



UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica

COMPRESSIONE DI RETI NEURALI IN PROBLEMI DI CLASSIFICAZIONE E REGRESSIONE

Relatore:
Prof. Dario MALCHIODI
Correlatore:
Dr. Marco FRASCA

Tesi di Laurea di:
Giosuè Cataldo Marinò
Matricola: 829404

Anno Accademico 2018/2019

Indice

1	Reti Neurali	7
1.1	Reti Neurali Biologiche	7
1.2	Reti Neurali Artificiali	8
1.3	Addestramento della rete	9
1.4	Funzioni di attivazione	9
1.5	MultiLayer Perceptron	10
1.5.1	Architettura del modello MLP	10
1.5.2	Addestramento	12
2	Metodi di compressione	15
2.1	Pruning	15
2.1.1	Strutture dati necessarie	15
2.1.2	Tecniche implementative	16
2.1.3	Tasso di compressione	16
2.2	Weight Sharing	16
2.2.1	Strutture dati necessarie	16
2.2.2	Tecniche implementative	17
2.2.3	Tasso di compressione	17
3	Esperimenti	19
3.1	MNIST	19
3.1.1	Addestramento e tuning parametri	19
3.1.2	Pruning	21
3.1.3	Weight sharing	23
3.2	Problema del predecessore	24
3.2.1	Pruning	25
3.2.2	Weight Sharing	27
3.2.3	MSE vs MAE	32
3.2.4	MSE vs LSE	32
3.2.5	Splitting in N modelli	36

Introduzione

Le reti neurali offrono un insieme di strumenti molto potente che permette di risolvere problemi nell'ambito della classificazione, della regressione e del controllo non-lineare. Un semplice esempio è l'analisi di un'immagine per verificare la presenza di determinati oggetti oppure di testi, ed eventualmente riconoscerli. Oltre ad avere un'elevata velocità di elaborazione, le reti neurali hanno la capacità di imparare la soluzione da un determinato insieme di esempi. In molte applicazioni questo permette di aggirare il bisogno di sviluppare un modello dei processi fisici alla base del problema, che spesso può essere difficile, se non impossibile, da trovare. L'ispirazione per le reti neurali deriva dagli studi sui meccanismi di elaborazione dell'informazione nel sistema nervoso biologico, in particolare il cervello umano; infatti, gran parte della ricerca sulle reti ha proprio lo scopo di capire più a fondo questi meccanismi. Daremo nel seguito un quadro semplificato del funzionamento delle reti neurali biologiche in modo da trovare una corrispondenza pratica dei concetti riguardanti le reti neurali artificiali. In questo elaborato le reti neurali artificiali verranno applicate a due problemi: classificazione e regressione. Due dei principali problemi delle reti neurali sono il tempo di addestramento e lo spazio occupato dalle strutture per l'implementazione; entrambi dovuti a numerosi fattori quali numero di parametri, specifiche della macchina, complessità e dimensione degli input. Per quest'ultimo fattore, la ricerca di metodi di compressione è fondamentale per cercare di ridurre l'impatto degli svantaggi di una rete su un sistema. Per provare a ridurre lo spazio occupato dalle reti neurali, cercando di mantenere il più possibile le performance, verranno esplicitati due algoritmi di compressione. Infine, riporteremo i risultati dell'applicazione delle tecniche di compressione su diversi dataset.

Capitolo 1

Reti Neurali

In questo capitolo descriveremo le reti neurali artificiale, partendo dall'analogia con il sistema nervoso. Successivamente vengono descritti i modelli multilayer-perceptron e un metodo di apprendimento di questo modello.

1.1 Reti Neurali Biologiche

I neuroni sono delle celle elettricamente attive e il sistema nervoso centrale ne contiene circa 10^{11} . La maggior parte di essi ha la forma indicata in Figura 1.1. I dendriti rappresentano gli ingressi del neurone mentre l'assone ne rappresenta l'uscita. La comunicazione tra i neuroni avviene alle giunzioni, chiamate sinapsi. Ogni neurone è tipicamente connesso ad un migliaio di altri neuroni e, di conseguenza, il numero di sinapsi nel cervello supera 10^{14} .

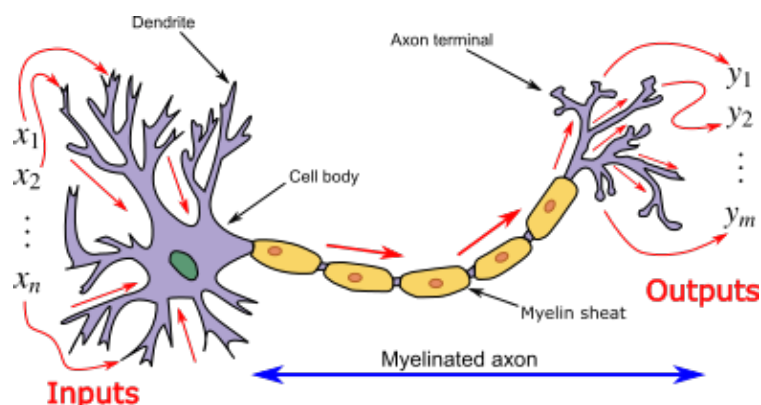


Figura 1.1: Neurone Biologico [10]

Ogni neurone si può trovare principalmente in 2 stati: attivo o a riposo. Quando il neurone si attiva, esso produce un potenziale di azione (impulso elettrico) che viene trasportato lungo l'assone. Una volta che il segnale raggiunge la sinapsi esso provoca il rilascio di sostanze chimiche (neurotrasmettitori) che attraversano la giunzione ed entrano nel corpo di altri neuroni. In base al tipo di sinapsi, che possono essere eccitatori o inibitori, queste sostanze aumentano o diminuiscono rispettivamente le possibilità che il successivo neurone si attivi. Ad ogni sinapsi è associato un peso che ne determina il tipo e l'ampiezza dell'effetto eccitatore o inibitore. Quindi, semplificando, si può dire che ogni neurone effettua una somma pesata degli ingressi provenienti dagli altri neuroni e, se questa somma supera una certa soglia, il neurone si attiva. Ogni neurone, operando ad un ordine temporale del millisecondo, rappresenta un sistema di

elaborazione relativamente lento; tuttavia, l'intera rete ha un numero molto elevato di neuroni e sinapsi che possono operare in modo parallelo e simultaneo, rendendo l'effettiva potenza di elaborazione molto elevata. Inoltre, la rete neurale biologica ha un'alta tolleranza ad informazioni poco precise (o sbagliate), ha la facoltà di apprendimento e generalizzazione.

1.2 Reti Neurali Artificiali

Ci concentreremo su una classe particolare di modelli di reti neurali: le reti a catena aperta. Queste reti possono essere viste come funzioni matematiche non lineari che trasformano un insieme di variabili indipendenti $x = (x_1, \dots, x_d)$, chiamate ingressi della rete, in un insieme di variabili dipendenti $y = (y_1, \dots, y_c)$, chiamate uscite della rete. La precisa forma di queste funzioni dipende dalla struttura interna della rete e da un insieme di valori $w = (w_1, \dots, w_d)$, chiamati pesi. Possiamo quindi scrivere la funzione della rete nella forma $y = y(x; w)$ che denota il fatto che y sia una funzione di x parametrizzata da w .

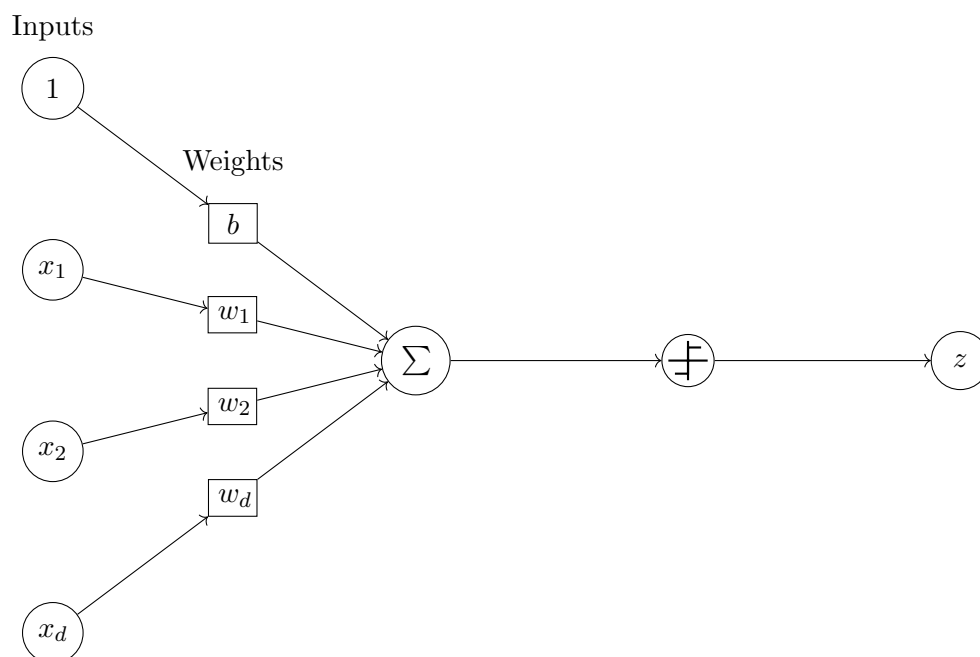


Figura 1.2: Modello di McCulloch-Pitts

Modello di McCulloch-Pitts

Un semplice modello matematico di un singolo neurone è quello rappresentato in Figura 1.2 ed è stato proposto da McCulloch e Pitts [4] alle origini delle reti neurali artificiali. Esso può essere visto come una funzione non lineare che trasforma le variabili di ingresso x_1, \dots, x_d nella variabile di uscita z . Nell'elaborato ci riferiremo a questo modello come unità di elaborazione, o semplicemente unità. In questo modello, viene effettuata la somma ponderata degli ingressi, usando come pesi i valori w_1, \dots, w_d (che sono analoghi alle potenze delle sinapsi nella rete biologica), ottenendo così

$$a = \sum_{i=1}^d w_i x_i + b \quad (1.1)$$

dove il parametro b viene chiamato bias (corrisponde alla soglia di attivazione del neurone biologico). Se definiamo un ulteriore ingresso x_0 , impostato costantemente a 1, possiamo scrivere (1.1) come

$$a = \sum_{i=0}^d w_i x_i \quad (1.2)$$

dove $x_0 = 1$. Precisiamo che i valori dei pesi possono essere di qualsiasi segno, che dipende dal tipo di sinapsi. L'uscita z (che può essere vista come tasso medio di attivazione del neurone biologico) viene ottenuta applicando ad a una trasformazione non lineare g , chiamata funzione di attivazione, ottenendo

$$z = g(a) = g\left(\sum_{i=0}^d w_i x_i\right). \quad (1.3)$$

Il modello originale di McCulloch-Pitts usava come attivazione la funzione gradino

$$g(a) = \begin{cases} 1 & \text{se } a \geq 0, \\ -1 & \text{altrimenti.} \end{cases} \quad (1.4)$$

1.3 Addestramento della rete

Abbiamo detto che una rete neurale può essere rappresentata dal modello matematico $y = y(x; w)$, che è una funzione di x parametrizzata dai pesi w . Prima di poter utilizzare questa rete, dobbiamo identificare il modello, ovvero dobbiamo determinare tutti i parametri w . Il processo di determinazione di questi parametri è chiamato addestramento e può essere un'azione molto intensa dal punto di vista computazionale. Tuttavia, una volta che sono stati definiti i pesi, nuovi ingressi possono essere elaborati molto rapidamente. Per addestrare una rete abbiamo bisogno di un insieme di esempi, chiamato insieme di addestramento (*training set*), i cui elementi sono coppie (x^q, t^q) , $q = 1, \dots, n$, dove t^q rappresenta il valore di uscita desiderato, chiamato target, in corrispondenza dell'ingresso x^q . L'addestramento consiste nella ricerca dei valori per i parametri w che minimizzano un'opportuna funzione di errore. Ci sono diverse forme di questa funzione, la più usata risulta essere la somma dei quadrati residui. I residui sono definiti come

$$r_{qk} = y_k(x^q; w) - t_k^q \quad (1.5)$$

dove k rappresenta l'indice dei neuroni di output. La funzione di errore E risulta allora essere

$$E = \sum_{q=1}^n \sum_{k=1}^c r_{qk}^2. \quad (1.6)$$

È facile osservare che E dipende da x^q e da t^q che sono valori noti e da w che è incognito, quindi E è in realtà una funzione dei soli pesi w .

1.4 Funzioni di attivazione

Come già introdotto nel Paragrafo 1.2, le funzioni di attivazione determinano l'output di ciascun neurone della rete neurale. Le funzioni utilizzate principalmente negli esperimenti di questo elaborato sono tre: *sigmoid*, *ReLU* (*Rectified Linear Units*) e *LeakyReLU* descritte in (1.7 - 1.9). Di solito sono funzioni non-lineari e la loro derivata è calcolabile in modo analitico per velocizzare la computazione.

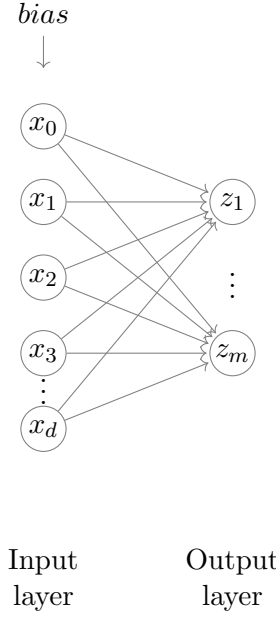


Figura 1.3: MLP a uno strato

$$\text{sigmoid}(a) = \frac{1}{1 + e^{-a}}, \quad \text{sigmoid}'(a) = \text{sigmoid}(a)(1 - \text{sigmoid}(a)). \quad (1.7)$$

$$\text{ReLU}(a) = \max(0, a), \quad \text{ReLU}'(a) = \begin{cases} 1 & \text{se } a \geq 0, \\ 0 & \text{altrimenti.} \end{cases} \quad (1.8)$$

$$\text{LeakyReLU}(a) = \begin{cases} a & \text{se } a \geq 0, \\ -\alpha a & \text{altrimenti.} \end{cases} \quad \text{LeakyReLU}'(a) = \begin{cases} 1 & \text{se } a \geq 0, \\ -\alpha & \text{altrimenti.} \end{cases} \quad (1.9)$$

dove α è un parametro numerico.

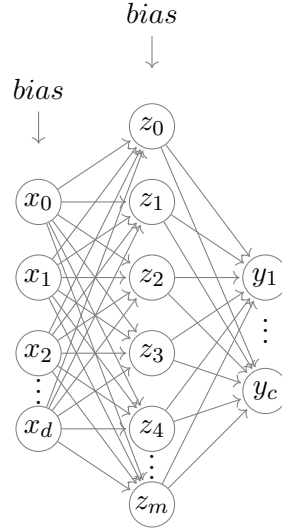
La scelta della funzione è guidata dal tipo di problema che si vuole affrontare, per esempio se vogliamo un output compreso tra 0 e 1 sarà più adeguato utilizzare una funzione *sigmoid* rispetto ad una *ReLU*.

1.5 MultiLayer Perceptron

In questo paragrafo parleremo in dettaglio del modello multilayer perceptron introducendo il concetto di multistrato e descriveremo il metodo di addestramento utilizzato principalmente nel Capitolo 3. Nell'elaborato ci riferiremo a questo modello come MLP.

1.5.1 Architettura del modello MLP

Modello a uno strato Nel paragrafo precedente abbiamo trattato la singola unità di elaborazione descritta in (1.4). Se consideriamo ora un insieme di m unità, con ingressi comuni, otteniamo una rete neurale a singolo strato come in Figura 1.3. Le uscite di questa rete sono



Input layer Hidden layer Output layer

Figura 1.4: MLP a due strati

date da

$$z_j = g \left(\sum_{i=0}^d w_{ij} x_i \right), \quad j = 1, \dots, m \quad (1.10)$$

dove w_{ij} rappresenta il peso che connette l'ingresso i con l'uscita j ; g è la funzione di attivazione e $x_0 = 1$ per incorporare il bias, come precedentemente spiegato.

Modello a due strati Per ottenere reti più potenti¹ è necessario considerare reti aventi più strati chiamate *multilayer perceptron* come in Figura 1.4. Le unità centrali rappresentano lo strato nascosto (*hidden*) perchè il valore di attivazione delle singole unità di questo strato non sono misurabili dall'esterno. L'attivazione di queste unità è data da (1.10). Le uscite della rete vengono ottenute tramite una seconda trasformazione, analoga alla prima, sui valori z_j ottenendo

$$y_k = \tilde{g} \left(\sum_{j=0}^m \tilde{w}_{jk} z_j \right), \quad k = 1, \dots, c, \quad (1.11)$$

dove \tilde{w}_{jk} rappresenta il peso del secondo strato che connette l'unità nascosta j all'unità di uscita k . Sostituendo (1.10) in (1.11) otteniamo

$$y_k = \tilde{g} \left(\sum_{j=0}^m \tilde{w}_{jk} g \left(\sum_{i=0}^d w_{ij} x_i \right) \right), \quad k = 1, \dots, c. \quad (1.12)$$

La funzione di attivazione \tilde{g} , applicata alle unità di uscita, può essere diversa dalla funzione di attivazione g , applicata alle unità nascoste.

¹un perceptrone mono-strato non è in grado di esprimere tutte le funzioni possibili, mentre un perceptrone a più strati lo è [1].

Per ottenere una capacità di rappresentazione universale, la funzione di attivazione g delle unità nascoste deve essere non lineare [1]. Se g e \tilde{g} fossero entrambe lineari, (1.12) diventerebbe un prodotto tra matrici, che è esso stesso una matrice. Inoltre, come vedremo più avanti, le funzioni di attivazione devono essere differenziabili.

1.5.2 Addestramento

L'addestramento consiste nella ricerca dei valori $\mathbf{w} = (w_1, \dots, w_n)$ ² che minimizzano la funzione di errore $E(\mathbf{w})$ (vista precedentemente nel Capitolo 1.3). La ricerca del minimo avviene in modo iterativo partendo da un valore iniziale \mathbf{w} , scelto in modo casuale o tramite un criterio. Alcuni algoritmi trovano il minimo locale più vicino al punto iniziale, mentre altri riescono a trovare il minimo globale.

Diversi algoritmi di ricerca del punto minimo fanno uso delle derivate parziali della funzione di errore E , ovvero del suo vettore gradiente ∇E . Questo vettore indica la direzione ed il verso di massima crescita di E nel punto \mathbf{w} .

Error back-propagation

L'algoritmo di *Error back-propagation* [7] confronta il valore in uscita con il valore desiderato. Sulla base della differenza calcolata, l'algoritmo modifica i pesi della rete neurale, facendo convergere progressivamente il set dei valori di uscita verso quelli desiderati. Consideriamo come funzione errore la somma dei quadrati residui (1.6).

$$E = \sum_{q=1}^n E^q, \quad E^q = \sum_{k=1}^c [y_k(x^q; w) - t_k^q]^2. \quad (1.13)$$

Possiamo vedere E come somma di E^q che corrisponde alla coppia (x^q, t^q) . Grazie alla linearità della derivazione possiamo calcolare la derivata di E come somma delle derivate dei termini E^q . Nel seguito omettiamo l'indice q : i passaggi indicati si riferiscono ad un singolo caso q ma le operazioni sono fatte per ogni valore di q . Consideriamo un esempio di rete neurale MLP con uno strato hidden.

$$y_k = \tilde{g}(\tilde{a}_k), \quad \tilde{a}_k = \sum_{j=0}^m \tilde{w}_{jk} z_j. \quad (1.14)$$

La derivata di E^q rispetto ad un generico peso w_{jk} dello strato hidden è

$$\frac{\partial E^q}{\partial \tilde{w}_{jk}} = \frac{\partial E^q}{\partial \tilde{a}_k} \frac{\partial \tilde{a}_k}{\partial w_{jk}} \quad (1.15)$$

e tramite (1.14) otteniamo

$$\frac{\partial \tilde{a}_k}{\partial \tilde{w}_{jk}} = z_j. \quad (1.16)$$

Con (1.14) e (1.13) otteniamo

$$\frac{\partial E^q}{\partial \tilde{a}_k} = \tilde{g}'(\tilde{a}_k)[y_k - t_k], \quad (1.17)$$

possiamo ora riscrivere (1.15) come

$$\frac{\partial E^q}{\partial w_{jk}} = \tilde{g}'(\tilde{a}_k)[y_k - t_k] z_j. \quad (1.18)$$

²con \mathbf{w} intendiamo i pesi di ogni strato

Definiamo

$$\tilde{\delta}_k = \frac{\partial E^q}{\partial \tilde{a}_k} = \tilde{g}'(\tilde{a}_k)[y_k - t_k] \quad (1.19)$$

ottenendo una semplice espressione per la derivata di E^q rispetto a w_{jk}

$$\frac{\partial E^q}{\partial \tilde{w}_{jk}} = \tilde{\delta}_k z_j. \quad (1.20)$$

Per quanto riguarda le derivate rispetto ai pesi del primo strato riscriviamo

$$z_j = g(a_j), \quad a_j = \sum_{i=0}^d w_{ij} x_i. \quad (1.21)$$

Possiamo quindi scrivere la derivata come

$$\frac{\partial E^q}{\partial w_{ij}} = \frac{\partial E^q}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}. \quad (1.22)$$

In modo analogo, osservando (1.21), otteniamo

$$\frac{\partial a_j}{\partial w_{ij}} = x_i. \quad (1.23)$$

Per il calcolo della derivata di E^q rispetto ad a_j , usando la *chain-rule* abbiamo

$$\frac{\partial E^q}{\partial a_j} = \sum_{k=1}^c \frac{\partial E^q}{\partial \tilde{a}_k} \frac{\partial \tilde{a}_k}{\partial a_j}, \quad (1.24)$$

dove la derivata di E^q rispetto ad \tilde{a}_k è data da (1.18), mentre la derivata di \tilde{a}_k rispetto ad a_j si trova usando (1.14) e (1.21); quindi

$$\frac{\partial \tilde{a}_k}{\partial a_j} = \tilde{w}_{jk} g'(a_j). \quad (1.25)$$

Usando (1.19), (1.24) e (1.25) otteniamo

$$\frac{\partial E^q}{\partial a_j} = g'(a_j) \sum_{k=1}^c \tilde{w}_{jk} \tilde{\delta}_k. \quad (1.26)$$

Possiamo quindi riscrivere (1.22) come

$$\frac{\partial E^q}{\partial w_{ij}} = g'(a_j) x_i \sum_{k=1}^c w_{jk} \delta_k \quad (1.27)$$

e, come abbiamo fatto in (1.19), poniamo

$$\delta_j = \frac{\partial E^q}{\partial a_j} = g'(a_j) \sum_{k=1}^c \tilde{w}_{jk} \tilde{\delta}_k, \quad (1.28)$$

ottenendo infine

$$\frac{\partial E^q}{\partial w_{ij}} = \delta_j x_i \quad (1.29)$$

che ha la stessa semplice forma di (1.20). Elenchiamo quindi i passi da seguire per valutare la derivata della funzione E :

- Per ogni coppia (x^q, t^q) valutare le attivazioni delle unità nascoste e di uscita usando le equazioni (1.21) e (1.14);
- Valutare il valore $\tilde{\delta}_k$ per $k = 1, \dots, c$ usando equazione (1.19);
- Valutare il valore δ_j per $j = 1, \dots, m$ usando equazione (1.28);
- Valutare il valore di E^q usando le equazioni (1.29) e (1.20);
- Ripetere i passi precedenti per ogni coppia (x^q, t^q) del *training set* e sommare tutte le derivate per ottenere la derivata della funzione errore E .

Dopo il calcolo delle derivate i pesi di ogni strato verranno aggiornati come

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t)}, \quad (1.30)$$

$$\Delta w_{ij}^{(t)} = -\eta \nabla E(w_{ij}^{(t)}), \quad (1.31)$$

dove $\eta > 0$ è il coefficiente di apprendimento (*learning rate*): più η è grande più imparerà velocemente, valori troppo grandi di η fanno divergere la rete. Questo verrà ripetuto iterativamente ogni epoca, dove con epoca intendiamo la visione dell'intero *training set*. L'aggiornamento dei pesi durante ogni epoca può avvenire dopo ogni elemento (*online*), dopo tutti gli elementi (*batch*) o dopo un numero parametrico di esempi (*mini-batch*). Negli esperimenti di questo elaborato viene utilizzata la modalità *mini-batch* perché permette al modello di convergere più velocemente.

Per migliorare la convergenza della rete abbiamo utilizzato la versione di discesa del gradiente con *momento* [5]. In questa versione l'equazione (1.31) diventa:

$$w_{ij}^{(t)} = -\eta \nabla E(w_{ij}^{(t)}) + \mu \Delta w_{ij}^{(t-1)}, \quad (1.32)$$

dove μ è un parametro aggiuntivo nell'intervallo $[0, 1)$ che valorizza quanto considerare il gradiente dell'epoca precedente. Questo tipo di aggiornamento accelererà la convergenza se il verso del gradiente è lo stesso dell'epoca precedente.

Criteri di arresto

La procedura di addestramento del paragrafo precedente viene iterata fino a che non risulta soddisfatto un prefissato criterio di arresto. Negli esperimenti descritti nel Capitolo 3, sono stati utilizzati diversi criteri, quali:

- Stop dopo un numero prefissato di epoche;
- Stop dopo che l'errore/accuratezza non migliora rispetto all'epoca precedente;
- Stop dopo che l'errore/accuratezza non migliora rispetto all'epoca precedente con *patience*, ovvero attendendo in ogni caso un numero finito di epoche senza miglioramento delle prestazioni prima di interrompersi.

Capitolo 2

Metodi di compressione

In questo capitolo descriveremo nel dettaglio come funzionano gli algoritmi di compressione utilizzati nel Capitolo 3.

2.1 Pruning

Il pruning consiste nel tagliare le connessioni da una rete addestrata per poi riaddestrarla senza le connessioni tagliate. Oltre a un vantaggio computazionale può portare a una generalizzazione che permette di ridurre l'overfitting (ovvero imparare l'associazione tra oggetti ed etichette solo per gli esempi del training set ma non per nuovi oggetti).

2.1.1 Strutture dati necessarie

Dopo aver tagliato, disattivato le connessioni e riaddestrato, la matrice sarà più o meno sparsa (in base a quante connessioni tagliamo). Per ridurre lo spazio viene utilizzata una rappresentazione matriciale CSC (Compressed Sparse Column)¹. Questo tipo di matrice è una struttura basata sull'indicizzazione tramite colonne di una matrice sparsa. Viene descritta da tre vettori:

- il primo in cui vengono salvati i valori non nulli dal primo elemento in alto a sinistra proseguendo verso il basso e successivamente a destra;
- il secondo corrisponde all'indice delle righe dei valori;
- il terzo indica gli indici dei valori in cui ogni colonna inizia.

$$\begin{pmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (2.1)$$

La matrice illustrata in 2.1, utilizzando la rappresentazione CSC, diventa

$$\begin{aligned} & [2 \ 3 \ 1 \ 5 \ 9 \ 1 \ 1 \ 4 \ 3 \ 3], \\ & [0 \ 4 \ 6 \ 2 \ 5 \ 0 \ 2 \ 3 \ 0 \ 4], \\ & [0 \ 3 \ 3 \ 5 \ 7 \ 7 \ 8 \ 10]. \end{aligned} \quad (2.2)$$

¹https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csc_matrix.html

Questo tipo di struttura dati richiede il salvataggio di $2a + c + 1$ dove a è il numero di valori non zero e c il numero di colonne.

2.1.2 Tecniche implementative

Durante la configurazione della rete viene aggiunta una procedura che esegue il pruning sulle matrici delle connessioni addestrate in precedenza. L'idea di base è quella di annullare le connessioni i cui pesi sono relativamente vicini a zero, nell'ipotesi che il loro contributo all'attivazione dei neuroni sia trascurabile. Se identifichiamo con τ la soglia entro cui i pesi verranno eliminati, la nuova matrice sarà definita come

$$\bar{w}_{ij} = \begin{cases} 0 & \text{se } |w_{ij}| < \tau, \\ w_{ij} & \text{altrimenti.} \end{cases} \quad (2.3)$$

Abbiamo scelto τ come il quantile q distribuzione del valore assoluto dei pesi, dove q assume valori in $[0,1]$.

2.1.3 Tasso di compressione

Il tasso di compressione serve a quantificare il risparmio in termini di spazio di memoria che si ottiene applicando la compressione della matrice dei pesi. Il tasso di compressione r_1 sarà calcolato come

$$r_1 = \frac{s'}{s}, \quad (2.4)$$

dove s' rappresenta lo spazio occupato dalle connessioni della rete compressa utilizzando la rappresentazione matriciale CSC e s lo spazio occupato dalle connessioni della rete senza compressione. Come ci accorgeremo nel Paragrafo 3.2.1, si ottiene $r_1 < 1$ raggiungendo una certa sparsità².

Per calcolare lo spazio occupato da una matrice espansa utilizziamo la formula $s = \# \text{righe} \cdot \# \text{colonne} \cdot b$, dove b è il numero di bit utilizzati per rappresentare un numero in binario. Invece, per calcolare lo spazio occupato dalla rappresentazione matriciale CSC utilizziamo la formula citata nel paragrafo precedente moltiplicata per b , ovvero $(2a + c + 1) \cdot b$. La matrice 2.1 occupa, quindi, $7 \cdot 7 \cdot 32 = 1568 \text{bit}$ mentre la sua rappresentazione matriciale $(2 \cdot 10 + 7 + 1) \cdot 32 = 896 \text{bit}$. Il tasso di compressione è $r_1 = \frac{896 \text{bit}}{1568 \text{bit}} = 0.54$.

2.2 Weight Sharing

In modo analogo a quanto visto per il pruning, la tecnica del weight sharing [2] viene utilizzata per ridurre lo spazio occupato per salvare le matrici dei pesi della rete neurale. Questa procedura consiste nel raggruppamento dei pesi simili, presi da una rete precedentemente addestrata, attraverso un algoritmo di clustering. Dopo aver suddiviso i pesi in cluster e definito un centroide per ogni cluster, tutti i pesi vengono sostituiti nella rete. Infine si riaddestrano i pesi mediante una variante dell'algoritmo di back propagation che aggiorna solo i centroidi, come descritto in dettaglio nel Paragrafo 2.2.2.

2.2.1 Strutture dati necessarie

Per la gestione di questa procedura viene salvato un array che contiene i valori dei centroidi e una matrice per salvare la corrispondenza peso-centroide (per ogni strato). Denotiamo con C

²percentuale di elementi uguali a zero nelle matrici

il vettore dei centroidi e con N la matrice delle corrispondenze, quindi

$$n_{ij} = \arg \min_k |c_k - w_{ij}|. \quad (2.5)$$

$$\begin{pmatrix} 3.1 & 7.1 & 6.9 & 2.6 & 12.3 \\ 2.9 & 0.1 & 0.0 & 4.5 & 0.0 \\ 21.0 & 21.7 & 5.0 & 3.4 & 0.2 \\ 4.4 & 0.0 & 4.6 & 0.3 & 0.0 \\ 1.0 & 0.2 & 22 & 5.1 & 12.4 \end{pmatrix} \quad (2.6)$$

Prendendo come esempio la matrice 2.6 e impostando $k = 6$, C e N risulterebbero

$$C = \begin{bmatrix} 0 \\ 3 \\ 5 \\ 7 \\ 12 \\ 22 \end{bmatrix}, \quad N = \begin{pmatrix} 1 & 3 & 3 & 1 & 4 \\ 1 & 0 & 0 & 2 & 0 \\ 5 & 5 & 5 & 3 & 0 \\ 2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 5 & 2 & 5 \end{pmatrix}. \quad (2.7)$$

2.2.2 Tecniche implementative

Alla rete neurale base vengono aggiunte due procedure, applicate a ciascuno strato separatamente:

- una procedura che crea il vettore C contenente i k centroidi, dove k è il numero di cluster scelti;
- una procedura che crea la matrice N , i cui elementi sono definiti in (2.5).

Alla normale fase di training vengono aggiunte quindi due procedure:

- una procedura che costruisce la matrice dei pesi effettiva \bar{W} per il calcolo degli output della rete (vedi Paragrafo 1.5.1) con i valori dei centroidi invece dei valori originali $\bar{w}_{ij} = c_{n_{ij}}$;
- una procedura per riaddestrare i centroidi ottenuti tramite il *cumulative gradient descent* [2] che consiste nel calcolare il gradiente della funzione di errore \mathcal{L} rispetto a ciascun centroide e non più rispetto ai singoli pesi. Specificatamente, il gradiente è calcolato come

$$\frac{\partial \mathcal{L}}{\partial c_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial w_{ij}} 1(n_{ij} = k). \quad (2.8)$$

Per la matrice 2.6

$$W' = \begin{pmatrix} 3 & 7 & 7 & 3 & 12 \\ 3 & 0 & 0 & 5 & 0 \\ 22 & 22 & 5 & 3 & 0 \\ 5 & 0 & 5 & 0 & 0 \\ 0 & 0 & 22 & 5 & 12 \end{pmatrix} \quad (2.9)$$

2.2.3 Tasso di compressione

Quantificare la compressione apportata dalla tecnica di weight sharing risulta meno immediato rispetto a quanto indicato per il pruning. Bisogna infatti prendere in considerazione il numero di cluster e il numero di bit gli elementi delle matrici delle connessioni e i centroidi dei cluster.

Se denotiamo con b il numero di bit con cui viene rappresentato un peso della rete e con s il numero di connessioni nella rete, il tasso di compressione teorico $r_2 \in [0, 1]$ viene calcolato come

$$r_2 = \frac{s \lceil \log_2 k \rceil + kb}{sb}. \quad (2.10)$$

In analogia con r_1 , r_2 rappresenta la percentuale di spazio occupato dalla rete compressa rispetto a quello originario. Gli esperimenti descritti nel Capitolo 3 fanno riferimento a una codifica che utilizza 8 o 16 bit per i numeri interi senza segno e 32 bit per i valori a virgola mobile, e in questa configurazione il tasso di compressione effettivo diventa

$$r_2 = \frac{sf(s) + 32k}{32s}, \quad f(s) = \begin{cases} 16 & \text{se } s \geq 256, \\ 8 & \text{altrimenti.} \end{cases} \quad (2.11)$$

Il tasso di compressione della matrice 2.6 risulta essere

$$s = 5 \cdot 5 = 25, \quad f(s) = 8 \quad \text{quindi } r_2 = \frac{25 \cdot 8 + 32 \cdot 6}{32 \cdot 25} = 0.49 \quad (2.12)$$

Capitolo 3

Esperimenti

In questo capitolo illustreremo gli esperimenti eseguiti su due problemi differenti:

- cifre MNIST: problema di classificazione in cui il dataset è composto da un insieme di immagini che rappresentano cifre scritte a mano,
- problema del predecessore: problema di regressione in cui i dataset sono composti da sequenze di numeri rappresentati in 64 bit.

3.1 MNIST

Il dataset è una vasta base di dati di cifre scritte a mano, spesso impiegata nel campo dell'apprendimento automatico (*machine learning*). Il dataset contiene 60000 immagini di training e 10000 immagini di testing delle quali nella Figura 3.1 vengono mostrati alcuni esempi.

3.1.1 Addestramento e tuning parametri

Per trovare una buona configurazione della rete abbiamo effettuato una *k-Fold Cross Validation* [6] con $k = 3$. Abbiamo inoltre utilizzato la tecnica di dropout per mitigare gli effetti di un possibile overfitting durante il processo di apprendimento.

K-Fold Cross Validation Il *training set* viene diviso in k sottoinsiemi, e a rotazione vengono usati come *training set* $k - 1$ sottoinsiemi, il rimanente sottoinsieme viene usato come *validation set*. Quest'ultimo serve per misurare la bontà della rete ottenuta fissando una configurazione



Figura 3.1: Esempi MNIST [11]

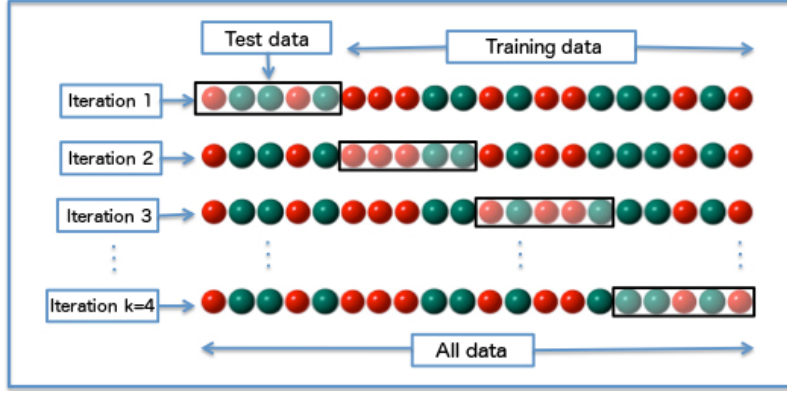


Figura 3.2: Suddivisione di un dataset in un processo di cross validation a quattro fold

dei parametri e addestrando sul training set. Alla fine si sceglie la rete che ha la bontà più alta e si usa il test set per valutare la capacità di generalizzazione. Nella Figura 3.2 possiamo vedere un esempio di splitting del *training set* con $k = 4$.

Negli esperimenti abbiamo fissato i seguenti parametri:

- learning rate $\eta = 3 \times 10^{-3}$;
- funzione errore: errore quadratico medio¹;
- metodo di apprendimento: discesa del gradiente con momentum ($\mu = 0.99$);
- funzione di attivazione degli strati *hidden*: *ReLU*;
- funzione di attivazione dello strato di output: *softmax*²;
- dimensione dei minibatch: 100 elementi;
- criterio di arresto: 100 epoche;
- inizializzazione dei pesi: generazione di numeri casuali (in particolare, ogni peso dello strato l è impostato a $z\sqrt{2/\text{size}^{l-1}}$, dove z è un valore pseudocasuale estratto da una distribuzione normale standard e size^{l-1} indica il numero di neuroni nello strato precedente³).

I parametri di cui vogliamo trovare i valori migliori sono invece:

- architettura della rete (ovvero quanti strati hidden e quanti neuroni per ogni strato hidden);
- dropout (percentuale di neuroni negli strati hidden non utilizzati nella fase di training).

Dropout Il *dropout* [8] è una tecnica utilizzata per evitare l'overfitting mediante l'eliminazione casuale di alcuni neuroni in ogni strato, a eccezione di quello di output. I neuroni da eliminare

¹ $\frac{1}{n} \sum_{i=1}^n (t_i - y_i)^2$, abbreviata negli ultimi esperimenti di questo capitolo con MSE

² La funzione softmax è definita come $\text{softmax}(a_i) = \frac{\exp^{a_i}}{\sum_j \exp^{a_j}}$

³ He et al initialization [3]

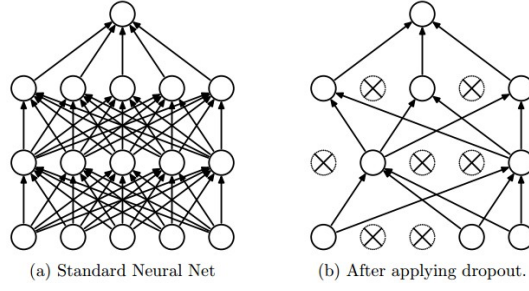


Figura 3.3: Visualizzazione della tecnica di dropout. (a) struttura di una rete neurale a tre strati. (b) modifica temporanea della struttura della rete durante un'iterazione dell'algoritmo di apprendimento, in cui i neuroni indicati con una croce non concorrono all'aggiornamento dei pesi della rete. [8]

cambiano a ogni epoca, durante la fase di predizione invece sono tutti attivi. Nella Figura 3.3 possiamo vedere un esempio di questa tecnica. Nell'elaborato è stata utilizzata la versione chiamata *inverted dropout*

$$\text{drop}_l = (\text{rand} < p)/p, \quad \text{output}_l = \text{output}_l * \text{drop}_l, \quad (3.1)$$

dove:

- output_l rappresenta l'output di un generico strato l
- rand denota una matrice di numeri casuali estratti da una distribuzione uniforme nell'intervallo $[0, 1)$ delle stesse dimensioni di output_l ,
- p rappresenta la probabilità di tenere attivo un neurone,
- $*$ indica l'operatore di prodotto componente per componente tra due vettori.

Nelle Tabelle 3.1 e 3.2 sono indicati i risultati. Negli esperimenti effettuati abbiamo trovato come configurazione ottimale una rete MLP con uno strato hidden da 300 neuroni senza dropout (indicato nell'intestazione della tabella come $p = 1$) che è anche la configurazione che abbiamo adottato negli esperimenti descritti di seguito.

3.1.2 Pruning

Applicando la tecnica di pruning alla rete ottenuta dopo la procedura di model selection descritta nel paragrafo precedente abbiamo ottenuto i risultati descritti nella Tabella 3.3. In particolare abbiamo applicato un pruning dal 10% al 95% a passi di 5%. Nella prima riga della Tabella 3.3 abbiamo riportato l'accuratezza sul *test set* del modello scelto nel paragrafo precedente senza compressioni. Questi risultati da un lato mostrano come nonostante la rimozione di numerose connessioni (fino al 80%) la rete appresa sia in grado di raggiungere accuratèzze maggiori rispetto a quella della rete che le usa tutte; dall'altro lato, che la rete iniziale era sovradimensionata per il problema.

Tabella 3.1: Risultati della model selection per un MLP con uno strato hidden. La colonna **neuroni** indica il numero di neuroni nello strato hidden, la colonna **accuratezza** indica la media delle accuratèzze sui tre *validation set*, mentre p denota la probabilità di tenere attivo un generico neurone durante la fase di dropout. In grassetto è evidenziata la configurazione migliore.

	$p = 0.75$	$p = 1$
neuroni	accuratezza	
100	97.64	97.38
125	97.63	97.57
150	97.72	94.41
175	97.75	97.81
200	97.83	94.29
225	97.82	94.75
250	97.82	94.88
275	97.84	95.25
300	97.86	98.08

Tabella 3.2: Risultati della model selection per un MLP con due strati hidden. Stesse notazioni della Tabella 3.1 a eccezione della colonna **neuroni** che rappresenta il numero di neuroni dei due strati hidden.

	$p = 0.75$	$p = 1$		$p = 0.75$	$p = 1$
neuroni	accuratezza		neuroni	accuratezza	
50 - 25	97.14	97.08	175 - 125	97.85	97.80
50 - 50	97.26	96.82	175 - 150	97.82	97.86
75 - 25	97.43	97.43	175 - 175	97.85	97.84
75 - 50	97.52	97.51	200 - 50	97.83	97.83
75 - 75	97.51	97.49	200 - 75	97.83	97.90
100 - 25	97.53	97.54	200 - 100	97.88	97.89
100 - 50	97.66	97.62	200 - 125	97.91	97.85
100 - 75	97.62	97.63	200 - 150	97.84	97.85
100 - 100	97.67	97.67	200 - 175	97.84	97.89
125 - 25	97.68	97.62	200 - 200	97.86	97.83
125 - 50	97.77	97.70	225 - 75	97.92	97.90
125 - 75	97.72	97.68	225 - 100	97.86	97.79
125 - 100	97.73	97.74	225 - 125	97.85	97.85
125 - 125	97.74	97.77	225 - 150	97.94	97.94
150 - 50	97.71	97.80	225 - 175	97.88	97.89
150 - 75	97.76	97.76	225 - 200	97.91	97.90
150 - 100	97.75	97.85	250 - 75	97.86	97.88
150 - 125	97.78	97.74	250 - 100	97.91	97.91
150 - 150	97.75	97.80	250 - 125	97.93	97.94
175 - 50	97.85	97.80	250 - 150	97.93	97.90
175 - 75	97.80	97.80	250 - 175	97.86	97.90
175 - 100	97.83	97.80	250 - 200	97.89	97.92

Tabella 3.3: Variazione dell'accuratezza della rete che ha appreso a classificare le cifre MNIST applicando la tecnica di pruning

pruning %	accuratezza	r_1	pruning %	accuratezza	r_1
ALL	98.34		55%	98.32	0.9
10%	98.34	1.8	60%	98.30	0.8
15%	98.34	1.7	65%	98.33	0.7
20%	98.34	1.6	70%	98.34	0.6
25%	98.35	1.5	75%	98.34	0.5
30%	98.36	1.4	80%	98.38	0.4
35%	98.35	1.3	85%	98.31	0.3
40%	98.36	1.2	90%	98.27	0.2
45%	98.36	1.1	95%	98.28	0.1
50%	98.36	1.0			

Tabella 3.4: Variazione dell'accuratezza della rete che ha appreso a classificare le cifre MNIST applicando la tecnica di weight sharing

centroidi	accuratezza	r_2	centroidi	accuratezza	r_2
ALL	98.34		256 - 64	98.36	0.5
4 - 2	94.00	0.5	512 - 32	98.38	0.5
16 - 8	98.14	0.5	512 - 64	98.35	0.5
32 - 8	98.32	0.5	1024 - 32	98.37	0.5
32 - 16	98.25	0.5	1024 - 64	98.37	0.5
64 - 16	98.29	0.5	2048 - 32	98.36	0.51
64 - 32	98.34	0.5	2048 - 64	98.36	0.51
128 - 32	98.38	0.5	4096 - 32	98.35	0.52
192 - 32	98.38	0.5	4096 - 64	98.37	0.52
192 - 64	98.36	0.5	8192 - 32	98.43	0.53
256 - 32	98.31	0.5	8192 - 64	98.34	0.53

3.1.3 Weight sharing

La Tabella 3.4 riporta i risultati dell'applicazione della tecnica di weight sharing alla rete ottenuta nel Paragrafo 3.1.1. La prima colonna rappresenta il numero di centroidi utilizzati per la compressione, specificati separatamente per i due strati della rete. Nella prima riga della Tabella 3.4 abbiamo riportato l'accuratezza sul *test set* del modello scelto senza compressioni. Come avvenuto con il pruning, possiamo notare che l'accuratezza rimane sopra il 98% anche considerando un numero abbastanza piccolo di centroidi. Nella colonna r_2 possiamo notare valori molto vicini al variare dei cluster, riprendendo (2.11): s (che ricordiamo essere il prodotto delle dimensioni della matrice) è sempre maggiore di 256, quindi $f(s) = 16$; semplificando la formula troviamo

$$\frac{1}{2} + \frac{k}{s},$$

per questo motivo r_2 non può essere minore di 0.5.

3.2 Problema del predecessore

Fissato un universo U , sul quale è definita una relazione di ordine totale \leq , consideriamo un sottoinsieme $X \subseteq U$ i cui elementi chiameremo *chiavi*. Il problema di ricerca del predecessore consiste nel trovare la posizione della chiave più grande in X che non sia maggiore di un valore x dato in input. Se per ogni $x \in X$ indichiamo con $\text{pos}(x)$ la posizione di x nella sequenza ottenuta ordinando gli elementi di X , il problema equivale a determinare il valore $\text{pred}(x) = \text{pos}(z)$, dove $z = \text{argmax}_y \in X \{y \leq x\}$. Se per esempio consideriamo l'insieme

$$X = \{2, 3, 4, 5, 12, 15, 18\}$$

e assumiamo $x = 7$, la ricerca del predecessore dovrebbe restituire la posizione 4.

Dato $|X| = n$, sia $F_X : U \mapsto [0, 1]$ la distribuzione cumulativa empirica degli elementi di U rispetto a X , per ogni $x \in U$, $F_X(x) = \frac{|\{x' \in X \text{ tale che } x' \leq x\}|}{n}$. Per semplificare la notazione, denotiamo F_X con F . Nell'esempio sopra indicato, si avrà dunque $F(2) = \frac{1}{7}$, $F(5) = \frac{4}{7}$, $F(6) = \frac{4}{7}$, $F(18) = 1$. La conoscenza di F ci dà una soluzione al nostro problema, in quanto dato un generico $x \in U$, una sola valutazione di F fornisce $\text{pred}(x) = \lceil (F(x) \cdot n) \rceil$. Il nostro obiettivo è quindi quello di trovare una rete neurale che calcoli una buona approssimazione \tilde{F} di F . La risoluzione di questo problema potrebbe consentire di trovare un elemento all'interno di una lista ordinata in tempo $O(1)$ invece che $O(\log n)$ tipico degli alberi binari di ricerca. La rete neurale però può compiere un errore nel dare la posizione di un elemento, per questo motivo teniamo traccia dell'errore massimo, indicato con ϵ , che la rete compie. Pertanto durante la ricerca, dopo che il modello ha predetto la posizione, dovremo effettuare una ricerca che coinvolgerà al più ϵ posizioni a destra e a sinistra rispetto a quanto indicato dal modello. Per questo motivo, dopo aver utilizzato delle funzioni di errore standard (come l'errore quadratico medio) che minimizzano l'errore medio, negli esperimenti abbiamo provato ad allenare la rete neurale minimizzando una funzione convessa che approssima il massimo. Tale funzione prende il nome di LogSumExp è descritta nel Paragrafo 3.2.4.

Per questo problema sono stati utilizzati tre dataset, rispettivamente con 512, 8192 e 1048576 esempi. Questi dataset saranno successivamente indicati rispettivamente come dataset 3, dataset 7 e dataset 10. Diversamente da MNIST, il modello MLP cercherà di risolvere un problema di regressione: la rete neurale dovrà imparare a restituire, data una chiave, il corrispondente valore di \tilde{F} . Sono state usate tre reti differenti:

- rete 1 con zero strati hidden;
- rete 2 con uno strato hidden di 256 neuroni;
- rete 3 con due strati hidden di 256 neuroni.

Per tutte e tre le reti, dove non indicato esplicitamente, i parametri utilizzati sono:

- funzione errore: errore quadratico medio;
- metodo di apprendimento: discesa del gradiente con momentum ($\mu = 0.9$);
- funzione di attivazione dei neuroni: *LeakyReLU* ($\alpha = 0.05$);
- $\lambda = 10^{-5}$ ⁴;
- dimensione dei minibatch: 64;

⁴usato per nella regolarizzazione l2, valore aggiunto alla funzione di errore che penalizza i pesi più grandi, riducendo così il problema di overfitting [9].

- epoche: 20000;
- criterio di arresto: l'apprendimento termina se non si hanno miglioramenti di performance con *patience* = 4 (vedi Paragrafo 1.5.2);
- inizializzazione dei pesi = numeri casuali estratti da una distribuzione normale con $\mu = 0$ e $\sigma = 0.05$.

Ci riferiremo a queste reti con NNX, dove X indica il numero totale di strati (compreso quello di output). Per NN1 è stato usato un learning rate $\eta = 5 \times 10^{-4}$, mentre per NN2 e NN3 si ha $\eta = 3 \times 10^{-3}$. Questi valori sono stati trovati eseguendo un tuning su η .

I risultati ottenuti sono indicati nelle Tabelle 3.5 - 3.21, che riportano le seguenti colonne:

- Pruning %: percentuale di connessioni eliminate (calcolate sulla base di quantili come indicato nel Paragrafo 2.1.2); in particolare, 0 indica la rete non compressa;
- r_2 : proporzione dello spazio originario occupato da quella compressa (1 indica la rete non compressa);
- Clusters: numero di cluster utilizzati per ogni matrice dei pesi;
- Space Overhead (KB): spazio del modello (in KB) rispetto al dataset, calcolata come

$$\frac{\sum_{i=1}^n (|\text{strato}_i|) \cdot \text{bytes} \cdot 100}{1024 \cdot |\text{dataset}|}$$

dove per strato_i viene preso in considerazione il prodotto delle dimensioni della matrice delle connessioni per memorizzare i pesi per l' i -esimo strato, bytes rappresenta il numero di bytes utilizzato per rappresentare i pesi;

- Training Time (s): tempo (in secondi) impiegato dalla rete per il training, compreso il tempo iniziale di applicazione della compressione (nella rete non compressa non viene indicato questo tempo, questi casi sono indicati con -);
- ϵ : errore massimo sugli esempi di training;
- Error %: errore massimo rispetto alla dimensione del dataset considerato;
- Mean Error: errore medio.

3.2.1 Pruning

Le Tabelle 3.5 - 3.13 mostrano i risultati dell'applicazione della tecnica di pruning alle reti ottenute per il problema del predecessore. Come accennato nel Paragrafo 2.1.3 prima di raggiungere una certa sparsità il modello compresso occupa più spazio del modello originario. I modelli compressi occupano meno spazio a partire da una percentuale di pruning compresa tra il 50% e il 60%. Nei risultati di questi esperimenti possiamo notare una scarsa efficacia del pruning in reti con pochi neuroni (NN1), mentre (come già visto con MNIST nel Paragrafo 3.1.2) con molti neuroni (NN2 e NN3) il pruning a percentuali molto elevate mantiene errori medi ed errori massimi equivalenti alla rete non compressa, se non migliori. Infatti sul dataset 10 si riesce ad arrivare a tassi di compressione maggiori, fino a 80%, migliorando l'errore. Possiamo osservare che, per la natura del problema (le posizioni rappresentano una successione monotona non decrescente), la rete NN1 risulta migliore per questo problema (il rapporto prestazioni/spazio è sempre a suo favore).

Tabella 3.5: NN1 dataset 3

Pruning %	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
0	8.93×10^{-2}	-	9	1.76	4.74×10^{-3}
10	8.56×10^{-2}	7.0×10^{-3}	8	1.56	4.58×10^{-3}
20	8.01×10^{-2}	7.6×10^{-3}	8	1.56	4.58×10^{-3}
30	7.10×10^{-2}	7.5×10^{-3}	8	1.56	4.59×10^{-3}
40	6.03×10^{-2}	7.2×10^{-3}	9	1.76	4.82×10^{-3}
50	5.11×10^{-2}	8.9×10^{-3}	14	2.73	8.91×10^{-3}
60	4.20×10^{-2}	8.9×10^{-3}	21	4.1	1.60×10^{-2}
70	3.13×10^{-2}	1.2×10^{-2}	36	7.03	3.13×10^{-2}
80	2.21×10^{-2}	1.1×10^{-2}	66	1.28×10	6.25×10^{-2}
90	1.30×10^{-2}	1.1×10^{-2}	65	1.27×10	6.25×10^{-2}

Tabella 3.6: NN1 dataset 7

Pruning %	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
0	3.09×10^{-3}	-	53	6.4×10^{-1}	1.62×10^{-3}
10	5.58×10^{-3}	8.1×10^{-2}	46	5.6×10^{-1}	1.55×10^{-3}
20	5.01×10^{-3}	8.1×10^{-2}	43	5.2×10^{-1}	1.52×10^{-3}
30	4.43×10^{-3}	8.9×10^{-2}	44	5.4×10^{-1}	1.53×10^{-3}
40	3.77×10^{-3}	9.3×10^{-2}	55	6.7×10^{-1}	1.77×10^{-3}
50	3.19×10^{-3}	1.2×10^{-1}	73	8.9×10^{-1}	2.41×10^{-3}
60	2.62×10^{-3}	1.2×10^{-1}	107	1.31	4.17×10^{-3}
70	1.96×10^{-3}	1.7×10^{-1}	107	1.31	4.17×10^{-3}
80	1.38×10^{-3}	2.4×10^{-1}	165	2.01	7.97×10^{-3}
90	8.11×10^{-4}	2.2×10^{-1}	165	2.01	7.97×10^{-3}

Tabella 3.7: NN1 dataset 10

Pruning %	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
0	2.42×10^{-5}	-	905	8.6×10^{-2}	1.85×10^{-4}
10	4.35×10^{-5}	1.1×10	770	7×10^{-2}	1.52×10^{-4}
20	3.91×10^{-5}	1.6×10	770	7×10^{-2}	1.52×10^{-4}
30	3.46×10^{-5}	1.1×10	770	7×10^{-2}	1.52×10^{-4}
40	2.94×10^{-5}	1.1×10	770	7×10^{-2}	1.52×10^{-4}
50	2.49×10^{-5}	1.4×10	770	7×10^{-2}	1.52×10^{-4}
60	2.05×10^{-5}	1.2×10	734	7×10^{-2}	1.50×10^{-4}
70	1.53×10^{-5}	1.1×10	736	7×10^{-2}	1.51×10^{-4}
80	1.08×10^{-5}	1.1×10	748	7×10^{-2}	1.51×10^{-4}
90	6.32×10^{-6}	1.1×10	5235	4.9×10^{-1}	1.97×10^{-4}

Tabella 3.8: NN2 dataset 3

Pruning %	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
0	1.289×10	-	4	7.8×10^{-1}	1.60×10^{-3}
10	2.324×10	2.1×10	2	3.9×10^{-1}	6.12×10^{-4}
20	2.070×10	2.3×10	2	3.9×10^{-1}	6.21×10^{-4}
30	1.817×10	2.1×10	3	5.8×10^{-1}	6.93×10^{-4}
40	1.563×10	2.0×10	3	5.8×10^{-1}	7.93×10^{-4}
50	1.309×10	2.1×10	3	5.8×10^{-1}	8.62×10^{-4}
60	1.055×10	2.3×10	3	5.8×10^{-1}	9.85×10^{-4}
70	8.01	2.1×10	4	7.8×10^{-1}	1.32×10^{-3}
80	5.47	1.7×10	5	9.7×10^{-1}	1.95×10^{-3}
90	2.93	1.2×10	6	1.17	2.78×10^{-3}

3.2.2 Weight Sharing

Le Tabelle 3.14 - 3.21 mostrano i risultati dell'applicazione della tecnica di weight sharing alle reti ottenute per il problema del predecessore. Il numero di centroidi è calcolato come

$$\text{cluster} = \left(\frac{r_2 \times 32s - sb}{32} \right) \quad (3.2)$$

dove r_2 è il tasso di compressione (vedi Paragrafo 2.2.3), s è il numero di connessioni e b è il numero di byte utilizzati per rappresentare i centroidi. Per (3.2) e (2.11) non possiamo

Tabella 3.9: NN2 dataset 7

Pruning %	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
0	8.06×10^{-1}	-	33	4.0×10^{-1}	1.12×10^{-3}
10	1.45	1.31×10^2	35	4.3×10^{-1}	5.94×10^{-4}
20	1.29	1.23×10^2	35	4.3×10^{-1}	5.98×10^{-4}
30	1.13	1.18×10^2	35	4.3×10^{-1}	6.04×10^{-4}
40	9.76×10^{-1}	1.18×10^2	35	4.3×10^{-1}	5.97×10^{-4}
50	8.18×10^{-1}	1.15×10^2	34	4.2×10^{-1}	5.99×10^{-4}
60	6.59×10^{-1}	1.11×10^2	35	4.3×10^{-1}	6.11×10^{-4}
70	5.00×10^{-1}	1.24×10^2	35	4.3×10^{-1}	6.12×10^{-4}
80	3.42×10^{-1}	2.2×10	33	4.0×10^{-1}	8.56×10^{-4}
90	1.83×10^{-1}	5	34	4.1×10^{-1}	1.19×10^{-3}

Tabella 3.10: NN2 dataset 10

Pruning %	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
0	6.29×10^{-3}	-	1031	9.0×10^{-2}	2.33×10^{-4}
10	1.13×10^{-2}	1.74×10^2	713	6.8×10^{-2}	1.46×10^{-4}
20	1.01×10^{-2}	1.25×10^2	713	6.8×10^{-2}	1.46×10^{-4}
30	8.87×10^{-3}	1.25×10^2	713	6.8×10^{-2}	1.46×10^{-4}
40	7.63×10^{-3}	1.24×10^2	713	6.8×10^{-2}	1.46×10^{-4}
50	6.39×10^{-3}	1.22×10^2	713	6.8×10^{-2}	1.46×10^{-4}
60	5.15×10^{-3}	1.22×10^2	713	6.8×10^{-2}	1.46×10^{-4}
70	3.91×10^{-3}	1.22×10^2	713	6.8×10^{-2}	1.46×10^{-4}
80	2.67×10^{-3}	1.22×10^2	715	6.8×10^{-2}	1.46×10^{-4}
90	1.43×10^{-3}	1.23×10^2	699	6.7×10^{-2}	1.45×10^{-4}

Tabella 3.11: NN3 dataset 3

Pruning %	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
0	6.308×10	-	4	7.8	1.83×10^{-3}
10	1.13×10^2	6.1×10	2	3.9×10^{-1}	6.63×10^{-4}
20	1.01×10^2	6.3×10	2	3.9×10^{-1}	6.86×10^{-4}
30	8.855×10	5.6×10	2	3.9×10^{-1}	7.66×10^{-4}
40	7.601×10	5.6×10	3	5.8×10^{-1}	8.79×10^{-4}
50	6.348×10	6.6×10	3	5.8×10^{-1}	8.82×10^{-4}
60	5.094×10	6.6×10	3	5.8×10^{-1}	1.02×10^{-3}
70	3.840×10	7.1×10	3	5.8×10^{-1}	1.15×10^{-3}
80	2.586×10	6.9×10	3	5.8×10^{-1}	1.50×10^{-3}
90	1.332×10	8.1×10	4	7.8×10^{-1}	2.03×10^{-3}

Tabella 3.12: NN3 dataset 7

Pruning %	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
0	3.94	-	40	4.8×10^{-1}	1.14×10^{-3}
10	7.10	1.97×10^2	36	4.3×10^{-1}	7.80×10^{-4}
20	6.31	2.27×10^2	36	4.3×10^{-1}	7.58×10^{-4}
30	5.53	9.5×10	32	3.9×10^{-1}	8.18×10^{-4}
40	4.75	1.04×10^2	33	4.0×10^{-1}	8.18×10^{-4}
50	3.97	9.6×10	33	4.0×10^{-1}	8.23×10^{-4}
60	3.18	1.25×10^2	34	4.1×10^{-1}	8.12×10^{-4}
70	2.40	9.9×10	34	4.1×10^{-1}	8.29×10^{-4}
80	1.61	9.3×10	34	4.1×10^{-1}	8.20×10^{-4}
90	8.32×10^{-1}	2.9×10	32	3.9×10^{-1}	1.24×10^{-3}

Tabella 3.13: NN3 dataset 10

Pruning %	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
0	3.08×10^{-2}	-	1270	1.2×10^{-1}	2.40×10^{-4}
10	5.54×10^{-2}	3.02×10^2	708	6.8×10^{-2}	1.47×10^{-4}
20	4.93×10^{-2}	3.03×10^2	708	6.8×10^{-2}	1.47×10^{-4}
30	4.32×10^{-2}	3.04×10^2	708	6.8×10^{-2}	1.47×10^{-4}
40	3.71×10^{-2}	3.05×10^2	707	6.8×10^{-2}	1.47×10^{-4}
50	3.09×10^{-2}	3.04×10^2	707	6.7×10^{-2}	1.47×10^{-4}
60	2.49×10^{-2}	3.00×10^2	707	6.7×10^{-2}	1.47×10^{-4}
70	1.87×10^{-2}	3.00×10^2	706	6.7×10^{-2}	1.47×10^{-4}
80	1.26×10^{-2}	3.02×10^2	700	6.7×10^{-2}	1.46×10^{-4}
90	6.50×10^{-3}	3.00×10^2	660	6.3×10^{-2}	1.45×10^{-4}

Tabella 3.14: NN1 dataset 3

r_2	Clusters	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
	-	4.95×10^{-2}	-	9	1.75	4.74×10^{-3}
0.3	3	1.60×10^{-2}	8.07×10^{-2}	2427	4.74×10^2	3.15
0.4	9	2.06×10^{-2}	8.69×10^{-2}	14373	2.80×10^3	1.78×10
0.5	16	2.59×10^{-2}	9.27×10^{-2}	5714	1.11×10^3	7.32
0.6	22	3.05×10^{-2}	1.93×10^{-1}	5296	1.03×10^3	6.80
0.7	28	3.51×10^{-2}	1.93×10^{-1}	8159	1.59×10^3	1.03×10
0.8	35	4.04×10^{-2}	2.08×10^{-1}	542	1.05×10^2	5.83×10^{-1}
0.9	41	4.50×10^{-2}	2.44×10^{-1}	8	1.56	4.60×10^{-3}

Tabella 3.15: NN1 dataset 7

r_2	Clusters	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
	-	3.09×10^{-3}	-	53	6.47×10^{-1}	1.62×10^{-3}
0.3	3	1.00×10^{-3}	2.71×10^{-1}	57×10^7	69.62×10^5	4.27×10^3
0.4	9	1.29×10^{-3}	2.71×10^{-1}	37×10^4	46.34×10^2	2.71×10
0.5	16	1.62×10^{-3}	3.02×10^{-1}	10×10^4	12.54×10^2	7.20
0.6	22	1.91×10^{-3}	3.55×10^{-1}	21×10^4	25.76×10^2	1.46×10
0.7	28	2.19×10^{-3}	3.55×10^{-1}	5528	6.74×10	2.44×10^{-1}
0.8	35	2.52×10^{-3}	3.98×10^{-1}	63	7.7×10^{-1}	2.26×10^{-3}
0.9	41	2.81×10^{-3}	4.71×10^{-1}	56	6.8×10	1.82×10^{-3}

raggiungere valori di r_2 inferiori a quelli mostrati nelle tabelle. Come ci si aspetta, al crescere del numero di centroidi le reti hanno una performance crescente e in alcuni casi migliore della rete non compressa. Anche da questi esperimenti NN1 si fa preferire per prestazioni e spazio occupato.

Tabella 3.16: NN1 dataset 3

r_2	Clusters	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
	-	2.42×10^{-5}	-	905	8.6×10^{-2}	1.85×10^{-4}
0.3	3	7.82×10^{-6}	3.8×10	2.22×10^5	2.12×10^2	1.11
0.4	9	1.00×10^{-5}	1.8×10	1.51×10^4	1.44×10	4.75×10^{-2}
0.5	16	1.26×10^{-5}	1.9×10	2.29×10^4	2.19×10	7.54×10^{-2}
0.6	22	1.49×10^{-5}	1.8×10	717	6.8×10^{-2}	1.49×10^{-4}
0.7	28	1.71×10^{-5}	1.9×10	715	6.8×10^{-2}	1.48×10^{-4}
0.8	35	1.97×10^{-5}	1.9×10	706	6.7×10^{-2}	1.48×10^{-4}
0.9	41	2.20×10^{-5}	1.9×10	706	6.7×10^{-2}	1.48×10^{-4}

Tabella 3.17: NN2 dataset 3

r_2	Clusters	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
	-	1.28×10	-	4	7.8×10^{-1}	1.6×10^{-3}
0.6	1638, 89	8.01	2.8×10	5	9.7×10^{-1}	2.21×10^{-3}
0.7	3276, 115	9.28	3.4×10	2	3.9×10^{-1}	5.70×10^{-4}
0.8	4915, 140	1.05×10	4.3×10	2	3.9×10^{-1}	5.43×10^{-4}
0.9	6553, 166	1.18×10	4.8×10	2	3.9×10^{-1}	5.48×10^{-4}

Tabella 3.18: NN2 dataset 7

r_2	Clusters	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
	-	8.05×10^{-1}	-	33	4.0×10^{-1}	1.12×10^{-3}
0.6	1638, 89	5.00×10^{-1}	1.71×10^2	33	4.0×10^{-1}	6.13×10^{-4}
0.7	3276, 115	5.80×10^{-1}	2.34×10^2	33	4.0×10^{-1}	5.95×10^{-4}
0.8	4915, 140	6.59×10^{-1}	2.80×10^2	33	4.0×10^{-1}	5.82×10^{-4}
0.9	6553, 166	7.38×10^{-1}	3.17×10^2	34	4.1×10^{-1}	5.90×10^{-4}

Tabella 3.19: NN2 dataset 10

r_2	Clusters	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
	-	6.29×10^{-3}	-	1031	9.8×10^{-2}	2.33×10^{-4}
0.6	1638, 89	3.91×10^{-3}	2.64×10^2	292	2.8×10^{-2}	5.11×10^{-5}
0.7	3276, 115	4.53×10^{-3}	2.44×10^2	753	7.2×10^{-2}	1.49×10^{-4}
0.8	4915, 140	5.15×10^{-3}	2.82×10^2	714	6.8×10^{-2}	1.45×10^{-4}
0.9	6553, 166	5.77×10^{-3}	3.17×10^2	684	6.5×10^{-2}	1.45×10^{-4}

NN3 dataset 3

r_2	Clusters	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
	-	6.31×10	-	4	7.8×10^{-1}	1.83×10^{-3}
0.6	1638, 6553, 89	3.84×10	2.00×10	3	5.8×10^{-1}	1.25×10^{-3}
0.7	3276, 13107, 115	4.46×10	1.43×10^2	2	3.9×10^{-1}	6.28×10^{-4}
0.8	4915, 19660, 140	5.09×10	1.87×10^2	4	7.8×10^{-1}	1.80×10^{-3}
0.9	6553, 26214, 166	5.72×10	2.03×10^2	2	3.9×10^{-1}	6.73×10^{-4}

Tabella 3.20: NN3 dataset 7

r_2	Clusters	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
	-	3.94	-	40	4.9×10^{-1}	1.14×10^{-3}
0.6	1638, 6553, 89	2.40	2.43×10^2	34	4.1×10^{-1}	6.01×10^{-4}
0.7	3276, 13107, 115	2.79	2.15×10^2	39	4.7×10^{-1}	9.25×10^{-4}
0.8	4915, 19660, 140	3.18	7.46×10^2	36	4.4×10^{-1}	7.40×10^{-4}
0.9	6553, 26214, 166	3.57	1.26×10^3	35	4.3×10^{-1}	6.88×10^{-4}

Tabella 3.21: NN3 dataset 10

r_2	Clusters	Space Overhead (KB)	Training Time (s)	ϵ	Error %	Mean Error
	-	3.08×10^{-2}	-	1270	1.2×10^{-1}	2.40×10^{-4}
0.6	1638, 6553, 89	1.87×10^{-2}	6.16×10^2	771	7.4×10^{-2}	1.66×10^{-4}
0.7	3276, 13107, 115	2.18×10^{-2}	8.28×10^2	678	6.5×10^{-2}	1.47×10^{-4}
0.8	4915, 19660, 140	2.49×10^{-2}	1.03×10^3	709	6.8×10^{-2}	1.47×10^{-4}
0.9	6553, 26214, 166	2.79×10^{-2}	2.00×10^3	675	6.4×10^{-2}	1.45×10^{-4}

3.2.3 MSE vs MAE

Le Tabelle 3.22 - 3.24 descrivono i risultati di alcuni esperimenti volti a confrontare le performance della rete neurale NN1 ottenute basandosi sulle funzioni di errore MSE (utilizzata in tutti i precedenti esperimenti) e Mean Absolute Error (MAE), quest'ultima è descritta in seguito:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |t_i - y_i|, \quad (3.3)$$

la cui derivata è

$$\frac{\partial \text{MAE}}{\partial (t_i - y_i)} = \begin{cases} +1 & \text{se } t_i - y_i > 0, \\ -1 & \text{altrimenti.} \end{cases} \quad (3.4)$$

La colonna **mb** indica diverse dimensioni dei minibatch. Dai risultati ottenuti si può vedere come tra le due diverse funzioni di errore non ci sia molta differenza di performance.

3.2.4 MSE vs LSE

La Tabella 3.25 descrive i risultati della rete neurale NN1 di alcuni esperimenti volti ad apprendere i pesi in modo da minimizzare l'errore massimo, e non quello medio. In tal proposito abbiamo provato ad effettuare un confronto tra le funzioni errore MSE e LogSumExp (LSE) descritta in seguito:

$$\text{LSE}(t_1 - y_1, \dots, t_n - y_n) = \log(\exp(t_1 - y_1) + \dots + \exp(t_n - y_n)), \quad (3.5)$$

la cui derivata è

$$\frac{\partial \text{LSE}(t_1 - y_1, \dots, t_n - y_n)}{\partial (t_i - y_i)} = \frac{\exp(t_i - y_i)}{\sum_j \exp(t_j - y_j)}. \quad (3.6)$$

Tabella 3.22: dataset 3

MSE			MAE		
$\eta = 1.0 \times 10^{-4}$					
mb	ϵ	Loss	mb	ϵ	Loss
16	8	3.16×10^{-5}	16	9	4.45×10^{-3}
32	8	3.15×10^{-5}	32	9	4.48×10^{-3}
64	8	3.15×10^{-5}	64	9	4.37×10^{-3}
$\eta = 1.0 \times 10^{-3}$					
16	8	3.15×10^{-5}	16	14	7.63×10^{-3}
32	8	3.15×10^{-5}	32	11	4.94×10^{-3}
64	8	3.15×10^{-5}	64	11	6.5×10^{-3}
$\eta = 5.0 \times 10^{-3}$					
16	8	3.16×10^{-5}	16	60	2.34×10^{-2}
32	8	3.16×10^{-5}	32	33	1.71×10^{-2}
64	8	3.15×10^{-5}	64	20	9.64×10^{-3}
$\eta = 1.0 \times 10^{-2}$					
16	8	3.22×10^{-5}	16	57	2.96×10^{-2}
32	8	3.34×10^{-5}	32	83	3.91×10^{-2}
64	8	3.15×10^{-5}	64	65	6.11×10^{-2}
$\eta = 5.0 \times 10^{-2}$					
16	11	6.54×10^{-5}	16	434	2.05×10^{-1}
32	8	3.22×10^{-5}	32	181	9.2×10^{-2}
64	8	3.25×10^{-5}	64	194	9.93×10^{-2}
$\eta = 1.0 \times 10^{-1}$					
16	44	9.45×10^{-4}	16	533	2.88×10^{-1}
32	8	3.24×10^{-5}	32	917	4.37×10^{-1}
64	8	3.49×10^{-5}	64	383	2.68×10^{-1}

Tabella 3.23: dataset 7

MSE			MAE		
$\eta = 1.0 \times 10^{-4}$					
mb	ϵ	Loss	mb	ϵ	Loss
16	43	3.48×10^{-6}	16	74	2.47×10^{-3}
32	42	3.48×10^{-6}	32	49	1.55×10^{-3}
64	43	3.48×10^{-6}	64	47	1.56×10^{-3}
$\eta = 1.0 \times 10^{-3}$					
16	42	3.49×10^{-6}	16	135	4.86×10^{-3}
32	43	3.49×10^{-6}	32	131	4.03×10^{-3}
64	43	3.48×10^{-6}	64	102	3.59×10^{-3}
$\eta = 5.0 \times 10^{-3}$					
16	45	3.51×10^{-6}	16	740	2.09×10^{-2}
32	42	3.48×10^{-6}	32	640	2.31×10^{-2}
64	42	3.49×10^{-6}	64	605	3.35×10^{-2}
$\eta = 1.0 \times 10^{-2}$					
16	46	3.57×10^{-6}	16	1080	3.29×10^{-2}
32	44	3.52×10^{-6}	32	1493	3.69×10^{-2}
64	42	3.49×10^{-6}	64	839	3.94×10^{-2}
$\eta = 5.0 \times 10^{-2}$					
16	54	4.15×10^{-6}	16	7958	2.83×10^{-1}
32	47	3.67×10^{-6}	32	4365	1.33×10^{-1}
64	56	3.85×10^{-6}	64	3848	1.51×10^{-1}
$\eta = 1.0 \times 10^{-1}$					
16	203	4.73×10^{-5}	16	11421	3.74×10^{-1}
32	75	1.01×10^{-5}	32	7154	2.28×10^{-1}
64	56	5.84×10^{-6}	64	8192	6.01×10^{-1}

Tabella 3.24: dataset 10

MSE			MAE		
$\eta = 1.0 \times 10^{-4}$					
mb	ϵ	Loss	mb	ϵ	Loss
16	613	3.36×10^{-8}	16	3120	6.22×10^{-4}
32	616	3.36×10^{-8}	32	1805	3.1×10^{-4}
64	620	3.36×10^{-8}	64	1242	2.24×10^{-4}
$\eta = 1.0 \times 10^{-3}$					
16	610	3.45×10^{-8}	16	1.55×10^4	2.88×10^{-3}
32	616	3.36×10^{-8}	32	1.640×10^4	3.39×10^{-3}
64	626	3.36×10^{-8}	64	1.175×10^4	2.39×10^{-3}
$\eta = 5.0 \times 10^{-3}$					
16	629	3.52×10^{-8}	16	9.410×10^4	1.75×10^{-2}
32	617	3.46×10^{-8}	32	1.048×10^5	3.31×10^{-2}
64	610	3.52×10^{-8}	64	5.29×10^4	1.33×10^{-2}
$\eta = 1.0 \times 10^{-2}$					
16	691	3.49×10^{-8}	16	2.129×10^5	3.88×10^{-2}
32	622	3.51×10^{-8}	32	1.620×10^5	2.9×10^{-2}
64	635	3.46×10^{-8}	64	1.245×10^5	2.41×10^{-2}
$\eta = 5.0 \times 10^{-2}$					
16	847	5.23×10^{-8}	16	1.153×10^6	1.91×10^{-1}
32	788	3.85×10^{-8}	32	6.299×10^5	1.1×10^{-1}
64	647	4.22×10^{-8}	64	3.650×10^5	7.41×10^{-2}
$\eta = 1.0 \times 10^{-1}$					
16	2884	4.09×10^{-7}	16	1.048×10^6	6.79×10^{-1}
32	909	4.45×10^{-8}	32	1.183×10^6	2.29×10^{-1}
64	984	7.45×10^{-8}	64	1.049×10^6	5.66×10^{-1}

Da (3.6) possiamo notare come la derivata di LSE sia la softmax (la funzione di attivazione utilizzata nel Paragrafo 3.1.1): questo permetterà, durante l'addestramento, di modificare in modo più marcato i pesi che causano l'errore maggiore. Dai risultati LSE sembra farsi preferire, anche se non in maniera significativa. Sembra avere la caratteristica di convergere prima, rispetto al MSE, e lo si nota soprattutto con i learning rate più bassi, dove la convergenza è più lenta.

3.2.5 Splitting in N modelli

L'ultimo esperimento considerato è consistito nel dividere il dataset ordinato in sottosequenze ordinate e usare una rete neurale senza strati hidden (NN1) per ogni sottosequenza, con l'obiettivo di ridurre l'errore massimo ϵ che sarà quindi il più grande degli errori massimi di ogni modello. I risultati sono riportati nella Tabella 3.26, in cui le intestazioni delle colonne sono:

- Split: numero di split sul dataset (e di conseguenza numero di modelli NN1 usati);
- ϵ : il più grande degli errori massimi di ogni NN1;
- μ : media degli errori massimi di ogni NN1;
- SpaceOvh: somma dello spazio (in KB) di ogni NN1 rispetto al dataset.

Come ci si aspetta, a eccezione di alcuni casi, al crescere del numero di split le performance migliorano occupando meno delle reti compresse precedenti, facendo un confronto performance/spazio occupato con le compressioni precedenti troviamo: dataset 7, NN2, pruning 80% ed errore massimo 33 (vedi Tabella 3.9); dataset 10, NN2, pruning 90% ed errore massimo 699 (vedi Tabella 3.10). Nella Tabella 3.26 sono evidenziati i modelli con lo stesso spazio occupato delle reti precedenti, a parità (o poco meno) di spazio occupato abbiamo un errore massimo nettamente più basso.

Tabella 3.25: Confronto dell'errore massimo della rete NN1 con funzione di errore MSE e LSE, la colonna mb indica diverse dimensioni dei minibatch

mb	dataset 3		dataset 7		dataset 10	
	MSE	LSE	MSE	LSE	MSE	LSE
	ϵ		ϵ		ϵ	
$\eta = 1.0 \times 10^{-5}$						
64	149	23	44	42	788	766
128	97	16	43	42	788	766
256	79	13	43	42	787	766
512	72	11	43	42	787	765
1024			43	42	787	765
$\eta = 1.0 \times 10^{-4}$						
64	12	8	42	42	624	623
128	9	8	42	42	626	625
256	8	8	42	42	628	627
512	8	8	42	42	629	628
1024			42	42	629	628
$\eta = 1.0 \times 10^{-3}$						
64	8	8	42	42	626	627
128	8	8	42	42	619	619
256	8	8	42	42	616	615
512	8	8	42	42	620	620
1024			42	42	619	619
$\eta = 1.0 \times 10^{-2}$						
64	8	8	42	42	635	636
128	8	8	42	43	636	594
256	8	8	42	42	660	659
512	8	8	43	43	617	615
1024			42	42	616	614
$\eta = 1.0 \times 10^{-1}$						
64	8	8	56	55	984	979
128	8	8	49	48	717	696
256	8	8	45	43	661	667
512	8	8	41	41	603	573
1024			41	41	671	716

Tabella 3.26: Risultati dello splitting dei dataset

Split	dataset 3			dataset 7			dataset 10		
	ϵ	μ	SpaceOvh	ϵ	μ	SpaceOvh	ϵ	μ	SpaceOvh
1	8	8.0	4.96×10^{-2}	41	41.0	3.1×10^{-3}	714	714.0	2.42×10^{-5}
2	8	6.5	9.92×10^{-2}	36	29.0	6.2×10^{-3}	593	557.5	4.84×10^{-5}
3	7	5.0	1.49×10^{-1}	33	27.67	9.3×10^{-3}	494	420.33	7.26×10^{-5}
4	9	6.0	1.98×10^{-1}	43	30.5	1.24×10^{-2}	515	428.75	9.69×10^{-5}
5	6	4.4	2.48×10^{-1}	27	20.2	1.55×10^{-2}	545	360.0	1.21×10^{-4}
6	5	3.5	2.98×10^{-1}	31	22.17	1.86×10^{-2}	561	319.17	1.45×10^{-4}
7	5	3.14	3.47×10^{-1}	32	19.86	2.17×10^{-2}	538	292.57	1.69×10^{-4}
8	5	3.25	3.97×10^{-1}	29	21.5	2.48×10^{-2}	465	298.5	1.94×10^{-4}
9	3	2.33	4.46×10^{-1}	33	19.0	2.79×10^{-2}	453	240.33	2.18×10^{-4}
10	4	2.8	4.96×10^{-1}	24	15.6	3.1×10^{-2}	425	228.6	2.42×10^{-4}
11	4	2.45	5.45×10^{-1}	25	15.09	3.41×10^{-2}	338	222.64	2.66×10^{-4}
12	3	1.92	5.95×10^{-1}	28	14.92	3.72×10^{-2}	311	200.08	2.91×10^{-4}
13	3	1.85	6.45×10^{-1}	24	14.46	4.03×10^{-2}	207	183.85	3.15×10^{-4}
14	2	1.57	6.94×10^{-1}	26	13.64	4.34×10^{-2}	238	175.07	3.39×10^{-4}
15	4	1.67	7.44×10^{-1}	24	12.4	4.65×10^{-2}	254	176.53	3.63×10^{-4}
16	2	1.19	7.93×10^{-1}	21	13.25	4.96×10^{-2}	305	196.75	3.87×10^{-4}
17	2	1.12	8.43×10^{-1}	23	12.65	5.27×10^{-2}	266	169.53	4.12×10^{-4}
18	2	1.17	8.93×10^{-1}	21	12.44	5.58×10^{-2}	246	154.06	4.36×10^{-4}
19	4	1.16	9.42×10^{-1}	19	10.79	5.89×10^{-2}	193	147.79	4.6×10^{-4}
20	1	1.0	9.92×10^{-1}	27	11.55	6.2×10^{-2}	233	144.3	4.84×10^{-4}
21	2	1.1	1.04	20	11.05	6.51×10^{-2}	277	150.67	5.08×10^{-4}
22	2	1.05	1.09	20	10.77	6.82×10^{-2}	357	143.0	5.33×10^{-4}
26	1	1.0	1.29	13	9.35	8.06×10^{-2}	209	131.85	6.3×10^{-4}
30	2	1.07	1.49	17	9.3	9.3×10^{-2}	233	116.7	7.26×10^{-4}
34				16	8.53	1.05×10^{-1}	196	106.15	8.23×10^{-4}
38				13	7.39	1.18×10^{-1}	177	104.55	9.2×10^{-4}
42				11	7.14	1.3×10^{-1}	214	101.38	1.02×10^{-3}
46				13	7.13	1.43×10^{-1}	153	88.57	1.11×10^{-3}
50				12	6.8	1.55×10^{-1}	143	84.88	1.21×10^{-3}
54				13	6.57	1.67×10^{-1}	145	85.07	1.31×10^{-3}
58				12	6.24	1.8×10^{-1}	133	82.4	1.4×10^{-3}
62				11	6.02	1.92×10^{-1}	143	80.34	1.5×10^{-3}
64				21	6.0	1.98×10^{-1}	142	79.77	1.55×10^{-3}
72				9	5.24	2.23×10^{-1}	121	69.71	1.74×10^{-3}
80				9	4.92	2.48×10^{-1}	134	66.75	1.94×10^{-3}
88				10	4.65	2.73×10^{-1}	153	65.88	2.13×10^{-3}
96				9	4.38	2.98×10^{-1}	109	60.6	2.32×10^{-3}
104				8	4.13	3.22×10^{-1}	107	59.13	2.52×10^{-3}
112				7	3.83	3.47×10^{-1}	99	56.11	2.71×10^{-3}
120				10	3.75	3.72×10^{-1}	118	54.53	2.91×10^{-3}
128				6	3.52	3.97×10^{-1}	302	56.22	3.1×10^{-3}

Conclusioni

In questo elaborato si sono studiati due metodi di compressione di reti neurali artificiali per valutarne l'efficacia in un problema di classificazione e in un problema di regressione. In particolare, si voleva valutare la capacità di tali tecniche di non degradare le performance. Dai risultati ottenuti durante i numerosi esperimenti si è riscontrato che le connessioni della rete neurale, sono spesso molte di più di quelle che realmente servono per ottenere ottimi risultati; per questo motivo l'elaborato conferma che gli algoritmi di compressione ricoprono un ruolo fondamentale in questo ambito. L'obiettivo di mantenere le performance ai livelli di una rete neurale senza compressione è stato raggiunto con successo, a discapito di altri fattori quali la complessità dell'implementazione e il tempo di esecuzione dell'apprendimento. Siccome i parametri della rete presentata sono numerosi, abbiamo ottenuto una vasta gamma di risultati, in cui si può notare come diverse configurazioni ottengono risultati simili: un numero maggiore di neuroni risultano in un'accuratezza ottimale con un pruning alto, a discapito del tempo di esecuzione; al contrario, un numero minore di neuroni portano ad avere un tempo di esecuzione più basso, a discapito di un pruning non troppo significativo, e talvolta l'impossibilità di eseguire un clustering adeguato per mancanza di sufficienti connessioni. Come si nota negli ultimi esperimenti riguardanti il problema del predecessore (in cui vogliamo approssimare una funzione monotona non decrescente) è conveniente non sovradimensionare la rete neurale, reti come NN1 in molti casi hanno dato performance migliori di reti neurali molto più grandi. In conclusione, si può constatare, dai dati ottenuti, che la compressione di una rete, con gli algoritmi utilizzati in questo lavoro, ha molteplici vantaggi e permette a dispositivi meno prestanti di far uso di reti neurali artificiali a uno spazio di memoria ridotto rispetto alla versione tradizionale.

Bibliografia

- [1] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.
- [2] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. 2015. cite arxiv:1510.00149Comment: Published as a conference paper at ICLR 2016 (oral).
- [3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. cite arxiv:1502.01852.
- [4] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
- [5] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.
- [6] Payam Refaeilzadeh, Lei Tang, and Huan Liu. *Cross-Validation*, pages 532–538. Springer US, Boston, MA, 2009.
- [7] David E. Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. 1 backpropagation : The basic theory. 2008.
- [8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.
- [9] Twan van Laarhoven. L2 regularization versus batch and weight normalization. *ArXiv*, abs/1706.05350, 2017.
- [10] Egm4313.s12 (Prof. Loc Vu-Quoc) Wikiedia Commons.
- [11] Josef Steppan Wikiedia Commons.