UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea F1X

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 [1].

Chapter 1

Reti Neurali

1.1 Reti Neurali Biologiche

I neuroni sono delle celle elettricamente attive ed il cervello umano 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}.

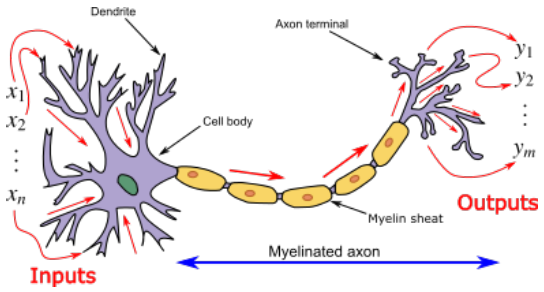

Figure 1.1: Neurone Biologico [5]

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.

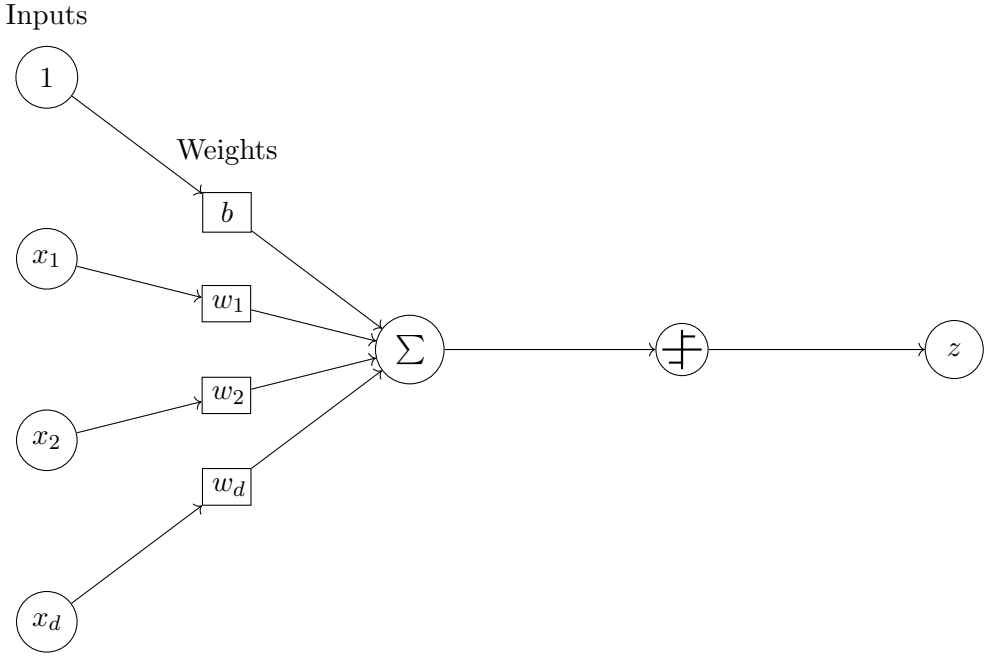

Figure 1.2: Modello di McCulloch-Pitts

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 [2] 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 l'equazione (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

$$z = g(a) = g\left(\sum_{i=1}^{d} w_i x_i\right). \tag{1.3}$$

Il modello originale di McCulloch-Pitts usava 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 processati 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 del 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

Le funzioni di attivazione sono equazioni matematiche che determinano l'output di una rete neurale. Le funzioni utilizzate principalmente negli esperimenti di questo elaborato sono tre: *sigmoid*, *ReLU* (*Rectified Linear Units*) e *Leaky-ReLU*. Le funzioni di attivazione sono non-lineari e la loro derivata è calcolabile in modo analitico per velocizzare la computazione.

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

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

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

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

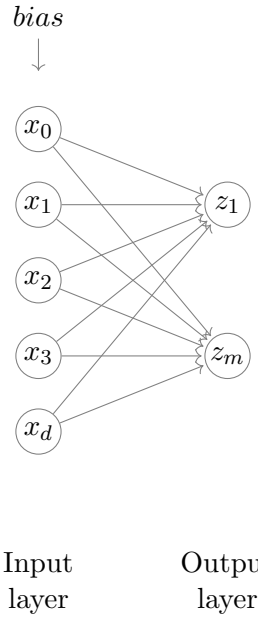

Figure 1.3: MLP a uno strato

Modello a uno strato Nella sezione precedente abbiamo trattato la singola unità di elaborazione descritta dall'equazione (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 è necessario considerare reti aventi più strati chiamate *multilayer perceptron*. 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 dall'equazione (1.10). Le uscite della rete vengono ottenute tramite una seconda trasformazione, analoga alla prima, sui valori z_j ottenendo

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

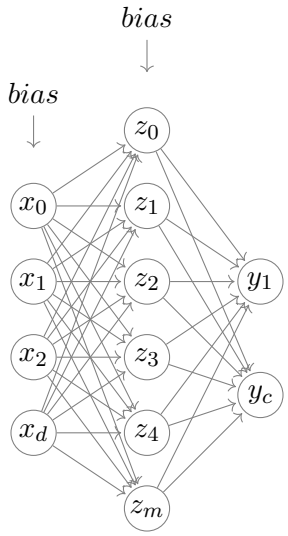

Figure 1.4: MLP a due strati

dove $\tilde{w}_{kj}$ rappresenta il peso del secondo strato che connette l'unità nascosta j all'unità di uscita k. Sostituendo l'equazione (1.10) nell'equazione (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 uscità 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 l'equazione (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 $\mathbf{w} = (w_1, .., w_d)$ che minimizzano la funzione di errore $E(\mathbf{w})$. 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 [4]

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 $(\mathbf{x}^q, \mathbf{t}^q)$ 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 η è il coefficiente di apprendimento (*learning rate*), più η è grande più imparera velocemente. 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* perchè permette al modello di convergere più velocemente.

Per migliorare la convergenza della rete abbiamo utilizzato la versione di discesa del gradiente con *momento* [3]. 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.

Stopping

In questa sezione vogliamo rispondere alla domanda: "quando fermiamo l'addestramento?". Negli esperimenti sono stati utilizzati diversi criteri, quali:

- Stop dopo un numero prefissate 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 che si aspetta un numero finito di epoche senza miglioramento delle prestazioni prima di interrompersi.

Chapter 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:
$$\frac{\partial \mathcal{L}}{\partial c_k} = \sum_{i,j} \frac{\partial \mathcal{L}}{\partial W_{ij}} 1(N_{ij} = k) \tag{2.4}$$

Chapter 3

Esperimenti

3.1 Problema del predecessore

Bibliography

[1] Nobody Jr. My article, 2006.

[2] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[3] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[4] David E. Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. 1 backpropagation : The basic theory. 2008.

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