Machine Learning Project

Marziano Giuseppe, g.marziano1@studenti.unipi.it
Rocchietti Guido, g.rocchietti@studenti.unipi.it

Machine Learning course (654AA), Academic Year: 2019/2020

Data: *10/07/2020*

Type of project: **A**

Abstract

Abbiamo sviluppato un progetto del tipo A, implementando una rete neurale artificiale in Python 3.7. Abbiamo usato la *grid search* e la *Cross Validation* per la selezione del modello. Infine, abbiamo valutato il nostro modello su un *test set* interno.

1 Introduzione

Lo scopo del seguente progetto è stato quello di creare un Multi Layer Perceptron in grado di effettuare predizioni, individuando una funzione h(x) dallo spazio delle ipotesi H che fosse in grado di approssimare al meglio la funzione target. Per fare ciò abbiamo deciso di implementare un'unica classe *neural_network* che contenesse al proprio interno tutti i metodi necessari per il corretto funzionamento della rete.
Abbiamo implementato la nostra rete MLP in maniera tale che fosse il più adattiva possibile. La nostra rete consente infatti di definire fino a un massimo di 5 layer nascosti di grandezza variabile indicando, in fase di inizializzazione della stessa, il task desiderato. Per quanto concerne il layer di output, la rete riconosce il numero di nodi necessari in maniera automatica. Per la regressione vengono creati tanti nodi quante le colonne di output, per la classificazione invece la rete utilizza un unico nodo in caso di classificazione binaria e tanti nodi quanti i valori unici dell'output in caso di *multi-class classification*.
Per verificare la correttezza della nostra implementazione, abbiamo in primis usato i tre benchmark Monk. Successivamente abbiamo svolto le euristiche necessarie per la selezione del modello migliore addestrato sul CUP dataset.

2 Metodo

2.1 Codice

Abbiamo implementato il nostro codice in linguaggio Python 3.7, suddividendolo in quattro file: **import_dataset.py**, **neural_network.py, cup_main.py e monk_main.py**. L'implementazione della rete è stata effettuata da scratch e le uniche librerie esterne utilizzate sono state le seguenti: *NumPy* e *Pandas* per le strutture dati, *math* per il calcolo numerico, *matplotlib* per la generazione dei plots e *os* per la definizione delle directory.
Di seguito descriviamo il codice implementato e i metodi ivi contenuti:

- Il file **import_dataset.py** contiene all'interno due metodi (*load_cup* e *load_monk*) che hanno la funzione di importare i due dataset con *training* e *test*

- Nel file **neural_network.py** abbiamo implementato la classe *neural_network* contenente al proprio interno tutti i metodi necessari per la creazione e utilizzo del MLP

sia per il task di regressione che per quello di classificazione. La classe è costituita da metodi pubblici e privati, che possono essere raggruppati in sei categorie:

- **Metodi di check**
  - *data_check*: controlla se i dati sono stati correttamente caricati all'interno dei metodi
  - *data_description*: fornisce una descrizione dei dati importati

- **Metodi di import**
  - *load_dataset*: per importare i dati già separati in input e output
  - *load_test*: carica un eventuale *test set* esterno
  - *split_dataset*: divide i dati importati in subset (*development*, *training*, *validation* e *test set*)

- **Metodi di processing**
  - *one_hot_encoding*: prende in input le colonne a cui applicare il one-hot encoding e lo esegue
  - *normalize_internal_data* e *__normalize_data*: normalizzano i dati di input ed eventualmente i dati di output secondo i metodi *minmax*, *z-score* e *simple*
  - *denormalize*: riporta i dati nella scala originale
  - *restore_internal_data*: ripristina i dati al momento dell'import

- **Metodi di inizializzazione**
  - *weight_inizialization*: permette l'inizializzazione dei pesi secondo i metodi *xavier*, *he*, *typeone* e *random*, dando la possibilità di indicare il numero di strati e nodi nascosti (con un massimo di cinque layer)
  - *weight_cleaning*: azzera i pesi e i layer

- **Metodi per l'addestramento e valutazione**
  - *training*: consente di addestrare la rete e prende come argomento gli iperparametri desiderati, ovvero il metodo di training (*online*, *batch e minibatch*), la funzione di attivazione (*sigmoid*, *tanh*, *relu* e *linear*), la funzione di costo (MEE e MSE), la regolarizzazione, il momentum, il numero di epoche, la dimensione del *batch*, con la possibilità di specificare se utilizzare o meno il metodo di *early_stopping*. All'interno del metodo *training* vengono richiamati i metodi privati necessari per l'addestramento *(__forward*, *__backpropagation* e *__update_weights*). Inoltre, all'interno del metodo *training*, vengono richiamati i metodi per la valutazione dell'addestramento appena compiuto (*evaluate* e *accuracy*)

- **Metodi per la Model Selection**
  - *grid_search*: prende in input i vettori contenenti gli iperparametri selezionati e restituisce i risultati (errore sul *validation set*, errore sul *training set*, numero di unità e layer nascosti, *learning rate*, regolarizzazione, momentum e numero di epoche), dal migliore al peggiore in base all'errore sul *validation set*
  - *cross_validation*: permette di effettuare il training definendo il numero di *fold* desiderati

- **Metodi di predizione**
  - *prediction*: permette di effettuare una predizione utilizzando l'ultimo modello addestrato con la possibilità di normalizzare i nuovi dati in

input con le stesse statistiche utilizzate per il dataset originale, nonché di de-normalizzare i dati in output in caso di addestramento del modello con i dati in output originali normalizzati

- Nei file **cup_main.py** e **monk_main.py** vengono creati gli oggetti di tipo *neural_network* rispettivamente per il dataset CUP e i dataset Monk. Nel file **cup_main.py** viene eseguita la *grid search* (divisa in *coarse* e *finer*), la *Cross Validation*, il *final training* e il plot dei risultati finali. Nel file **monk_main.py** si esegue una piccola *grid search*, l'addestramento sui dataset Monk 1, Monk 2 e Monk 3 e il plot dei risultati finali.

2.2 Preprocessing

2.2.1 Monk

I dataset Monk sono costituiti da sei colonne di feature di tipo categoriale. Abbiamo dunque deciso di applicare il *1-of-k schema*, ottenendo in totale 17 colonne di feature con valori binari.

2.2.2 CUP

Per quel che riguarda il CUP abbiamo normalizzato i dati secondo il metodo *minmax*: la normalizzazione è stata effettuata sull'intero dataset originale relativamente all'input. Per quel che concerne i dati di output, considerando che l'output della rete è lineare (si tratta di regressione), la rete dovrebbe riportare i risultati nella giusta scala in maniera automatica: abbiamo dunque deciso di non normalizzarli. Per quanto concerne la predizione sul *blind test set*, abbiamo normalizzato i dati utilizzando minimo e massimo relativi al dataset originale.

2.3 Schema di Validazione

2.3.1 Monk

I dataset Monk, avendo a disposizione sia il *training set* che il *test set*, sono stati importati entrambi e, a partire dal *training set*, abbiamo creato un *validation set* corrispondente a un terzo del dataset disponibile. Nella fase di *final training* l'algoritmo ottiene automaticamente il *development set* come unione di *training set* e *validation set*.

2.3.1 CUP

Per quanto riguarda il dataset CUP, in primo luogo abbiamo creato un *internal test set* corrispondente al 25% del dataset. A partire dal *development set* (75% del dataset originale) è stato creato un *validation set* (33% del *development set*) e un *training set* (66% del *development set*). Nella fase di *final training* del modello abbiamo creato, a partire dal *development set*, un *mini-validation set* corrispondente al 10% del *development set* e utilizzato per l'*early stopping*. Durante la fase di ricerca degli iperparametri ottimali abbiamo effettuato la *grid search* (*coarse* e *finer*) con la quale abbiamo ottenuto tre insiemi di iperparametri tra i quali abbiamo selezionato il migliore utilizzando la *Cross Validation*. Successivamente, individuati gli iperparametri migliori, abbiamo addestrato la rete sul *development set* e calcolato i risultati sull'*internal test set*.

3 Esperimenti

3.1 Risultati Monk

Nella Tabella 1 presentiamo i valori degli iperparametri e i risultati per i tre dataset Monk. Per tutti e tre i dataset abbiamo utilizzato una rete neurale con 17 nodi in input e un nodo in output, la funzione *sigmoid* come funzione di attivazione per hidden e output layer e il *batch gradient descent* come metodo di addestramento. Gli iperparametri scelti sono stati selezionati in seguito a delle piccole *grid search*.

Dataset	Unità	η	α	λ	MSE (*train*/*test*)	Accuracy (*train*/*test*)
Monk 1	4	10	0.9	0	0.0002/0.0013	100% / 100%
Monk 2	3	10	0.9	0	3.91e-05 / 6.21e-05	100% / 100%
Monk 3 (noReg)	3	10	0.9	0	0.0017 / 0.021	100% / 94.67%
Monk 3 (Reg)	3	10	0.9	0.001	0.0084 / 0.016	100% / 96.06%

Tabella 1: Risultati ottenuti sui dataset Monk

Di seguito, riportiamo i grafici dei risultati relativi all'MSE e Accuracy ottenuti sui dataset Monk.

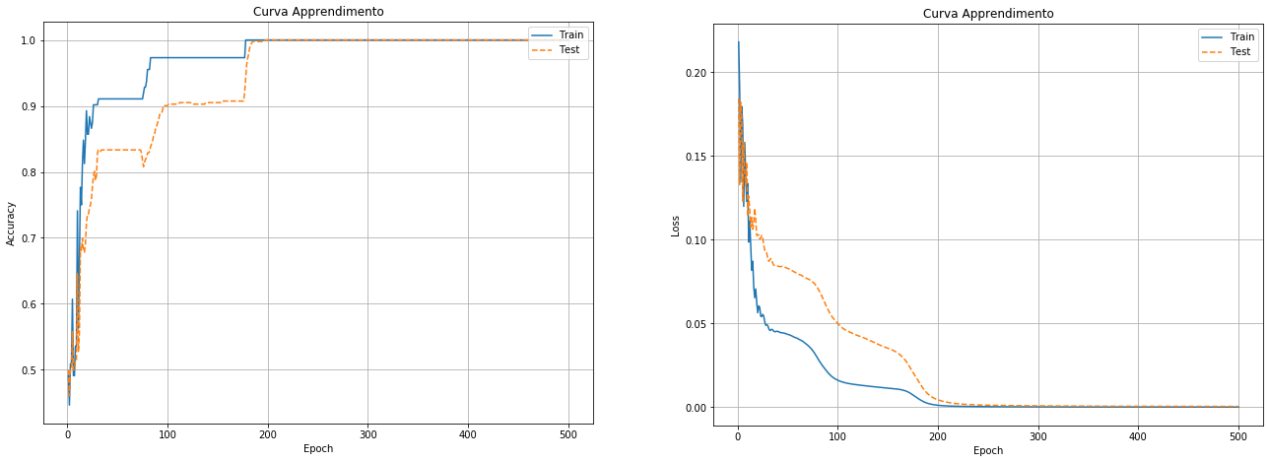

Figura 1: Risultati sul dataset Monk 1. A sinistra l'Accuracy e a destra la MSE.

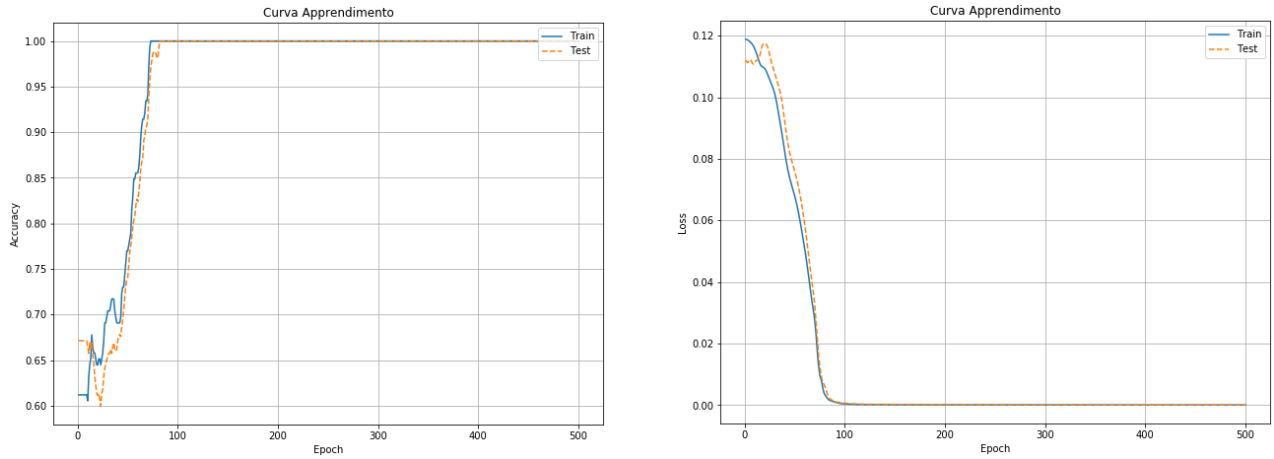

Figura 2: Risultati sul dataset Monk 2. A sinistra l'Accuracy e a destra la MSE.

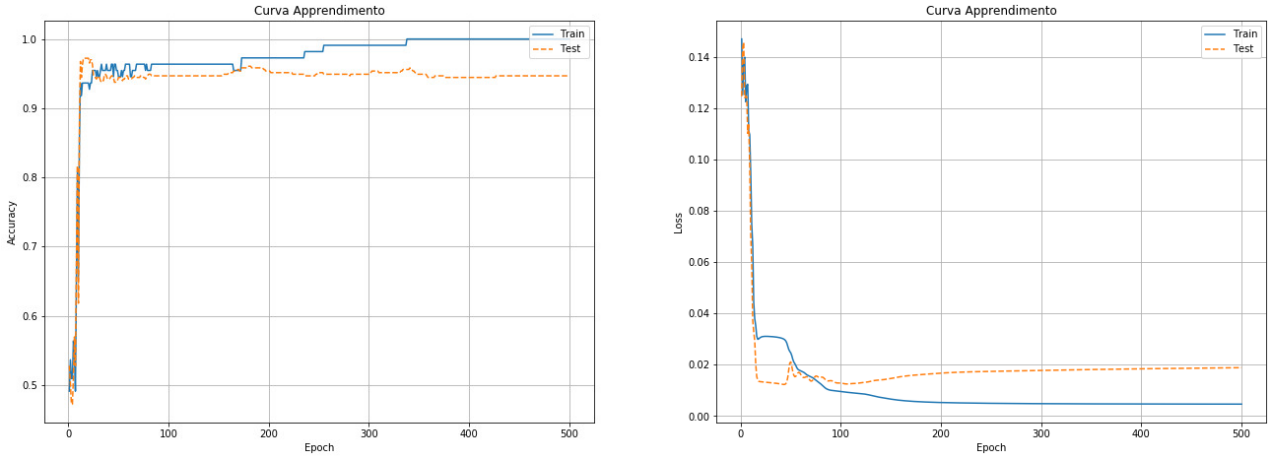

Figura 3: Risultati sul dataset Monk 3 (noReg). A sinistra l'Accuracy e a destra la MSE.

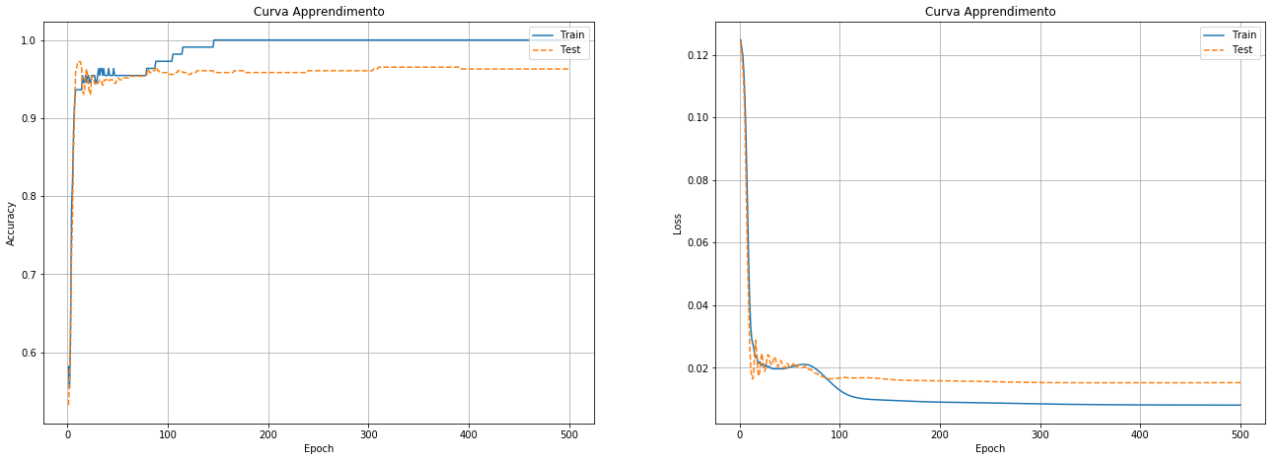

Figura 4: Risultati sul dataset Monk 3 (Reg). A sinistra l'Accuracy e a destra la MSE.

3.2 Risultati CUP

In questa sezione prendiamo in considerazione le fasi di model selection e model evaluation per il dataset CUP. Come modello abbiamo utilizzato una rete neurale con 20 neuroni di input e due neuroni di output. In primo luogo, per la fase di model selection, abbiamo definito quali iperparametri utilizzare: numero di epoche di addestramento, metodo di apprendimento (*batch*, *minibatch* e *online*) metodo di inizializzazione dei pesi (*xavier* e valori *random* compresi tra -0.7 e 0.7), funzione di attivazione (*tanh* e *sigmoid*), numero di neuroni e layer nascosti, *learning rate*, momentum e termine di regolarizzazione. Per quel che concerne le epoche di addestramento abbiamo stabilito un numero fisso di 2000 epoche di partenza. Come menzionato nel paragrafo 2.1, abbiamo implementato una tecnica di *early stopping* che ci ha consentito di prevenire l'overfitting. In base alla metodologia di apprendimento abbiamo determinato il numero di epoche di *patience* mediante un approccio empirico basato sull'osservazione dei plot delle curve di apprendimento: per il metodo *batch* abbiamo selezionato 20 epoche di *patience*, per il *minibatch* 75 e per l'*online* 50. Come da letteratura, abbiamo implementato la tecnica di *early stopping* basandola sulla *validation loss*: se 20/75/50 epoche prima la *validation loss* era più bassa, l'addestramento viene interrotto, lo stesso succede se il decremento dell'errore è inferiore a 0.01 per tante

epoche quanto è il valore della *patience*. In seguito, per determinare i valori ottimali per i rimanenti iperparametri, abbiamo adottato la seguente strategia:

- individuate il miglior insieme di iperparametri usando la *grid search*,
- usare la *Cross Validation* per valutare i modelli selezionati dalla *grid search*,
- addestrare il modello migliore sull'intero *development set* e infine valutare il modello sull'*internal test set*.

3.2.1 Grid-search

Abbiamo usato la *grid search* per stabilire il migliore insieme di iperparametri: il metodo implementato restituisce i risultate in base alla MEE sul *validation set* in modo decrescente. Innanzitutto, abbiamo provato a lanciare la *grid search* per tutti gli iperparametri ma questo approccio non ci consentiva di tenere sotto controllo il quadro generale. Per cui abbiamo deciso di effettuare prima una ricerca a "*grana grossa*" su alcuni iperparametri, e successivamente una ricerca a "*grana fine*" sugli iperparametri restanti. Quindi per la *coarse grid search* abbiamo prima di tutto individuato i risultati migliori relativi al metodo di addestramento, inizializzazione dei pesi, funzione di attivazione, numero di neuroni/layer nascosti e *learning rate*. In seguito, per la *finer grid search*, abbiamo preso in considerazione il momentum e il termine di regolarizzazione.
La prima *coarse grid search* è stata effettuata su un range di valori parametrici molto vasto. La Tabella 2 riporta i valori degli iperparametri utilizzati. I risultati della *grid search* sono mostrati nella Tabella 3: per ogni metodo di addestramento riportiamo i tre migliori risultati.

Metodo	**Unità/Layer nascosti**	**η**	**Funzione di attivazione**	**Inizializzazione pesi**
online, *batch*, *minibatch*	[2, 4, 8, 11, 16, 32, [2,5],[2,3,2]]	[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]	*sigmoid*, *tanh*	*xavier*, *random* (-0.7, 0.7)

Tabella 2: Valori degli iperparametri utilizzati nella prima grid search

Metodo	**Risultato**	**Inizializzazione pesi**	**Funzione di attivazione**	**Numero di neuroni/layer**	**η**	**MEE**
online	1°	*random*	*sigmoid*	16	0.01	1.2589
online	2°	*random*	*sigmoid*	8	0.01	1.2748
online	3°	*random*	*sigmoid*	11	0.01	1.2749
batch	1°	*xavier*	*sigmoid*	11	0.1	1.3398
batch	2°	*random*	*sigmoid*	11	0.1	1.3541
batch	3°	*random*	*sigmoid*	8	0.1	1.3559
minibatch	1°	*random*	*sigmoid*	11	0.1	1.3804
minibatch	2°	*random*	*tanh*	16	0.1	1.3884
minibatch	3°	*xavier*	*sigmoid*	8	0.1	1.4006

Tabella 3: Risultati della prima grid search

Analizziamo dunque i risultati. In primo luogo, prendiamo in considerazione il numero di unità/layer nascosti: come è possibile notare dalla Tabella 3, i risultati migliori sono stati ottenuti su reti con un solo layer nascosto, con un numero di neuroni da 8 a 16. Le reti con

due layer (il primo con 2 e il secondo con 5 neuroni) e con tre layer (2, 3 ,5 neuroni) hanno ottenuto risultati peggiori, motivo per cui non sono stati riportati.
In secondo luogo, per quel che concerne il metodo di apprendimento, osserviamo che i risultati migliori sono stati ottenuti con il metodo *online*: benché con i metodi *batch* e *minibatch* i risultati siano stati buoni, abbiamo deciso di effettuare una seconda *coarse grid search* solamente con il metodo *online*. La Tabella 4 riporta i valori degli iperparametri utilizzati. Nella Tabella 5 vengono presentati i tre migliori risultati.

Metodo	**Unità nascoste**	**η**	**Funzione di attivazione**	**Inizializzazione pesi**
online	[9, 10, 11, 12, 13, 14, 15, 16, 17, 18]	[0.005, 0.0075, 0.01, 0.025, 0.05]	*sigmoid*	*random* (-0.7, 0.7)

Tabella 4: Valori degli iperparametri utilizzati nella seconda grid search

Numero di neuroni	**η**	**MEE**
10	0.01	1.2582
16	0.01	1.2624
14	0.01	1.2628

Tabella 5: Risultati seconda grid search

Come è possibile osservare nella Tabella 4, abbiamo scelto il numero di unità nascoste e il range del *learning rate* basandoci sui tre migliori risultati ottenuti dalla prima *grid search*: abbiamo dunque preso in considerazione, per ogni valore, un'intorno sufficientemente grande. Analizzando la Tabella 5 notiamo inoltre che i risultati migliori vengono ottenuti con lo stesso *learning rate* utilizzato nella prima *grid search*.
A questo punto abbiamo effettuato una *finer grid search* nella quale abbiamo aggiunto il momentum e il termine di regolarizzazione. La Tabella 6 riporta i valori degli iperparametri utilizzati. Nella Tabella 7 vengono presentati i migliori risultati.

Metodo	**Unità nascoste**	**Funzione di attivazione**	**Inizializzazione pesi**	**α**	**λ**	**η**
online	[10, 14, 16]	*sigmoid*	*Random* in [-0.7, 0.7]	[0, 0.2, 0.4, 0.6, 0.8, 0.9]	[0, 0.001, 0.01, 0.1, 1, 10]	0.01

Tabella 6: : Valori degli iperparametri utilizzati nella terza grid search

Numero di Neuroni	**α**	**λ**	**MEE**
16	0.4	0	1.2290
14	0.4	0	1.2584
10	0.2	0	1.2701

Tabella 7: Risultati terza grid search

Come si osserva in Tabella 7, effettuando l'ultima *grid search*, abbiamo ottenuto dei valori compresi tra 0.2 e 0.4 di momentum, mentre il termine di regolarizzazione rimane a zero.

Di seguito mostriamo come abbiamo scelto, tra i tre modelli appena riportati, il modello migliore, usando la *Cross Validation*.

3.2.2 Cross Validation

Durante le fasi precedenti abbiamo dunque selezionato i tre modelli migliori riportati di seguito:

- **Modello 1**: metodo *online*, 16 neuroni nascosti, *sigmoid* come funzione di attivazione, inizializzazione *random* dei pesi, *learning rate* di 0.01, *momentum* di 0.4 e termine di regolarizzazione pari a 0.
- **Modello 2**: metodo *online*, 14 neuroni nascosti, *sigmoid* come funzione di attivazione, inizializzazione *random* dei pesi, *learning rate* di 0.01, *momentum* di 0.4 e termine di regolarizzazione pari a 0.
- **Modello 3**: metodo *online*, 10 neuroni nascosti, *sigmoid* come funzione di attivazione, inizializzazione *random* dei pesi, *learning rate* di 0.01, *momentum* di 0.2 e termine di regolarizzazione pari a 0.

Arrivati a questo punto, per la selezione del modello migliore, abbiamo optato per l'utilizzo della *Cross Validation* sui tre migliori risultati, optando per un numero di *fold* pari a 5.
I risultati ottenuti sono riportati di seguito.

	Media MEE
Modello 1	1.2717
Modello 2	**1.2226**
Modello 3	1.2731

Tabella 8: Risultati della Cross Validation sui tre modelli migliori

Come possibile osservare nella Tabella 8, riportiamo per ogni modello la media delle MEE. In questo modo è stato possibile individuare il modello migliore corrispondente al Modello 2. Una volta scelto il modello più adeguato, abbiamo proceduto con il *final training*.

3.2.3 Internal test set

Il modello scelto per la fase finale è stato dunque un modello con **14** neuroni nascosti, la ***sigmoid*** come funzione di attivazione, *learning rate* pari a **0.01**, momentum di **0.4** e termine di regolarizzazione uguale a **0** con un'inizializzazione dei pesi ***random***.
Abbiamo dunque effettuato un nuovo addestramento sull'intero *development set* e l'abbiamo poi valutato sull'*internal test set*. Nella Tabella 9 vengono riportati i risultati di tale addestramento. Come indicato nel paragrafo 2.3.1, per il metodo di *early stopping*, abbiamo utilizzato un dataset corrispondente al 10% del *development set* come *validation set*.

Modello 2	**MEE**
Training	1.1171
Cross Validation	1.2226
Test	1.1429

Tabella 9: Risultati del modello finale

Infine, riportiamo anche il plot della curva di apprendimento, sul *training* e *test set* del nostro modello finale.

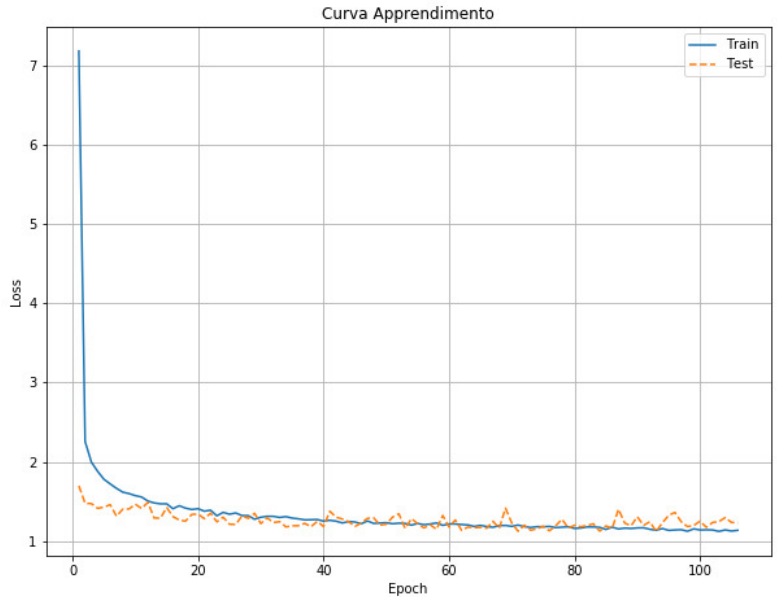

Figura 5: Curva di apprendimento del modello finale

3.2.4 Hardware e tempistiche

Il costo computazionale per la scelta del modello è dipeso principalmente dal numero di neuroni/layer e dalla metodologia di apprendimento: la modalità *batch* è risultata quella computazionalmente meno costosa mentre la modalità *online*, pur convergendo in un numero inferiore di epoche rispetto alle altre modalità, è risultata essere la più costosa. Per quanto riguarda il *minibatch*, il costo computazionale è fortemente dipendente dal numero dei *batch* scelti.
Per la compilazione ed esecuzione del codice è stato usato un processore *Intel i5-8300H* da 2,30 GHz di frequenza di base con 8Gb di RAM. Per l'esecuzione di tutte le fasi elencate nel report, dunque tutte le *grid search*, la *Cross Validation* e i vari addestramenti, il tempo richiesto è stato di circa 120 minuti, per quanto riguarda l'addestramento finale il tempo richiesto è invece di 9,8 secondi.

4 Conclusioni

Per quanto riguarda il dataset Monk abbiamo ottenuto dei risultati ottimali, in linea con la letteratura sul tema. Per quanto invece concerne il dataset CUP, le difficoltà maggiori sono state riscontrate nella ricerca degli iperparametri ottimali e, soprattutto, nella scelta del metodo di addestramento. Abbiamo infatti svolto delle prove con sia con il metodo *batch* che con il *minibatch* (*coarse* e *finer grid search*, *Cross Validation*, *final training*) ottenendo comunque dei buoni risultati, seppur inferiori a quelli ottenuto con il metodo *online*.
In merito allo sviluppo del codice, abbiamo optato per una soluzione unitaria, creando un'unica classe che contenesse al proprio interno tutti i metodi necessari per il corretto funzionamento, cercando di rendere l'algoritmo più adattivo possibile.
Per la predizione sul *blind test set*, abbiamo normalizzato i dati in input utilizzando le stesse statistiche del dataset originale (min e max) e proceduto con la predizione dei risultati di output.

I risultati della predizione si trovano nel file *PianoG_ML-CUP19-TS.csv* (Il nick del team è *PianoG*)

We agree to the disclosure and publication of my name, and of the results with preliminary and final ranking.