Training di una Neural Network con un Algoritmo Momentum Descent

Relazione di Machine Learning a.a. 2017/18
progetto A con CM

Carlo Alessi
carloalessi94@gmail.com

Davide Italo Serramazza
davide.serramazza@gmail.com

3 febbraio 2018

Sommario

Lo scopo di questo progetto è stato di implementare una rete neurale che usi il metodo di discesa del gradiente per effettuare il traning sui data set monk ed ML_Cup e di confrontare i risultati ottenuti con il traning effettuato con BFGS. La tecnica di validazione utilizzata è stata l'hold out. Abbiamo inoltre confrontato l'apprendimento della rete, sul data set ML_Cup, utilizzando come funzione errore il MSE e il MEE, ambedue sia con dati normalizzati che con dati non normalizzati.

1 Introduzione

Nella relazione seguente vengono sintetizzati i risultati ottenuti dal nostro lavoro.

In sezione 2 viene specificata la metodologia seguita nello sviluppo del progetto. In particolare viene specificato il linguaggio di programmazione e le librerie utilizzate, le scelte di desing effettuate, come le classi principali e strutture dati utilizzate nella implementazione della rete, le funzioni di attivazione implementate, e le metodologie di valutazione dei nostri esperimenti.

In sezione 3 vengo riportati i risultati degli esperimenti effettuati. In tale sezione è specificato il preprocessing dei dati, lo schema di valutazione utilizzato per selezionare il modello finale, i risultati ottenuti sui 3 data set monk e sul data set ML_Cup ed infine è riportata una stima del tempo di allenamento sul data set ML_Cup.

In sezione 4 vengono riportati i risultati del confronto fra il training effettuato con l'algoritmo di discesa del gradiente e quello effettuato con il metodo BFGS, in particolare vengono analizzate le capacità di generalizzazione e il tempo necessario per effettuare il traning.

2 Metodo

Tools e librerie Il progetto è stato sviluppato con i seguenti strumenti: Python come linguaggio base, per la facilità di utilizzo e la grande quantità di librerie open source a disposizione. In particolare le librerie Pandas e Scikit-learn sono state usate per il pre-processing dei dataset. La libreria Numpy è stata usata per i calcoli matematici (sopratutto per i prodotti scalari). Infine Matplotlib è stata usata per il rendering dei risultati degli esperimenti.

Software overview e scelte di design Lo sviluppo del software ha seguito un approccio *Object Oriented*. Il software è composto di tre classi principali: NeuralNetwork, Layer, e Neuron. In particolare la classe NeuralNetwork è costituita da una lista di Layer, a loro volta aventi una lista di Neuron. Ogni Neuron ha una lista contenente i pesi associati ai suoi input, e implementa le funzioni di attivazione e le relative derivate.

Sebbene lo scopo del progetto non era di implementare un framework, è stato cercato di riutilizzare il più possibile ogni porzione di codice. In particolare tutti gli esperimenti sono stati parametrizzati ed automatizzati.

Infine per assicurare la "correttezza" di quanto implementato, i metodi più importanti della rete neurale (per esempio le funzioni che implementano *forward-propagation*, *back-propagation* e il *training*) sono state supportate da unit tests costantemente aggiornati.

Scelte implementative Sono state fatte le seguenti scelte implementative. L'architettura e la topologia della rete neurale sono completamente determinate dai due parametri del costruttore della classe NeuralNetwork, *architettura* e *neuroni*. In particolare il primo è un array di interi in cui l'i-esima entry indica il numero di neuroni presenti all'i-esimo layer, mentre il secondo è un array, della stessa lunghezza del primo, che indica il tipo di neuroni presenti in ogni layer.

Sono state usate le seguenti funzioni di attivazione: Sigmoid, TanH e ReLU. Esse sono state implementate rispettivamente nella classe SigmoidNeuron, TanHNeuron e ReLUNeuron. Di seguito verranno omessi i suffissi "Neuron" e si indicherà con, per esempio, TanH un neurone che implementa la funzione di attivazione TanH. Inoltre verrà indicato con TanH* una serie di layer i cui neuroni hanno TanH come funzione di attivazione (il numero di layer sarà deducibile dal contesto). Altri tipi di neurone implementati sono stati InputNeuron, LinearNeuron e OutputNeuron. Questi sono unità lineari che implementano la funzione identità che possono solo essere usati rispettivamente nei layer di input, hidden e output.

Il tipo di allenamento (batch, mini-batch, online) è interamente definito dal parametro batch_size della funzione *train*(). Il metodo di allenamento è il classico gradient descent con momentum, cui viene aggiunto l'errore di regolarizzazione l_2-norm. Come criterio di stop del training è stato usato il numero di epoche. Inoltre è possibile salvare lo stato dell'allenamento scrivendo il tipo di architettura e i pesi della rete su file tramite la funzione *dump_ weights()*. Successivamente è possibile caricare lo stato della rete tramite la funzione *load_ weights()*. Queste funzioni sono per avere diversi "snapshot" di vari modelli al fine di confrontarli tra loro.

Preprocessing In ogni monk-dataset i pattern presentano 6 features categoriche. Come step di preprocessing ogni feature è stata trasformata in un vettore binario secondo la codifica *1-of-k*. Mentre i pattern della ML_Cup sono stati normalizzati.

Validazione Il rischio empirico è stato calcolato usando la tecnica di validazione *holdout*, in cui viene diviso il data set in due partizioni disgiunte, *training set* e *validation set*. La prima partizione viene usata per allenare la rete neurale, mentre la seconda per validarne le prestazioni e decidere la miglior impostazione di iper-parametri. Infine la capacità di generalizzazione dei modelli è stata valutata misurando le prestazioni ottenute su un *test set* indipendente dai due precedenti.

La selezione di migliori modelli è stata fatta tramite grid search. Le metriche usate per valutare le prestazioni dei modelli sono state Accuracy, Misclassification Error, Mean Squared Error e Mean Euclidean Error.

Esperimenti preliminari Gli esperimenti preliminari hanno individuato a grandi linee, per ogni iper-parametro di allenamento, gli intervalli di valore che risultavano in migliori prestazioni e/o tempo di convergenza. Inoltre uno studio più o meno approfondito è stato svolto per individuare il numero di hidden layer e neuroni necessari per il *fitting* dei dataset.

3 Esperimenti

In questa sezione viene riportata un'ampia gamma di esperimenti, concentrandosi sulla tecnica di validazione. I risultati degli esperimenti sono poi analizzati in termini di stabilità dell'apprendimento, velocità di convergenza e capacità di generalizzazione. Nelle sezioni successive indichiamo con η il *learning rate*, con α il termine moltiplicativo del momentum e con λ la forza di regolarizzazione.

Preprocessing Nei Monk datasets ogni esempio di allenamento è composto da 6 features categoriche $a_1, ..., a_6$ che rispettivamente prendono valori nei seguenti domini: $dom(a_1) = \{1, 2, 3\}$, $dom(a_2) = \{1, 2, 3\}$, $dom(a_3) = \{1, 2\}$, $dom(a_4) = \{1, 2, 3\}$, $dom(a_5) = \{1, 2, 3, 4\}$, $dom(a_6) = \{1, 2\}$. Si è verificato empiricamente che la rete neurale, allenata sul data set così codificato, è meno performante. Per questo ogni feature a_i è stata trasformata, secondo la codifica *1-of-k* (*one-hot* encoding), in un vettore binario di lunghezza $\mid dom(a_i) \mid$, in cui solo il j-esimo bit, relativo al j-esimo valore assunto dalla feature, è impostato uguale ad 1, mentre i restanti $j - 1$ sono imposti uguali a 0. Combinando i vettori binari così ottenuti, si è ottenuto un dataset i cui pattern presentano 17 features binarie.

Per quanto riguarda il dataset ML_Cup, gli esempi di allenamento erano composti da 10 features quantitative, più 2 target quantitativi (target_x e target_y). Le features assumevano valori approssimativamente nell'intervallo $(-3, 3)$, mentre i due target erano tali che $target\_x \in [0.89, 27.49]$ e $target\_y \in [-20.48, -1.68]$. Un esperimento effettuato è stato normalizzare features e target nell'intervallo $[-1, 1]$, per ridurre l'influenza sull'allenamento delle feature che sono definite su scala più grande di altre. Durante il training, l'output della rete e i target sono stati riportati in scala originale prima di calcolare l'errore.

Schema di validazione e selezione del modello Lo schema di validazione usato è stato il metodo *holdout*. Per prima cosa ogni dataset è stato mischiato (*data shuffle*), per evitare il rischio che il traning set ed il validation set non fossero rappresentanti dell'intero data set (ovvero che in ciascuno di essi non fosse contenuto ogni possibile tipologia di esempio). Le strategie di ripartizione dei Monk e ML_Cup datasets differiscono di poco.

Per i Monk datasets il 70% dei dati di allenamento è stato usato come *training set* e il rimanente 30% come *validation set*. Le due partizioni sono disgiunte, e tali che le distribuzioni relative delle variabili target seguono la stessa distribuzione dell'intero dataset. Un ulteriore insieme di dati è stato usato come *test set*.

Per la ML_Cup è stato lasciato il 20% dei dati per il *test set*, mentre la rimanente partizione (componente l'80% dei dati) è stata ulteriormente suddivisa in *training set* e *validation set* (rispettivamente 80% e 20% della partizione). In sintesi, le percentuali dell'intero dataset usate per training, validation e test set sono state rispettivamente 64%, 16%, 20%.

Per selezionare il miglior modello è stata effettuata una grid search sugli iper-parametri. Con il fine di non dipendere da inizializzazioni fortuite dei pesi della rete, ogni combinazione di valori è stata valutata 5 volte. Ad ogni prova venivano inizializzati i pesi della rete ad un valore casuale nell'intervallo $[-0.7, +0.7]$ (deciso arbitrariamente), e si ripeteva il training salvando i risultati. La bontà dell'impostazione degli iper-parametri è stata valutata considerando la media delle prestazioni ottenute nelle 5 prove. Il range degli iper-parametri usati nella grid search, per l'allenamento sui Monk e ML_cup datasets, sono riportati rispettivamente in Table 1

Una volta trovati gli iper-parametri che risultavano in migliori prestazioni, la rete neurale è stata allenata da zero sull'insieme di dati formato dalle prime 2 partizioni (*training set* + *validation set*) e il modello così ottenuto è stato valutato sul *test set*.

3.1 Risultati sui Monk datasets

Table 2 riassume le prestazioni ottenute su ogni Monk data set riportando gli iper-parametri scelti dalla grid search, e le prestazioni ottenute sul *training set* e *test set*, in termini di errore quadratico medio (MSE) e Accuracy. Come si nota dalla tabella, si è raggiunto il 100 % di accuratezza e un valore molto basso di errore quadratico medio (dell'ordine di 10^{-3} e 10^{-4}) sia sul Monk 1 che sul Monk 2. Un risultato diverso avrebbe sollevato dubbi sulla corretta implementazione della rete, dato che i

Tabella 1: Range iper-parametri della grid search per l'allenamento sui dataset Monk e ML_Cup.

Iper-parametri	Valori Monk	Valori ML_Cup
η	[0.15, 0.25]	[0.05, 0.5]
α	[0.4, 0.6]	[0.1, 0.7]
λ	[0, 0.03]	[0.01]
batch_size	[10, 30]	[10, lunghezza(*training set*)]
Epoche	100	[100, 300]
Architettura	[17, 10, 1] [17, 5, 5, 1]	[10, 20, 20, 2] [10, 20, 15, 10, 2] [10,10,2]
Neuroni	[Input, TanH*]	[Input, TanH*, Output] [Input, Sigmoid, Output]

dataset sono "facili" (in particolare il *test set* include il *training set*). Mentre nel Monk 3, anche se non si è raggiunta un'accuratezza del 100% a causa del rumore nei dati (*noise*), i risultati sono stati soddisfacenti, raggiungendo il 98.36% di accuratezza sul training set. Inoltre, si nota che inserendo un piccolo valore $\lambda = 0.001$ per il termine di regolarizzazione, si migliorano le capacità di generalizzazione della rete, anche se di poco.

Tabella 2: Prestazioni ottenute sui Monk datasets e relative impostazioni degli iper-parametri.

Task	Monk 1	Monk 2	Monk 3	Monk 3 (reg)
η	0.25	0.25	0.25	0.15
α	0.5	0.6	0.5	0.6
λ	0.0	0.0	0.0	0.001
batch_size	10	10	10	10
architettura	[17,10,1]	[17,10,1]	[17,10,1]	[17,10,1]
neuroni	[Input, TanH*]	[Input, TanH*]	[Input, TanH*]	[Input, TanH*]
epoche	50	30	30	50
MSE (TR/TS)	0.0005 / 0.0035	0.0005 / 0.0007	0.0197 / 0.0454	0.0176 / 0.0370
Accuracy (TR/TS) (%)	100 / 100	100 / 100	98.36 / 94.44	98.36 / 96.01

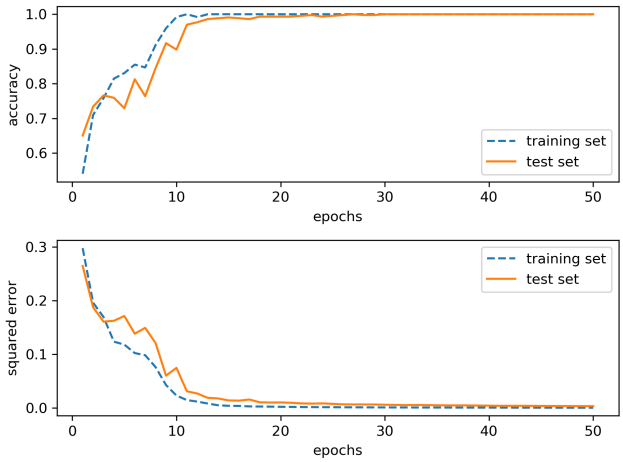
Figure 1 mostra le curve di apprendimento per ogni Monk dataset, riportando l'Accuracy e il MSE, sul *training set* e *test set*, al variare delle epoche di allenamento.

La rete neurale apprende molto velocemente sia sul Monk 1 che sul Monk 2, convergendo approssimativamente in rispettivamente 30 e 10 epoche. In particolare la curva di apprendimento sul Monk 2 è *smooth*. L'apprendimento sul Monk 3 è stato un po' instabile, in quanto le curve non avevano un andamento monotono. L'effetto di aggiungere un basso termine di regolarizzazione ha marginalmente stabilizzato l'apprendimento. Al contrario con valori di regolarizzazione troppo grandi l'apprendimento risultava ancora più instabile. Questo perché un valore di λ troppo elevato fa diventare la componente dell'errore relativa alla regolarizzazione troppo influente rispetto alla componente dell'errore relativa ai dati. Conseguentemente anche i gradienti vengono influenzati e quindi possono essere fatti passi di ottimizzazione lungo una direzione che non è di discesa. L'andamento delle curve di apprendimento sul *training set* e *validation set* viene mostrato in Figure 3 in A.1.

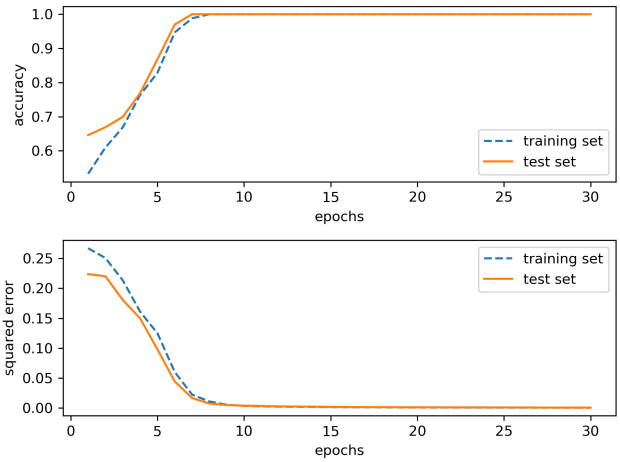
3.2 Risultati ML_Cup

In questa sezione vengono riportati i risultati ottenuti sul dataset ML_Cup. In particolare viene fatto il confronto tra le prestazioni ottenute minimizzando il MEE con quelle ottenute minimizzando il MSE, sia nel caso di pattern normalizzati che non normalizzati. In entrambi i casi verranno prima mostrati i migliori modelli ottenuti con la grid search, riportando l'errore sul *training set* e *validation set*. Infine vengono mostrate le curve di apprendimento dei modelli finali (allenati su *training set* + *validation set*) e riportate le loro prestazioni sul *test set*.

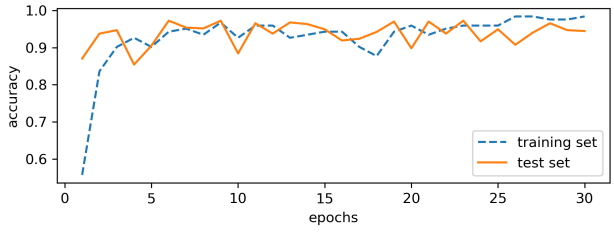
La Table 3 e Table 4 riassumono le prestazioni dei migliori 3 modelli ottenuti con la grid search rispettivamente minimizzando il MEE e il MSE. Figure 2 mostra le curve di apprendimento dei due mo-

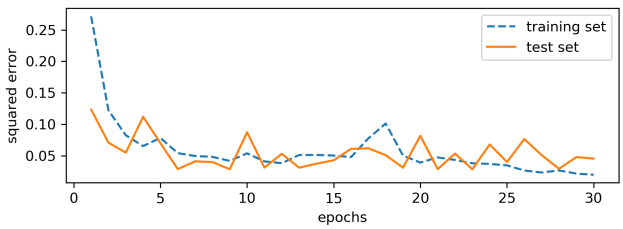


(a) Monk 1

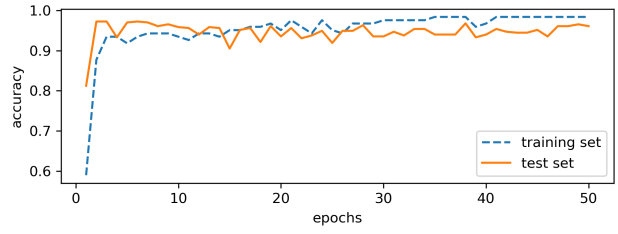


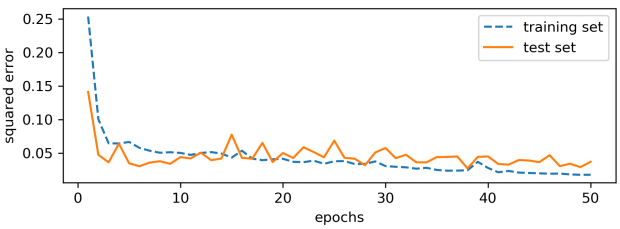
(b) Monk 2





(c) Monk 3





(d) Monk 3 con regolarizzazione

Figura 1: Curve di apprendimento su *training set* e *test set* relative ai Monk datasets ottenute impostando gli iper-parametri con i valori riportati in Table 2.

delli finali rispettivamente ottenute minimizzando il MSE e MEE. Table 5 riassume gli iper-parametri e le prestazioni ottenute dai due modelli finali.

Analizzando Figure 2 risulta chiaramente come i risultati migliori si ottengono con la rete allenata minimizzando il MSE, in quanto l'errore risultante raggiunge un valore minore al 2 mentre nel caso della rete allenata minimizzando il MEE l'errore finale è all'incirca di 7.

Questo comportamento potrebbe essere dovuto al fatto che la derivata per il MSE è uguale a $\frac{\partial E(w)}{\partial o} = -2 \times (d - o)$, mentre la derivata per l'MEE è $\frac{\partial E(w)}{\partial o} = \frac{\sum_i d_i - o_i}{\sqrt{\sum_i (d_i - o_i)^2}}$, dove d ed o sono rispettivamente il vettore dei target e l'output della rete. Nella maggior parte dei casi, il denominatore sarà maggiore di 1 e quindi la derivata del MEE è solo una frazione della derivata del MSE.

Questo potrebbe far si che il fenomeno del gradient vanishing si verifichi molto prima nel caso della rete allenta con l'MEE e quindi non permette di raggiungere lo stesso errore raggiunto con la rete allenta per minimizzare il MSE. Come si può notare inoltre da Table 3 e da Table 4, la rete che minimizza l'MEE è stata allenata per 300 epoche e la sua architettura è di 2 hidden layer da 20 neuroni l'uno, mentre nel secondo caso sono stati sufficienti 100 epoche ed un architettura con un solo hidden layer da 10 neuroni.

Un'altra comparazione eseguita ha riguardato il data set, ed in particolare i risultati che vengono ottenuti a partire dal data set normalizzato (con i valori sia delle feature che dei target compresi fra $[-1, 1]$) e dal data set denormalizzato (ovvero quello a cui non è stata applicata nessuna trasformazione).

Per la rete che minimizza il MEE la normalizzazione ha portato dei benefici: con i dati originali la

rete ha raggiunto un valore di MEE di 17.7523, mentre nel caso di normalizzazione il valore raggiunto è stato di 7.1046; dunque il migliore risultato è stato ottenuto normalizzando il data set.

Invece per la rete allenare minimizzando il MSE la normalizzazione non ha portato vantaggi: il miglior risultato con i dati originali è stato di 1.3801, mentre nel caso di normalizzazione il miglior risultato è stato solo di 2.2135.

In conclusione, il migliore risultato è stato ottenuto dalla rete allenata per minimizzare il MSE con i dati originali.

Tabella 3: Migliori 3 modelli della ML_Cup. Per ogni modello sono riportati gli iper-parametri di allenamento e il MEE ottenuto sul *training set* e *validation set*. Gli errori sono riportati nel caso di features normalizzate (N) e nel caso di features in scala originale. Gli errori sono comunque riportati in scala originale.

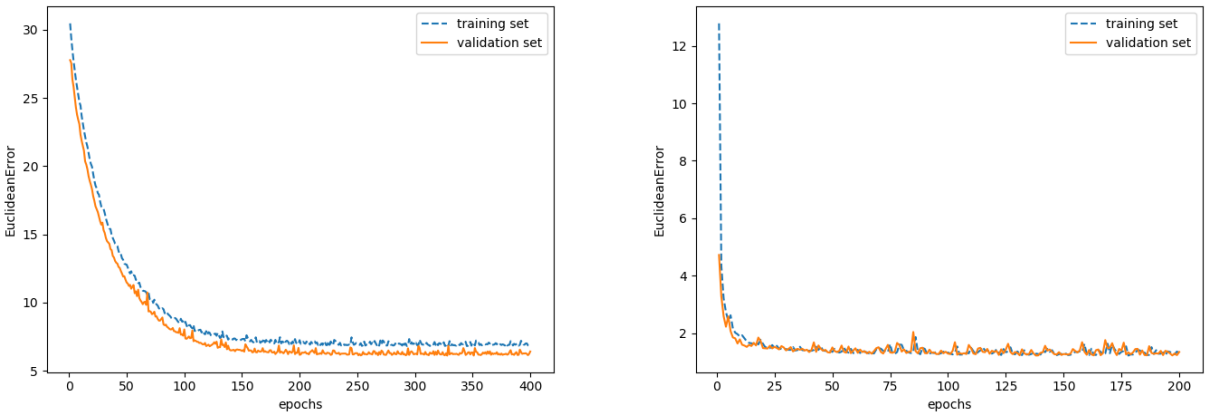
	Modello 1	Modello 2	Modello 3
η	0.2	0.3	0.05
α	0.4	0.4	0.25
λ	0.01	0.01	0.01
batch_size	256	256	10
Architettura	[10, 20, 20, 2]	[10, 20, 20, 2]	[10, 20, 20, 2]
Neuroni	[Input, TanH*, Output]	[Input, TanH*, Output]	[Input, TanH*, Output]
Epoche	300	300	300
MEE (TR / VL)	18.3266 / 18.0085	18.0691 / 17.7523	18.1582 / 17.8416
MEE (TR / VL) (N)	7.1358 / 7.4810	7.1046 / 7.4206	7.4948 / 7.6634

Tabella 4: Migliori 3 modelli della ML_Cup. Per ogni modello sono riportati gli iper-parametri di allenamento e il MSE ottenuto sul *training set* e *validation set*. Gli errori sono riportati nel caso di features normalizzate (N) e nel caso di features in scala originale.

	Modello 1	Modello 2	Modello 3
η	0.2	0.2	0.3
α	0.1	0.3	0.1
λ	0.01	0.01	0.01
batch_size	256	256	256
Architettura	[10, 10, 2]	[10, 10, 2]	[10, 10, 2]
Neuroni	[Input, Sigmoid, Output]	[Input, Sigmoid, Output]	[Input, Sigmoid, Output]
Epoche	100	100	100
MEE (TR / VL)	1.3846 / 1.4438	1.3801 / 1.4531	1.4430 / 1.5800
MEE (TR / VL) (N)	2.3182 / 2.5195	2.2951 / 2.5131	2.2135 / 2.4556

3.3 Stima del tempo di allenamento della ML_Cup

L'analisi della stima del tempo di allenamento si è svolta su una CPU Intel Core i7-7500U 2.7 GHz. La prova è stata effettuata su una rete con $architettura = [10, 20, 20, 2]$ e $neuroni = [Input, TanH, TanH, Output]$ su una partizione di 812 esempi del dataset della ML_Cup. L'analisi è stata effettuata al variare della $batch\_size \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 812\}$. Si è osservato che il tempo di allenamento aveva un andamento quasi non-crescente per valori crescenti di batch_size (vedi Figure 4 in A.2). In particolare il tempo di allenamento medio per l'allenamento *online* era di circa 1.09 secondi, mentre per l'allenamento *batch* era di circa 0.91 secondi. Quindi passando dall'allenamento online al batch si ottiene uno speed-up di $\frac{1.09}{0.91} \simeq 1.199$. Questo perché con batch_size più piccole gli aggiornamenti dei pesi sono più frequenti. Mediamente la rete effettua un forward e un backward step in meno di un secondo.



(a) MEE (b) MSE

Figura 2: Curve di apprendimento per il modello finale ottenute minimizzando il MEE e il MSE.

Tabella 5: Risultati su training e test set dei modelli finali.

	Modello Finale MSE	Modello Finale MEE
η	0.2	0.2
α	0.1	0.4
λ	0.01	0.01
batch_size	256	256
Architettura	[10, 10, 2]	[10, 20, 20, 2]
Neuroni	[Input, Sigmoid, Output]	[Input, TanH*, Output]
Epoche	130	328
MEE (TR)	1.402334	6.991842
MEE (TE)	1.283643	6.121687

4 Confronto algoritmi di apprendimento

Oltre agli esperimenti precedentemente citati è stata implementata una funzione che realizza il train della rete neurale col metodo BFGS, il più noto metodo della famiglia quasi Newton. A differenza dell'algoritmo di discesa del gradiente, il metodo BFGS fa utilizzo anche di un'approssimazione della Hessiana oltre che del gradiente, per minimizzare una determinata funzione.

Uno degli scopi del nostro lavoro è stato confrontare il metodo di discesa del gradiente col metodo BFGS: per fare ciò è stata implementata una "grid search" in cui vengono variati gli iper-parametri del BFGS (ovvero quelli della line search, la funzione che viene utilizzata dal BFGS per decidere la lunghezza dello step), mentre gli altri iper-parametri (come ad esempio l'architettura della rete o il numero di epoche) sono stati posti uguali a quelli utilizzati con l'algoritmo di discesa del gradiente, per avere un confronto il più possibile equo.

Così facendo abbiamo trovato i migliori iper-parametri con cui allenare il nostro modello finale su i dataset elencati precedentemente (ovvero i 3 dataset monk e la ML_Cup).

Riportiamo in Table 6 i risultati ottenuti nella grid search e quelli del successivo traning finale utilizzando gli stessi iper-parametri. Tutti i risultati riportati in tale tabella sono riferiti al data set utilizzato per la valutazione (validation nel caso della grid search e test nel caso del modello finale). Inoltre nel caso del monk utilizziamo il MSE, mentre nel caso della ML_Cup utilizziamo l'MEE.

Per ottenere la migliore combinazione di iper-parametri nella grid search, sono stati effettuati 5 tentativi per ogni combinazione e successivamente è stata presa la media di questi tentativi (ovvero abbiamo utilizzato lo stesso approccio seguito precedentemente).

Tabella 6: Valori dei migliori iper-parametri e corrispondenti risultati ottenuti col metodo BFGS. Il tempo di allenamento è riferito ad una CPU intel i5-6200u con 2 core da 2,3 Ghz

	Monk 1	Monk 2	Monk 3	Monk 3 regolarizzato	Ml Cup
$c1$	0.005	0.005	0.001	0.005	0.001
$c2$	0.9	0.9	0.9	0.85	0.9
θ	0.9	0.7	0.7	0.7	0.9
MSE/MEE (VL)	0.054	0.369	0.093	0.077	2.34
accuracy(VL)	0.963	0,874	0.907	0.924	-
MSE/MEE (TS)	0.332	0.853	0.204	0.203	1.369
accuracy(TS)	0.992	0.707	0.96	0.984	-
tempo di allenamento	29,5s	36s	12,5s	13,4s	589s

Monk 1

In media l'algoritmo BFGS è stato in grado di ottenere dei buoni risultati di generalizzazione sul monk 1: in tutte le configurazioni di iper-parametri testate la rete è stata in grado di raggiungere un accuracy maggiore o uguale a 0.9. La configurazione di iper-parametri migliore in 4 trial su 5 è stato in grado di convergere (ovvero classificare correttamente ogni esempio) sul traning set.

Il tempo medio di allenamento è stato di 29,5 secondi per effettuare 50 iterazioni, mentre nel caso della discesa del gradiente il tempo medio è stato di 4,5 secondi.

Monk 2

Sul monk 2 l'algoritmo BFGS si è comportato in maniera peggiore rispetto al monk 1: in media l'accuracy raggiunta dalle varie configurazioni di iper-parametri si è attestata su un valore di 0.8 sul training set.

Anche il tempo di allenamento del monk 2 è stato peggiore del monk 1, infatti in media il tempo di allenamento è stato di 36 secondi. Analizzando questo risultato esso non può essere giustificato dalla dimensione del data set (maggiore nel caso del monk 2) poiché in tale caso sono state effettuate solamente 30 epoche a differenza del monk 1 in cui ne sono state effettuate 50.

Per la discesa del gradiente il tempo di allenamento è stato di 5,1 secondi.

Monk 3

Per quanto riguarda il monk 3 facciamo prima una analisi relativa al caso in cui non è stata utilizzata regolarizzazione, e poi del caso in cui il coefficiente di regolarizzazione è stato fissato a 0.01.

Senza l'uso di regolarizzazione l'algoritmo BFGS ha ottenuto risultati comparabili con quelli ottenuti sul monk 1: nella maggior parte dei casi l'accuracy ottenuta è stata superiore allo 0,9. Con la configurazione di iper-parametri più performante sul traning set, in 4 trial su 5 si è ottenuta una misclassification di 0,011 e nell'ultimo trial è stata in grado di convergere.

Il tempo di allenamento medio è stato di 12,5 secondi, mente nel caso di discesa del gradiente di 3,5 secondi.

Includendo anche la regolarizzazione, come presumibile, la capacità di generalizzazione della rete sono ulteriormente migliorate: in media l'accuracy si è attestata attorno allo 0,95. Il tempo medio di allenamento è stato all'incirca uguale al caso precedente.

ML_Cup

Sul data set ML_Cup il trainig effettuato col BFGS, ha mostrato delle performance peggiori al metodo di discesa del gradiente, come accaduto sul monk 2. Il MEE medio è stato all'incirca di 2,79, ed anche il tempo medio di allenamento per effettuare 200 epoche è stato di 589 secondi, contro i 53 secondi della discesa del gradiente

Analisi dei risultati ottenuti

Analizzando i risultati ottenuti, la rete allenata col metodo BFGS ha ottenuto delle performance paragonabili alla discesa del gradiente in termini di generalizzazione per i data set più piccoli. Infatti sia l'accuracy ottenuta sul monk 1 che sul monk 3 è stata paragonabile al metodo di discesa del gradiente.

Il punto debole del BFGS è stato il comportamento con i data set più grandi. Oltre ad una generalizzazione peggiore, anche il tempo di allenamento è stato considerevolmente maggiore, in particolare all'aumentare del numero di epoche già effettuate. Infatti possiamo vedere come la discrepanza maggiore, per quanto riguarda il tempo di training, si è riscontrata nel monk 2 (il data set più grande fra quelli del monk) e nella ML_Cup.

La ragione di ciò è probabilmente dovuta alla line search, che come accennato precedentemente, viene utilizzata dal BFGS per decidere la lunghezza dello step da compiere. In particolare essa prova inizialmente uno step di lunghezza 1, e nel caso non migliori la valutazione della funzione (nel nostro caso il MSE o MEE), diminuisce tale valore finché non trova uno step di lunghezza adeguata.

In tale processo è richiesto il calcolo della derivata e la valutazione della funzione in un punto diverso per ogni tentativo effettuato della line search: questo processo chiaramente è molto costoso in termini computazionali.

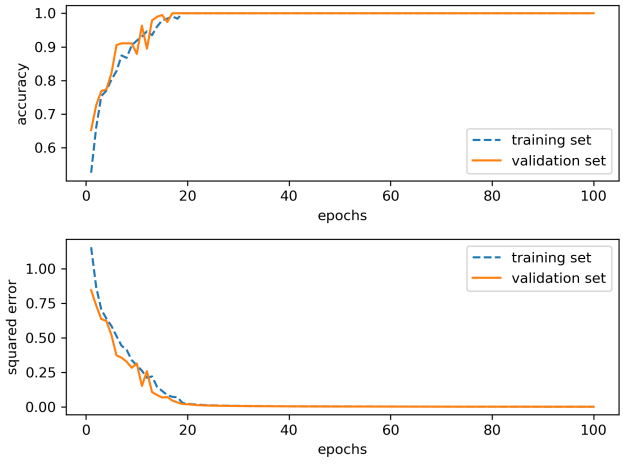
5 Conclusioni

In questo progetto abbiamo disegnato e implementato un piccolo framework che permette di definire, allenare e valutare semplici reti neurali di tipo *fully-connected* e *feed-forward*. In particolare abbiamo implementato gli algoritmi di allenamento di discesa del gradiente e BFGS, e i metodi per valutare le loro prestazioni. Sui tre dataset per il benchmark si sono ottenuti buoni risultati. In particolare sui dataset monk-1 e monk-2 si è ottenuto il 100% di accuracy, mentre nel monk-3 si è verificato come il termine di regolarizzazione aumenti le capacità di generalizzazione della rete nel caso in cui sia presente rumore nei dati (vedi Table 2). Nel dataset della ML_Cup si sono studiati gli effetti della normalizzazione dei dati sull'apprendimento, e sono stati analizzati due modelli allenati rispettivamente minimizzando il MSE e il MEE. Dagli esperimenti è risultato che il modello finale ottenuto minimizzando il MSE ha avuto prestazioni migliori lavorando con il dataset originale, raggiungendo un MEE di 1.4923 e 1.2836 rispettivamente sul training e test set. Al contrario minimizzando il MEE la rete lavorava meglio con il dataset normalizzato, raggiungendo però solo un MEE di 6.9918 e 6.1217 rispettivamente sul training e test set (vedi Table 5). Inolte sono stati messi a confronto le prestazioni dei modelli ottenuti con discesa del gradiente e con metodo BFGS. I risultati emersi dicono che l'algoritmo BFGS converge più lentamente del metodo standard, a causa delle continue valutazioni di funzione ed aggiornamento dei pesi effettuate dalla line search. Tuttavia i risultati sul dataset della ML_Cup sono altrettanto buoni, dove si raggiunge un MEE di 2.34 e 1.369 rispettivamente sul validation e test set. Mentre i modelli ottenuti sui monk datasets non sempre generalizzano bene (vedi Table 6).

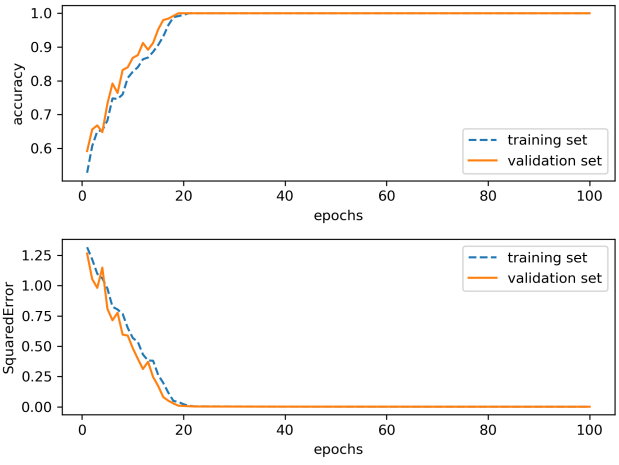
A Appendici

A.1 Curve di apprendimento su training e validation set dei Monk

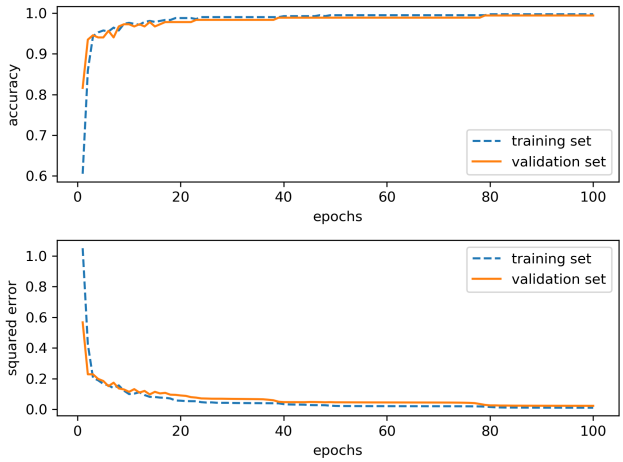
Figure 3 giustifica la scelta degli iper-parametri riportati in Table 2, mostrando le curve di apprendimento della rete, sul *training set* e *validation set*, relative ai Monk datasets.



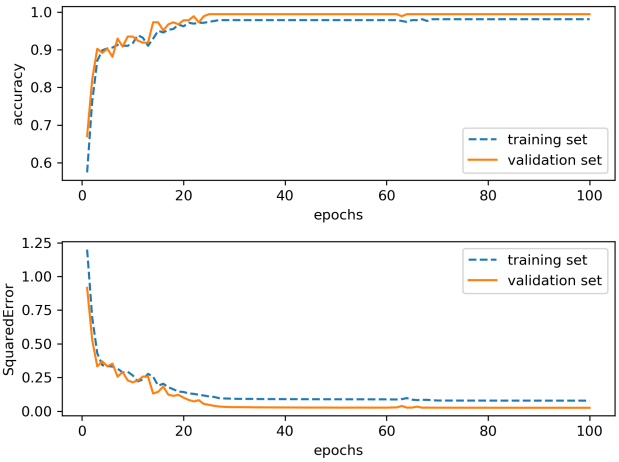
(a) Monk 1



(b) Monk 2



(c) Monk 3



(d) Monk 3 con regolarizzazione

Figura 3: Curve di apprendimento su *training set* e *validation set* sui Monk datasets ottenute impostando gli iper-parametri con i valori riportati in Table 2.

A.2 Tempo di allenamento al variare della batch_size

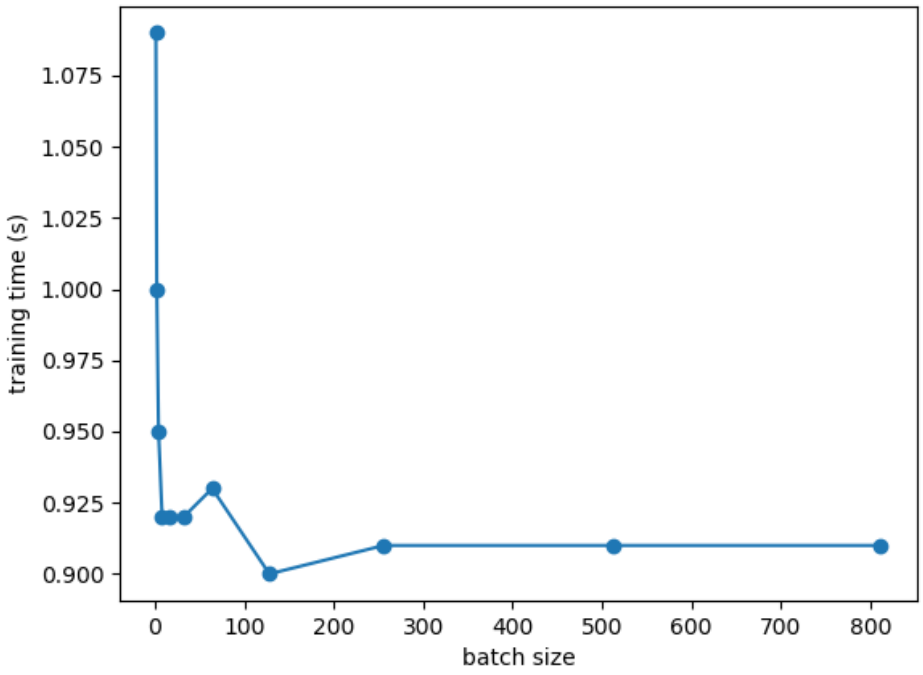


Figura 4: Andamento medio del tempo di un epoca di allenamento al variare della batch_size.