UNIVERSITÀ DI PISA
DIPARTIMENTO DI INFORMATICA
MACHINE LEARNING 2018/2019 (654AA)

Project Report (A)

Matteo Colanero, Simone Bombari

Maggio 2019

Abstract

Il nostro progetto consiste in una rete neurale MLP fully connected, costituita da un singolo hidden layer nel caso del modello finale utilizzato nella ML-CUP, ma utilizzabile per costruire una rete neurale con un numero arbitrario di layer. Abbiamo implementato, e usato nei casi in cui si sono mostrati rilevanti, la regolarizzazione e il momentum; come tecnica di validazione abbiamo usato una k-fold cross validation, assieme ad una stop condition dinamica.

1 Introduzione

Descriviamo i nostri obiettivi e le prime assunzioni che abbiamo adottato nello svolgimento del progetto:

- Il nostro obiettivo è quello di costruire una rete neurale MLP in *python*, che consenta di prevedere i target di alcuni *data-set* (*monkset* e ML-CUP), risolvendo problemi sia di classificazione binaria che di *regression*. In questo modo possiamo sperimentare varie tecniche di regolarizzazione e applicare alcune varianti per valutarne i risultati. Ragionando in maniera empirica cerchiamo di capire come e quando sperimentare tali varianti.
- Abbiamo utilizzato come modello una rete neurale *fully-connected*. Le caratteristiche del modello sono state opportunamente variate nel corso del lavoro (per esempio il numero di *hidden-layer* e il numero di neuroni per ciascuno di essi) per esplorare maggiori possibilità. Abbiamo utilizzato l'algoritmo di *backpropagation* con uno *Stochastic Gradient Descent* per aggiornare i pesi w_{ij} e i bias b_i appartenenti ai *layer* della rete.
- La nostra principale assunzione a priori è stata utilizzare una rete *fully-connected*, per non imporre una topologia in particolare e fare in modo che il nostro modello sia utilizzabile per un'ampia classe di problemi.

2 Metodo

Elencheremo di seguito una sintetica descrizione del nostro progetto:

- Per costruire il progetto abbiamo utilizzato *Python* come linguaggio di programmazione. Abbiamo utilizzato le librerie `numpy`, `random`, `copy` e `csv`.
- Abbiamo creato la classe **NN** contenente in particolare le funzioni `backpropagation` e `update` (che aggiorna i pesi e i bias della rete nel processo di *training*); e la classe **Layer** contenente i pesi, i bias. La funzione per la *cross-validation* è definita al di fuori della classe **NN**.
- Abbiamo lavorato in genere con un solo *hidden layer* (dove non specificato); come funzione di attivazione abbiamo utilizzato la *sigmoid* per i tre *monkset*, per quanto sono stati fatti tentativi anche con la tangente iperbolica.
  Nel caso della ML-CUP nel layer di neuroni di output abbiamo utilizzato l'identità come funzione di attivazione, dato che a differenza dei *monkset* i target non sono più compresi fra 0 e 1.
- Abbiamo utilizzato sia il *batch-method*, sia l'*on-line-method*. La maggior parte del lavoro è stato svolto però con delle *mini-batch* contenenti ciascuna un numero di *sample* dell'ordine della decina, in modo da conciliare il tempo di computazione e la stabilità. Riporteremo anche questo parametro nella presentazione dei risultati.
- L'inizializzazione dei pesi della nostra rete avviene in maniera stocastica (per evitare di ostacolare il *training*): ciascuno di essi è un numero casuale compreso fra -0.7 e 0.7.
- Per ogni *dataset* abbiamo fatto una *grid-search* sugli iperparametri migliori (*learning rate* η, *regularization parameter* λ, *momentum parameter* α) effettuando una *k-fold cross-validation*, imponendo generalmente $k = 5$ (dove non specificato). Anche se non necessario, abbiamo adottato tale approccio anche nel caso dei *monkset*. Sono state effettuate diverse *grid-search*, cercando di affinare sempre meglio la precisione sui parametri ottimali. Abbiamo cercato di ottimizzare questa *screen phase* interpretando empiricamente i risultati ottenuti di volta in volta.
- Come *loss function* abbiamo utilizzato (senza contare il termine di regolarizzazione) la MSE per i *monkset* e la MEE per la ML-CUP, in modo da usare le stesse funzioni con cui poi abbiamo valutato la performance del modello.
- Abbiamo impostato come segue la *stop-condition* (nel nostro caso il numero di iterazioni è uguale al numero di *epochs* moltiplicato per una costante c, dove c è il rapporto tra il numero di sample presenti nel *training-set* e la dimensione delle nostre *mini-batch*):
  - Nella *cross-validation* impostiamo un numero massimo di *epochs*, oltre il quale interrompiamo il *training* della rete (in genere dell'ordine delle migliaia);
  - Ogni s iterazioni ($s = 1000$ per i *monkset*, $s = 4000$ per la ML-CUP) mediamo la *loss function* sulle ultime a iterazioni (o l'*error rate* nel caso dei *monkset*), con $a = 200$ per i *monkset* e $a = 400$ per la ML-CUP. Se tale media si presenta maggiore o uguale alla precedente usciamo precocemente dal processo di training, salvando il numero totale di iterazioni k_{max}.
  - Valutiamo k_{max} al variare dei processi di *training* sugli iperparametri migliori. In questo modo effettuiamo una stima sul numero di iterazioni necessarie per il *training* su tutto il dataset, in vista del controllo sul *test-set*.

- Per il preprocessing dei dati nel caso dei *monkset* abbiamo optato per la codifica *"1-of-k"*, ciò corrispondeva ad avere 17 unità di input. Per la ML-CUP ci siamo limitati ad inserire i valori reali del *dataset* in 10 unità di input.
- Come già accennato sopra abbiamo effettuato una *k-fold cross validation* sia sui *monkset* sia per la ML-CUP. Inoltre:
  - Nel caso dei *monk* abbiamo diviso randomicamente il *training-set* in k parti disgiunte, abbiamo usato ciascuna come *validation-set* effettuando il *training* sulle altre $k-1$ unite. In questo modo abbiamo potuto effettuare indipendentemente k *grid-search* per valutare quali fossero i migliori iperparametri e i migliori modelli (da utilizzare poi per effettuare il *training* su tutto il *training-set*, per valutare infine i risultati sul *test-set*).
  - Nel caso della ML-CUP abbiamo suddiviso randomicamente il *training-set* fornito in due file complementari: *internal-training-set* e *internal-test-set*, con proporzione 3:1. Questi due file li abbiamo poi utilizzati in maniera identica a come abbiamo utilizzato rispettivamente *training-set* e *test-set* nel caso dei *monk-set*.
  - In entrambi i casi valutavamo la media dei risultati del MSE (*monk-set*) o del MEE (ML-CUP) sui k *validation-set*. A seguire di questa valutazione, decidevamo opportunamente se affinare la *grid-search*, o se fissare la sequenza di iperparametri dal nostro punto di vista ottimali.

3 Esperimenti

Elencheremo di seguito i risultati ottenuti, indicando come segue gli iperparametri e le variabili costituenti i modelli:

- η - *learning rate*
- λ - *regularization parameter*
- α - *momentum parameter*
- k_{hl} - Numero di neuroni nell'*hidden-layer*
- n_{batch} - Numero di sample che costiscono ciascuna *mini-batch*
- ES - *stopping-point* in unità di *epochs*

3.1 Monk-set

Riportiamo nella Tabella 1 gli iperparametri e le variabili del modello scelte dopo la *grid-search* su ciascun *monk-set*; in Tabella 2 riportiamo invece il MSE e l' *Accuracy* rispettivamente per il *training-set* (TR) e per il *test-set* (TS), tali valori sono il risultato di una media su 10 *training* successivi. L'ultima riga di ciascuna tabella contiene il task su *Monk3* dove abbiamo imposto $\lambda = 0$.

Task	η	λ	α	k_{hl}	n_{batch}	ES
Monk1	0.4	10^{-5}	0.9	10	10	167
Monk2	0.8	10^{-5}	0.9	15	30	333
Monk3	0.2	10^{-4}	0	30	30	250
Monk3 (no reg)	0.2	0	0	30	30	250

Tabella 1: Iperparametri e variabili del modello scelti per i *monk-set*.

Task	MSE (TR)	MSE (TS)	Accuracy (TR)	Accuracy (TS)
Monk1	$2.1 \cdot 10^{-3}$	$5.1 \cdot 10^{-3}$	100%	100%
Monk2	$1.1 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$	100%	100%
Monk3	$1.1 \cdot 10^{-1}$	$9.0 \cdot 10^{-2}$	93.4%	97.2%
Monk3 (no reg)	$1.1 \cdot 10^{-1}$	$8.8 \cdot 10^{-2}$	94.1%	96.7%

Tabella 2: *Mean square error* e *Accuracy* per i *monk-set*.

Riportiamo di seguito i plot del MSE e dell' *Accuracy* (Figura 1 e 2), cercando di evidenziare gli aspetti principali di ogni task. Le linee nere verticali indicano lo *stopping-point* ottenuto precedentemente tramite *cross-validation*; plottiamo comunque anche *epochs* successive per poter apprezzare stabilità o possibili situazioni di *over-training*.

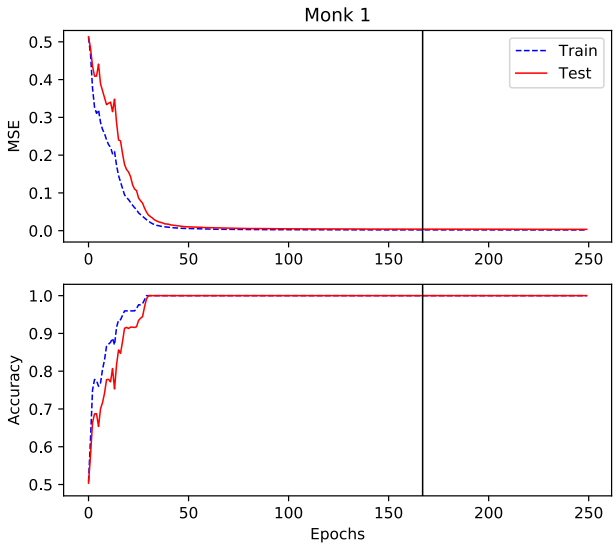

Figura 1: Monk1 con i parametri indicati in Tabella 1.

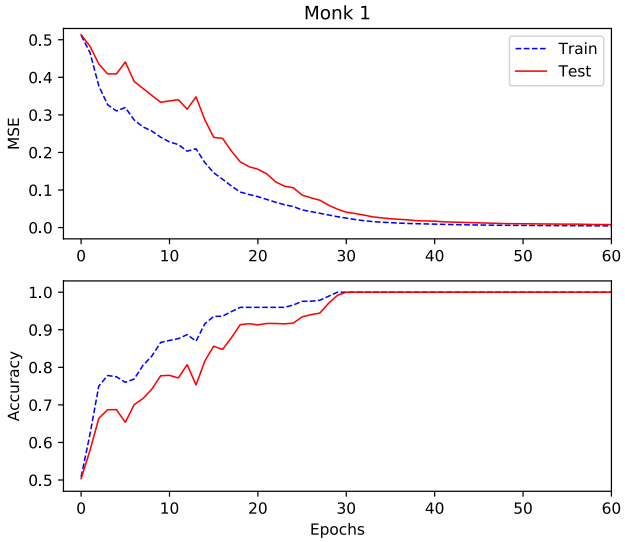

Figura 2: Monk1 con i parametri indicati in Tabella 1, zoom sulla prime *epochs*.

A seguire (Figura 3 e Figura 4) riportiamo due plot di *Monk1* con parametri non ottimali. Questi parametri li abbiamo esplorati all'interno una *grid-search* precedentemente svolta, e abbiamo ritenuto opportuno riportare i grafici come esempio di instabilità. In particolare vediamo come quest'ultima aumenta da Figura 3 a Figura 4 a causa di un aumento dell'iperparametro di regolarizzazione.

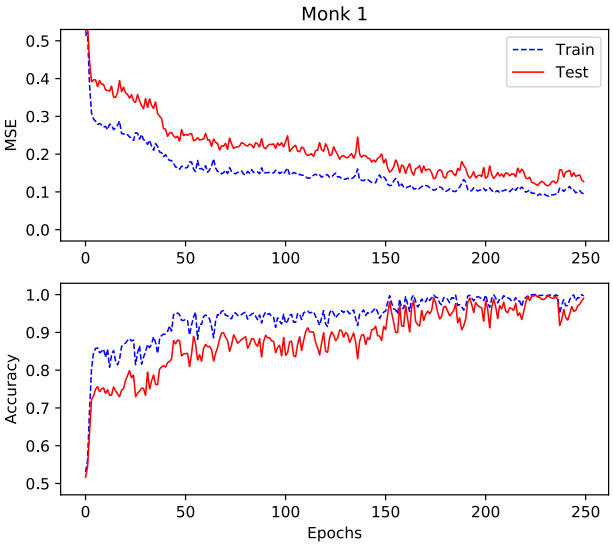

Figura 3: Monk1 con $\eta = 0.3$, $\lambda = 4 \cdot 10^{-4}$, $\alpha = 0.9$, $k_{hl} = 15$, $n_{batch} = 10$.

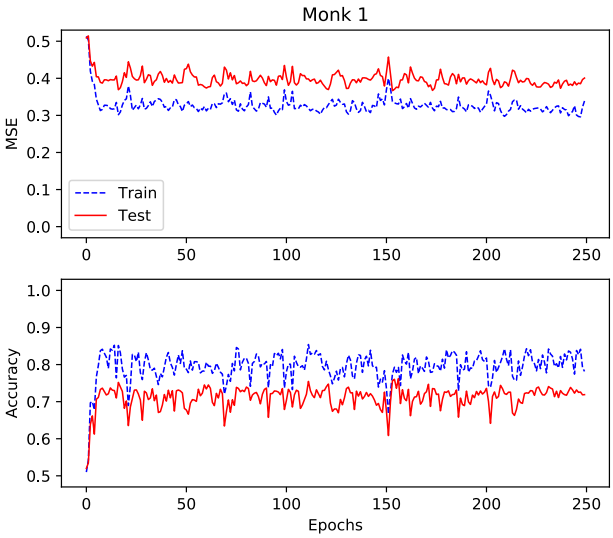

Figura 4: Monk1 con i parametri indicati nella didascalia di Figura 3, ma con $\lambda = 10^{-3}$.

Riportiamo ora due plot per *Monk2* in Figura 5 e Figura 6. Rispetto a prima riportiamo direttamente lo zoom sulle prime *epochs* (Figura 5), e un esempio di *under-fitting* variando gli iperparametri ottimali (in particolare abbiamo utilizzato $k_{hl} = 3$ e $\lambda = 10^{-3}$)

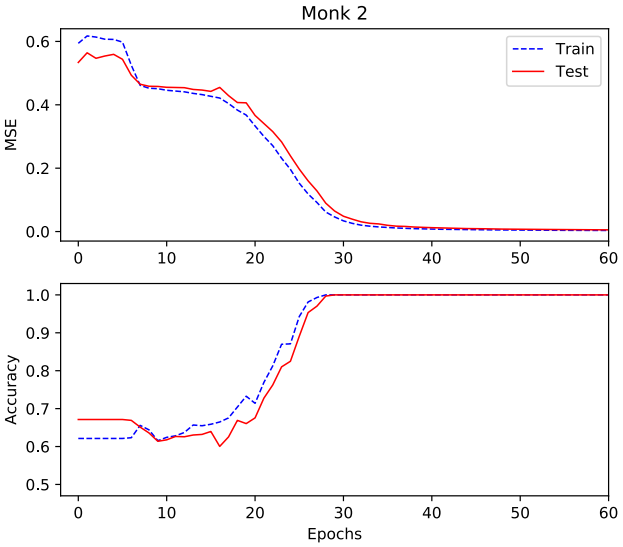

Figura 5: Monk2 con i parametri indicati in Tabella 1, zoom sulla prime *epochs*.

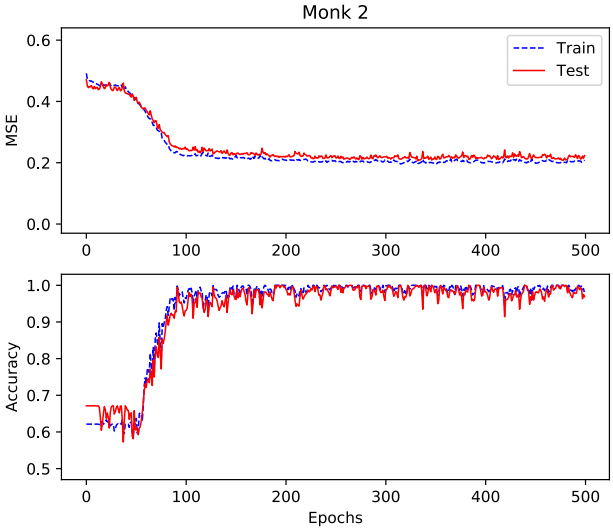

Figura 6: Monk2 con $\eta = 0.45$, $\lambda = \cdot 10^{-3}$, $\alpha = 0.9$, $k_{hl} = 3$, $n_{batch} = 30$

Riportiamo infine i grafici di *Monk3* (Figure 7, 8, 9), rispettivamente utilizzando un asse delle *epochs* che permetta di vedere lo *stopping-point*, uno zoom sulla prima parte e uno zoom che evidenza il fenomeno di *over-training*. Ci aspettiamo in quest'ultimo task la presenza di questo fenomeno, data l'aggiunta di rumore sui dati di *training*.

In Figura 10 riportiamo invece i risultati di *Monk3* senza regolarizzazione (dei quale è presente un'ulteriore analisi nelle Conclusioni). Tutti i parametri fanno riferimento a quelli indicati in Tabella 1.

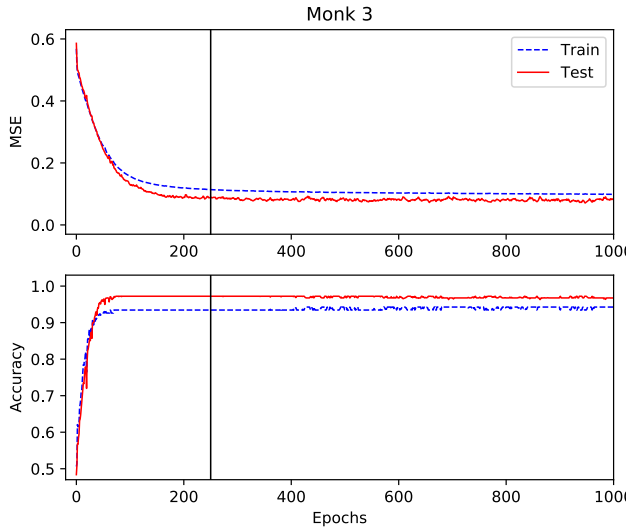

Figura 7: Monk3 con i parametri indicati in Tabella 1, evidenziato in nero lo *stopping-point*.

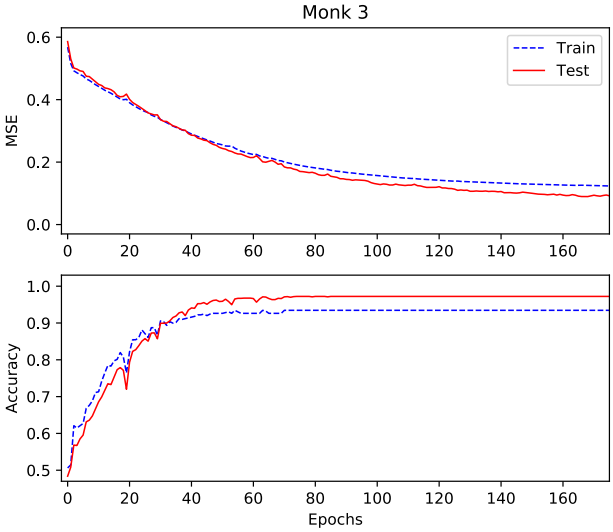

Figura 8: Monk3 con i parametri indicati in Tabella 1, zoom sulla prime *epochs*.

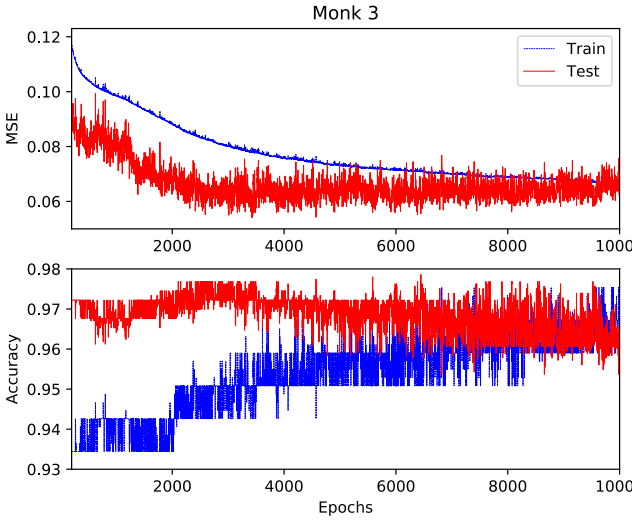

Figura 9: Monk3 con i parametri di Tabella 1, zoom sulla parte successiva allo *stopping-point*.

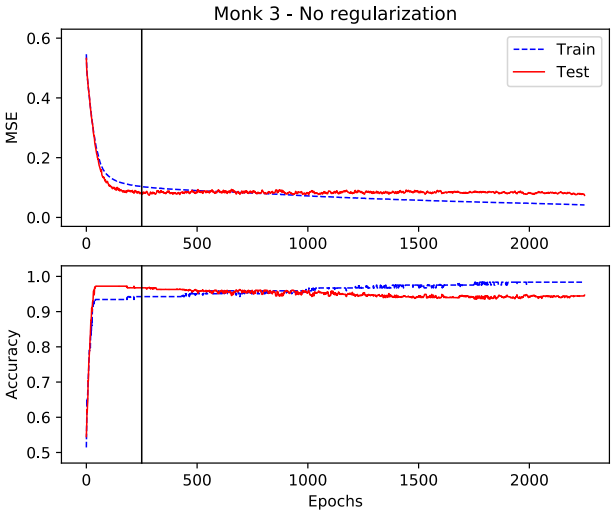

Figura 10: Monk3 no reg con i parametri indicati in Tabella 1.

3.2 ML-CUP

Abbiamo approcciato questo task come in precedenza cercando di scegliere il modello ottimale ed effettuando una *screen-phase* preliminare. Abbiamo scelto una rete neurale *fully-connected* con un solo *hidden layer* (non erano apprezzabili miglioramenti aumentando questo parametro).

Abbiamo preliminarmente considerato anche un modello con due *hidden-layer* e preprocessando i dati in modo che ciascuno dei 10 input fosse rappresentato da un array lunghezza 100 composto da 99 zeri e 1 unico uno, la cui posizione nell'array dipendeva dal valore dell'input in relazione al massimo e al minimo tra tutti gli input dello stesso tipo (codifica *"1-of-k"*). In questo modo abbiamo ottenuto un layer di input composto da 1000 unità; già nella fase di screening e in una prima *grid-search* molto ampia però questo modello dava risultati significativamente peggiori del modello precedente (MEE circa doppio). Nel seguito quindi tutte le considerazioni saranno riferite al modello di rete neurale *fully-connected* con un solo *hidden layer*.

Il numero di neuroni e la dimensione delle *minibatch* non fornivano variazioni apprezzabili nel range esplorato, e abbiamo deciso di fissarli rispettivamente a $k_{hl} = 30$ e $n_{batch} = 50$.

Abbiamo provato anche a cambiare la funzione di attivazione sul primo *layer*, ma non apprezzavamo differenze tra tangente iperbolica e la sigmoide. Abbiamo in conclusione scelto la sigmoide sul primo *layer* e l'identità sul secondo.

Come prima cosa abbiamo diviso i dati in *train-set* e *internal-test-set*, con una proporzione di 3:1; all'interno del *train-set* abbiamo utilizzato uno schema di *validation* come quello indicato nella sezione 2, ovvero una *k*-fold *cross-validation* con $k = 5$.

Cercando un'approccio empirico abbiamo esplorato come migliorava il risultato sul *validation-set* al variare degli iperparametri. Abbiamo effettuato poi una prima *grid-search* poco fine sugli iperparametri, ciò ci ha permesso di escludere valori di $\eta > 0.1$, $\lambda > 10^{-5}$, $\alpha > 0.45$. Abbiamo quindi fatto una nuova *grid-search* più fine nella regione che abbiamo considerato più importante; elenchiamo di seguito i valori che abbiamo assegnato agli iperparametri. Nelle *grid-search* effettuate abbiamo, come in precenza, imposto la *stop-condition* descritta nella sezione 2.

- η: 0.001, 0.005, 0.01, 0.05, 0.1;
- λ: 10^{-6}, 10^{-5};
- α: 0, 0.45.

Riportiamo in Tabella 3 i risultati più significativi, con i rispettivi MEE per la *validation* e il *training-set*. Questi ultimi valori li abbiamo ottenuti mediando sui 5 *fold*, in modo che fossero statisticamente più significativi.

η	λ	α	MEE (VL)	MEE (TR)
0.05	10^{-6}	0	1.11	0.91
0.01	10^{-6}	0	1.16	0.99
0.05	10^{-5}	0	1.12	0.90
0.05	10^{-6}	0.45	1.13	0.87
0.1	10^{-6}	0	1.12	0.87

Tabella 3: Iperparametri con i risultati migliori nell'ultima *grid-search*.

Scegliamo come iperparametri quelli indicati nella prima riga della Tabella 3. Con questi effettuiamo il *training* su tutto il *training-set*, e valutiamo il MEE su quest'ultimo e sull'*internal-test-set*. Tramite la *validation* otteniamo anche una stima dello *stopping-point*, che abbiamo determinato essere 1500 *epochs*. Con questi valori il tempo di computazione è di circa 7 minuti (con un computer Intel i7 *quad-core*).

Riportiamo in Figura 11 e 12 i plot della MEE per il *training-set* e per l'*internal-test-set*. In particolare in Figura 12 zoomiamo sull'asse della MEE per poter apprezzare l'*over-training* ad *epochs* troppo elevate. In nero come sopra indichiamo il nostro *stopping-point*. Questi grafici dunque riportano la MEE per TR e TS, sul modello finale che utilizzeremo per il *blind-test*.

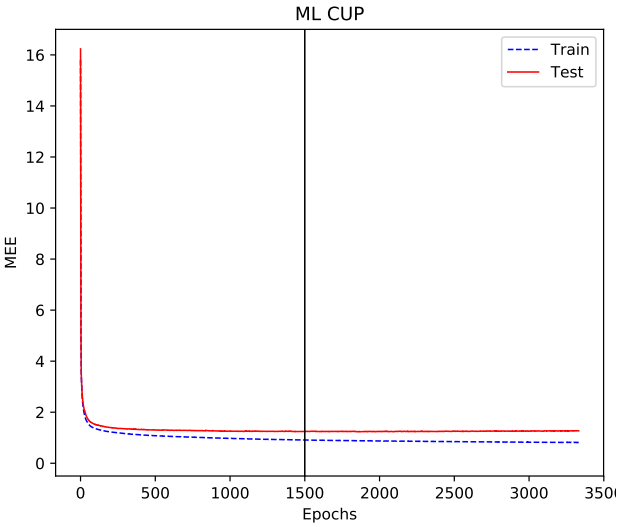

Figura 11: MEE per il *training-set* e per l'*internal-test-set*.

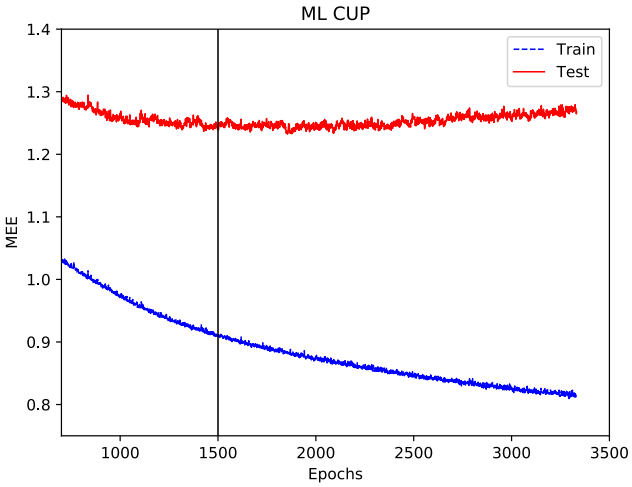

Figura 12: Stesso grafico di Figura 11, zoom sull'asse del MEE.

Dopo aver valutato diversi risultati sull'*internal-test-set*, procediamo con gli iperparametri ottimali ottenuti tramite *cross-validation*, per eseguire il *blind-test*.

In particolare, come riportato sopra $\eta = 0.05$ & $\lambda = 10^{-6}$ & $\alpha = 0$. Riportiamo la MEE media ottenuta su più *training* eseguiti sul *training-set* e sull'*internal-test-set* (con diverse inizializzazioni):

$$\text{MEE}_{\text{TR}} = 0.92 \qquad \text{MEE}_{\text{TS}} = 1.24$$

4 Conclusioni

Vogliamo aggiungere un ulteriore commento sui grafici ottenuti variando gli iperparametri sui *monk*.

In Figura 3 e Figura 4 (Monk1) stiamo sostanzialmente vedendo un'eccessiva costante di regolarizzazione che rende instabile il modello. Aumentarla di un fattore 40 (Figura 3) rispetto al valore ottimale rende più lento il *learning*, che tuttavia arriva a fornire valori dell'*accuracy* prossimi al 100%. Aumentarla di un fattore 100 (Figura 4) non ci permette di ottenere un'*accuracy* accettabile e aumenta ulteriormente l'instabilità. Abbiamo così verificato come la costante λ è inversamente correlata alla complessità del nostro modello.

Una diminuzione della complessità è associata anche da una diminuzione del numero di neuroni nell'*hidden-layer*, infatti anche in Figura 6 (Monk2) notiamo un aumento della MSE minima rispetto alla Figura 5. In questo caso il grafico è più stabile, ma siamo comunque in condizione di *under-fitting*. Caso opposto è quello raffigurato in Figura 10 (Monk3 senza regolarizzazione) dove, specialmente nel sub-plot dell'*accuracy*, si nota (ad *epochs* elevate) *over-fitting* (la condizione di *stopping-point* ci permette di non overtrainare la rete in questo caso). In particolare $\lambda = 0$ fa mancare il termine di regolarizzazione, portando il modello a fittare il rumore (presente in Monk3) sui dati di input.

BLIND TEST RESULTS: LiteWeight_ML-CUP18-TS.csv
Nickname: LiteWeight

Accettiamo la divulgazione e la pubblicazione dei nostri nomi e dei risultati della classifica preliminare e finale.