ML: Report del progetto

Nicolas Manini[1]
Gabriele Pisciotta[2]

November 15, 2021

[1] Matricola 530705; manini.nicolas@gmail.com
[2] Matricola 596217; ga.pisciotta@gmail.com

Abstract. Progetto tipo A (no CM), implementazione python di una neural network di tipo fully-connected, con riguardo alla modularità e alla possibilità di aggiungere nuove funzioni per attivazione, regolarizzazione, loss e momentum; implementazione di k-fold e grid search per la validazione.

1 Introduzione

L'obbiettivo del progetto è implementare una rete neurale fully connected (multi-layer-perceptron) con back-propagation, per interfacciarsi con le problematiche relative all'implementazione degli algoritmi visti a lezione. Abbiamo cercato di ottenere un implementazione modulare e facilmente estendibile per quanto riguarda la scelta delle funzioni (attivazione, regolarizzazione, loss, momentum). Al fine della validation abbiamo implementato moduli per eseguire k-fold (introducendo anche la possibilità di parallelizzare le computazioni) e grid search dei parametri del modello.

2 Metodo

L'implementazione è stata fatta in python v.3 (con librerie *pandas*, *numpy*, *matplotlib* e *sklearn* (per lo shuffling dei record)) e supporta:

- Funzioni di attivazione:
  - Sigmoid
  - Linear
  - Softplus
- Metodi di regolarizzazione:
  - Tikhonov regularization (L2)
- Funzioni loss:
  - Squareloss

- Momentum:
  - Standard momentum (visto a lezione)
  - Momentum con moving average (per minibatch/online)
- Inizializzazione dei pesi
  - Random con distribuzione uniforme
  - Xavier initialization

2.1 Struttura del codice

Il codice è suddiviso in più file:

NeuralNetwork: Contiene l'implementazione della classe neural network, permette di istanziare il modello attraverso i parametri, effettuare training e predizioni.

TrainingAlgorithm: Permette di istanziare un modulo per l'esecuzione di uno specifico algoritmo di learning (nel nostro caso minibatch, batch o online come casi limite di minibatch).

Layer: Implementazione di un singolo layer della rete, istanziato sulla base del tipo di livello (interno, output) e che permette di aggiornare i pesi tramite una iterazione di back-propagation.

WeightUpdater: Permette di istanziare moduli contenenti la logica per la computazione dell'aggiornamento dei pesi, nel nostro caso *CompoundWeightUpdater* permette di includere termini di regolarizzazione e momentum (rispettivamente determinati da sotto-moduli) ognuno pesato per un parametro separato.

RegularizationFunctions: Permette di istanziare moduli che computano funzioni di regolarizzazione.

LossFunctions: Permette di istanziare moduli che computano funzioni loss.

ActivationFunctions: Permette di istanziare moduli che computano funzioni di attivazione.

Metric: Permette di istanziare moduli per la computazione di metriche al fine di testare i risultati di predizione (MEE, Accuracy).

KFoldCrossValidation: Modulo che implementa i tool per effettuare k-fold cross validation, con parametro k.

GridSearch: Modulo che permette di eseguire una grid search completa date liste di valori per ogni parametro della rete neurale

Utilities Contiene definizioni di funzioni di utilità, per il preprocessing (One of K encoding, rescaling degli output binari) e per la lettura dei dati.

Il programma supporta minibatch/batch/online learning (batch e online sono considerati casi limite di minibatch) con learning rate decrementale (in due varianti: come da slide decrementato per le prime $\tau = 200$ iterazioni fino ad un learning rate pari all'1% del valore iniziale; oppure decrementato linearmente all'infinito) e implementiamo come stop conditions controlli su quando il valore di loss computato su train e validation set scende sotto ad una soglia, oltre a controlli sull'errore relativo sul training set. Abbiamo implementato una strategia di restart del training che seleziona il migliore modello prendendo quello che converge ad un minore errore sul training set, al fine di escludere modelli convergenti a minimi locali.

Abbiamo effettuato la validazione attraverso una grid search con k-fold; per individuare i range di valori per la grid search abbiamo effettuato controlli preliminari a campione per verificare combinazioni di parametri che permettessero convergenza, e cercato di individuare i valori limite, procedendo prima con grid search più rade e poi perfezionare la ricerca con altre grid search progressivamente più fitte nell'intorno dei parametri ottimali.

3 Esperimenti

3.1 MONK Datasets

Per questi dataset abbiamo effettuato alcuni step di preprocessing:

- 1-of-k encoding delle variabili, ottenendo 17 unità di input
- Rescaling dei valori per la variabile target ($0 \to 0.1$; $1 \to 0.9$)

Per poi procedere con un 3-fold ed effettuando una grid search separata per batch e minibatch (testando per le dimensioni dei minibatch $mb \in \{5, 10, 20\}$), osservando migliori prestazioni con l'algoritmo minibatch abbiamo deciso di mantenerlo, con $mb = 5$ per MONK1 e $mb = 20$ per gli altri.

In tabella 1 sono riportati i valori di MSE e Accuracy (TR/TS) con un solo hidden layer e per le configurazioni di parametri individuate (η learning rate, λ parametro della regularization (L2), α parametro del

momentum, β peso della moving average per il momentum). I learning rate utilizzati per il training sono quelli mostrati in tabella moltiplicati per $\sqrt{mb/l}$, questa scelta è stata fatta per cercare di rendere più facile il confronto avendo i valori su scala simile al variare della dimensione dei minibatch.

Table 1. Risultati medi per la predizione dei task MONK su 30 restart.

Task	**#Unità**	η	λ	α	β	**MSE**	**Accuracy**
MONK 1	4	4.5	0	0.8	0.8	(0.0011/0.0069)	(100%/100%)
MONK 2	4	0.3	0.0001	0.9	0.7	(0.0012/0.0096)	(100%/100%)
MONK 3	5	0.5	0.00001	0.9	0.7	(0.0033/0.0098)	(99.67%/99.67%)

Grafici Riportiamo in tabella 2 i grafici di MSE e Accuracy ottenuti con i parametri riportati in tabella 1.

3.2 CUP Dataset

Abbiamo innanzitutto estratto l'internal test set dai dati, eseguendo uno shuffling dei dati e prendendo 80% dei dati come design set, e il rimanente 20% come test set; dal design set abbiamo estratto un altro 80% come training set lasciando un 20% del design set come validation set.

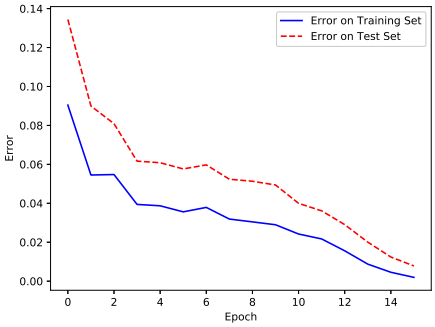
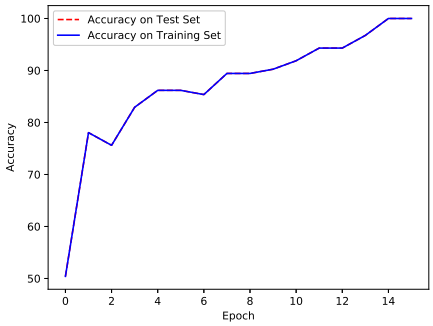
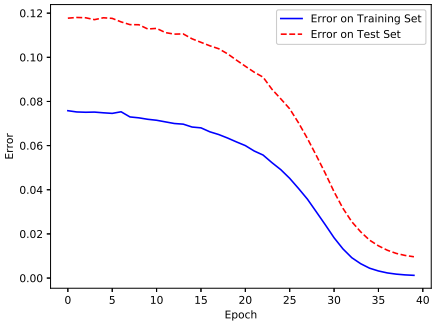
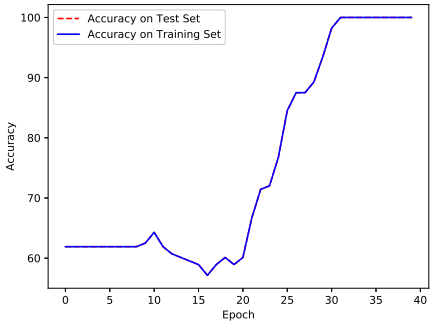
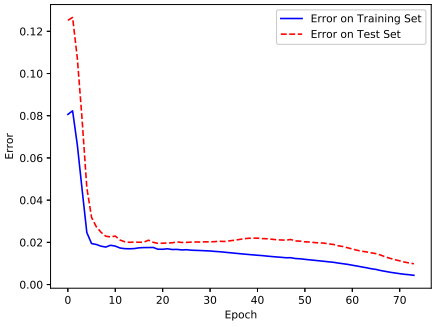
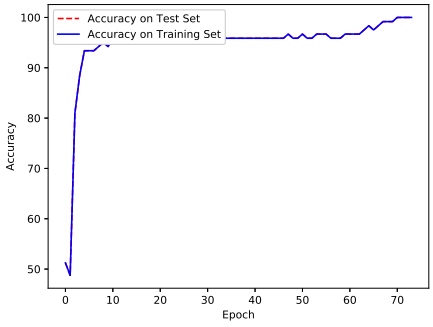
Inizialmente abbiamo effettuato dei test preliminari per esplorare la convergenza del modello al variare dei parametri, individuando alcuni valori per cui il modello converge, su cui definire poi i parametri per la grid search. Abbiamo quindi eseguito una grid search con 10-fold sul design set per i seguenti parametri:

- MiniBatch Size: 20, 60, 100, 140
- Hidden Layers Count: 5, 10, 20
- Units per Hidden Layer: 10, 20, 30
- Learning Rate: 0.0015, 0.001, 0.0005
- Regularization: 0, 0.00001, 0.0001
- Momentum Alfa: 0, 0.5, 0.9
- Momentum Beta: 0.2, 0.5, 0.8

E successivamente affinato la ricerca con:

- MiniBatch Size: 100
- Hidden Layers Count: 1, 3, 5

Table 2. Grafici per i tre monk benchmark, training set in linea blu continua, test set in tratteggio rosso; i grafici dell'accuracy risultano sovrapposti.

MSE	**Accuracy**
MONK 1	MONK 1
	
MONK 2	MONK 2
	
MONK 3	MONK 3
	

– Units per Hidden Layer: 10, 20, 30
– Learning Rate: 0.0015, 0.001, 0.0005
– Regularization: 0, 0.00001, 0.0001
– Momentum Alfa: 0.5, 0.7, 0.9
– Momentum Beta: 0.4, 0.6, 0.8

Si riportano in tabella 3 alcuni risultati (i primi tre sono i migliori forniti dalla grid search, gli altri sono stati scelti manualmente tra i primi che presentano variazioni nei valori dei parametri, al fine di mostrare l'impatto che tali cambiamenti hanno sulle prestazioni). Abbiamo deciso di considerare come modello finale quello della prima riga, in quanto generava un grafico smooth e presenta un MEE sul validation set (estratto come spiegato all'inizio di sez.3.2) minore dei successivi, oltre al fatto che la sua learning curve risulta più smooth di altri primi modelli, ad esempio del modello della quarta riga.

Table 3. Alcune tra le configurazioni più performanti (per media del valore di MSE nella 10-fold) ottenute tramite grid search, con regularization L2 e $mb = 100$, con inizializzazione dei pesi tramite Xavier initialization (valori medi su 5 esecuzioni).

N°	**Layers**	**#Unità**	η	λ	α	β	**MEE (TR/VL/TS)**
1	3	30	0.001	0	0.9	0.8	(0.9361/1.0160/1.0168)
2	3	30	0.0015	0	0.9	0.8	(0.9444/1.0451/1.0238)
3	5	30	0.001	0	0.9	0.4	(0.9486/1.0334/1.0417)
...	...	...	...	...	...	...	...
4	3	30	0.0015	0	0.7	0.6	(0.9495/1.0298/1.0180)
5	3	30	0.0005	0	0.9	0.8	(0.9539/1.0215/1.0202)
...	...	...	...	...	...	...	...
6	3	30	0.0015	0	0.5	0.6	(0.9656/1.0435/1.0685)
...	...	...	...	...	...	...	...
7	3	20	0.0015	0,00001	0.7	0.6	(1.0291/1.0818/1.0819)
8	1	30	0.0015	0,00001	0.5	0.6	(1.1407/1.1766/1.1808)
...	...	...	...	...	...	...	...
9	3	10	0.0005	0	0.5	0.4	(1.0746/1.0992/1.0994)
...	...	...	...	...	...	...	...
10	1	30	0.0015	0,0001	0.9	0.8	(1.4993/1.4403/1.4379)

Abbiamo osservato che i risultati migliori si ottengono per i valori più alti di alfa, e che in generale valori alti di lambda comportano un deterioramento delle prestazioni; i test effettuati con un solo layer sono risultati poco prestanti: le prestazioni sono risultate migliori in genere per modelli con 3 layer (per le quantità di nodi in ogni layer che abbiamo testato) e in generale tra i primi risultati della grid search non ci sono molte con-

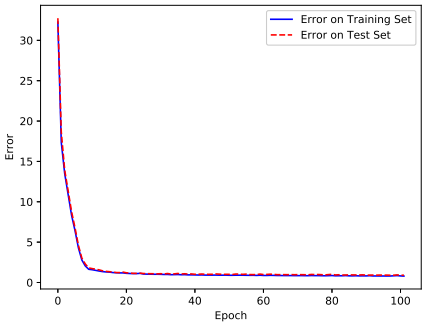
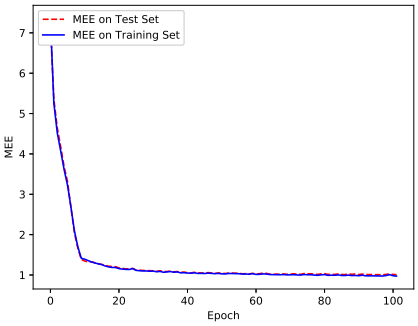
figurazioni con 5 layer. Abbiamo inoltre notato che le configurazioni con 10 unità per livello si collocano tra le peggiori.

Sperimentazioni aggiuntive Abbiamo provato a cambiare alcuni moduli scelti nell'algoritmo testandone gli effetti sulla configurazione di parametri migliore per la grid search (la prima riga in tabella 3):

- Usando l'inizializzazione casuale dei pesi in un range fisso (al posto di Xavier initialization), non notiamo particolari differenze nel grafico, ma prestazioni leggermente inferiori (TR/VL/TS): (0.9417/1.0340/1.0361).
- Cambiando la modalità di decremento del learning rate non notiamo differenze sostanziali nel grafico, prestazioni leggermente inferiori arrestando il decremento dopo 200 iterazioni: (0.9452/1.0194/1.0196).

Modello finale Il modello finale è quindi quello riportato nella prima riga di tabella 3, riportiamo in tabella 4 i grafici degli andamenti della learning curve su MSE e del MEE per tale modello.

Table 4. Grafico della learning curve (MSE) e di MEE per il modello finale; training set in linea blu continua, test set in tratteggio rosso.

MSE	MEE
	

3.3 Discussione

Nello svolgere il progetto abbiamo osservato una facilità nel trovare parametri adatti ad effettuare una classificazione per i primi due MONK, come anticipato dal professore, mentre per il terzo benchmark dei MONK abbiamo notato un incremento nella difficoltà del task, immaginiamo

che possa essere dovuto alla presenza di rumore nei dati, in quanto è facile trovare parametri che portano a prestazioni quasi ottimali. Per quanto riguarda il dataset della cup abbiamo notato un netto incremento nei tempi di esecuzione del training rispetto ai MONK, dovuto sia alla grandezza del dataset che alla topologia della rete. Abbiamo inoltre notato che l'esecuzione di online training sui MONK risulta piu lenta (in termini di tempo di esecuzione) rispetto a batch e minibatch, supponiamo che tale fenomeno sia dovuto al fatto che per l'online si usi un valore di η più piccolo, e che complessivamente si facciano piu aggiornamenti ai pesi della rete, il che si traduce maggiori operazioni. Immaginiamo che tale effetto possa risultare particolarmente evidente sui MONK in quanto sono task semplici che non necessitano di un training lungo, e che pertanto il tempo necessario all'aggiornamento dei pesi sia rilevante, rispetto a task più complessi in cui il costo potrebbe essere dominato dai trasferimenti di dati in memoria (nel caso dei MONK si lavora su pochi record e trasferimenti dovrebbero essere minimi).

Abbiamo notato inoltre sul cup dataset che l'esecuzione della grid search richiede un tempo considerabile (ogni training richiede circa 50 secondi di tempo, con alcune variazioni in configurazioni più lente o configurazioni più veloci; tutte le computazioni sono state effettuate su computer portatili con 4 core fisici, ognuno dotato di 2 contesti virtuali, per un totale di 8 processori virtuali, a 3.2GHz), pertanto pensiamo che un buon approccio allo studio delle prestazioni al variare dei parametri sia approssimare la grid search implementando una ricerca random tra le configurazioni della grid search esaustiva (eventualmente concentrandosi nell'intorno di configurazioni che si nota portano a risultati migliori).

4 Conclusioni

Le predizioni per il blind test sono nel file "LeLontre_ML-CUP19-TS.csv", il nickname del gruppo è "Le Lontre".

5 Acknowledgements

We agree to the disclosure and publication of our names, and of the results with preliminary and final ranking.