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

Introduzione

BLABLA Blablabla said Nobody [3].

Capitolo 1

Reti Neurali

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}.

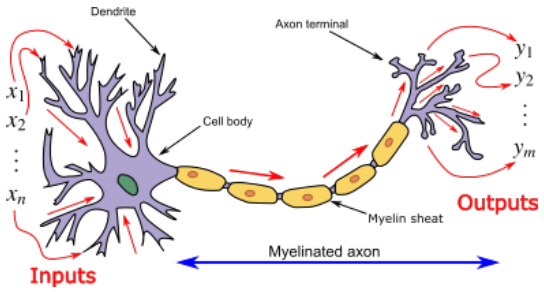


Figura 1.1: Neurone Biologico [9]

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 la probabilità 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, in poche parole, 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 (feedforward). Queste reti possono essere viste come funzioni matematiche non lineari che trasformano un insieme di variabili indipendenti $x = (x_1, ..., x_d)$, chiamate ingressi della rete, in un insieme di variabili dipendenti $y = (y_1, ..., y_c)$, chiamate uscite della rete. La precisa forma di queste funzioni dipende dalla struttura interna della rete e da un insieme di valori $w = (w_1, ..., w_d)$, 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.

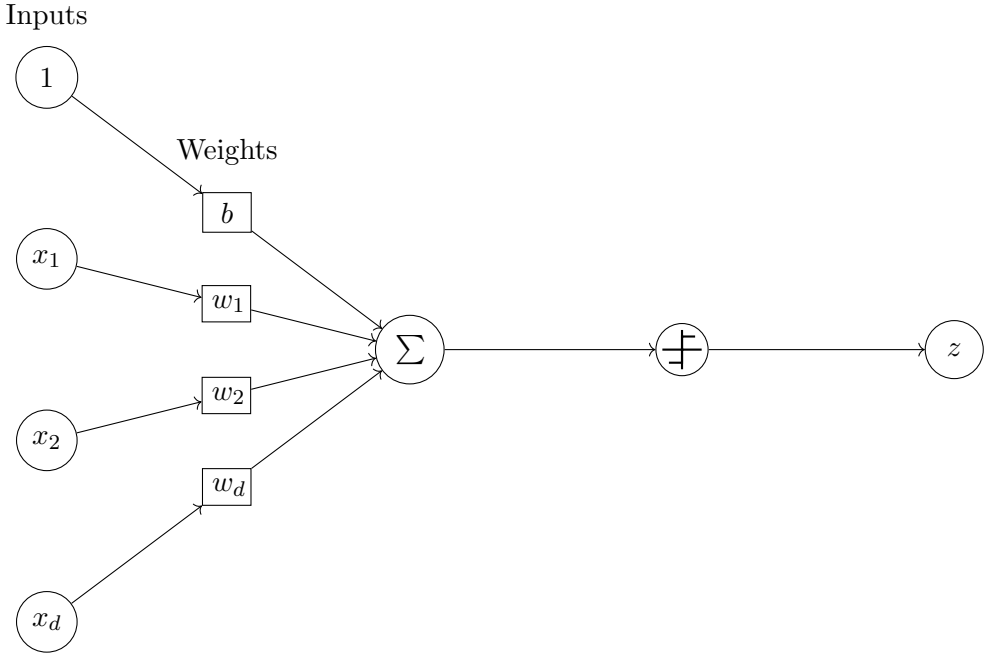


Figura 1.2: Modello di McCulloch-Pitts

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. Esso può essere visto come una funzione non lineare che trasforma le variabili di ingresso $x_1, ..., x_d$ 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 $w_1, ..., w_d$ (che sono analoghi alle potenze delle sinapsi nella rete biologica), ottenendo così

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

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

$$a = \sum_{i=0}^{d} w_i x_i \tag{1.2}$$

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

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

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

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

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), $q = 1, ..., n$, 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:

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

La funzione di errore E risulta allora essere:

$$E = \sum_{q=1}^{n} \sum_{k=1}^{c} r_{qk}^2 \tag{1.6}$$

é 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 una 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). Le funzioni di attivazione sono non-lineari e la loro derivata è calcolabile in modo analitico per velocizzare la computazione.

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

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

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

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

1.5.1 Architettura del modello MLP

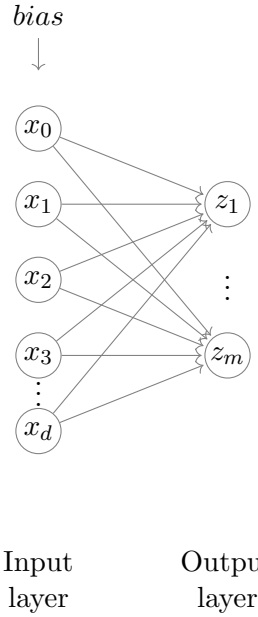


Figura 1.3: MLP a uno strato

Modello a uno strato Nella sezione precedente abbiamo trattato la singola unità di elaborazione descritta in (1.4). 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 date da

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

dove w_{ij} rappresenta il peso che connette l'ingresso i con l'uscita j; $g()$ è una funzione di attivazione. $x_0 = 1$ per utilizzare l'equazione semplificata.

Modello a due strati Per ottenere reti più potenti[1] è necessario considerare reti aventi più strati chiamate *multilayer perceptron* come in Figura 1.4. Le unità centrali rappresentano lo

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

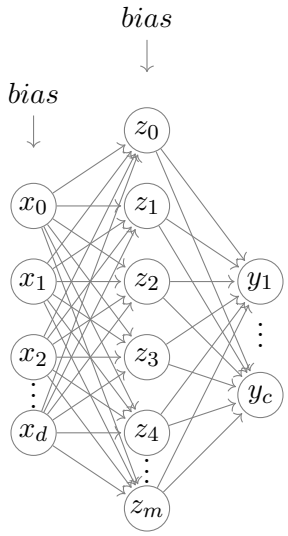


Figura 1.4: MLP a due strati

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

$$y_k = \tilde{g}\left(\sum_{j=0}^{m} \tilde{w}_{kj} z_i\right), \quad k = 1, .., c \tag{1.11}$$

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

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

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

Per ottenere una capacità di rappresentazione universale, la funzione di attivazione g delle unità nascoste deve essere scelta non lineare. 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 $\mathbf{w} = (w_1, .., w_n)$ [2] che minimizzano la funzione di errore $E(\mathbf{w})$ (Capitolo 1.3). La ricerca del minimo avviene in modo iterativo partendo da

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

un valore iniziale **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 **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).

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

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. Omettiamo l'indice q, i passaggi che seguiranno si riferiranno ad un singolo caso q ma le operazioni sono fatte per ogni valore di q. Consideriamo un esempio di rete neurale MLP con 1 strato hidden.

$$y_k = \tilde{g}(\tilde{a}_k) \qquad a_k = \sum_{j=0}^{m} \tilde{w}_{kj} z_j \tag{1.14}$$

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

$$\frac{\partial E^q}{\partial \tilde{w}_{kj}} = \frac{\partial E^q}{\partial \tilde{a}_k} \frac{\partial \tilde{a}_k}{\partial w_{kj}} \tag{1.15}$$

Con l'equazione (1.14) troviamo:

$$\frac{\partial \tilde{a}_k}{\partial \tilde{w}_{kj}} = z_j \tag{1.16}$$

Dalle equazioni (1.14) e (1.13):

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

Otteniamo quindi:

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

Definendo

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

otteniamo una semplice espressione per la derivata di E^q rispetto a w_{kj}:

$$\frac{\partial E^q}{\partial \tilde{w}_{kj}} = \tilde{\delta}_k z_j. \tag{1.20}$$

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

$$z_j = g(a_j) \qquad a_j = \sum_{i=0}^{d} w_{ji} x_i. \tag{1.21}$$

Possiamo quindi scrivere la derivata come

$$\frac{\partial E^q}{\partial w_{ji}} = \frac{\partial E^q}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{1.22}$$

In modo analogo, osservando (1.21), troviamo:

$$\frac{\partial a_j}{\partial w_{ji}} = x_i \tag{1.23}$$

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

$$\frac{\partial E^q}{\partial a_j} = \sum_{k=1}^{c} \frac{\partial E^q}{\partial \tilde{a}_k}\frac{\partial \tilde{a}_k}{\partial a_j} \tag{1.24}$$

dove la derivata di E^q rispetto ad $\tilde{a}_k$ è data dall'equazione (1.18), mentre la derivata di $\tilde{a}_k$ rispetto ad a_j la troviamo usando le equazioni (1.14) e (1.21):

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

Usando le equazioni (1.19), (1.24) e (1.25):

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

Possiamo quindi riscrivere l'equazione (1.22):

$$\frac{\partial E^q}{\partial w_{ji}} = g'(a_j) x_i \sum_{k=1}^{c} w_{kj}\delta_k \tag{1.27}$$

Come abbiamo fatto nell'equazione (1.19), poniamo

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

Ottenendo infine

$$\frac{\partial E^q}{\partial w_{ji}} = \delta_j x_i \tag{1.29}$$

che ha la stessa semplice forma dell'equazione (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 $k = 1, .., c$ usando equazione (1.19);
- Valutare il valore δ_j per $j = 1, .., m$ 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 $(\mathbf{x}^q, \mathbf{t}^q)$ 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:

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

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

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:

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

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 **??**, 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

2.1 Pruning

Il pruning consiste nel tagliare le connessioni da una rete addestrata per poi riaddestrarla senza le connessioni tagliate. Oltre ad un vantaggio computazionale può portare a una generalizzazione che permette di ridurre l'overfitting (ovvero imparare troppo dagli esempio di training).

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). 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 destra 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

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 addesstrate in precedenza. Identifico con τ la soglia entro cui i pesi verranno eliminati, la nuova matrice sarà definita come:

$$w_{ij} = \begin{cases} 0 \text{ se } |w_{ij}| < \tau \\ w_{ij} \text{ altrimenti.} \end{cases} \tag{2.1}$$

Per comodità abbiamo scelto τ come il quantile q della distribuzione del valore assoluto dei pesi, dove q assume i valori in [0,1].

2.1.3 Tasso di compressione

2.2 Weight Sharing

La tecnica del weight sharing 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 per aver definito un centroide per ogni cluster tutti i pesi vengono sostituiti nella rete con i centroidi più vicini.

2.2.1 Strutture dati necessarie

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

$$N_{ij} = \arg\min_k |C_k - w_{ij}| \tag{2.2}$$

2.2.2 Tecniche implementative

Alla rete neurale base vengono aggiunte 2 procedure:

- procedura che crea i vettori contenenti i k centroidi dove k è il numero di cluster scelti (per ogni strato)
- procedura che crea la matrice N definita in (2.2)

Alla normale fase di training vengono aggiunte 2 procedure:

- costruisce la matrice dei pesi effettiva per il feed forward con i valori dei centroidi invece dei valori originali:
$$W'_{ij} = C_{N_{ij}} \tag{2.3}$$
- calcolare il gradiente dei centroidi tramite il *cumulative gradient descent*. Denotato con $\mathcal{L}$ il delta relativo alla funzione di loss, con N la matrice degli indici dei cluster e con C il vettore dei centroidi; il gradiente dei centroidi è calcolato come:
$$\frac{\partial \mathcal{L}}{\partial c_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} 1(N_{ij} = k) \tag{2.4}$$

Capitolo 3

Esperimenti

In questo capitolo spiegheremo gli espereminti eseguiti su due dataset differenti.

- MNIST: problema di classificazione, il dataset è composto da un insieme di immagini che rappresentano cifre scritte a mano
- Sequenze ordinate di numeri: problema di regressione, il dataset è composto da sequenze di numeri rappresentati in 64 bit

Classificazione e Regressione I classificatori separano i dati in due o più classi, nel caso di questo elaborato negli esperimenti con MNIST abbiamo 10 classi (le cifre tra 0 e 9). I regressori invece si basano sull'interpolazione dei dati per associare tra loro due o più caratteristiche (*feature*). Simile alla classificazione con la differenza che l'output ha un dominio continuo. Entrambe le categorie sono affrontate, in questo elaborato, con apprendimento *supervisionato* in cui si ha un insieme di input di esempio e il corrispettivo output desiderato con lo scopo di apprendere una regola generale in grado di mappare gli input negli output.

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, nella Figura 3.1 sono presenti alcuni esempi.

Figura 3.1: Esempi MNIST [11]

3.1.1 Addestramento e Tuning Parametri

Per trovare una buona configurazione abbiamo svolto una *K-Fold Cross Validation* [6].

K-Fold Cross Validation Il *training set* viene diviso in k subset, a rotazione vengono usati come *training set* $k-1$ subset mentre il rimanente con *validation set* (un *test set* fittizio).

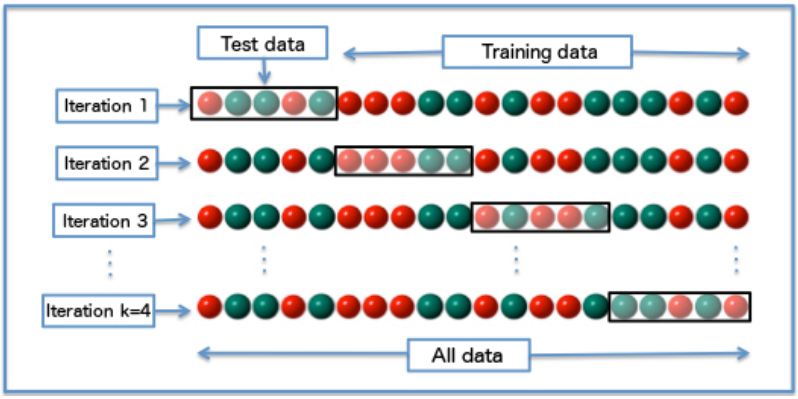


Figura 3.2: Esempio k-fold con k=4 [10]

Negli esperiamenti abbiamo fissato i seguenti parametri:

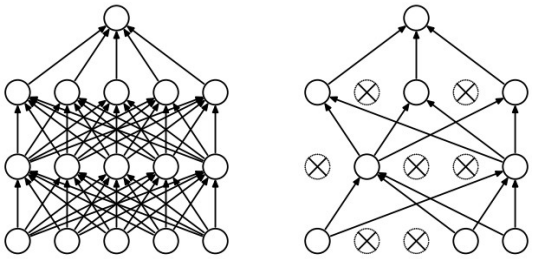
- $\eta = 0.003$
- Funzione errore = Errore quadratico medio
- Funzione di attivazione degli strati *hidden* = $ReLU$
- Funzione di attivazione dello strato di output = *softmax*
- Dimensione dei minibatch = 100 elementi
- Funzione di stop = 100 epoche
- Inizializzazione pesi = $\mathbf{W^l} = \mathbf{randn}\sqrt{\frac{2}{size^{l-1}}}$ (He et al initialization [2]) dove **randn** è una matrice di numeri casuali estratti da una distribuzione normale standard, l rappresenta lo strato e $size^{l-1}$ il numero di neuroni dello strato precedente.

I parametri di cui vogliamo trovare la configurazione migliore sono invece:

- Modello (ovvero quanti neuroni e quanti neuroni per ogni strato hidden)
- Dropout (% di neuroni negli strati hidden non utilizzati nella fase di training)

Dropout Il *dropout* [8] è una tecnica utilizzata per evitare l'overfitting (ovvero imparare troppo dagli esempi del *training set* senza imparare a generalizzare) mediante l'eliminazione di alcuni neuroni casuali in ogni strato ad eccezione dello strato di output. I neuroni da eliminare cambiamo ogni epoca, durante la fase di predizione invece sono attivi tutti i neuroni. Nell'elaborato è stata utilizzata la versione chiamata *inverted dropout*

$$\mathbf{drop_l} = (\mathbf{rand} < p)/p \qquad \mathbf{output_l} = \mathbf{output_l} * \mathbf{drop_l} \tag{3.1}$$



(a) Standard Neural Net (b) After applying dropout.

Figura 3.3: Esempio dropout [8]

Tabella 3.1: MLP con uno strato hidden

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

dove $\mathbf{output_l}$ rappresenta l'output di un generico strato l, mentre **rand** rappresenta una matrice di numeri casuali estratti da una distribuzione uniforme nell'intervallo $[0, 1)$ della stessa dimensione di $output_l$. p invece è un parametro che rappresenta la probabilità di tenere attivo un neurone.

Tabella 3.2: MLP con due strati hidden

$p = 0.75$		$p = 1$	
neuroni	accuratezza	neuroni	accuratezza
50 - 25	97.14	50 - 25	97.08
50 - 50	97.26	50 - 50	96.82
75 - 25	97.43	75 - 25	97.43
75 - 50	97.52	75 - 50	97.51
75 - 75	97.51	75 - 75	97.49
100 - 25	97.53	100 - 25	97.54
100 - 50	97.66	100 - 50	97.62
100 - 75	97.62	100 - 75	97.63
100 - 100	97.67	100 - 100	97.67
125 - 25	97.68	125 - 25	97.62
125 - 50	97.77	125 - 50	97.7
125 - 75	97.72	125 - 75	97.68
125 - 100	97.73	125 - 100	97.74
125 - 125	97.74	125 - 125	97.77
150 - 25	97.76	150 - 25	97.59
150 - 50	97.71	150 - 50	97.8
150 - 75	97.76	150 - 75	97.76
150 - 100	97.75	150 - 100	97.85
150 - 125	97.78	150 - 125	97.74
150 - 150	97.75	150 - 150	97.8
175 - 50	97.85	175 - 50	97.8
175 - 75	97.8	175 - 75	97.8
175 - 100	97.83	175 - 100	97.8
175 - 125	97.85	175 - 125	97.8
175 - 150	97.82	175 - 150	97.86
175 - 175	97.85	175 - 175	97.84
200 - 50	97.83	200 - 50	97.83
200 - 75	97.83	200 - 75	97.9
200 - 100	97.88	200 - 100	97.89
200 - 125	97.91	200 - 125	97.85
200 - 150	97.84	200 - 150	97.85
200 - 175	97.84	200 - 175	97.89
200 - 200	97.86	200 - 200	97.83
225 - 75	97.92	225 - 75	97.9
225 - 100	97.86	225 - 100	97.79
225 - 125	97.85	225 - 125	97.85
225 - 150	97.94	225 - 150	97.94
225 - 175	97.88	225 - 175	97.89
225 - 200	97.91	225 - 200	97.9
250 - 75	97.86	250 - 75	97.88
250 - 100	97.91	250 - 100	97.91
250 - 125	97.93	250 - 125	97.94
250 - 150	97.93	250 - 150	97.9
250 - 175	97.86	250 - 175	97.9
250 - 200	97.89	250 - 200	97.92

Tabella 3.3: Pruning su MNIST

%Pruning	accuratezza
ALL	98.34
10%	98.34
15%	98.34
20%	98.34
25%	98.35
30%	98.36
35%	98.35
40%	98.36
45%	98.36
50%	98.36
55%	98.32
60%	98.3
65%	98.33
70%	98.34
75%	98.34
80%	98.38
85%	98.31
90%	98.27
95%	98.28

Tabella 3.4: Weight Sharing su MNIST

Centroidi	accuratezza
ALL	98.34
4 - 2	94.0
16 - 8	98.14
32 - 8	98.32
32 - 16	98.25
64 - 16	98.29
64 - 32	98.34
128 - 32	98.38
192 - 32	98.38
192 - 64	98.36
256 - 32	98.31
256 - 64	98.36
512 - 32	98.38
512 - 64	98.35
1024 - 32	98.37
1024 - 64	98.37
2048 - 32	98.36
2048 - 64	98.36
4096 - 32	98.35
4096 - 64	98.37
8192 - 32	98.43
8192 - 64	98.34

3.1.2 Pruning

3.1.3 Weight Sharing

3.2 Problema del predecessore

Sia X un sottoinsieme delle parti di un universo U, ordinato in base all'ordine $\leq$ definito su U. Partiamo dal presupposto che ogni elemento di U può essere rappresentato da d bit. Supponiamo anche che l'ordinamento lessicografico della rappresentazione binaria di ogni elemento in U mantiene la relazione d'ordine $\leq$. Il problema di ricerca del predecessore consiste nel trovare la posizione della chiave più grande in X non maggiore di una data in input x, indichiamo con $pos(x)$ la posizione di x appartenente a X nella sequenza ordinata, il problema è determinare $pred(x) = pos(z)$ con $z = \max y \in X$ dove $y \leq x$.

$$x = \{2, 3, 4, 5, 12, 15, 18\} \qquad \text{(Esempio)}$$

Assumiamo che 7 è la chiave da cercare. La ricerca del predecessore dovrebbe restituire la posizione 4. Dato $|X| = n$, sia F_X la distribuzione cumulativa empirica degli elementi di U rispetto a X, per ogni $x \in U$, $F_X(x) = \frac{|y \in X | y \leq z|}{n}$. Per semplificare la notazione denotiamo F_X con F. Nell'esempio sopra, $F(2) = \frac{1}{7}$, $F(5) = \frac{4}{7}$, $F(6) = \frac{4}{7}$, $F(18) = 1$. La conoscenza di F ci da una soluzione al nostro problema, dato un elemento di x appartenente a U, una sola valutazione di F fornisce $pred(x) = \lceil (F(z) * n) \rceil$. Il nostro obiettivo è trovare 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 $O(\log n)$ degli alberi 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 (ϵ) che la rete compie. Questo perchè durante la ricerca dopo che il modello ci ha dato la posizione dobbiamo effettuare una ricerca di ϵ a destra e sinistra della posizione restituita dal modello. Per questo motivo negli esperimenti abbiamo provato a minimizzare una funzione che approssima il massimo convessa, ovvero la *Log Sum Exp*. Diversamente da MNIST il modello MLP cerchera di risolvere un problema di regressione. La rete neurale dovrà imparare a restituire, data una chiave, $\tilde{F}$.

3.2.1 Addestramento e Tuning Parametri

Diversamente da MNIST il modello MLP cerchera di risolvere un problema di regressione. La rete neurale dovrà imparare a restituire, data una chiave, $\tilde{F}$. I parametri soggetti al tuning sono la funzione di errore, il learning rate (η), la taglia dei minibatch, le funzioni di attivazione. Per quanto riguarda la loss ho effettuato dei confronti tra MSE (Mean Square Error), MAE (Mean Absolute Error) e LSE (log sum exp).

3.2.2 Applicazione degli algoritmi di compressione

3.2.3 Splitting in N modelli

L'idea usata in questa sezione è quella di dividere il dataset ordinato in sottosequenze ordinate e usare una rete neurale senza strati hidden per ogni sottosequenza con l'obiettivo di ridurre l'errore massimo ϵ che sarà quindi il massimo degli errori massimi di ogni modello.

- Split = Numero di split sul dataset (e di conseguenza numero di modelli NN0 usati)
- ϵ = Massimodegli errori massimi di ogni NN0
- μ = Media degli errpri massimi di ogni NN0
- SpaceOVH = spazio dei modelli in KB rispetto al dataset

NN0 file 3

Split	ϵ	μ	SpaceOVH
1	8	8.0	4.96×10^{-2}
2	8	6.5	9.92×10^{-2}
3	7	5.0	1.49×10^{-1}
4	9	6.0	1.98×10^{-1}
5	6	4.4	2.48×10^{-1}
6	5	3.5	2.98×10^{-1}
7	5	3.14	3.47×10^{-1}
8	5	3.25	3.97×10^{-1}
9	3	2.33	4.46×10^{-1}
10	4	2.8	4.96×10^{-1}
11	4	2.45	5.45×10^{-1}
12	3	1.92	5.95×10^{-1}
13	3	1.85	6.45×10^{-1}
14	2	1.57	6.94×10^{-1}
15	4	1.67	7.44×10^{-1}
16	2	1.19	7.93×10^{-1}
17	2	1.12	8.43×10^{-1}
18	2	1.17	8.93×10^{-1}
19	4	1.16	9.42×10^{-1}
20	1	1.0	9.92×10^{-1}
21	2	1.1	1.04
22	2	1.05	1.09
26	1	1.0	1.29
30	2	1.07	1.49

NN0 file 7

Split	ϵ	μ	SpaceOVH
1	41	41.0	3.1×10^{-3}
2	36	29.0	6.2×10^{-3}
3	33	27.67	9.3×10^{-3}
4	43	30.5	1.24×10^{-2}
5	27	20.2	1.55×10^{-2}
6	31	22.17	1.86×10^{-2}
7	32	19.86	2.17×10^{-2}
8	29	21.5	2.48×10^{-2}
9	33	19.0	2.79×10^{-2}
10	24	15.6	3.1×10^{-2}
11	25	15.09	3.41×10^{-2}
12	28	14.92	3.72×10^{-2}
13	24	14.46	4.03×10^{-2}
14	26	13.64	4.34×10^{-2}
15	24	12.4	4.65×10^{-2}
16	21	13.25	4.96×10^{-2}
17	23	12.65	5.27×10^{-2}
18	21	12.44	5.58×10^{-2}
19	19	10.79	5.89×10^{-2}
20	27	11.55	6.2×10^{-2}
21	20	11.05	6.51×10^{-2}
22	20	10.77	6.82×10^{-2}
26	13	9.35	8.06×10^{-2}
30	17	9.3	9.3×10^{-2}
34	16	8.53	1.05×10^{-1}
38	13	7.39	1.18×10^{-1}
42	11	7.14	1.3×10^{-1}
46	13	7.13	1.43×10^{-1}
50	12	6.8	1.55×10^{-1}
54	13	6.57	1.67×10^{-1}
58	12	6.24	1.8×10^{-1}
62	11	6.02	1.92×10^{-1}
64	21	6.0	1.98×10^{-1}
72	9	5.24	2.23×10^{-1}
80	9	4.92	2.48×10^{-1}
88	10	4.65	2.73×10^{-1}
96	9	4.38	2.98×10^{-1}
104	8	4.13	3.22×10^{-1}
112	7	3.83	3.47×10^{-1}
120	10	3.75	3.72×10^{-1}
128	6	3.52	3.97×10^{-1}

NN0 file 10

Split	ϵ	μ	SpaceOVH
1	714	714.0	2.42×10^{-5}
2	593	557.5	4.84×10^{-5}
3	494	420.33	7.26×10^{-5}
4	515	428.75	9.69×10^{-5}
5	545	360.0	1.21×10^{-4}
6	561	319.17	1.45×10^{-4}
7	538	292.57	1.69×10^{-4}
8	465	298.5	1.94×10^{-4}
9	453	240.33	2.18×10^{-4}
10	425	228.6	2.42×10^{-4}
11	338	222.64	2.66×10^{-4}
12	311	200.08	2.91×10^{-4}
13	207	183.85	3.15×10^{-4}
14	238	175.07	3.39×10^{-4}
15	254	176.53	3.63×10^{-4}
16	305	196.75	3.87×10^{-4}
17	266	169.53	4.12×10^{-4}
18	246	154.06	4.36×10^{-4}
19	193	147.79	4.6×10^{-4}
20	233	144.3	4.84×10^{-4}
21	277	150.67	5.08×10^{-4}
22	357	143.0	5.33×10^{-4}
26	209	131.85	6.3×10^{-4}
30	233	116.7	7.26×10^{-4}
34	196	106.15	8.23×10^{-4}
38	177	104.55	9.2×10^{-4}
42	214	101.38	1.02×10^{-3}
46	153	88.57	1.11×10^{-3}
50	143	84.88	1.21×10^{-3}
54	145	85.07	1.31×10^{-3}
58	133	82.4	1.4×10^{-3}
62	143	80.34	1.5×10^{-3}
64	142	79.77	1.55×10^{-3}
72	121	69.71	1.74×10^{-3}
80	134	66.75	1.94×10^{-3}
88	153	65.88	2.13×10^{-3}
96	109	60.6	2.32×10^{-3}
104	107	59.13	2.52×10^{-3}
112	99	56.11	2.71×10^{-3}
120	118	54.53	2.91×10^{-3}
128	302	56.22	3.1×10^{-3}

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] 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.

[3] Nobody Jr. My article, 2006.

[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] Egm4313.s12 (Prof. Loc Vu-Quoc) Wikiedia Commons.

[10] Fabian Flöck Wikiedia Commons.

[11] Josef Steppan Wikiedia Commons.