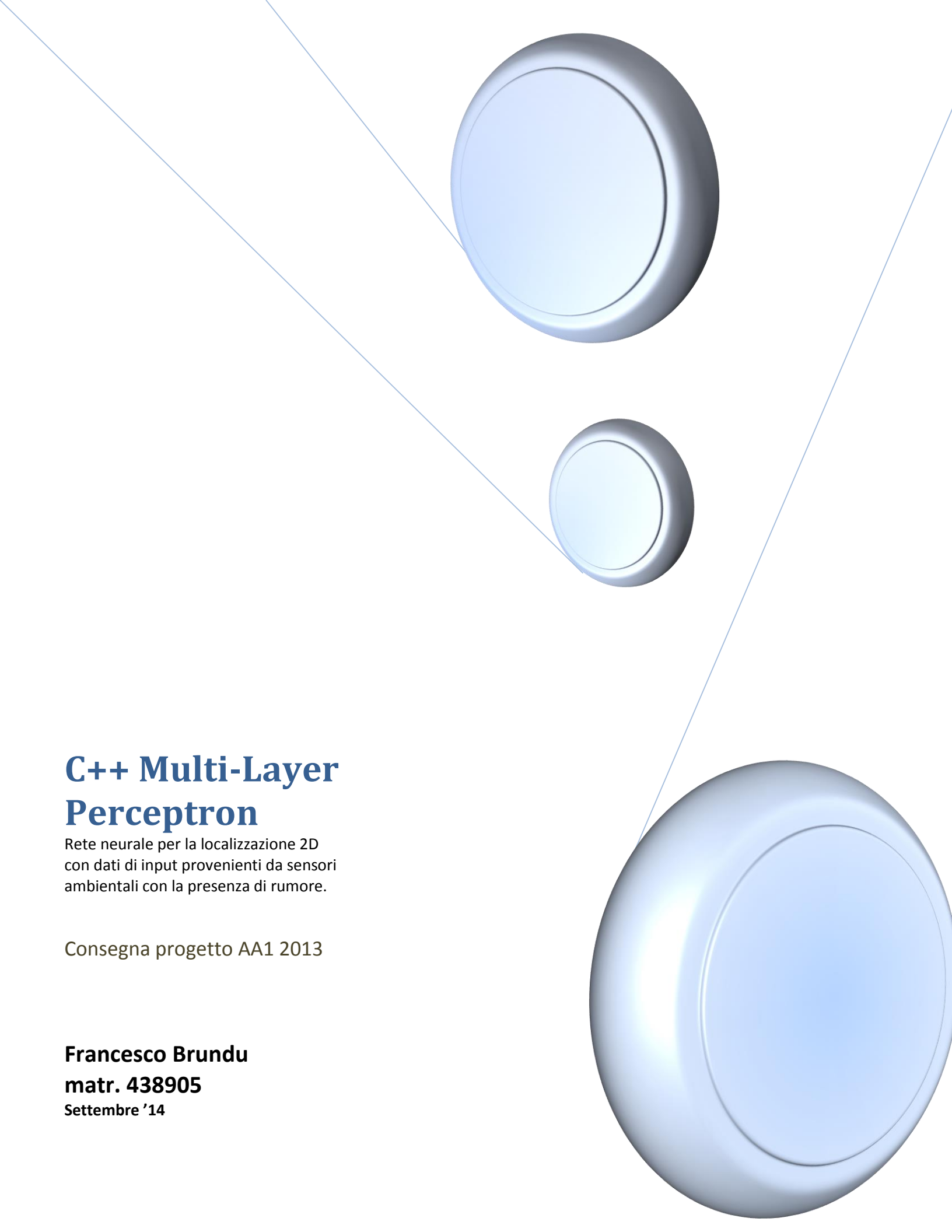


C++ Multi-Layer Perceptron

Rete neurale per la localizzazione 2D con dati di input provenienti da sensori ambientali con la presenza di rumore.

Consegna progetto AA1 2013

Francesco Brundu
matr. 438905
Settembre ’14

Sommario

Introduzione 2
Metodo e scelte di design 3
Model selection e valutazione del modello prodotto 4
Struttura del codice 5
Design delle sperimentazioni 6
Risultati sperimentali 7
  Collaudo della rete 7
  Monk 1 7
  Monk 2 9
  Monk 3 9
  Regressione 1 11
Cup task 12
  Analisi preliminari e preprocessing 12
  Risultati della rete implementata e dei tool 15
Conclusioni 20
Bibliografia & Link 20

Introduzione

Per la Cup è richiesto di sviluppare una rete neurale feedforward multi-Layer, per risolvere un task di regressione su due variabili target, rappresentanti delle coordinate 2D (x,y).

È fornito un training set con 990 esempi di training nel formato <id, var1, var2, var3, var4, var5, target X, target Y> dove la colonna 1 nome pattern (id), le 5 colonne centrali sono variabili a valori continui (da sensori ambientali) è presente del rumore, e le 2 colonne finali sono il target di due valori continui che rappresentano la posizione sull'asse X e Y rispettivamente.

È fornito anche un blind test set: 353 dati, stesso formato di input, mancano le 2 colonne di target.

Per l'apprendimento del modello si usa l'algoritmo back-propagation nella sua versione in-line (stocastico).

La loss usata per la valutazione della performance nella competition è il Mean Euclidean Error:

$$E_{MEE} = \frac{1}{N}\sum_{p=1}^{N}\left\|o_p - t_p\right\|_2 = \frac{1}{N}\sum_{p=1}^{N}\sqrt{(o_{p,x} - t_{p,x})^2 + (o_{p,y} - t_{p,y})^2}$$

La loss per il training è il (Root) Mean Squared Error (loss del least means square)

$$E_{MSE} = \frac{1}{N}\sum_{p=1}^{N}(o_p - t_p)^2 = \frac{1}{N}\sum_{p=1}^{N}\left((o_{p,x} - t_{p,x})^2 + (o_{p,y} - t_{p,y})^2\right)$$

Mentre la loss modificata per Weight decay è:

$$E_{wd}(w) = E_{MSE}(w) + \frac{\lambda}{N}\mathrm{w}^{\mathrm{H}}w = E_{MSE}(w) + \frac{\lambda}{N}\sum_{l=1}^{L}\sum_{i=0}^{\mathrm{d}^{(l-1)}}\sum_{j=1}^{\mathrm{d}^{(l)}}(w_{ij}^{(l)})^2$$

La rete neurale sviluppata è stata prima testata su dei dataset di prova (monk) per risolvere dei problemi di classificazione con e senza la presenza di rumore.

In seguito sono stati creati dei dataset di prova per dei problemi di regressione con e senza la presenza di rumore.

In fine verificate le capacità della rete neurale a risolvere i problemi descritti in precedenza, la rete è stata allenata sul dataset per risolvere il task della competition. Durante i vari task le performance della rete sviluppata sono state confrontate con quelle di tool esterni messi a disposizione da knime.

I vari dataset sopra citati saranno descritti successivamente. Per il pre-processing dei dati ho usato il software knime e excell, usato anche per il confronto dei risultati con altri modelli, il post-processing e la visualizzazione dei dati e dei risultati.

Metodo e scelte di design

La rete neurale (feedforward multi layer perceptron) implementata è una rete a più livelli con le unità di un livello connesse con tutte quelle del livello successivo. L'utente tramite una stringa decide il numero di livelli e il numero di unità per livello della rete, un numero rappresentante il numero di unità del livello separato con uno spazio dal numero rappresentante le unità del livello successivo. Per esempio "2 4 1" per indicare una rete con 2 unità di input, 4 unità nascoste e un unità di output. In fase di creazione della rete per ogni livello, oltre a quelle indicate dall'utente, viene aggiunta un'ultima unità di bias con un output fissato a 1. I pesi associati alle sinapsi che collegano un neurone a tutti quelli del livello successivo sono modellati con un vettore di pesi interno ad ogni unità. Ogni livello/layer è implementato come un vettore di unità/neuroni e la rete è un vettore di layer. Per rendere il codice più uniforme ed evitare complessità superflue, tutte le unità e i layer sono uguali, il che implica che anche le unità di output hanno un vettore dei pesi in uscita, che verrà ignorato, ed una unità di baias anch'essa ignorata. Per la cup è stata usata la rete con un livello nascosto.

Per l'allenamento della rete si usa l'algoritmo standard di back-propagation nella sua versione On-line. La funzione di attivazione adottata è la funzione logistica/sigmoide, questa velocizza l'algoritmo di backpropagation dato che il valore di output dell'unità calcolato nella fase feedforward è riusato per il calcolo dei gradienti delle unità di output.

La rete prevede diverse modalità di allenamento, una tramite la funzione train, che implementa il training proposto per questi task, e una tramite le funzioni feedforward, backprop, getresults ecc. che permettono di definire un'altra modalità di apprendimento secondo altri criteri.

All'inizio del training i pesi sono inizializzati con valori random piccoli (-0.5, 0.5), questo permette all'algoritmo di partire ogni volta da un ipotesi diversa, così due allenamenti consecutivi sugli stessi dati non finiscono sullo stesso minimo locale. Questo permette di calcolare una stima dell'errore commesso da un modello anche come l'errore medio tra vari "trial" con gli stessi dati e parametri.

Un'altra accortezza adottata è quella di randomizzare l'ordine dei pattern di training prima di ogni epoca.

La regola di aggiornamento dei pesi è stata provata in varie versioni, la prima derivata dall'errore MSE e la seconda derivata dal MEE, è stata scelta la prima in quanto fornisce una convergenza più rapida rispetto la seconda che divide il delta ottenuto per le unità di output dalla prima per la norma dell'errore totale commesso. La funzione minimizzata dall'algoritmo di training è E_{MSE} nel caso in cui i dati non presentano rumore, e nel caso di dati rumorosi una versione modificata E_{wd} con una componente di regolarizzazione basata sul valore dei pesi della rete, weight decay, che favorisce soluzioni con pesi piccoli. L'effetto di questo regolarizzatore è quello di ridurre le non linearità/salti bruschi nella funzione calcolata, con l'assunzione che il rumore favorisca invece tali soluzioni. Si passa dall'una all'altra modificando appositamente il parametro lambda, es. con 0 non viene contata la penalità sulla complessità e si ottiene la backpropagation standard. In entrambe le regole di aggiornamento dei pesi si fa uso del momento.

La rete può risolvere problemi sia di classificazione che di regressione, decidibile tramite un parametro al momento della creazione della rete. Nel caso di un problema di regressione si usano delle unità di output lineari, il che si traduce nella fase di feedforward nell'uso per tali unità della funzione di attivazione identità, nella fase di backpropagation la derivata per il calcolo dell'errore delle unità di output è quindi 1.

La rete proposta può usare insieme regolarizzazione indiretta (early stopping) e diretta (weight decay).

Principali criteri di stop presi in considerazione (verificati ogni k = 3 epoche):

- Numero eccessivo di epoche;
- Loss sul validation set interno inferiore a una data soglia; (metodo non usato per il task della Cup)
- Il gradiente è inferiore a una data soglia (es. 0.0001), implicando una punto di minimo locale; (metodo non usato per il task della Cup)
- Nel caso di **early stop** si verifica l'incremento dell'errore su un validation set interno, salvando i pesi della rete nel momento in cui si presenta un nuovo minimo, dopo di cui si prosegue per un prefissato numero di epoche, e se in questo intervallo non si è presentato un nuovo minimo si può interrompere il training; I pesi salvati sono ripristinati e il modello restituito.

Il validation set interno alla procedura di training nel caso di early stopping è creato all'interno del metodo train ed è tenuto separato dai dati con cui viene allenata la rete.

Normalizzazione z-score:

$$Normalized\ (e_i) = \frac{e_i - \bar{E}}{std(E)}$$

Normalizzazione [0,1] degli input e dei target secondo la funzione:

$$Normalized\ (e_i) = \frac{e_i - E_{min}}{E_{max} - E_{min}}$$

Dove: E_{min} = valore minimo per la variabile E. E_{max} = valore massimo per la variabile E.

Denormalizzazione dei target e output interna al codice, dove - E_{min} corrisponde a translate ed E_{max} - E_{min} a scale. Funzione usata per calcolare l'errore MEE nella scala originale, procedimento usato solo nella costruzione del modello finale e nell'output dei risultati del blind test.

Model selection e valutazione del modello prodotto

La stima dell'errore su ogni modello è stata pensata e implementata con un procedimento di cross validatione con k variabile e validazione semplice. Funzionalità messe a disposizione dalle classi sviluppate per la gestione dei dataset.

Nel caso di cross validazione scelgo un k = 3 o 5. Un k piccolo produce un bias dovuto al validation set, un'euristica comune è quella di scegliere k = 10, ma computazionale troppo costoso. Si divide il dataset in k sezioni, a turno se ne usa una per la stima dell'errore di generalizzazione e le restanti per il training del modello. In fine l'errore stimato di un dato modelo è l'errore di generalizzazione medio su tutti i fold. Mentre nel caso di validazione semplice (holdout) uso 1/3 dei dati come validation set e il restante come training set, e la stima è fatta sul singolo validation set, in questo caso si ha una stima meno accurata, ma il tempo per effettuarla è notevolmente inferiore. Per la model selection quindi andrò ad usare un holdout.

Per stimare l'errore lo stesso fold è eseguito 3 volte (10 trials è troppo costoso), così da avere diverse ipotesi di partenza ed arrivare ad un minimo locale differente, e si stima l'errore su quel fold come la media sui trials. Tramite la procedura appena descritta scelgo il modello finale.

I modelli tra qui scegliere sono definiti dagli iper-parametri scelti, la scelta è fatta provando varie combinazioni tra una griglia di valori da associare ai vari parametri da scegliere stimandone l'efficacia tramite il processo di validazione.

Iper-parametri ottimizzati con una griglia di valori, euristica scelta provando i vari valori limite, e quelli scelti sembrano essere abbastanza rappresentativi:

- numero di unità nascoste scelto tra: {min,..,max} dove min e max variano a seconda della necessità;
- η learning rate scelto tra: $\{1/2^i\}_i$, in seguito ho usato la griglia {0.5, 0.1, 0.01, 0.001};
- α scelto tra: {1/2^i} in seguito ho invece usato la griglia {0.0, 0.1, 0.5, 0.7};
- λ scelto tra: {1/2^i} in seguito ho usato la griglia {0.1, 0.01, 0.001, 0}.

Una variante proposta dal haykin per la scelta del numero di neuroni nascosti, cercando il minor numero di neuroni necessario per raggiungere le performance di un classificatore bayesiano (entro l'1%), partendo da 2 neuroni e incrementando se necessario. Io ho usato una variante in cui mi confronto anche con un MLP rprop.

Per la scelta dei valori per il parametro di learning η e la costante α del momento prevede η,α che in media:

- convergono ad un minimo locale nel minor numero di epoche;
- o che ottengono la miglio generalizzazione nel minor numero di epoche.

Un miglioramento è usare questa procedura per rilevare un intervallo su cui effettuare poi una ricerca con una grana più accurata. Questa proposta non è stata messa in atto sempre a causa dei lunghi tempi di elaborazione, ma solo quando descritto nel capitolo sperimentale.

La funzione di test prevede un parametro che abilita la stampa su file "error.csv" dell'errore di training e di validazione (nel caso di validation set interno al training) e la stampa dei risultati su "output.csv" così da poter fare le analisi e i grafici di confronto tra output della rete e target. Il blind test invece stampa i risultati su "finalout.csv". La rete accetta l'input per il training e la validazione tramite un file csv "input.txt" e l'input per il test tramite "test.txt".

Struttura del codice

Descrivo brevemente quali sono le componenti principali sviluppate, disponibili come allegato e ampiamente commentate nel codice. I commenti nel codice sono dati in lingua inglese, così come tutto il resto del codice.

Il codice è stato progettato per essere modulare così da poter sostituire o modificare delle funzionalità quali per esempio la funzione di attivazione di un'unità senza dover toccare il codice relativo alla gestione della rete.

Classe Neuron: Rappresenta l'entità base della rete. Implementa le funzionalità base di un unità, quali calcolare l'output tramite un'opportuna funzione di attivazione definita internamente alla classe, o aggiornare i pesi delle connessioni in input in base alla formula scelta.

Classe Net: classe che implementa la rete vera e propria e ne coordina il funzionamento e l'apprendimento, tramite la composizione di neuroni. Coordina le operazioni learning quali feedforward e backprop, e quelle di test o applicazione della rete.

Classe csvdataloader: Offre delle funzionalità base per leggere un file csv riga per riga, evitando i commenti e le righe vuote.

Classe datasetmanager: Tramite il csvdataloader carica il dataset all'interno di una matrice, così da caricare il dataset in memoria. Offre anche la possibilità di eseguire una cross validation di k fold definiti nel costruttore, mettendo a disposizione un iteratore su fold che per ogni fold inizializza un train e un validation dataset.

È stato sviluppato a parte anche il codice per la creazione del task sintetico di regressione da usare per il collaudo della rete. Quest'ultimo sarà descritto nel prossimo capitolo.

Mentre fungenerator è il codice relativo la creazione del dataset sintetico di regressione, molto semplicemente crea due file funout_test.csv e funout.csv nei quali salva rispettivamente la funzione senza il rumore e quella con il rumore.

Design delle sperimentazioni

Per lo sviluppo, e la verifica della rete in preparazione al task della competition sono previste tre fasi di collaudo e sperimentazione.

1. Fase di collaudo della rete durante l'implementazione con un dataset personalizzato rappresentante la funzione xor, 2 input e un target senza errori;
2. Fase di collaudo della rete con i 3 problemi di classificazione Monk, con 6 attributi discreti in codifica 1ofk come input e un target binario, i primi due problemi sono senza rumore, mentre il terzo no, così in quest'ultimo caso si può provare con delle tecniche di regolarizzazione a migliorare il risultato ottenuto senza; tali tecniche come prova dovranno poi essere in grado di apprendere anche il concetto espresso dai due problemi precedenti;
3. Fase di collaudo della rete con un problema sintetico di regressione su due variabili target dato un input, il dataset è rumoroso; Questa è l'ultimo collaudo su un dataset diverso da quello della competition.

I problemi Monk sono problemi di classificazione su sei attributi discreti e ogni dato di training/test è della forma: < attribute#1> < attribute#2> < attribute#3> < attribute#4> < attribute#5> < attribute#6> -> <class>

attribute#1, attribute#2, attribute#4 ϵ {1, 2, 3}

attribute#3, attribute#6 ϵ {1, 2} attribute#5 ϵ {1, 2, 3, 4} class ϵ {0,1}

1. MONK-1: (attribute_1 = attribute_2) or (attribute_5 = 1)
2. MONK-2: (attribute_n = 1) per esattamente due n (scelti in {1,2,...,6})
3. MONK-3: (attrib_5 = 3 and attrib_4 =1) or (attrib_5 != 4 and attrib_2 != 3)
   3.1. con 5% di rumore

Una descrizione dettagliata dei monk si trova sul sito delle repository [6].

Risultati sperimentali

Collaudo della rete

Prima prova di collaudo con un dataset senza rumore rappresentante la funzione xor. La rete è stata usata nella sua versione più semplice con due unità di input, tre unità nascoste e una di output con fun di attivazione sigmoidale, con eta 0.3 e alfa 0.1, il criterio di stop usato misura la loss sul training set confrontandone il valore con la soglia simbolica di 0.001. Questo è stato possibile grazie al fatto che il dataset non presenta rumore ed è rappresentativo della funzione obbiettivo da cercare. Dopo circa 800 epoche la loss (MSE) assume valori sotto la soglia prefissata.

Questo dovuto al target scelto nel dataset che è 1 e 0 rispettivamente positivo e negativo, infatti modificando leggermente i target rispettivamente a 0.9 e 0.1 si ha una convergenza più rapida senza saturare i pesi delle unità in circa 600 epoche.

Con una funzione di attivazione in output lineare, eseguendo così una regressione si ottiene un errore (MSE) di training dopo circa 120 epoche < 0.001.

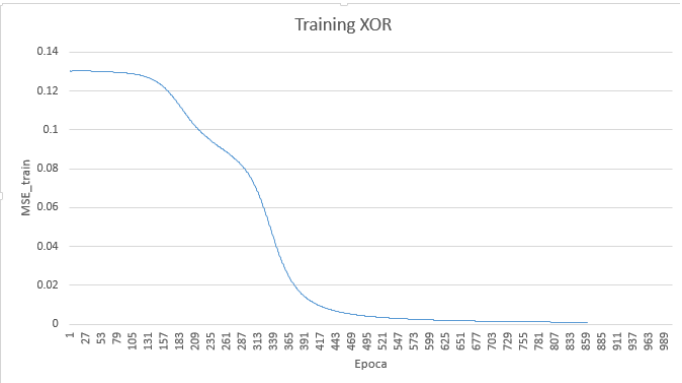


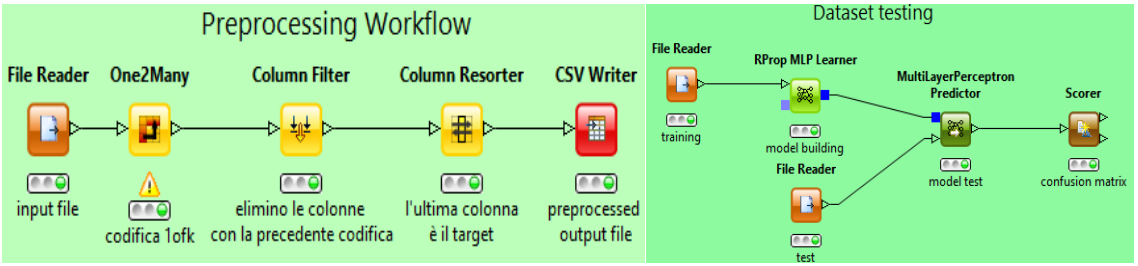
Figure 1- Training con output unit sigmoidale.

Figure 2- training con output unit lineare.

Monk 1

Il dataset Monk 1 è stato applicato a una rete con 4 unità nascoste, un learning rate di 0.3 e un coefficiente alfa per il momentum pari a 0.3.

L'input preso dalla repository, nel formato con le colonne separate da uno spazio, è stato preprocessato in un formato compatibile con knime, eliminando il primo spazio che è riconosciuto come colonna vuota, e a questo punto le features di input sono state codificate 1ofK, producendo 18 features binarie di cui l'ultima come target. Il file di training usato è il csv risultante da questo preprocessing.



Per assicurarmi del corretto funzionamento del preprocessing verifico l'effettiva correttezza dei dati tramite un tool di knime esterno che implementa un multi layer perceptron; L'algoritmo di apprendimento usato è RProp, 4 hidden unit (con 2 e 3 non riesce ad apprendere correttamente la funzione obiettivo) e 1000 iterazioni limite. L'apprendimendo converge con meno di 90 epoche raggiungendo il 100% di accuracy.

Verifico a questo punto se la mia rete riesce a fare altrettanto. Le unità hanno la funzione di attivazione di tipo sigmoidale. Sono stati eseguiti vari trial con inizializzazione random dei pesi delle unità, e con questa configurazione esempio si ottiene una accuracy del 100%.

Già dopo poche epoche sia l'errore di training che l'errore in validation si approssimano allo zero, si osserva come inizialmente i due si discostano a causa dell'iniziale assestamento della rete, che prosegue poi in una convergenza asintotica, fermata non appena l'errore di validazione scende sotto la soglia dello 0,001.

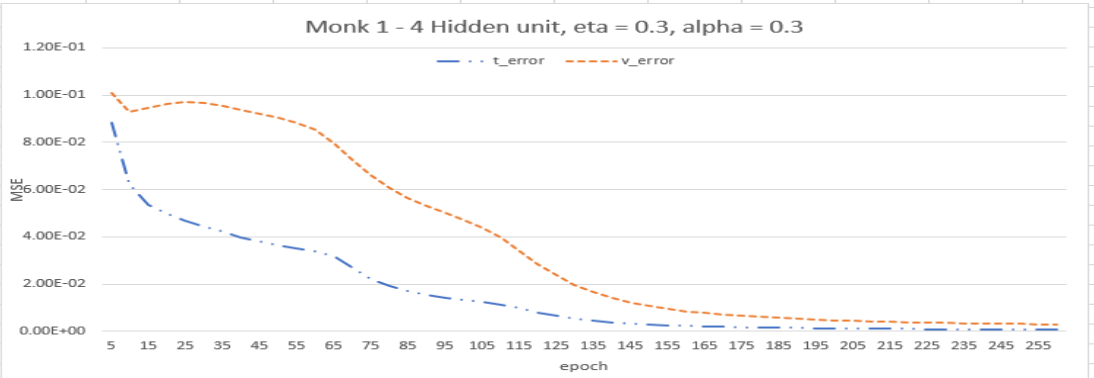


Figure 3-Curve di training error e validation error.

Stesso preprocessing per il file del test set applicato alla rete con un file secondario apposito per il test finale sul modello prodotto, da cui è stato generato un file con i risultati, e confrontandoli con i risultati di target sono state calcolate la matrice di confusione e le altre misure mostrate.

I risultati sul test set sono riportati su un file in formato csv, e con gli appositi nodi/funzionalità di knime sono stati analizzati i risultati tramite la matrice di confusione e la curva ROC

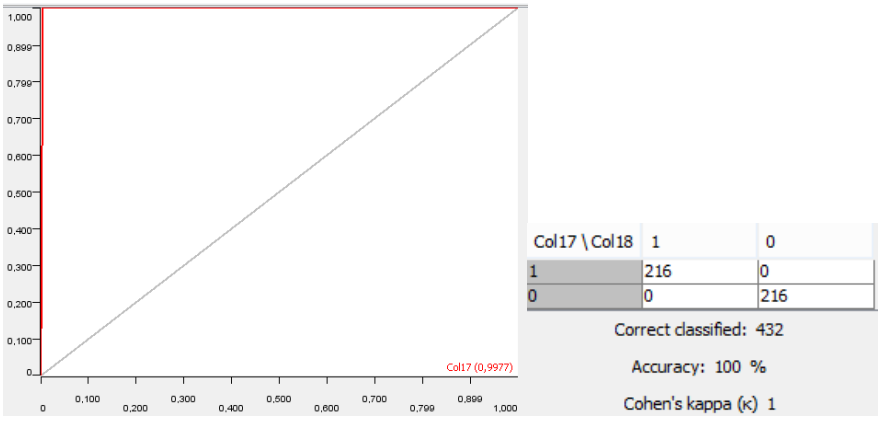


Figure 4-Curva di ROC e matrice di confusione con accuracy.

Monk 2

Il dataset Monk 2 è stato applicato a una rete con 4 unità nascoste, un learning rate di 0.3 e un coefficiente alfa per il momentum pari a 0.3. L'input è stato decodificato 1ofK con knime, producendo 18 features binarie di cui l'ultima come target. Tutte le unità usano la funzione di attivazione di tipo sigmoidale. Sono stati eseguiti vari trial con inizializzazione random dei pesi delle unità, e con questa configurazione esempio si ottiene una accuracy del 100%. Già dopo poche epoche sia l'errore di training che l'errore in validation si approssimano allo zero.

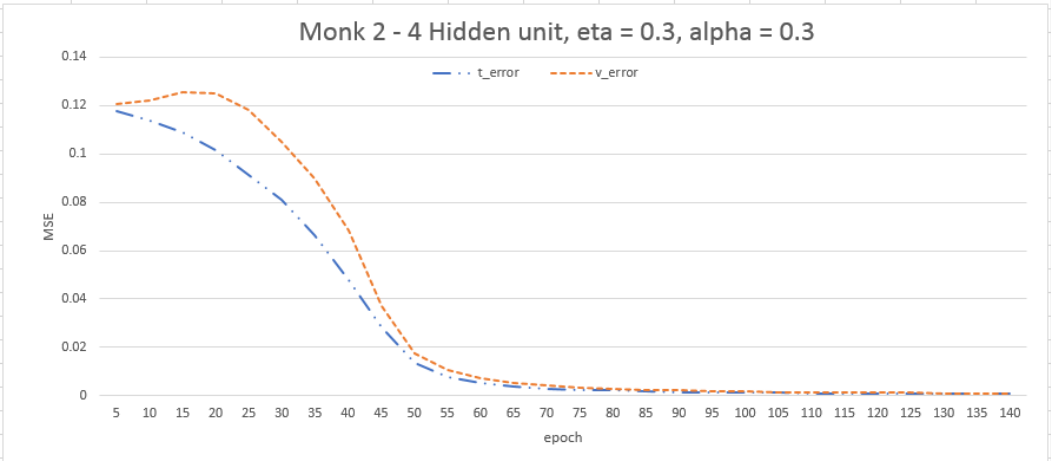


Figure 5-Curva di training con train error e validation error.

I risultati sul test set sono riportati su un file in formato csv, e con knime sono stati analizzati i risultati tramite la matrice di confusione e la curva ROC.

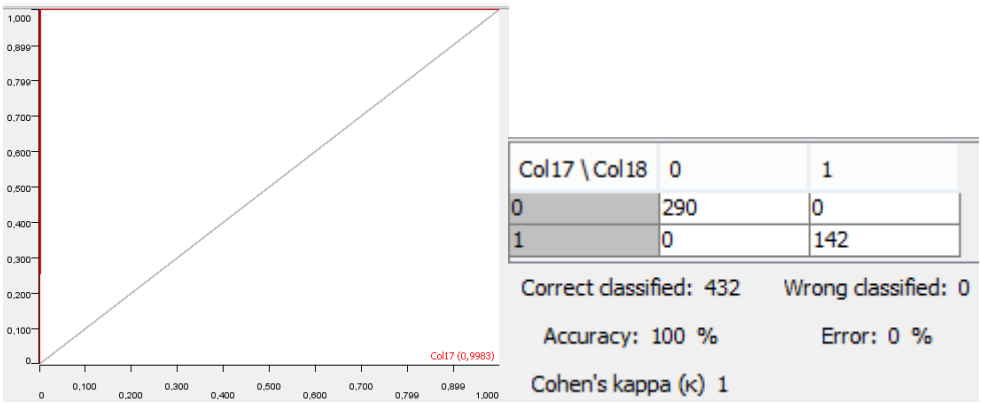
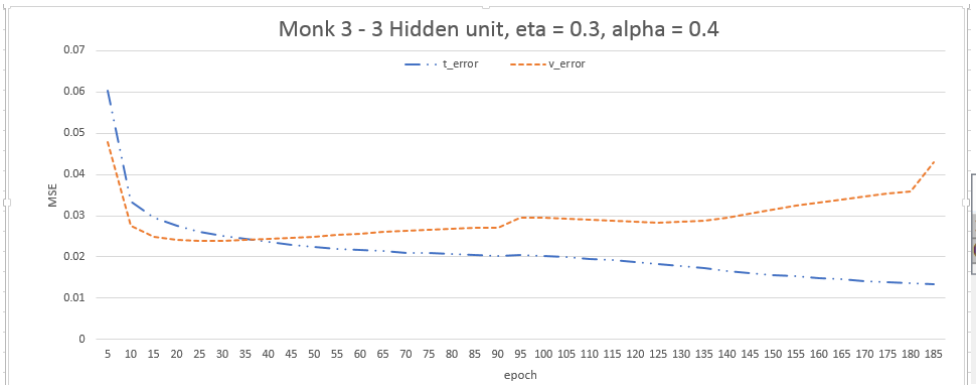


Figure 6-Curva ROC e matrice di confusione dei risultato.

Monk 3

Con 2 unità nascoste η,α costanti la rete ottiene 91,6% di accuratezza sul validation set, che non cambia aumentando il numero di unità mantenendo i parametri costanti. Meno rispetto a quelle usate nel paper di confronto (4 unità). Ho usato tre unità nascoste, eta 0.3 e alfa 0.4, con un criterio di arresto basato sull'errore in validazione, nel momento in cui ha percepito una crescita del 10% dell'errore è terminato l'algoritmo.

Il risultato su vari trial si mantiene intorno al 92% di accuracy, osservando la già nota presenza di rumore nei dati.



Col17 \ Col18	1	0
1	196	1
0	32	203

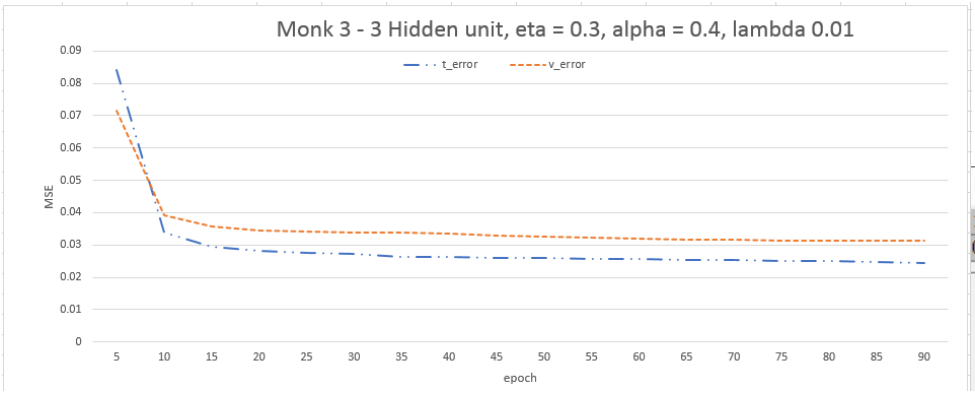
Correct classified: 399 Wrong classified: 33

Accuracy: 92,361 % Error: 7,639 %

Cohen's kappa (κ)

Figure 7- Grafico delle curve di training e matrice di confusione.

Provo ad applicare la regolarizzazione weight decay con lambda 0.01 e osservo su vari run un miglioramento in media nell'accuracy, intorno al 96-97%.



Col17 \ Col18	1	0
1	213	0
0	15	204

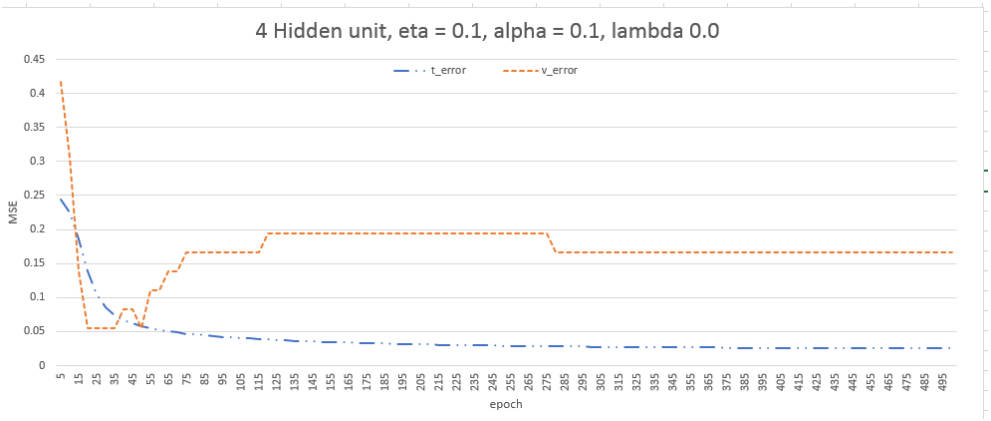
Correct classified: 417 Wrong classified: 15

Accuracy: 96,528 % Error: 3,472 %

Cohen's kappa (κ)

Figure 8- Grafico delle curve di training e matrice di confusione.

La regolarizzazione weight decay è stata applicata anche con i due problemi precedenti ed ha confermato la capacità di generalizzazione con un accuracy del 100%.



Stesso risultato (96,7 accuracy su test set) raggiunto con early stopping, e un altro modello con 4 unità nascoste parametri alfa = 0.1 eta=0.1. In 20 epoche è stato raggiunto il minimo errore sul validation set. Il v_error rappresenta l'error rate = 1-accuracy.

Figure 9- Grafico delle curve di training.

Si osserva per completezza che lanciando più volte l'algoritmo l'errore sul test set esterno ha comunque una certa variabilità, dovuto tra gli altri anche al punto di partenza, raggiunge anche un picco del 98%, così come durante i vari trial durante la model selection. Da osservare che il test set non è usato per prendere decisioni, ma solo come valutazione finale del modello scelto.

Regressione 1

Finora la rete è stata provata su problemi di classificazione, dato che il problema della competizione è un problema di regressione su due variabili target, per il collaudo finale uso un problema di regressione costruito artificialmente. Il preprocessing di questo problema consiste nella normalizzazione delle features di input, in modo da avere media 0 e la deviazione standard 1. Il dataset è stato creato appositamente applicando un rumore con media 0 e deviazione standard 0.2 ai target generati. Il dominio di x è stato generato con una distribuzione uniforme tra -3 e 3.

$$\langle x, x + 2 * \sin(1.5 * x) + N(0,0.2)\,, x + 2 * \cos(0.5 * x) + N(0,0.2)\rangle$$

Le due funzioni target sono mostrate in figura. Funzione con l'aggiunta di rumore come input di training, test set invece senza.

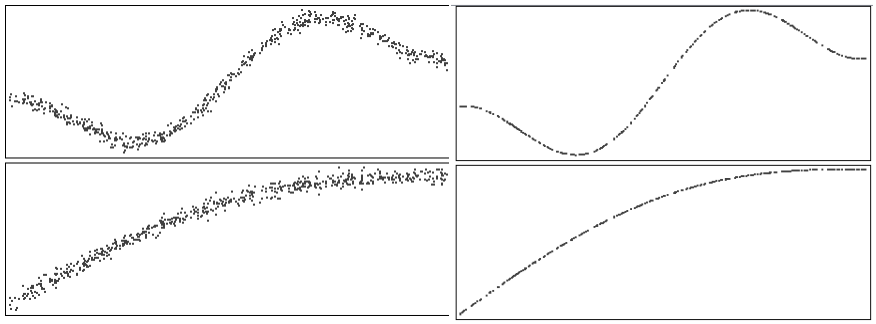


Figure 10-Grafico dei target di training e di test.

La rete in questo caso è configurata per avere 1 unità di input e 2 unità lineari di output, il numero di unità nascoste è scelto provando varie configurazioni e scegliendo quella con il minor numero di unità e il miglior errore di generalizzazione, misurato su un validation set.

Grafico del training con early stopping, che raggiunge rapidamente il valore minimo dell'errore nelle prime 150 epoche.

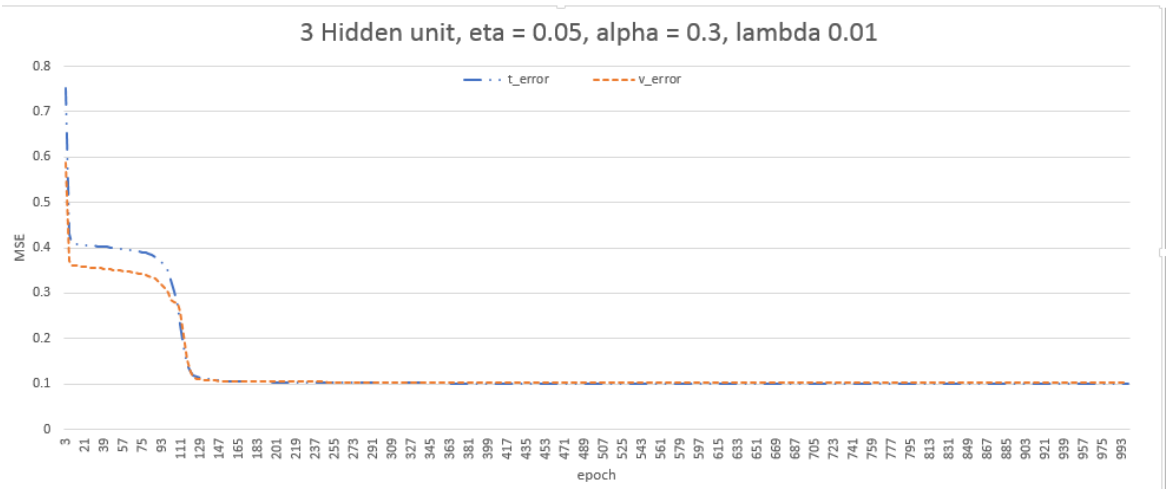


Figure 11- Grafico delle curve di training.

Loss (MSE) calcolata sul test set = 0.013

La rete con 3 unità nascoste è riuscita ad approssimare molto bene l'output cercato, con 2 invece si riscontravano errori maggiori e graficamente non si ruisciva ad ottenere la curva corretta.

La correlazione (Pearson's product-moment coefficient) tra il target y1 e quello calcolato è 0.999 così come si può riscontrare graficamente dal plot di y1 rispetto alla predizione y1. Risultati analoghi sono stati ottenuti per y2.

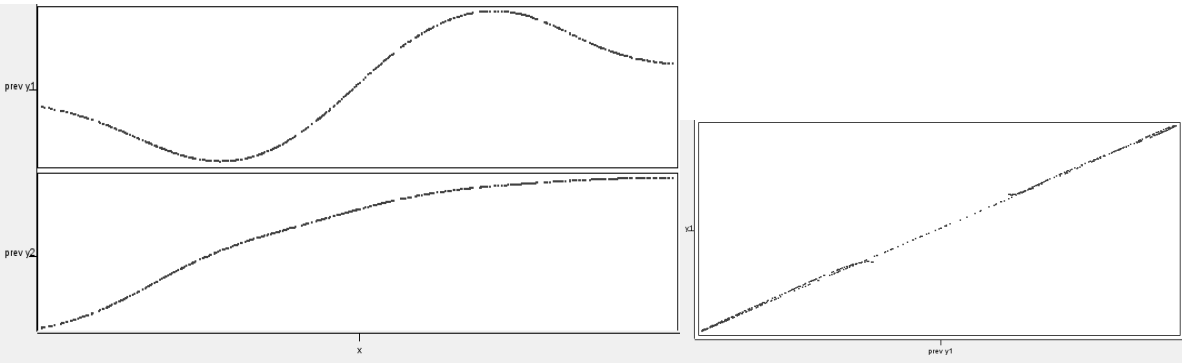


Figure 12-Grafico dell'output calcolato dalla rete e grafico si y1 e y1 calcolato a confronto.

Questo tipo di analisi sarà riproposta per il Cup task, in quanto fornisce un semplice strumento per verificare la capacità di corretta predizione del target.

Cup task

Analisi preliminari e preprocessing

Iniziamo con l'analisi delle statistiche sul dataset fornito. Non risultano valori mancanti (missing value). Gli input presentano una distribuzione sbilanciata, così come la variabile target y (Col6), mentre la variabile target x (Col5) ha una distribuzione uniforme su tutto il range di valori.

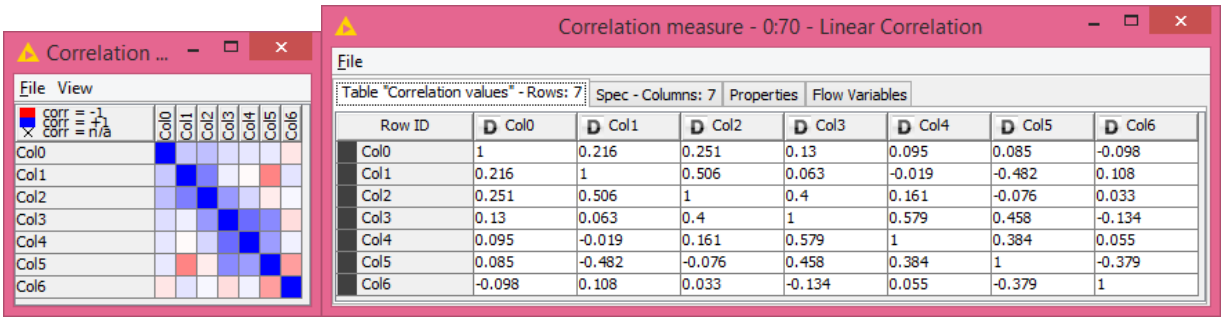
Row ID	Min	Max	Mean	Std. dev...	Variance	Histogram	Skewness	Kurtosis	Row count	No. missings
Col0	-7.567	5.627	0.001	1	0.999	-8 6	-1.556	19.305	990	0
Col1	-5.819	2.263	-0.021	0.997	0.994	-6 2	-1.469	5.557	990	0
Col2	-4.665	2.623	-0.014	0.991	0.983	-5 3	-1.275	3.881	990	0
Col3	-4.753	2.478	0.002	1.003	1.005	-5 2	-1.005	3.267	990	0
Col4	-4.997	2.484	0.035	1.021	1.043	-5 2	-1.558	4.741	990	0

Figure 13-Statistiche delle variabili di input.

Row ID	Min	Max	Mean	Std. dev...	Variance	Histogram	Skewness	Kurtosis	Row count	No. missings
Col5	36.86	58.643	47.048	6.415	41.148	37 59	0.042	-1.235	990	0
Col6	44.859	46.864	46.137	0.377	0.142	45 47	-0.722	0.069	990	0

Figure 14-Statistiche sulle variabilii target.

Dal Pearson's product-moment coefficient si vede che le due variabili target hanno una correlazione negative, e che alcuni degli input sono correlate l'un l'altro.



Row ID	Col0	Col1	Col2	Col3	Col4	Col5	Col6
Col0	1	0.216	0.251	0.13	0.095	0.085	-0.098
Col1	0.216	1	0.506	0.063	-0.019	-0.482	0.108
Col2	0.251	0.506	1	0.4	0.161	-0.076	0.033
Col3	0.13	0.063	0.4	1	0.579	0.458	-0.134
Col4	0.095	-0.019	0.161	0.579	1	0.384	0.055
Col5	0.085	-0.482	-0.076	0.458	0.384	1	-0.379
Col6	-0.098	0.108	0.033	-0.134	0.055	-0.379	1

Figure 15-Matrice di correlazione in forma visuale e tabellare.

Analizzando i box plot delle variabili (in scala originale), si osserva la presenza di molti outlier nelle variabili di input. Selezionandoli (evidenziati) possiamo notare che non corrispondono a valori particolari o estremi delle variabili target. Si può osservare selezionando un punto per volta che gli un outlier per una colonna corrisponde ad un outlier anche nelle altre, ma non nei target.

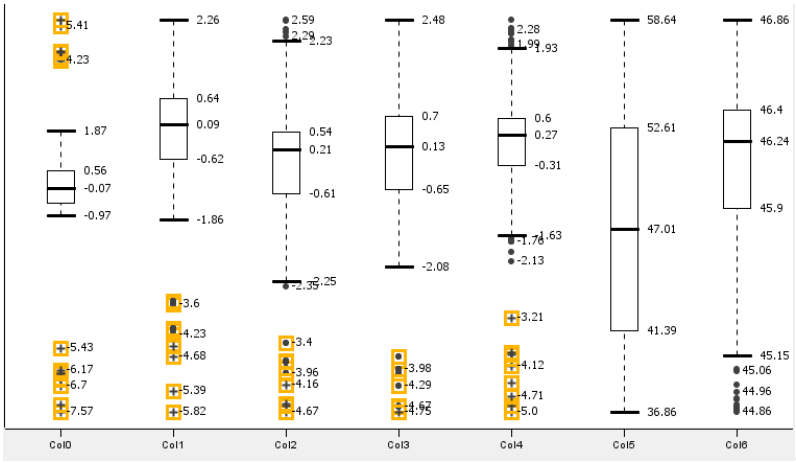


Figure 16-Bx plot delle variabili del dataset di input.

Continuando la nostra indagine per capire se si riferiscono a rilevazioni erronee sul plot dei target, osserviamo che i corrispettivi punti target e si concentrano interamente nell'estremità destra del grafico, si potrebbe dedurre che i punti estremi si concentrano in quella zona.

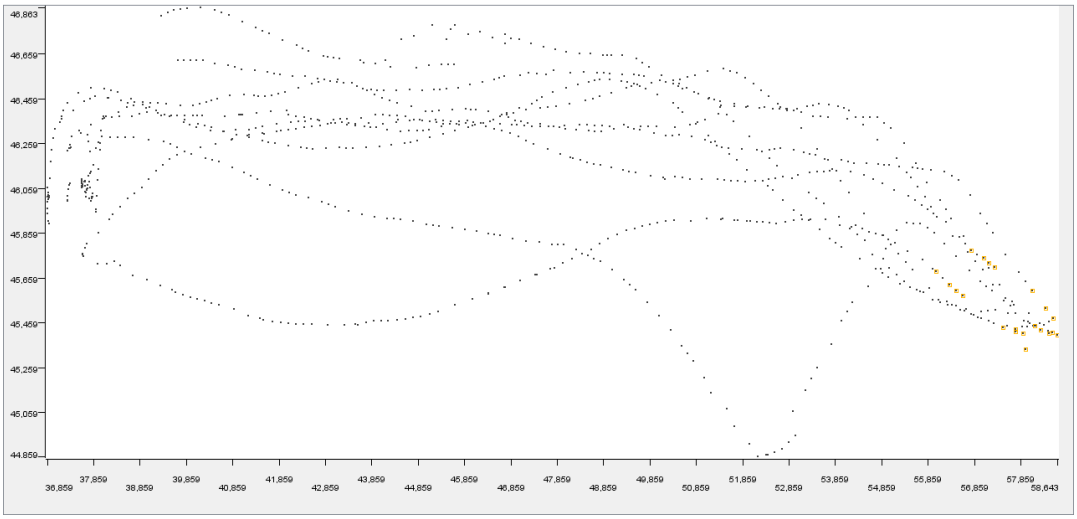


Figure 17_ Grafico dei target x e y. Evidenziati gli outlier trovati.

Eseguendo un filtro sui punti selezionati (16 punti) si ottiene una funzione obiettivo praticamente invariata.

Per vedere se questa operazione ha effetti positivi o meno sul learning, la prova è stata fatta apprendendo sul dataset filtrato e testando su quello originale. Il risultato ha confermato che non sapendo come comportarsi la rete "spara" i punti in posizioni anche molto lontane rispetto a dove dovrebbero essere, fatto osservabile anche da un errore, in questo caso, molto più alto. Gli effetti di questo preprocessing in questo caso sono quindi dannosi, restringendo il dominio di apprendimento non siamo più capaci di predire correttamente per i punti estremi in fase di test, questi punti contengono un'informazione utile ai fini della generalizzazione.

Dallo scatter plot delle variabili target x e y (figure 17) si osservano dei pattern nei dati, il primo obiettivo è quello di trovare un modello capace di rappresentare "fittare" questi pattern, o parte di loro in quanto è nota la presenza di rumore. Una prima prova è verificare allenando vari modelli a partire dai dati a disposizione se si riesce ad ottenere una buona approssimazione del training set. Se si riesce a trovare un modello capace di rappresentare con una buona approssimazione i dati, si può sperare di ottenere anche una buona generalizzazione.

Per questo fase mi avvalgo di due tool esterni offerti da knime che implementano un MLP con Rprop e MLP con backprop, e confronto i risultati con quelli ottenuti dalla mia rete.

Per la stabilità numerica della rete sviluppata e per l'uso dei tool una è necessario effettuare una normalizzazione tra [0,1], le prove con la z-score causano overflow numerici sulla mia rete.

I due tool sono stati eseguiti con varie configurazioni, principalmente è stato analizzato il comportamento al variare del numero di unità nascoste e tempo massimo di apprendimento, che per quanto l'algoritmo Rprop sono i due parametri principali, mentre per il backprop dopo varie prove sono stati usati alfa ed eta 0.3 e 0.1 che sono risultati una configurazione con cui poter provare varie dimensioni del livello nascosto, al fine di cercare un numero di unità per la generalizzazione risultasse migliore. Nelle prove il range di ricerca per il numero di iterazioni è arrivato fino a 50,000 epoche e per le unità nascoste fino a 100 nel caso rprop e 200 nel caso backprop.

Il risultato di ogni prova è stato analizzato

- Graficamente: confrontando i grafici ottenuti con i due target predetti e con uno solo per volta.
- Analiticamente: verificando la correlazione tra il target predetto e quello dato; con la misura dell'errore commesso.

Scelta delle unità nascoste fissando alfa e eta, osservando il variare dell'errore al variare del numero di unità, e si prende quello che da un errore minore. Deciso il numero di unità nascoste con il procedimento a griglia cerco i valori migliori per alfa eta e gamma. Il modello selezionato lo alleno su tutti i dati e lo applico al blind test set.

Risultati della rete implementata e dei tool

Iniziamo con un confronto dell'effetto dei parametri di learning sulla curva dell'errore su un training di 300 epoche, i dati sono stati normalizzati, i valori sono solo indicativi relativamente al confronto. Il numero di epoche piccolo è stato scelto solo per avere una visione qualitativa del comportamento dell'algoritmo.

Da questi confronti si osserva che al diminuire del parametro eta corrisponde una variazione della curva dell'errore più smussata e più lenta, mentre al crescere la variazione diventa più netta e discontinua, questo in accordo alla teoria, in quanto viene data più o meno importanza durante l'apprendimento al nuovo pattern visto rispetto ai precedenti. Come poi si verificherà con la model selection i parametri di alfa ed eta che ottengono le migliori prestazioni su questo problema sono alfa di circa 0.5 ed eta con valori tra 0.01 e 0.1.

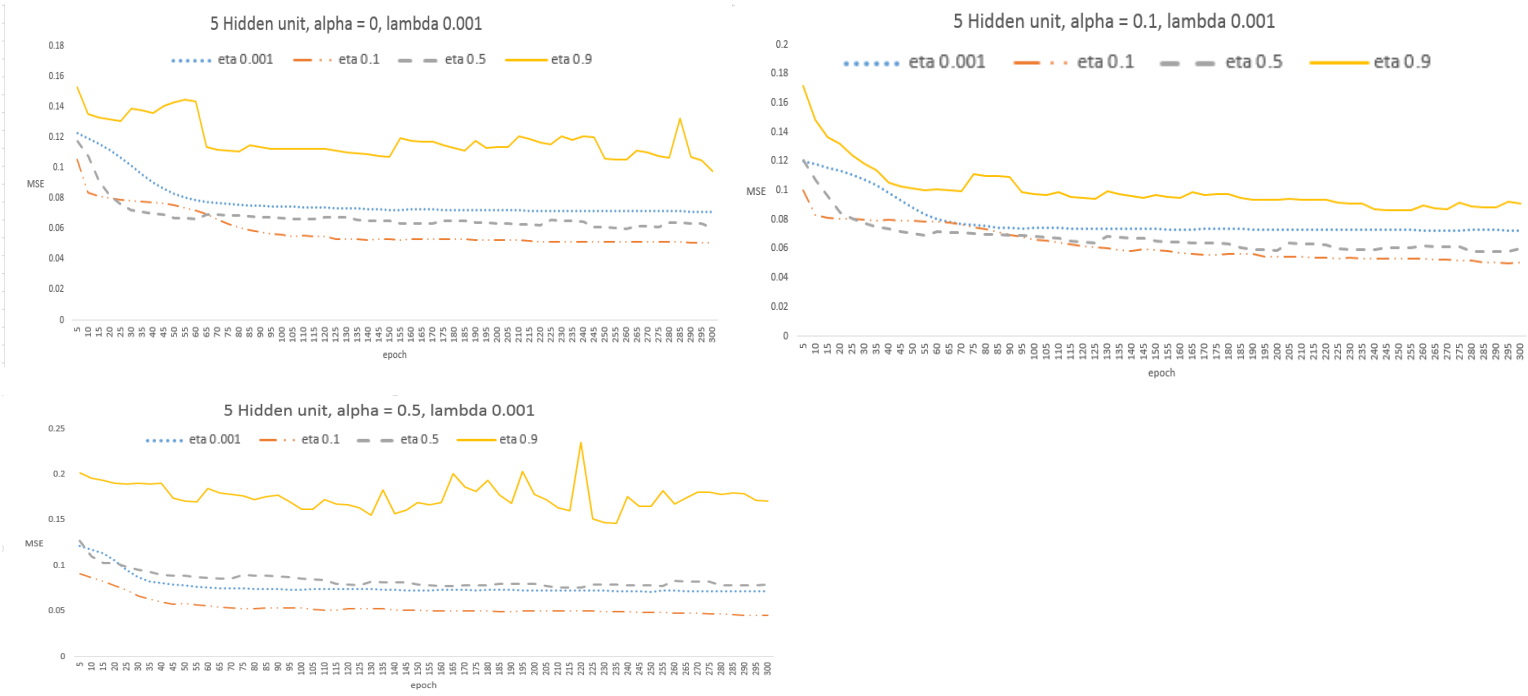


Figure 18_Grafici durante il training del validation error al variare degli iperparametri alfa ed eta.

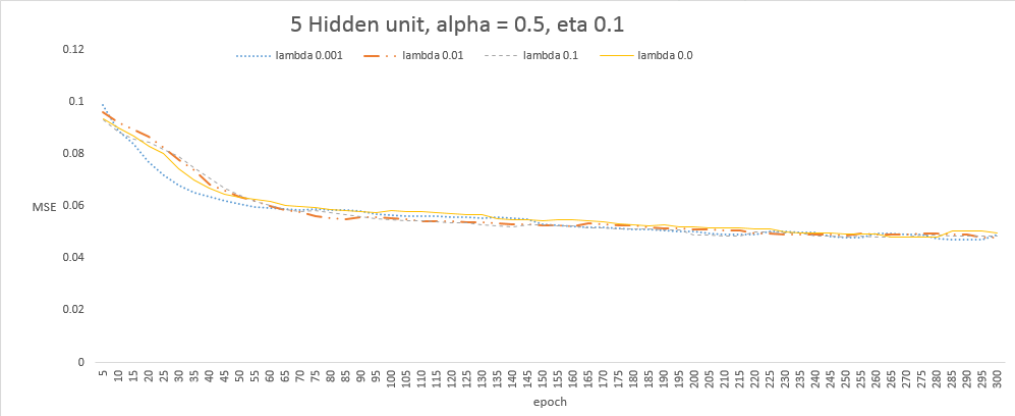


Figure 19_Grafico al variare di lambda.

Questo grafico invece confronta come varia in base al parametro di regolarizzazione lambda. Osservando che un lambda piccolo apporta la maggiore differenza rendendo la curva più regolare e smussata, e soprattutto sembrerebbe aumentare la velocità con cui raggiunge il "minimo".

Il miglior risultato è stato ottenuto con il MLP Rprop (knime), con 25 unità nascoste e 50,000 non è stato molto incoraggiante. Plottando gli assi target, e gli assi appresi, usando come test set il training set, osserviamo che la rete non riesce a rappresentare i dati di training. La difficoltà maggiore sta nella variabile target y (col 6). Dal confronto grafico tra y target e y calcolato si vede che la precisione è bassa, non si ha una diagonale precisa ma sparsa, l'indice di correlazione tra le varie configurazioni ha raggiunto con questa la punta di 0.80. Usando la x target e la y calcolata si raggiunge ugualmente una approssimazione nella forma complessiva, ma senza troppi dettagli.

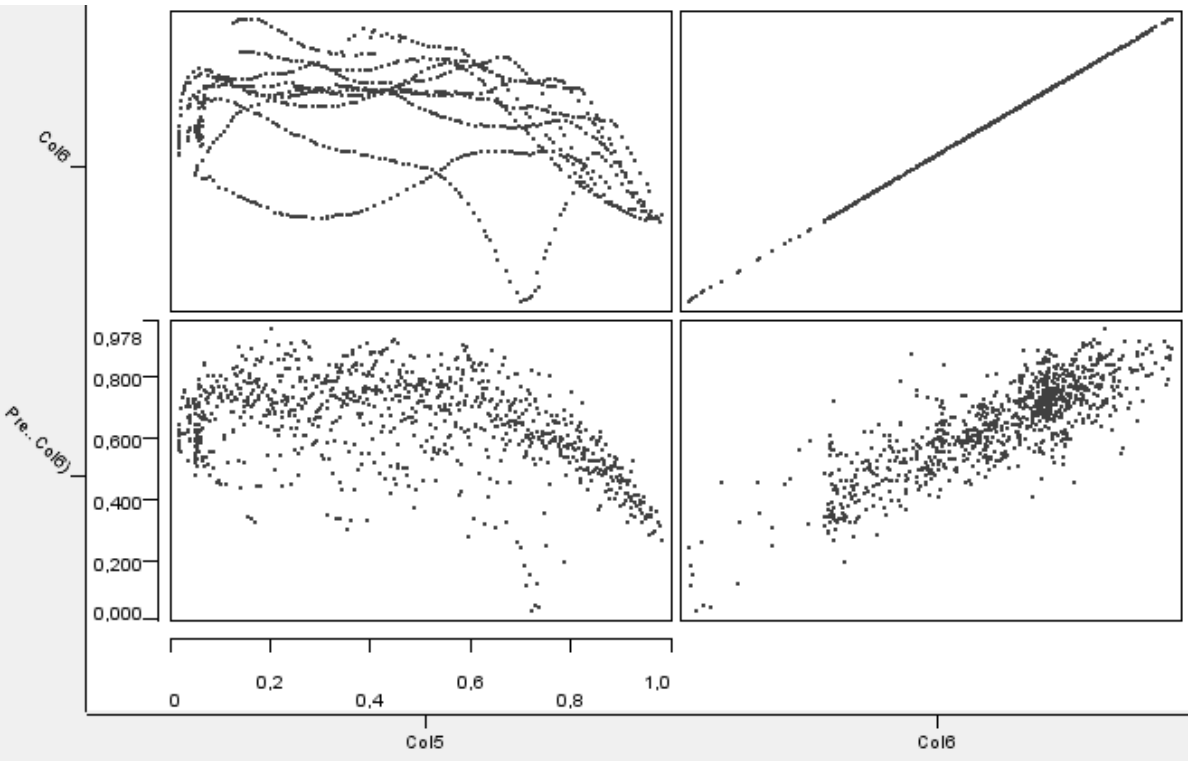


Figure 20_Grafico a confronto tra i target e le variabili calcolate, col5=X e col6 =Y.

Sulla variabile x (col5) i risultati sono migliori, l'indice di correlazione è 0.98, visibile dalla diagonale più compatta nel grafico. Un altro riscontro di una migliore approssimazione della x si ha plottando la x predetta con la y target, e ovviamente dalle misure dell'errore relativo a x.

Table "Correlation values" - Rows: 3 | Spec - Columns: 3

Row ID	Col5	Col6	Prediction (Col5)
Col5	1	-0.339	0.98
Col6	-0.339	1	-0.348
Prediction (Col5)	0.98	-0.348	1

R^2:	0,961
Mean absolute error:	0,042
Mean squared error:	0,003
Root mean squared error:	0,058
Mean signed difference:	-0

Figure 21_Tabella di correlazione ed errori calcolati sulla sola approssimazione di X.

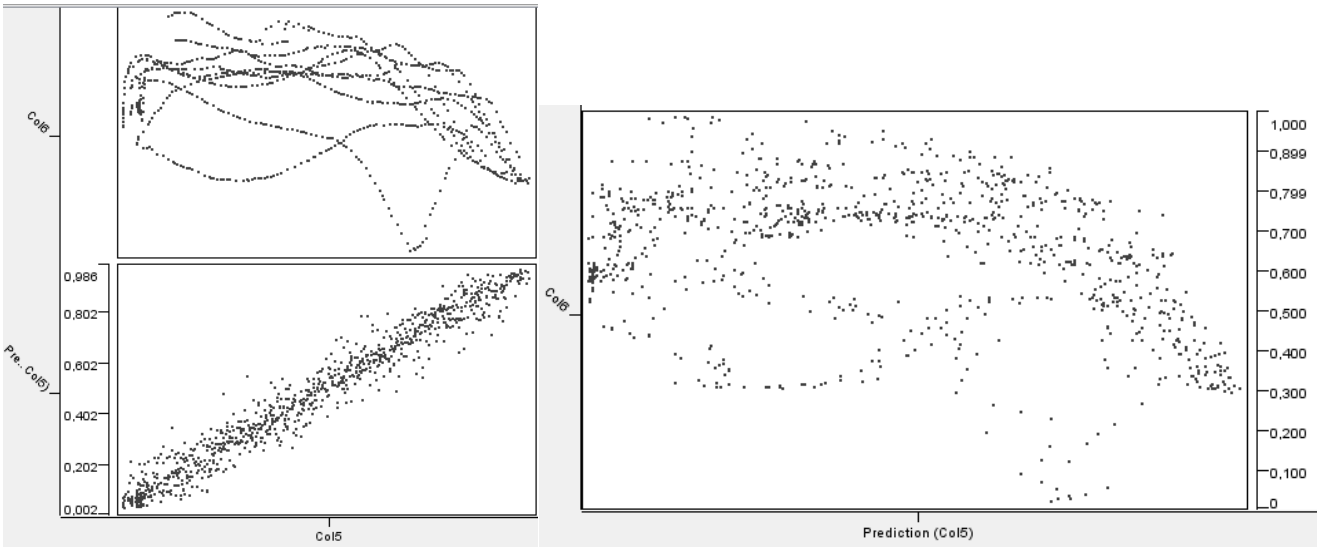


Figure 22_Grafici relativi al confronto tra X target (col5) e calcolata.

Senza degradare troppo l'accuratezza del risultato, Rprop su y con 14 unità nascoste e 50,000 iterazioni ottiene dei risultati molto simili, usando un modello molto più semplice.

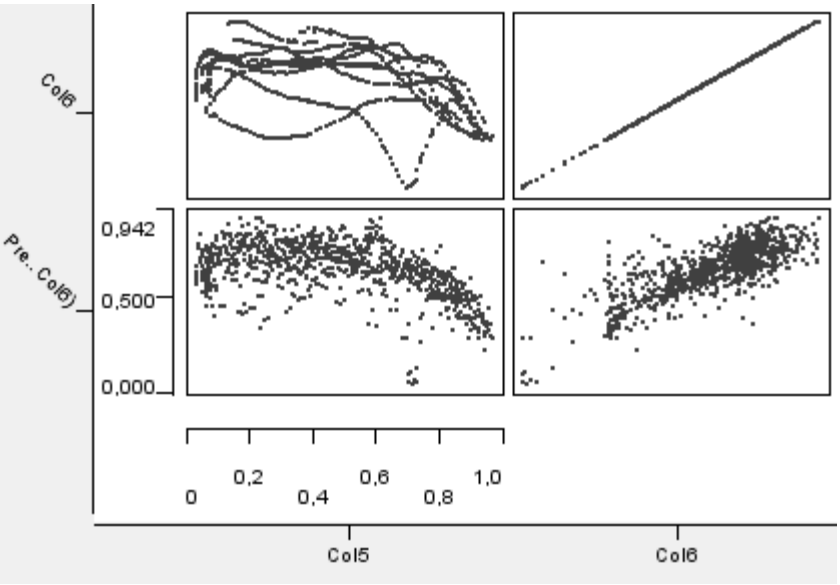


Figure 23_Grafici di confronto tra Y (col6) e Y calcolata.

Con questi risultati, non mi aspetto di ottenere un errore molto basso su questo dataset. Ma sono riuscito ad identificare un range di valori su cui effettuare una ricerca per selezionare il modello finale.

Dalle prove effettuate con la mia implementazione, seguendo la procedura di selezione del modello precedentemente descritta, con 25 unità ottengo risultati simili ma peggiori rispetto al tool prima analizzato. Si conferma la difficoltà nell'approssimare la variabile Y, la cui correlazione come si vede dai grafici di confronto è bassa (0.5). Meglio su X, mantenendo la y originale riesce a intravedersi un approssimazione della figura originale.

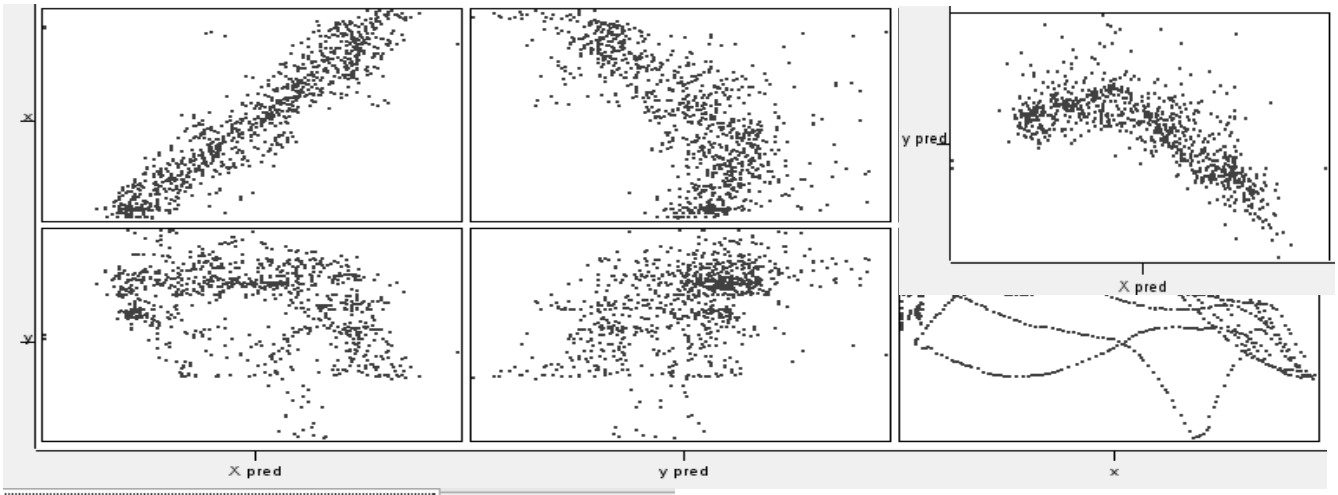


Table "Correlation values" - Rows: 4 | Spec - Columns: 4

Row ID	X pred	y pred	x	y
X pred	1	-0.567	0.954	-0.301
y pred	-0.567	1	-0.555	0.504
x	0.954	-0.555	1	-0.339
y	-0.301	0.504	-0.339	1

Figure 24_Grafico e matrice di correlazione di confronto tra variabili target e quelle calcolate dalla mia rete con 25 unità nascoste.

Mettendo insieme x e y predetti dal modello si ottiene una rappresentazione semplificata della funzione appresa, evidente la perdita delle forme inferiori della figura.

Confrontando i risultati appena ottenuti con un modello con 25 unità nascoste, dove la model selection non è scesa sotto un errore (mse) di 0.158, 0.161 sui dati normalizzati, posso constatare che le performance non sono degradate significativamente. Con parametri Eta 0.01 alfa 0.5 lambda 0.01 la curva di apprendimento è abbastanza regolare senza comportamenti particolari, mostra una convergenza asintotica, che però non scende sotto una certa soglia.

Figure 25_Curva di learning con mse sul training e validation set.

Test sul training set (mse) 0.15. Continua la scarsa predizione sull'asse y, mentre la predizione sull'asse x va leggermente meglio. Incremento che però non vale l'aumentata complessità del modello, la quale rischia di peggiorarne la capacità di generalizzazione. Dalla stima con hold out infatti non si ha un miglioramento apprezzabile.

Ulteriori prove e una ricerca sempre basata su holdout ha mostrato la possibilità di ridurre ulteriormente il numero di unità fino a 5 unità. Da considerare che una stima dell'errore più accurata si otterrebbe con una cross validation, ma a causa dei costi computazionali non è stata fatta.

Unità nascoste	Validation MSE media su 3 trial	Media epoche Emin
5	5.85	15k
10	7.42	8k
20	8	12k
30	7.6	16k
40	6.7	7k

Figure 26_Tabella prove con diverse unità nascoste.

Rieseguita la ricerca con una grana più fine tra 5 unità e 10 unità, ottenendo 5 unità con un errore sul training set di 27.

A questo punto si fa il tuning dei parametri eta, alfa e lambda, come descritto con una ricerca a griglia, ottenendo: eta = 0.1, alfa = 0.1, lambda = 0.001; I tre trial relativi ai parametri migliori hanno ottenuto sui dati normalizzati una loss rispettivamente di 5.4 in 10k epoche, 5.4 in 30k epoche, 7 in 20k epoche, con una media di 5.5 sul validation set. Mentre il grafico mostra l'allenamento del modello finale usato per produrre l'output per la competition, con 11,5k epoche e un minimo raggiunto dopo 6,5k epoche.

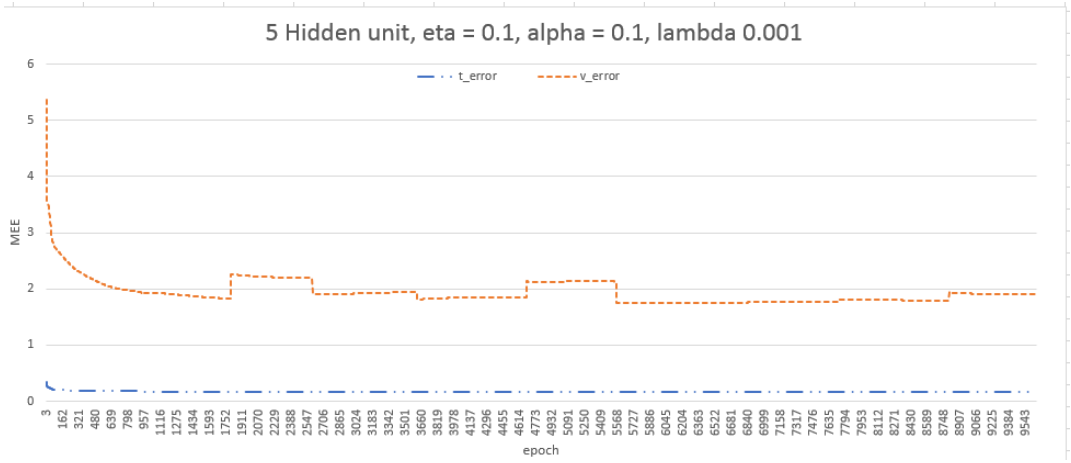


Figure 27_Andamento del training del modello finale usato per il test.

L'errore si stabilizza a un certo valore senza risalire grazie alla normalizzazione dovuta al weight decay.

La stima dell'errore di predizione sul modello finale è fatta tramite cross validazione con 5 fold, con risultato una loss mee media di 1.9.

Conclusioni

In conclusione mostro il grafico relativo alla predizione sul test set.

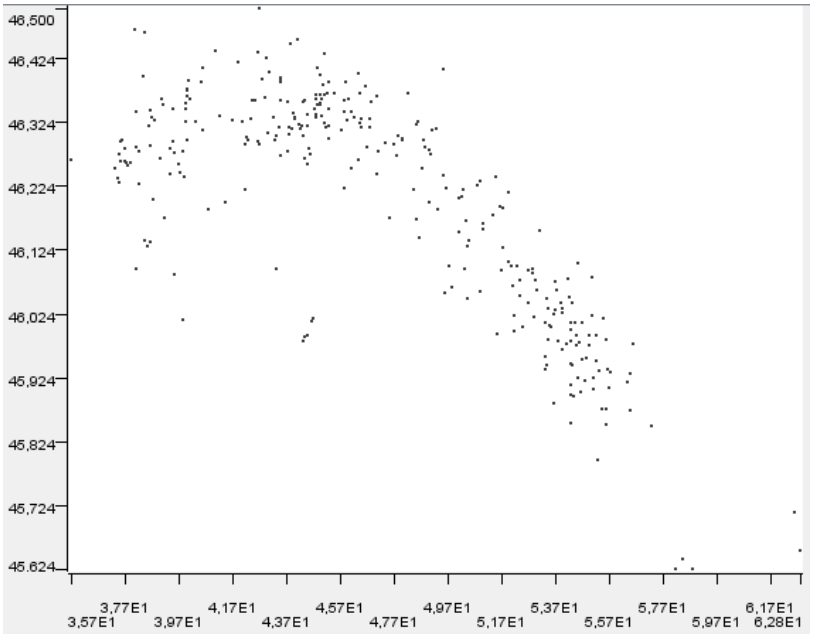


Figure 28_Grafico della predizione sul test set.

Per il progetto ho scelto il linguaggio C++ sia per ottenere delle buone performance computazionali sia per imparare un nuovo linguaggio.

La rete implementata non ha raggiunto una precisione di predizione molto elevata, confrontata con gli altri modelli, per risolvere il problema con una maggiore precisione bisogna proseguire l'indagine su altri modelli, o con una maggiore potenza di calcolo provare a cercare un modello migliore con una ricerca più ampia.

In allegato FBAA1_abstract.txt, FBAA1_LOC-UNIPI-TS.csv e la cartella FBAA1_Project2013.rar con i sorgenti.

Autorizzo la pubblicazione dei risultati.

Bibliografia & Link

[1] Haykin
[2] Mitchell
[3] Early Stopping | but when? Lutz Prechelt (http://page.mi.fu-berlin.de/prechelt/Biblio/stop_tricks1997.pdf)
[4] Metodi di ottimizzazione per le reti neurali L. Grippo (http://www.disp.uniroma2.it/users/piccialli/retineurali.pdf)
[5] ftp://ftp.sas.com/pub/neural/FAQ.html
[6] https://archive.ics.uci.edu/ml/datasets/MONK's+Problems
[7] http://scott.fortmann-roe.com/docs/MeasuringError.html#fnref:1