Progetto Machine Learning

Danilo Sorano , Valerio Bonsignori
soranod@gmail.com, valerio12bonsignori@gmail.com

ML, AA 2017/2018
Date: *04/02/2018*
Progetto Tipo: **A**

ABSTRACT

Abbiamo voluto implementare una rete neurale di tipo FeedFoward da zero per comprendere maggiormente quali siano le complicazioni che sorgono nell'implementazione delle strutture della rete e gli algoritmi di apprendimento; per capire approfonditamente quali siano i punti cruciali per la costruzione e l'utilizzo di percettroni a più livelli.

1. INTRODUZIONE

Il nostro obbiettivo è stato quello di esplorare e comprendere bene, non solo la struttura di una rete neurale FeedFoward, ma il modo in cui questa riesce ad apprendere. Quindi in cosa consista in termini di statement un algoritmo di apprendimento e una *predizione* di una rete neurale.
Il seguente progetto si pone l'obiettivo di realizzare una rete neurale che approssimi funzioni per la classificazione e per la regressione. I dataset utilizzati per testare le capacità della rete sono i dataset "Monk" e "Cup". Prima di descrivere nei dettagli le metodologie adottate per l'implementazione è bene descrivere in maniera generale il tipo di rete realizzata e i dataset utilizzati per la classificazione e la regressione.
La nostra rete neurale è un grafo orientato aciclico. I livelli del grafo sono definiti come "Layer", con delle unità al suo interno. I Layer sono collegati tra di loro per mezzo di archi a cui è associato un peso. Affinché la rete neurale possa essere utile per approssimare con un errore *accettabile* la funzione, deve essere prima addestrata adeguatamente. La fase di Training avviene per mezzo dell'algoritmo di Backpropagation [1] basato sull'errore residuo, tramite il quale si calcola il gradiente necessario all'aggiornamento dei pesi della rete, atto a minimizzare l'errore che la rete produce.

Abbiamo utilizzato la rete per i seguenti compiti:

- Classificazione
- Regressione

La fase di regressione si occupa invece dell'analisi del dataset "Cup", consistente in un training set ed un set senza target, chiamato "Blind test set".

I dataset hanno 10 features reali per gli input e 2 attributi reali target.

Per ottenere una buona approssimazione della funzione target è necessario addestrare le reti.
Per ottenere i migliori iperparametri per l'addestramento è necessario effettuare una grid search.
Per verificare i rischi e le capacità di approssimazione valutiamo la funzione approssimante sul test set.

2. METODI

La rete neurale da sviluppare, si occuperà di approssimare funzioni di classificazione e di regressione. Qui di seguito verranno descritte le metodologie adottate per poterla implementare nel linguaggio Python.
Una descrizione dettagliata dell'implementazione richiede come prima cosa un introduzione sulle librerie utilizzate per lo sviluppo. La necessità di dover lavorare con delle matrici, rappresentazioni di grafi densi, ci è stata facilitata tramite l'utilizzo della libreria "Numpy" e dei suoi array ottimizzati. Per le scelte casuali da intraprendere ci siamo affidati alla la libreria "Random". Ci siamo serviti della libreria "Matplotlib" per la visualizzazione dei grafici relativi all'errore sul training sul test. Per salvare e caricare le istanze di reti abbiamo usato "Pickle".
L'implementazione della rete parte dalla costruzione dei singoli layer e successiva aggregazione di quest'ultimi in una rete. La creazione dei layer avviene tramite due classi:

- Layer
- Output_Layer

La prima si occupa della creazione dei layer nascosti con il calcolo dei relativi gradienti per poter poi aggiornare i pesi. La classe "Output_Layer" eredita da Layer e permette il calcolo diretto dell'influenza dell'output del neurone sul risultato finale (i delta). Entrambi permettono di definire quale funzione di attivazione utilizzare. Nei layer vengono calcolati tutti le componenti necessarie per implementare la backpropagation.
La classe Net crea la rete prendendo in input le istanze layer create e permette tramite il metodo "fit" di eseguire i metodi per calcolare il gradiente ed aggiornare la rete secondo le modalità 'batch', 'mini-batch' e 'online'.
Nonostante la grande efficienza delle librerie, non abbiamo voluto venir meno alla leggibilità del codice con conseguente maggiore chiarezza per le prime fasi di debug.
Nell'implementazione abbiamo preferito la leggibilità e la somiglianza sintattica alle formule piuttosto che all'efficienza, questo ci ha permesso di avere maggiore padronanza sul ruolo degli iperparametri. In appendice, figura **A1**, ci sono i plot di tre buoni risultati ottenuti nella task Monks-3 che ci ha aiutati a comprendere gli effetti della regolarizzazione.
Le reti utilizzate per il Monks e per la CUP sono differenti tra di loro. Le caratteristiche che le caratterizzano sono state determinate tramite delle prime prove iniziali per definire il criterio di stop, il numero di layer interni e le funzioni di attivazione da utilizzare. Le prove effettuate ci hanno permesso di definire i seguenti aspetti:

- **MONKS:**
  - Criterio di Stop → 500 epoche per la batch mode e 200 per online

Nella modalità batch abbiamo voluto essere sicuri di sfruttare fino in fondo la plasticità della rete, per cui, nel caso del batch abbiamo scelto di arrivare fino a 500 epoche.

- Funzione di attivazione layer interni e di output → Sigmoide

  La funzione sigmoidale ci è sembrata la più consona per un problema risolvibile con una funzione approssimabile che impone un output binario. Inoltre al layer di output è stato aggiunta una treshold (0,1) per poter affrontare il problema della classificazione.
- Numero di layer interni → 1

  L'implementazione della rete ci ha permesso di provare facilmente più di un hidden layer, ma il numero di parametri addestrabili cresceva così tanto che il gradiente andava diminuendo sempre più, anche a causa della scelta della funzione di attivazione, e l'accuratezza della funzione ipotizzata dalla rete non riusciva ad aumentare. Ci è sembrato comunque eccessivo utilizzare più di un hidden layer.

- **CUP**:
  - Criterio di Stop → 300 epoche

    Questo perché abbiamo un numero di pattern maggiori per cui sono necessarie meno epoche per convergere.
  - Funzione di attivazione layer interni → Sigmoide
  - Funzione di attivazione layer di output → Identità
  - Numero di layer interni → 1

Le altre caratteristiche fondamentali sono state determinate tramite una "Grid Search" necessaria per determinare i milgiori iperparametri.
Il numero di epoche è elevato in quanto vengono eseguite più modalità("Online", "Batch", "Mini-batch") per comprendere quale sia la più idonea.
Un altro importante aspetto riguarda la fase di "preprocessing". Per quanto riguarda i tre Monks abbiamo subito voluto utilizzare la codifica 1-of-k per gli attributi in input, la natura del problema (classificazione) e alcune prove preliminari ci hanno spinti a scegliere questa codifica.
I modelli analizzati sono stati validati tramite il metodo "Hold-out" che prevede l'esclusione di parte del development set dal training per ottenere un validation set.

3. ESPERIMENTI

La ricerca del modello migliore, sia per il "monks" che per la "cup", ha richiesto una serie di esperimenti per trovare i migliori iperparametri, modalità di esecuzione ("batch", "online", "mini-batch") e numero di unità del layer interno. Gli iperparametri su cui ci si è soffermati sono: learning rate (eta), momentum (alfa) e batch-size (solo per il mini-batch). Inoltre per il Monks-3 e per la CUP si è analizzato anche l'influenza di un eventuale regolarizzazione (lambda). L'analisi di questi fattori è stata fatta in due fasi: una manuale, nella quale si sono provati alcuni parametri (quali funzione di attivazione, numero di unità, numero di layer) per vedere il comportamento della rete, ed un'altra automatica dove sono state effettuate delle Grid Search basate sull'analisi manuale effettuata in precedenza.

3.1 RISULTATI DEI MONKS

Dopo una prima ricerca Grid Search abbiamo selezionato alcuni tra i modelli migliori, per poi raffinare i range di ricerca tra i migliori valori degli iperparametri, tramite una *fine*-Grid Search.

Nella **Tabella 1** sono presentati i risultati medi ottenuti dei modelli migliori.

I raffinamenti nella ricerca degli iperparametri migliori sono stati testati su 20 (Monks-1) e 10 (Monks-2,3) reti con identica topologia ma configurazione dei pesi iniziali differente.

Task	**#Units, eta, lambda, mode**	**MSE (TR/TS)**	**Accuracy (TR/TS) (%)**
MONK 1	**Unità**: 5, **eta**: 0.35, **alfa**: 0.3, **modalità**: "online"	TR: 0.0019 TS: 0.0031	TR: 99.68% TS: 99.52%
MONK 2	**Unità**: 5, **eta**: 0.3, **alfa**: 0.8, **modalità**: "online"	TR: 5.2466e-05 TS: 6.3883e-05	TR: 100% TS: 100%
MONK 3	**Unità**: 7, **eta**: 0.3, **alfa**:0.3, **batch-size**:50, **modalità**: "mini-batch"	TR: 0.0286 TS: 0.0226	TR: 93.52% TS: 97.12%
MONK 3 con regolarizzazione	**Unità**: 10, **eta**: 0.5, **lambda**:0.01, **alfa**:0.75, **modalità**:"mini-batch", **batch-size**: "10"	TR: 0.0392 TS: 0.0295	TR: 93.44% TS: 97.22%

Tabella 1. Risultati sulla predizione media nei vari Monks

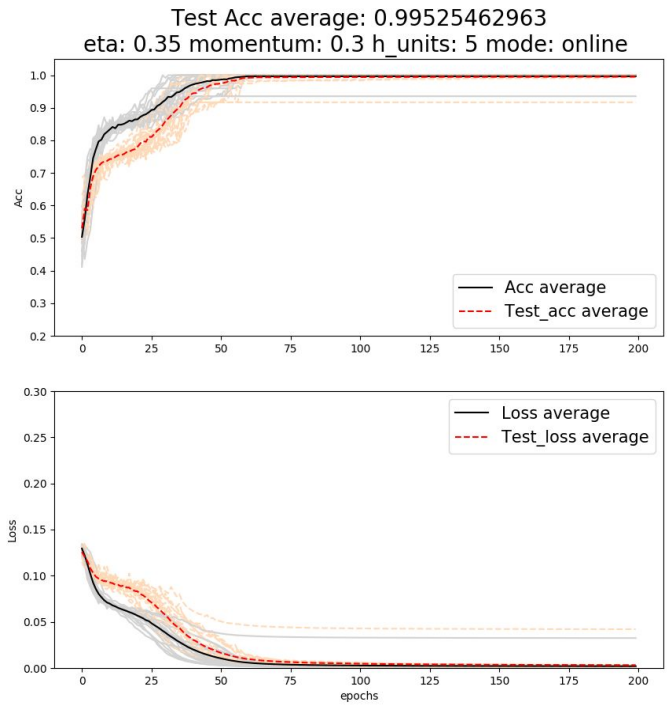

Figura 1. Accuracy e Loss(MSE) medi sul TR e sul TS del modello migliore sul Monks-1

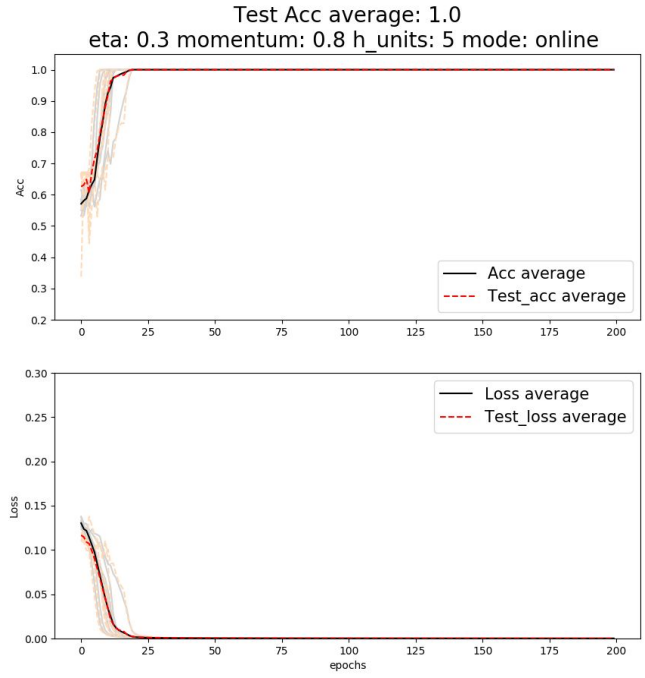

Figura 2. Accuracy e Loss(MSE) medi sul TR e sul TS del modello migliore sul Monks 2

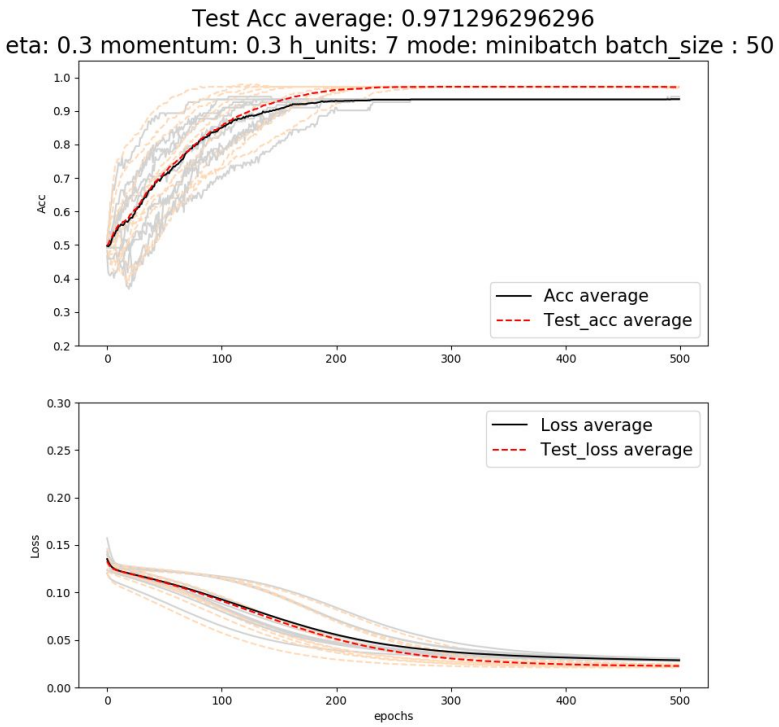

Figura 3. Accuracy e Loss(MSE) medi sul TR e sul TS del modello migliore sul Monks 3 senza regolarizzazione. Il grafico mostra l'instabilità del modello senza regolarizzazione, il quale riesce ad ottenere un risultato stabile solo intorno alle 400 epoche

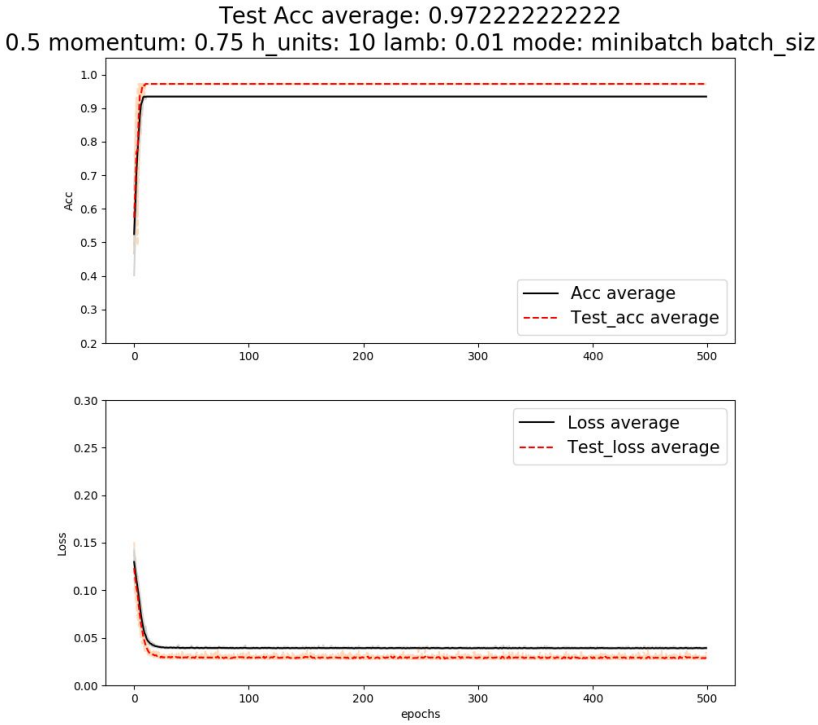

Figura 4. Accuracy e Loss(MSE) medi sul TR e sul TS del modello migliore sul Monks 3 con regolarizzazione. Osservando i risultati si può notare che l'accuratezza della rete neurale ottenuta tramite l'addestramento sul dataset del Monks-3 con regolarizzazione (*weight-decay*) è leggermente più alta rispetto ai migliori risultati ottenuti senza regolarizzazione, ottenendo risultati stabili molto rapidamente.

Per il Monks 1 è presente un altro modello(**Figura A.2**) interessante riportato nell'appendice, questo però possiede elevati parametri (eta = 0.95), per cui si è preferito selezionare quello con eta più basso. Abbiamo voluto escluderlo poiché non l'abbiamo ritenuto molto affidabile per la natura dell'eta molto alto, preferendo un modello che convergesse più lentamente ma con un *learning rate* più basso, quindi più affidabile.

3.2 CUP RESULTS

In questa fase l'implementazione della rete testata sui 3 monk è stata utilizzata per poter approssimare una funzione di regressione sul dataset *Cup*. La prima parte di questa esperienza si è focalizzata nel trovare una topologia che si adattasse plasticamente nell'addestramento della stessa tramite il *training-set* fornito.

Il dataset Cup è stato suddiviso in tre dataset più piccoli Training(60%), Validation(20%) e Test(20%). La divisione del dataset in training e validation è fondamentale per la fase di *Model Selection* durante la quale viene selezionato il modello migliore. La selezione del modello è avvenuta tramite l'applicazione del metodo Hold-Out che utilizza il training per addestrare la rete e il validation per validare le capacità predittive.

Per ottenere un modello che ci permettesse di ottenere buoni risultati sulla task *Cup* abbiamo seguito i seguenti step:

- abbiamo diviso il Dataset *Cup* in Development set e Test Set.
- abbiamo fatto una ricerca manuale delle topologie (numero di nodi nel layer interno) e delle funzioni di attivazione da utilizzare.
- ricerca automatica dei migliori iperparametri da utilizzare tramite Grid Search con il training set e il validation set ottenuti dal Development. (Model Selection).
- scelti i migliori iperparametri abbiamo eseguito un'ulteriore grid Search per approfondire i range degli stessi. (Model Selection).
- un retraining finale della rete con unendo il training con il validation e misurando le capacità predittiva sul test sett (Model Assessment).

La prima fase di selezione del modello migliore inizia con la selezione delle funzioni di attivazione e delle topologie. Le funzioni di attivazione scelte per il dataset sono, per i layer interni sempre la sigmoide, per il layer di output la funzione identità. La funzione identità ci è sembrata la più idonea considerato il range delle features, abbiamo escluso infatti altre funzioni di attivazione che avessero come codominio (-inf,+inf).

La definizione di una prima topologia da utilizzare è stata eseguita valutando il numero dei nodi del layer interno. Le prime prove con unità basse riportano risultati mediocri con valori di MEE sul validation intorno a 2.00 - 1.50.

Ci siamo chiesti se i pattern nel dataset avessero particolari valori che potessero risultare fuorvianti affinché la rete generalizzasse bene. Per cui abbiamo fatto alcune prove con la regolarizzazione delle features di input: scalati tutti i valori in un range [0,1] non abbiamo ottenuto significativi miglioramenti; abbiamo provato una standardizzazione dei valori con media 0 e varianza 1, prima considerando tutti le features come appartenenti alla stessa

distribuzione, poi rispetto alla media e varianza delle singole features, ma in entrambi i casi non ha aiutato in modo significativo a diminuire i valori della funzione di perdita.

Un successivo e progressivo aumento delle unità sembra invece portare ad un miglioramento in termini di MEE raggiungendo risultati efficienti utilizzando tra le 20 e le 30 unità. Un altro aspetto topologico da considerare è l'utilizzo di due unità nel layer di output, per far fronte ai due attributi di target presenti. Non abbiamo fatto alcuna esperimento che utilizzasse 2 reti differenti per predirre i singoli output, rispettivamente.

Tenendo conto dei lunghi tempi a cui saremmo andati incontro con così tante unità nell'hidden layer, abbiamo voluto impostare lo stesso esperimento con la libreria Keras e o valori che abbiamo ottenuto con la liberia ci hanno convinto a proseguire verso la scelta di un numero elevato di unità per la grid search.

La seconda fase cerca di comprendere, a partire dall'analisi, quali siano i migliori iperparametri da utilizzare nella Grid Search. Abbiamo provato due Grid Search, una per una topologia con 20 unità e l'altra per una topologia da 30 unità nel layer interno.

Gli iperparametri scelti per questa Grid Search sono riportati nella Tabella 2.

Per ogni configurazione di iperparametri sono state testate 5 istanze diverse.

Eta (Learning Rate)	0.01, 0.05, 0.1
Alfa (Momentum)	0.3, 0.7
Lambda (Regolarizzazione)	0.00, 0.01
Modalità	"Online", "Batch", "Mini-batch"
Batch Size	50, 100

Tabella 2. Iperparametri Grid Search sulle reti da 20 e 30 unità

Dopo aver avviato le due Grid abbiamo cercato i risultati medi migliori sulle cinque configurazioni con pesi iniziali diversi, ottenendo i risultati riportati nella Tabella 3. Per ogni configurazione di iperparametri sono state testate 5 istanze diverse.

Unità	**Eta**	**Alfa**	**Lambda**	**Modalità**	**Batch Size**	**Validation MEE**
30	0.1	0.3	0	mini-batch	50	**1.1567**
30	0.1	0.7	0	mini-batch	100	**1.1649**
20	0.1	0.7	0	mini-batch	100	**1.1926**

20	0.1	0.3	0	mini-batch	100	**1.2574**

Tabella 3. Risultati migliori della Grid Search sulle reti da 20 e 30 unità

La scelta dei modelli migliori ricade, nonostante gli MEE più bassi, su quelli con 20 unità perchè tendono ad essere più stabili rispetto ai modelli da 30 (**Figura A.3**).

Una volta scelta la topologia da 20 unità effettuiamo nuovamente una grid search più raffinata con gli iperparametri riportati nella **Tabella 4**. Per ogni configurazione di iperparametri sono state testate 5 istanze diverse

Eta (Learning Rate)	0.05, 0.15, 0.1
Alfa (Momentum)	0.3, 0.7
Modalità	"Mini-batch"
Batch Size	100, 128

Tabella 4. Iperparametri Grid Search sulla rete da 20 unità

Una volta eseguita la Grid Search più raffinata abbiamo ottenuti i modelli migliori riportati nella **Tabella 5**. Per ogni configurazione di iperparametri sono state testate 5 istanze diverse.

Unità	**Eta**	**Alfa**	**Lambda**	**Modalità**	**Batch Size**	**Validation MEE**
20	0.15	0.7	0	mini-batch	128	**1.1601**
20	0.15	0.3	0	mini-batch	128	**1.1848**

Tabella 5. Risultati Migliori della Grid Search sulle 20 unità

Abbiamo deciso di scegliere la seconda combinazione(**Figura 5** e **Tabella 6**) di iperparametri poiché i valori dei diversi tentativi si distaccano in modo minore dalla media, rispetto alla prima configurazione (**Figura A4** in appendice). Abbiamo tenuto in considerazione il fatto che per sfruttare al massimo i dati disponibili avremmo potuto rinunciare a dati per il training solo per valutare il rischio, quindi abbiamo optato per una combinazione di iperparametri che ci garantisse una maggior affidabilità sulle predizioni, meno lontana dalla media calcolata durante la fase di model Assessment.

Unità	**Eta**	**Alfa**	**Lambda**	**Modalità**	**Batch Size**	**Validation MEE**
20	0.15	0.3	0	mini-batch	128	**1.1848**

Tabella 6. Miglior Modello selezionato nella fase di Model Selection

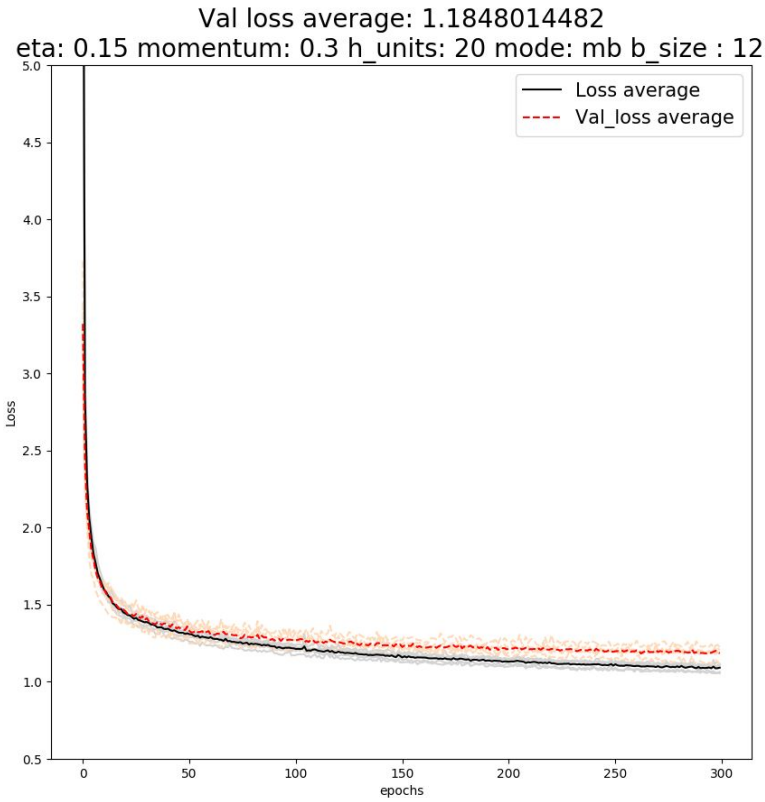

Figura 5. Grafico MEE modello scelto alla fine del Model Selection

Dopo aver selezionato il modello migliore addestriamo nuovamente la rete tenendo in considerazione anche il test set(**Tabella 7**).

LOSS MEE	VALIDATION LOSS MEE	TEST MEE
1.0056	**1.3638**	**1.1465**

Tabella 7. Risultati addestramento della rete tenendo in considerazione il test sul MEE

Prima di applicare il potere della nostra rete sul blind test è necessario addestrarla nuovamente unendo il training e il validation per poi misurare l'errore sul test set, che noi abbiamo deciso comporsi del 20% del dataset *Cup*. Cosi facendo stimiamo l'errore che la rete potrà commettere nella predizione di risultati futuri. L'errore sul test set ottenuto in questa fase è pari a **1.2581**. Un rappresentazione della predizione della rete è mostrato in **Figura A.5:** è il plot bidimensionale degli output predetti sovrapposti ai valori target.

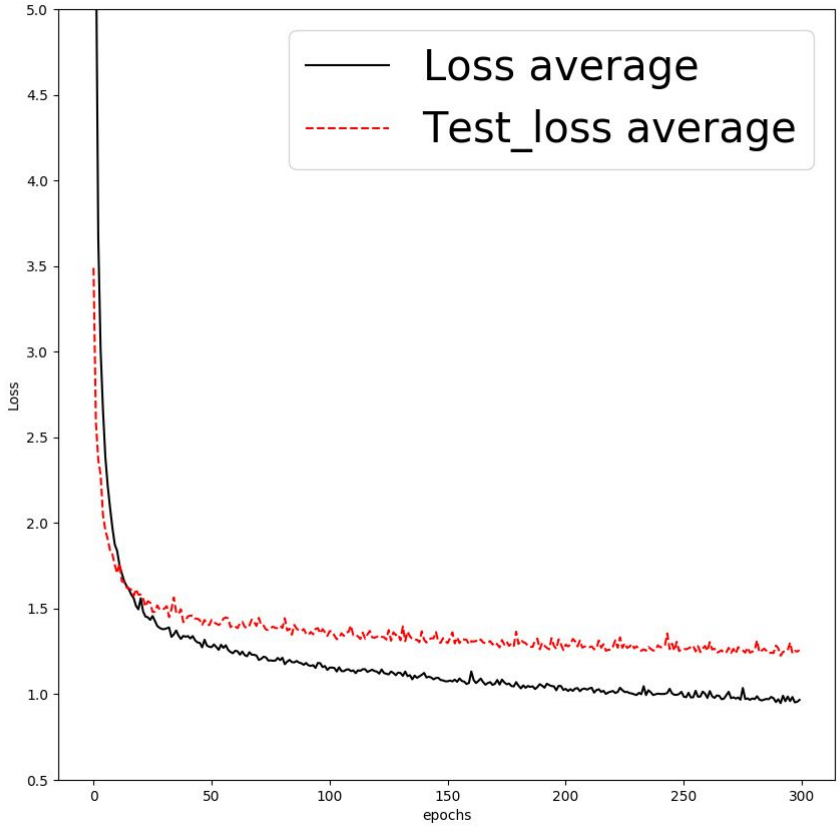

Figura 6. Grafico MEE del training effettuato sul develop set e test set della rete finale.

Infine è fondamentale addestrare un ultima volta la rete utilizzando l'intero dataset, cosi da ottenere sperabilmente i migliori risultati sul Blind Test e sfruttare completamente la capacità plastica della rete di approssimare la funzione rappresentata nei dati.

4. CONCLUSIONS

L'utilizzo di una rete neurale di tipo FeedFoward inizia con la selezione dei migliori iperparametri da utilizzare. Gli iperparametri determinano la capacità della nostra rete di poter predire e generalizzare bene sui test set. Dai risultati ottenuti, riportati nelle tabelle e nei grafici, si può osservare come i monk non abbiano bisogno di troppe unità per poter predire il test(5-10 unità), è soprattutto, in generale, convergono velocemente in poche epoche. I risultati della regressione mostrano invece la necessità di utilizzare più unità per avere un calo dell'errore abbastanza significativo. Inoltre addestrare nuovamente la rete misurando l'efficienza sul test set, sembra dare comunque dei buoni risultati nonostante un leggero aumento dell'errore sul test set(**Figura 6**).

BLIND TEST RESULTS: networks_benders_ML-CUP17-TS.csv , *networks_benders*

ACKNOWLEDGEMENTS

Il gruppo *networks_benders* acconsente alla propria partecipazione alla Blind Competition. Con annessa pubblicazione dei nomi e della classifica preliminare e finale dei risultati sul Blind Test.

REFERENCES

1. Mitchell TM. Machine Learning [Internet]. [cited 4 Feb 2018]. Available: http://www.cs.ubbcluj.ro/~gabis/ml/ml-books/McGrawHill%20-%20Machine%20Learning%20-Tom%20Mitchell.pdf

APPENDICE. A.

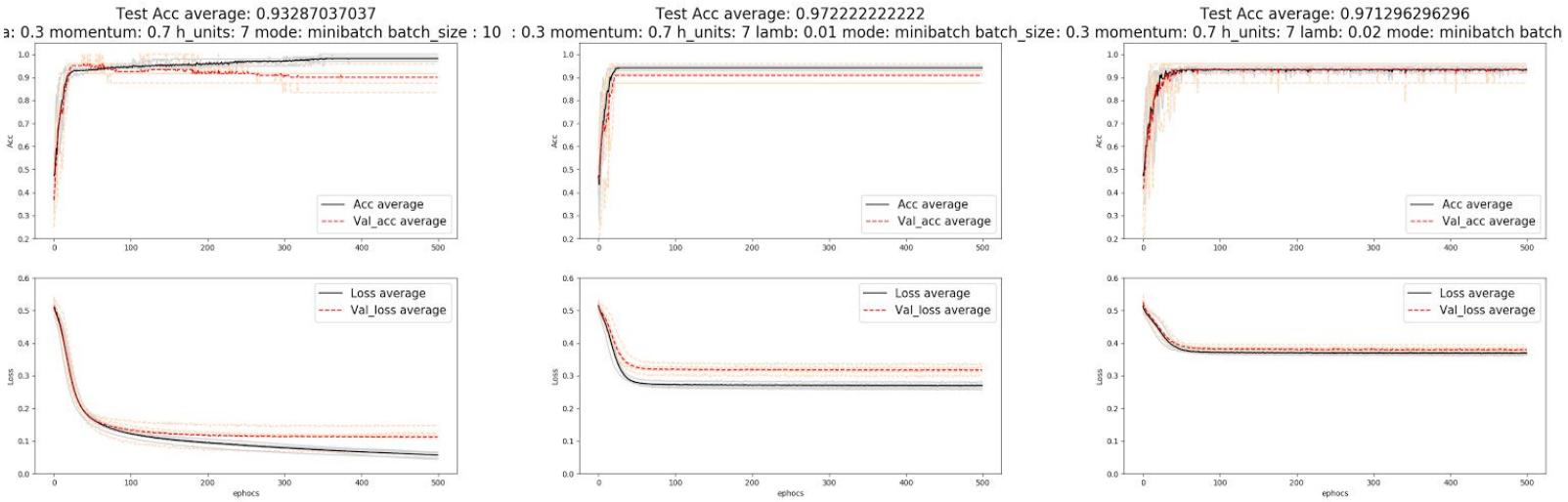

Figura A.1. Effetti che provoca la regolarizzazione su diversi tentativi di training.
A parità di iperparametri la variazione la vediamo nel parametro di regolarizzazione, 0.0, 0.01, 0.02
L'intervallo della funzione di perdita, quindi il mean square error, aumenta proporzionalmente con lambda.
Analogamente accade alla distanza tra le funzioni di perdita. Con un lambda uguale a 0, la linea continua della loss del training è piuttosto lontana da quella della loss sul validation. Con un lamba più alto, 0.01 la distanza tra le due linee diminuisce, fino alla distanza che si vede nella terza figura, con un lambda pari a 0.02.
Analogamente accade all'accuratezza: seppur un coefficiente di penalità lambda inesistente aumenti l'accuratezza nel training, la distanza che lo separa dalla linea della funzione di loss sulla validazione è grande. Ad aumentare il parametro lambda, anche la distanza tra l'accuratezza ottenuta nei pattern usati per il train e la linea dell'accuratezza nel validation set va diminuendo (come la distanza tra le loss).
Utilizzare la penalizzazione dei pesi sembra quindi peggiorare le prestazioni della rete nel training ma migliorarne le capacità di generalizzare.

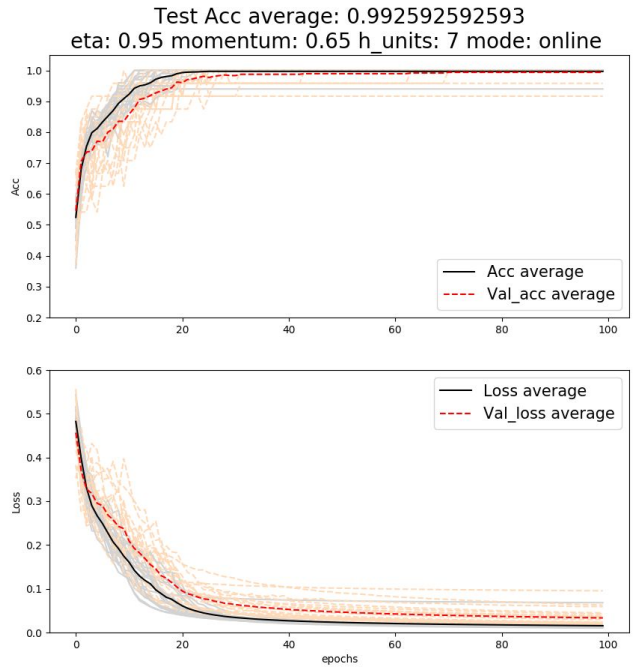

Figura A.2. Accuracy e Loss(MSE) medi sul TR e sul TS del secondo modello migliore sul Monks-1.

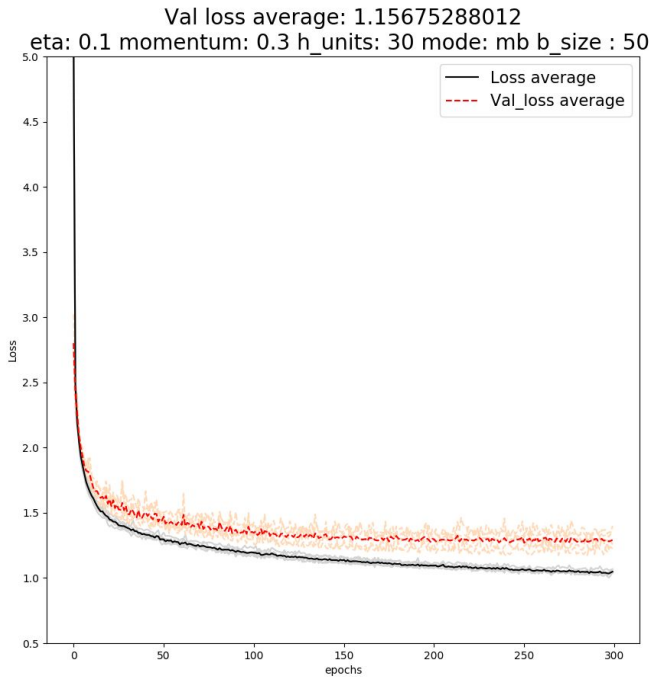

Figura A.3. Loss(MEE) medi sul TR e sul VL della topologia con 30 hidden units, con i migliori risultati.
Le linee che disegnano le funzioni di perdita si distaccano molto dalla media, per cui abbiamo ritenuto la topologia meno affidabile.

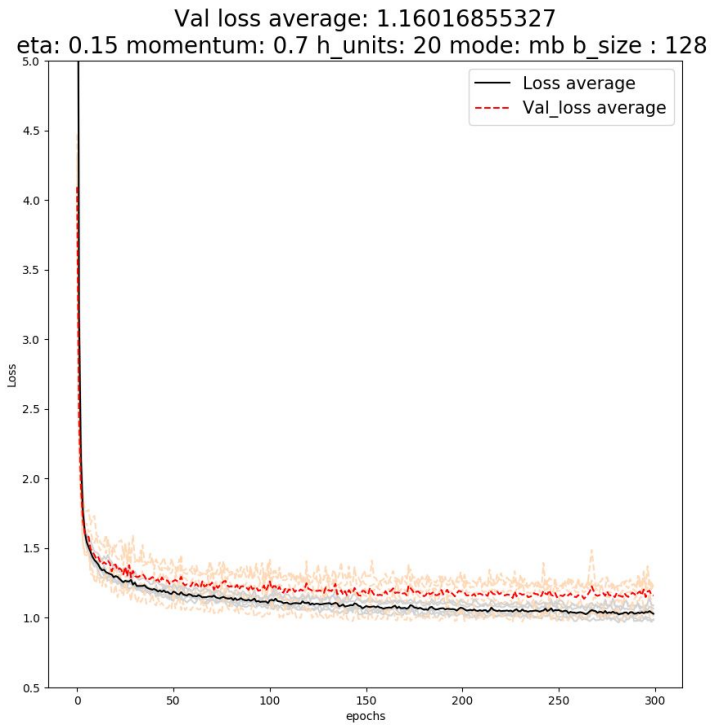

Figura A4. Con la configurazione presentata nella prima riga della Tabella 5 abbiamo ottenuto risultati migliori rispetto agli iperparametri che abbiamo scelto per il modello finale.

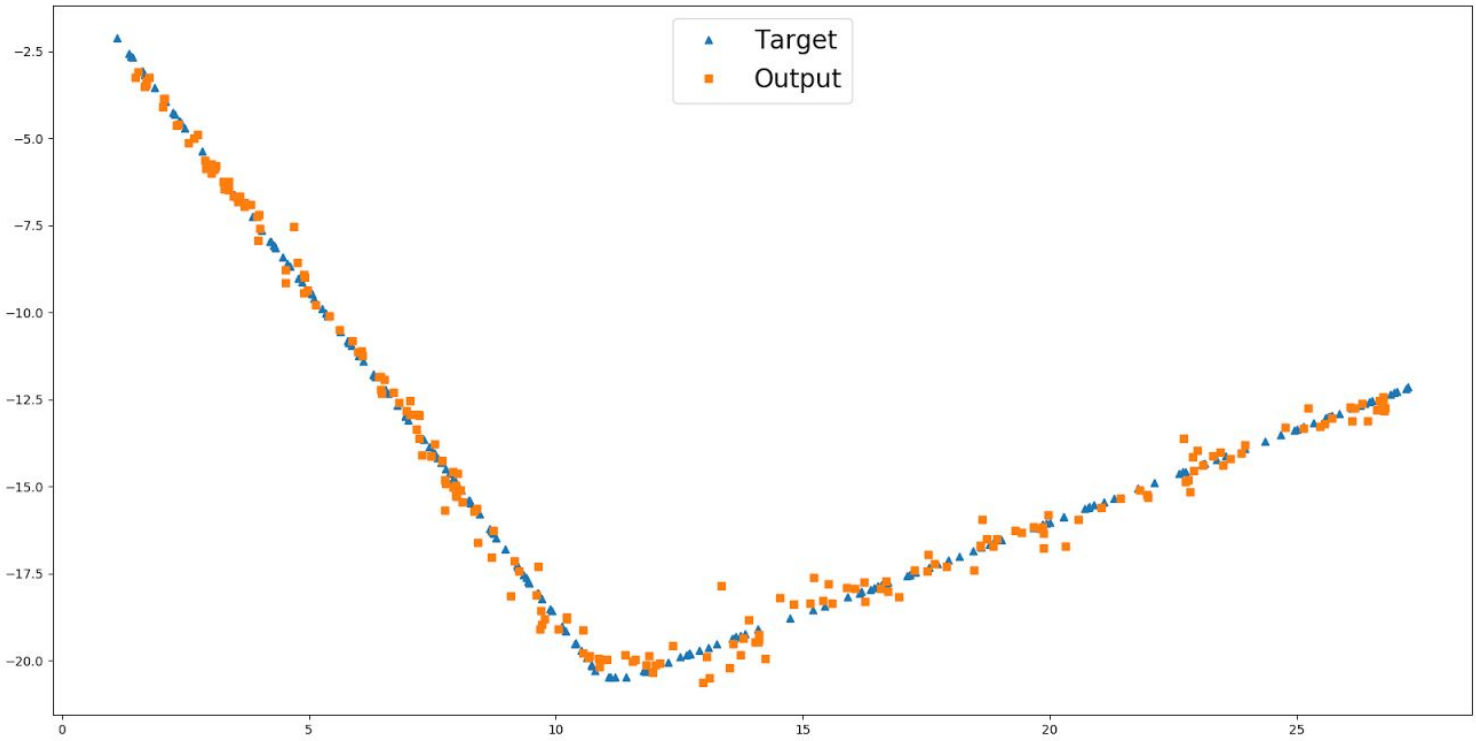

Figura A.5. La rappresentazione della predizione della rete rispetto ai valori target