strato

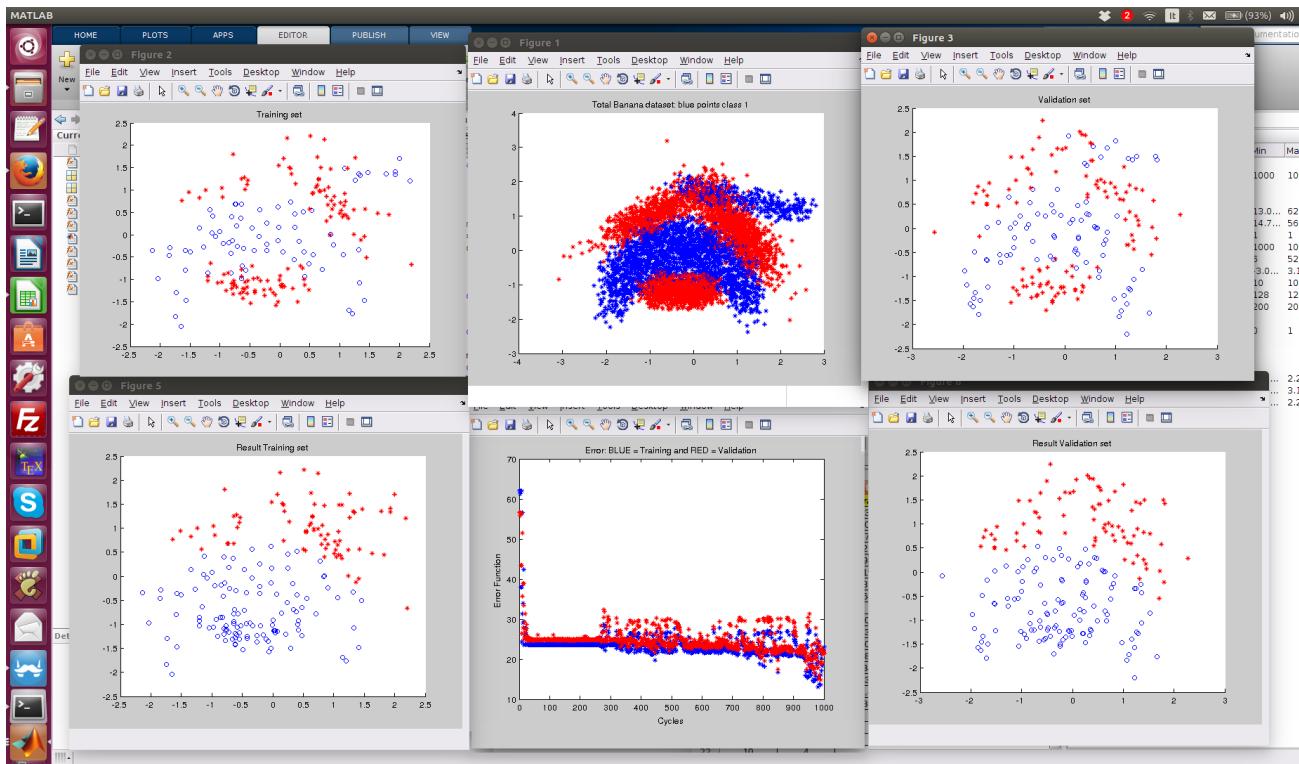


Figure 26: 10 strati interni - 128 nodi per strato

4.2.4 Grafici per 20 strati interni

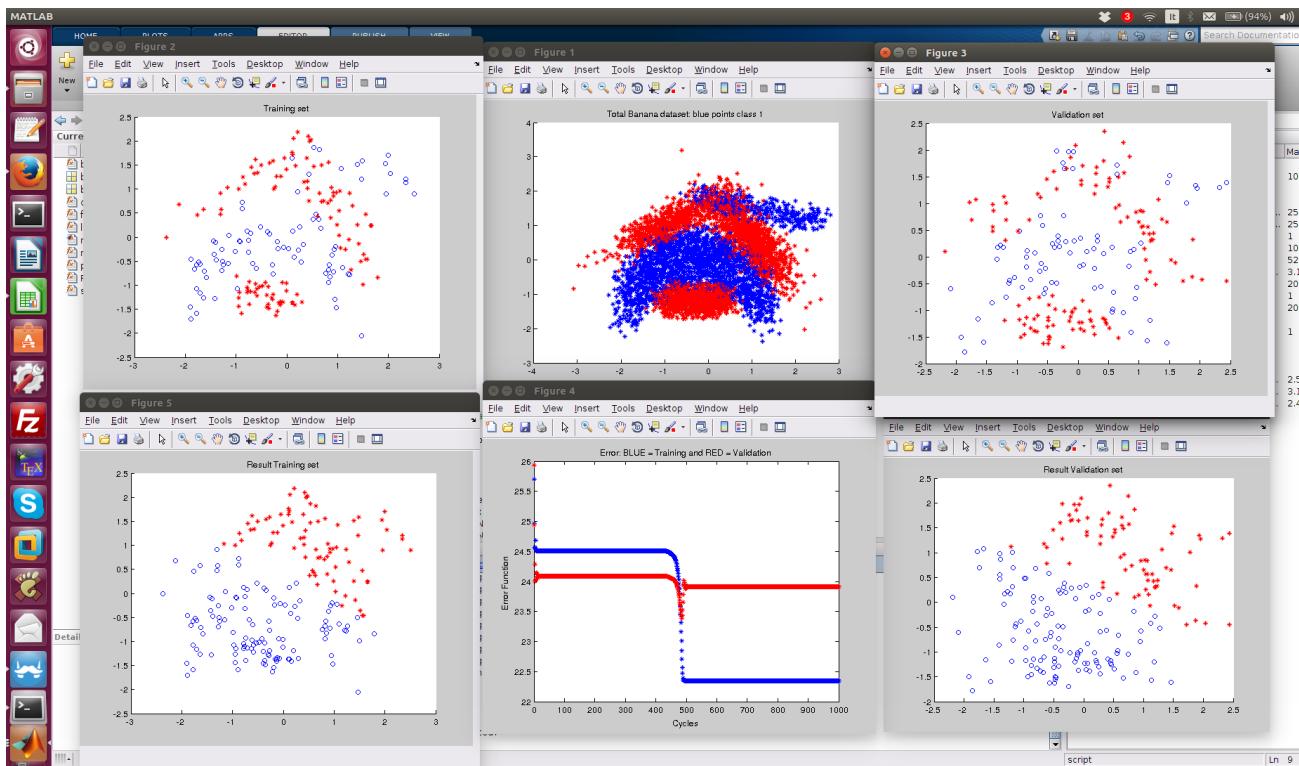


Figure 27: 20 strati interni - 1 nodo per strato

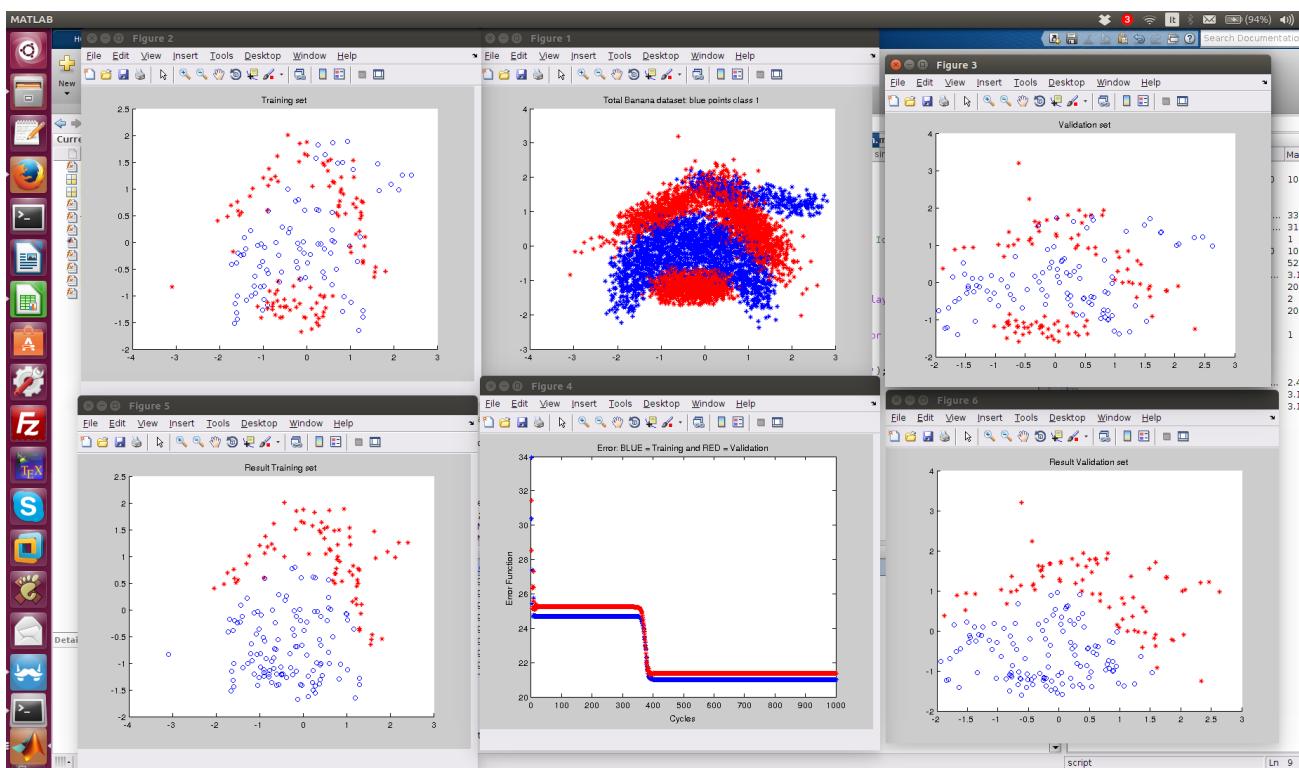


Figure 28: 20 strati interni - 2 nodi per strato

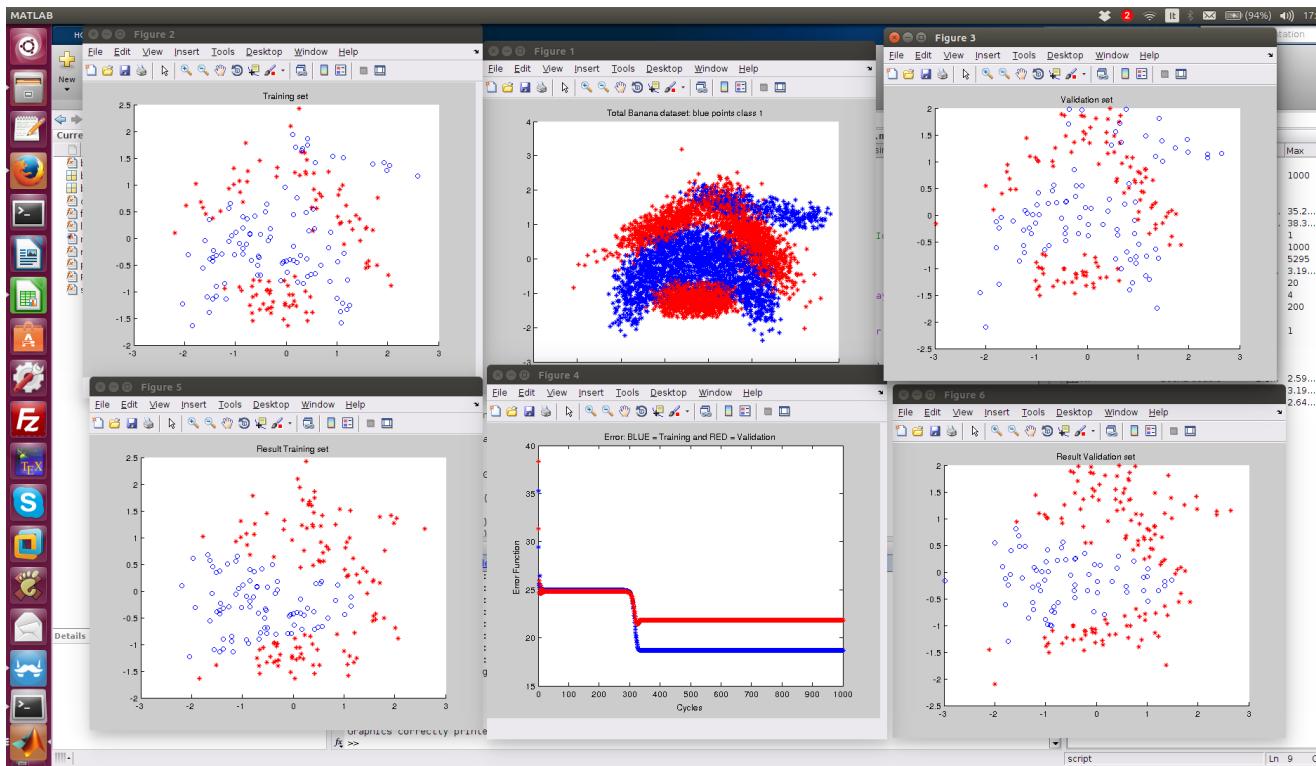


Figure 29: 20 strati interni - 4 nodi per strato

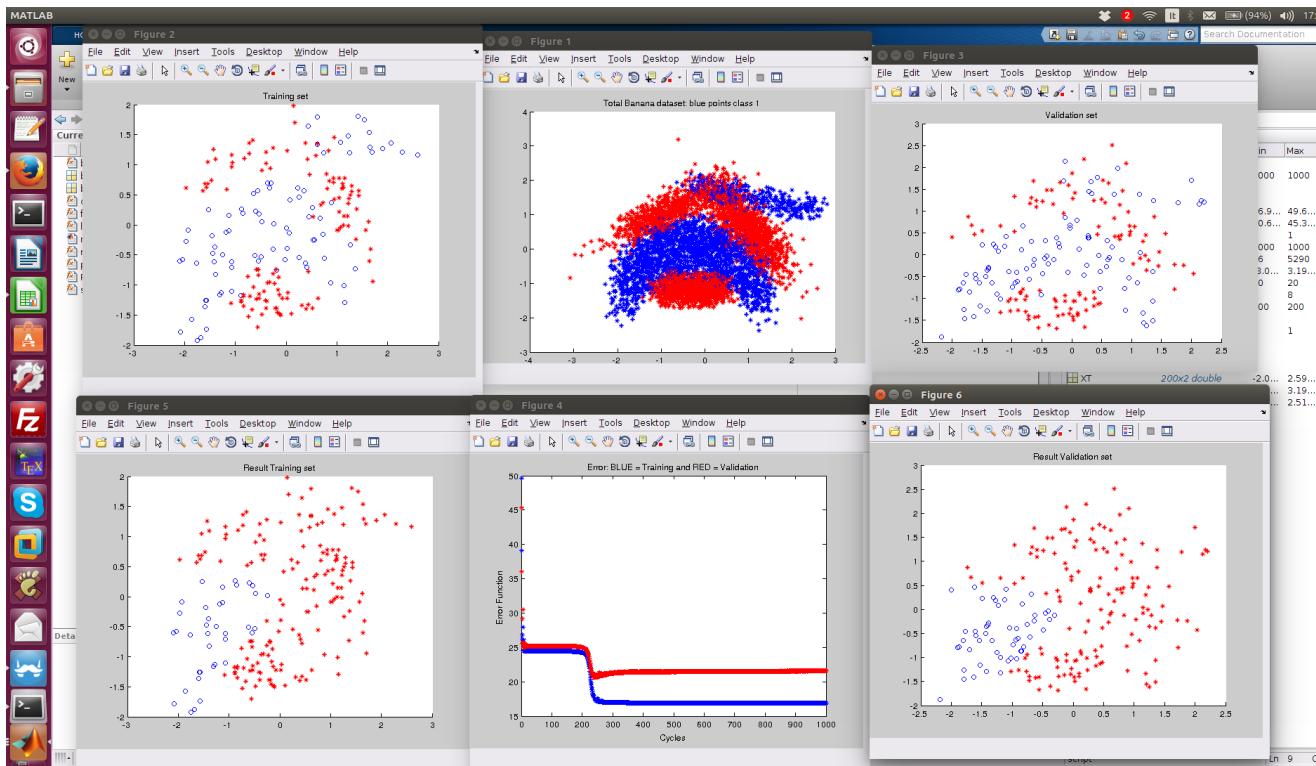


Figure 30: 20 strati interni - 8 nodi per strato

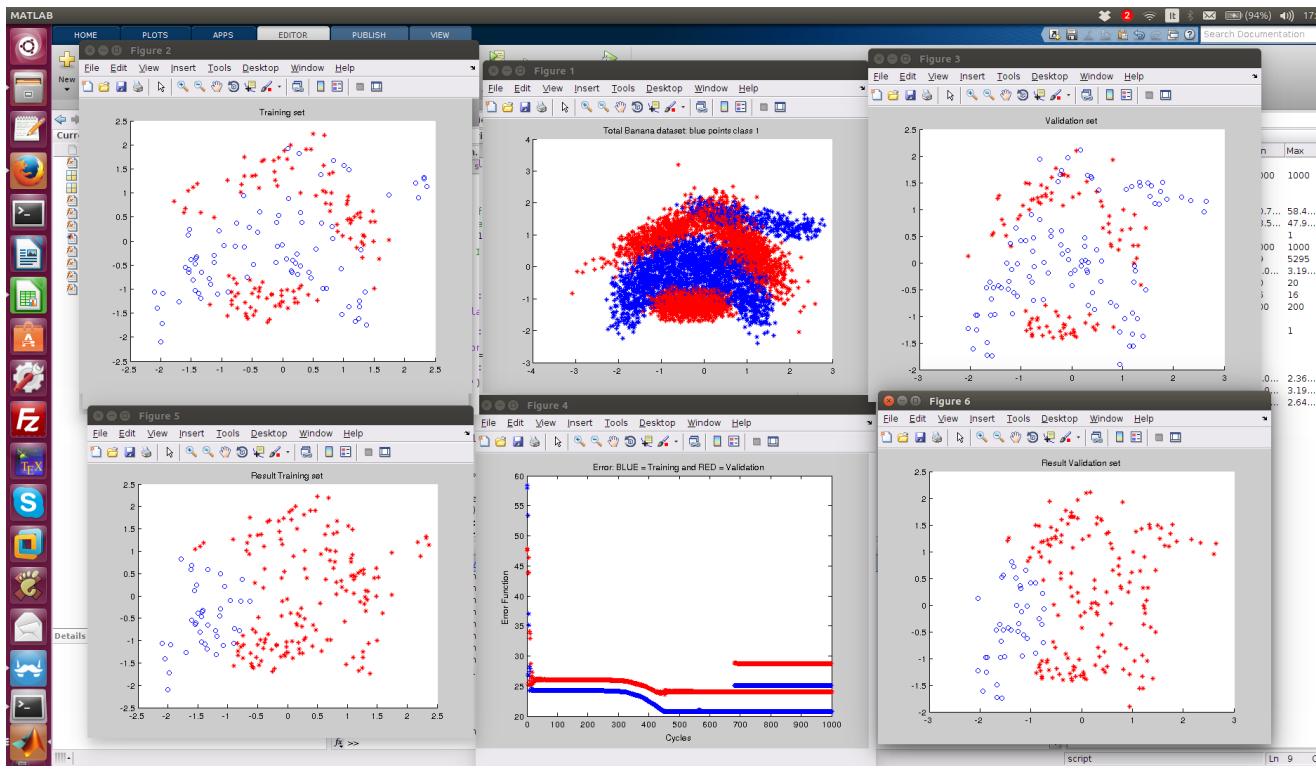


Figure 31: 20 strati interni - 16 nodi per strato

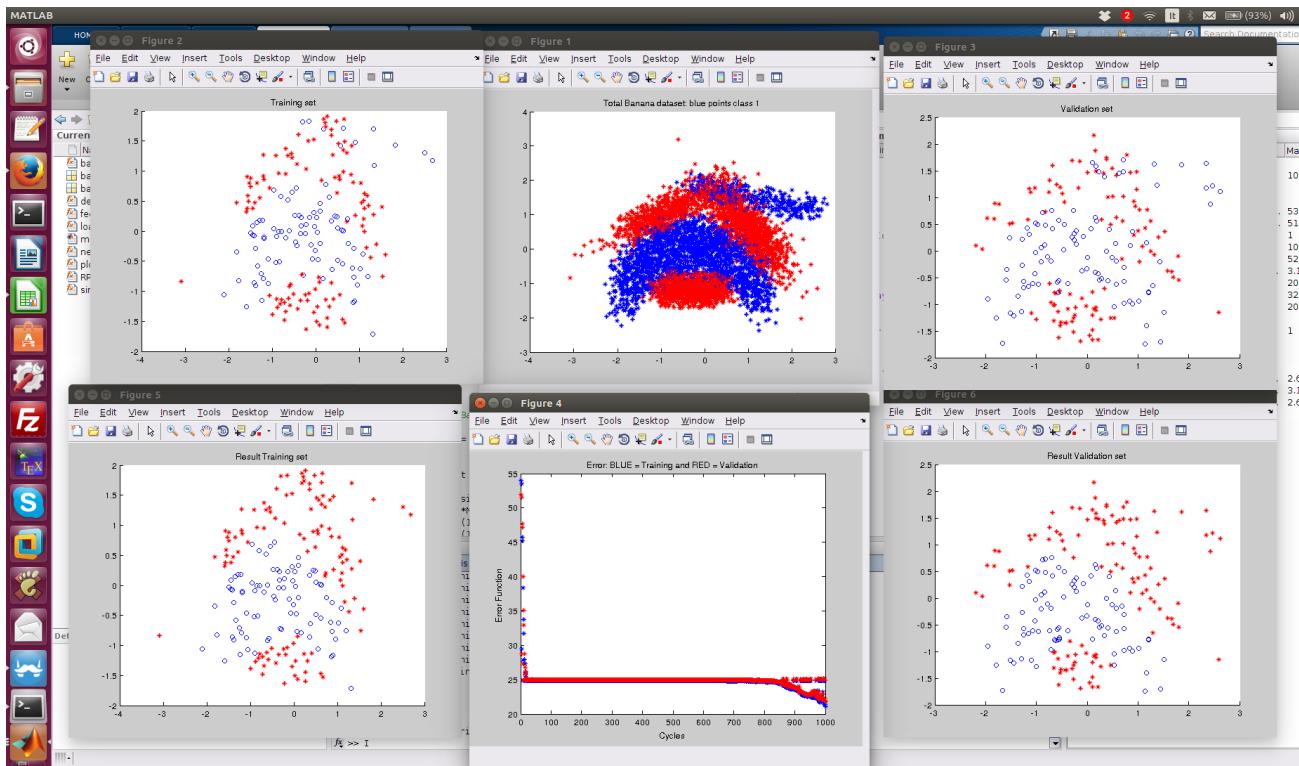


Figure 32: 20 strati interni - 32 nodi per strato

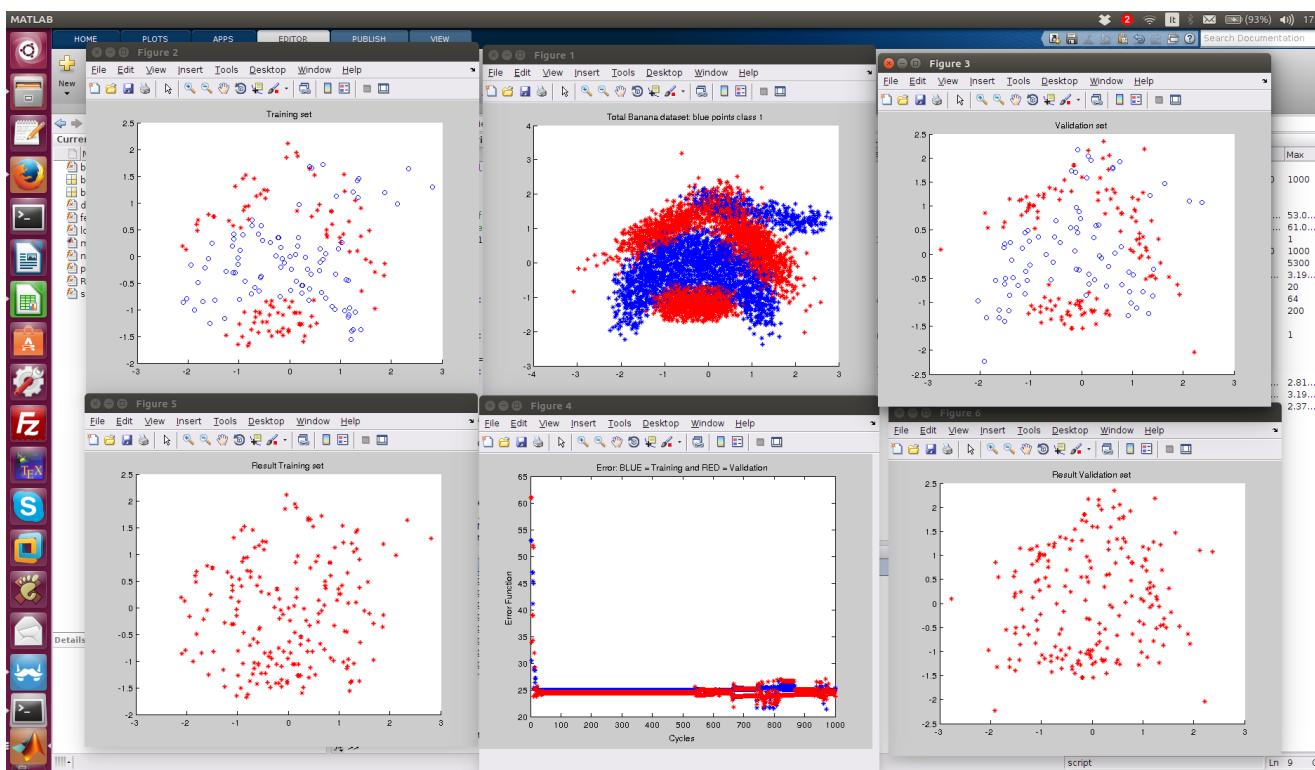


Figure 33: 20 strati interni - 64 nodi per strato

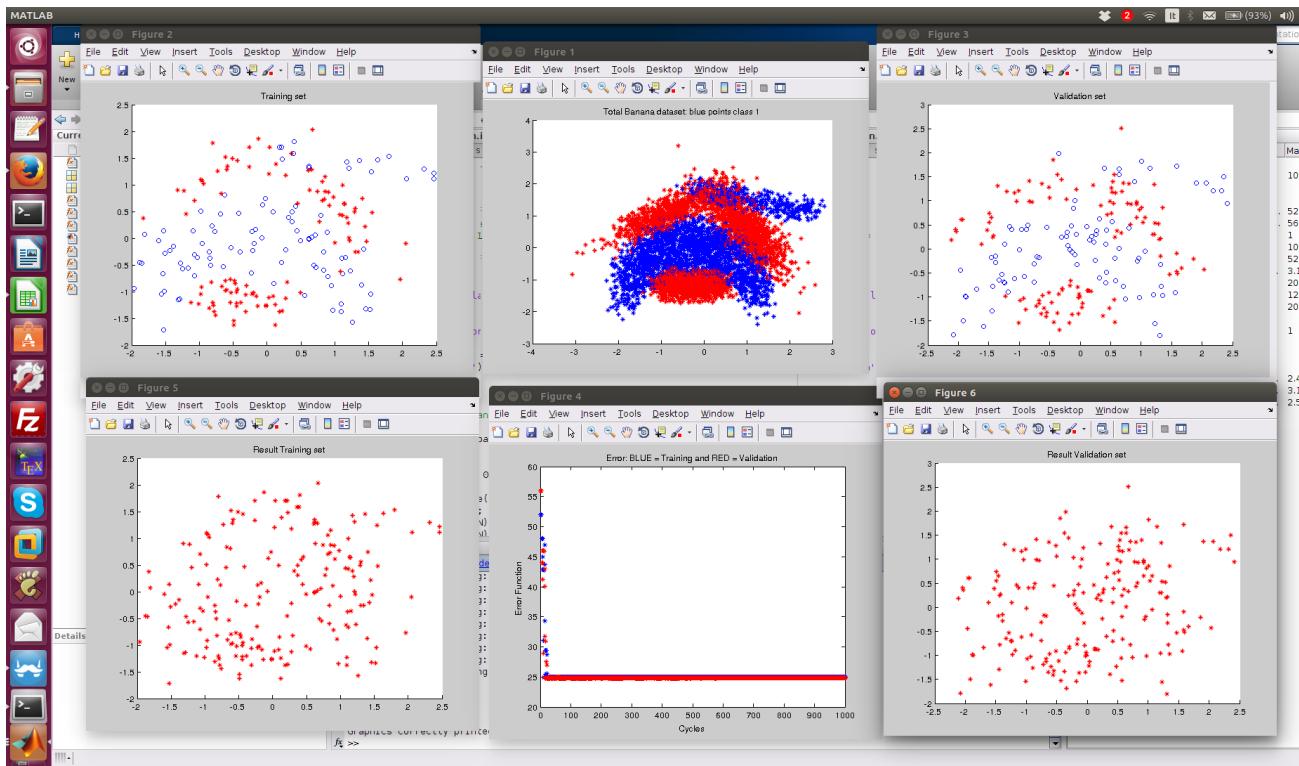


Figure 34: 20 strati interni - 128 nodi per strato

4.3 Risultati ottenuti ed osservazioni

Come già accennato, lo scopo principale è stato quello di studiare l'apprendimento della rete neurale implementata variando il numero dei parametri.

4.3.1 1 strato interno

Il primo ed il più semplice caso di testing è stato quello di considerare una rete neurale con uno strato interno; inserendo 1 nodo o 2 si è notato che la rete non ricostruisce né il training set né il validation set e nel primo caso abbiamo un insieme linearmente separabile di punti; questa è la situazione in cui abbiamo una rete estremamente semplice con grande capacità di generalizzazione, ma con un errore sul training e sul validation sicuramente non accettabile. Il processo di apprendimento vero e proprio inizia con un numero di nodi per strato pari a 4 (con un errore certamente più significativo dei casi successivi); inserendo un maggior numero di nodi l'errore sul training e sul validation tende a decrescere rapidamente e come si è potuto notare dai grafici e dalla tabella, passare da 64 nodi a 128, comporta un aumento dell'errore del training e del validation; ciò è dovuto al fatto che la rete diventa molto più complessa, perdendo quindi la capacità di generalizzazione risultando più sensibile al *rumore* dei dati.

4.3.2 5 strati interni

Analoghe considerazioni valgono per una rete neurale con 5 strati interni. Con 1 o 2 nodi interni per strato, la rete è troppo semplice e non riuscirà ad adattarsi bene ai dati; conseguenza di ciò è che sia il training che il validation non verranno ricostruiti. Con un numero di 4 nodi per strato fino a 16 nodi, abbiamo una situazione di corretto bilanciamento, poiché l'errore sui dati decresce all'aumentare del numero di epoch con la rete neurale che riesce a ricostruire quasi perfettamente i data set.

Aumentando il numero di nodi per strato (da 32 a valori più grandi), osserviamo che la rete è estremamente complessa e si osservano casi di overfitting, con conseguente perdita di generalità della rete neurale. In questi casi, si osserva che l'errore è estremamente basso (consultare la tabella sopra illustrata all'inizio del paragrafo).

4.3.3 10,20 strati interni

Abbiamo effettuato tali test per mettere in luce i casi di overfitting della rete con un numero di strati interni pari a 10 ed a 20; nel primo caso a partire da 32 nodi per strato mentre nel secondo con 16 nodi per strato.

Con 10 strati interni, e con 128 nodi per strato (1280 nodi totali), la rete non ricostruisce il training (come anche il validation) correttamente ciò é dovuto al fatto che la rete oltre ad essere molto complessa ha appreso per troppo tempo portando l'errore del training ad oscillare in un intervallo di valori molto ampio. Tale considerazione vale anche nel caso in cui abbiamo 20 strati interni, inserendo un numero di nodi per strato pari a 64 e 128.

4.3.4 Conclusioni e una possibile soluzione

Dopo aver effettuato questi test, possiamo dire che all'aumentare del numero degli strati della rete neurale, essa andra in **over-training**, cioe che l'errore calcolato nel training set ha subito un leggero incremento a causa di un rumore e, quindi, ´e inutile generare un validation set, visto che il calcolo dell'errore di quest'ultimo sicuramente subira un incremento causato da un rumore. Si pu notare questo fenomeno, confrontando i grafici di errore del training set calcolati dalla rete neurale con un solo strato di nodi interni con quelli calcolati dalla rete neurale con pi strati interni.

Una soluzione possibile per ovviare a difficola di questo tipo ´e limitare la classe delle funzioni che la rete pu apprendere alle funzioni smooth, scegliendo un termine di regolarizzazione opportuno, $\Omega(w)$, che vada a penalizzare le funzioni della rete non smooth.

Il termine di regolarizzazione va necessariamente sommato alla funzione di errore e una delle possibili forme ´e quella denominata *decadimento dei pesi* che di seguito riportiamo per completezza:

$$\Omega(w) = \frac{1}{2} \sum w_i^2$$

Tale termine fa s che la rete assuma piccoli valori per i pesi, in modo tale che i nodi interni della rete possano realizzare funzioni quasi lineari (ovvero funzioni dolci).

5 Codice Sorgente e Riferimenti

main.m

```
close all; clear; clc;

% Parameters

N = 200; %Number of Banana Dataset Values
cycles=1000; %Number of epoches
function_output = 1; %Output function for the neural
    network (1 = Sigmoid, 2 = TanH, 3 = Identity)
m = 1; %Number of hidden layer
n = 32; %Number of nodes for every hidden layer

if m < 0
    fprintf('Error: you cannot create a neural network_
        with a negative number of hidden layer\n');
    return;
elseif n < 0
    fprintf('Error: you cannot create a neural network_
        with a negative number of nodes for every hidden
        layer\n');
    return;
elseif m > 0 && n == 0
    fprintf('Error: you cannot create a neural network_
        with hidden layer without nodes\n');
    return;
end

%Load and show Banana DataSet

[input, target] = loadAndShowBananaDataSet('bananaInput.
    mat', 'bananaTarget.mat');

XT_all = input;
TT_all = (target > 0);

index=randperm(size(XT_all,1));
index=index(1:2*N);
XT=XT_all(index(1:N),:);
```

```

TT=TT_all(index(1:N));
XV=XT_all(index(N+1:2*N),:);
TV=TT_all(index(N+1:2*N));

net = newNet(m,n);

plotDataDS(XT,TT);
title('Training_set');
plotDataDS(XV,TV);
title('Validation_set');

err_training=zeros(1,cycles);
err_validation=zeros(1,cycles);

for i = 1:cycles
    [DW,DB] = backPropagation(net,XT,TT,
        function_output);
    net = RProp(net,DW,DB);
    [y_training,a] = feedForward(net,XT,
        function_output);
    err_training(i)= sum(sum((y_training{size(net.W,2)
        }-TT) .^ 2))/2;
    [y_validation,a] = feedForward(net,XV,
        function_output);
    err_validation(i)= sum(sum((y_validation{size(net.
        W,2)}-TV) .^ 2))/2;
    fprintf('Cycle: %d; Err_Training: %.4f; -
        Err_Validation: %.4f\n',i,err_training(i),
        err_validation(i));
end

fprintf('\\n\\nPrinting_graphics...');

figure;
plot(err_training,'b*');
hold on;
plot(err_validation,'r*');
title('Error: BLUE_= Training_and_RED_= Validation');
xlabel('Cycles');
ylabel('Error_Function');
plotDataDS(XT,y_training{size(net.W,2)}>0.5);
title('Result_Training_set');

```

```

plotDataDS(XV, y_validation{size(net.W,2)}>0.5);
title('Result - Validation - set');
fprintf('\n\nGraphics - correctly - printed!\n');

```

```

function net = newNet (m,n)
d = 2;
c = 1;
if m == 0
    net.W{1} = rand (c,d);
    net.b{1} = rand (1,c);
    net.DW{1} = zeros (c,d);
    net.DB{1} = zeros (1,c);
    net.deltaW{1} = repmat (0.1,c,d);
    net.deltaB{1} = repmat (0.1,1,c);
else
    net.W{1} = rand (n,d);
    net.b{1} = rand (1,n);
    net.DW{1} = zeros (n,d);
    net.DB{1} = zeros (1,n);
    net.deltaW{1} = repmat (0.1,n,d);
    net.deltaB{1} = repmat (0.1,1,n);
    for i = 2:m
        net.W{i} = rand (n,n);
        net.b{i} = rand (1,n);
        net.DW{i} = zeros (n,n);
        net.DB{i} = zeros (1,n);
        net.deltaW{i} = repmat (0.1,n,n);
        net.deltaB{i} = repmat (0.1,1,n);
    end
    net.W{m+1} = rand (c,n);
    net.b{m+1} = rand (1,c);
    net.DW{m+1} = zeros (c,n);
    net.DB{m+1} = zeros (1,c);
    net.deltaW{m+1} = repmat (0.1,c,n);
    net.deltaB{m+1} = repmat (0.1,1,c);
end
end

```

```

function [DW,DB] = backPropagation ( net ,input ,target ,
    fun )
    dim = size( net .W,2) ;
    f = derivativeFunction ;

    [y,a] = feedForward( net ,input ,fun ) ;

    delta_output = (y{dim} - target) .* f{fun}(a{dim}) ;
    delta = delta_output ;
    if dim > 1
        DW{dim} = delta_output ' * y{dim-1} ;
    else
        DW{dim} = delta_output ' * input ;
    end
    DB{dim} = sum(delta_output) ;

    for i=dim-1:-1:1
        delta_hidden = (delta * net .W{i+1}) .* f{fun}(a{i}) ;
        delta = delta_hidden ;
        DB{i} = sum(delta) ;
        if i > 1
            DW{i} = delta_hidden ' * y{i-1} ;
        else
            DW{i} = delta_hidden ' * input ;
        end
    end

end

```

```

function [y,a] = feedForward( net ,x ,fun )
    dim = size( net .W,2) ;
    N = size(x,1) ;
    f = simulationNetwork ;

    a{1} = net .W{1} * x ' ;
    a{1} = a{1}'+repmat( net .b{1},N,1) ;
    y{1} = f{fun}(a{1}) ;

```

```

for i=2:dim
    a{i} = net.W{i} * y{i-1}';
    a{i} = a{i}' + repmat(net.b{i},N,1);
    y{i} = f{fun}(a{i});
end

end

```

```

function fh = simulationNetwork
    fh = localfunctions;
end

function y = computeSig(a)
    y = 1./(1+exp(-a));
end

function y = computeTanH(a)
    y = (exp(a)-exp(-a)) ./ (exp(a)+exp(-a));
end

function y = computeIdentity(a)
    y = a;
end

```

```

function net = RProp (net,DW,DB)
    ETAp = 1.1;
    ETAm = 0.5;
    DeltaMAX = 50;
    DeltaMIN = power(10,-6);
    dim = size(net.W,2);

    for i=1:dim

        if DW{i} .* net.DW{i} > 0
            net.deltaW{i} = min(ETAp .* net.deltaW{i},DeltaMAX);
        end
    end

```

```

        net.W{i} = net.W{i} - sign(DW{i}) .*  

            net.deltaW{i};  

elseif DW{i} .* net.DW{i} < 0  

    net.deltaW{i} = max(ETAm .* net.deltaW{  

        i},DeltaMIN);  

    net.W{i} = net.W{i} - net.deltaW{i};  

    DW{i} = 0;  

else  

    net.W{i} = net.W{i} - sign(DW{i}) .*  

        net.deltaW{i};  

end  

if DB{i} .* net.DB{i} > 0  

    net.deltaB{i} = min(ETAp .* net.deltaB{  

        i},DeltaMAX);  

    net.b{i} = net.b{i} - sign(DB{i}) .*  

        net.deltaB{i};  

elseif DB{i} .* net.DB{i} < 0  

    net.deltaB{i} = max(ETAm .* net.deltaB{  

        i},DeltaMIN);  

    net.b{i} = net.b{i} - net.deltaB{i};  

    DB{i} = 0;  

else  

    net.b{i} = net.b{i} - sign(DB{i}) .*  

        net.deltaB{i};  

end  

    net.DW{i} = DW{i};  

    net.DB{i} = DB{i};  

end  

end

```

```

function plotDataDS(X,Y)
N=size(X,1);
labels=unique(Y);
figure;
hold on;
for i=1:N

```

```

    if Y(i)==labels(1)
        plot(X(i,1),X(i,2), 'r*' );
    else
        plot(X(i,1),X(i,2), 'bo' );
    end
end
return

```

```

function [bananaInput bananaTarget]=
loadAndShowBananaDataSet(bananaInputFile ,
bananaTargetFile)
%function [bananaInput bananaTarget]=
%loadAndShowBananaDataSet(bananaInputFile ,
%bananaTargetFile)

input=load(bananaInputFile);
bananaInput=input.bananaInput;
target=load(bananaTargetFile);
bananaTarget=target.bananaTarget;

figure;
hold on;
for i=1:size(bananaInput,1)
    if bananaTarget(i)==-1
        bananaTarget(i)=0;
        plot(bananaInput(i,1),bananaInput(i,2), 'r*' ,
);
    else
        plot(bananaInput(i,1),bananaInput(i,2), 'b*' ,
);
    end
end
title('Total_Banana_dataset:_blue_points_class_1');
end

```

```

function fh = derivativeFunction
fh = localfunctions;

```

```
end

function y = dSig(x)
    f = simulationNetwork;
    y = f{1}(x) .* f{1}(1-x);
end

function y = dTanh(x)
    f = simulationNetwork;
    y = 1 - (f{2}(x) .^ 2);
end

function y = dIdentity(x)
    y = ones(size(x));
end
```

Riferimenti:

1. Christopher M. Bishop. *Neural Network for pattern recognition*. Clarendon Press, 1995
2. Appunti del corso di Machine Learning mod.B
3. User Manual Matlab: <http://it.mathworks.com/help/matlab/>
4. Latex Documentation: <https://latex-project.org/guides/>