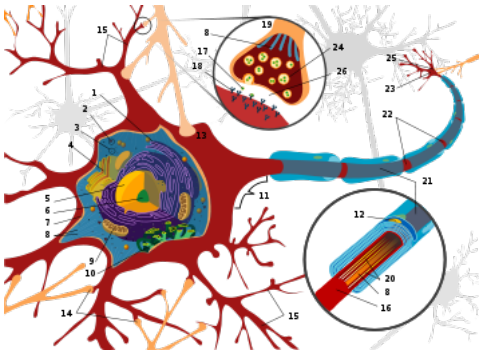
Università degli Studi di Firenze

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Modelli per i pattern migratori della *Cerithidea Decollata*

Autore:
Tommaso Papini
tommy39@gmail.com



29 marzo 2013

Indice

Elenco delle figure

Elenco dei codici

Capitolo 1

Introduzione al problema

Da diversi anni ormai viene studiato da biologi di tutto il mondo l'affascinante comportamento della *Cerithidea decollata*, comunemente chiamata **chiocciola delle mangrovie** (dall'inglese, *truncated mangrove snail*).
Questo particolare tipo di chiocciole ha sviluppato la capacità, a causa dell'ambiente nel quale vive, di prevedere l'altezza della marea a distanza di diverse ore, così da poter salire sul tronco di un albero e non rimanere sott'acqua.

1.1 Marea

La marea è un moto periodico di masse d'acqua che si innalzano e si abbassano (rispettivamente alta e bassa marea).
Il fenomeno delle maree è provocato, principalmente, dalla combinazione di due fattori:

- l'attrazione esercitata sulla Terra da parte della Luna e del Sole;
- la forza centrifuga della rotazione del sistema Terra-Luna attorno al proprio centro di massa.

L'altezza raggiunta dalla marea e la sua frequenza dipendono strettamente dal luogo e dal periodo dell'anno. In generale si hanno due picchi di alta marea e due riflussi di bassa marea ogni giorno.
Per gli studi effettuati in questo documento è stata utilizzata una formula in funzione del tempo (espresso in ore, come riportato di seguito) che approssima la funzione di marea:

$$f(t) = Aa\cos\left(\frac{2\pi t}{Ta}\right) + Ab\cos\left(\frac{2\pi t}{Tb}\right) - D\sin\left(\frac{\pi t}{Tm} - \frac{\pi}{4}\right)$$

assegnando ai parametri presenti i seguenti valori

- $Aa = 1.06$

- $Ab = 0.65$
- $D = 0.15$
- $Ta = 12.42$
- $Tb = 12$
- $Tm = 12.21$

Con questa approssimazione la marea risulta avere un periodo totale di circa 28 giorni.

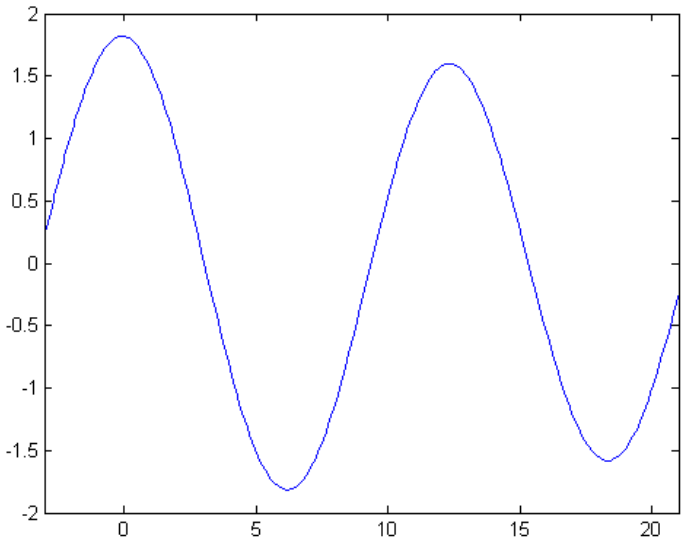


Figura 1.1: La funzione di marea in una giornata

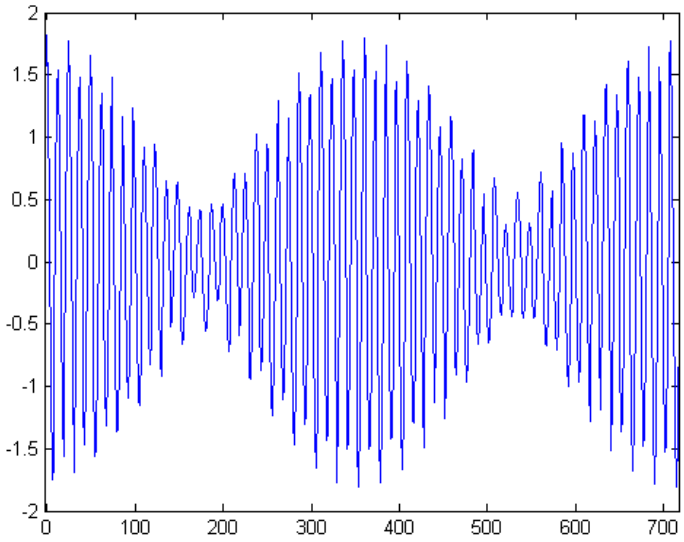


Figura 1.2: La funzione di marea in un mese

1.2 Cerithidea decollata

La *Cerithidea decollata*, comunemente chiamata **chiocciola delle mangrovie** (dall'inglese *truncated mangrove snail*), è una specie di chiocciola di mare, ovvero un mollusco gasteropodo della famiglia delle *Potamididae*.

Gli esemplari adulti hanno una conchiglia lunga circa 3 cm composta da 5 spire, solitamente dalla punta spezzata, con circa 20 nervature per ogni spira.
Queste chiocciole proliferano nelle mangrovie costiere, in particolare nella parte occidentale di Kenia, Tanzania, Mozambico, Sud Africa e Madagascar.
La *Cerithidea decollata* si nutre principalmente di detriti organici e di alghe portate dalla marea. Per questo vivono principalmente in piani mesolitorali (zone del litorale che dipendono dalla marea) che presentano due picchi di alta marea e due riflussi di bassa marea ogni giorno.
Ogni volta che la marea si abbassa, queste chiocciole scendono al suolo per potersi nutrire. Quindi, una o due ore prima dell'innalzamento della marea, iniziano a salire sul tronco degli alberi, per poi fermarsi dai 20 ai 60 cm sopra al livello della marea futura, aspettando che questa arrivi e si abbassi di nuovo.
Questo comportamento permette loro di evitare sfavorevoli effetti fisiologici legati

all'immersione e sfuggire a predatori marini come granchi.
È stato scoperto che queste chiocciole calcolano la propria altitudine misurando l'energia impiegata per salire: infatti, se caricate artificialmente con dei pesi, queste chiocciole salgono molto meno, credendo di essere già arrivate a destinazione. Non è ancora chiaro,tuttavia, come queste chiocciole possano prevedere l'imminente altezza di marea: la differenza del peso corporeo, dovuto alle fluttuazioni gravitazionali che causano la marea, risulta essere troppo piccola per essere rilevata da un organismo di quelle dimensioni. Si pensa che abbiano una sorta di orologio interno, combinato con l'identificazione di un qualche tipo di evento precedente alla marea, come il rilascio di acido solfidrico dal terreno, il riconoscimento degli infrasuoni prodotti dalle onde o anche le vibrazioni causate dalla cosiddetta *marea di terra*.

In questo studio non ci soffermeremo su come possa la *Cerithidea decollata* predire effettivamente l'altezza delle maree, ma solo su quale sia la minima rete neurale necessaria per eseguire tale predizione, sia in un ambiente privo di errori che in un ambiente in cui i dati sensoriali possono essere soggetti ad errori.
Quindi, per semplicità, assumeremo che queste chiocciole siano in grado di osservare, ad intervalli prefissati, il livello attuale del mare ed in base alle informazioni ottenute predire il livello della marea dopo un numero fissato di ore.

Figura 1.3: Un gruppo di *Cerithidea decollata* aggrappate al tronco di un albero, in attesa della marea

Capitolo 2

Reti neurali

Per la simulazione della rete neurale della chiocciola, relativa alla sola predizione della marea, ed il conseguente studio della complessità neurale necessaria alla chiocciola per poter eseguire tale predizione con un errore accettabile, si è scelto di iniziare con una **rete lineare di Widrow**.
Con questa tipologia, molto semplice, di rete neurale si sono ottenuti, come vedremo in maggior dettaglio nei due capitoli successivi, ottimi risultati. Quindi, per quanto concerne la *predizione semplice* e *con errori*, è risultato innecessario avventurarsi nell'utilizzo di reti neurali più complesse.
Tuttavia, per lo studio del problema relativo alla predizione del prossimo picco di alta marea (Capitolo 5) una semplice rete lineare di Widrow non è risultata sufficiente e si è quindi utilizzato un **Multilayer Perceptron**, ovvero una rete di tipo *Feed-Forward* con un certo numero di neuroni aggiuntivi nello strato nascosto.

2.1 Rete lineare di Widrow

La rete di Widrow-Hoff rappresenta una classe di filtri adattivi denominata **ADALINE** (in un primo momento **Ada**ptive **Li**near **Ne**uron e successivamente **Ada**ptive **Lin**ear **E**lement).
La rete è composta da un unico strato di output ed utilizza una funzione di sommatoria ed una funzione di apprendimento. Ogni input x della rete ha un peso w associato, quindi l'output corrispondente ad un set di input $\{x_1, x_2, ..., x_n\}$ è calcolato semplicemente con la media pesata

$$y = \sum_{j=1}^{n} x_j w_j$$

La struttura risulta quindi molto simile a quella del *Perceptron semplice* e, più in generale, al modello neurale di McCulloch-Pitts, tuttavia differisce per la funzione di apprendimento: l'algoritmo di apprendimento di Widrow-Hoff (anche detto $\alpha - LMS$) aggiorna, ad ogni iterazione, i pesi in base alla media pesata degli input. Indicando con t_k l'output atteso (target) al passo k, y_k l'output ottenuto

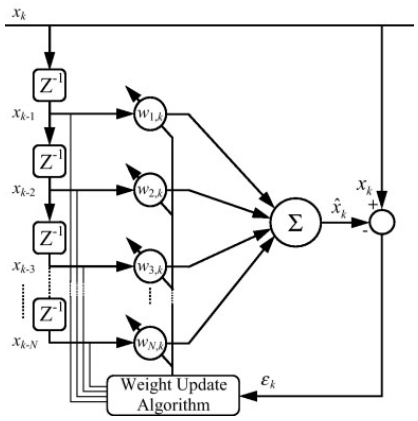


Figura 2.1: Struttura generica della rete ADALINE

al passo k, x_k e w_k, rispettivamente, il vettore di input ed il vettore di pesi al passo k e α un parametro che regola la velocità di convergenza del metodo (detto anche **learning rate**), allora l'algoritmo di apprendimento risulta

$$w_{k+1} = w_k + \alpha(t_k - y_k)\frac{x_k}{|x_k|^2}$$

che, quindi, prescinde dall'appartenenza ad una classe come il Perceptron standard (ricordiamo che la regola di apprendimento del perceptron doveva innanzitutto capire come doveva essere classificato l'output e come era effettivamente classificato, in 1 o -1, e quindi aggiornare i pesi di conseguenza) ed inoltre normalizza l'input in modo automatico.
Quindi risulta che la rete lineare di Widrow converge più velocemente del Perceptron ed inoltre riesce a risolvere problemi di calcolo che il Perceptron, che si limita ad una semplice classificazione binaria, non può risolvere.

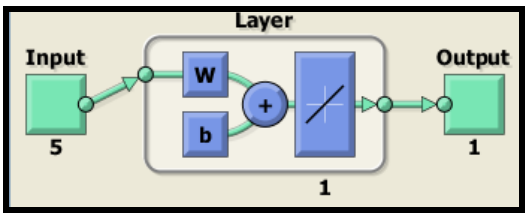


Figura 2.2: Struttura della rete utilizzata per la simulazione in MatLab

La rete utilizzata in MatLab per questo studio corrisponde al comando *newlin*, che prende in input una matrice NxM di M vettori di input (per l'addestramento), ognuno dei quali composto da N elementi, una matrice KxM di M output target, ognuno di K elementi, un vettore di ritardi, non utilizzato per questo progetto, ed il parametro di *learning rate*, impostato a **0.01**.

La rete è stata addestrata su **20000** input, per la predizione semplice e con errori, e **10000** input, per la predizione della prossima alta marea; ognuno dei quali composto da **5 orari** a distanza prefissata, distribuiti casualmente nell'arco di **28 giorni**.

2.2 Multilayer Perceptron

Il Multilayer Perceptron rappresenta, come si può intuire, la naturale evoluzione del Perceptron standard. Esso fu ideato da Rumelhart et al. e consiste nell'aggiungere al Perceptron semplice uno o più **strati intermedi** (o **nascosti**) di neuroni tra lo strato di input e quello di output.
Il grande vantaggio del Multilayer Perceptron rispetto al Perceptron standard è di poter risolvere anche problemi non lineari e quindi di poter essere applicato a quei problemi che il Perceptron semplice non potrebbe risolvere (come ad esempio il problema dello XOR): in altre parole, il Multilayer Perceptron riuscì a superare le limitazioni che Minsky e Papert individuarono nello studio del Perceptron lineare.
Ogni strato nascosto può utilizzare una funzione di trasferimento diversa dalla semplice Θ, detta *theta di Heaviside*, che assume soltanto i valori binari 0 ed 1: ad esempio una scelta molto comune è una particolare *funzione sigmoidale* detta **funzione logistica**, per la quale vale

$$logsig(x) = \frac{1 + \tanh(x)}{2}$$

con $logsig(x)$ funzione sigmoidale rispetto ad x e $\tanh(x)$ tangente iperbolica rispetto ad x.

Il metodo di apprendimento del Multilayer Perceptron è una regola chiamata **Backpropagation** in quanto, durante l'apprendimento, per ogni pattern di input esaminato (cioè ogni **epoca**), i pesi vengono aggiornati dallo strato di output verso lo strato di input. Al contrario, il metodo utilizzato dalla rete per produrre un output viene detto di **Feed-Forward** in quanto i segnali passano dallo stato di input verso quello di output.

In MatLab questo tipo di rete viene creato richiamando la funzione *newff*, la quale richiede come parametri una matrice di input, una matrice di output, una vettore con indicato il numero di neuroni presenti negli (eventuali) strati nascosti, una lista delle funzioni di trasferimento per ogni strato (dal primo strato nascosto fino allo strato di input) ed, eventualmente, la funzione di apprendimento.
Nelle simulazioni effettuate con questa rete si è utilizzato un Multilayer Perceptron con uno strato nascosto composto da **6 neuroni**; le funzioni di trasferimento utilizzate sono **logsig**, ovvero la funzione logistica, per lo strato nascosto, e **purelin**, cioè la funzione di trasferimento lineare, per lo strato di output. La funzione di apprendimento utilizzata in questo caso è **trainlm**, che corrisponde alla regola di Backpropagation di Levenberg-Marquardt.

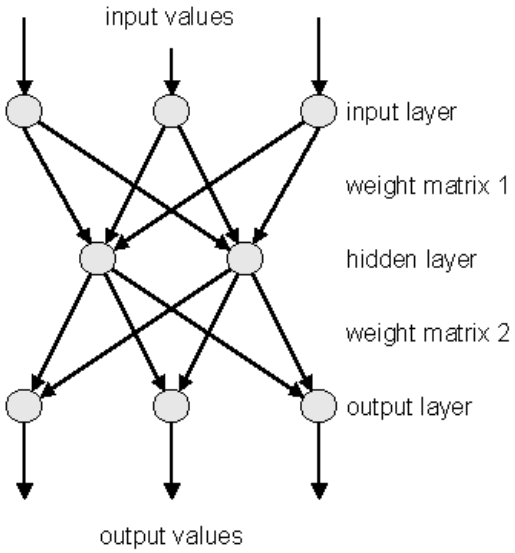


Figura 2.3: Multilayer Perceptron con 3 input, 3 output e 2 neuroni nello strato nascosto

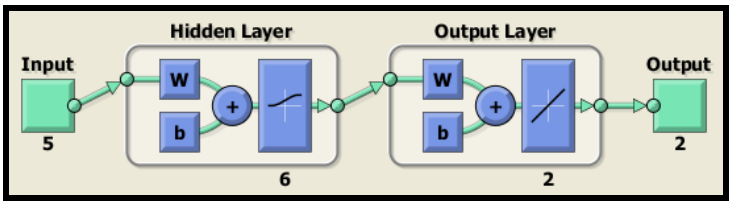


Figura 2.4: Struttura del Multilayer Perceptron utilizzato in MatLab

L'apprendimento della rete è stato effettuato su **10000** gruppi di orari, ognuno composto da **5 ore** a distanza di **1 ora**.

Capitolo 3

Predizione semplice

Come già accennato nell'introduzione, per semplicità di studio, nonché per ovvi motivi, quali l'effettiva impossibilità di uno studio approfondito del fenomeno, si è ipotizzato che le chiocciole oggetto di studio siano in grado di osservare un certo numero di altezze di marea, ognuna a distanza di un numero prefissato di ore, e quindi prevedere l'altezza della marea dopo un numero arbitrario di ore.

Per **predizione semplice** si intende la simulazione della predizione dell'altezza della marea futura da parte della *Cerithidea decollata* in un ambiente privo di errori sensoriali, ovvero una situazione in cui la chiocciola è in grado di osservare le altezze di marea con precisione assoluta e di valutare gli intervalli temporali con altrettanta precisione.
Per il codice MatLab relativo alla predizione semplice si veda il Codice 1 a pagina 29, con l'accortezza di impostare i valori di rumore sulla marea ($minTN$ e $maxTN$) e sulle ore ($minHN$ e $maxHN$) tutti a zero.

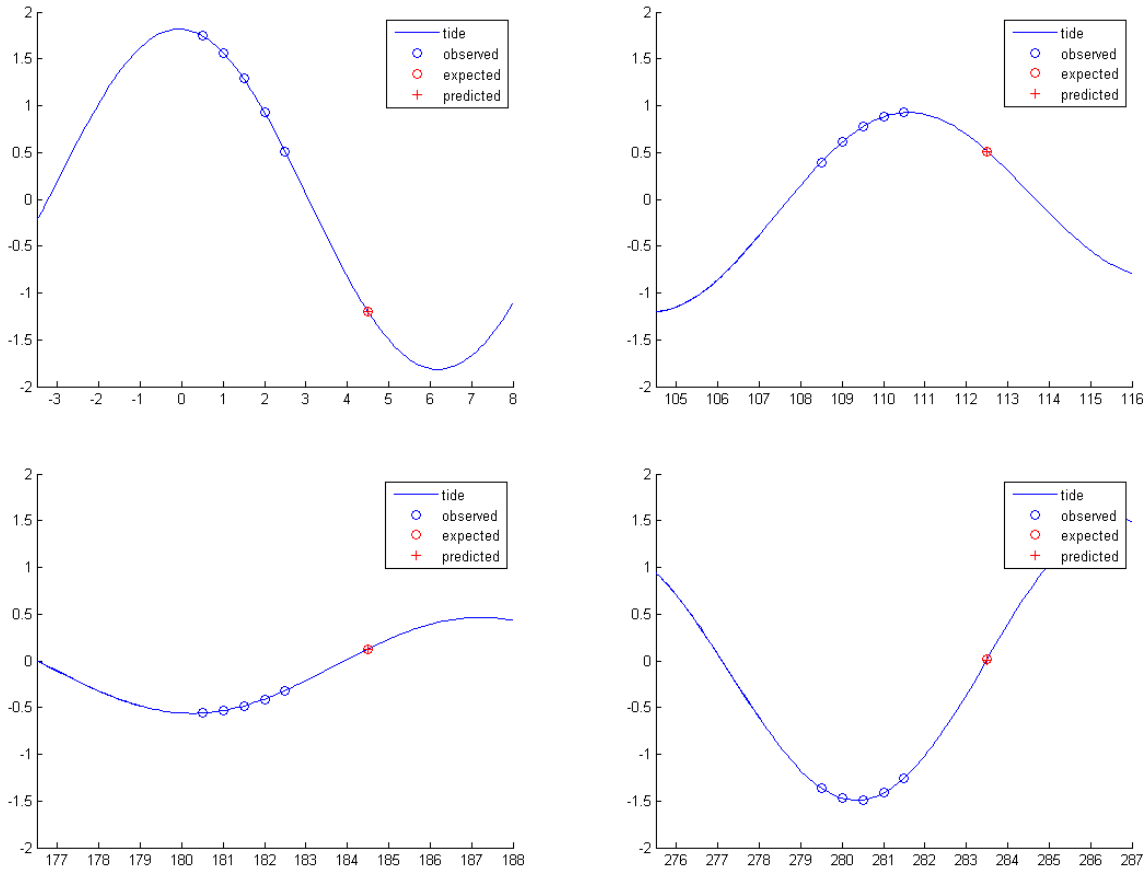
3.1 Predizione semplice a due ore di distanza

Per avvicinarsi il più possibile al comportamento della *Cerithidea decollata* le prime simulazioni effettuate sono state quelle in cui la chiocciola osserva 5 altezze di marea, ognuna a distanza di mezz'ora l'una da l'altra, per effettuare quindi la predizione di come sarà l'altezza di marea due ore dopo.

Da queste simulazioni si sono ottenuti ottimi risultati: lo scarto quadratico medio raggiunto durante l'addestramento su 28 giorni è dell'ordine di 10^{-5}, il quale, considerando che la funzione di marea assume valori tra circa -2 e 2, e che quindi il massimo errore assoluto possibile è dell'ordine di 4, significa che in media le chiocciole sbagliano l'altezza della marea a due ore di distanza di circa lo **0,0006** %.

Per ottenere queste simulazioni basta assegnare alle variabili *int* e *forecast* nel

Codice 1 i valori, rispettivamente, di *0.5* e *2*.
Nei grafici relativi a queste simulazioni, che proponiamo di seguito la linea continua di colore blu indica la funzione di marea esatta, i punti cerchiati in blu indicano le altezze di marea osservate prima della previsione e la croce ed il cerchio rossi indicano, rispettivamente, l'altezza futura di marea prevista dalla chiocciola e quella effettiva.



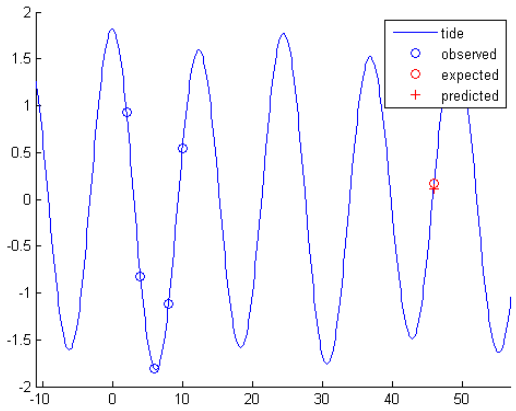
3.2 Predizione semplice massima

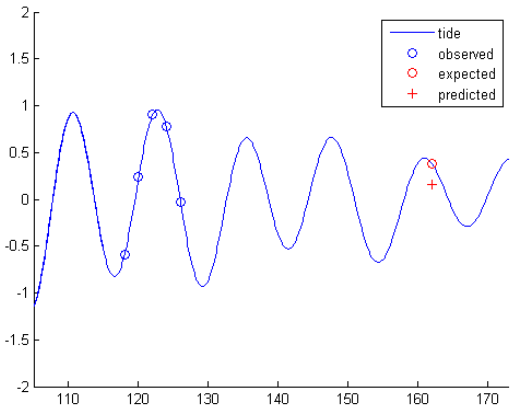
Per uno studio più approfondito dei limiti computazionali della rete lineare che simula il calcolo dell'altezza della marea si è provato quindi a determinare il numero massimo di ore di distanza dalla previsione, mantenendo lo scarto quadratico medio inferiore all'*1%*.

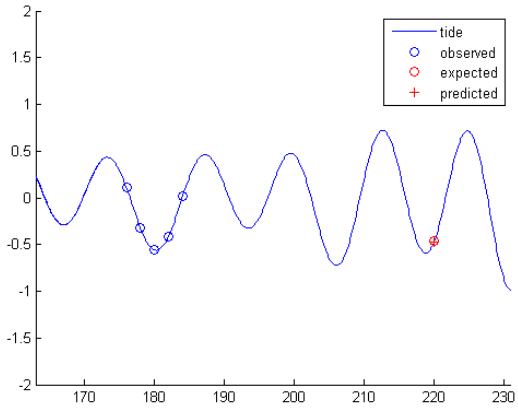
Il risultato ottenuto, in via del tutto sperimentale, è di **36 ore** come intervallo massimo tra l'ultima marea osservata e la previsione che si vuole effettuare (osservando la marea ogni 2 ore) con un errore medio sull'altezza di marea inferiore all'**1%**. Si noti che l'errore dell'*1%* risulta essere un valore molto basso, mentre *36 ore* disponibili alla chiocciola per mettersi in salvo sul tronco di un

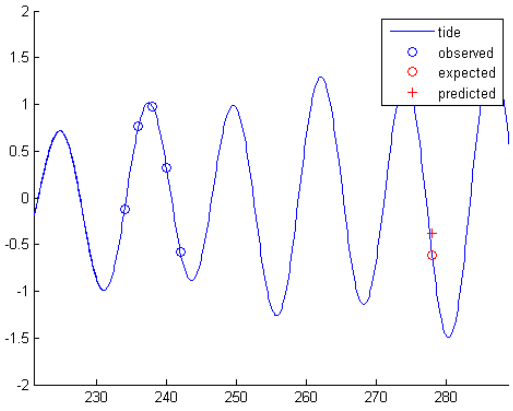
albero sono più che sufficienti, per non dire eccessive: per errori più alti si può facilmente raggiungere un numero di ore per la previsione molto più alto, anche se inutile a fini pratici.

Per riprodurre queste simulazioni basta impostare le variabili *int* e *forecast* a *2* e *36* rispettivamente (Codice 1).
Di seguito alcuni grafici relativi a queste simulazioni:









Capitolo 4

Predizione con errori

Si è visto nel paragrafo precedente che la simulazione semplice, cioè priva di errori, ha dato ottimi risultati.
Studiamo adesso la previsione dell'altezza di marea aggiungendo degli errori nella percezione che le chiocciole hanno del mondo che le circonda.

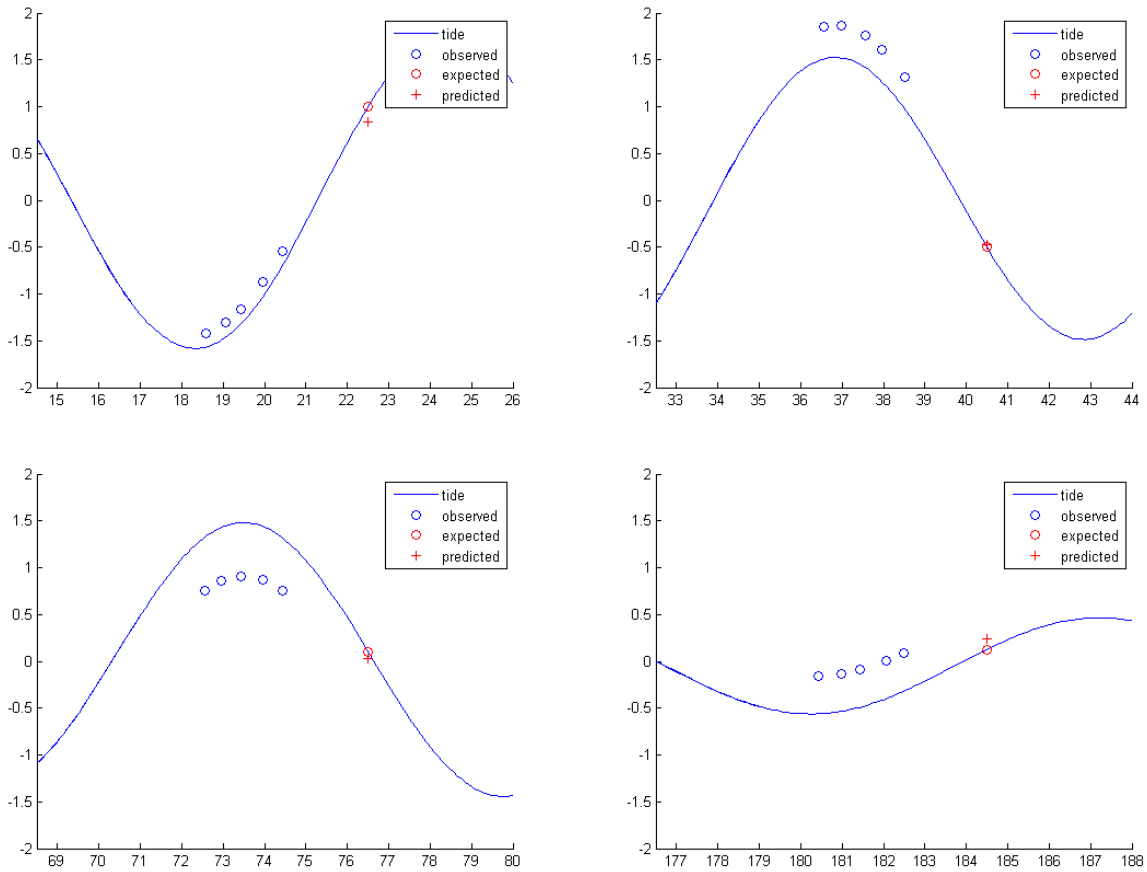
Gli errori introdotti saranno quindi di due tipi: errori sulla valutazione dell'altezza della marea osservata ed errori sulla valutazione del tempo trascorso tra un'osservazione e l'altra.
È plausibile pensare che queste chiocciole non abbiano una misura perfetta delle altezze di marea o del trascorrere del tempo e che facciano affidamento su una stima, più o meno accurata, di queste.
Nel file relativo all'addestramento e al testing della rete neurale (Codice 1), e nelle simulazioni che vedremo in questo capitolo, si sono utilizzati errori relativi all'altezza di marea di massimo 0.6 (in valore assoluto, quindi tra -0.6 e 0.6), che rappresenta quindi un errore massimo del **15%** sull'altezza di marea osservata, ed un errore relativo agli intervalli di tempo di massimo circa il **16%**. Per quanto riguarda gli errori sulle altezze di marea, si è deciso di assegnare lo stesso errore a tutte e 5 le osservazioni successive in quanto si è supposto che, anche se la chiocciola non è sicuramente in grado di determinare l'altezza della marea con precisione assoluta, le risulterà certamente più facile capire se la marea è alzata o abbassata da un'osservazione all'altra.
Nel caso si volessero cambiare tali parametri basta cambiare il valore di $minTN$ e $maxTN$, per gli errori di altezza, e $minHN$ e $maxHN$, per gli errori di tempo.

4.1 Predizione con errori a due ore di distanza

Come nel capitolo precedente, si cercherà, innanzitutto, di valutare con che errore le chiocciole riescono a predire la marea a due ore di distanza con 5 osservazioni a mezz'ora di distanza l'una dall'altra.

Lo scarto quadratico medio ottenuto con questo tipo di simulazione è circa **0.124**, che corrisponde al **3.1%** sull'altezza di marea. Questo ci dice subito che la predizione con errori di valutazione risulta essere più complessa e più difficoltosa da portare a termine.
Se confrontiamo questo risultato con quello ottenuto in una previsione semplice a due ore di distanza (errore medio inferiore allo **0,0006 %**) notiamo che l'errore è aumentato di circa un fattore **5000**. Certo un errore medio del *3.1%* rimane pur sempre un risultato ampiamente accettabile, ma questo ci fa capire come piccoli errori sui dati sensoriali possano portare ad un grandissimo aumento di complessità per la valutazione della marea futura, soprattutto per il carattere casuale che presentano tali errori: un giorno la chiocciola può sbagliarsi e vedere la marea alta 0.5 in più del normale, mentre il giorno dopo può valutarne l'altezza in modo esatto.

Per riprodurre questa simulazione basta eseguire il file Codice 1 senza apportare alcuna modifica ai parametri presenti. In questo caso un errore di massimo 5 minuti corrisponde a circa il *16%* quando l'intervallo tra due osservazioni è di mezz'ora.
Di seguito alcuni grafici:

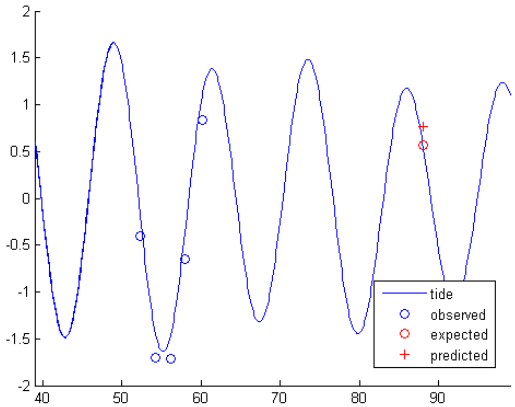


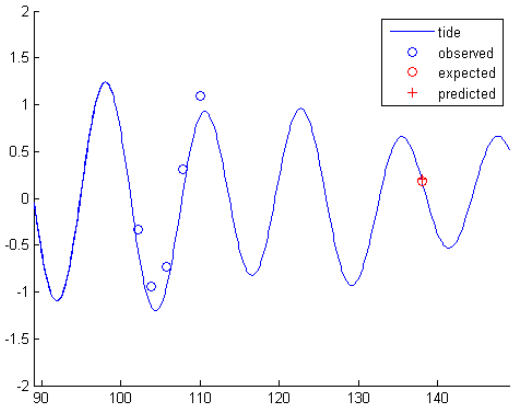
4.2 Predizione con errori massima

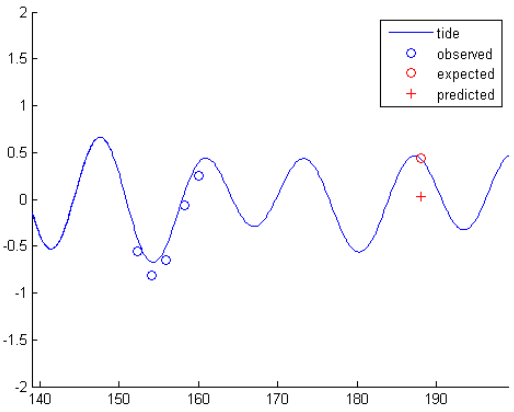
Fissiamo stavolta l'obiettivo di determinare il numero massimo di ore disponibili per la predizione con un errore medio sull'altezza di marea predetta inferiore al *5%*.

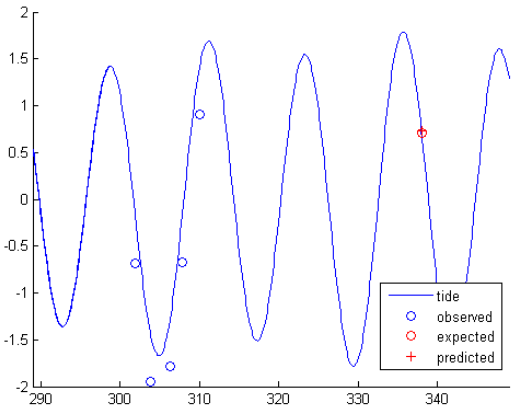
Sperimentalmente si è verificato che il numero massimo di ore che possono intercorrere tra l'ultima osservazione e la predizione della marea, con cinque osservazioni ogni due ore, risulta essere **28 ore**, se si vuole mantenere lo scarto quadratico medio inferiore a **0.2**, che corrisponde appunto al **5%**.
Come accennato nel capitolo precedente, se si accettano valori dell'errore medio più alti si possono ottenere molte più ore a disposizione per la previsione: ad esempio se si accetta un errore medio inferiore al **10%** risulta che si possono fare previsioni sull'altezza di marea fino a **76 ore** dopo l'ultima osservazione. Come prima, però, un tale lasso di tempo risulta inutile alla chiocciola la quale, intuitivamente, preferisce fare una stima più accurata a breve termine piuttosto che una grossonala a distanza di giorni.

Per quanto riguarda il file relativo a questa simulazione si può fare riferimento al file fin'ora menzionato (Codice 1), facendo attenzione a cambiare opportunamente i parametri: *int* deve valere *2*, *forecast* deve valere *28* (o *76*) e *minHN* e *maxHN* devono valere, rispettivamente, $-^{20}/_{60}$ e $+^{20}/_{60}$ (un errore di massimo 20 minuti corrisponde a circa il *16%* quando l'intervallo tra due osservazioni è di due ore).
I grafici che seguono sono relativi alla predizione *28 ore* dopo:






tide
observed
expected
predicted
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
140
150
160
170
180
190


tide
observed
expected
predicted
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
290
300
310
320
330
340

Capitolo 5

Predizione della prossima alta marea

Oltre alla predizione dopo un numero fissato di ore un altro importante esperimento è quello della predizione del prossimo picco di alta marea. Come nei capitoli precedenti, la chiocciola osserva **5** altezze di marea, ad **1 ora** di distanza ciascuna, per poi predire sia tra quante ore avverrà la prossima alta marea, sia l'altitudine che raggiungerà.
Per quanto riguarda gli errori sensoriali a cui la chiocciola può essere soggetta si è deciso di mantenere il **15%** di errore sulla percezione del livello dell'acqua e del **16%** sulla percezione del trascorrere del tempo che, essendo in questo caso l'intervallo tra due osservazioni successive di un'ora, corrisponde a $\pm$ **10 minuti**.

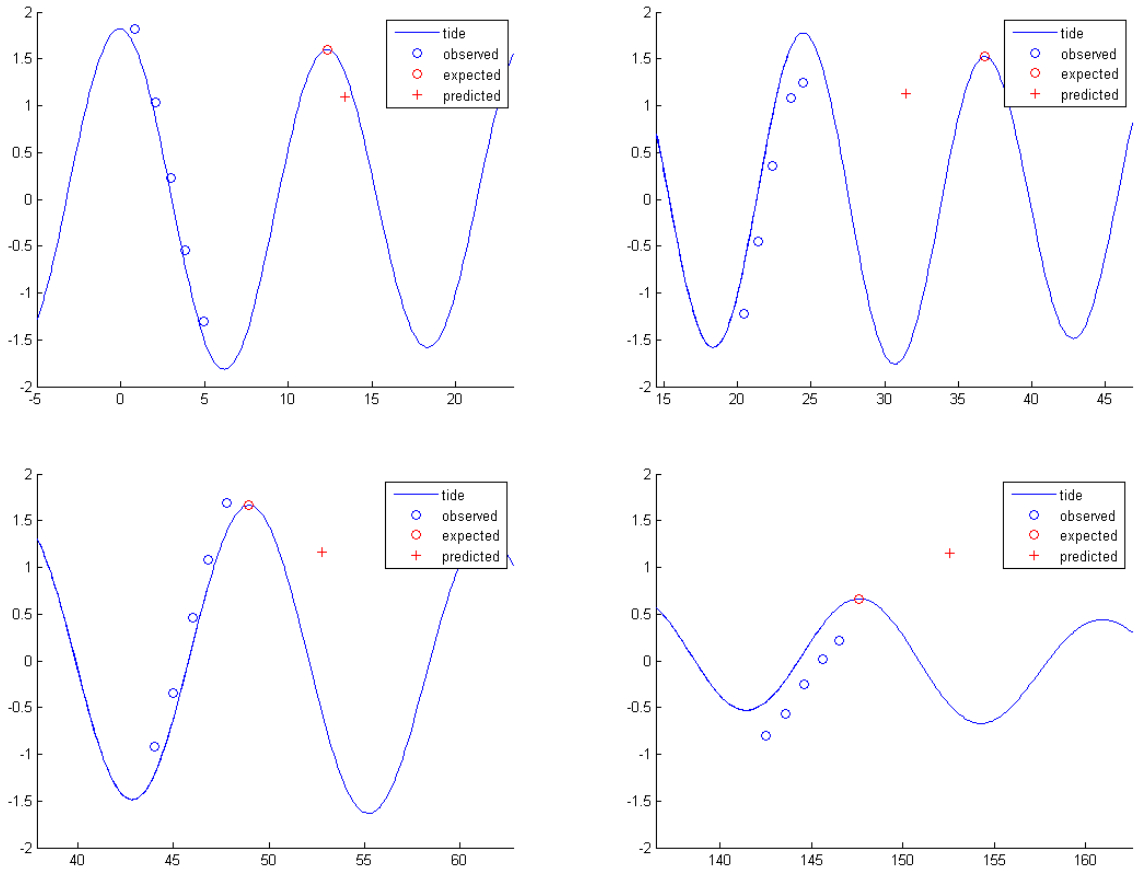
5.1 Predizione con rete lineare di Widrow

Il primo tentativo effettuato per la simulazione di questa particolare predizione consiste nell'utilizzo di una rete lineare di Widrow, cioè come quella utilizzata nei Capitoli 3 e 4.

Purtroppo, come ci potevamo aspettare, la rete di Widrow non è sufficiente per la valutazione di questa predizione, considerando che deve essere in grado di generare due output, uno relativo al tempo e l'altro relativo all'altezza, prendendo soltanto cinque altezze di marea come input.
L'errore ottenuto da questo tipo di simulazione è calcolato come la media degli errori sui due output, cioè tra un massimo di 4 per l'altezza di marea ed un massimo di 13 per l'orario in cui si verificherà l'alta marea (questo errore vale massimo *13* in quanto in realtà la chiocciola calcola il numero di ore che separano l'ultima osservazione dal prossimo picco di marea, i quali si verificano con frequenza di poco più di 12 ore). Quindi l'errore assoluto totale vale massimo $\frac{4+13}{2} = 8.5$, che graficamente corrisponde all'intorno circolare con centro l'effettivo picco di marea e raggio *8.5*.
Quello che si è ottenuto è stato un errore medio totale pari a **3.36** (in valore assoluto), che corrisponde a circa il **39.5%**. Ovviamente un errore del genere è

inaccettabile, quindi non ci rimane che affidarci a reti più complesse della rete lineare di Widrow, come vedremo nella prossima sezione.

Il Codice relativo a questa, fallimentare, simulazione è il Codice 4, consultabile all'interno del Listato del codice.
Di seguito alcuni grafici mostrano quanto alto possa essere l'errore simulando la predizione con questa rete:



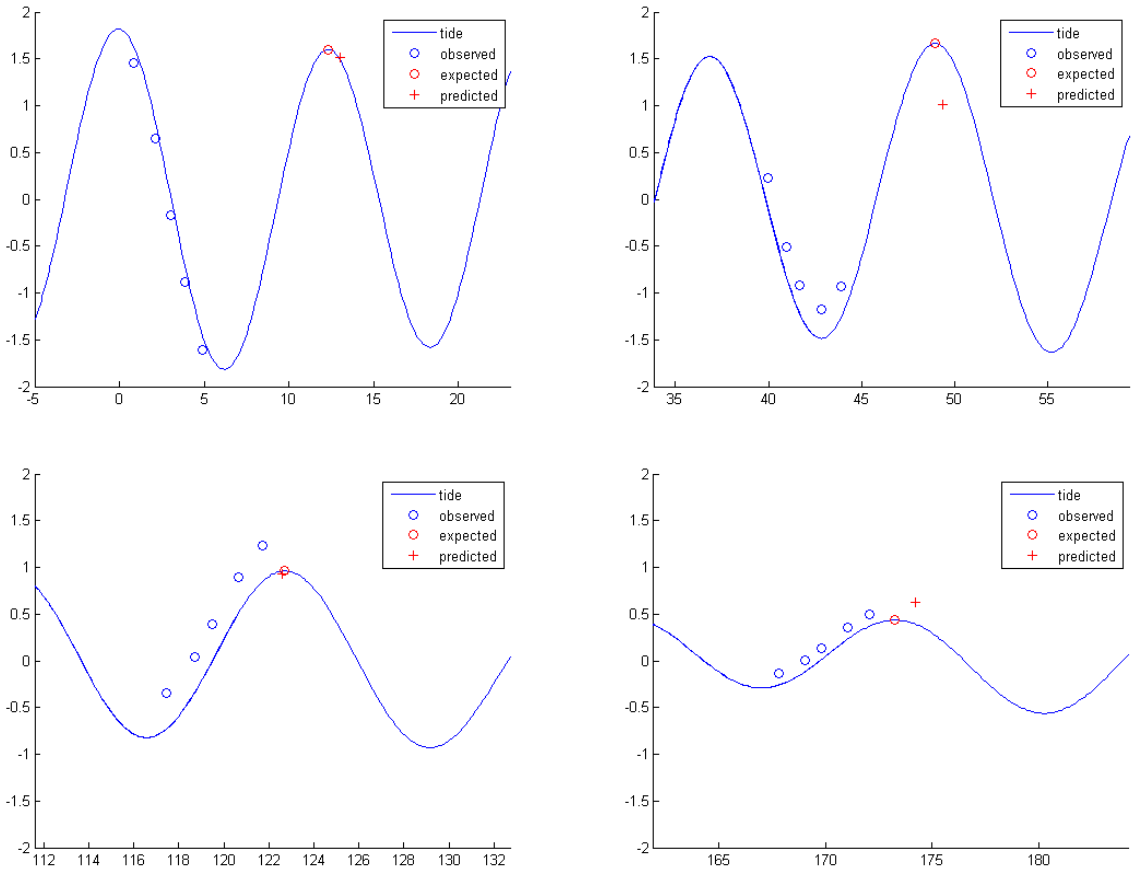
5.2 Predizione con Multilayer Perceptron

Il seguente tentativo è stato con una rete Feed-Forward, in particolare un Multilayer Perceptron.

Si è subito notato come questo tipo di rete risponda molto meglio alla predizione del prossimo picco di marea rispetto alla rete lineare di Widrow, quindi si è cercato di trovare il minimo numero di neuroni dello strato nascosto affinché l'errore medio totale potesse scendere sotto un valore accettabile, che ci è sembrato opportuno fissare al **12%**.
Il risultato è stato che **6 neuroni** rappresentano il minimo per avere un errore

minore uguale a **1.02**, che corrisponde appunto al **12%** su 8.5. Ovviamente, più neuroni fanno scendere ulteriormente l'errore, ma la crescita è molto lenta rispetto al numero di neuroni necessari, quindi il 12% di errore con 6 neuroni sembra essere un buon compromesso.

Per riprodurre tale simulazione si veda il Codice 5.



Capitolo 6

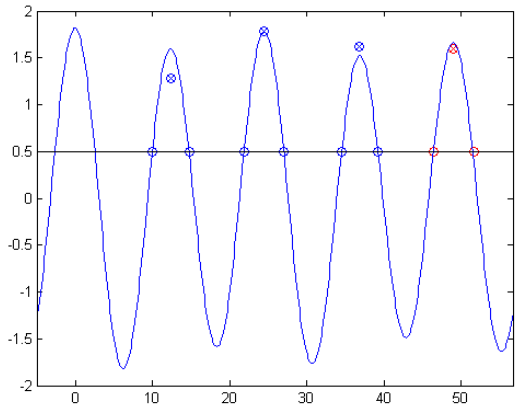
Predizione e osservazione dell'alta marea

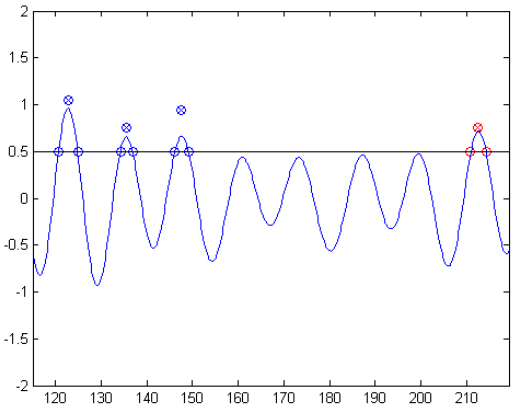
In questo capitolo esamineremo il problema in cui la chiocciola osserva il momento d'inizio, di picco e di fine dell'alta marea e l'altezza raggiunta nel momento di picco di *3 alte maree* consecutive che superino il livello del terreno, per poi fare una previsione sul momento di inizio, picco e fine della prossima alta marea, come dell'altezza raggiunta dal picco.
Abbiamo quindi fissato il livello del terreno a **0.5**, dove 0 è il livello del mare in assenza di marea e 2 è il massimo picco di marea. Gli errori sui dati sensoriali corrispondono al **10%** sulla percezione del livello di marea raggiunto ed un errore minimo, di massimo **5 minuti**, sullo scorrere del tempo.
L'addestramento della rete, un Multilayer Perceptron, è avvenuto utilizzando *500* pattern di esempio casuali, con relativi output obiettivo.

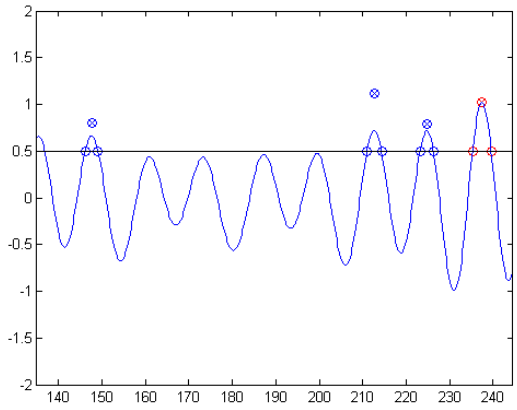
Il risultato ottenuto è che il minimo numero di neuroni nello strato nascosto necessari per ottenere un errore medio di circa lo **0.1%** risulta essere **17 neuroni**. Ancora una volta, quindi, si è arrivati alla conclusione che per effettuare una previsione del genere, anche se più complessa e anche in presenza di errori di percezione, la minima rete neurale necessaria risulta essere molto semplice. Per ottenere, invece, un errore medio inferiore all'**1%** bastano **4 neuroni**, mentre per errori inferiori al **5% 2 neuroni** risultano essere sufficienti.
Come ci si poteva aspettare da una previsione del genere, la rete simula molto meglio quando si hanno le 3 alte maree di osservazione e la marea da prevedere consecutive, ovvero 4 maree successive che superano il livello del terreno: infatti per questo tipo di predizione funzionano molto bene anche reti più semplici, anche con *10 neuroni*. Tuttavia, specialmente nei periodi di *neap tide*, ovvero quando si ha la luna in quadratura, le alte maree possono anche non riuscire a superare il livello del terreno e si possono avere anche giorni interi senza vedere la marea oppure vedendo una sola alta marea al giorno (al posto di due, come durante le *spring tide*). In questi casi è richiesta appunto una rete neurale minimamente più

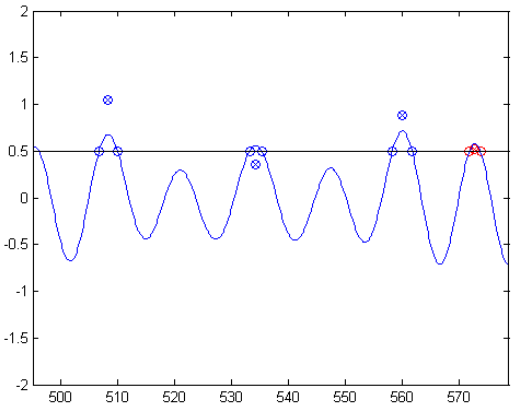
complessa in quanto anche solo riuscire a predire dopo quanto tempo la marea riuscirà a superare nuovamente il livello del terreno è sicuramente un calcolo più complesso che calcolare il prossimo inizio di marea durante la spring tide (dove gli intervalli di tempo tra una marea e l'altra sono molto regolari).

Di seguito vediamo alcuni grafici relativi a queste simulazioni: la funzione in blu rappresenta la funzione esatta di marea; la linea nera orizzontale indica il livello del terreno; i punti cerchiati in blu indicano l'osservazione dell'istante in cui è arrivata o finita la marea; i punti con un cerchio ed una X blu indicano i punti in cui si osserva l'arrivo e l'altezza del picco di marea; i punti cerchiati in rosso rappresentano la previsione sull'arrivo e fine della prossima alta marea; infine i punti con un cerchio ed una X rossa indicano la previsione fatta sul momento di arrivo e l'altezza del prossimo picco di marea.
Per riprodurre queste simulazioni si veda il Codice 8, Listato del codice.









Capitolo 7

Adattamento a nuove altitudini

Dopo aver cercato di stimare, nei Capitoli precedenti, la minima rete neurale in grado di effettuare le previsioni delle maree, analizzando il problema sotto vari aspetti, entreremo adesso più nel dettaglio del problema, tentando di costruire un modello sempre più accurato della rete neurale della *Cerithidea Decollata*.

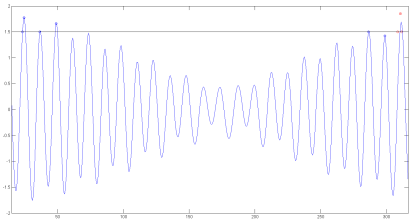
Nello specifico, in questo Capitolo tenteremo di riprodurre un esperimento condotto dal Prof. Vannini, nel quale degli esemplari di *Cherithidea Decollata*, residenti ad una determinata altitudine, vengono traslocati ad una nuova altitudine (in questo caso, più bassa dell'iniziale), per poter studiare come le chiocciole si adattano alla nuova situazione.

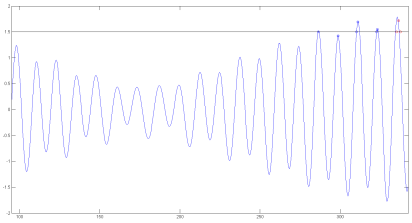
Innanzitutto, sono state apportate delle modifiche al modello definito nei precedenti Capitoli. Le osservazioni consistono adesso nell'osservare soltanto l'arrivo della marea (ovvero, quando questa supera il livello del terreno) e arrivo ed altezza del picco di marea. L'osservazione del riflusso della marea è stato eliminato, in quanto informazione ridondante conoscendo le prime due informazioni. Inoltre, quando queste chiocciole sono salite sul tronco e si sono ancorate, entrano in una specie di sonno, rendendo quindi improbabile l'osservazione del riflusso di marea. È stata inoltre introdotta una memoria per le chiocciole, della durata di circa 28 giorni (ovvero un interno ciclo di marea), e la possibilità di utilizzare o meno un orologio lunare, che aiuti le chiocciole nella previsione.
È stata quindi utilizzata una rete neurale di tipo *feed-forward* (come visto nei Capitoli precedenti, un Multilayer Perceptron) con **8 neuroni** nello strato nascosto, inizialmente addestrata su un insieme di informazioni relative all'altitudine iniziale, fissata ad **1.5** (con 2 massimo picco di marea, 0 livello del mare e -2 massimo riflusso di marea). Quindi, le chiocciole vengono "traslocate" alla nuova altitudine, fissata a **0.5**, e vengono addestrate con le nuove informazioni, suddividendo l'addestramento in giorni. Ogni giorno, vengono fatte *4 osservazioni* di marea, durante le quali le chiocciole modificheranno la propria rete neurale fino a trovare un riscontro con le informazioni appena acquisite e con le informazioni

che hanno ancora in memoria. Quindi viene aggiornata la memoria della chiocciola, eliminando le 4 osservazioni più vecchie ed inserendo le 4 osservazioni appena eseguite. Infine, viene calcolato l'errore che la chiocciola commetterà prevedendo la marea nel giorno corrente e nei **27 giorni** successivi (si veda il parametro `validationDays`, del Codice 14). Questo procedimento viene reiterato per più giorni finché l'errore commesso nel giorno corrente e nei 27 successivi (ovvero, per un intero ciclo di marea) non risulta essere minore di un valore prefissato (vedi parametro `errPerc`), in questo caso il **5%**.
Eseguendo quindi una serie di test con i parametri appena elencati è risultato che le chiocciole si adattano alla nuova altitudine in circa **9 giorni** di addestramento. Non è quindi necessario che esse osservino l'intero ciclo di marea nella nuova condizione per poter prevedere la marea con un errore medio inferiore al **5%**.

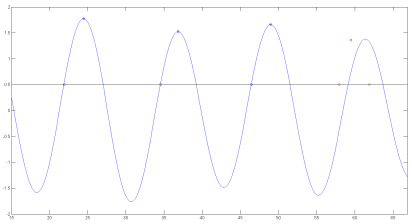
Si è introdotto anche un orologio lunare, in questo modello, per studiare come cambiano le previsioni quando le chiocciole sono in possesso di quest'informazione aggiuntiva. I risultati sono stati piuttosto deludenti: se viene utilizzato soltanto l'orologio lunare, le chiocciole non riescono ad adattarsi alla nuova altitudine entro l'errore prefissato; se utilizzato in concomitanza con l'osservazione diretta della marea, non sono stati registrati risultati degni di nota: nella maggior parte dei casi si hanno differenze di un solo giorno, o anche nessuno, per quanto riguarda l'addestramento.
Un'osservazione degna di nota riguarda invece il fatto che, aumentando il numero di neuroni della rete neurale, i giorni necessari per l'adattamento aumentano. Questo è dovuto al fatto che un maggior numero di neuroni implica una maggior complessità della rete neurale e di conseguenza un numero maggiore di giorni necessari per l'adattamento. Questo risultato non fa altro che evidenziare, ancora una volta, che la rete neurale necessaria per questo tipo di previsioni è estremamente semplice.

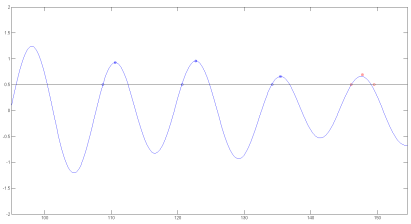
Vediamo di seguito due previsioni all'altitudine iniziale





e due previsioni alla nuova altitudine dopo l'addestramento





Il Codice MATLAB utilizzato per queste simulazioni è il Codice 14, Listato del codice.

Listato del codice

Codice 1: Previsioni di marea semplici e con errori.

```
% tideLin
%
% Function used for simple forecasting and the forecast with errors.
%
% Author: Tommaso Papini,
% Last modified: 20th March 2013, 17:12 CET.

%PARAMETERS
cycle = 28*24;  % tidal cycle duration (28 days) in hours
int = 0.5;    % tide observation intervals in hours
forecast = 2;   % hours away from the prediction
sample1 = 20000;    % tidal heights observed for training
sample2 = 5;    % observations for each forecast
minTN = -0.6;   % minimum tide noise
maxTN = +0.6;   % maximum tide noise
minHN = -5/60; % minimum hour noise
maxHN = +5/60; % maximum hour noise

%TRAINING DATA
P0 = ones(sample2, 1)*randR(1, sample1, 0, cycle) + (1 : sample2)'*ones
    (1, sample1)*int + randR(sample2, sample1, minHN, maxHN);
P = tide(P0) + ones(sample2, 1)*randR(1, sample1, minTN, maxTN);
T = tide(P0(sample2, :) + forecast) + randR(1, sample1, minTN, maxTN);

%TRAINING
net = newlin(P, T, 0, 0.01);
net.trainFcn = 'trainlm';
net.trainParam.epochs = 200;
net.trainParam.goal = 0.04;
net = init(net);
net = train(net, P, T);

%TEST
start = input('Insert the starting point (-1 to terminate): ');
while start~=-1
    graphicS = start-sample2*int-1;
```

```
    graphicE = start+2*sample2*int+forecast+1;
    h = start + (1 : sample2)'*int + randR(sample2, 1, minHN, maxHN);
    start = start + int*sample2 + forecast;
    t = tide(h) + ones(sample2, 1)*randR(1, 1, minTN, maxTN);
    e = tide(start);
    p = sim(net, t);
    hold off
    fplot(@tide, [graphicS graphicE], 'b')
    hold on
    axis ([graphicS graphicE -2 2]);
    plot(h', t', 'bo')
    plot(start, e, 'ro')
    plot(start, p, 'r+')
    legend({'tide', 'observed', 'expected', 'predicted'})
    pause
    start = input('Insert the starting point (-1 to terminate): ');
end
```

Codice 2: Funzione di marea.

```
% height = tide(t)
%
% Calculates the exact tide function.
%
% Input:
%   -t: time, in hours.
% Output:
%   -height: the corresponding tidal height.
%
% Author: Tommaso Papini,
% Last modified: 20th March 2013, 17:11 CET.

function height = tide(t)
    Aa = 1.06;
    Ab = 0.65;
    D = 0.15;
    Ta = 12.42;
    Tb = 12;
    Tm = 12.21;
    height = Aa*cos((2*pi*t)/Ta) + Ab*cos((2*pi*t)/Tb) - D*sin((pi*t)/
        Tm - pi/4);
end
```

Codice 3: Funzione per ottenere una matrice di valori casuali compresi tra due estremi.

```
% random = randR(M, N, A, B)
%
% Creates a matrix of random values in a given range.
%
% Input:
%   -M: matrix rows;
%   -N: matrix columns;
%   -A: lower limit;
%   -B: upper limit.
% Output:
%   -random: final random matrix.
%
% Author: Tommaso Papini,
% Last modified: 20th March 2013, 17:08 CET.

function random = randR(M, N, A, B)
    random = rand(M, N)*(B-A) +A;
end
```

Codice 4: Predizione del prossimo picco di marea con una rete lineare.

```
% tideLinEdge
%
% Prediction of the next tidal edge using a linear Widrow network.
%
% Author: Tommaso Papini,
% Last modified: 20th March 2013, 17:04 CET.

%PARAMETERS
cycle = 28*24;  % tidal cycle duration (28 days) in hours
int = 1;    % tide observation intervals in hours
sample1 = 10000;    % tidal heights observed for training
sample2 = 5;    % observations for each forecast
minTN = -0.6;   % minimum tide noise
maxTN = +0.6;   % maximum tide noise
minHN = -10/60; % minimum hour noise
maxHN = +10/60; % maximum hour noise

%TRAINING DATA
P0 = ones(sample2, 1)*randR(1, sample1, 0, cycle) + (1 : sample2)'*ones
    (1, sample1)*int + randR(sample2, sample1, minHN, maxHN);
P = tide(P0) + ones(sample2, 1)*randR(1, sample1, minTN, maxTN);
T1 = zeros(1, sample1);
i = 1;
while i<=sample1
   T1(i) = findNextEdge(P0(sample2, i));
   i = i+1;
end
T2 = tide(T1);
T1 = T1 - P0(sample2, :);
```

```
T = [T1; T2];

%TRAINING
net = newlin(P, 2, 0, 0.01);
net.trainFcn = 'trainlm';
net.trainParam.epochs = 200;
net.trainParam.goal = 0.0001;
net = init(net);
net = train(net, P, T);

%TEST
start = input('Insert the starting point (-1 to terminate): ');
while start~=-1
    h = start + (1 : sample2)'*int + randR(sample2, 1, minHN, maxHN);
    t = tide(h) + ones(sample2, 1)*randR(1, 1, minTN, maxTN);
    eX = findNextEdge(h(sample2, 1));
    eY = tide(eX);
    p = sim(net, t);
    p1 = p(1, 1)+h(sample2, 1);
    p2 = p(2, 1);
    graphicS = start -5;
    graphicE = max(eX, p1) +10;
    hold off
    fplot(@tide, [graphicS graphicE], 'b')
    hold on
    axis ([graphicS graphicE -2 2]);
    plot(h', t', 'bo')
    plot(eX, eY, 'ro')
    plot(p1, p2, 'r+')
    legend({'tide', 'observed', 'expected', 'predicted'})
    pause
    start = input('Insert the starting point (-1 to terminate): ');
end
```

Codice 5: Predizione del prossimo picco di marea con una rete Feed-Forward.

```
% tideFFEdge
%
% Prediction of the next tidal edge using a Multilayer Perceptron.
%
% Author: Tommaso Papini,
% Last modified: 20th March 2013, 17:04 CET.

%PARAMETERS
cycle = 28*24;  % tidal cycle duration (28 days) in hours
int = 1;    % tide observation intervals in hours
sample1 = 10000;    % tidal heights observed for training
sample2 = 5;    % observations for each forecast
minTN = -0.6;   % minimum tide noise
maxTN = +0.6;   % maximum tide noise
```

```
minHN = -10/60; % minimum hour noise
maxHN = +10/60; % maximum hour noise

%TRAINING DATA
P0 = ones(sample2, 1)*randR(1, sample1, 0, cycle) + (1 : sample2)'*ones
    (1, sample1)*int + randR(sample2, sample1, minHN, maxHN);
P = tide(P0) + ones(sample2, 1)*randR(1, sample1, minTN, maxTN);
T1 = zeros(1, sample1);
i = 1;
while i<=sample1
   T1(i) = findNextEdge(P0(sample2, i));
   i = i+1;
end
T2 = tide(T1);
T1 = T1 - P0(sample2, :);
T = [T1; T2];

%TRAINING
net = newff(P, T, 6, {'logsig', 'purelin'}, 'trainlm');
net.trainParam.epochs = 200;
net.trainParam.goal = 0.0001;
net = init(net);
net = train(net, P, T);

%TEST
start = input('Insert the starting point (-1 to terminate): ');
while start~=-1
    h = start + (1 : sample2)'*int + randR(sample2, 1, minHN, maxHN);
    t = tide(h) + ones(sample2, 1)*randR(1, 1, minTN, maxTN);
    eX = findNextEdge(h(sample2, 1));
    eY = tide(eX);
    p = sim(net, t);
    p1 = p(1, 1)+h(sample2, 1);
    p2 = p(2, 1);
    graphicS = start -5;
    graphicE = max(eX, p1) +10;
    hold off
    fplot(@tide, [graphicS graphicE], 'b')
    hold on
    axis ([graphicS graphicE -2 2]);
    plot(h', t', 'bo')
    plot(eX, eY, 'ro')
    plot(p1, p2, 'r+')
    legend({'tide', 'observed', 'expected', 'predicted'})
    pause
    start = input('Insert the starting point (-1 to terminate): ');
end
```

Codice 6: Inverso della funzione di marea.

```
% height = tideNeg(t)
%
% Calculates the negation of the tide function.
%
% Input:
%   -t: time, in hours.
% Output:
%   -height: the negation of the corresponding tidal height.
%
% Author: Tommaso Papini,
% Last modified: 20th March 2013, 17:11 CET.

function height = tideNeg(t)
    Aa = 1.06;
    Ab = 0.65;
    D = 0.15;
    Ta = 12.42;
    Tb = 12;
    Tm = 12.21;
    height = - (Aa*cos((2*pi*t)/Ta) + Ab*cos((2*pi*t)/Tb) - D*sin((pi*t
        )/Tm - pi/4));
end
```

Codice 7: Funzione che determina il prossimo picco di marea.

```
% x_max = findNextEdge(x)
%
% Finds the next tide edge, between the passed time (x) and the next 13
    hours.
%
% Input:
%   -x: present time (in hours).
% Output:
%   -x_max: hour of the next tide edge.
%
% Author: Tommaso Papini,
% Last modified: 20th March 2013, 16:50 CET.
function x_max = findNextEdge(x)
   x_max = fminbnd(@tideNeg, x, x+13);
end
```

Codice 8: Osservazione e predizione di picchi di marea.

```
% tideEdges
%
```

```
% Trains a Multylayer Perceptron to forecast the incoming tide
% observing tidal edges.
%
% Author: Tommaso Papini,
% Last modified: 20th March 2013, 16:42 CET.

%PARAMETERS
cycle = 28*24;  % tidal cycle duration (28 days) in hours
sample1 = 500;    % tidal heights observed for training
sample2 = 3;    % observations for each forecast
minTN = -0.4;   % minimum tide noise
maxTN = +0.4;   % maximum tide noise
hN = 5/60; % maximum (absolute) hour noise
terrain = 0.5;  % altitude of the ground (0 = sea level, 2 = maximum
    tidal edge)
errPerc = 5;    % desired error (in percentage)

%TRAINING DATA
P0 = randR(1, sample1, 0, cycle);
P1 = zeros(3, sample1);
for i=1:sample1
   x = findNextEdge(P0(i));
   while tide(x)<terrain, x=x+1; x=findNextEdge(x); end
   while tide(x)>terrain, x=x-hN; end
   P0(i)=x;
   P0(i) = findNextEdge(P0(i));
   P1(1, i) = P0(i) - x;
   P1(2, i) = tide(P0(i))+randR(1, 1, minTN, maxTN);
   x = P0(i);
   while tide(x)>terrain, x = x+hN; end
   P1(3, i) = x-P0(i);
   P0(i) = x;
end
P = [P1 ; zeros(4*(sample2-1), sample1)];
for i=2:sample2
    y=3+(i-2)*4;
    for j=1:sample1
        x = P0(j);
        while tide(x)<terrain, x = x+hN; end
        P(y+1, j) = x-P0(j);
        P0(j) = x;
        x = findNextEdge(x);
        P(y+2, j) = x-P0(j);
        P0(j) = x;
        P(y+3, j) = tide(P0(j))+randR(1, 1, minTN, maxTN);
        while tide(x)>terrain, x = x+hN; end
        P(y+4, j) = x-P0(j);
        P0(j) = x;
    end
end
T = zeros(4, sample1);
for i=1:sample1
    x = P0(i);
    while tide(x)<terrain, x = x+hN; end
    T(1, i) = x-P0(i);
```

```
    P0(i) = x;
    x = findNextEdge(x);
    T(2, i) = x-P0(i);
    P0(i) = x;
    T(3, i) = tide(P0(i))+randR(1, 1, minTN, maxTN);
    while tide(x)>terrain, x = x+hN; end
    T(4, i) = x-P0(i);
end

%TRAINING
net = newff(P, T, 2, {'logsig', 'purelin'}, 'trainlm');
net.trainParam.epochs = 200;
net.trainParam.goal = (errPerc*10.75)/100;
net = init(net);
net = train(net, P, T);

%TEST
start = input('Insert the starting point (-1 to terminate): ');
while start~=-1
    H = zeros(1, 3*sample2);    % hours observed
    E = zeros(1, sample2);  % edges observed
    H(1) = start;
    H(1) = findNextEdge(H(1));
    while tide(H(1))<terrain, H(1)=H(1)+1; H(1)=findNextEdge(H(1)); end
    while tide(H(1))>terrain, H(1)=H(1)-hN; end
    H(2)= findNextEdge(H(1));
    E(1)=tide(H(2))+randR(1, 1, minTN, maxTN);
    H(3) = H(2);
    while tide(H(3))>terrain, H(3)=H(3)+hN; end
    for i=2:sample2
        x = (i-1)*3;
        H(x+1) = H(x);
        while tide(H(x+1))<terrain, H(x+1)=H(x+1)+hN; end
        H(x+2) = findNextEdge(H(x+1));
        E(i)=tide(H(x+2))+randR(1, 1, minTN, maxTN);
        H(x+3) = H(x+2);
        while tide(H(x+3))>terrain, H(x+3)=H(x+3)+hN; end
    end
    hold off
    I = zeros(1, 3+(sample2-1)*4);  %input vector
    i = sample2;
    while i>1
        x = (i-1)*3;
        y=3+(i-2)*4;
        I(y+4)=H(x+3)-H(x+2);
        plot(H(x+3), terrain, 'bo')
        hold on
        I(y+3)=E(i);
        I(y+2)=H(x+2)-H(x+1);
        plot(H(x+2), E(i), 'bo')
        plot(H(x+2), E(i), 'bx')
        I(y+1)=H(x+1)-H(x);
        plot(H(x+1), terrain, 'bo')
        i=i-1;
    end
```

```
    I(3)=H(3)-H(2);
    plot(H(3), terrain, 'bo')
    I(2)=E(1);
    I(1)=H(2)-H(1);
    plot(H(2), E(1), 'bo')
    plot(H(2), E(1), 'bx')
    plot(H(1), terrain, 'bo')
    I = I';
    O = sim(net, I);    %output vector
    x = H(3*sample2) + O(1);
    plot(x, terrain, 'ro')
    x = x + O(2);
    plot(x, O(3), 'ro')
    plot(x, O(3), 'rx')
    x = x + O(4);
    plot(x, terrain, 'ro')
    graphicS = start -5;
    graphicE = x+5;
    fplot(@tide, [graphicS graphicE], 'b')
    fplot(@(x) terrain, [graphicS graphicE], 'black')
    axis ([graphicS graphicE -2 2]);
    pause
    start = input('Insert the starting point (-1 to terminate): ');
end
```

Codice 9: Inizializzazione della memoria.

```
% [Pmem, Tmem] = memoryInit(dataPerDay, samples, tN, hN, terrain, cycle
    , memoryLength, lunar)
%
% Initializes the snail's memory after the training at the first
% altitude.
%
% Input:
%   -dataPerDay: number of training data (observations & forecast) per
%   day;
%   -samples: observations for each forecast;
%   -tN: maximum (absolute) tidal noise;
%   -hN: maximum (absolute) hour noise;
%   -terrain: ground level;
%   -cycle: tidal cycle in hours;
%   -memoryLength: memory length (in days);
%   -lunar: true if the lunar clock has to be used, false otherwise.
% Output:
%   -Pmem: input vector to be stored in memory;
%   -Tmem: target vector to be stored in memory.
%
% Author: Tommaso Papini,
% Last modified: 28th December 2012, 11:06 CET.
```

```
23 function [Pmem, Tmem] = memoryInit(dataPerDay, samples, tN, hN, terrain
       , cycle, memoryLength, lunar)
       [Pmem, Tmem] = trainDataEdges(dataPerDay*memoryLength, samples, tN,
           hN, terrain, cycle-memoryLength*24, cycle, true, lunar);
25 end
```

Codice 10: Aggiornamento della memoria.

```
1  % [Pmem, Tmem] = memoryUpdate(dataPerDay, memoryLength, Pmem, Tmem, P,
       T)
   %
3  % Updates the snail's memory with the information of the last training
   % day.
5  %
   % Input:
7  %   -dataPerDay: number of training data (observations & forecast) per
   %    day;
9  %   -memoryLength: memory length (in days);
   %   -Pmem: input vector stored in memory;
11 %   -Tmem: target vector stored in memory;
   %   -P: input vector of the last training day;
13 %   -T: target vector of the last training day.
   % Output:
15 %   -Pmem: updated input vector;
   %   -Tmem: updated target vector.
17 %
   % Author: Tommaso Papini,
19 % Last modified: 28th December 2012, 11:09 CET.

21 function [Pmem, Tmem] = memoryUpdate(dataPerDay, memoryLength, Pmem,
       Tmem, P, T)
       Pmem = [Pmem(:, dataPerDay+1:dataPerDay*memoryLength), P];
23     Tmem = [Tmem(:, dataPerDay+1:dataPerDay*memoryLength), T];
   end
```

Codice 11: Test per il Codice 14.

```
   % test_adapt(net, samples, tN, hN, terrain)
2  %
   % Test script for the adapting model: draws observation and prevision
4  % starting from points selected by the user.
   %
6  % Input:
   %   -net: the neural network to be tested;
8  %   -samples: observations for each forecast;
```

```
%   -tN: maximum (absolute) tidal noise;
%   -hN: maximum (absolute) hour noise;
%   -terrain: ground level.
%
% Author: Tommaso Papini,
% Last modified: 28th December 2012, 11:13 CET.

function [] = test_adapt(net, samples, tN, hN, terrain)

    start = input('Insert the starting point (-1 to terminate): ');
    while start~=-1
        [I, ~] = trainDataEdges(1, samples, tN, hN, terrain, start,
            start, true, true);
        x = I(1);
        i=0;
        figure(1);
        hold off;
        for i=1:samples
            y=(i-1)*3+1;
            x = x + I(y +1);
            plot(x, terrain, 'bo');
            hold on;
            x = x + I(y +2);
            plot(x, I(y +3)+terrain, 'bo');
            plot(x, I(y +3)+terrain, 'bx');
        end
        O = net(I);
        x = x + O(1);
        plot(x, terrain, 'ro');
        x = x + O(2);
        plot(x, O(3)+terrain, 'ro');
        plot(x, O(3)+terrain, 'rx');
        x = x + O(4);
        plot(x, terrain, 'ro');

        graphicS = start -5;
        graphicE = x+5;
        fplot(@tide, [graphicS graphicE], 'b')
        fplot(@(x) terrain, [graphicS graphicE], 'black')
        axis ([graphicS graphicE -2 2]);

        pause
        start = input('Insert the starting point (-1 to terminate): ');
    end

end
```

Codice 12: Funzione che calcola lo *score* raggiunto dalla rete neurale.

```
% score = getScore(net, validationDays, Pnew, Tnew, i)
```

```
%
% Calculates the root mean square of the net for the next
% validationDays days.
%
% Input:
%   -net: the neural network used;
%   -validationDays: number of validation days;
%   -Pnew: input vector from the last training day;
%   -Tnew: target vector from the last training day;
%   -i: current day of training.
% Output:
%   -score: the calculated root mean square.
%
% Author: Tommaso Papini,
% Last modified: 28th December 2012, 11:01 CET.

function score = getScore(net, validationDays, Pnew, Tnew, i)
    i=mod(i, 28)+1;
    lastDay = i+validationDays;
    if lastDay>28
        P=[Pnew(:,i:28), Pnew(:, 1:mod(lastDay, 28))];
        T=[Tnew(:,i:28), Tnew(:, 1:mod(lastDay, 28))];
    else
        P=Pnew(:, i:lastDay);
        T=Tnew(:, i:lastDay);
    end
    score = mse(net, T, net(P));
end
```

Codice 13: Generatore dei dati di addestramento per la rete neurale.

```
% [P, T] = trainDataEdges(sample1, sample2, minTN, maxTN, hN, terrain)
%
% Produces training data for the network that observes the arrival of
% tides and tidal edges.
%
% Input:
%   -sample1: tidal heights observed for training;
%   -sample2: observations for each forecast;
%   -tN: maximum (absolute) tidal noise;
%   -hN: maximum (absolute) hour noise;
%   -terrain: ground level (0 = sea level, 2 = maximum tidal edge);
%   -tStart: starting time for observations;
%   -tEnd:  ending time for observations.
% Output:
%   -P: input matrix: each column represent an input vector;
%   -T: target matrix: each column represent a target vector.
%
% Author: Tommaso Papini,
% Last modified: 28th December 2012, 11:15 CET.
```

```
20
function [P, T] = trainDataEdges(sample1, sample2, tN, hN, terrain,
    tStart, tEnd, lin, lunar)
22
    if lin, P0 = linspace(tStart, tEnd, sample1);
24  else P0 = sort(randR(1, sample1, tStart, tEnd)); end
    if lunar, P = zeros(1+3*sample2, sample1);
26  else P = zeros(3*sample2, sample1); end
    for j=1:sample1
28      while tide(P0(j))>terrain, P0(j)=P0(j)+hN; end
        if lunar, P(1, j)=mod(P0(j), 28*24); end
30      for i=1:sample2
            if lunar, y=(i-1)*3+1;
32          else y=(i-1)*3+1; end
            x = P0(j);
34          while tide(x)>terrain, x=x+hN; end
            while tide(x)<=terrain, x = x+hN; end
36          P(y+1, j) = x-P0(j);
            P0(j) = x;
38          x = findNextEdge(x);
            P(y+2, j) = x-P0(j);
40          P0(j) = x;
            P(y+3, j) = tide(P0(j))+randR(1, 1, -tN, tN)-terrain;
42      end
    end
44  T = zeros(4, sample1);
    for i=1:sample1
46      x = P0(i);
        while tide(x)>terrain, x=x+hN; end
48      while tide(x)<=terrain, x = x+hN; end
        T(1, i) = x-P0(i);
50      P0(i) = x;
        x = findNextEdge(x);
52      T(2, i) = x-P0(i);
        P0(i) = x;
54      T(3, i) = tide(P0(i))-terrain;
        while tide(x)>terrain, x = x+hN; end
56      T(4, i) = x-P0(i);
    end
58
end
```

Codice 14: Rete neurale per l'adattamento a nuove altitudini.

```
% tideFF_adapt
2 %
% Neural Network model for the Cerithidea Decollata snail adapting to
4 % new altitudes.
%
6 % Author: Tommaso Papini,
```

```
% Last modified: 28th December 2012, 11:14 CET.

%PARAMETERS
samples = 3;     % observations for each forecast
dataPerDay = 4; % sets of observations for each day
tN = 0.0;    % maximum (absolute) tidal noise
hN = 5/60; % maximum (absolute) hour noise
errPerc = 5;     % desider error in percentage
cycle = 28*24;  % tidal cycle (28 days) in hours
memoryLength = 28;  % memory length
n = 8;  % neurons
lunar = true;    % lunar clock usage
maxDays = 100;
validationDays = 27;
terrainOld = 1.5;
terrainNew = 0.5;

%TRAINING DATA
err = (errPerc*mean([13, 13, 4]))/100;
[Pold, Told] = trainDataEdges(500, samples, tN, hN, terrainOld, 0,
    cycle, true, lunar);

% FEED-FORWARD NET
net = newff(Pold, Told, n, {'logsig', 'purelin'}, 'trainlm');
net.trainParam.epochs = 500;
net.trainParam.goal = err;
net = init(net);
net = train(net, Pold, Told);

[Pmem, Tmem] = memoryInit(dataPerDay, samples, tN, hN, terrainOld,
    cycle, memoryLength, lunar);

test_adapt(net, samples, tN, hN, terrainOld);

%-------------------------------------------------------------------------

i=0;
net.trainParam.showWindow = false;
[Pnew, Tnew] = trainDataEdges(dataPerDay*28, samples, tN, hN,
    terrainNew, 0, cycle, true, lunar);
score = getScore(net, validationDays, Pnew, Tnew, i);
figure;
while score>err && i<=maxDays
    [P, T] = trainDataEdges(dataPerDay, samples, tN, hN, terrainNew,
        24*i, 24*(i+1), false, lunar);
    [Pmem, Tmem] = memoryUpdate(dataPerDay, memoryLength, Pmem, Tmem, P
        , T);
    net = train(net, Pmem, Tmem);
    score = getScore(net, validationDays, Pnew, Tnew, i);
    scoreOld = getScore(net, validationDays, Pold, Told, i);
    i=i+1;
    str=sprintf('\nDay %d:\n\tgoal: %5.4f\n\tmse new (%d days): %5.12f\
        n\tmse old (%d days): %5.12f\n', i, err, validationDays, score,
         validationDays, scoreOld); disp(str);
end
```

```
if score<=err
    if i==1, str=sprintf('\nThe network adapted to the new altituded in
         %d day with an error on the prediction of less than %d%%', i,
        errPerc);
    else str=sprintf('\nThe network adapted to the new altituded in %d
        days with an error on the prediction of less than %d%%', i,
        errPerc); end
else
    str=sprintf('\nThe network couldn''t adapt to the new altitude in %
        d days with an error on the prediction of less than %d%%',
        maxDays, errPerc);
end
disp(str);

net = train(net, Pnew, Tnew);
test_adapt(net, samples, tN, hN, terrainNew);
```