Machine Learning Project

Samuel Fabrizi
samuel.fabrizi97@gmail.com

Riccardo Amadio
riccardo.amadio1@gmail.com

ML course (654AA), 2019/2020
Type of project: A

Abstract

Questo report descrive l'implementazione in Python di una Feedforward Neural Network che offre la possibilità di utilizzare diversi algoritmi di ottimizzazione. Abbiamo aggiunto alcune tecniche di model selection basate sull'utilizzo del K-fold cross validation per permettere un'accurata ricerca dei migliori iperparametri. La rete neurale è stata testata su task supervisionati di classificazione (Monk) e di regressione (Cup).

1 Introduzione

Il progetto consiste nell'implementazione di una rete neurale in grado di svolgere task supervisionati di classificazione e regressione.
L'obiettivo del progetto è stato quello di costruire un modello funzionante prestando particolare attenzione alla ricerca dei migliori iperparametri durante la fase di model selection. In particolare ci siamo soffermati sulla creazione di un modello scalabile lasciando all'utente la possibilità di poter scegliere l'architettura reputata più oppurtuna.
Al fine di raggiungere questi obiettivi abbiamo creato un modello che offre la possibilità di utilizzare diversi ottimizzatori (SGD, RMSprop, Adam), diverse funzioni di inizializzazione dei pesi e diversi criteri di arresto (vedi Sez.2.1). Infine ci siamo dedicati all'implementazione di tecniche robuste di model selection che permettano la ricerca dei migliori iperparametri per un dato task. Abbiamo implementato due tecniche di model selection (grid e random search) che utilizzano come tecnica di validazione il K-fold (vedi Sez. 2.3).

2 Metodo

In questa sezione andremo a descrivere l'architettura del nostro modello e una panoramica delle scelte adoperate per la realizzazione del progetto. Più in dettaglio nel paragrafo 2.1 presentiamo le funzionalità del modello implementato e la struttura dello stesso; successivamente nel paragrafo 2.2 passiamo alla descrizione dei principali aspetti del codice e delle scelte implementative; infine, nel paragrafo 2.3 descriviamo le tecniche di model selection disponibili.
Il software è stato implementato utilizzando il linguaggio di programmazione *Python*. Inoltre abbiamo utilizzato le seguenti librerie:

- *numpy* per poter usufrire dei metodi vettorizzati che fornisce;
- *pandas* unicamente per la scrittura dei risultati in formato *csv*;
- *scsikit-learn* per l'esecuzione della Principal Component Analysis;
- *multiprocessing* per l'implementazione parallela delle tecniche di model selection.

2.1 Panoramica del modello

Il modello implementato rappresenta una Feedforward Neural Network composta da livelli fully connected. Per lasciare all'utente la scelta dell'ottimizzatore reputato più oppurtuno abbiamo implementato le seguenti tecniche:

- Steepest Gradient Descent con learning rate, momentum e Nesterov momentum. Inoltre, è possibile utilizzare la strategia del *decay learning rate* [1];
- RMSprop;
- Adam [2].

Abbiamo anche lasciato la possibilità all'utente di aggiungere un regolarizzatore L2 a tutti gli ottimizzatori per prevenire l'overfitting. Bisogna però precisare come nel caso degli ottimizzatori RMSprop e Adam, abbiamo deciso di implementare la tecnica del weight decay piuttosto che il regolarizzatore L2. Nel caso dello SGD le due tecniche risultano essere uguali [3].
Ogni livello della rete neurale accetta diversi parametri che permettono la personalizzazione del livello in base alle esigenze del problema (come ad esempio la funzione di attivazione, il numero di nodi, ecc. . .). In particolare ci siamo preoccupati di offrire diversi metodi di inizializzazione al fine di poter favorire il training. A tal proposito abbiamo aggiunto le seguenti funzioni di inizializzazione:

- Inizializzazione di He [4]: genera campioni da una distribuzione normale con media 0 e deviazione standard $\sqrt{\frac{2}{fan_{in}}}$;
- Inizializzazione uniforme e normale di Xavier [5]: genera rispettivamente campioni nell'intervallo $[-\alpha, +\alpha]$ con $\alpha = \sqrt{\frac{6}{fan_{in}+fan_{out}}}$ e campioni da una distribuzione normale con media 0 e deviazione standard $\sqrt{\frac{2}{fan_{in}+fan_{out}}}$

Dopo aver aggiunto i diversi livelli alla rete neurale è possibile invocare la funzione *compile* al fine di inizializzare i vari parametri della rete, passando come argomento l'algoritmo di ottimizzazione scelto.
Per allenare il modello possono essere utilizzate la versione online, la minibatch o l'intera batch a discrezione dell'utente (batch size di default = 32). La condizione di terminazione del training può essere una condizione assoluta fissando una tolleranza oppure relativa, specificando una funzione di *early stopping* mediante l'argomento *callback*. Le funzioni di *early stopping* implementate sono GL_α e PQ_α [6].

2.2 Scelte implementative

Di seguito verranno brevemente descritte le funzionalità dei principali moduli:

- *parser*: si preoccupa della lettura dei dataset (Monks e Cup) e della divisione tra samples e targets. Inoltre nel caso del Monk effettua il one-hot encoding dell'input;
- *neural_network*: rappresenta lo scheletro di una Feedforward Neural Network;
- *layer*: implementa la struttura di un livello neurale fully connected;
- *optimizers*: contiene gli ottimizzatori implementati (vedi sezione 2.1);
- *kernel_initialization*: contiene le funzioni per l'inizializzazione dei parametri;
- *functions_factory*: factory utilizzata per la creazione delle diverse funzioni (loss, funzione di attivazione e metrica) e delle rispettive derivate (se necessario);
- *model_selection*: contiene le funzioni utilizzate per la model selection (vedi paragrafo 2.3)
- *utility*: contiene diverse funzioni di utilità come la train_test_split e i metodi per la scrittura su files;

Il modello di rete neurale utilizzato è contenuto nella classe *NeuralNetwork* definita nel modulo *neural_network*. Offre le diverse funzioni per il calcolo dell'output e dell'aggiornamento dei pesi chiamando le omonime funzioni di ogni livello. Specificando nel livello di output diverse funzioni di attivazione è possibile svolgere sia task di regressione che di classificazione. Nella scrittura del codice abbiamo provato a separare quanto possibile le diverse funzionalità del modulo così da poter rendere maggiormente scalabile il progetto. A tal proposito è possibile migliorare il progetto inserendo nuove classi (ad esempio nuove funzioni di attivazione e di inizializzazione) che estendono la classe base già esistente. Analogo il discorso per gli algoritmi di ottimizzazione.

2.3 Model selection

Per effettuare la model selection è possibile utilizzare la random search o la grid search che utilizzano come tecnica di validazione la K-fold. Entrambe prendono come parametro una funzione per la creazione del modello, un dizionario contenente coppie (nome dell'iperparametro, range di valori), il *monitor_value* che rappresenta il criterio di valutazione, e il valore del parametro K. In questo modo è stato possibile effettuare valutazioni di modelli con diverse architetture ad esempio modificando il numero di livelli o modificando l'ottimizzatore utilizzato. Nel caso in cui venga chiamata la funzione senza specificare l'argomento K, verrà eseguito un semplice hold-out considerando il validation set passato come parametro.
Considerando che ogni esecuzione della random/grid search è indipendente dalle altre, abbiamo implementato una versione parallela utilizzando il modulo *multiprocessing* al fine di poter diminuire il costo computazionale. Per i dataset Monk non abbiamo utilizzato la model selection, data la semplicità del problema (vedi Sez. 3.1). Per quanto riguarda la Cup, abbiamo optato per l'utilizzo del K-fold con $K = 4$ come tecnica di validazione. Nella sezione 3.2 descriviamo in modo più approfondito le scelte adoperate per la risoluzione di questo task.

3 Esperimenti

3.1 Monk

I dataset Monk [7] sono stati molto utili per poter valutare il comportamento del nostro modello e le diverse funzionalità offerte. Considerando la sensibilità all'inizializzazione dei pesi, è stato interessante analizzare il comportamento del modello al variare delle funzioni di inizializzazione dei parametri. Per questo motivo abbiamo utilizzato diverse configurazioni per ogni task del Monk. Per codificare l'input abbiamo utilzzato il one-hot encoding ottenendo un vettore binario di 17 posizioni. Nel caso in cui il livello di output del modello abbia come funzione di attivazione la tangente iperbolica, modifichiamo i targets del dataset al fine di rispettare il range della funzione *tanh*.

Task	**#Nodi, lr, momentum, l2**	**MSE(TR/TS)**	**Accuracy(TR/TS)(%)**
MONK1	4, 0.6, 0.8, 0.0	0.001/0.0014	100/100
MONK2	2, 0.5, 0.8, 0.0	0.011/0.012	100/100
MONK3	3, 0.5, 0.8, 0.0	0.13/0.23	95.90/93.52
MONK3(reg)	3, 0.5, 0.8, 0.01	0.19/0.18	95.08/95.60

Tabella 1: Media della loss e dell'accuracy ottenuti sui dataset Monk.

Per i Monk 2 e 3 abbiamo scelto di utilizzare come funzioni di attivazione la funzione *sigmoid* per il livello nascosto e la *tanh* per quello di output. Invece, per il Monk 1 abbiamo optato per una diversa configurazione. Abbiamo utilizzato la *ReLU* per il livello hidden e la *sigmoid* per il livello di output. Per calcolare l'errore abbiamo utilizzato la Mean Squared Error sull'intero batch e come metrica di valutazione del modello abbiamo usato la funzione di accuracy. Per allenare il modello utilizziamo l'algoritmo Batch Gradient Descent. La tabella 1 riporta i risultati ottenuti su ogni Monk calcolando una media su un totale di 5 esecuzioni. Per quanto riguarda il numero di epoche utilizzate per il training abbiamo deciso di attenerci allo stato dell'arte [8] ottenendo ottimi risultati. Abbiamo utilizzato rispettivamente un numero di epoche pari a 350, 130 e 200. Nella figura 5 abbiamo riportato i grafici della loss e dell'accuracy ottenuti sui dataset Monk. Importante da notare è l'effetto del regolarizzatore applicato al Monk 3 (vedi Fig. 4g) che ha permesso di incrementare il valore dell'accuracy sul test set da 93.52% a 95.60%.

3.2 Cup

Per la competizione Cup abbiamo utilizzato un modello di regressione con una funzione di attivazione lineare per il livello di output.
Prima di iniziare la fase di model selection abbiamo effettuato singole esecuzioni modificando i diversi iperparametri per comprendere quali sarebbero potuti essere i risultati più promettenti. Abbiamo notato che il nostro modello portava a pessimi risultati utilizzando le funzioni di attivazione *ReLU* e *LeakyReLU* nei livelli nascosti. Analogo il discorso per la dimensione della batch. Utilizzando una minibatch con dimensione superiore a 64 non siamo riusciti ad ottenere buoni risultati.
Abbiamo diviso il dataset in training e test interno con una percentuale rispettivamente del 75% e 25%. Come tecnica di validazione abbiamo utilizzato il K-fold cross-validation con un valore di K pari a 4. Dato il grande numero di iperparametri abbiamo deciso di procedere per step. Prima di tutto abbiamo eseguito una random search per comprendere i range degli iperparametri più promettenti. Successivamente abbiamo eseguito un'ulteriore random search finale fissando il numero delle valutazioni a 3500 considerando i range determinati dal passo precedente. La tabella 4 mostra i range degli iperparametri utilizzati nella ricerca. L'esecuzione della model selection ha impiegato 8 ore con un processore 64-core (AWS SageMaker Studio).
Nella tabella 2 abbiamo riportato il valore della MEE sul validation set per i 10 modelli migliori. Il valore riportato è stato ottenuto calcolando la media dei risultati ottenuti sui diversi fold. Dai risultati della model selection emerge che l'ottimizzatore più performante è RMSprop. L'algoritmo SGD non riesce a raggiungere buoni risultati a meno che non si utilizzi la strategia del *linear learning rate decay*. Infatti, come è possibile osservare nella figura 5b, la learning curve ha un forte decremento iniziale dopo di ché tende a stabilizzarsi effettuando piccole oscillazioni. Utilizzando un passo più piccolo durante l'algoritmo di ottimizzazione è stato possibile procedere con un decremento della Mean Squared Error. Alla luce di queste

osservazioni è possibile comprendere il motivo per il quale algoritmi adattivi come RMSprop e Adam riescano ad ottenere risultati più performanti.
Nella figura 1 abbiamo riportato il comportamento di un modello, con iperparametri fissati, variando la funzione di inizializzazione dei pesi. Come è possibile notare, indifferentemente dall'inizializzatore utilizzato, dopo sole 9 epoche il valore della *MEE* converge allo stesso ordine. Per questo abbiamo deciso di utilizzare, a parità di risultato sul validation, una distribuzione normale con media 0 e deviazione standard 0.05 poiché permetteva la determinazione di un grafico molto più regolare almeno nella fase iniziale.
Per quanto riguarda il criterio di arresto, abbiamo considerato una tolleranza assoluta piut-

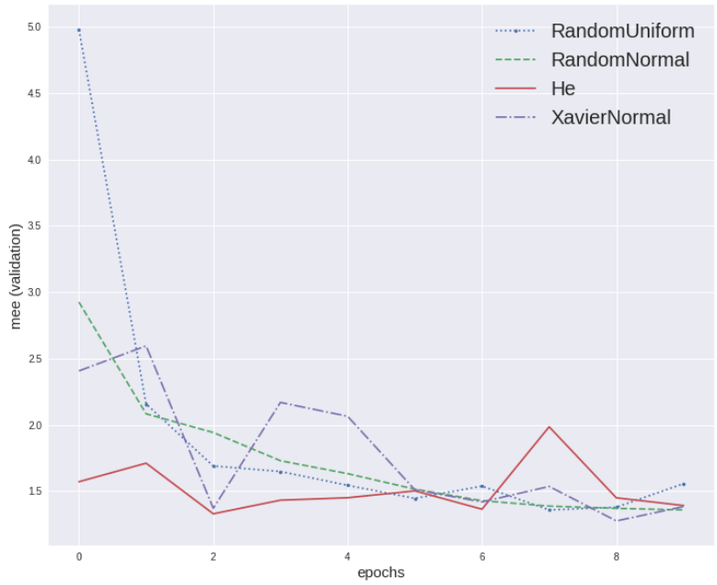


Figura 1: Confronto tra l'utilizzo di diversi inizializzatori con un modello fissato

tosto bassa infatti nessun modello raggiunge tale condizione. I vari criteri di early stopping definiti non hanno condotto a buoni risultati date le oscillazioni riscontrate con l'utilizzo di una minibatch di piccole dimensioni portando ad una terminazione prematura in una condizione di underfitting.
Il modello finale utilizzato per il blind test è il modello 1 definito nella tabella 2. Abbiamo deciso di considerare il modello con il valore della metrica MEE più basso ottenuto tramite la model selection. Inoltre la scelta è stata confermata osservando la regolarità della learning curve riportata nella figura 5b. Nella tabella 3 abbiamo riportato gli iperparametri utilizzati da questo modello, mentre nella tabella 5 vengono mostrati i valori numerici ottenuti su training, validation e test interno. Infine, per avere una conferma visiva della correttezza del lavoro svolto abbiamo utilizzato la Principal Component Analysis per visualizzare il dataset in due dimensioni (vedi figura 2). Dopo aver invocato la funzione *predict* sul modello allenato, abbiamo visualizzato il risultato per comprendere il discostamento tra i risultati reali del test interno e quelli determinati dal nostro modello come è possibile osservare nella figura 3. Nonostante la visualizzazione sia piuttosto approssimata, attraverso questa tecnica è stato possibile ottenere maggiori informazioni sul comportamento del modello.

4 Conclusioni

La Feedforward Neural Network implementata rappresenta un modello scalabile e altamente configurabile che può essere utilizzato per task supervisionati di classificazione e regressione. Ci aspettiamo che il valore della Mean Eucledian Error sul blind test sia circa 1.1 considerando i risultati ottenuti con la model selection e confermati dalla valutazione sul test interno che non era mai stato preso in considerazione. Le predizioni sul blind test sono state salvate nel file *fabamad_ ML-CUP19TS.csv*.

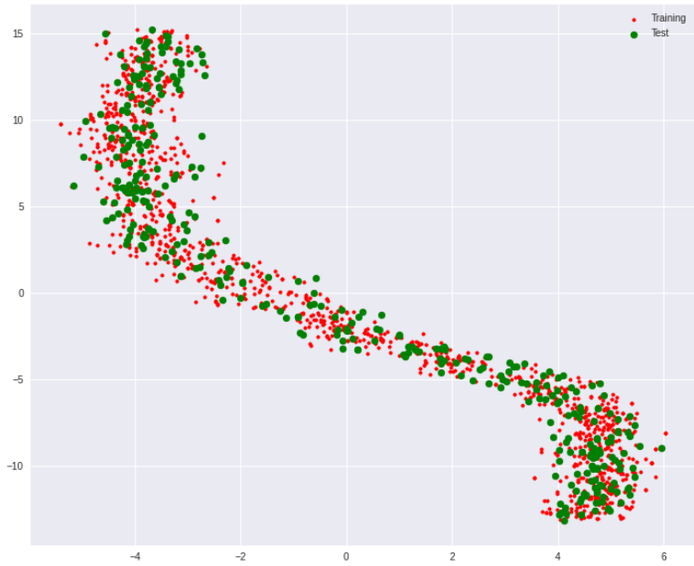


Figura 2: Visualizzazione del dataset Cup proiettato in due dimensioni

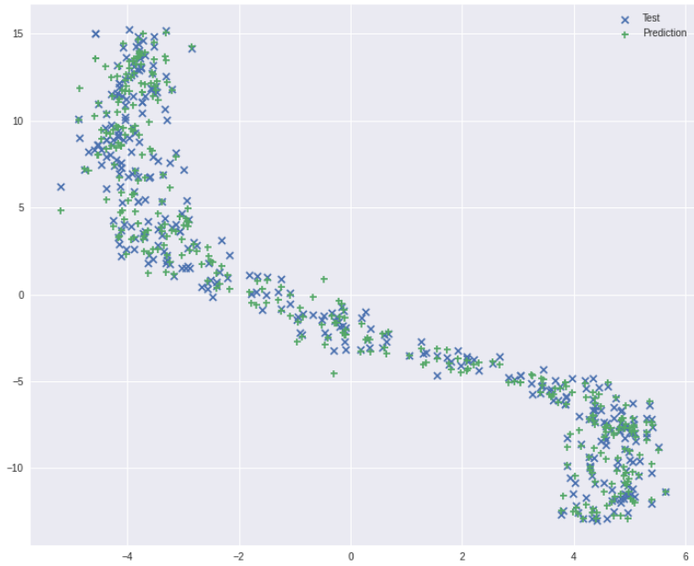


Figura 3: Confronto tra risultati reali e previsti dal modello proiettato in due dimensioni

	Validation set MEE
Modello 1	1,0204052
Modello 2	1,0216020
Modello 3	1,0323365
Modello 4	1,0316442
Modello 5	1,0326774
Modello 6	1,0469049
Modello 7	1,0477150
Modello 8	1,0495128
Modello 9	1,0503326
Modello 10	1,0512871

Tabella 2: I 10 migliori risultati ottenuti dalla fase di model selection sul dataset Cup

Nome iperaparametro	Valore
Ottimizzatore	RMSprop
Learning rate	0.0001
Moving average	0.9
Weight decay	0.0005
Batch size	64
N. livelli hidden	2
N. nodi hidden	[40, 30]
Funzioni di attivazione	[tanh, sigmoid]
Inizializzazione pesi	RandomNormal

Tabella 3: Iperparametri utilizzati dal migliore modello sulla competizione Cup

Nome iperaparametro	Ranges
Ottimizzatore	[RMSprop,Adam,SGD]
Learning rate	[0.01, 0.0001]
Momentum	[0.5, 0.9]
Moving average	[0.9, 0.999]
τ *(lr decay)*	[200, 300, 350]
ϵ_τ *(lr decay)*	[1%, 5%]
Beta 1	[0.9, 0.999]
Beta 2	[0.9, 0.999]
Lambda	[0.0001, 0.0009]
Batch size	[8,16,32,64]
N. livelli hidden	[1,2]
N. nodi hidden	[20,30,40]
N. nodi secondo hidden	[20,30,40]
Funzioni di attivazione	[tanh, sigmoid]
Inizializzazione pesi	[RandomNormal,He,XavierNormal]
Media Mobile	[0.9, 0.99, 0.999]

Tabella 4: Range degli iperparametri utilizzati per la model selection

	MEE training	MEE validation (media)	MEE test (interno)
Modello 1	0.47269	1.0204052	1.01235

Tabella 5: Risultati su training, validation e test interno ottenuti dal miglior modello

Bibliografia

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[3] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2017.

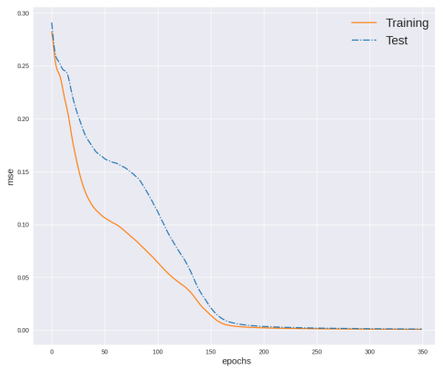
[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[6] Lutz Prechelt. Early stopping - but when? 03 2000.

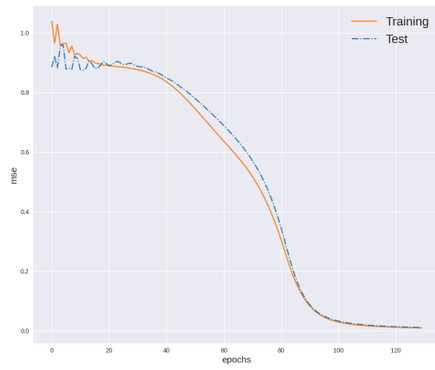
[7] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[8] S. Thrun, Jerzy Bala, Eric Bloedorn, Ivan Bratko, J. Cheng, Kenneth De Jong, Sašo Džeroski, Scott Fahlman, Doug Fisher, R. Hamann, Kenneth Kaufman, S. Keller, I. Kononenko, J. Kreuziger, T. Mitchell, P. Pachowicz, Yoram Reich, Haleh Vafaie, and J. Wnek. The monk's problems a performance comparison of different learning algorithms. 01 1992.



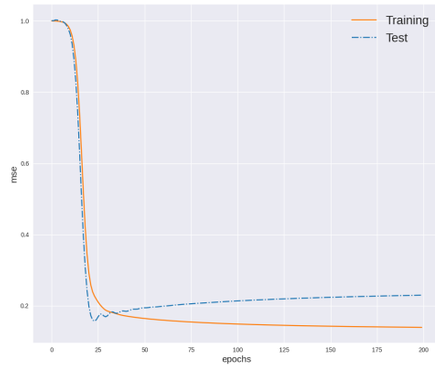
(a) Loss Monk 1

(b) Accuracy Monk 1

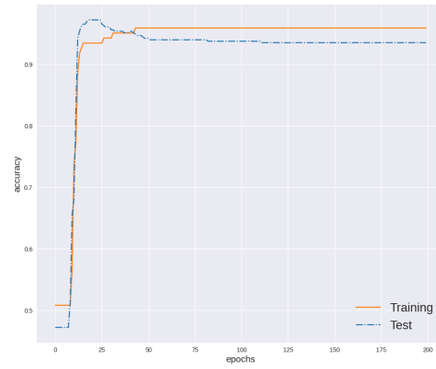


(c) Loss Monk 2

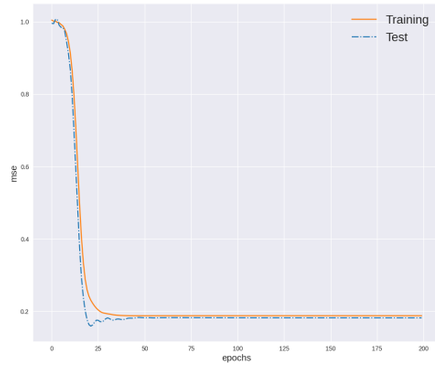
(d) Accuracy Monk 2



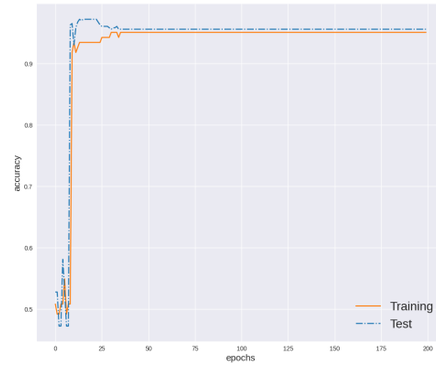
(e) Loss Monk 3



(f) Accuracy Monk 3

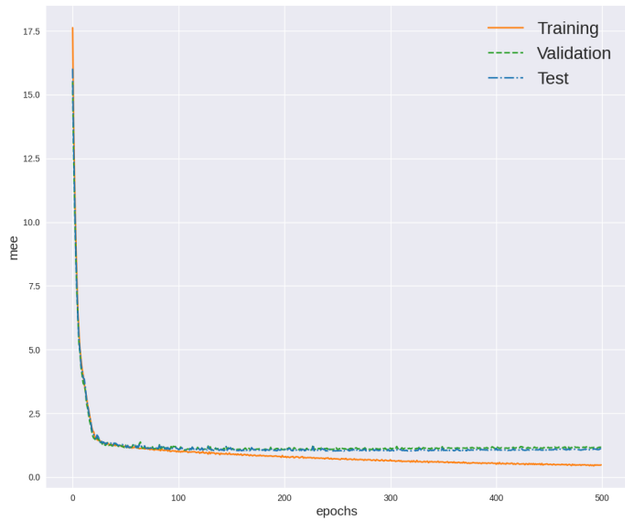


(g) Loss Monk 3 (con regolarizzazione)

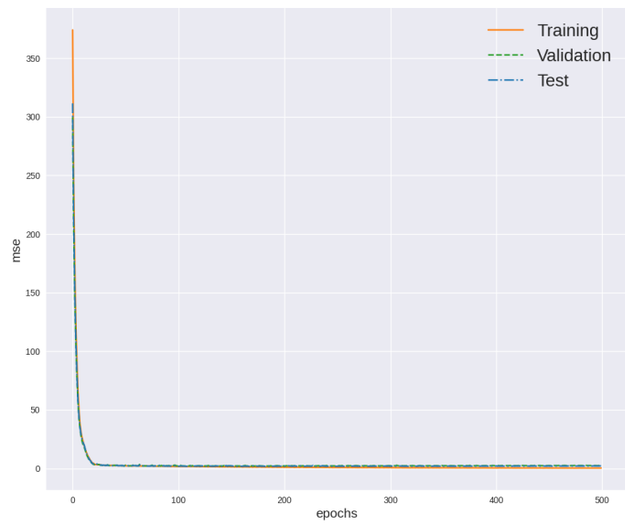


(h) Accuracy Monk 3 (con regolarizzazione)

Figura 4: Risultati sui dataset Monk



(a) MEE



(b) MSE

Figura 5: Learning curve e metrica del miglior modello sulla competizione Cup