



UNIVERSITÀ DI PARMA

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA CIVILE

UN MODELLO DI INTELLIGENZA ARTIFICIALE PER LA PROPAGAZIONE DELLE PIENE SUL TORRENTE PARMA

AN ARTIFICIAL INTELLIGENCE MODEL FOR THE
PROPAGATION OF FLOODS ON THE PARMA STREAM

RELATORE:

DOTT. ING. RENATO VACONDIO

CORRELATORE:

PROF. ING. PAOLO MIGNOSA

DOTT. ING. SUSANNA DAZZI

Laureando:
Gabriele Simonetta
Matr. 288832

Anno accademico 2019 – 2020

L'intelligenza non è dimostrata dalla facilità di apprendimento, ma dalla
comprendere di ciò che apprendiamo.
(Joseph Whitney)

Indice

Glossario	IX
1 Introduzione	1
2 Le reti neurali artificiali	3
2.1 Breve storia sulle reti neurali	4
2.2 Rete neurale: caratteristiche e funzioni	5
2.3 Perceptron	7
2.4 Struttura della rete	7
2.4.1 Single layer/multi layer	7
2.5 Apprendimento	8
2.5.1 Funzione di ottimizzazione e di costo	8
2.5.2 Supervisionato o non supervisionato	10
2.5.3 Back-propagation	12
2.5.4 Feedforward Neural Network	12
2.6 Il neurone	13
2.6.1 Activation function	13
2.6.2 Pesi sinaptici	15
2.7 Epoche e Batch	15
2.8 Casi analoghi in idrologia e idraulica	15
3 Caso di studio	17
3.1 Area di studio: il torrente Parma	17
3.1.1 Aspetti idrologici	17
3.1.2 Livello di protezione	18
3.2 Dati utilizzati: i dati reali	19
3.2.1 Verifica dei dati	23
4 La rete neurale realizzata	25
4.1 Librerie usate	26
4.2 Rielaborazione dati reali per la rete neurale	26

Indice

4.2.1 Elaborazione dati input	26
4.2.2 Elaborazione dati output	27
4.3 Finestra di previsione	28
4.4 Training set e test set	29
4.5 Prima elaborazione	29
4.6 Seconda elaborazione	30
4.7 Terza elaborazione	31
4.7.1 Aggiunta dei dati da Casalmaggiore	31
4.7.2 Aggiunta dei dati della stazione di Colorno in input	33
4.8 Ottimizzazione della rete	35
4.8.1 Implementazione validation set	35
4.8.2 Implementazione di cicli	37
4.8.3 Verifica finestra di osservazione	40
5 Analisi risultati ottenuti	43
5.1 Rete ottimale	43
5.1.1 Sei ore di previsione	43
5.1.2 Nove ore di previsione	46
5.1.3 Problematiche rilevate	48
5.1.4 Previsione delle onde significative del 2014 e del 2017	50
5.2 Applicazione alla previsione di due eventi di piena del 2020	52
6 Conclusioni	55
Bibliografia	57
A Dati utilizzati	59
B Script Matlab: interpolazione dati	65
B.1 Discretizzazione ed elaborazione dati delle stazioni	65
B.2 Funzione richiamata in B.1 per l'interpolazione	69
C Script Python: reti neurali	71
C.1 Percettrone	71
C.2 Rete neurale conclusiva	73
C.3 Caricamento modello neurale salvato	80
C.4 Tempi di elaborazione a differenti batch	84

Elenco delle figure

2.1 Rappresentazione di un neurone artificiale	5
2.2 Rappresentazione di una rete neurale, in verde gli strati di input e output, in blu gli strati nascosti	8
2.3 Esemplificazione delle funzioni MAE e MSE	9
2.4 Esempio grafico underfitting e overfitting	11
2.5 Esempio grafico della funzione ReLu	14
3.1 Inquadramento del bacino del Parma, figura estratta da: Linee generali dell'assetto idrogeologico e quadro interventi, bacino del Parma	18
3.2 Ricucitura dei punti mancanti, esempio sull'anno 2012	21
3.3 Quota idrica a Parma Ponte Verdi	23
3.4 Andamenti salvati nelle tre stazioni nei differenti anni. In legenda P.Verdi = Ponte Verdi, Cas.mag. = Casalmaggiore	24
4.1 Finestra di previsione a 18 ore, onda del 04/04/2019 alla stazione idrometrica Ponte Verdi	28
4.2 Previsione onde del 12/2017 nel training set e 11/2018 nel test set	30
4.3 Previsione onde del 12/2017 e 11/2018; aggiunta dei dati da Ca- salmaggiore	32
4.4 Previsione onda del 06/12/2017 del training set a diverse ore, input Casalmaggiore	32
4.5 Previsione onda del 25/11/2018 del test set a diverse ore, input Casalmaggiore	33
4.6 Previsione onda del 25/11/2018 del test set a diverse ore, input Casalmaggiore e Colorno	34
4.7 Validation set confronto MAE fra 10% e 34%	36
4.8 Validation set confronto MAE fra 10% e 34%, media su 10 cicli .	37
4.9 Particolare dell'onda del 10/21/2019 (prime 10 ore) con tre cicli di previsione a sei ore e relativa media	38

Elenco delle figure

4.10 Previsione a sei ore dell'onda del 21/11/2018 a Colorno mediata su dieci cicli	40
4.11 Finestra di previsione a 18 ore, onda del 25/03/2015 alla stazione idrometrica Ponte Verdi	40
5.1 Rappresentazione dell'onda del 20/11/2018 con differenti neuroni a sei ore di previsione	45
5.2 Onda del 20/11/2018 in diverse configurazioni con previsione a 6 ore	46
5.3 Rappresentazione dell'onda del 04/03/2018 con differenti batch a sei ore di previsione	47
5.4 Tempi di elaborazione a differenti batch	47
5.5 Rappresentazione dell'onda del 20/11/2018 con differenti neuroni a nove ore di previsione	48
5.6 Errori di previsione nelle onde del 18 e 21 dicembre 2019	49
5.7 Periodo delle onde in esame 18-21 dicembre 2019	50
5.8 Previsione delle piene del dicembre 2017, in alto, e del ottobre 2014, in basso	51
5.9 Previsione delle onde del 29/02/2020, in alto, e del 3/12/2020, in basso	53
A.1 Andamenti salvati nelle tre stazioni negli anni usati come testing	60
A.2 Andamenti salvati nelle tre stazioni negli anni usati come training	61
A.3 Andamenti salvati nelle tre stazioni negli anni usati come training	62
A.4 Andamenti salvati nelle tre stazioni nell'anno 2020	63

Elenco delle tabelle

3.1 Dati scaricati da Aegis	20
3.2 Output script di Matlab	22
4.1 Rielaborazione dati in input esempio con 10 valori	27
4.2 Rielaborazione dati in output esempio con 10 valori	28
4.3 Distribuzione dei dati iniziale	29
4.4 Aggiunta dei dati di Casalmaggiore	31
4.5 Risultati di previsione con Colorno in input	34
4.6 Validation set, varie prove	35
4.7 Risultati di differenti cicli con differenti neuroni	39
4.8 Finestra dati input per previsione a nove ore	41
5.1 Tempi osservati nelle varie configurazioni	43
5.2 Valori medi di dieci cicli e previsione a sei ore	44
6.1 RMSE medio nei vari orizzonti di predizione	56

Glossario

A

ANN Artificial Neural Network. 5, 7, 10, 14,
16, 26

F

FNN Feedforward Neural Network. 6, 12

M

MAE Mean Absolute Error. 9, 35, 36
MSE Mean Square Error. 9

R

RMSE Root Mean Square Error. 9, 29, 38, 56
RNN Recurrent Neural Network. 6, 12

S

std standard deviation. 39

1

Introduzione

Le reti neurali sono utilizzate in moltissimi campi, dall'ambito automobilistico, sino alla scelta del punto di atterraggio dei rover su Marte¹, passando per la realizzazione di modelli afflussi deflussi.

Le reti neurali, originariamente inventate nel 1943, sono tornate recentemente alla ribalta grazie all'aumento delle capacità computazionali e alle ricerche nel campo dell'intelligenza artificiale. Le applicazioni sono di conseguenza cresciute esponenzialmente e le librerie, specie nel campo dell'opensource, hanno seguito il medesimo trend.

In questo contesto il presente elaborato di tesi si pone l'obiettivo di realizzare una rete neurale per la previsione in tempo reale delle piene a Colorno, partendo dai dati reali disponibili.

Nella prima parte vengono esposti i concetti base delle reti neurali. Nella seconda vengono descritti i dati utilizzati. Nella terza si riportano i dettagli della rete neurale realizzata per lo scenario in studio. Infine, nella quarta e ultima parte della tesi sono mostrate le previsioni ottenute per alcuni eventi storici significativi, compreso il recente evento verificatosi nel dicembre 2020.

¹Il rover Perseverance ha utilizzato tale tecnica per l'atterraggio su Marte avvenuto il 18 febbraio del 2021.

2

Le reti neurali artificiali

La rete neurale artificiale si distingue dagli altri sistemi di calcolo per un elemento fondamentale: la non linearità di risposta. Mentre i sistemi seriali di elaborazione sono piuttosto efficienti nel determinare in modo univoco una soluzione ad un quesito "chiuso", in cui si conoscono tutti i parametri in gioco e le relazioni fra gli stessi, un sistema neurale artificiale permette di giungere ad una soluzione accettabile di problemi anche complessi, senza dover conoscere ogni singolo legame fra le grandezze in gioco. I sistemi chiusi, per giungere a soluzione passano attraverso passaggi definiti quali soluzioni di sistemi o risoluzione di equazioni più o meno complicate, predefiniti e questo ne limita l'uso in scenari controllati e ben delineati.

Prendiamo un esempio banale: l'identificazione di un oggetto di uso comune dopo una serie di domande poste all'intervistato. Se chiedessimo ad un bambino di identificare l'oggetto "cucchiaio" avremmo infiniti modi per descriverlo, partendo dalla forma, da un disegno, dall'uso o dalla sua collocazione in cucina. In un sistema seriale dovremo passare per step per giungere a soluzione; il sistema migliore sarebbe attraverso un albero binario che escluda di domanda in domanda le alternative (questo fra l'altro è lo stratagemma su cui si basano i giochi Twenty Quest, e il più recente Akinator¹). Una rete neurale, invece, può determinare il cucchiaio semplicemente da una foto o un disegno, come è possibile fare su Autodraw².

¹Akinator: <https://it.akinator.com/>

²Autodraw: <https://www.autodraw.com/>

2. Le reti neurali artificiali

I sistemi neurali artificiali, generalizzando, sono quindi utili negli scenari in cui:

- non è possibile identificare in modo univoco l'insieme di equazioni caratterizzanti l'evento;
- l'onere computazionale di un sistema classico è eccessivo.

2.1 Breve storia sulle reti neurali

McCulloch and Pitts (1943) con la pubblicazione dal titolo *A logical calculus of the ideas immanent in nervous activity* misero le basi della storia delle reti neurali artificiali (ANN, dall'inglese *artificial neural network*) con un primo modello: esso era costituito da più neuroni collegati in parallelo, in vari strati e permetteva di svolgere funzioni booleane. Il singolo neurone era caratterizzato da una funzione di attivazione e da una soglia per l'attivazione. Come riporta (*Floreano and Mattiussi*, 2002, pg.27) le reti così formate non erano però «in grado di apprendere e i valori sinaptici delle loro connessioni dovevano essere prestabiliti dallo sperimentatore».

Il primo passo per l'apprendimento venne svolto da *Hebb* (1949) nel libro *The organization of behavior*. Data la sua formazione psicologica, il suo lavoro era incentrato sull'apprendimento umano e animale. Formulò la nota regola di Hebb che, sebbene non fosse strettamente legata al mondo delle reti neurali, fu di significativo studio da parte di molti scienziati e specialisti del settore: «se due neuroni collegati fra loro sono attivi contemporaneamente il valore sinaptico della loro connessione viene aumentato» (*Floreano and Mattiussi*, 2002, pg.27).

Grazie all'opera di *Rosenblatt* (1958), viene introdotto il procedimento di correzione iterativa dell'errore: la rete così creata, denominata *percettrone*, è in grado di apprendere e di migliorare nel tempo. Il suo studio è anche alla base delle moderne reti neurali, essendo volto all'analisi di funzioni quali l'immagazzinamento delle informazioni e il riconoscimento di pattern. *Widrow* e *Hoff* migliorarono il meccanismo di apprendimento, a cui diedero il nome di «Regola delta», permettendo di estendere l'uso del percettrone in più contesti. I limiti del percettrone vennero mostrati nell'opera *"An introduction to computational geometry"* di *Minsky and Papert* (1969): tramite procedimento matematico si dimostrò come le reti fino a quel periodo realizzate (single layer) non permettevano di svolgere funzioni logiche linearmente non separabili, quali ad esempio la funzione Xor³. Inoltre *Minsky* e *Papert* mostrarono come i computer disponibili all'epoca fossero incapaci di gestire il lavoro e l'elaborazione di

³f(0,0)=f(1,1)=0; f(1,0)=f(0,1)=1

grandi reti neurali.

L'opera di Minsky e l'avvento della robotica portarono al progressivo abbandono degli studi a riguardo delle ANN.

Il ritorno in auge delle reti neurali si ebbe grazie alle reti multistrato a seguito della tesi di dottorato di Paul Werbos e all'introduzione dell'algoritmo di retro-propagazione dell'errore (*error backpropagation*) proposto da Rumelhart *et al.* (1986). Il processo di *backpropagation* è una rielaborazione e miglioramento della funzione delta nello scenario di reti multistrato.

La disponibilità di computer più potenti, la realizzazione di reti multistrato e l'uso della backpropagation permise di risolvere funzioni non separabili linearmente e di rilanciare le reti neurali, al giorno d'oggi sempre più usate.

2.2 Rete neurale: caratteristiche e funzioni

La rete neurale artificiale deve il suo nome al tentativo di somigliare alla struttura neurale di un cervello e di emularne il processo di apprendimento, ossia, dato un input, ricevere buone o cattive ricompense a seconda dell'output emesso. Le similitudini però finiscono qui: la ANN è composta da neuroni (nodi di calcolo) e dal tipo di apprendimento supervisionato o non supervisionato; il resto non ha nulla in comune con una rete neurale naturale.

Il neurone è l'unità fondamentale delle ANN: si tratta di una equazione più o meno complessa che, dato un input, restituisce un valore in uscita in relazione ai dati precedenti immagazzinati e a quelli che si stanno aggiungendo.

Può essere rappresentato come in figura 2.1, avente X_i ingressi ed un'unica uscita.

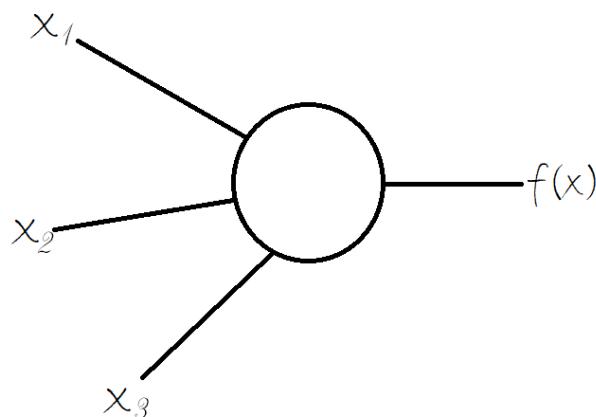


Figura 2.1: Rappresentazione di un neurone artificiale

2. Le reti neurali artificiali

La formula matematica esprimente il neurone rappresentato in figura 2.1 può essere similare a quella riportata nella formula 2.1.

$$f(x) = \sigma \left(\sum_i w_i g(x) + b(x) \right) \quad (2.1)$$

ove

- σ indica la funzione di attivazione del neurone;
- w_i indica il peso (**weight**) che ciascun ingresso ha sul neurone;
- $g(x)$ indica la funzione di ottimizzazione;
- $b(x)$ indica il *bias*.

Di tutti gli elementi riportati nella formula si tratterà approfonditamente di seguito.

Se il neurone, come detto, è l'unità fondamentale, l'iterazione fra i neuroni crea un ulteriore grado di variabili in gioco da tenere conto, con cui si concretizza la rete neurale.

Il primo parametro da valutare è sicuramente il numero di neuroni all'interno della rete che comporterà, al suo crescere, l'incremento dell'onere computazionale della rete. Il secondo parametro è il legame che intercorre fra gli stessi, che può essere:

- Progressivo (**Feedforward Neural Network**);
- Ricorsivo (**Recurrent Neural Network**).

Oltre ai due parametri appena scritti, l'esistenza o meno di strati di neuroni comporta la possibilità di una maggior o minor complessità della rete che non si esplica direttamente, nel caso di più strati, in un miglioramento del risultato finale, ma permette altresì di studiare più adeguatamente scenari molto complessi che un singolo strato non riuscirebbe a carpire: in questo caso si parlerà di *multi-layers*. Ovviamente, non avendo ancora introdotto un feedback, una struttura del genere porterebbe a non verificare mai la correttezza del risultato ottenuto: tale procedura modifica i pesi interni per ciascun neurone a seconda dell'errore che si genera fra i dati simulati e i dati voluti. Questo feedback può essere più o meno verificato dall'esterno: in tal caso si parlerà di *apprendimento supervisionato* altrimenti l'apprendimento sarà *non supervisionato*. L'apprendimento non avviene una sola volta ma può essere iterato: tali iterazioni vengono definite *epoches* e permettono di affinare i pesi sul set di dati di allenamento.

Per brevità, poiché non si tratta di un testo relativo alla spiegazione delle ANN ma una tesi ove rendere conto della complessità nell'applicazione pratica, tratteremo solo le parti utili alla comprensione della ANN realizzata nel capitolo 4.

2.3 Perceptrone

Per comprendere i meccanismi e toccare con mano la complessità di una rete neurale si è provveduto a realizzare "manualmente" un perceptrone: per farlo si è realizzato uno script in Python, il cui codice è fruibile in appendice C.1. Il sistema è composto da un singolo neurone che, tramite apprendimento supervisionato, determina in modo automatico se un punto nel piano cartesiano si trova al di sopra o al di sotto di una funzione assegnata. L'uso è puramente didattico ma ha permesso di apprendere i rudimenti del linguaggio Python e di comprendere meglio come è strutturata una rete neurale.

2.4 Struttura della rete

Come sopra accennato le parti necessarie alla realizzazione della rete neurale sono molteplici e a seconda dell'obiettivo da raggiungere possono essere differenti. In questo capitolo approfondiremo tutti gli iperparametri necessari per giungere alla previsione delle altezze idriche partendo dai dati disponibili in altre stazioni.

2.4.1 Single layer/multi layer

La scelta di una ANN single layer o multi layer è da definire a seconda dello scenario in esame.

Una rete neurale generica è costituita da tre tipologie di layers:

- Input layer: lo strato di neuroni legati ai dati di input, a seconda del quantitativo di dati in input esso conterrà lo stesso numero di neuroni. L'input layer è rappresentato in figura 2.2 col colore verde a sinistra;
- hidden layers: lo strato di neuroni nascosto, possono essere in numero qualunque, in figura 2.2 di colore blu;
- output layer: il numero di neuroni in questo layer deve essere uguale al numero che si vuole ottenere in output, in figura 2.2 di colore verde a destra;

2. Le reti neurali artificiali

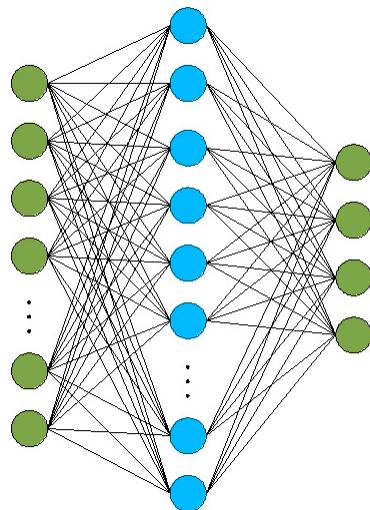


Figura 2.2: Rappresentazione di una rete neurale, in verde gli strati di input e output, in blu gli strati nascosti

Bisogna però spendere qualche parola riguardo agli *hidden layers*. Si definiscono tali tutti i layer compresi fra quelli di input e output: sono quindi "nascosti", cioè interni ed usati solo dalla rete stessa. In letteratura esistono numerose regole empiriche per la determinazione di quanti layers e quanti neuroni debbano contenere, ma è prassi comune partire da un numero congruo di layers e ridurne il numero fino a quando l'accuratezza dei dati in uscita non è più accettabile: normalmente un paio di hidden layers sono sufficienti per molti problemi. Tutte le librerie odierne costituiscono reti multi layer per sopperire ai problemi non lineari altrimenti insolvibili mostrati da Minsky.

Rimane la problematica di definire da quanti layer deve essere costituita la rete, quesito solitamente risolto dall'operatore per tentativi: nella costruzione dovrà sempre tenere conto che al crescere del numero di layers cresce l'onere computazionale, ma risulta anche più difficoltosa l'applicazione della backpropagation.⁴.

2.5 Apprendimento

2.5.1 Funzione di ottimizzazione e di costo

Funzione di costo La funzione di costo (loss function) è quella funzione che permette di tarare i pesi e i *bias* all'interno della rete confrontando i dati previsti con quelli reali. Esistono numerose funzioni e gli ambienti di sviluppo quali Tensorflow o Theano ne mettono a disposizione sempre nuove. La scelta fra

⁴Per approfondire: <https://www.heatonresearch.com/2017/06/01/hidden-layers.html>

una funzione e l'altra dipende dall'obiettivo e dalla tipologia di dati disponibili, quelle più note sono:

MSE Mean Square Error: errore quadratico medio. È la funzione più utilizzata e consiste nella media dei quadrati delle differenze fra il valore reale e quello previsto, in formula:

$$MSE = \frac{\sum_{i=1}^n (y_i - y_i^p)^2}{n} \quad (2.2)$$

L'unico inconveniente è che l'errore mostrato è il quadrato dell'unità di misura di partenza, può quindi trarre in inganno. Un'alternativa può essere il MAE.

MAE Mean Absolute Error: errore medio assoluto, espresso come

$$MAE = \frac{\sum_{i=1}^n |y_i - y_i^p|}{n} \quad (2.3)$$

La problematica con tale funzione è che non è ovunque derivabile, avendo un punto angoloso e la derivata, ove fattibile, presenta sempre il medesimo valore: quando, quindi, l'errore è piccolo la correzione apportata è la stessa di un errore grande. Entrambe le funzioni sono esemplificate in figura 2.3.

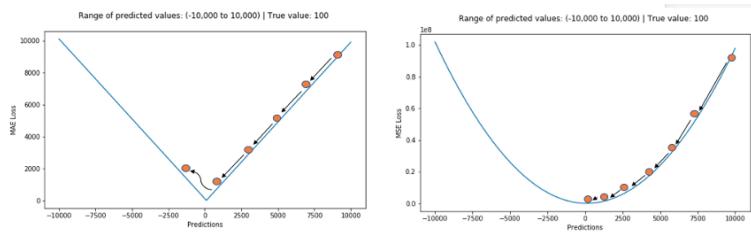


Figura 2.3: Esemplificazione delle funzioni MAE e MSE

Per sopperire ad entrambe le problematiche si è deciso di optare per la radice quadra dell'errore quadratico medio.

RMSE Root Mean Square Error: errore quadratico medio posto, sotto radice quadra:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (y_i - y_i^p)^2}{n}} \quad (2.4)$$

2. Le reti neurali artificiali

Con esso sono state determinate le scelte relative al numero di neuroni, il numero di layers e tutti gli iperparametri della rete.

Vi sono altre funzioni di ottimizzazione, tra cui:

- mean absolute percentage error function;
- mean squaredlogarithmic error function;
- cosine similarity function;
- huber function;
- log cosh function.

Funzione di ottimizzazione La funzione di ottimizzazione (optimizer function) permette di determinare quale possa essere la strada più rapida per giungere alla miglior soluzione per il proprio scenario. In Keras vi sono numerose funzioni di ottimizzazione (si veda il sito), in particolare:

- SGD;
- RMSprop;
- Adam;
- Adadelta;
- Adagrad;
- Adamax;
- Nadam;
- Ftrl.

2.5.2 Supervisionato o non supervisionato

L'apprendimento non supervisionato lascia alla rete l'onere di determinare se la risposta ottenuta è coerente o meno con quanto calcolato. Il compito della rete è quello di determinare opportune ricorsività usando funzioni probabilistiche: l'apprendimento non supervisionato trova quindi applicazione per la comprensione dei dati raccolti, come nei motori di ricerca.

L'apprendimento supervisionato consiste nel rendere disponibile alla rete anche l'output richiesto: è la metodologia usata per la realizzazione della rete nel capitolo successivo. Esso permette di tarare l'output del **ANN** in un range prestabilito e di ridurne l'eventuale errore, avendo un valore su cui confrontarsi. Può capitare che la rete venga allenata al punto da non essere più in grado

di generalizzare e di prevedere eventuali valori differenti da quelli appresi: in caso si dirà che la rete è in overfitting o sovra-adattata.

Underfitting e Overfitting Il sovra-adattamento e il sotto-adattamento sono problemi comuni nella gestione e nell'apprendimento dei dati per le reti neurali. In entrambi i casi si osserva un peggioramento dei risultati rispetto alla funzione di costo.

Si riporta una esemplificazione di questa problematica nella figura 2.4 Per superare questo limite è quindi necessario impostare nel codice quanta parte di dati è bene demandare al solo *validation set* per la rete stessa.

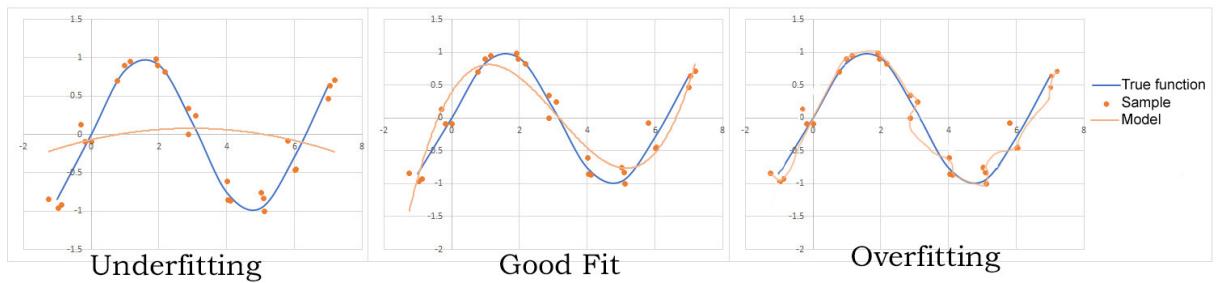


Figura 2.4: Esempio grafico underfitting e overfitting

Dataset Si definiscono a tal proposito all'interno dei dati a disposizione tre sottoinsiemi, citando (*Kuhn et al.*, 1996, pg. 354):

1. *Training set:* A set of examples used for learning, that is to fit the parameters of the classifier.
2. *Validation set:* A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network.
3. *Test set:* A set of examples used only to assess the performance of a fully-specified classifier.

I macro insiemi sono il *training set* e il *test set*. Nel *training set* parte dei dati, il cosiddetto *validation set*, è usata per verificare che l'allenamento effettuato e i pesi tarati siano corretti: è l'insieme di dati a cui la rete può far affidamento per migliorare i risultati. Il *test set* è semplicemente l'applicazione della rete su un set mai visto e di cui la rete deve "indovinare" l'output.

Variando il numero di dati nel *validation set* si riesce, ottimizzando la funzione di costo, ad evitare l'overfitting o l'underfitting. Un altro modo è interrompere l'allenamento (il numero di epoche) quando la funzione di regressione non

2. Le reti neurali artificiali

migliora più⁵. In letteratura si consiglia di impostare, per il *validation set*, un numero di dati compreso fra il 10 e il 20% del *training test*.

2.5.3 Back-propagation

Il metodo di *back-propagation*, come suggerisce il nome, permette di apportare la correzione dell'errore a ritroso nei vari layers e su ciascun neurone. È stato più volte scoperto e riscoperto ma solo con la più recente pubblicazione di Rumelhart prende il nome con cui oggi si identifica.

Il metodo di *back-propagation* è applicabile a tutte le reti neurali e corregge, come per la regola delta, i pesi di ciascun neurone in base alla funzione costo sopra espressa riportando a ritroso l'errore. Un'equazione semplificata che possa riassumere queste nozioni è stata presentata da *on Application of Artificial Neural Networks in Hydrology* (2000):

$$\Delta w_{ij}(n) = -\varepsilon \cdot \frac{\delta E}{\delta w_{ij}} + \alpha \cdot \delta w_{ij}(n-1) \quad (2.5)$$

ove:

- $\Delta w_{ij}(n)$ e $\Delta w_{ij}(n-1)$ sono l'incremento dei pesi fra i nodi i e j nell'epoca n-esima e n-1;
- ε e α sono rispettivamente learning rate e momentum. sono coefficienti specifici per la back-propagation che i nuovi ambienti di sviluppo tarano in autonomia, « *The momentum factor can speed up training in very flat regions of the error surface and help prevent oscillations in the weights. A learning rate is used to increase the chance of avoiding the training process being trapped in a local minima instead of global minima* » *on Application of Artificial Neural Networks in Hydrology* (2000)
- E è l'errore identificato all'epoca n .

2.5.4 Feedforward Neural Network

La tipologia di iterazione nella rete fra i neuroni di uno strato con quello successivo, come accennato, può essere di due tipologie:

1. Recurrent RNN;
2. Feedforward FNN.

⁵Le metodologie per tale operazione sono tante, quella utilizzata nel capitolo 4, ove si esporrà adeguatamente, è l'*Early stopping*.

Le prime sono costituite da interazioni fra gli stessi neuroni che ne permettono l'immagazzinamento delle risposte precedenti, così da essere in grado di determinare pattern ricorrenti come nel riconoscimento di immagini o nel riconoscimento vocale.

Le seconde sono costituite dalla progressione dell'informazione dal neurone del layer precedente a quello successivo senza punti ricorsivi all'interno della rete stessa.

2.6 Il neurone

Esposte le varie scelte a livello globale sulla rete si possono ora affrontare gli argomenti strettamente legati al singolo neurone.

2.6.1 Activation function

È quella funzione che determina l'uscita in base ai valori presenti in ingresso, nella equazione 2.1 si è identificata con il simbolo σ . All'interno del nodo di calcolo, gli input vengono dapprima moltiplicati per i relativi pesi e poi ne viene fatta la somma; il risultato entra nella funzione di attivazione che determina se e che tipo di output emettere. Non ogni valore in input a tale funzione deve corrispondere ad un valore in uscita, questa soglia dà il nome alla funzione, che, quindi, permette o meno di attivare il nodo.

Sigmoid function La prima funzione di attivazione è la funzione sigma, *sigmoid function*, aente formula:

$$y = \frac{1}{1 + e^{-x}} \quad (2.6)$$

Essa restituisce in output un valore compreso fra 0 e 1 ed è molto utilizzata nei casi di classificazione, e citando (Rashid, 2016, pg.62) « *is much easier to do calculations with than other S-shaped functions*», essendo continua e derivabile in ogni punto.

Altre funzioni come la tangente iperbolica restituiscono però risultati migliori rispetto alla funzione sigma, come suggerisce (Bengio et al., 2017, pg.195): « *the hyperbolic tangent activation function typically performs better than the logistic sigmoid.*»

La funzione sigma soffre del fatto che per valori alti, positivi o negativi, restituisce sempre l'unità con segno concorde, risultando altresì molto sensibile

2. Le reti neurali artificiali

quando il valore in input è prossimo allo zero. Si preferisce quindi usare funzioni lineari a tratti quali la funzione *ReLU*: essa permette di avere, inoltre, un output differente dall'unità.

ReLU function La funzione *ReLU* è una funzione lineare a tratti e può essere rappresentata come in figura 2.5, con un tratto nullo fino alla soglia e una retta obliqua indicante la proporzionalità diretta fra ingresso e uscita.

Essendo quasi lineare, la funzione *ReLU* permette di applicare al meglio le

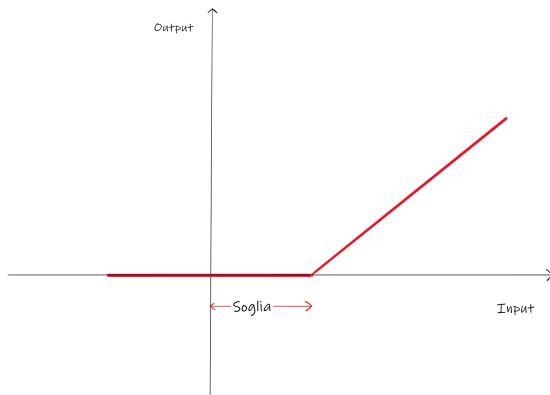


Figura 2.5: Esempio grafico della funzione *ReLU*

funzioni di ottimizzazione basate sul gradiente e di mantenere le proprietà consentite ai modelli lineari, cioè di avere una buona generalizzazione come riporta (*Bengio et al., 2017, pg.175*) «*Because rectified linear units are nearly linear, they preserve many of the properties that make linear models easy to optimize with gradient-based methods. They also preserve many of the properties that make linear models generalize well*».

In generale si possono identificare i seguenti punti a favore della funzione lineare:

- Semplicità computazionale, perché non è necessario alcun calcolo esponenziale;
- Fino alla soglia il valore è effettivamente zero, cosa che difficilmente avviene con le funzioni sigma o tan;
- Comportamento lineare, più facile da ottimizzare e che quindi rende possibile addestrare facilmente **ANN** multistrato.

Le funzioni di attivazione in keras sono molteplici e si rimanda alla loro libreria per maggiori informazioni:

- *ReLU function*;

- *sigmoid function;*
- *softmax function;*
- *softplus function;*
- *softsign function;*
- *tanh function;*
- *selu function;*
- *elu function;*
- *exponential function.*

2.6.2 Pesi sinaptici

I pesi sinaptici vengono inizializzati su un valore randomico e ad ogni epoca vengono corretti attraverso la funzione di costo. Se si vuole rilanciare la rete per eventuali migliorie o correzioni si deve tenere in considerazione che i valori iniziali dei pesi saranno differenti rispetto alle simulazioni precedenti e questo comporterà risultati differenti fra una simulazione e l'altra.

2.7 Epoche e Batch

Gli ultimi parametri da definire sono il numero di epoche e la batch. Per epoche si intende il quantitativo di iterazioni che la rete dovrà affrontare prima di ritenerla allenata.

La variabile *Batch* fa riferimento al numero di dati di *training set* di cui «*ad ogni passo i pesi vengono aggiornati utilizzando informazioni relative a tutti i campioni dell'insieme di addestramento*» (*Grippo and Sciandrone*, 2003, pg.19).

Si potrebbe ipotizzare di rendere queste due variabili più grandi possibili, così da allenare al meglio la rete: le batch ovviamente dovranno essere comprese fra 1 e il numero dei di *training set* mentre le epoche possono essere infinite. Purtroppo bisogna tenere conto che, seguendo tale strada, si andrà verso l'overfitting, per maggiori informazioni a riguardo dell'overfitting si può fare riferimento nel paragrafo 2.5.2.

2.8 Casi analoghi in idrologia e idraulica

Le reti neurali sono utilizzate in moltissimi campi, e si stanno affermando anche nel mondo dell'ingegneria idraulica. In questo capitolo si riportano alcuni

2. Le reti neurali artificiali

esempi. Nell'ambito idraulico e idrologico, le reti sono spesso usate per determinare l'andamento delle piene, partendo dall'andamento delle piogge afferenti al bacino in esame; fra le numerose pubblicazioni disponibili si cita, a solo esempio, quella di (Campolo *et al.*, 2003), riguardante la previsione dell'altezza d'acqua alla sezione di chiusura del bacino del fiume Arno. La configurazione risultata migliore, per quel contesto, è costituita da 57 nodi in input, 30 nodi nel layer nascosto e 6 per il layer di output. Il processo è stato lanciato 10 volte partendo da configurazioni randomiche dei pesi. La migliore finestra di previsione risulta corrispondente a 6 ore, avendo una perdita di accuratezza del 5% nel caso di previsione a 12 ore.

Con il progredire delle potenze di calcolo e il consolidamento delle nozioni riguardo alle ANN si è riusciti a costruire un sistema di previsione delle inondazioni partendo dalle stazioni pluviometriche nei bacini in studio: è il caso degli articoli Kia *et al.* (2012) e Kabir *et al.* (2020); il primo riguarda il fiume Johor situato in Malaysia, dove è stata generata una rete neurale sviluppata in ambiente Matlab e i suoi risultati, importati in GIS⁶, mostravano una buona concordanza con i risultati reali. La rete neurale in questo caso era composta da due strati nascosti, un numero di epoche non indifferente (2'000) ma che venivano interrotte nel caso di non miglioramento del risultato finale. Di tutti i dati disponibili il 20% venne destinato al *validation set*. Nel più recente articolo di Kabir *et al.* (2020) si fa riferimento all'uso del linguaggio di programmazione in Python: in esso si sono anche confrontate le potenzialità della rete neurale rispetto al sistema di equazioni alle acque basse fisicamente basato. Il caso studio è il River Eden nel Regno Unito, la rete neurale è stata costruita su GPU ed in 8 minuti fra apprendimento e simulazione è arrivata al risultato che per il sistema fisicamente basato è costato 92 minuti su CPU.

Altro campo di interesse è la modellazione dei processi di filtrazione nelle falde acquifere. L'università di Parma si è già avviata da tempo nello studio delle reti neurali per vari ambiti, non si può non citare le tesi di Daniele Secci "Applicazione di reti neurali per la simulazione surrogata di falde acquifere sotterranee" e la tesi di Varesi Marco "Apprendimento e predizione di dati fluviali tramite reti neurali", studi da cui questa tesi prende lo spunto per ampliare e migliorare alcuni aspetti.

⁶Geographic Information System

3

Caso di studio

Di seguito vengono sintetizzate le informazioni di carattere generale relative all'area di studio fornite nelle relazioni *Linee generali di assetto idraulico e idrogeologico nel bacino del Parma* e *Progetto di Piano stralcio per l'Assetto Idrogeologico (PAI)*, *Interventi sulla rete idrografica e sui versanti* redatte dall' Autorità di Bacino del Fiume Po (rispettivamente 2018 e 2013).

3.1 Area di studio: il torrente Parma

Il torrente Parma si sviluppa per circa 100 km e ha origine dai laghi Santo, Gemio e Scuro. Scorre in direzione nord-est, immettendosi in Po presso Mezzano Superiore. Il suo affluente principale, a sinistra, è il torrente Baganza, che si immette nel Parma in corrispondenza della città omonima. Si riporta in figura 3.1 l'ambito fisiografico del bacino del torrente Parma.

3.1.1 Aspetti idrologici

Il regime, come si evince dalle registrazioni agli idrometri, è di carattere pluviale di tipo torrentizio, con magre nel periodo estivo e forti piene nel periodo autunnale. La morfologia del bacino stesso, stretta e allungata, porta a eventi di piena molto accentuati e rapidi: il bacino si estende su un'area di 815 km^2 e presenta precipitazioni medie che variano da 800 mm/anno in pianura a 2000 mm/anno nelle zone del crinale appenninico come riportato da *Autorità del Fiume Po* (2018)(pg.110)

3. Caso di studio

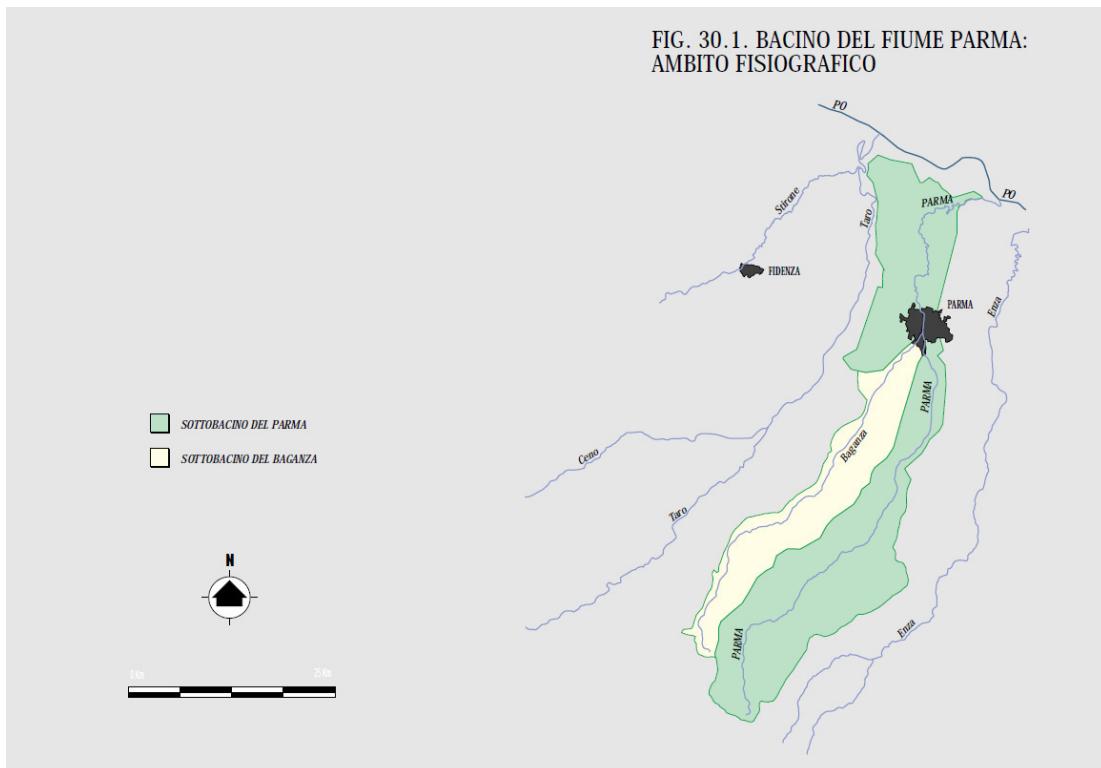


Figura 3.1: Inquadramento del bacino del Parma, figura estratta da: Linee generali dell'assetto idrogeologico e quadro interventi, bacino del Parma

3.1.2 Livello di protezione

Protezione passiva Le arginature a monte della città di Parma, come riportato da Autorità del Fiume Po (2003) (pg.74), risultano essere discontinue mentre la parte di città fino alla confluenza con il Po risulta essere arginata con l'ultimo tratto, a Colorno, più ristretto rispetto al resto dell'alveo. Come riporta anche l'Autorità del Fiume Po (2003), l'alveo, a valle di Parma, «è strettamente vincolato da opere di difesa e arginature».

Protezione attiva La realizzazione della cassa di espansione sul torrente Parma e il progredire della progettazione della cassa sul Baganza, entrambe poste a monte della confluenza in città, permettono attivamente la regimazione degli eventi intensi, potendo invasare e laminare le piene più gravose. È ovvio che una corretta manovra degli organi di regimazione consenta la miglior ottimizzazione nella laminazione nelle casse stesse.

Tale regolazione però risulta ottimale conoscendo come evolverà la piena nelle ore successive, previsione non facile da realizzare. È altresì vero che le simulazioni con sistemi fisicamente basati permetterebbero, attraverso vari scenari, la

3.2. Dati utilizzati: i dati reali

determinazione delle tempistiche di intervento: l'onere computazionale di tali sistemi sarebbe superiore al guadagno ottenuto e quindi risulta poco efficiente seguire tale via. È in questo solco che si inserisce la presente tesi, nel tentativo di sopperire in altri modi al calcolo dei parametri necessari a mettere in sicurezza il corso d'acqua.

Ad onor del vero, date le difficoltà incontrate e alla necessità di realizzare in prima battuta uno scenario, seppur complesso, il più gestibile possibile, la regione in studio si è ridotta all'ultimo tratto del torrente Parma, più precisamente dalla stazione idrometrica di Ponte Verdi fino alla cittadina di Colorno, ove è presente un'altra stazione di rilevamento delle altezze idriche.

Questa contrazione si è resa necessaria per ridurre le variabili in ingresso e semplificare il sistema realizzato rimanendo fedeli alla realtà degli eventi.

3.2 Dati utilizzati: i dati reali

I dati storici sono stati scaricati dall'archivio informatico dell'AIPO, sulle stazioni AIPO di Parma Ponte Verdi (nelle legende sintetizzata in P.Verdi), Colorno e in successivo momento anche Casalmaggiore (nelle legende sintetizzata in Cas.mag.).

L'archivio è reperibile¹ al sito <https://idrometri.agenziapo.it/Aegis/map/map2d>: il sito presenta un'interfaccia grafica che permette di selezionare, una volta effettuato il login, la stazione d'interesse e di estrarne i dati con la discretizzazione temporale desiderata. Un altro sito per il reperimento dei dati idrometrici è Dext3r dell'ARPAE Emilia-Romagna, raggiungibile al sito <https://simc.arpae.it/dext3r/>: l'unica limitazione è data dalla discretizzazione temporale fissa a 30 minuti dei dati scaricati.

Fra le due alternative, si è optato per il sito dell'AIPO per un maggior quantitativo di dati disponibili ed una maggiore facilità di estrazione. Si è notato che la stazione più recente, quella di Colorno, è attiva dal 17/10/2011. Per completezza si è quindi deciso di utilizzare i dati dal 2012 al 2019 compresi.

I dati reperiti sono in formato .xlsx e hanno struttura simile a quella riportata in tabella 3.1, hanno discretizzazione ogni 10 minuti.

¹ultima verifica in data 14/01/2021

3. Caso di studio

Tabella 3.1: Dati scaricati da Aegis

Orario	Parma Ponte Verdi Livello Idrometrico - 23050 (m)
01/01/2018 00:00	0.49
01/01/2018 00:10	0.48
01/01/2018 00:20	0.47
01/01/2018 00:30	0.49
01/01/2018 00:40	0.48
01/01/2018 00:50	0.49
01/01/2018 01:00	0.48
01/01/2018 01:10	0.47
01/01/2018 01:20	0.49
01/01/2018 01:30	0.49
01/01/2018 01:40	0.48
01/01/2018 01:50	0.49
01/01/2018 02:00	0.48

I dati scaricati sono stati processati con uno script in Matlab (in appendice B.1). Esso si è reso necessario per vari scopi:

- coprire tramite semplice interpolazione i dati mancanti, di cui un esempio è riportato in figura 3.2;
- discretizzare ed unificare i dati con un unico passo di 30 minuti.

Le modifiche si sono svolte nell'ordine presentato per migliorare l'accuratezza dei dati in ingresso alla rete neurale.

Interpolazione L'interpolazione si è svolta tramite funzione accessoria di cui si riporta di seguito il punto centrale:

Matlab script in appendice B.2

```
hidro(i) = y1+((y2-y1)*(i-t)/(T-t));
```

ove:

- hidro(i) sta ad indicare l'i-esimo valore mancante nelle quote idriche;
- y1 la più prossima quota idrica precedente;
- y2 la più prossima quota idrica successiva;
- t la distanza temporale fra il valore mancante e quello precedente;
- T la distanza temporale fra il valore mancante e quello successivo.

3.2. Dati utilizzati: i dati reali

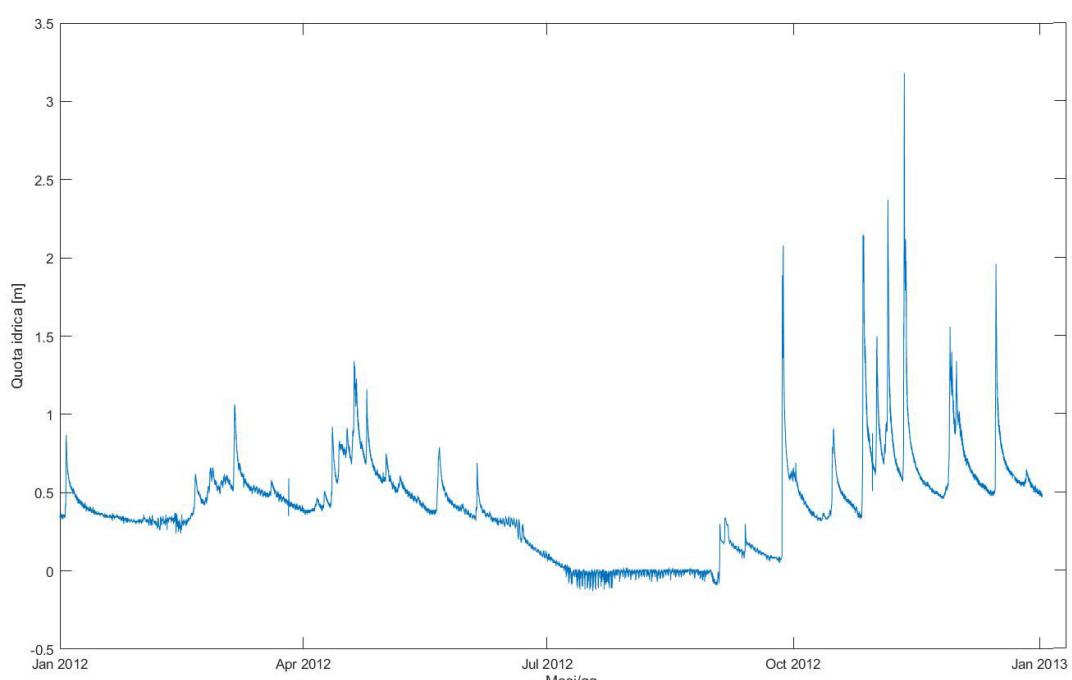
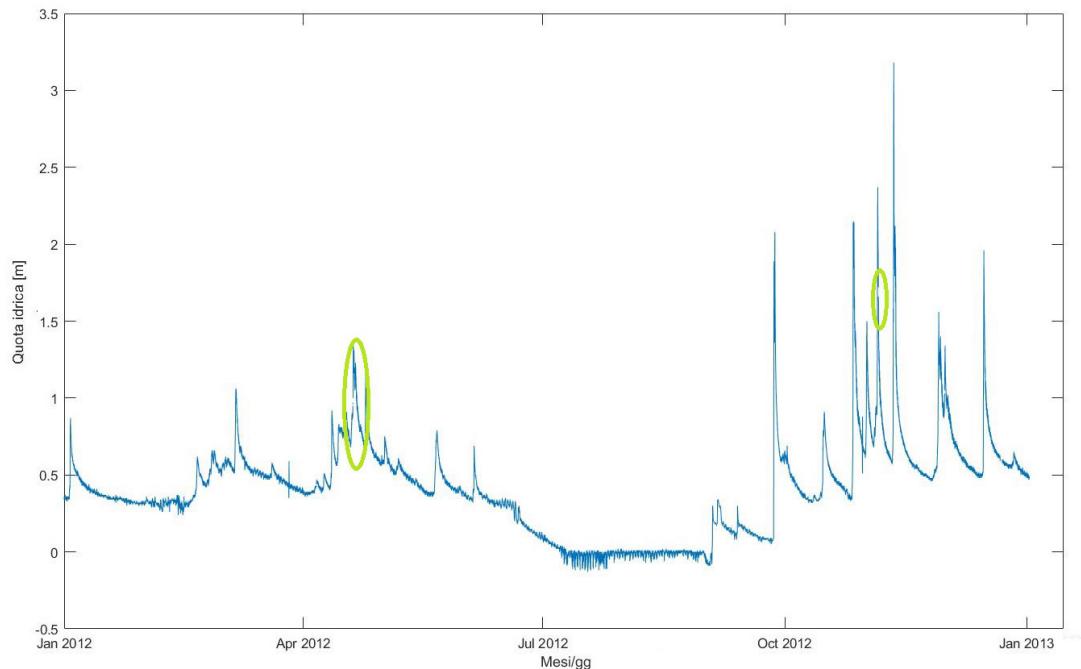


Figura 3.2: Ricucitura dei punti mancanti, esempio sull'anno 2012

3. Caso di studio

Discretizzazione La discretizzazione con passo di 30 minuti è stata scelta per uniformare tutte le stazioni. Lo script completo è presente in appendice, mentre lo stralcio è riportato di seguito:

Matlab script in appendice B.1

```
if mn==0 || mn==30  
dat_(j)=date(i);  
hidroC_(j)=hidroC(i);  
hidroPV_(j)=hidroPV(i);  
j=j+1;
```

Inserito in un ciclo for su tutto l'anno in esame in quel momento.

Il prodotto finale dello script è riportato in tabella 3.2 e consiste in un file .txt contenente la misurazione nelle tre stazioni con discretizzazione di 30 minuti.

Tabella 3.2: Output script di Matlab

Orario	Parma Ponte Verdi	Casalmaggiore	Colorno
01/01/2012 00:00	0.36	-3.42	0.76
01/01/2012 00:30	0.35	-3.45	0.75
01/01/2012 01:00	0.35	-3.42	0.76
01/01/2012 01:30	0.36	-3.43	0.76
01/01/2012 02:00	0.36	-3.44	0.75
01/01/2012 02:30	0.35	-3.43	0.74
01/01/2012 03:00	0.35	-3.44	0.74
01/01/2012 03:30	0.35	-3.43	0.75
01/01/2012 04:00	0.36	-3.43	0.77
01/01/2012 04:30	0.35	-3.42	0.74
01/01/2012 05:00	0.36	-3.44	0.74
01/01/2012 05:30	0.35	-3.44	0.75
01/01/2012 06:00	0.35	-3.43	0.75
01/01/2012 06:30	0.35	-3.42	0.76
01/01/2012 07:00	0.35	-3.42	0.75
01/01/2012 07:30	0.36	-3.44	0.75
01/01/2012 08:00	0.34	-3.45	0.75
01/01/2012 08:30	0.35	-3.43	0.75
01/01/2012 09:00	0.36	-3.44	0.76
01/01/2012 09:30	0.37	-3.43	0.75

L'aggiunta dei dati di Casalmaggiore sul fiume Po (nelle legende sintetizzato con la sigla Cas.mag.) ai dati in input si è resa necessaria in un secondo momento, come spiegato nel capitolo 4. Si è scelto inoltre di non procedere ad ulteriore manipolazione dei dati in input per restare il più possibile coerenti con lo scenario in cui si vuole applicare la rete, cioè un uso in tempo reale dei dati disponibili nelle tre stazioni.

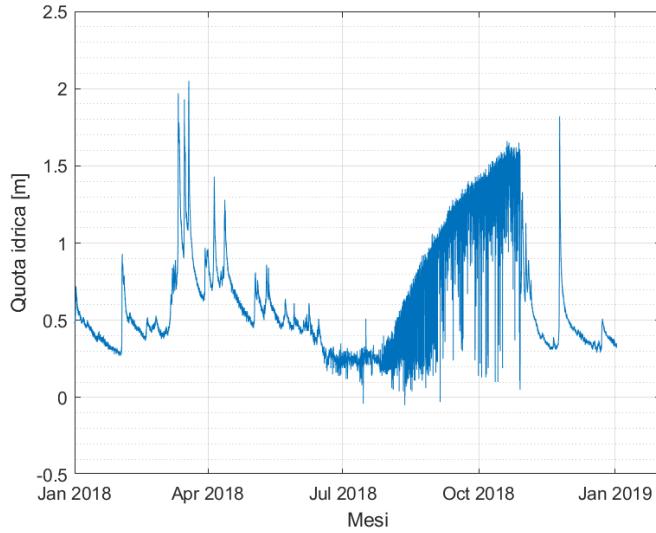


Figura 3.3: Quota idrica a Parma Ponte Verdi

3.2.1 Verifica dei dati

L'eliminazione delle magre dal set di dati si è resa necessaria per migliorare la rete neurale: volendo avere un focus maggiore sulle piene, sarebbe risultato inutile e superfluo, o addirittura controproducente, allenare la rete a prevedere i valori più bassi dei livelli idrici. I periodi di magra sono stati individuati a partire dai dati del Parma rilevati a Ponte Verdi. Si è inoltre notato che il sensore a Parma Ponte Verdi nel giugno 2018 ha subito un malfunzionamento per cui si è proceduto alla eliminazione dalla serie storica di tale periodo per tutte le stazioni: se ne riporta l'andamento in figura 3.3 prima dell'eliminazione dei mesi da giugno ad ottobre.

I risultati finali sono visionabili nelle figure in appendice A di cui un esempio è riportato alla figura 3.4.

3. Caso di studio

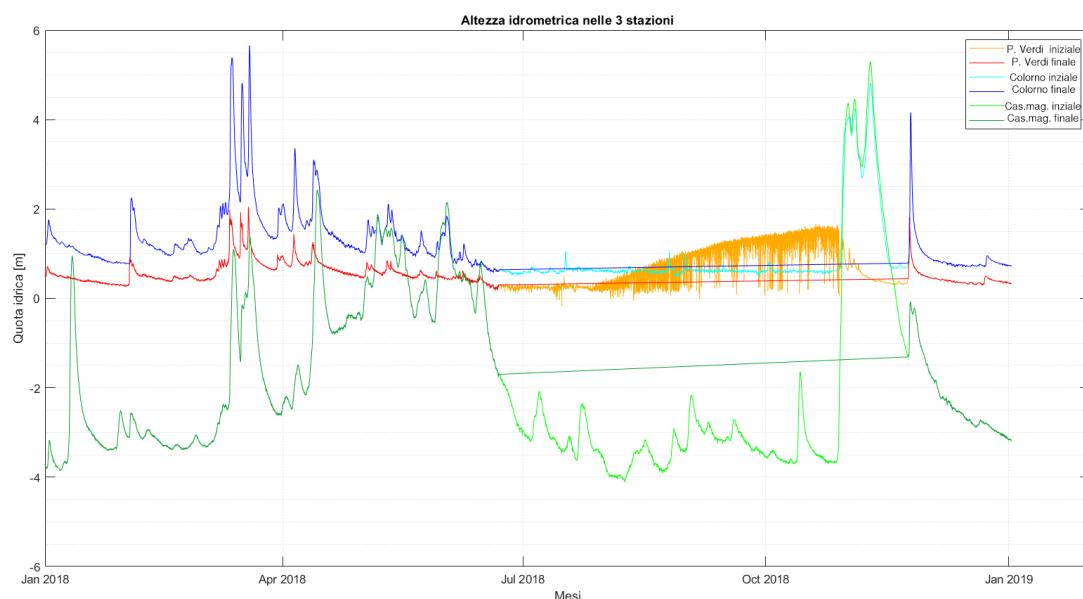


Figura 3.4: Andamenti salvati nelle tre stazioni nei differenti anni.
In legenda P.Verdi = Ponte Verdi, Cas.mag. = Casalmaggiore

4

La rete neurale realizzata

In questo capitolo vengono descritte le procedure utilizzate per la realizzazione della rete neurale avente come obiettivo la previsione delle onde di piena in tempo reale sul torrente Parma nel tratto compreso tra l'omonima città e l'attraversamento dell'abitato di Colorno.

Nella prima parte si mostrerà la struttura della rete e come sono stati rielaborati i dati di input per arrivare alla struttura finale; nella seconda parte si mostrerà quanto fatto per permettere la previsione dei dati futuri della stazione di Colorno. Le previsioni saranno svolte con orizzonti temporali dalle 3 alle 12 ore, mostrando la previsione di piene significative rispetto al dato reale.

Nota Per la scelta dei parametri e della struttura della rete si farà riferimento al coefficiente RMSE esposto nella formula 2.4. La libreria Keras non offre tale funzione fra quelle di costo disponibili. È stato pertanto necessario definire in modo esplicito tale funzione all'interno del codice:

Funzione RMSE in Python

```
def rmse(predictions , targets):  
    differences = predictions - targets # Differenza.  
    differences_squared = differences ** 2 # Quadrato della differenza  
    mean_of_differences_squared = differences_squared.mean() #Media  
    rmse_val = np.sqrt(mean_of_differences_squared) # Radice quadra  
    return rmse_val #Valore output
```

Funzione richiamata nel codice della rete neurale

4. La rete neurale realizzata

4.1 Librerie usate

Per la realizzazione pratica si è fatto riferimento al linguaggio di programmazione Python e a diverse librerie che hanno permesso di velocizzare e semplificare la realizzazione della rete:

1. *Pandas* e *Numpy* sono librerie per i calcoli matematici quali, a mero esempio, la media, la deviazione standard e le operazioni matriciali;
2. *Keras* è una libreria utilizzata per l'effettiva realizzazione della rete neurale;
3. *Matplot lib* ha permesso velocemente di porre in forma grafica i risultati ottenuti.

Eventuali ulteriori librerie sono state utilizzate occasionalmente, ad esempio, per tenere traccia del tempo impiegato.

4.2 Rielaborazione dati reali per la rete neurale

I dati in input elaborati dallo script in Matlab sono riportati in tabella 3.2 e sono suddivisi per anni. Il primo passaggio è stato quello di unirli in un'unica matrice riportante nella prima colonna la data, nella seconda i livelli idrometrici a Ponte Verdi e nella terza quelli a Colorno. In un secondo momento sono stati aggiunti nella quarta colonna anche i livelli idrometrici a Casalmaggiore. La colonna relativa alle date è stata rielaborata come semplice variabile testo (*string*).

Ultima doverosa premessa prima di procedere alla spiegazione del codice è che i dati input, come già detto, sono discretizzati ogni 30 minuti.

4.2.1 Elaborazione dati input

Si è definito il vettore x come insieme dei dati in input alla rete neurale, mentre il vettore y come vettore costituito dai dati output per quanto concerne sempre la ANN (si veda la tabella 4.1).

Per comprendere al meglio quanto fatto ipotizziamo di voler prevedere l'istante t con n valori in input precedenti a quell'istante. Lo script realizzato partendo dall'istante t prende a ritroso n valori nel vettore x e li traspone da formato colonna a formato righe. Al tempo $t+1$ la trasposizione avverrà partendo a ritroso da un valore successivo a x_n , cioè x_{n+1} . Si creerà quindi una matrice I formata dalla trasposizione e ripetizione del vettore x ; nella tabella 4.1 si riporta un

4.2. Rielaborazione dati reali per la rete neurale

esempio della creazione della matrice I . È ovvio che $t \geq n$ e da come è strutturato lo script si avrà inoltre che il numero di righe della matrice creata I e quelle del vettore y sarà il medesimo.

Il procedimento deve chiaramente essere ripetuto se sono presenti più di una serie temporale nei dati in input.

Tabella 4.1: Rielaborazione dati in input esempio con 10 valori

vettore x	vettore y	time	Matrice input I					vettore output	time
x_1	y_1	1						y_5	5
x_2	y_2	2						y_6	6
x_3	y_3	3						y_7	7
x_4	y_4	4						y_8	8
x_5	y_5	5						y_9	9
x_6	y_6	6						y_{10}	10
x_7	y_7	7							
x_8	y_8	8							
x_9	y_9	9							
x_{10}	y_{10}	10							

4.2.2 Elaborazione dati output

Fin ora la previsione come si può notare dalla tabella 4.1 è nulla, o meglio, i valori a Colorno (vettore y) all'istante t sono determinati dai valori nello stesso istante della matrice I . In generale quindi per prevedere il valore atteso tra T ore bisognerà effettuare $2T$ step in avanti. Per fare questi step si è optato per il "traslare verso l'alto" il vettore y tanto quanti sono gli step da effettuare: per prevedere ad un'ora in avanti gli step necessari saranno due. Concretamente sono stati eliminati i primi $2T$ valori nel vettore y e le ultime $2T$ righe nella matrice input I rielaborata come mostrato sopra.

In tabella 4.2 viene mostrato l'esempio di uno scostamento di un'ora per semplicità

La matrice input $I = n \times r$, supponendo la lunghezza dei vettori iniziali pari a N , avrà quindi dimensioni finali pari a:

- n colonne corrispondente al numero di dati con cui si vuole inizializzare la rete;
- il numero di righe pari a $r = N + 1 - (2T + n)$.

4. La rete neurale realizzata

Tabella 4.2: Rielaborazione dati in output esempio con 10 valori

Matrice input I					vettore output	time
x_1	x_2	x_3	x_4	x_5	y_7	7
x_2	x_3	x_4	x_5	x_6	y_8	8
x_3	x_4	x_5	x_6	x_7	y_9	9
x_4	x_5	x_6	x_7	x_8	y_{10}	10
x_5	x_6	x_7	x_8	x_9		
x_6	x_7	x_8	x_9	x_{10}		
Matrice input I					vettore output	time
x_1	x_2	x_3	x_4	x_5	y_7	7
x_2	x_3	x_4	x_5	x_6	y_8	8
x_3	x_4	x_5	x_6	x_7	y_9	9
x_4	x_5	x_6	x_7	x_8	y_{10}	10

4.3 Finestra di previsione

In un secondo momento è stata valutata l'ampiezza della finestra di previsione entro la quale ricavare i dati necessari. Si è notato come per diciotto ore le piene si sviluppassero quasi completamente (si veda un esempio nella figura 4.1) e la finestra è stata quindi fissata a quel valore. Nel paragrafo 4.8.3 è stata comunque portata avanti un'indagine sui miglioramenti ottenibili incrementando tale periodo. Tale finestra è identificata nello script con la variabile $temp$. Identificare una serie temporale per la previsione è come considerare che al tempo generico t_0 si voglia conoscere il valore in output al tempo $t_0 + m$, con m ore di previsione futura: se si fissa la finestra a 18 ore, allora la serie in input sarà formata da $18 - m$ valori. Per come è stato costruito lo script significa che le n colonne di cui è costituita la matrice input I saranno dipendenti dalla variabile $temp$, concretamente $n = (temp - m) \cdot 2$, essendo i dati $\Delta T = 30$ min.

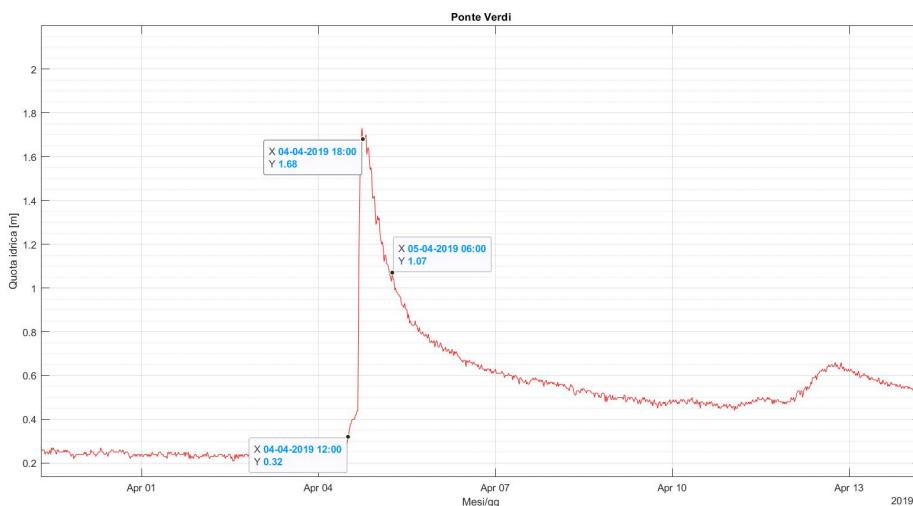


Figura 4.1: Finestra di previsione a 18 ore, onda del 04/04/2019 alla stazione idrometrica Ponte Verdi

4.4 Training set e test set

Come dichiarato nel capitolo 3 la serie storica delle altezze idrometriche copre il periodo temporale 2012 - 2019, estremi compresi; di questi anni, il 2018 e il 2019 sono stati considerati solo per il *test set*.

Tabella 4.3: Distribuzione dei dati iniziale

Stazione	Data set	
	Training	Testing
Ponte Verdi	range -0.31 ÷ 4.83	range 0.09 ÷ 2.23
Colorno	range 0.49 ÷ 9.49	range 0.58 ÷ 6.66

4.5 Prima elaborazione

La rete neurale è stata modificata e aggiornata spesso per ottenere valori della funzione di costo i più bassi possibili.

Il primo processo è stato quello di implementare una rete funzionante in grado di prevedere le altezze idrometriche a Colorno, considerate note quelle a Ponte Verdi a Parma. In prima analisi la previsione effettuata era a tempo zero, cioè le quote idriche della stazione di Colorno erano stimate utilizzando come input le quote alla stazione di Ponte Verdi sino al medesimo istante. Per ottenere i primi risultati si è fatto riferimento alle caratteristiche della rete neurale realizzata nella tesi di Varesi *Apprendimento e predizione di dati fluviali tramite reti neurali*. La rete è stata inizializzata con quattro layers nascosti i cui parametri sono riportati nella tabella seguente:

Hidden Layers	Neuroni	Funzione attivazione
1	64	<i>ReLU</i>
2	256	<i>sigmoid</i>
3	128	<i>sigmoid</i>
4	64	<i>ReLU</i>

Una prima analisi con questa configurazione ha determinato un **RMSE** di circa 50 cm.

I risultati, anche se non propriamente incoraggianti, mostrano comunque il corretto funzionamento della struttura e la possibilità di correzioni e miglioramenti.

4. La rete neurale realizzata

4.6 Seconda elaborazione

Si è notato come, riducendo drasticamente il numero di layers e di neuroni, i risultati ottenuti erano i medesimi, si è quindi optato per ridurne il numero, alleggerendo l'onere computazionale.

Si sono fatte numerose simulazioni provando a tenere tutte le variabili (batch, epoche e la finestra di osservazione) contemporaneamente in conto. Il risultato sono state 512 simulazioni effettuate scegliendo come funzione di attivazione la *ReLU*, adattata anche nelle simulazioni successive.

Nella configurazione migliore si è ottenuto un RMSE pari a 0.496 [m].

Nelle figure 4.2 si riportano le previsioni di due onde nella stazione idrometrica di Colorno, la prima appartenente al *training set* la seconda al *test set*. Le previsioni sono state effettuate utilizzando i parametri:

Finestra	15
Batch	15
Epoche	70
Neuroni	34

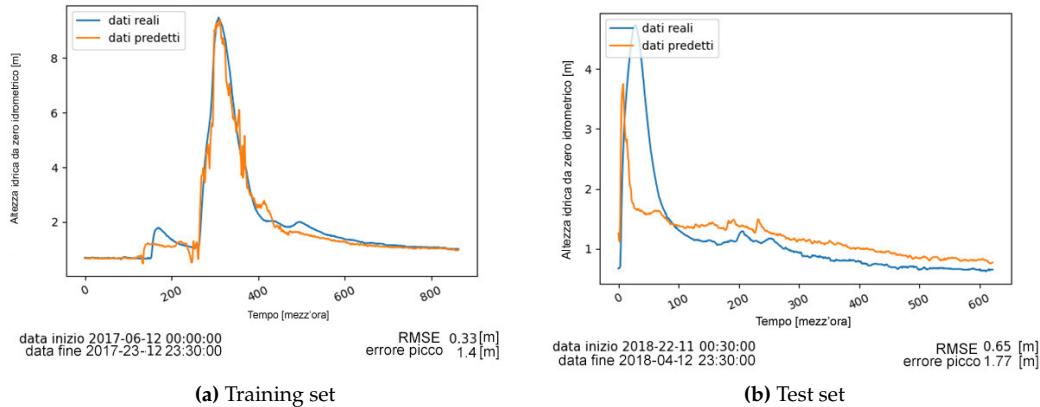


Figura 4.2: Previsione onde del 12/2017 nel training set e 11/2018 nel test set

Si può notare da entrambe le figure 4.2 come la rete non riesca a simulare correttamente le onde. I problemi più evidenti sono due: le oscillazioni non fisiche che si riscontrano sia nel ramo di risalita, che in quello di discesa non solo nel *training set*, ma anche nel *test set*; il secondo è l'incapacità della rete di simulare alcune onde, come quella in figura 4.2(b).

Si è notato, infatti, come alcune onde nella stazione di Colorno non dipendano solo dalle altezze idrometriche registrate a Ponte Verdi a Parma, ovvero dalle portate transitanti nel torrente Parma, ma anche dall'andamento dei livelli del Po. La sezione di Colorno si trova infatti ad una distanza dalla confluenza del

torrente Parma in Po sufficientemente breve da risentire dell'effetto di rigurgito del Po, quando in esso sta transitando un'onda di piena. Si notino, a tal proposito, le onde riportate in appendice A.

4.7 Terza elaborazione

Si è reso necessario rielaborare i dati in input per ottenere risultati più accurati.

4.7.1 Aggiunta dei dati da Casalmaggiore

Come accennato, l'aggiunta dei livelli idrometrici del Po alla stazione di Casalmaggiore¹ è essenziale per permettere alla rete di prevedere le altezze idrometriche a Colorno anche in quegli eventi in cui Parma e Po erano in piena contemporanea.

La distribuzione dei dati con Casalmaggiore fra training set e test set è riportata in tabella 4.4.

I dati precedenti sono stati fissati pari a 36 (18 ore) e le epoche pari a 48.

Tabella 4.4: Aggiunta dei dati di Casalmaggiore

Stazione	Data set	
	Training	Testing
Ponte Verdi	range -0.31 ÷ 4.83	range 0.09 ÷ 2.23
Colorno	range 0.49 ÷ 9.49	range 0.58 ÷ 6.66
Casalmaggiore	range -4.33 ÷ 7.01	range -4.5 ÷ 6.99

Le ricerche hanno riguardato:

- Il numero di neuroni, da 40 a 48 con passo 2;
- La batch, da 40 a 48 con passo 2.

La funzione di costo si è attestata come media a 0.184 [m] ed il valore minimo pari a 0.157 [m] si è ottenuto con 40 neuroni e 48 batch.

L'aggiunta dei dati di Casalmaggiore ha permesso di aumentare di molto la capacità della rete di prevedere i livelli a Colorno.

Si è quindi deciso di implementare nello script la possibilità di definire le previsioni future di 3/6/9/12 ore, come spiegato nel capitolo 4.3.

Il numero totale di simulazioni è stato pari a 125 e il valore più basso di RMSE era, come prevedibile, quello della previsione a 0 ore, già riportato in figura 4.1. Nelle figure 4.4 e 4.5 sono riportate le medesime onde nei 4 orizzonti di previsione (3/6/9/12 ore).

4. La rete neurale realizzata

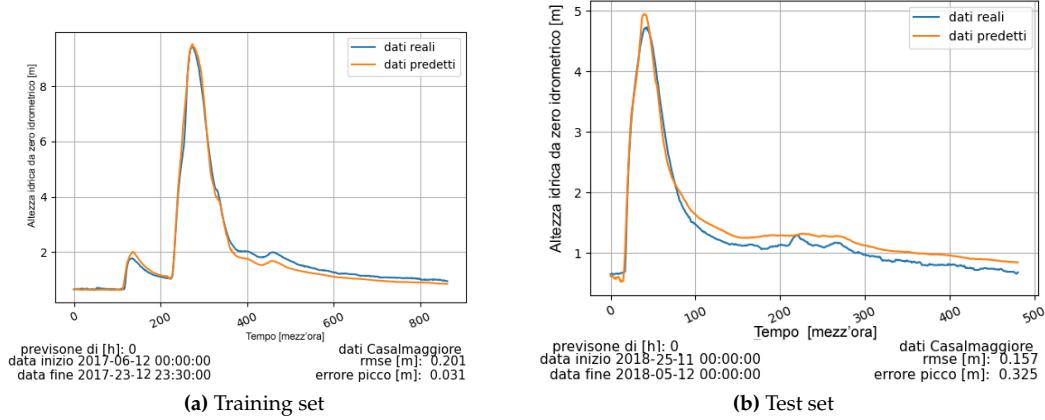


Figura 4.3: Previsione onde del 12/2017 e 11/2018; aggiunta dei dati da Casalmaggiore

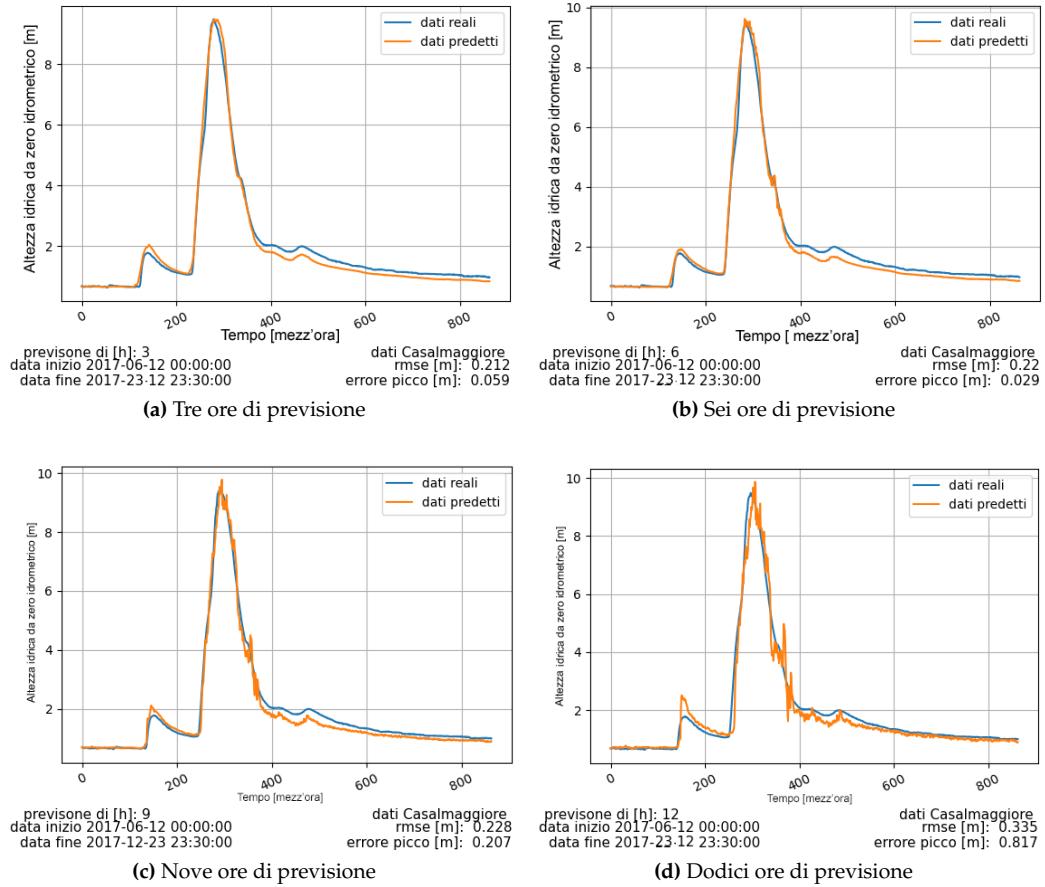


Figura 4.4: Previsione onda del 06/12/2017 del training set a diverse ore, input Casalmaggiore

¹La stazione idrometrica di Casalmaggiore si trova pochi chilometri prima della confluenza col Parma.

4.7. Terza elaborazione

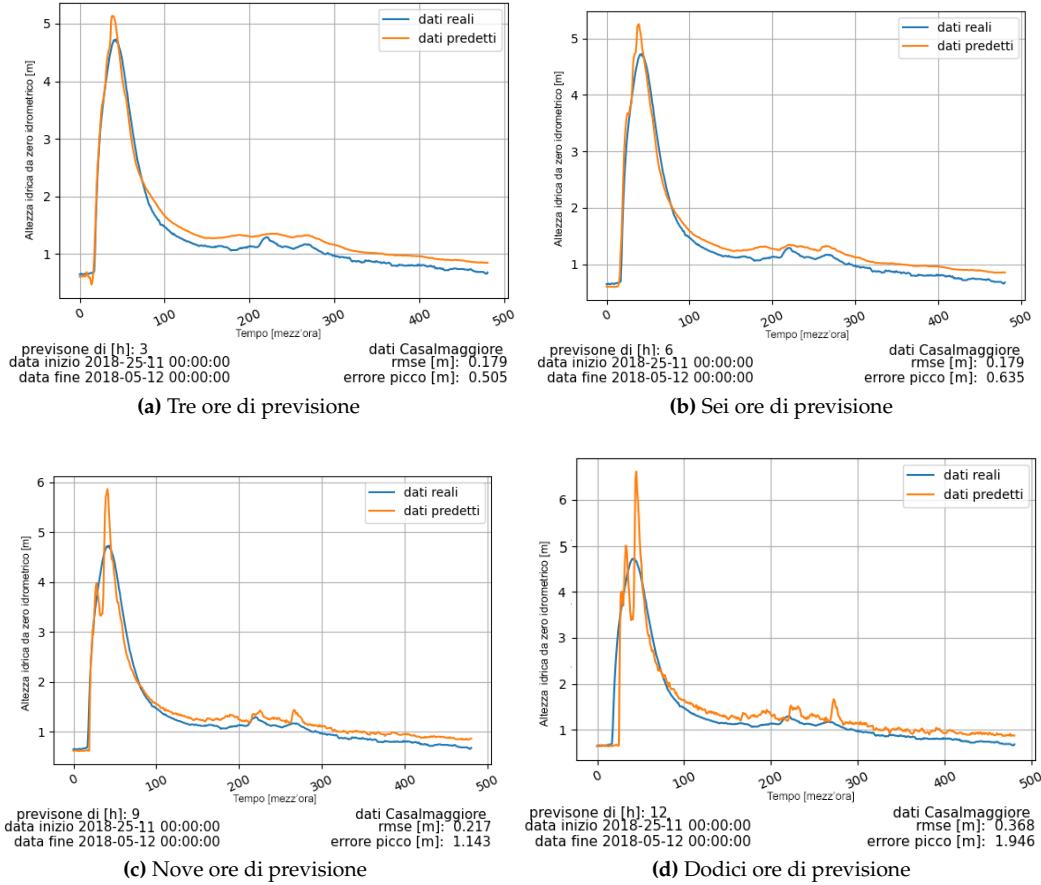


Figura 4.5: Previsione onda del 25/11/2018 del test set a diverse ore, input Casalmaggiore

Si è notato però che alcune onde con l'aumentare delle ore di previsione presentavano la stessa problematica: la presenza di oscillazioni di breve periodo sul ramo di discesa e anche sul picco, si riportano ad esempio come l'onda a 12 di ore previsione del 2017 (figura 4.4) o l'onda del 2018 sempre con 12 ore di previsione (figura 4.5). Si è provato anche ad ottimizzare batch e neuroni per ogni orizzonte di previsione, ma non si sono ottenuti miglioramenti significativi.

4.7.2 Aggiunta dei dati della stazione di Colorno in input

Poiché 3 ore risulterebbero insufficienti per quasi tutte le operazioni di salvaguardia, si sono considerati gli orizzonti di previsione a 6/9/12 ore. Si possono quindi fornire in input alla rete tutti i livelli idrometrici osservati a Colorno fino all'istante t_0 e fare una previsione al tempo $t_0 + m$ dove $m = 6/9/12$ ore. L'aggiunta di dati in input coerenti con il problema rende più robusta la rete e

4. La rete neurale realizzata

permette di ottenere risultati più accurati. Sono state quindi effettuate 45 simulazioni, per definire il set di parametri della rete migliori, riportate in tabella 4.5.

Tabella 4.5: Risultati di previsione con Colorno in input

Ore previste	Dati precedenti	Batch	Epochs	Neuroni	RMSE [m]
0	36	44	48	42	0.00514
3	30	44	48	42	0.07041
6	24	44	48	42	0.16398
9	18	44	48	42	0.22299
12	12	44	48	42	0.33295

Rilanciando la simulazione si è notato come l'RMSE con previsione a 0 ore risultasse pari a 0.00691 [m], valore molto basso e praticamente analogo all'errore nullo.

Alcune delle onde simulate sono riportate nella figura 4.6

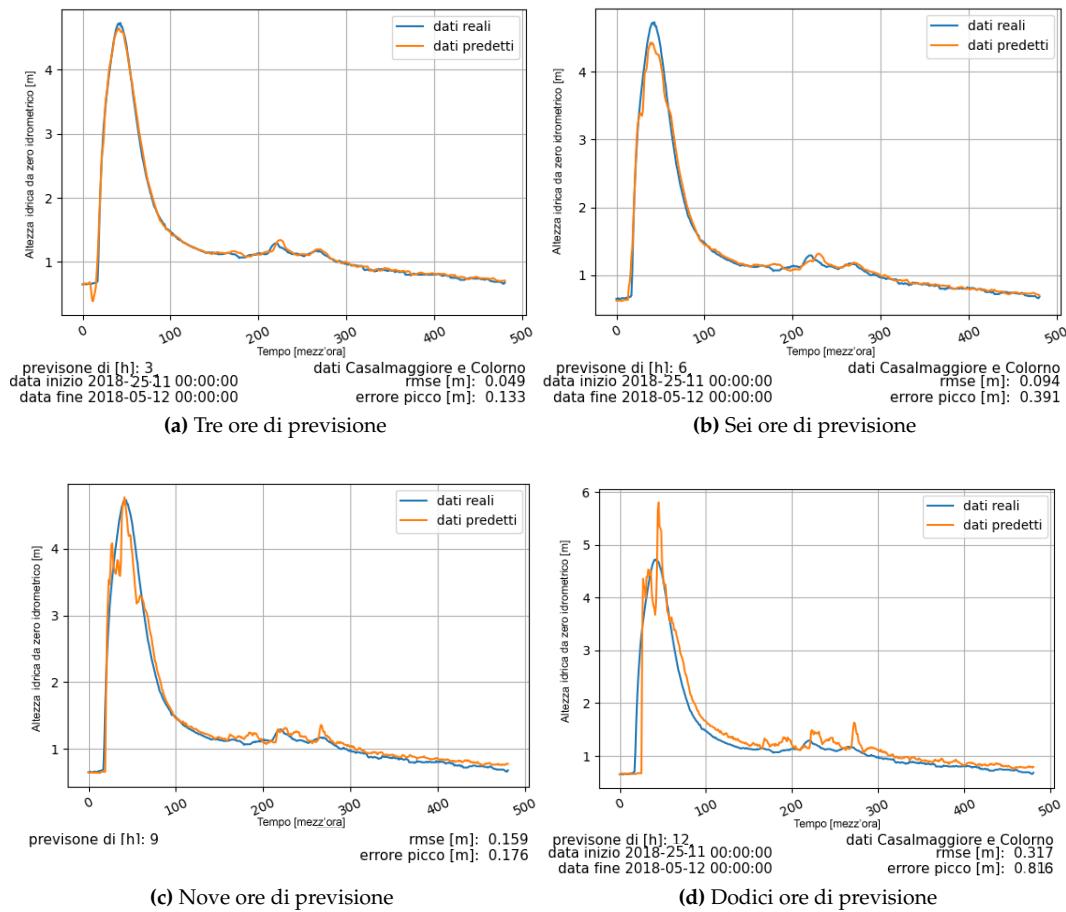


Figura 4.6: Previsione onda del 25/11/2018 del test set a diverse ore, input Casalmaggiore e Colorno

Tabella 4.6: Validation set, varie prove

Periodo testato	% dati di training	RMSE [m]
01/06/2012 - 22/11/2012	5	0.108
01/06/2012 - 01/02/2013	10	0.1
09/06/2013 - 22/03/2014	20	0.1055
09/06/2013 - 09/10/2014	34	0.1105

4.8 Ottimizzazione della rete

L'aggiunta dei dati dalla stazione di Colorno, permettono una buona previsione. Si rende quindi necessario migliorare la rete per restituire valori previsione a 3, 6, 9 e 12 ore affidabili e ripetibili.

4.8.1 Implementazione validation set

La prima implementazione effettuata è stata quella di definire correttamente il *validation set* utile a risolvere il problema del *overfitting*. Come già dichiarato nel paragrafo 2.5.2, è consigliabile utilizzare dal 10 al 20 % dei dati in input.

Nel caso in esame, si è scelto di estrarre una porzione di dati ininterrotti dal dataset di training.

La regola empirica definisce il range di *validation set* pari a 10 - 20% del *training set*, si è quindi considerato un intervallo di dati (in tabella 4.6) variabile fra il 5 e 34 % del *training set*.

Dalla tabella si evince come il minimo RMSE si ottenga con un *validation set* pari al 10% dei dati completi di training.

Per visualizzare la variazione della funzione di costo **MAE** durante l'allenamento della rete (ossia all'aumentare delle epoche), si è realizzato nel codice il salvataggio di tale funzione (si ricorda che la funzione RMSE non è presente nelle funzioni di costo di Keras).

Visualizzazione della funzione di costo

```
# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=4000,
                      verbose=0)
# evaluate the model
_, train_acc = model.evaluate(trainX, trainy, verbose=0)
_, test_acc = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# plot training history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

4. La rete neurale realizzata

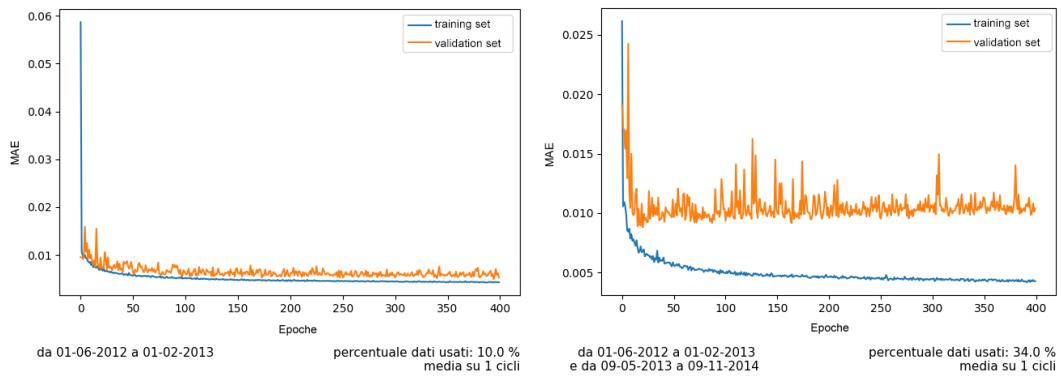


Figura 4.7: Validation set confronto MAE fra 10% e 34%

Dalle due immagini riportate in figura 4.7 si può notare come dopo 100 epoche il valore **MAE** per il validation set formato dal 10% dei dati di training non abbia più apprezzabili riduzioni, mentre se si utilizza il 34% dei dati il **MAE** è troppo variabile per definire un minimo, anche se dopo circa 20-30 epoche non sembra diminuire più. Per evitare che l'inizializzazione randomica dei pesi di ciascun neurone possa incidere eccessivamente sui risultati, l'allenamento della rete è stato ripetuto per dieci volte facendo poi la media dei risultati ottenuti, come discusso successivamente nel paragrafo 4.8.2. In figura 4.8 sono riportati i valori fino a 100 epoche.

Quello che si nota è come il valore di MAE per un paio di epoche sia alto per entrambe le simulazioni. Poi superate le 50/60 epoche tende a risalire per il *validation set* al 34%, mentre rimane pressoché invariato, per il *validation set* al 10%. Nell'intervallo compreso tra le 5 e le 50/60 epoche si ha un valore di cicli di allenamento tale da evitare, perciò, l'*overfitting* o l'*underfitting*.

La scelta del miglior numero di epoche può essere gestita automaticamente durante la fase di allenamento grazie alle funzionalità di Keras. La funzione specifica è denominata *early stopping* e permette di demandare alla rete il controllo della funzione costo, per definire quando interrompere l'allenamento.

Earlystopping

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=100,
                     verbose=0, callbacks=[es])
# evaluate the model
```

Funzione richiamata nel codice della rete neurale

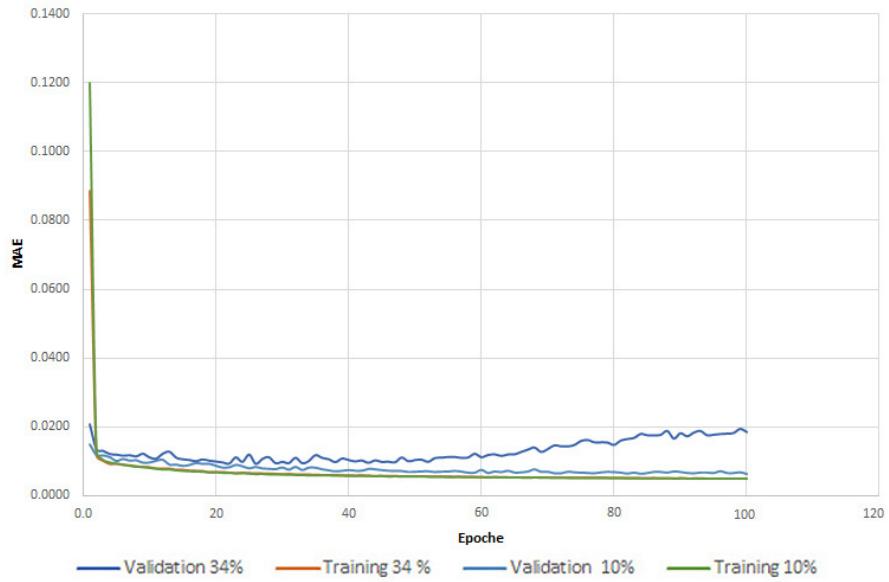


Figura 4.8: Validation set confronto MAE fra 10% e 34%, media su 10 cicli

La funzione *early stopping* necessita come parametri²:

- la funzione da monitorare (*monitor*);
- se determinare il minimo o il massimo (*mode*) a seconda della funzione di costo utilizzata;
- il grado di confidenza, (*patience*), con cui interrompere l'apprendimento se, ad esempio, dopo n epochi il valore della funzione costo rimane costante o peggio inizia ad aumentare.

L'implementazione del codice ha messo in luce quello che intuitivamente si notava: l'interruzione delle epoche avviene fra le 50 e 60 iterazioni, con punti massimi di 70/80. Si è quindi provveduto a impostare un valore massimo di epochi pari a 100 con una tolleranza (*patience*) pari a 10.

4.8.2 Implementazione di cicli

Le reti neurali sono per natura stocastiche. Infatti, a causa dell'inizializzazione randomica dei pesi dei neuroni, non restituiscono lo stesso risultato se l'allenamento viene ripetuto più volte utilizzando la stessa struttura della rete e gli stessi dati di training.

Per sopperire a questo problema, che può determinare l'interruzione "precoce" dell'allenamento in presenza di minimi locali nella funzione costo, si è deciso di ripetere l'allenamento della rete un numero di volte sufficiente a restituire

²Per approfondire: <https://machinelearningmastery.com>

4. La rete neurale realizzata

un valore medio non più affetto (o affetto non significativamente) da questo problema.

In figura 4.9 si è rappresentata, parte dell'onda del 10/21/2019 con tre previsioni a sei ore denominate 1° , 2° , 3° ciclo e media degli stessi.

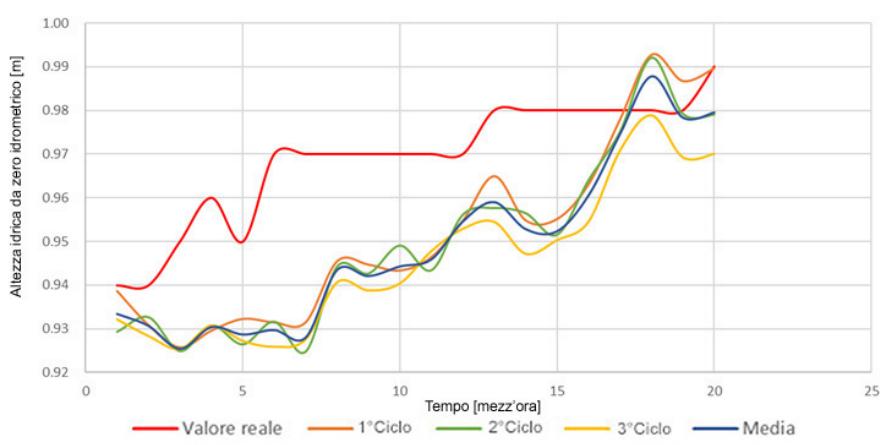


Figura 4.9: Particolare dell'onda del 10/21/2019 (prime 10 ore) con tre cicli di previsione a sei ore e relativa media

Fino a questo momento si era fatto riferimento in generale agli anni 2018-2019 che contengono oltre a onde di piena significative, anche alcune onde moderate, con altezze idrometriche che non superano il paio di metri, e anche periodi di magra fra un'onda e l'altra, di cui non interessa la ricostruzione accurata. Ricordiamo infatti che l'obiettivo primario è prevedere le piene che possono provocare esondazioni in corrispondenza dell'attraversamento dell'abitato di Colorno.

Per migliorare quindi la valutazione dell'efficacia della rete nel prevedere i livelli di piena, è stato utilizzato un nuovo criterio di valutazione: anziché calcolare il valore dell'RMSE sull'intero *test set*, sono state identificate solo cinque onde di piena sulle quali valutare l'RMSE.

In tabella 4.7 si possono osservare i risultati dopo 10-20-30 iterazioni della rete.

4.8. Ottimizzazione della rete

Tabella 4.7: Risultati di differenti cicli con differenti neuroni

Neuroni	Cicli	Media sulle onde di test				
		RMSE [m]	Errore sul colmo [m]	Std [m]	RMSE medio [m]	RMSE std [m]
5	10	0.096	0.388	0.053	0.111	0.014
5	20	0.101	0.403	0.048	0.115	0.017
5	30	0.102	0.41	0.048	0.116	0.019
10	10	0.098	0.312	0.055	0.113	0.017
10	20	0.102	0.341	0.054	0.119	0.018
10	30	0.103	0.355	0.057	0.12	0.025
20	10	0.104	0.243	0.049	0.115	0.018
20	20	0.1	0.261	0.046	0.111	0.017
20	30	0.097	0.285	0.045	0.108	0.016

Nella tabella 4.7, ricordando che il valore espresso è la media effettutata fra le cinque onde in esame, sono riportate:

- nella terza colonna il valore del RMSE fra la media dei cicli e il valore corretto;
- nella quarta colonna l'errore sul colmo fra la media dei cicli e il valore corretto;
- nella quinta colonna la deviazione standard fra i valori predetti e la media dei cicli stessi;
- nella sesta colonna il valore medio dell'RMSE fra i cicli;
- nella settima colonna la deviazione standard fra la la media dell'RMSE fra i cicli e il valore RMSE di ciascun ciclo.

I risultati sono pressoché identici, all'aumentare del numero di cicli, con differenze che non superano i pochi millimetri, ininfluenti dato l'obiettivo dello studio. Il risultato di tale implementazione è riportato in figura 4.10.

Per cogliere l'85-esimo percentile delle variazioni delle previsioni, supponendo che la distribuzione sia normale, si può considerare il doppio della **standard deviation** così da rappresentare al meglio come si distribuiscono, rispetto al dato reale, le singole previsioni. Questo risulta quindi essere un ulteriore parametro per definire la bontà dei risultati ottenuti.

4. La rete neurale realizzata

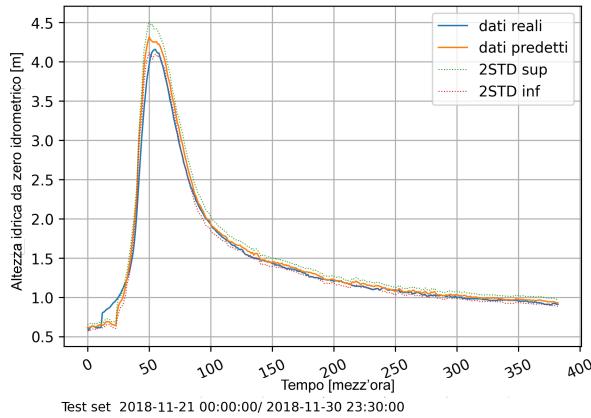


Figura 4.10: Previsione a sei ore dell'onda del 21/11/2018 a Colorno mediata su dieci cicli

4.8.3 Verifica finestra di osservazione

L'ultima analisi riguarda la verifica della finestra di osservazione utilizzata per identificare i dati di input da fornire alla rete.

La dimensione della finestra come spiegato nel paragrafo 4.3 dipende dal fenomeno fisico in esame, per il torrente Parma le piene si sviluppano nell'arco delle 18 ore, si veda a tal proposito la figura 4.1 o quella di seguito riportata 4.11. Finora si è testata con successo la previsione a sei ore con una finestra totale di diciotto ore, avendo quindi dodici ore precedenti. Per la previsione a 9 ore, tale finestra garantisce di considerare in input solo i dati delle 9 ore precedenti. Si è quindi deciso di provare ad allargare la finestra a ventuno ore, prendendo così sempre dodici ore precedenti.

I risultati sono riportati nelle tabelle 4.8.

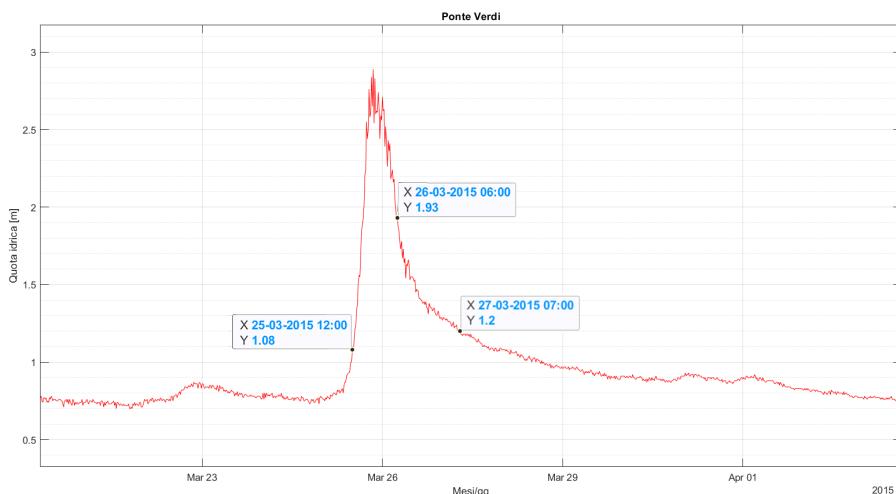


Figura 4.11: Finestra di previsione a 18 ore, onda del 25/03/2015 alla stazione idrometrica Ponte Verdi

4.8. Ottimizzazione della rete

Tabella 4.8: Finestra dati input per previsione a nove ore

Finestra di 18 ore previsione a 9 ore					
Neuroni	RMSE [m]	errore picco [m]	media std [m]	RMSE medio[m]	RMSE std [m]
2	0.22	0.25	0.28	0.31	0.61
4	0.21	0.60	0.07	0.22	0.05
8	0.21	0.62	0.06	0.21	0.04
10	0.20	0.60	0.05	0.21	0.03
20	0.21	0.61	0.05	0.22	0.04
30	0.20	0.60	0.05	0.21	0.03

Finestra di 21 ore previsione a 9 ore					
Neuroni	RMSE [m]	errore picco [m]	media std [m]	RMSE medio[m]	RMSE std [m]
2	0.21	0.56	0.05	0.22	0.02
4	0.20	0.26	0.29	0.31	0.61
8	0.20	0.61	0.07	0.22	0.06
10	0.19	0.55	0.05	0.20	0.03
20	0.20	0.58	0.06	0.21	0.03
30	0.20	0.62	0.06	0.21	0.04

L'incremento della finestra sembra non portare alcun miglioramento, la fisica del fenomeno reale non permette di estendere ulteriormente la finestra. Si è quindi deciso di mantenere fissa la finestra a 18 ore sia per previsioni a sei ore che per quella a nove ore.

Nel capitolo successivo saranno descritti nel dettaglio i risultati ottenuti con il miglior set.

5

Analisi risultati ottenuti

Attraverso la serie di prove riportate al capitolo precedente si è giunti a definire la rete ottimale. L'uso dell'*early stopping* ha permesso di ridurre il tempo di calcolo, che si attestava sui 10 minuti per esecuzione, a 5 minuti per 10 simulazioni. Esso ha anche evitato eventuali *overfitting* o *underfitting* ottimizzando la rete. La media su dieci cicli ha permesso di eliminare ogni aleatorietà, presentando anche quale grado di confidenza è possibile dare alla rete una volta calcolata la deviazione standard. In tabella 5.1 si riportano le tempistiche di allenamento e previsione delle reti in varie configurazioni, che si attestano inferiori alla decina di minuti.

Tabella 5.1: *Tempi osservati nelle varie configurazioni*

	Minuti
Senza <i>early stopping</i>	8.16
Con <i>early stopping</i>	3.9
Con <i>early stopping</i> e 10 cicli	4.42

5.1 Rete ottimale

5.1.1 Sei ore di previsione

Come si può evincere dalla tabella 5.2, in cui si riportano i dati relativi alla previsione a sei ore variando il numero di neuroni, già utilizzando otto neuroni la rete presenta buoni risultati; questa affermazione è corroborata dalla figura 5.1, che riporta le ricostruzioni dell'onda del 20/11/2018 al variare del numero di

5. Analisi risultati ottenuti

neuroni.

Si ricorda che la tabella 5.2 presenta i valori medi rispetto alle cinque onde presenti in esame nel test set.

Tabella 5.2: Valori medi di dieci cicli e previsione a sei ore

Neuroni	RMSE [m]	Errore sul colmo [m]	Media std [m]	RMSE medio[m]	RMSE std [m]
1	0.35	0.79	0.40	0.43	0.96
2	0.25	0.45	0.35	0.33	0.84
4	0.10	0.25	0.06	0.12	0.04
8	0.11	0.27	0.05	0.12	0.03
10	0.11	0.26	0.05	0.12	0.04
20	0.11	0.26	0.05	0.12	0.04
30	0.11	0.28	0.05	0.12	0.05
40	0.11	0.26	0.05	0.12	0.04
50	0.10	0.23	0.04	0.11	0.03
60	0.11	0.22	0.04	0.11	0.03
70	0.11	0.22	0.06	0.12	0.04
80	0.10	0.24	0.04	0.11	0.03
90	0.10	0.24	0.04	0.11	0.04
100	0.10	0.23	0.06	0.12	0.04

A titolo di esempio, in figura 5.2 si riporta l'onda del 04/2018 ricostruita con due, quattro, otto e dieci neuroni. Da essa si può notare come la previsione con due neuroni presenti deviazioni standard molto elevate e in tutte le onde ricostruite con questa configurazione si osserva una sottostima dei livelli idrometrici. Come si può notare anche dalla figura 5.1, la previsione migliora nel caso di quattro neuroni e presenta pochissimi cambiamenti dagli otto neuroni in poi.

Date queste premesse, non risulta utile utilizzare più di otto neuroni per tale contesto. L'ulteriore aumento del numero di neuroni comporta solo un indesiderato aumento dell'onere computazionale.

Definito il numero di neuroni, si è proceduto a determinare la batch: in figura 5.3 si riporta l'onda del 20/11/2018 in differenti configurazioni di batch.

Non vi è da stupirsi nel constatare come, a differenti batch, i risultati ottenuti risultino praticamente identici: una volta resa robusta la rete raggiungere il minimo velocemente (batch alte) o lentamente (batch basse) porta praticamente al medesimo risultato.

5.1. Rete ottimale

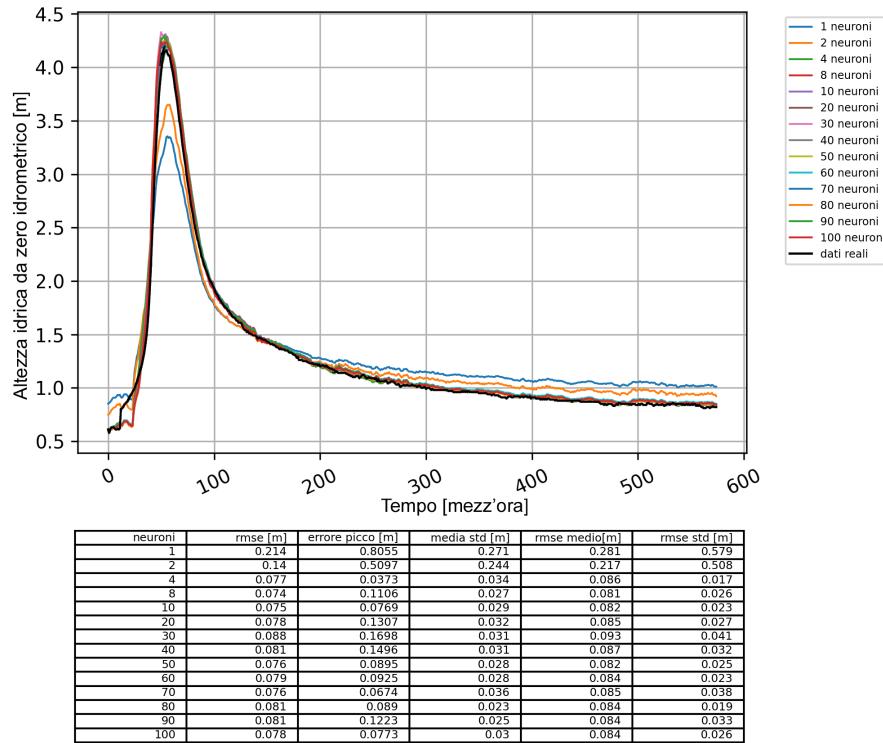


Figura 5.1: Rappresentazione dell'onda del 20/11/2018 con differenti neuroni a sei ore di previsione

In figura 5.4 è riportato il tempo di calcolo in minuti, in funzione del numero di batch usate nella simulazione. Le simulazioni sono state svolte in serie, partendo da 100 batch: tale valore quindi è affetto dall'inizializzazione¹ della procedura ed escludendolo risulta evidente che dopo 50 batch la rete è perciò in grado di allenarsi ogni volta, in tempi inferiori ai 5 minuti. I dati della figura sono riportati in appendice C.4. La rete ottimale per previsione a sei ore ha quindi i seguenti parametri:

Epoche (con earlystopping)	100
Hidden layers	1
Neuroni utilizzati	8
Batch	60
Funzione di attivazione	<i>ReLU</i>

¹Durante il primo ciclo lo script deve recuperare i dati salvati in formato .txt ed elaborare la matrice input e il vettore output, tali azioni si svolgono solo al primo avvio.

5. Analisi risultati ottenuti

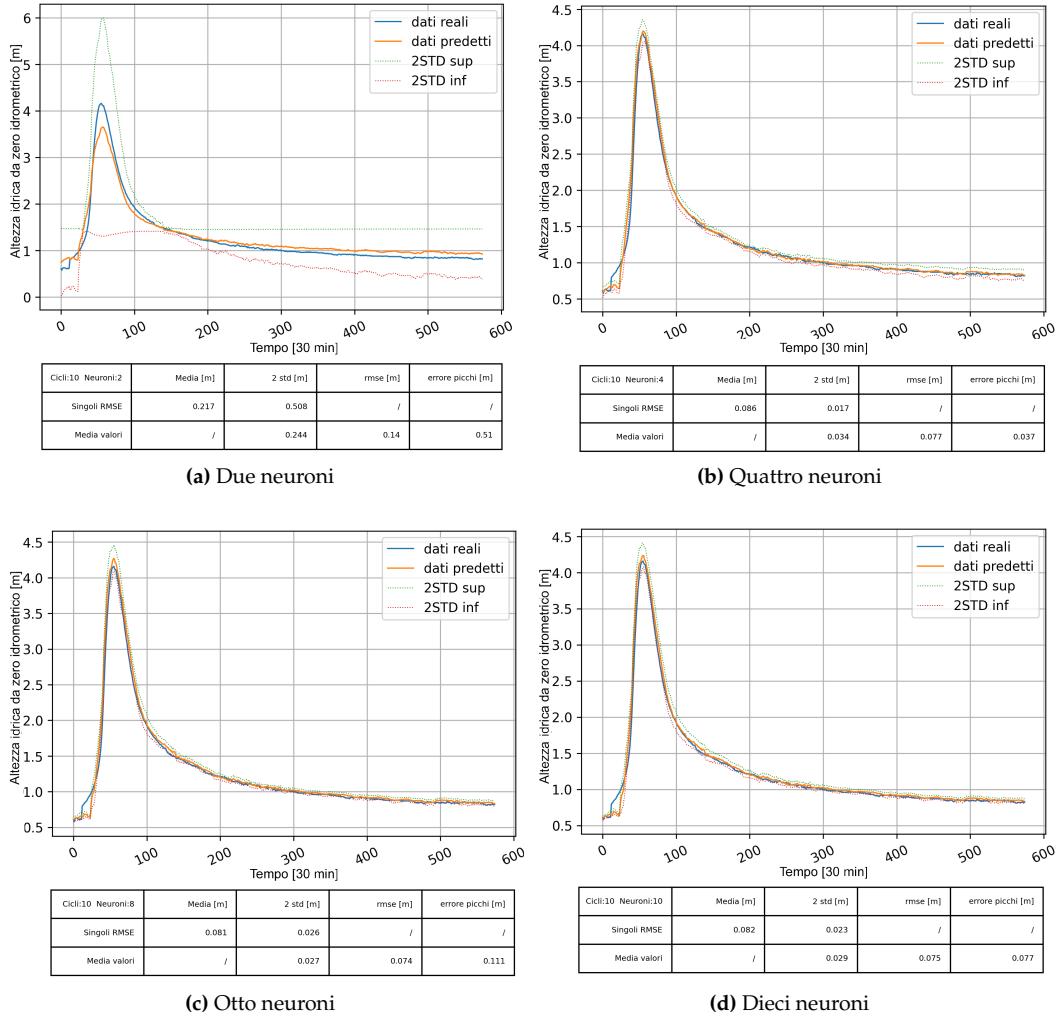


Figura 5.2: Onda del 20/11/2018 in diverse configurazioni con previsione a 6 ore

5.1.2 Nove ore di previsione

I risultati con differenti neuroni per la previsione sono già stati mostrati in tabella 4.8 nel paragrafo 4.8.3. Anche in questo scenario le differenze oltre gli otto neuroni non sono rilevanti: dato il contesto si è quindi scelto di utilizzare i medesimi parametri definiti per la previsione a sei ore.

In figura 5.5 si riportano comunque per comparazione i risultati per l'onda del 20/11/2018.

In appendice C.2 si riporta lo script della rete ottimale a sei e a nove ore.

5.1. Rete ottimale

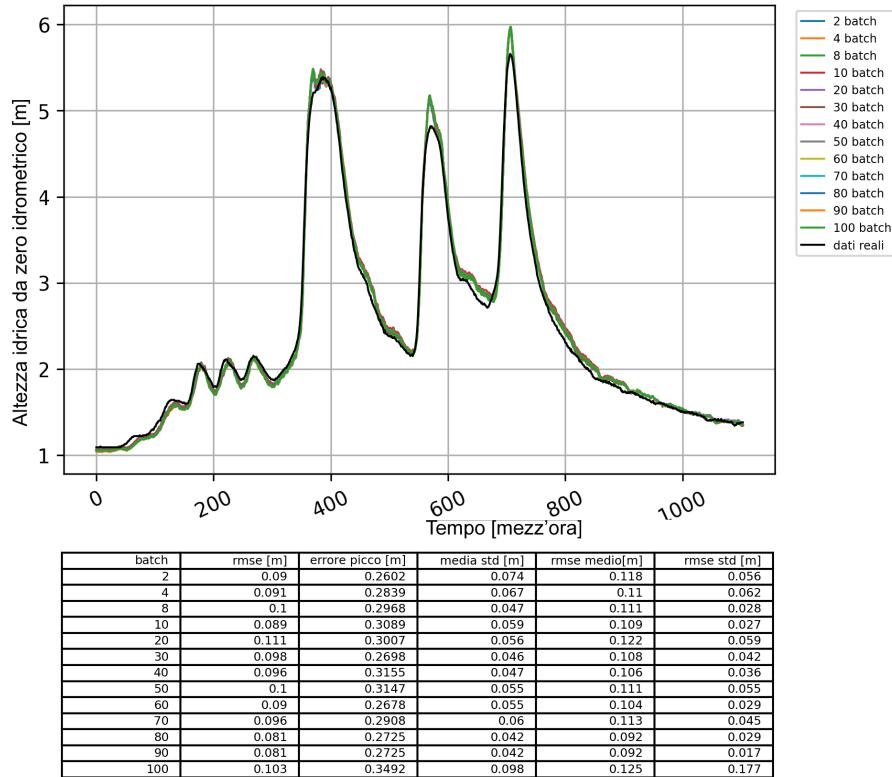


Figura 5.3: Rappresentazione dell'onda del 04/03/2018 con differenti batch a sei ore di previsione

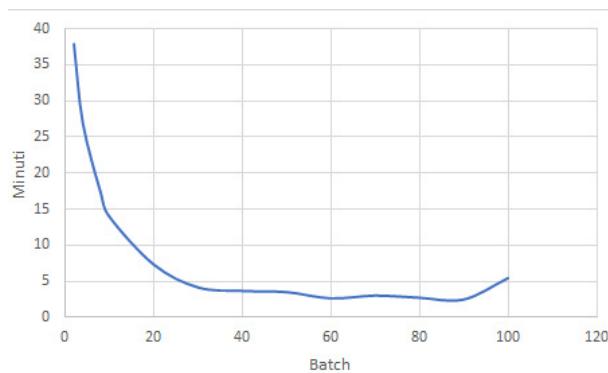


Figura 5.4: Tempi di elaborazione a differenti batch

5. Analisi risultati ottenuti

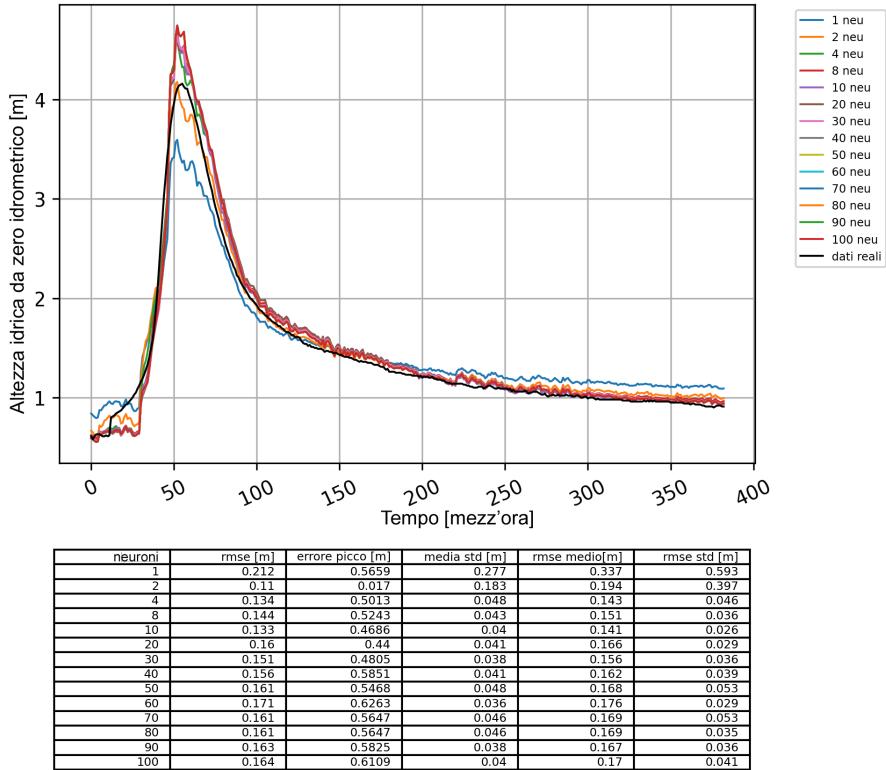


Figura 5.5: Rappresentazione dell'onda del 20/11/2018 con differenti neuroni a nove ore di previsione

5.1.3 Problematiche rilevate

Strutturata la rete si possono definitivamente osservare i risultati al netto delle migliorie e modifiche apportate. Si è ottenuto un errore al picco medio di 27 [cm] nel caso di previsione a sei ore e di 62 [cm] nella previsione a nove ore. Il valore è mediato sulle onde prese in esame. Per alcuni eventi si notano incoerenze sui dati registrati, in figura 5.6 si riportano due onde del 18 e 21 dicembre 2019 simulate e confrontate con i dati reali. Si nota come la previsione non sia praticamente coerente con il dato. In figura 5.7 è riportato, con lo stesso periodo delle onde mal previste, l'andamento dei livelli registrati nelle tre stazioni considerate (P.te Verdi, Colorno, e Casalmaggiore): i livelli idrometrici registrati a Ponte Verdi presentano alcuni gradini inspiegabili e sicuramente non fisici (prima del 22 ottobre e dopo il 3 novembre), probabilmente ascrivibili ad un mal funzionamento dello strumento². Ciò ha sicuramente inciso sulla corretta previsione delle onde riportate in figura 5.6.

Tale problematica qui palese può essersi verificata più volte portando ad un errata stima delle altezze idriche generali.

²I dati scaricati dal sito AIPO sono stati confrontati con l'archivio Dext3r che presenta i medesimi scalini.

5.1. Rete ottimale

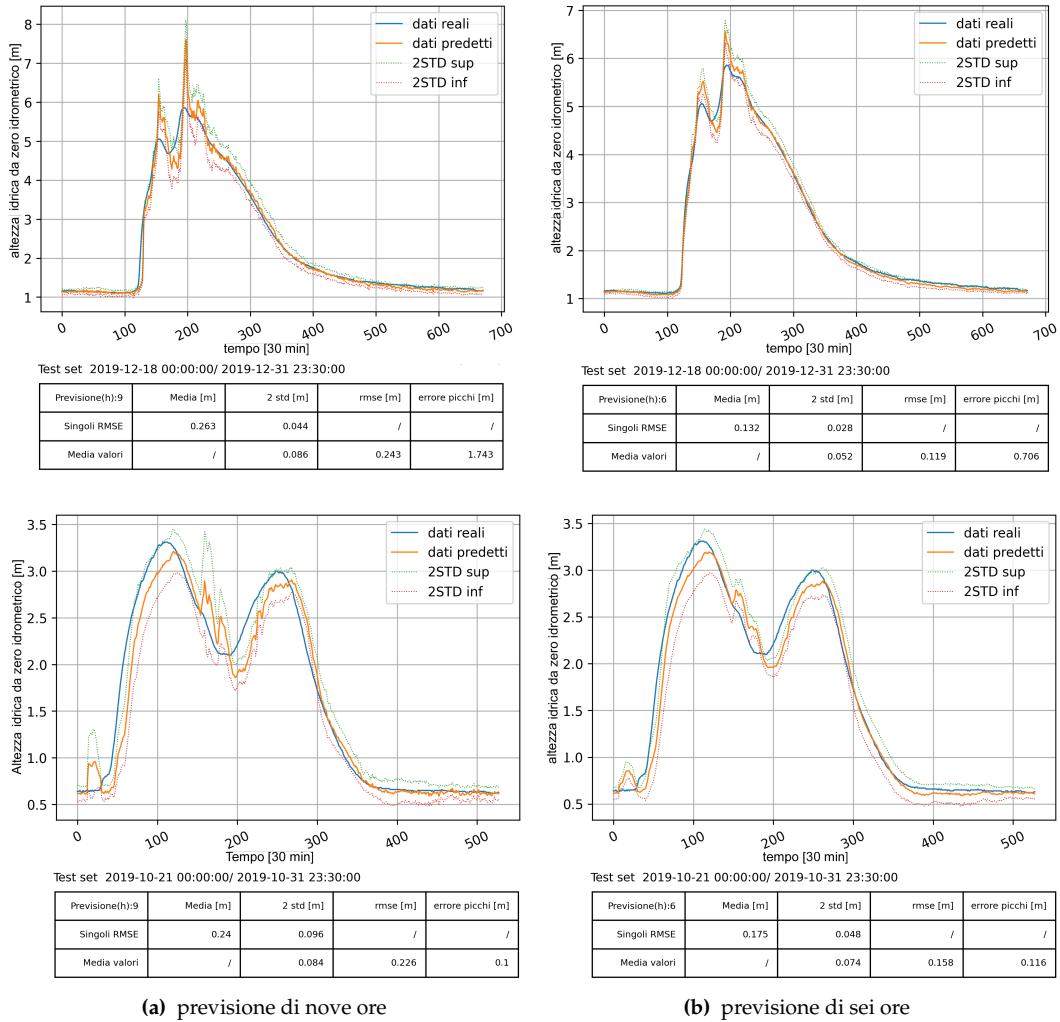


Figura 5.6: Errori di previsione nelle onde del 18 e 21 dicembre 2019

5. Analisi risultati ottenuti

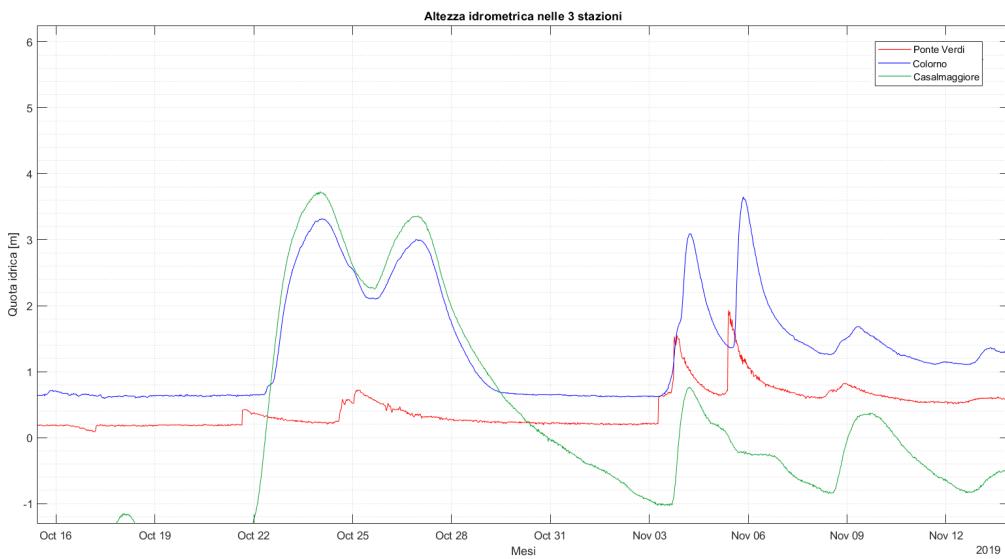


Figura 5.7: Periodo delle onde in esame 18-21 dicembre 2019

Una possibile soluzione sarebbe quella di effettuare uno smoothing³ dei dati in input per ridurre eventuali errori di questo genere. Essendo un errore comune occasionale, nel periodo di training la rete dovrebbe essere stata in grado di compensarlo con le altre osservazioni corrette.

5.1.4 Previsione delle onde significative del 2014 e del 2017

Per testare la bontà della rete si è deciso di eliminare dal *training set* le onde del 14 ottobre 2014 e quella del 12 dicembre 2017 che presentano i massimi valori di altezza idrometrica a Parma e a Colorno. Esse, con la robusta rete creata, non presentano alcun errore significativo nell'essere previste come onde del *test set* nello scenario a sei ore. Presentano invece numerose oscillazioni nel caso di previsione a nove ore. A tal proposito si riportano, in figura 5.8, le due onde per i due orizzonti di previsione nello scenario peggiore, cioè in assenza di entrambe le onde nel *training set*.

³In statistica si definisce smoothing l'applicazione di una funzione di filtro per eliminare eventuali errori locali.

5.1. Rete ottimale

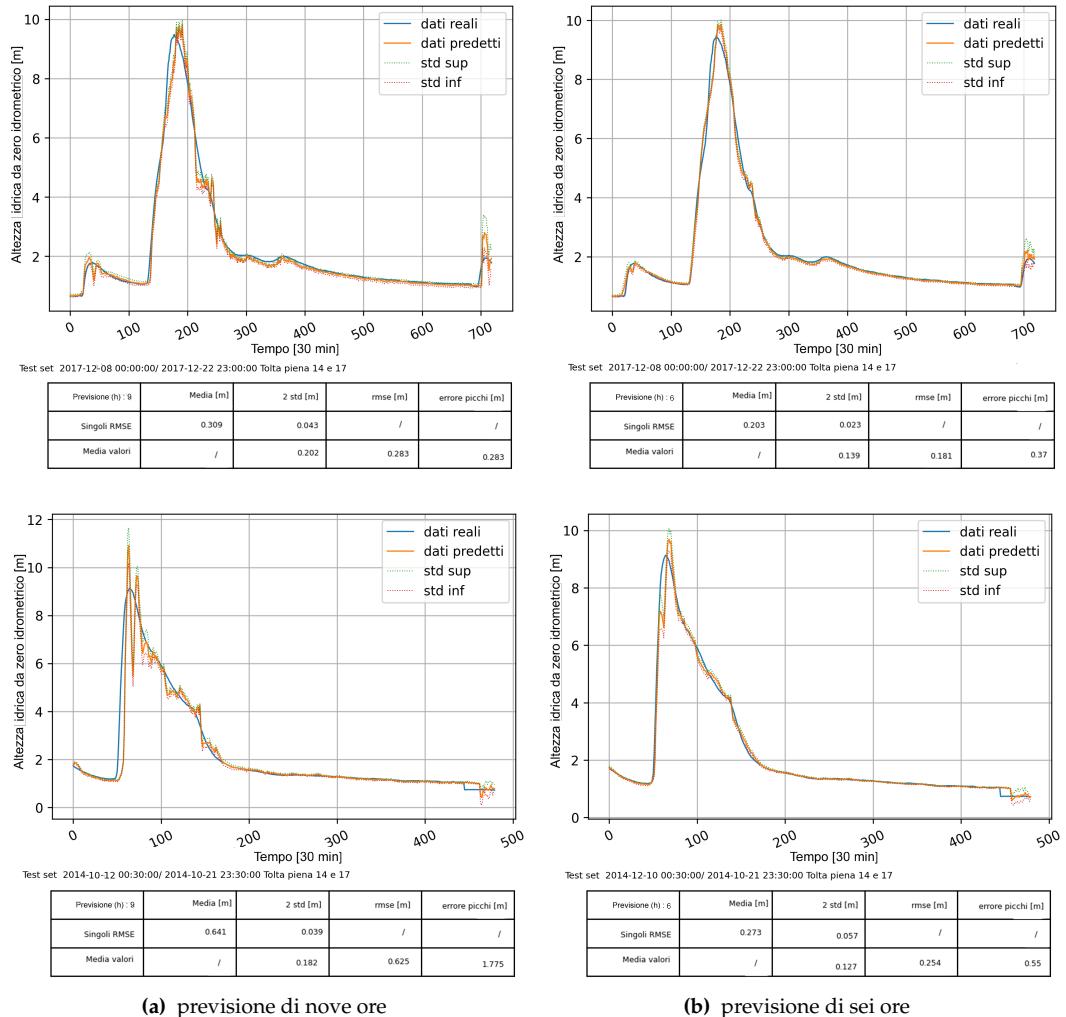


Figura 5.8: Previsione delle piene del dicembre 2017, in alto, e del ottobre 2014, in basso

5. Analisi risultati ottenuti

5.2 Applicazione alla previsione di due eventi di piena del 2020

In questo paragrafo si riporterà un esempio di utilizzo dello script per previsioni future.

Vi sono due possibili procedimenti per l'aggiunta e la previsione di nuovi dati: la prima è quella di allenare nuovamente la rete, inserendo l'evento da prevedere nel *test set*. La seconda è quella di salvare la rete allenata e rilanciare il modello per la previsione dei nuovi dati, con un notevole risparmio di tempo di calcolo. Si è strutturato lo script in modo da consentire di salvare la rete neurale per utilizzi futuri, senza necessità di nuovi allenamenti, grazie alla funzione **ModelCheckpoint**. Si è quindi proceduto nel salvare il miglior modello realizzato nei due contesti di previsione a sei e a nove ore.

Il modello che si salverà è la miglior rete realizzata in quel ciclo, si è quindi provveduto a salvare tanti modelli quanti cicli sono stati eseguiti.

Salvataggio e caricamento della rete

```
# funzione per salvataggio modello
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max',
    verbose=1, save_best_only=True)
# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=4000,
    verbose=0, callbacks=[es, mc])
# Caricamento modello salvato
saved_model = load_model('best_model.h5')
# valutazione modello
_, train_acc = saved_model.evaluate(trainX, trainy, verbose=0)
_, test_acc = saved_model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

È richiesta l'installazione della funzione h5py <https://docs.h5py.org/en/latest/build.html>

Gli eventi considerati sono quelli del 29/02/2020 e del 3/12/2020. Le previsioni sono riportate in figura 5.9. Si nota una maggiore difficoltà nel prevedere tali onde che presentano una notevole sovrastima dei livelli massimi (dell'ordine di 50-60 cm). Una possibile causa sono gli interventi di pulizia dell'alveo recentemente effettuati, che hanno certamente portato ad un riduzione, a parità di livelli idrometrici registrati a Ponte Verdi, dei corrispondenti livelli idrometrici a Colorno. La stessa sovrastima si è infatti ottenuta mediante simulazione con un modello idraulico bidimensionale, tarato su eventi di piena degli anni precedenti, a parità di coefficienti di scabrezza.

5.2. Applicazione alla previsione di due eventi di piena del 2020

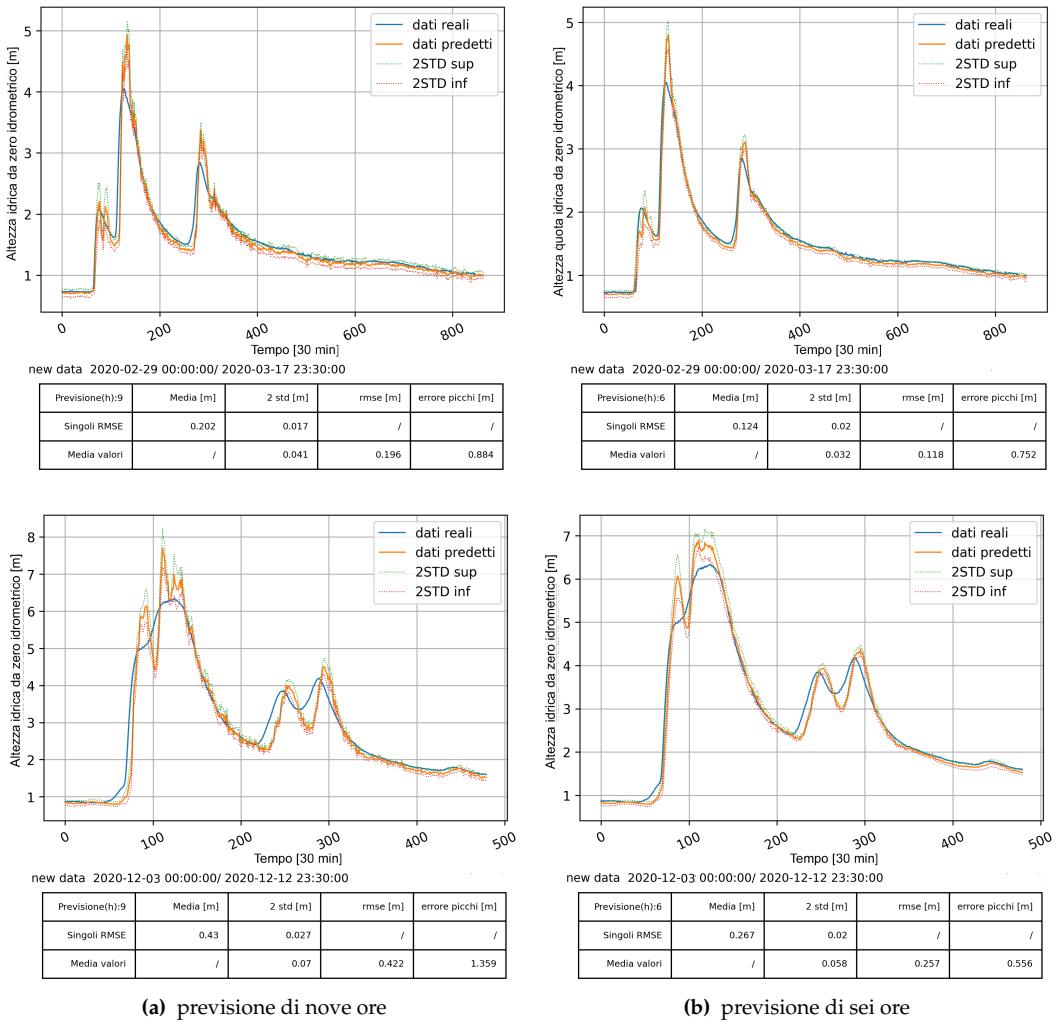


Figura 5.9: Previsione delle onde del 29/02/2020, in alto, e del 3/12/2020, in basso

In appendice C.3 si riporta lo script per utilizzare il *modelcheckpoint* per la previsione delle onde mostrate.

Come detto i tempi si sono notevolmente ridotti passando da 2/3 minuti per l'allenamento e previsione a soli 5.68 secondi per la sola previsione di dati.

6

Conclusioni

Il lavoro svolto era incentrato sul tentativo di prevedere i livelli di piena in corrispondenza della stazione idrometrica di Colorno. La rete neurale artificiale realizzata può essere un ottimo strumento per tempi di previsione relativamente brevi. Si ha una buona previsione per orizzonti fino a sei ore, mentre per orizzonti di nove ore le previsioni peggiorano sensibilmente.

Le cause possono essere molteplici e potranno essere affrontate in futuri lavori. Senza quindi togliere valore al lavoro svolto si può affermare che:

- la presenza di errori di misura isolati alla stazione di Ponte Verdi, porta ad una non corretta previsione a Colorno. Gli errori sono ancora abbastanza limitati nella previsione a sei ore, grazie presumibilmente alle correzioni introdotte considerando i dati già osservati a Colorno, mentre a nove ore cominciano a diventare consistenti. Un filtraggio potrebbe migliorare le previsioni, a patto che ciò non limiti le possibilità di applicazione immediata "chiavi in mano" nel contesto reale;
- i dati in input comprendono anche magre e periodi che possono non essere funzionali alla previsione di onde di piena importanti. Un miglioramento dei risultati si potrebbe forse ottenere attraverso una scelta manuale delle onde o un ulteriore codice che le selezioni. Lo script o la scelta manuale dovrebbero essere guidate da senso critico e da precise direzioni;
- l'uso di risorse computazionali più performanti permetterebbe di effettuare un numero maggior simulazioni e ottenere più celermente risultati, permettendone un più rapido confronto.

6. Conclusioni

Complessivamente comunque il linguaggio di programmazione Python e la libreria Keras hanno permesso di realizzare una buona previsione delle onde, cogliendo la complessità del mondo reale.

Si è riusciti ad esempio a prevedere la piena del 2017 con errori al picco pari a 37 [cm] con una previsione a sei ore, e di 28 [cm] per quella a nove, l'**RMSE** nella prima previsione corrisponde a 0.181 [m] mentre per le nove ore è leggermente superiore, pari a 0.281[m].

In generale la previsione a 3 ore è quasi coincidente con il dato reale mentre le onde a 12 ore (ma già a 9) risultano poco coerenti con i dati reali. Si riporta in tabella 6.1 l'RMSE medio delle onde in esame nei vari orizzonti di previsione.

Tabella 6.1: RMSE medio nei vari orizzonti di previsione

Orizzonte di previsione [ore]	RMSE [m]
3	0.07
6	0.11
9	0.21
12	0.33

Le reti neurali risultano quindi un buon mezzo per prevedere le onde di piena in molteplici contesti, una volta addestrate correttamente, permettendo l'attivazione anticipata di interventi di protezione civile quali l'indicazione di portarsi ai piani più elevati, l'uso di sacchettature o l'evacuazione della popolazione.

Dato il campo di applicazione e l'interesse crescente rispetto alle reti neurali, si spera che tale elaborato possa essere utile e di ispirazione per futuri lavori.

Bibliografia

- Bengio, Y., I. Goodfellow, and A. Courville (2017), *Deep learning*, vol. 1, MIT press Massachusetts, USA:.
- Campolo, M., A. Soldati, and P. Andreussi (2003), Artificial neural network approach to flood forecasting in the river arno, *Hydrological Sciences Journal*, 48(3), 381–398.
- del Fiume Po, A. d. B. (2003), Progetto di piano stralcio per l'assetto idrogeologico (pai), *Documento di sintesi*.
- del Fiume Po, A. d. B. (2018), Linee generali di assetto idraulico e idrogeologico nel bacino del parma.
- Floreano, D., and C. Mattiussi (2002), Manuale sulle reti neurali, *Tech. rep.*, Il mulino.
- Grippo, L., and M. Sciandrone (2003), Metodi di ottimizzazione per le reti neurali, *Rapporto Tecnico*, 8, 09–03.
- Hebb, D. O. (1949), *The organization of behavior: a neuropsychological theory*, J. Wiley; Chapman & Hall.
- Kabir, S., S. Patidar, X. Xia, Q. Liang, J. Neal, and G. Pender (2020), A deep convolutional neural network for rapid fluvial flood inundation modelling, *arXiv preprint arXiv:2006.11555*.
- Kia, M. B., S. Pirasteh, B. Pradhan, A. R. Mahmud, W. N. A. Sulaiman, and A. Moradi (2012), An artificial neural network model for flood simulation using gis: Johor river basin, malaysia, *Environmental Earth Sciences*, 67(1), 251–264.
- Kuhn, M., K. Johnson, et al. (1996), *Applied predictive modeling*, vol. 26, Springer.
- McCulloch, W. S., and W. Pitts (1943), A logical calculus of the ideas immanent in nervous activity, *The bulletin of mathematical biophysics*, 5(4), 115–133.
- Minsky, M., and S. Papert (1969), An introduction to computational geometry, *Cambridge tiass.*, HIT.
- on Application of Artificial Neural Networks in Hydrology, A. T. C. (2000), Artificial neural networks in hydrology. i: Preliminary concepts, *Journal of Hydrologic Engineering*, 5(2), 115–123.
- Rashid, T. (2016), *Make your own neural network*, CreateSpace Independent Publishing Platform.
- Rosenblatt, F. (1958), The perceptron: a probabilistic model for information storage and organization in the brain., *Psychological review*, 65(6), 386.
- Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986), Learning representations by back-propagating errors, *nature*, 323(6088).

A

Dati utilizzati

A. Dati utilizzati

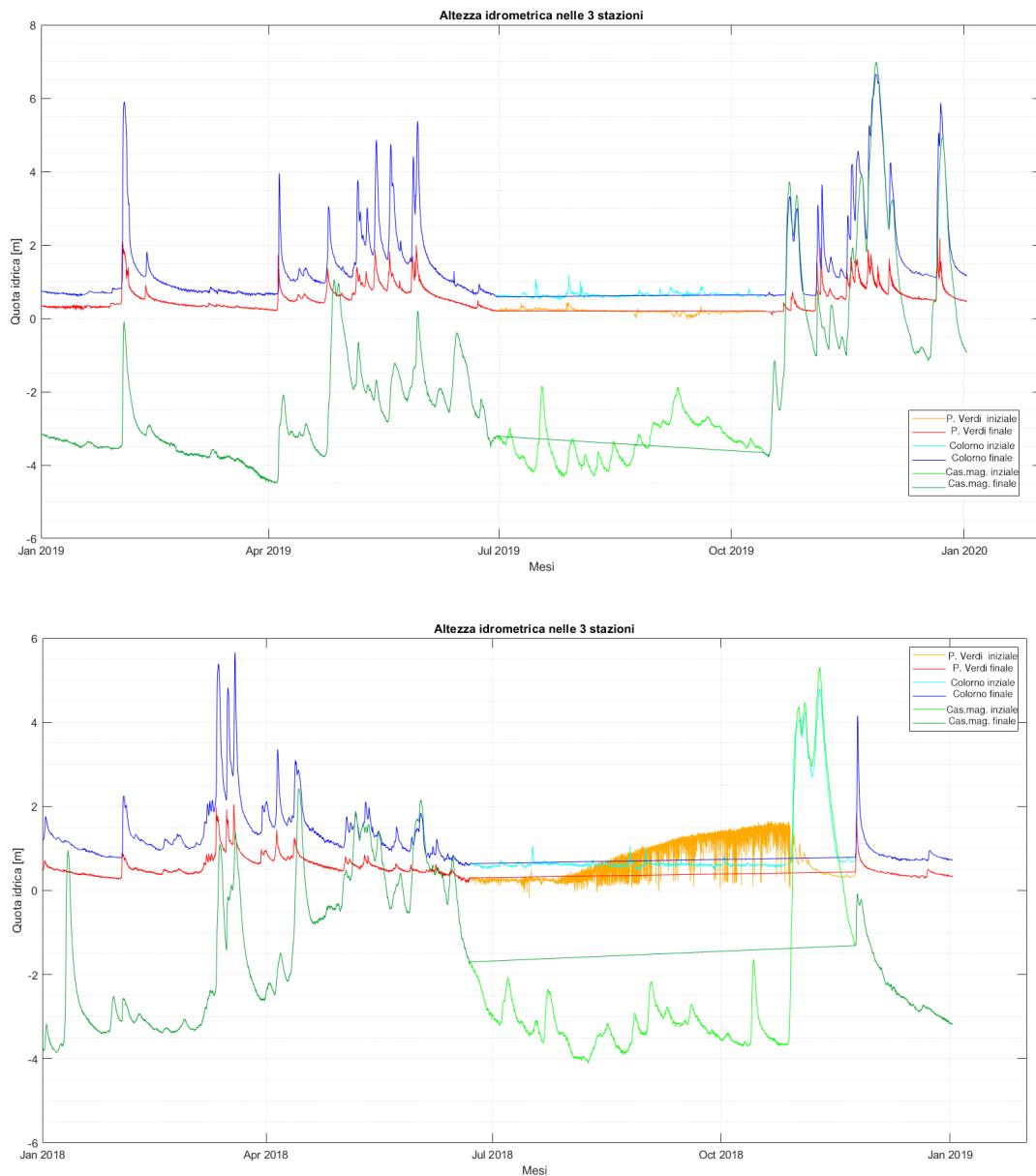


Figura A.1: Andamenti salvati nelle tre stazioni negli anni usati come testing

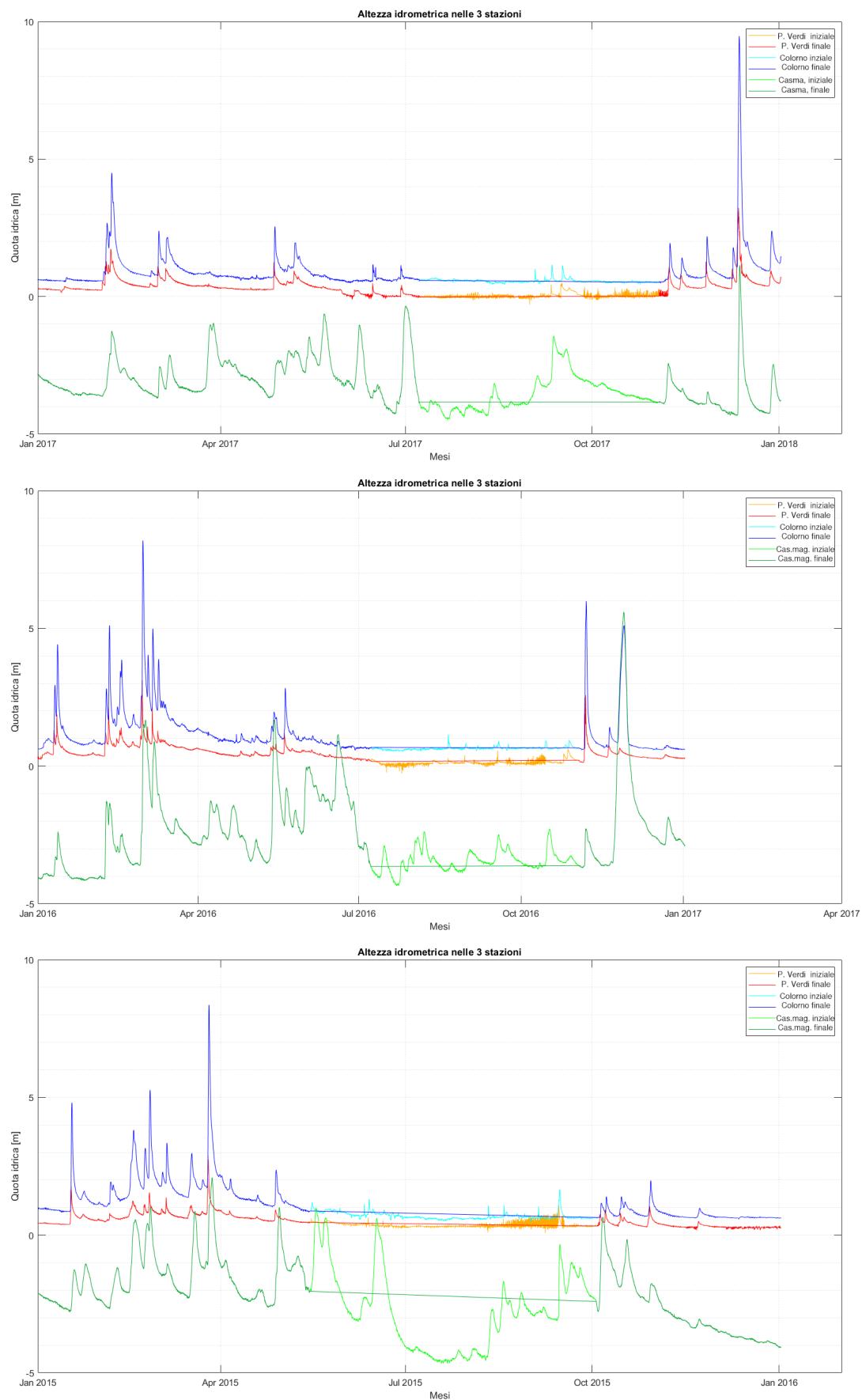


Figura A.2: Andamenti salvati nelle tre stazioni negli anni usati come training

A. Dati utilizzati

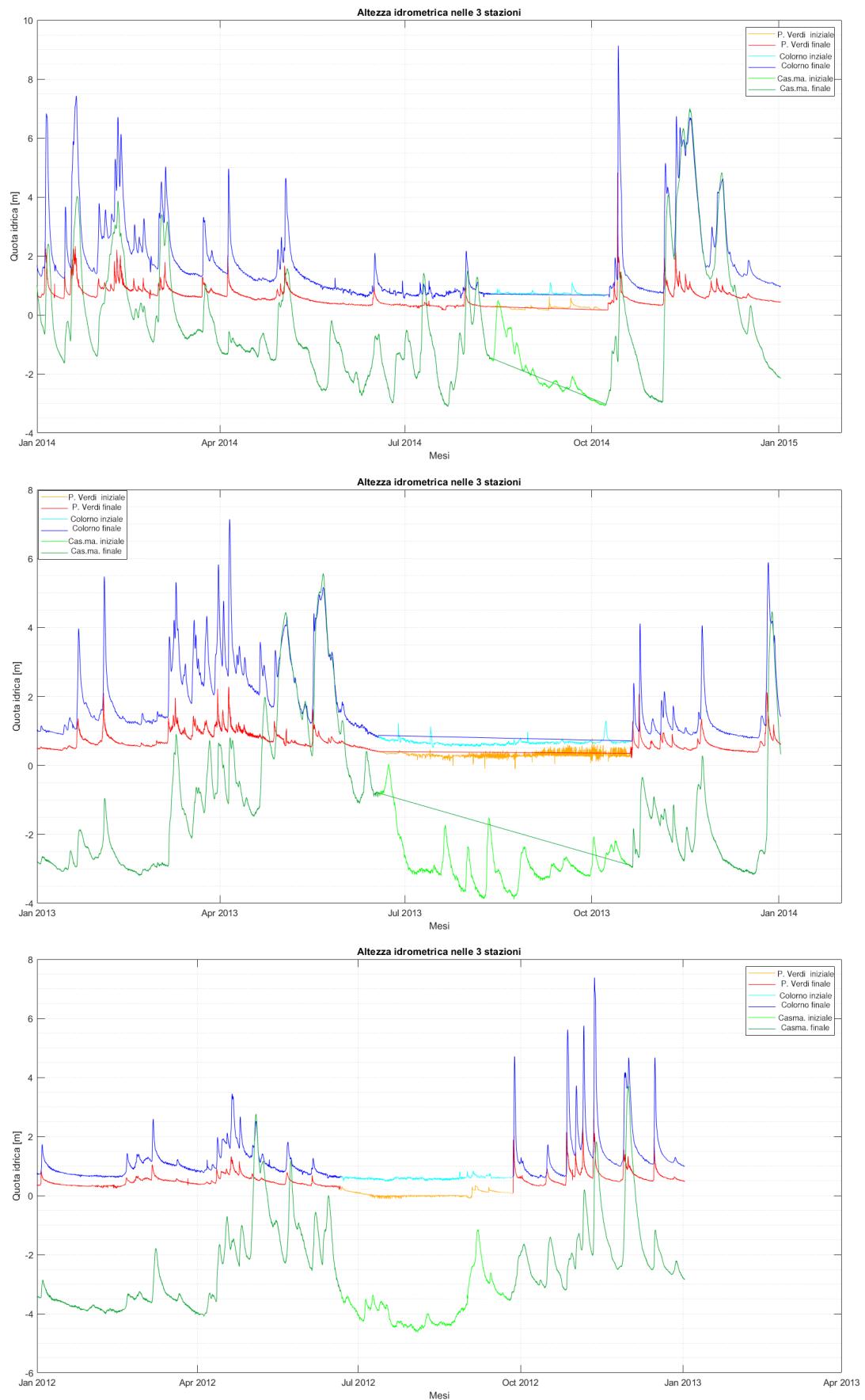


Figura A.3: Andamenti salvati nelle tre stazioni negli anni usati come training

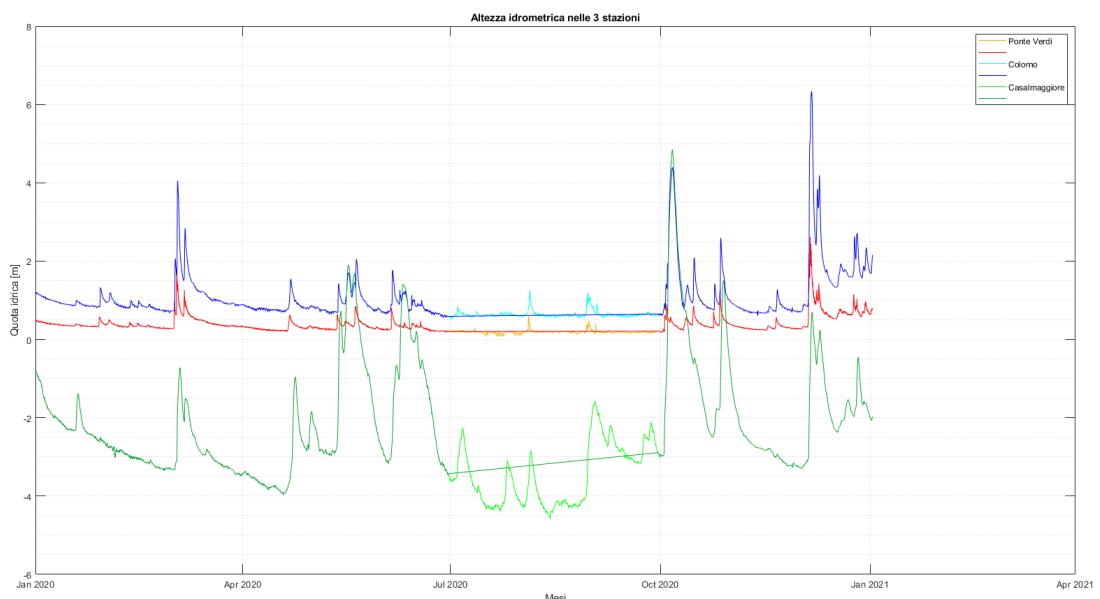


Figura A.4: Andamenti salvati nelle tre stazioni nell'anno 2020

B

Script Matlab: interpolazione dati

B.1 Discretizzazione ed elaborazione dati delle stazioni

```
clear all
clc
close all
f=1;
%% Importdata
for w=2012:2020 %Impostare l'anno da elaborare o la serie consecutiva di
    anni
    anni1=[2017,2015,2014,2013,2019,2018];
    anni2=[2016,2012,2020];
    if find(anni1==w) ~= ' '
        Dmax = '52705'; % Dimensione del vettore dati
    else
        Dmax = '52849';
    end
    anno=num2str(w);
    filename = "Colorno Aipo - Idrometro Torrente Parma.xlsx"; %nome del
        file
    hmin=0.10;
    %slope=0.02;
    DT=24*7;
    [hidroPV,hidroC,hidroCas,date,nameC,namePV,nameCas]=importdata_f(Dmax,
        anno,filename); %funzione che recupera i dati di Ponte verdi (PV) e
        Colorno (C) e Casalmaggiore (Cas) da tabella excel
    date=datetime(date, 'InputFormat', 'dd/MM/yyyy HH:mm', 'Format', 'dd-MM
        -yyyy HH:mm');
    %% Interpolazione valori nulli
    [hidroPV] = interpolate_f(hidroPV);
```

B. Script Matlab: interpolazione dati

```
[hidroC] = interpolate_f(hidroC);
[hidroCas] = interpolate_f(hidroCas);
%% Discretizzazione ogni 30 minuti (mn==30) NB! dipende dai valori
    presenti inizialmente!
j=1;
for i=1:length(date)
    [yyyy, mm, dd, h, mn,ss] = datevec(date(i));
    if mn==0 || mn==30 || mn==40 discretizzazione tempi
        dat_(j)= date(i);
        hidroC_(j)=hidroC(i);
        hidroPV_(j)=hidroPV(i);
        hidroCas_(j)=hidroCas(i);
        j=j+1;
    end
end
hidroPV_=hidroPV_;
hidroC_=hidroC_;
hidroCas_=hidroCas_;
dat_=dat_;
%% Taglio magre
inizio=["21-06-2012 00:00", "18-06-2013 00:00", "12-08-2014
00:00", "15-05-2015 00:00", "08-07-2016 00:00", "08-07-2017
00:00", "22-06-2018 00:00", "30-06-2019 00:00", "30-06-2020 00:00"];
fine=["25-09-2012 00:00", "20-10-2013 00:00", "08-10-2014
00:00", "03-10-2015 00:00", "03-11-2016 00:00", "03-11-2017
00:00", "24-11-2018 00:00", "14-10-2019 00:00", "30-09-2020 00:00"];
posin = find(dat_==(inizio(w-2011)));
posfin = find(dat_==(fine(w-2011)));
pi=0;
j=1;
i=1;
for i=posin:posfin
    pi(j)=i;
    j=j+1;
end
hidroPVa_=hidroPV_;
hidroCa_=hidroC_;
hidroCasa_=hidroCas_;
data_=dat_;
data_(pi,:)=[];
hidroCasa_(pi,:)=[];
hidroCa_(pi,:)=[];
hidroPVa_(pi,:)=[];
onde=table(data_,hidroPVa_,hidroCasa_, hidroCa_);
filename=strcat('30min_onde_corr_',num2str(w),'.txt');
writetable(onde,filename,'Delimiter','','','WriteVariableNames',0)
% %% Figure
f=f+1;
figure(f); %plot
```

B.1. Discretizzazione ed elaborazione dati delle stazioni

```
plot(dat_, hidroPV_, 'Color', [1.0 0.667 0.0]);
hold on
plot(data_, hidroPVA_, 'r');
grid on
grid minor
title('Ponte Verdi')
xlabel('Mesi/gg');
ylabel('Quota idrica [m]');
saveas(gcf,strcat('Ponte Verdi diff','_',num2str(w),'.fig')); %
    salvataggio grafico formato .fig
saveas(gcf,strcat('Ponte Verdi diff','_',num2str(w),'.png')); %
    salvataggio grafico formato .png
f=f+1;
figure(f); %plot
plot(dat_, hidroC_, 'c');
hold on
plot(data_, hidroCa_, 'b');
grid on
grid minor
title('Colorno')
xlabel('Mesi/gg');
ylabel('Quota idrica [m]');
saveas(gcf,strcat('Colorno diff','_',num2str(w),'.fig')); %salvataggio
    grafico formato .fig
saveas(gcf,strcat('Colorno diff','_',num2str(w),'.png')); %salvataggio
    grafico formato .png

f=f+1;
figure(f); %plot
plot(dat_, hidroCas_, 'g');
hold on
plot(data_, hidroCasa_, 'Color', [ 0.0275      0.6392      0.1882]);
grid on
grid minor
title('Casalmaggiore')
xlabel('Mesi/gg');
ylabel('Quota idrica [m]');
saveas(gcf,strcat('Casalmaggiore diff','_',num2str(w),'.fig')); %
    salvataggio grafico formato .fig
saveas(gcf,strcat('Casalmaggiore diff','_',num2str(w),'.png')); %
    salvataggio grafico formato .png

f=f+1;
figure(f); %plot
plot(dat_, hidroPV_, 'Color', [1.0 0.667 0.0], 'DisplayName', 'Ponte
    Verdi');
hold on
plot(data_, hidroPVA_, 'r');
plot(dat_, hidroC_, 'c', 'DisplayName', 'Colorno');
```

B. Script Matlab: interpolazione dati

```
plot(data_, hidroCa_, 'b');
plot(dat_, hidroCas_, 'g','DisplayName',' Casalmaggiore');
plot(data_, hidroCasa_, 'Color', [ 0.0275      0.6392      0.1882]);
grid off
grid minor
legend('Ponte Verdi','','Colorno','','Casalmaggiore','')
title('Altezza idrometrica nelle 3 stazioni')
xlabel('Mesì');
ylabel('Quota idrica [m]');
saveas(gcf,strcat('Tre stazioni a confronto diff','_',num2str(w),'.fig'
));
%salvataggio grafico formato .fig
saveas(gcf,strcat('Tre stazioni a confronto diff','_',num2str(w),'.png'
));
%salvataggio grafico formato .png
clearvars -except w f
end
```

B.2 Funzione richiamata in B.1 per l'interpolazione

```
function [hidro1] = interpolate(hidro)
row = find(isnan(hidro));
for i=1:length(hidro)
    if ismember(i ,row)
        T=i+1;
        while ismember(T ,row)
            T=T+1;
        end
        t=i -1;
        while ismember( t ,row)
            t=t -1;
        end
        y1=hidro(t);
        y2=hidro(T);
        hidro1(i) = y1+((y2-y1)*(i -t)/(T-t));
    else
        hidro1(i)=hidro(i);
    end
    i=i +1;
end
end
```

C

Script Python: reti neurali

C.1 Percettrone

```
# Made by: Gabriele Simonetta
## Simple Perceptron Example
# basato sul codice del video di Daniel Shiffman, maggiori informazioni
# al link: https://www.youtube.com/watch?v=ntKn5TPHHAk&t=1976s
import time # per vedere quanto impiega la rete neurale per giungere a soluzione
start = time.time()
import matplotlib.pyplot as plt # per creare i grafici
import numpy as np
# Create a list of evenly-spaced numbers over the range
x = np.linspace(0, 10, 1000) # larghezza finestra da 0 a 100 con 1000 punti totali
lr = 0.1 # learning rate
pt = 1000 # punti randomici generati
## Funzioni accessorie
def funzione(x): # scegli la funzione che vuoi usare e commenta le altre
    #y = x**2+5*x+5 #il quadrato si fa con il doppio per
    # y = x ** 3 + 7 * x ** 2 + 5 * x - 4
    #y=5*x+2
    #y = np.sin(x)
    #y = np.cos(x)
    y = (4*np.log(x+0.01)**2+x**2)*np.cos(x**3/(x**2+4))
    return y
def activator(y): # activator function
    if y > 0:
        target = 1
    else:
        target = -1
    return target
def rand(minimum, maximum, pt): # definisce vettore di lunghezza pt compreso nel
```

C. Script Python: reti neurali

```
range di minimum e maximum
r = np.array(minimum + (maximum - minimum) * np.random.rand(pt, 1))
return r

y = funzione(x)
a = rand(min(x), max(x), pt) # x variabili randomiche
b = rand(min(y), max(y), pt) # y variabili randomiche
plt.plot(x, y) # Plot the f(x) of each x point
plt.plot(a, b, marker=".", linestyle="") # Plot point
plt.gca().legend((f'f(x)', 'p.random'))
plt.show() # Display the plot

# funzione per vedere nelle immagini se la rete non commette errori colorando tutti i punti di verde
def pltcolor(lst):
    cols = []
    for u in lst:
        if u == 0:
            cols.append('green') # previsione corretta
        else:
            cols.append('red') # previsione sbagliata
    return cols

## pongo a valori noti le nuove variabili
yaspx = np.zeros(len(b))
target = np.zeros(len(b))
active = np.zeros(len(b))
guess = np.zeros(len(b))
error = np.zeros(len(b))
accuracy = np.zeros(len(b))
accuratezza = 0
errore = False
epoch = 1 # tiene in memoria quante volte il ciclo while si ripete
# pesi
weights_a = np.random.rand(len(b))
weights_b = np.random.rand(len(b))
##Neurone
while abs(errore) != True:
    for i in range(len(b)):
        yasp[i] = funzione(a[i]) # calcola la y relativa alla funzione
        if b[i] > yasp[i]: # se il valore delle y randomiche maggiore della y=f(x)
            allora assegna 1
            target[i] = 1
        else:
            target[i] = -1
        active[i] = a[i] * weights_a[i] + b[i] * weights_b[i] # funzione della cellula neurale
        guess[i] = activator(active[i]) # attivatore
        error[i] = target[i] - guess[i] # calcolo errore
        if error[i] == 0:
            accuracy[i] = 1
        else:
            accuracy[i] = 0
    # backward correzione errori
    weights_a[i] = weights_a[i] + error[i] * a[i] * lr
    weights_b[i] = weights_b[i] + error[i] * b[i] * lr
```

```

# Visualizzazione grafica della progressione della rete neurale
accuratezza = (sum(accuracy) / len(b)) * 100
cols = pltcolor(error)
plt.scatter(a, b, c=cols, marker=".") # Pass on the list created by the function
here
plt.plot(x, y, ) # Plot the sine of each x point
plt.annotate('accuracy: ' + str(round(accuratezza, 2)) + ' %', xy=(1, 0), xycoords='axes fraction', fontsize=11,
xytext=(0, -17), textcoords='offset points',
ha='right', va='top')
plt.annotate('epochs: ' + str(round(epochs, 2)), xy=(0, 0), xycoords='axes fraction',
, fontsize=11,
xytext=(0, -17), textcoords='offset points',
ha='right', va='top')
plt.show()
errore = all(value == 0 for value in error)
epochs = epochs + 1
end = time.time()
print('tempo impiegato: ' + str(round(end - start, 2)) + ' [sec] \nepochs: ' + str(epochs
)) # stampa a schermo il tempo impiegato e le epoche

```

C.2 Rete neurale conclusiva

```

## 
# final neural network to predict 2018 and 2019 from Ponte Verdi, Colorno and
Casalmaggiore station
import matplotlib
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
import winsound
from sklearn.model_selection import train_test_split
import numpy as np
from numpy import var
import matplotlib.pyplot as plt # per creare i grafici
import matplotlib.dates as md
import time # per vedere quanto impiega la rete neurale per giungere a soluzione
import statistics as st
import pandas as pd

def rmse(predictions, targets):
    differences = predictions - targets # Differenza.

    differences_squared = differences ** 2 # Quadrato della differenza

    mean_of_differences_squared = differences_squared.mean() #Media

    rmse_val = np.sqrt(mean_of_differences_squared) # Radice quadra

    return rmse_val #Valore output

```

C. Script Python: reti neurali

```
start = time.time()
# load the dataset
cartella = "C:\\\\Users\\\\...\\\\envs\\\\cer\\\\Scripts\\\\onde\\\\"
file = "30min_onde_corr_"
data2012 = pd.read_csv(cartella + file + "2012.txt", header=None, date_parser=[0])
data2013 = pd.read_csv(cartella + file + "2013.txt", header=None, date_parser=[0])
data2014 = pd.read_csv(cartella + file + "2014.txt", header=None, date_parser=[0])
data2015 = pd.read_csv(cartella + file + "2015.txt", header=None, date_parser=[0])
data2016 = pd.read_csv(cartella + file + "2016.txt", header=None, date_parser=[0])
data2017 = pd.read_csv(cartella + file + "2017.txt", header=None, date_parser=[0])
test2018 = pd.read_csv(cartella + file + "2018.txt", header=None, date_parser=[0])
test2019 = pd.read_csv(cartella + file + "2019.txt", header=None, date_parser=[0])
PV = pd.concat((data2012[1], data2013[1], data2014[1], data2015[1], data2016[1],
    data2017[1])).to_numpy() # --Dati Ponteverdi per il treaning--#
Cas = pd.concat((data2012[2], data2013[2], data2014[2], data2015[2], data2016[2],
    data2017[2])).to_numpy() # --Dati Casalmaggiore per il treaning--#
Col = pd.concat((data2012[3], data2013[3], data2014[3], data2015[3], data2016[3],
    data2017[3])).to_numpy() # --Dati Colorno per il treaning--#
data = pd.concat((pd.to_datetime(data2012[0]), pd.to_datetime(data2013[0]), pd.
    to_datetime(data2014[0]),
    pd.to_datetime(data2015[0]), pd.to_datetime(data2016[0]),
    pd.to_datetime(data2017[0]))) # --Date delle singole altezze--#
PV_test = pd.concat((test2018[1], test2019[1])).to_numpy() # --Dati Ponteverdi per il
    test--#
Cas_test = pd.concat((test2018[2], test2019[2])).to_numpy() # --Dati Casalmaggiore
    per il test--#
Col_test = pd.concat((test2018[3], test2019[3])).to_numpy() # --Dati Colorno per il
    test--#
data_t = pd.concat(
    (pd.to_datetime(test2018[0]), pd.to_datetime(test2019[0]))) # --Date delle singole
    altezze idriche per il test--#

# -- Validation data --
tracce = 1
datavalidation0 = np.zeros((2, 1))
datavalidation0 = np.concatenate((np.where(data == "01-06-2012 00:00"), np.where(
    data == "01-02-2013 00:00")))
datavalidation = np.zeros(0)
data_tre = pd.Series.tolist(data)
j = 0
for o in range(0, tracce, 1):
    exec(f'datavalidation = datavalidation{o} ')
    del data_tre[datavalidation[0, 0]: datavalidation[1, 0]]
data_tre = pd.Series(data_tre)
# ---Selezione onde--#
onda1 = np.concatenate((np.where(data_tre == "29-04-2012 00:00"), np.where(data_tre ==
    "17-05-2012 00:00")))
onda2 = np.concatenate((np.where(data_tre == "19-04-2013 00:00"), np.where(data_tre ==
    "08-06-2013 00:00")))
onda3 = np.concatenate((np.where(data_tre == "10-10-2014 00:00"), np.where(data_tre ==
    "18-10-2014 00:00")))
onda4 = np.concatenate((np.where(data_tre == "04-11-2014 00:00"), np.where(data_tre ==
    "26-12-2014 00:00")))
onda5 = np.concatenate((np.where(data_tre == "22-03-2015 00:00"), np.where(data_tre ==
    "8-04-2015 00:00")))
onda6 = np.concatenate((np.where(data_tre == "08-01-2016 00:00"), np.where(data_tre ==
    "17-01-2016 00:00")))
```

C.2. Rete neurale conclusiva

```

onda7 = np.concatenate((np.where(data_tre == "05-11-2016 00:00"), np.where(data_tre ==
    "10-11-2016 00:00")))
onda8 = np.concatenate((np.where(data_tre == "06-12-2017 00:00"), np.where(data_tre ==
    "21-12-2017 00:00")))
onda9 = np.concatenate((np.where(data_t == "04-03-2018 00:00"), np.where(data_t == "
    27-03-2018 00:00")))
onda10 = np.concatenate((np.where(data_t == "03-04-2018 00:00"), np.where(data_t == "
    18-04-2018 00:00")))
onda11 = np.concatenate((np.where(data_t == "21-06-2018 00:00"), np.where(data_t == "
    01-12-2018 00:00")))
onda12 = np.concatenate((np.where(data_t == "29-01-2019 00:00"), np.where(data_t == "
    16-02-2019 00:00")))
onda13 = np.concatenate((np.where(data_t == "22-04-2019 00:00"), np.where(data_t == "
    03-05-2019 00:00")))
onda14 = np.concatenate((np.where(data_t == "03-05-2019 00:00"), np.where(data_t == "
    11-06-2019 00:00")))
onda15 = np.concatenate((np.where(data_t == "21-10-2019 00:00"), np.where(data_t == "
    01-11-2019 00:00")))
onda16 = np.concatenate((np.where(data_t == "03-11-2019 00:00"), np.where(data_t == "
    08-11-2019 00:00")))
onda17 = np.concatenate((np.where(data_t == "14-11-2019 00:00"), np.where(data_t == "
    13-12-2019 00:00")))
onda18 = np.concatenate((np.where(data_t == "18-12-2019 00:00"), np.where(data_t == "
    01-01-2020 00:00")))
traning = 8
test = 18
reqm = np.zeros(test)
maxerror = np.zeros(test)
# --Variabili interne ai cicli for--#
niter = 10 + 1 # int(((a - da) / passo) + 1)**2) # --Quanti cicli for si
    effettueranno? meglio abbondare che deficere--#
rmse_test = np.zeros(niter)
acc = np.zeros(niter)
col = np.zeros(niter)
bat = np.zeros(niter)
ep = np.zeros(niter)
err = np.zeros(niter)
fin = np.zeros(niter)
cicl = np.zeros(niter)
neur = np.zeros(niter)
mederr = np.zeros(niter)
medrmse = np.zeros(test)
stdrmse = np.zeros(test)
reqm = np.zeros([test, niter - 1])
for k in range(1, test + 1, 1):
    exec(f'wave{k}= pd.DataFrame(np.zeros((onda{k}[1, 0] - onda{k}[0, 0], niter+1))).
        to_numpy()')
r = 0
REQM = np.zeros(test + 1) # RMSE
ErMAX = np.zeros(test + 1)
STDm = np.zeros(test + 1)
temp = 18 # -- prendo in considerazione serie temporali di 18 ore--#
# -- Cicli for--#
print('fine caricamento dati')
shift = 9 # ore
shiftmez = shift * 2 # -- !!! Dipende dalla discretizzazione dei dati di input, ad
    ora di mezzora, nel caso bisogner cambiare shiftmezz --#

```

C. Script Python: reti neurali

```
column = (temp - shift) * 2 # -- Dati precedenti --#
# -- Preparo variabili --#
PV_tren = pd.DataFrame(np.zeros((len(PV) - column, column))).to_numpy()
Cas_tren = pd.DataFrame(np.zeros((len(Cas) - column, column))).to_numpy()
Col_tren = pd.DataFrame(np.zeros((len(Col) - column, column))).to_numpy()
PV_testi = pd.DataFrame(np.zeros((len(PV_test) - column, column))).to_numpy()
Cas_testi = pd.DataFrame(np.zeros((len(Cas_test) - column, column))).to_numpy()
Col_testi = pd.DataFrame(np.zeros((len(Col_test) - column, column))).to_numpy()
X_val = pd.DataFrame(np.zeros((int(
(datavalidation0[1, 0] - datavalidation0[0, 0]) + (datavalidation1[1, 0] -
datavalidation1[0, 0])),
column * 3)).to_numpy()
Y_val = pd.DataFrame(np.zeros((int(
(datavalidation0[1, 0] - datavalidation0[0, 0]) + (datavalidation1[1, 0] -
datavalidation1[0, 0])), 1))).to_numpy()
for i in range(len(PV) - column): # -- Preparo set dati input di training --#
    k = 0
    for j in range(column, 0, -1):
        PV_tren[i, k] = PV[j + i]
        Cas_tren[i, k] = Cas[j + i]
        Col_tren[i, k] = Col[j + i]
        k = k + 1
for i in range(len(PV_test) - column): # -- Preparo set dati input di test --#
    k = 0
    for j in range(column, 0, -1):
        PV_testi[i, k] = PV_test[j + i]
        Cas_testi[i, k] = Cas_test[j + i]
        Col_testi[i, k] = Col_test[j + i]
        k = k + 1
# -- Applico lo scostamento temporale per prevedere il futuro --#
Y_tr = pd.DataFrame(Col).iloc[(shiftmez + column):]
y_test = pd.DataFrame(Col_test).iloc[(shiftmez + column):].to_numpy()
if shiftmez == 0:
    X_tr = pd.DataFrame(np.hstack((PV_tren, Cas_tren, Col_tren)))
    x_test = pd.DataFrame(np.hstack((PV_testi, Cas_testi, Col_testi))).to_numpy()
else:
    X_tr = pd.DataFrame(np.hstack((PV_tren, Cas_tren, Col_tren))).iloc[:-shiftmez]
    x_test = pd.DataFrame(np.hstack((PV_testi, Cas_testi, Col_testi))).iloc[:-shiftmez].
        to_numpy()
X_tr = X_tr.to_numpy()
Y_tr = Y_tr.to_numpy()
j = 0
for o in range(0, tracce, 1):
    exec(f'datavalidation = datavalidation{o} ')
    for ond in range(datavalidation[0, 0], datavalidation[1, 0], 1):
        X_val[j, :] = X_tr[ond, :]
        Y_val[j] = Y_tr[ond]
        X_tr = X_tr.drop(ond)
        Y_tr = Y_tr.drop(ond)
        j = j + 1
Y_tr = Y_tr.to_numpy()
X_tr = X_tr.to_numpy()
# -- Scelta set iperparametri --#
batch = 60
neu = 8
for ciclo in range(1, niter, 1):
    # -- define the keras model --#
```

C.2. Rete neurale conclusiva

```

model = Sequential()
model.add(Dense(neu, input_dim=(column * 3), activation='relu'))
model.add(Dense(1, activation='linear'))
# -- compile the keras model --#
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
# -- fit the keras model on the dataset --#
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
mc = ModelCheckpoint('b60n8shift'+ str(shift)+ '_'+str(ciclo)+ '_model.h5', monitor='val_loss', mode='min', verbose=1, save_best_only=True)
model.fit(X_tr, Y_tr, validation_data=(X_val, Y_val), epochs=100, batch_size=batch, verbose=0, callbacks=[es, mc])
saved_model = load_model('b60n8shift'+ str(shift)+ '_'+str(ciclo)+ '_model.h5')
# -- evaluate the keras model --#
_, accuracy = model.evaluate(X_tr, Y_tr)
print('Accuracy: %.2f' % (accuracy * 100), '%', '_end compile model')
# -- make class predictions with the model --#
predictions = model.predict(x_test)
predizione = model.predict(X_tr)
# -- Rimodellazione variabili per usarle nei grafici --#
o = np.zeros((len(y_test)))
y_ = np.zeros((len(y_test)))
p = np.zeros((len(Y_tr)))
Y_ = np.zeros((len(Y_tr)))
for j in range(len(y_test)):
    o[j] = predictions[j]
    y_[j] = y_test[j]
for j in range(len(Y_tr)):
    p[j] = predizione[j]
    Y_[j] = Y_tr[j]
Y_tr = Y_
y_test = y_
j = 0
# -- Plot e calcoli sulle onde scelte--#
# -- la separazione      dovuta alle differenti variabili per il set treaning e per il set test--#
onda = {}
for k in range(1, traning + 1, 1):
    exec(f'onda = onda{k} ')
    ordinate1 = np.zeros(onda[1, 0] - onda[0, 0])
    ordinate2 = np.zeros(onda[1, 0] - onda[0, 0])
    datee = np.zeros(onda[1, 0] - onda[0, 0])
    ascisse = data_tre.iloc[onda[0, 0]: onda[1, 0]]
    for ond in range(onda[0, 0], onda[1, 0], 1):
        ordinate1[j] = Y_tr[ond - shiftmez]
        ordinate2[j] = predizione[ond - shiftmez]
        datee[j] = j
        j = j + 1
    j = 0
    reqm[(k - 1), r] = rmse(ordinate2, ordinate1)
    exec(f'wave{k}[:, 0] = ordinate1')
    exec(f'wave{k}[:, ciclo] = ordinate2')
j = 0
# -- Onde test anni 18-19 --#
# -- la separazione      dovuta alle differenti variabili per il set treaning e per il set test--#
onda = {}
for k in range(traning + 1, test + 1, 1):

```

C. Script Python: reti neurali

```
exec(f 'onda = onda{k} ')
j = 0
ordinate1 = np.zeros(onda[1, 0] - onda[0, 0])
ordinate2 = np.zeros(onda[1, 0] - onda[0, 0])
datee = np.zeros(onda[1, 0] - onda[0, 0])
ascisse = data_t.iloc[onda[0, 0]: onda[1, 0]]
for ond in range(onda[0, 0], onda[1, 0], 1):
    ordinate1[j] = y_test[ond]
    ordinate2[j] = predictions[ond]
    datee[j] = j
    j = j + 1
j = 0
reqm[(k - 1), r] = rmse(ordinate2, ordinate1)
exec(f 'wave{k}[:, 0] = ordinate1')
exec(f 'wave{k}[:, ciclo] = ordinate2')
# -- salvataggio variabili per txt riassuntivo --
rmse_test[r] = rmse(o, y_test) # --Tutto il 18-19--
mederr[r] = st.mean(maxerror[range(traning, test, 1)])
acc[r] = accuracy
fin[r] = shift
cicl[r] = ciclo
r = r + 1
print('GIRO: %.2f' % r) # -- stampo a schermo la progressione dei i cicli for
    stanno progredendo --#
wave = {}
for t in range(1, test + 1, 1):
    exec(f 'wave = wave{t} ')
    exec(f 'onda = onda{t} ')
    j = 0
    ordinate1 = wave[:, 0]
    datee = np.zeros(onda[1, 0] - onda[0, 0])
    ordinate2 = np.zeros(len(wave))
    STD = np.zeros(len(wave))
    if t <= traning:
        ascisse = data_tre.iloc[onda[0, 0]: onda[1, 0]]
        dati = "Treanning set"
    else:
        ascisse = data_t.iloc[onda[0, 0]: onda[1, 0]]
        dati = "Test set"
    dates = matplotlib.dates.date2num(ascisse)
    ascisse = ascisse.array
    for q in range(0, len(wave), 1):
        datee[q] = q
        ru = pd.Series(wave[q, range(1, niter, 1)])
        ordinate2[q] = ru.mean()
        STD[q] = ru.std()
        exec(f 'wave{t}[[{q}], niter]= ordinate2[{q}]')
    ordinate1 = wave[:, 0]
    maxonda1 = max(ordinate1)
    maxonda2 = max(ordinate2)
    val = reqm[(t - 1), :]
    medrmse[t - 1] = val.mean()
    stdrmse[t - 1] = 2*val.std()
    REQM[t - 1] = rmse(ordinate1, ordinate2) # (st.mean((ordinate1 - ordinate2) ** 2)
        ** 0.5)
    STD[t - 1] = STD.mean()
```

C.2. Rete neurale conclusiva

```

ErMAX[t - 1] = abs(maxonda1 - maxonda2)
plt.subplots_adjust(bottom=0.2)
plt.xticks(rotation=25)
ax = plt.gca()
line1 = plt.plot(datee, ordinate1, label='dati reali', ls='--', linewidth=1)
line2 = plt.plot(datee, ordinate2, label='dati predetti', ls='--', linewidth=1)
line3 = plt.plot(datee, (ordinate2 + STD+STD), label='2STD sup', ls=':', mec='lime',
                  linewidth=0.7)
line4 = plt.plot(datee, (ordinate2 - STD-STD), label='2STD inf', ls=':', mec='lime',
                  linewidth=0.7)
plt.annotate(dati + " " + str(ascisse[0]) + '/' + str(ascisse[len(ascisse) - 1]),
             xy=(0.1, 0),
             xycoords='axes fraction',
             fontsize=9,
             xytext=(200, -25), textcoords='offset points',
             ha='right', va='top')
plt.annotate('onda n: ' + str(t), xy=(1, 0), xycoords='axes fraction', fontsize
            =9, xytext=(0, -25), textcoords='offset points', ha='right', va='top')
plt.legend(loc="upper right")
plt.grid()
plt.ylabel('[m] da zero idrometrico')
ax = plt.gca()
table_data = [
    ["Previsione(h):" + str(shift), " Media [m]", "2 std [m]",
     "rmse [m]",
     "errore picchi [m]"],
    ["Singoli RMSE", np.around(medrmse[t - 1], 3), np.around(stdrmse[t - 1], 3), "/",
     "/"],
    ["Media valori", "/", np.around(STDM[t - 1], 3), np.around(REQM[t - 1], 3), np.
     arround(ErMAX[t - 1], 3)]
]
table = ax.table(cellText=table_data, loc='bottom', bbox=[0.0, -0.45, 1, .28])
plt.subplots_adjust(bottom=0.2)
table.set_fontsize(9)
table.scale(1, 1)
plt.savefig("Onda_Col" + str(t) + "_prediz(h)" + str(shift) + "Finestra(h)" + str(
    temp) + '_Batch' + str(
batch) + '_Neu' + str(neu) + '.png',
            dpi=300, bbox_inches='tight')
plt.show()
# -- Save onda in txt mode --
ondaa = "Onda_Col" + str(t) + "_prediz(h)" + str(shift) + "Finestra(h)" + str(temp)
    + "_Batch" + str(
batch) + "_Neu" + str(neu) + ".txt"
with open(ondaa, "w") as out_file:
    for i in range(len(ascisse)):
        out_string = ""
        out_string += str(ascisse[i])
        out_string += "," + str(ordinate1[i]) # --ordinate2 il dato reale
        out_string += "," + str(ordinate2[i]) # --ordinate2 la predizione
        out_string += "," + str(ordinate2[i] + 2 * STD[i])
        out_string += "," + str(ordinate2[i] - 2 * STD[i])
        out_string += "\n"
        out_file.write(out_string)
MediaREQM = st.mean(REQM[range(8, 13, 1)])
Mediastd = 2*STD[range(8, 13, 1)].mean()
MediaErMAX = st.mean(ErMAX[range(8, 13, 1)])

```

C. Script Python: reti neurali

```
print('END, SAVE ALL RMSE') # -- fine dei cicli for--#
end = time.time()
t = str((end - start) / 60)
print('tempo: ' + t + ' [min] _end prediction model') # -- stampa a schermo il tempo
    impiegato--#
dat = "Colorno3_Finestra(h)" + str(temp) + "_recap_Batch" + str(batch) + "_Neu" + str(neu) + ".txt" # -- salvataggio in txt delle variabili riassuntivi dei cicli for--#
with open(dat, "w") as out_file:
    for t in range(1, test + 1, 1):
        out_string = ""
        out_string += str(t)
        out_string += "," + str(np.around(STDM[t - 1], 3)) # --std sulla media
        out_string += "," + str(np.around(REQM[t - 1], 3)) # --RMSE sulla media
        out_string += "," + str(np.around(medrmse[t - 1], 3)) # --media degli RMSE
        out_string += "," + str(np.around(stdrmse[t - 1], 3)) # --std sugli RMSE
        out_string += "\n"
        out_file.write(out_string)
frequency = 900 # Set Frequency To 2500 Hertz
duration = 900 # Set Duration To 1000 ms == 1 second
winsound.Beep(frequency, duration)
winsound.Beep(frequency, duration)
```

C.3 Caricamento modello neurale salvato

```
# load model to predict 2020 without train the entire neural networkimport matplotlib
from keras.models import load_model
import winsound
import numpy as np
import matplotlib.pyplot as plt # per creare i grafici
import matplotlib.dates as md
import time # per vedere quanto impiega la rete neurale per giungere a soluzione
import pandas as pd
def rmse(predictions, targets):
    differences = predictions - targets # Differenza.
    differences_squared = differences ** 2 # Quadrato della differenza
    mean_of_differences_squared = differences_squared.mean() #Media
    rmse_val = np.sqrt(mean_of_differences_squared) # Radice quadra
    return rmse_val #Valore output
start = time.time()
cartella = "C:\\\\Users\\\\...\\\\envs\\\\cer\\\\Scripts\\\\onde\\\\"
file = "30min_onde_corr_"
test2020 = pd.read_csv(cartella + file + "2020.txt", header=None, date_parser=[0])
PV_test = test2020[1].to_numpy() # --Dati Ponteverdi per il test--#
Cas_test = test2020[2].to_numpy() # --Dati Casalmaggiore per il test--#
Col_test = test2020[3].to_numpy() # --Dati Colorno per il test--#
data_t = pd.to_datetime(test2020[0]) # --Date delle singole altezze idriche per il
    test--#
onda1 = np.concatenate((np.where(data_t == "29-02-2020 00:00"), np.where(data_t == "
    18-03-2020 00:00")))
onda2 = np.concatenate((np.where(data_t == "03-12-2020 00:00"), np.where(data_t == "
    13-12-2020 00:00")))
test = 2
reqm = np.zeros(test)
maxerror = np.zeros(test)
# --Variabili interne ai cicli for--#
```

C.3. Caricamento modello neurale salvato

```

niter = 10 + 1 # int(((a - da) / passo) + 1)**2) # --Quanti cicli for si
# effettueranno? meglio abbondare che deficere --#
rmse_test = np.zeros(niter)
acc = np.zeros(niter)
col = np.zeros(niter)
bat = np.zeros(niter)
ep = np.zeros(niter)
err = np.zeros(niter)
fin = np.zeros(niter)
cicl = np.zeros(niter)
neur = np.zeros(niter)
mederr = np.zeros(niter)
medrmse = np.zeros(test)
stdrmse = np.zeros(test)
reqm = np.zeros([test, niter - 1])
for k in range(1, test + 1, 1):
    exec(f'wave{k}= pd.DataFrame(np.zeros((onda{k}[1, 0] - onda{k}[0, 0], niter+1))).
        to_numpy() ')
r = 0
REQM = np.zeros(test + 1) # RMSE
ErMAX = np.zeros(test + 1)
STDm = np.zeros(test + 1)
temp = 18 # -- prendo in considerazione serie temporali di 18 ore--#
# -- Cicli for--#
print('fine caricamento dati')
shift = 6 # ore
shiftmez = shift * 2 # -- !!! Dipende dalla discretizzazione dei dati di input, ad
# ora di mezzora, nel caso bisogner cambiare shiftmezz --#
column = (temp - shift) * 2 # -- Dati precedenti --#
# -- Preparo variabili --#
PV_testi = pd.DataFrame(np.zeros((len(PV_test) - column, column))).to_numpy()
Cas_testi = pd.DataFrame(np.zeros((len(Cas_test) - column, column))).to_numpy()
Col_testi = pd.DataFrame(np.zeros((len(Col_test) - column, column))).to_numpy()
for i in range(len(PV_test) - column): # -- Preparo set dati input di test --#
    k = 0
    for j in range(column, 0, -1):
        PV_testi[i, k] = PV_test[j + i]
        Cas_testi[i, k] = Cas_test[j + i]
        Col_testi[i, k] = Col_test[j + i]
    k = k + 1
# -- Applico lo scostamento temporale per prevedere il futuro --#
y_test = pd.DataFrame(Col_test).iloc[(shiftmez + column):].to_numpy()
if shiftmez == 0:
    x_test = pd.DataFrame(np.hstack((PV_testi, Cas_testi, Col_testi))).to_numpy()
else:
    x_test = pd.DataFrame(np.hstack((PV_testi, Cas_testi, Col_testi))).iloc[:-shiftmez].
        to_numpy()
j = 0
# --Ciclo di previsione --#
for ciclo in range(1, niter, 1):
    model = load_model('b60n8shift' + str(shift) + '_' + str(ciclo) + '_model.h5')
    predictions = model.predict(x_test)
    o = np.zeros((len(y_test)))
    y_ = np.zeros((len(y_test)))
    for j in range(len(y_test)):
        o[j] = predictions[j]
        y_[j] = y_test[j]

```

C. Script Python: reti neurali

```
y_test = y_
j = 0
# -- Plot e calcoli sulle onde scelte--#
# -- la separazione dovuta alle differenti variabili per il set treaning e per
# il set test--#
onda = {}
for k in range(1, test + 1, 1):
    exec(f'onda = onda[{k}] ')
    j = 0
    ordinate1 = np.zeros(onda[1, 0] - onda[0, 0])
    ordinate2 = np.zeros(onda[1, 0] - onda[0, 0])
    datee = np.zeros(onda[1, 0] - onda[0, 0])
    ascisse = data_t.iloc[onda[0, 0]: onda[1, 0]]
    for ond in range(onda[0, 0], onda[1, 0], 1):
        ordinate1[j] = y_test[ond]
        ordinate2[j] = predictions[ond]
        datee[j] = j
        j = j + 1
    j = 0
    reqm[(k - 1), r] = rmse(ordinate2, ordinate1)
    exec(f'wave{k}[:, 0] = ordinate1')
    exec(f'wave{k}[:, ciclo] = ordinate2')
# -- salvataggio variabili per txt riassuntivo--#
rmse_test[r] = rmse(o, y_test) # --Tutto il 18-19--#
fin[r] = shift
cicl[r] = ciclo
r = r + 1
print('GIRO: %.2f' % r) # -- stampo a schermo la progressione dei i cicli for
# stanno progredendo --#
wave = {}
for t in range(1, test + 1, 1):
    exec(f'wave = wave{t} ')
    exec(f'onda = onda{t} ')
    j = 0
    ordinate1 = wave[:, 0]
    datee = np.zeros(onda[1, 0] - onda[0, 0])
    ordinate2 = np.zeros(len(wave))
    STD = np.zeros(len(wave))
    ascisse = data_t.iloc[onda[0, 0]: onda[1, 0]]
    dati = "new data"
    dates = matplotlib.dates.date2num(ascisse)
    ascisse = ascisse.array
    for q in range(0, len(wave), 1):
        datee[q] = q
        ru = pd.Series(wave[q, range(1, niter, 1)])
        ordinate2[q] = ru.mean()
        STD[q] = ru.std()
        exec(f'wave{t}[[{q}], niter]= ordinate2[{q}] ')
    ordinate1 = wave[:, 0]
    maxonda1 = max(ordinate1)
    maxonda2 = max(ordinate2)
    val = reqm[(t - 1), :]
    medrmse[t - 1] = val.mean()
    stdrmse[t - 1] = 2 * val.std()
    REQM[t - 1] = rmse(ordinate1, ordinate2) # (st.mean((ordinate1 - ordinate2) ** 2)
    ** 0.5)
    STD[t - 1] = STD.mean()
```

C.3. Caricamento modello neurale salvato

```

ErMAX[t - 1] = abs(maxonda1 - maxonda2)
plt.subplots_adjust(bottom=0.2)
plt.xticks(rotation=25)
ax = plt.gca()
# xfmt = md.DateFormatter('%Y-%m-%d')
# ax.xaxis.set_major_formatter(xfmt)
line1 = plt.plot(datee, ordinate1, label='dati reali', ls='-', linewidth=1)
line2 = plt.plot(datee, ordinate2, label='dati predetti', ls='-', linewidth=1)
line3 = plt.plot(datee, (ordinate2 + STD+STD), label='2STD sup', ls=':', mec='lime',
                  linewidth=0.7)
line4 = plt.plot(datee, (ordinate2 - STD-STD), label='2STD inf', ls=':', mec='lime',
                  linewidth=0.7)
plt.annotate(dat + " " + str(ascisse[0]) + ' / ' + str(ascisse[len(ascisse) - 1]),
             xy=(0.1, 0), xycoords='axes fraction', fontsize=9, xytext=(200, -25), textcoords
             ='offset points', ha='right', va='top')
plt.annotate('onda n: ' + str(t), xy=(1, 0), xycoords='axes fraction', fontsize=9,
             xytext=(0, -25), textcoords='offset points', ha='right', va='top')
plt.legend(loc="upper right")
plt.grid()
plt.ylabel('[m] da zero idrometrico')
ax = plt.gca()
table_data = [
    ["Previsione(h):" + str(shift), " Media [m]", "2 std [m]",
     "rmse [m]",
     "errore picchi [m]"],
    ["Singoli RMSE", np.around(medrmse[t - 1], 3), np.around(stdrmse[t - 1], 3), "/",
     "/"],
    ["Media valori", "/", np.around(STDm[t - 1], 3), np.around(REQM[t - 1], 3), np.
     arround(ErMAX[t - 1], 3)]
]
table = ax.table(cellText=table_data, loc='bottom', bbox=[0.0, -0.45, 1, .28])
plt.subplots_adjust(bottom=0.2)
table.set_fontsize(9)
table.scale(1, 1)
plt.savefig("Onda_Col" + str(t) + "_prediz(h)" + str(shift) + "Finestra(h)" + str(
    temp) + '_Batch' + str(
    60) + '_Neu' + str(8) + '.png',
            dpi=300, bbox_inches='tight')
plt.show()
# -- Save onda in txt mode --#
ondaa = "Onda_Col" + str(t) + "_prediz(h)" + str(shift) + "Finestra(h)" + str(temp)
+ "_Batch" + str(
60) + "_Neu" + str(8) + ".txt"
with open(ondaa, "w") as out_file:
    for i in range(len(ascisse)):
        out_string = ""
        out_string += str(ascisse[i])
        out_string += "," + str(ordinate1[i]) # --ordinate2 il dato reale
        out_string += "," + str(ordinate2[i]) # --ordinate2 la predizione
        out_string += "," + str(ordinate2[i] + 2 * STD[i])
        out_string += "," + str(ordinate2[i] - 2 * STD[i])
        out_string += "\n"
        out_file.write(out_string)

print('END, SAVE ALL RMSE') # -- fine dei cicli for--#
end = time.time()
t = str((end - start) / 60)

```

C. Script Python: reti neurali

```
print('tempo: ' + t + ' [min] _end prediction model') # -- stampa a schermo il tempo  
    impiegato--#  
dat = "Colorno3_Finestra(h)" + str(temp) + "_recap_Batch" + str(60) + "_Neu" + str(  
8) + ".txt" # -- salvataggio in txt delle variabili riassuntivi dei cicli for--#  
with open(dat, "w") as out_file:  
for t in range(1, test + 1, 1):  
    out_string = ""  
    out_string += str(t)  
    out_string += "," + str(np.around(STDM[t - 1], 3)) # --std sulla media  
    out_string += "," + str(np.around(REQM[t - 1], 3)) # --RMSE sulla media  
    out_string += "," + str(np.around(medrmse[t - 1], 3)) # --media degli RMSE  
    out_string += "," + str(np.around(stdrmse[t - 1], 3)) # --std sugli RMSE  
    out_string += "\n"  
    out_file.write(out_string)  
frequency = 900 # Set Frequency To 2500 Hertz  
duration = 900 # Set Duration To 1000 ms == 1 second  
winsound.Beep(frequency, duration)  
winsound.Beep(frequency, duration)  
end = time.time()  
t = str((end - start))  
print('tempo: ' + t + ' [sec] _end prediction model') # -- stampa a schermo il tempo  
    impiegato--#
```

C.4 Tempi di elaborazione a differenti batch

Batch	Minuti
2	37.9347
4	26.9983
8	17.4251
10	14.03639
20	7.4063
30	4.2297
40	3.7401
50	3.5806
60	2.7151
70	3.0963
80	2.7968
90	2.555
100	5.51537

Ringraziamenti

Si conclude qui un capitolo, fatto di gioie, ansie e periodi belli e brutti. Ma così è la vita, fatta di alti e bassi, quello che è certo è stata la determinazione con cui sono finalmente giunto a questo risultato.

Vorrei ringraziare il professore Paolo Mignosa, l'ingegner Renato Vacondio e l'ingegner Susanna Dazzi che hanno reso possibile concretamente questa tesi, la loro conoscenza e disponibilità sarà per me sempre un esempio da seguire. Grazie ai miei genitori, mi avete insegnato l'arte del vivere. E lo avete fatto con dolcezza e con amore.

Grazie ad Alessandro, che anche se spesso cozziamo so di poter sempre contare su dite. Grazie ai miei nonni e i miei parenti vicini e lontani, mi avete sicuramente instradato e accompagnato in questo percorso. Un ringraziamento speciale lo vorrei anche dedicare ad Antonietta, hai visto quello che nessuno prima aveva visto e hai tirato fuori il meglio di me. Grazie!

Grazie a Federica e Valentina senza le quali i drammi e le gioie della mia vita non avrei saputo condividere con nessuno. Grazie a Frencio, amico di cui difficilmente posso far a meno di interpellare in ogni momento importante della mia vita.

Grazie a tutti i miei amici vicini e lontani, con cui ho condiviso un progetto, parte di vita o che ci vediamo ogni giorno, potrei fare un elenco numerato e ringraziarvi uno ad uno ma sarebbe alquanto dispendioso, in termini ecologici e di tempi, vi ringrazierò appena ci sarà occasione di persona.

Un ringraziamento lo vorrei dedicare inoltre a tutti quelli che in un modo o nell'altro non hanno creduto in me e nelle mie capacità. È proprio grazie ai rifiuti, alle disapprovazioni e alle critiche che ho trovato la mia via, il modo migliore di esprimermi, di migliorare e definire dove posso eccellere.