UNIVERSITÀ DEGLI STUDI DI MILANO

Corso di Laurea Magistrale in Fisica

Infinity State Machine

Relatore:
Stefano Carrazza
Correlatore:
Daniel Krefl

Andrea Papaluca
Matricola: 902166

Anno Accademico 2018/2019

Indice

Capitolo 1

Introduzione

Quello che oggi conosciamo con il nome di machine learning, altro non è che l'evoluzione di ciò che era stato introdotto da Hopfield [3] attorno agli anni '80. Vale a dire un nuovo paradigma per la programmazione dei calcolatori che emulava lo schema di funzionamento del cervello umano.
In questo sistema, delle unità semplici, i neuroni, erano collegate ad altre unità tramite dei link, le sinapsi, e l'unico compito che svolgevano era quello di fare la media degli input che ricevevano da altri neuroni, pesata sulla forza delle singole connesioni. Poi, nel caso in cui questo numero superasse una certa soglia di attivazione, veniva trasmesso ai neuroni successivi.

Il sistema prese il nome di "rete neurale" e in realtà non fu accolto da subito con grandissimo calore, anzi i dieci anni successivi vengono spesso definiti "inverno dell'intelligenza artificiale", stando ad identificare proprio un periodo caratterizzato dal basso interesse per l'argomento.
È forse nel 1997, con la vittoria del calcolatore Deep Blue di IBM sul campione di scacchi Gary Kasparov, che si ebbe una nuova esplosione di interesse per l'intelligenza artificiale.
Il periodo che va dai primi anni duemila fino ai giorni nostri, è stato caratterizzato inoltre da un enorme sviluppo della complessità dei calcolatori e da una sempre maggiore disponibilità di dati, essenziali per il meccanismo di funzionamento delle reti neurali, due fattori che favorirono l'ascesa del machine learning non solo in ambito accademico, ma anche soprattutto in quello commerciale con le grandi multinazionali come Google, Facebook,

Amazon, ecc...

In questa tesi viene discussa una nuova architettura introdotta in [4], che si basa sul concetto di macchina Boltzmann introdotta da Geoffrey Hinton nel 1985 [1], del tutto affine a ciò che aveva presentato Hopfield nel 1982 ma con delle sue peculiarità e proprietà differenti.
In questo capitolo introduttivo descriverò prima l'architettura alla base della macchina di Boltzmann e in seguito introdurrò il concetto di "Riemann-Theta Boltzmann Machine", illustrandone le proprietà e le possibili applicazioni. L'obbiettivo ultimo di questa tesi è proprio la presentazione dei nuovi sviluppi di questa architettura, che riguardano la possibilità di generare densità di probabilità condizionate e cumulative distribution functions come vedremo poi nel capitolo 2. Infatti spesso, oltre alla necessità di modellizzare delle densità, si ha bisogno di ricavare delle quantità ad esse legate e perciò fa comodo avere a disposizione un unico modello che sia in grado assolvere a questo compito efficacemente.

1.1 La Macchina di Boltzmann

Una macchina di Boltzmann è un particolare tipo di architettura introdotta in [1], basata sul formalismo delle reti neurali di Hopfield [3].
La macchina consiste in due settori, quello visibile che è il layer di input, ovvero che accoglie nuovi dati all'interno della rete, e quello hidden che è il layer di uscita, cioè quello da cui si legge la risposta della macchina ai dati di input.
Ognuno dei due layer è composto da un certo numero di unità binarie, i nodi della rete, che quindi si possono trovare unicamente negli stati acceso/spento. Ogni unità è connessa ad altre unità dello stesso layer o del layer opposto, e ad ogni connesione è associato un peso w che rappresenta la forza del legame. Una rappresentazione della macchina si puó vedere in figura 1.1.

A partire dagli stati delle singole unità si può definire lo stato globale della macchina, a cui è possibile associare un numero che viene chiamato energia dello stato. Il ruolo delle singole unità diventa quindi quello di agire in modo da minimizzare quest'energia. In altre parole questa minimizzazione

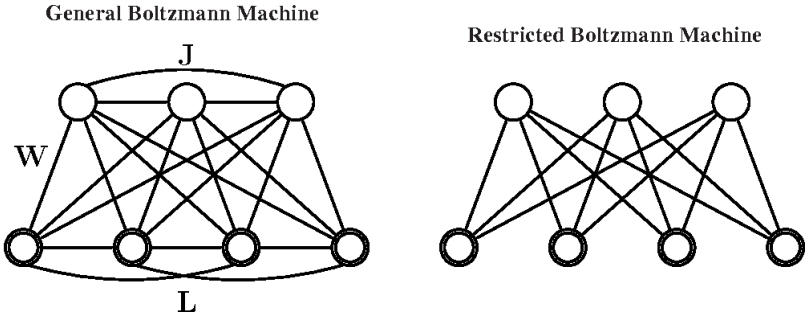


Figura 1.1: *Una General Boltzmann Machine a sinistra ed una Restricted Boltzman Machine a destra. Nel primo caso sono permesse anche connessioni tra i nodi dello stesso layer. Immagine presa da* [2].

rappresenta il tentativo della macchina di interpretare ed adattarsi il meglio possibile ai dati che le vengono forniti in input.

L'energia dello stato globale viene quindi definita in questo modo:

$$E = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i \tag{1.1}$$

dove s_i rappresenta lo stato del i-esima unità, $s_i = 1$ se è accesa $s_i = 0$ se è spenta, w_{ij} è il peso della connessione tra i nodi i e j, e θ_i è il treshold, ovvero la soglia di attivazione relativa al i-esima unità.
In realtà per semplificare la trattazione vengono aggiunte delle unità fittizie, dette bias, ciascuna connessa unicamente al relativo nodo. Infatti mantenendole sempre attive e impostando il peso della loro connessione come $w_i = \theta_i$, esse riproducono esattamente il comportamento dei treshold che non sono quindi più necessari. La nuova espressione dell'energia diventa perciò:

$$E = -\sum_{i<j} w_{ij} s_i s_j \tag{1.2}$$

mentre il gap di energia dovuto all'accensione/spegnimento della k-esima unità è dato semplicemente da:

$$\Delta E_k = \sum_i w_{ki} s_i. \tag{1.3}$$

1.1.1 Minimizzazione dell'energia

Ora che abbiamo introdotto a grandi linee la struttura del sistema, possiamo concentrarci sul problema della minimizzazione dell'energia.

L'approcio più semplice è sicuramente quello di muoversi nello spazio dei parametri seguendo la direzione opposta al gradiente della funzione energia, anche noto come "gradient descent". Quindi data l'energia in funzione dei parametri:

$$E(\mathbf{p}) = E(p_1, ..., p_n)$$

se ne calcola il gradiente:

$$\nabla E(\mathbf{p})$$

e si aggiorna il valore dei parametri allo step t muovendosi nella direzione opposta:

$$\mathbf{p}^{t+1} = \mathbf{p}^t - \eta \nabla E(\mathbf{p}) \tag{1.4}$$

dove η viene spesso definito "learning rate" e stabilisce semplicemente la dimensione di ciascuno step.

Il problema di questa strategia è che si rimane bloccati al primo minimo locale, che in alcuni casi potrebbe anche rappresentare una buona approssimazione del minimo globale, e quindi una buona rappresentazione dei dati ricevuti in input dalla macchina, ma in generale non è assolutamente vero.
Per ovviare a ciò si è pensato di permettere alla macchina di saltare occasionalmente a configurazioni con energia più alta, dove questo "occasionalmente" è legato al gap di energia del salto.
Infatti considerando la k-esima unità, con costo di attivazione/spegnimento ΔE_k dato dalla (1.3), decidiamo di impostarla su "on" ($s_k = 1$) con probabilità:

$$p_k = \frac{1}{1 + e^{-\Delta E_k / T}}$$

dove la T è semplicemente un parametro che ha il ruolo di temperatura del sistema.

In questo modo la macchina si comporta esattamente come un sistema di particelle a due livelli, $s_k = \{0, 1\}$, a contatto con un bagno termico alla temperatura T, e perciò lo stato globale della macchina seguirà la distribuzione

di Boltzmann:

$$P_\alpha = \frac{e^{-\beta E_\alpha}}{Z} \tag{1.5}$$

con $Z = \sum_\alpha e^{-\beta E_\alpha}$ funzione di partizione e $\beta = 1/k_b T$ come di consueto. Da questo deriva proprio il nome "Macchina di Boltzmann".
Perciò, raffreddando gradualmente il sistema analogamente a quanto si fa nel "simulated annealing", esso tenderà naturalmente allo stato con energia minore, ovvero alla configurazione della macchina che rappresenta al meglio i dati ricevuti in input.

In seguito sono state presentate diverse incarnazioni di questa architettura, in particolare quella con cui io ho lavorato viene detta "Riemann-Theta Boltzmann Machine" e nella prossima sezione ne introdurrò le caratteristiche.

1.2 Riemann-Theta Boltzmann Machine

Questo tipo di architettura è stata introdotta in [4] e ovviamente ricalca le caratteristiche della macchina di Boltzmann originale. La differenza fondamentale sta nel tipo di unità utilizzate, non più semplicemente binarie ma piuttosto reali o intere.
Già in [5] era stata introdotta una generalizzazione della macchina di Boltzmann classica, in cui le unità visibili prendevano valori in $\mathbb{R}$ mentre si mantenevano quelle nascoste binarie. Quindi quello che si potrebbe pensare è di cambiare anche il layer nascosto sostituendo le unità binarie con altre unità reali. Questo però, come dimostrato in [4], porterebbe ad una densità di probabilità semplicemente multivariata e quindi non particolarmente interessante.

La scelta che perciò viene adottata per la Riemann-Theta Boltzman Machine è quella di unità visibili che variano in $\mathbb{R}$ e unità nascoste che variano in $\mathbb{Z}$. Come vedremo nella prossima sezione questa scelta porta ad una densità di probabilità che è il prodotto di una parte multivariata moltiplicata per certe particolari funzioni dette "Riemann-Theta functions".

1.2.1 RTBM: il modello

Consideriamo una Riemann-Theta Boltzmann Machine, in breve RTBM, come quella rappresentata in figura 1.2. Come si vede abbiamo un layer visibile con un numero N_v di unità e un layer nascosto che ne contiene un numero N_h.
Tutti i nodi possono essere completamente interconnessi e sono ammesse anche le autoconnessioni. I pesi delle connessioni interne al layer visibile sono codificati nella matrice T di dimensione $N_v \times N_v$, mentre quelli relativi al layer nascosto si trovano nella matrice Q di dimensione $N_h \times N_h$. Infine W è la matrice $N_v \times N_h$ che contiene i pesi relativi alle connessioni tra i due layer.

Possiamo combinare le tre matrici dei pesi in un'unica matrice A di dimensione $(N_h + N_v) \times (N_h + N_v)$ fatta in questo modo:

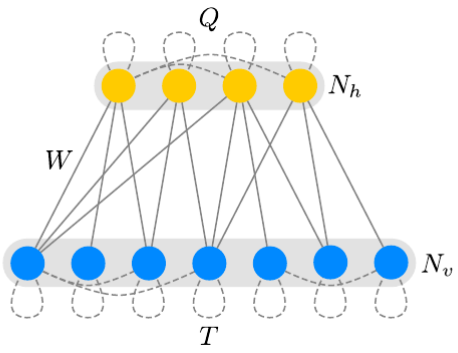


Figura 1.2: *Una RTBM con N_v unità visibili ed N_h unità nascoste. Sono ammesse tutte le possibili interconnessioni tra i nodi, incluse le autoconnessioni. Le matrici T, Q codificano i pesi delle connessioni rispettivamente per il layer visibile e quello nascosto, mentre la matrice W contiene i pesi delle connessioni tra i due layer. Immagine presa da* [4].

$$A = \begin{pmatrix} Q & W^t \\ W & T \end{pmatrix},$$

e oltre alla matrice dei parametri, dobbiamo tenere in considerazione anche il vettore dei bias:

$$B = \begin{pmatrix} B_h \\ B_v \end{pmatrix},$$

dove il vettore B_h è quello relativo ai nodi nascosti, mentre B_v è il corrispondente visibile.
Ora supponiamo di raggruppare gli stati delle unità visibili e nascoste in uno stesso vettore:

$$x = \begin{pmatrix} h \\ v \end{pmatrix},$$

in questo modo l'energia dello stato globale della macchina si scrive:

$$E(v,h) = \frac{1}{2} x^t A x + B^t x\,,$$

e in particolare:

$$x^t A x = v^t T v + h^t Q h + 2 v^t W h\,.$$

È importante notare che dobbiamo richiedere che la matrice A sia definita positiva per garantire che $E > 0$ anche per grandi x. Come vedremo nei prossimi capitoli questo vincolo è uno dei fattori che complica l'allenamento della macchina.

Ora che abbiamo l'energia, possiamo andare a ricavarci la funzione di partizione del sistema integrando su tutto lo spazio degli stati:

$$Z = \int_{-\infty}^{+\infty} [dv] \sum_{[h]} e^{-E(v,h)} \tag{1.6}$$

dove $[h]$ sta per $h_1, h_2, ..., h_{N_h}$ e $[dv]$ per $dv_1 dv_2 ... dv_{N_v}$.
In più ci fa comodo introdurre anche un'energia libera:

$$F(v) = -\log \sum_{[h]} e^{-E(v,h)} \tag{1.7}$$

tale che:

$$Z = \int_{-\infty}^{+\infty} [dv] e^{-F(v)} \,.$$

È a questo punto che fanno il loro ingresso le funzioni theta di Riemann, così definite:

$$\theta(z|\Omega) = \sum_{n \in \mathbb{Z}^{N_h}} e^{2\pi i (\frac{1}{2} n^t \Omega n + n^t z)} \,, \tag{1.8}$$

infatti ponendo:

$$\tilde{\theta}(z|\Omega) = \theta\left(\frac{z}{2\pi i} \middle| \frac{i\Omega}{2\pi}\right) \tag{1.9}$$

è possibile riscrivere l'energia libera come:

$$F(v) = \frac{1}{2} v^t T v + B_v^t v - \log \tilde{\theta}(v^t W + B_h^t | Q) \,. \tag{1.10}$$

Inoltre possiamo calcolare esplicitamente l'espressione della funzione di partizione integrando prima sul layer visibile. Infatti noto l'integrale gaussiano:

$$\int [dx] e^{-\frac{1}{2} x^t M x + y^t x} = \frac{(2\pi)^{N/2}}{\sqrt{\det M}} e^{\frac{1}{2} y^t M^{-1} y} \,,$$

la Z si scrive:

$$Z = \sum_{[h]} \frac{(2\pi)^{N_v/2}}{\sqrt{\det T}} e^{-\frac{1}{2} h^t Q h - B_h^t h + \frac{1}{2} (h^t W^t + B_v^t) T^{-1} (W h + B_v)} \,.$$

Ora non ci resta che sommare sul layer nascosto e sfruttando la definizione (1.8) otteniamo l'espressione chiusa:

$$Z = \frac{(2\pi)^{N_v/2}}{\sqrt{\det T}} e^{\frac{1}{2}B_v^t T^{-1} B_v} \tilde{\theta}(B_h^t - B_v^t T^{-1} W | Q - W^t T^{-1} W)\,. \tag{1.11}$$

Però a noi interessa la densitá di probabilitá, che é legata alla funzione di partizione da:

$$P(v,h) = \frac{e^{-E(v,h)}}{Z}\,,$$

o in alternativa marginalizzando su h:

$$P(v) = \frac{e^{-F(v)}}{Z}$$

con F dato dalla (1.10). Abbiamo le espressioni in forma chiusa sia per F che per Z e possiamo scrivere perció la $P(v)$:

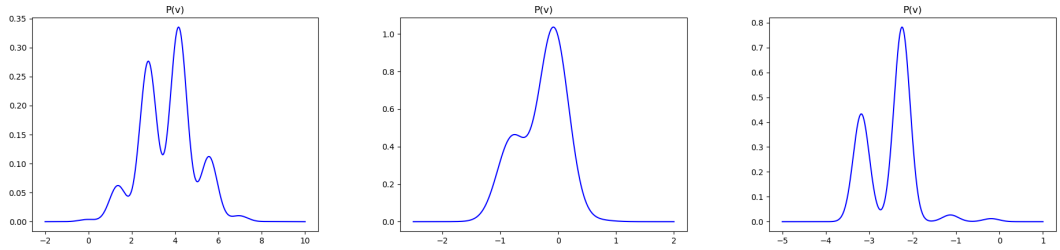
$$P(v) = \sqrt{\frac{\det T}{(2\pi)^{n_v}}} e^{-\frac{1}{2}v^t T v - B_v^t v - \frac{1}{2}B_v^t T^{-1} B_v} \frac{\tilde{\theta}(B_h^t + v^t W | Q)}{\tilde{\theta}(B_h^t - B_v^t T^{-1} W | Q - W^t T^{-1} W)}\,. \tag{1.12}$$

Come preannunciato la densità di probabilità è composta da una parte multivariata ed una legata alle funzioni theta di Riemann. Da notare che tutto questo ha senso finché T, Q e il complemento di Shur A/T sono definite positive, ovvero se la matrice A è definita positiva come avevamo richiesto inizialmente.
Nella figura 1.3 riporto tre esempi della $P(v)$ per tre diverse RTBM inizializzate a random. Ciascuna macchina aveva $N_v = 1$ ed $N_h = 2$, come si nota anche con così poche unità a disposizione la RTBM riesce a modellizzare distribuzioni piuttosto interessanti.

Ora che abbiamo introdotto la teoria alla base della RTBM, possiamo mostrare qualche esempio di utilizzo e alcune delle sue proprietà più interessanti.

1.2.2 Density estimation

Uno dei problemi principe per cui vengono usate in generale le macchine di Boltzmann, e in particolare le RTBM, è quello della density estimation.
Supponiamo di avere un array di dati disordinati e non etichettati (unlabelled), ovviamente non abbiamo a disposizione l'espressione analitica della

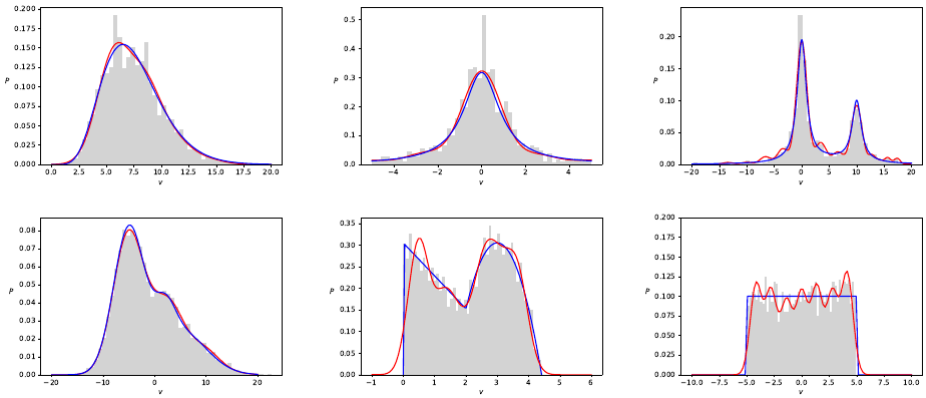


(a) (b) (c)

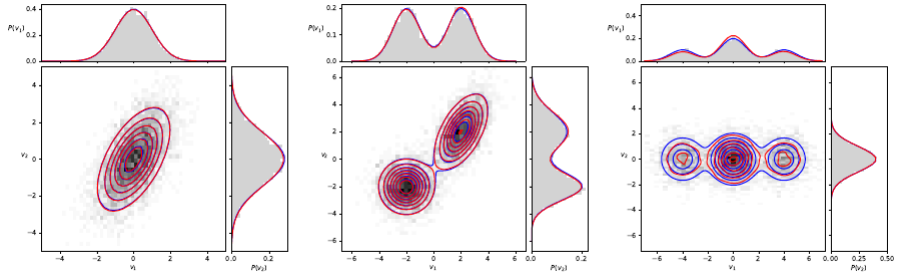
Figura 1.3: *Tre esempi delle $P(v)$ di tre diverse RTBM inizializzate a random. Ciascuna macchina aveva $N_v = 1$ e $N_h = 2$.*

distribuzione di questi dati e non è nemmeno detto che ne esista una. Quindi quello che ci proponiamo di fare è trovare un modello che li rappresenti al meglio.
In pratica è il problema di codificare l'informazione contenuta nei dati e va anche sotto il nome di "dimensionality reduction", poiché tutta l'informazione viene memorizzata nel modello che li rappresenta, che presumibilmente avrà dimensionalità minore.
I modelli che affrontano questo tipo di problemi vengono detti "autoencoders" e tipicamente utilizzano tecniche di machine learning e in particolare di quella branca che viene chiamata learning non supervisionato.

La RTBM quindi può essere usata a tutti gli effetti come un autoencoder, e in linea di principio anche piuttosto buono. Infatti grazie alla complessità delle sue singole unità, anche con pochi nodi, e quindi pochi parametri, riesce a modellizzare distribuzioni non banali.
Abbiamo quindi il vantaggio di una enorme potenza di modellizzazione, al costo però, come vedremo, di una maggiore complessità in fase di allenamento. Non bisogna dimenticare inoltre, che le funzioni theta di Riemann possono essere piuttosto "costose" dal punto di vista computazionale, rendendo difficile aumentare di molto la dimensione della rete.
In ogni caso alcuni esempi, tratti da [4] e riportati in figura 1.4, mostrano come anche solo una macchina con $N_v = 1$ e $N_h = 2$ riesca modellizzare distribuzioni assolutamente non banali.



(a) *Delle RTBM allenate su alcune distribuzioni* 1*-D, in ogni esempio si ha* $N_v = 1$ *mentre* N_h *va da* 2 *a* 4 *in base al caso.*



(b) *Delle RTBM allenate su alcune distribuzioni* 2*-D. Nel primo e nell'ultimo caso sono state usate delle singole RTBM rispettivamente con* $N_h = 1$ *e* $N_h = 2$*, mentre per l'esempio al centro è stato usato un layer di due RTBM con* $N_h = 1$*. In tutti gli esempi abbiamo* $N_v = 2$*.*

Figura 1.4: *Density estimation di alcune distribuzioni* 1*-D (a) e* 2*-D (b). In blu è rappresentata la curva analitica usata per generare i dati rappresentati negli istogrammi in grigio, mentre la curva rossa è la RTBM allenata. Immagine presa da* [4].

1.2.3 Sampling

Uno strumento molto utile a disposizione delle RTBM è quello di poter generare sampling di dati, molti degli esempi mostrati nel capitolo 2 fanno proprio affidamento a questo. L'idea di base è che una volta allenata una macchina su una certa distribuzione, essa può generare un sampling di dati che segua tale distribuzione.
Questo è uno strumento senza dubbio molto utile, poichè mentre per le distribuzioni di cui è nota l'espressione analitica generare un sample è un problema tutto sommato abbastanza banale, il discorso cambia completamente per quelle di cui non si conosce l'espressione analitica, spesso inoltre non è nemmeno detto ne abbiano una.
La descrizione approfondita del meccanismo e la trattazione rigorosa si possono trovare in [6], quì io mi limiterò a riassumerne brevemente i punti chiave.

A partire dall' espressione della probabilità del layer nascosto:

$$P(h) = \frac{e^{-\frac{1}{2}h^t(Q-W^tT^{-1}W)h-(B_h^t-B_v^tT^{-1}W)h}}{\tilde{\theta}(B_h^t - B_v^tT^{-1}W|Q - W^tT^{-1}W)} \tag{1.13}$$

che si ricava in maniera del tutto analoga a quanto fatto per la (1.12), è possibile trovare la probabilità condizionata:

$$P(v|h) = \frac{P(v,h)}{P(h)} = \frac{1}{(2\pi)^{N_v/2}\sqrt{\det T^{-1}}} \quad e^{-\frac{1}{2}(v-\mu(h))^tT(v-\mu(h))} \tag{1.14}$$

dove abbiamo definito $\mu(h) = -T^{-1}(Wh + B_v)$.
Ora usando la (1.13) e la (1.14) appena scritte è facile convincersi che possiamo riscrivere la $P(v)$ in questo modo:

$$P(v) = \sum_{[h]} P(v|h)P(h). \tag{1.15}$$

In altre parole abbiamo trovato che, poiché la $P(v|h)$ è semplicemente una multivariata, la densità di probabilità del layer visibile non è altro che una mistura gaussiana pesata dalla $P(h)$.
Di conseguenza se generiamo un sample $h \sim P(h)$ e successivamente un sample $v \sim P(v|h)$, avremo che v è distribuito secondo la $P(v)$ come volevamo. Inoltre estrarre un sample dalla $P(v|h)$ è relativamente semplice

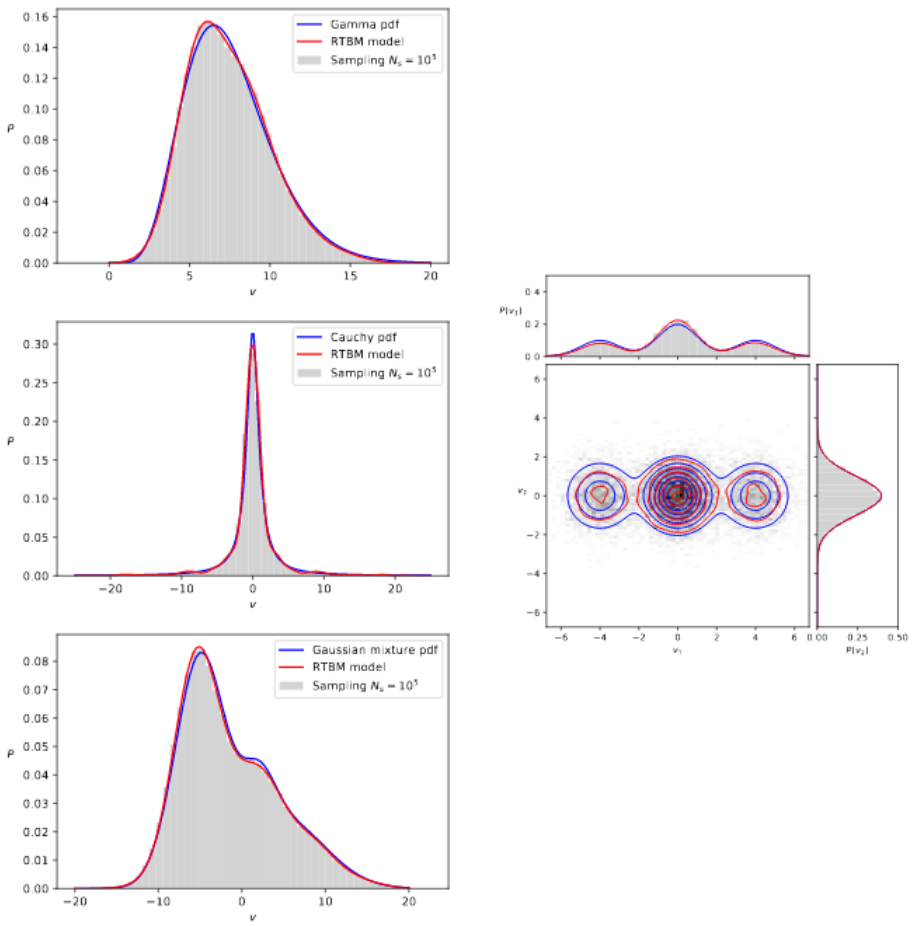


Figura 1.5: *A sinistra tre distribuzioni monodimensionali disegnate in blu, una Gamma, una Cauchy ed una mistura Gaussiana. In rosso è riportato il risultato dell'allenamento di tre RTBM rispettivamente con* $N_h = 2$, $N_h = 3$ *e* $N_h = 3$. *La macchina allenata poi è stata usata per generare un sample di* 10^5 *punti, rappresentato dall'istogramma in grigio.*
Analogamente sulla destra è mostrata una mistura multivariata bidimensionale in blu, su di essa è stata allenata una RTBM con $N_h = 2$ *e successivamente quest'ultima è stata usata per generare il sample di dati rappresentato dall'istogramma in grigio. In questo caso sono rappresentate anche le proiezioni sui due assi. Immagine presa da [6]*

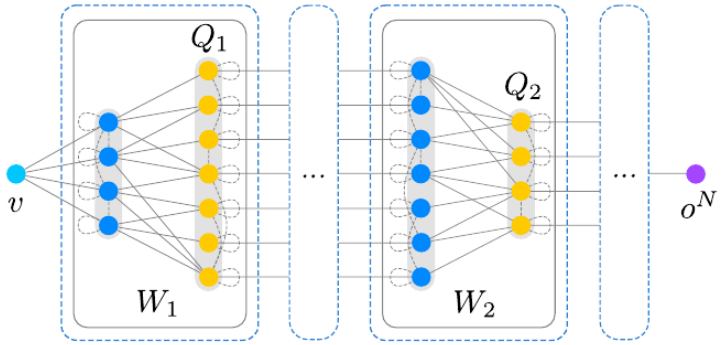


Figura 1.6: *Rappresentazione schematica di una TNN, "theta neural network", le singole RTBM possono essere usate come layer di ordinarie "feed forward neural network". L'output dell' i-esimo nodo del layer è dato dal valore di aspettazione (1.18). Immagine presa da [4].*

essendo una multivariata, è più complicato per quanto riguarda la $P(h)$ per via della presenza di un theta al denominatore, ma tuttavia come viene spiegato in [6] è possibile farlo efficientemente.

In figura 1.5 ho riportato alcuni esempi tratti da [6], di RTBM prima allenate su distribuzioni monodimensionali e bidimensionali e poi usate per generare dei sample.

1.2.4 Theta Neural Networks

È possibile connettere in serie più RTBM per formare una "feed forward neural network", come mostrato in figura 1.6. Infatti nota la probabilità condizionata:

$$P(h|v) = \frac{P(v,h)}{P(v)} = \frac{e^{-\frac{1}{2}h^tQh-(v^tW+B_h^)h}}{\tilde{\theta}(v^tW+B_h^t|Q)}\,, \tag{1.16}$$

possiamo calcolare il valore di aspettazione di ciascun nodo di output:

$$E(h_i|v) = \frac{1}{\tilde{\theta}(v^tW+B_h^t|Q)} \sum_{[h]} h_i\, e^{-\frac{1}{2}h^tQh-(v^tW+B_h^)h}\,, \tag{1.17}$$

o facendo un ulteriore step:

$$E(h_i|v) = -\frac{1}{2\pi i}\,\frac{\nabla_i\tilde{\theta}(v^tW+B_h^t|Q)}{\tilde{\theta}(v^tW+B_h^t|Q)}\,, \tag{1.18}$$

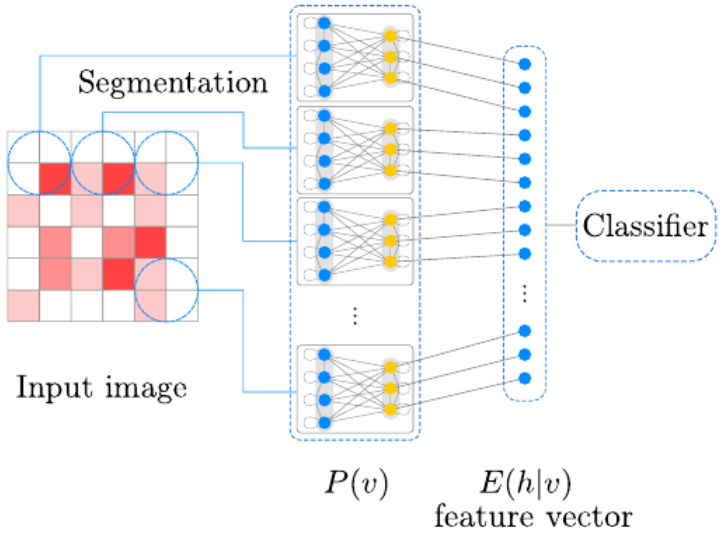


Figura 1.7: *Un classificatore costituito da un layer di RTBM, usato per il preprocessing dell'immagine, connesso all'architettura necessaria per la classificazione. Immagine presa da [4].*

e questi valori di aspettazione possono essere usati come funzioni di attivazione della rete.

In pratica, se abbiamo una RTBM con Q di dimensione $N_h \times N_h$, possiamo costruire un layer di rete neurale formato da N_h nodi, dove l'output di ciascun nodo sarà dato da $E(h_i|v)$. Questi layer artificiali poi possono essere concatenati ad altri layer dello stesso tipo o a normali layer di reti neurali ordinarie, a patto che le dimensioni coincidano, per formare quelle che vengono chiamate TNN, "theta neural networks".

Ciò può essere utile per risolvere problemi di classificazione. Infatti possiamo costruire una TNN formata da un layer artificiale di RTBM che funga da preprocessore per un'altra rete neurale, che poi avrà il ruolo di classificatore. Una rappresentazione schematica dell'idea è data in figura 1.7, per una trattazione esaustiva dell'esempio rimando comunque a [4].

1.2.5 Trasformazioni Affini

Infine c'è un'ultima proprietà degna di nota che le RTBM ereditano dalla multivariata, ovvero la chiusura rispetto alle trasformazioni affini. Infatti come viene spiegato in [6], presa una trasformazione affine del tipo:

$$\mathbf{w} = A\mathbf{v} + b$$

si può dimostrare che $\mathbf{w} \sim P_{A,b}(\mathbf{v})$, ovvero $\mathbf{w}$ è distribuito come la densità di probabilità di una RTBM, a patto di eseguire la seguente riparametrizzazione:

$$\begin{aligned} T^{-1} &\longrightarrow AT^{-1}A^t \\ W &\longrightarrow (A^+)^t W \\ B_v &\longrightarrow (A^+)^t B_v - Tb \\ B_h &\longrightarrow B_h - W^t b \end{aligned} \tag{1.19}$$

dove A^+ non è altro che la pseudo-inversa sinistra, definita come:

$$A^+ = (A^t A)^{-1} A^t$$

e in particolare quindi $A^+ A = 1$ e $A^t (A^+)^t = 1$.
Questo significa che una volta allenata una RTBM su una certa distribuzione, è possibile ruotarla e traslarla a piacimento al costo semplicemente di una riparametrizzazione, in pratica al costo di qualche prodotto di matrice. In primo luogo ciò potrebbe essere utile per fittare un'altra distribuzione senza dover allenare nuovamente la machina, ma semplicemente rototraslandola. Chiaramente si avrebbe un buon fitting solo se la nuova distribuzione si presentasse con la stessa shape di quella originale, ma sarebbe comunque utile nel caso fosse sufficiente un'approsimazione più grossolana.
Inoltre la capacità delle RTBM di generare sampling di dati combinata con questa proprietà, ci danno veramente delle enormi possibilità e flessibilità nella generazione di dati, il tutto con un singolo modello e con un costo computazionale relativamente basso.
Anche in questo caso riporto un esempio di una RTBM prima allenata su una mistura multivariata e poi ruotata, scalata e traslata. Come si vede in figura 1.8 c'è una perfetta corrispondenza trai il sample di dati a cui è stata

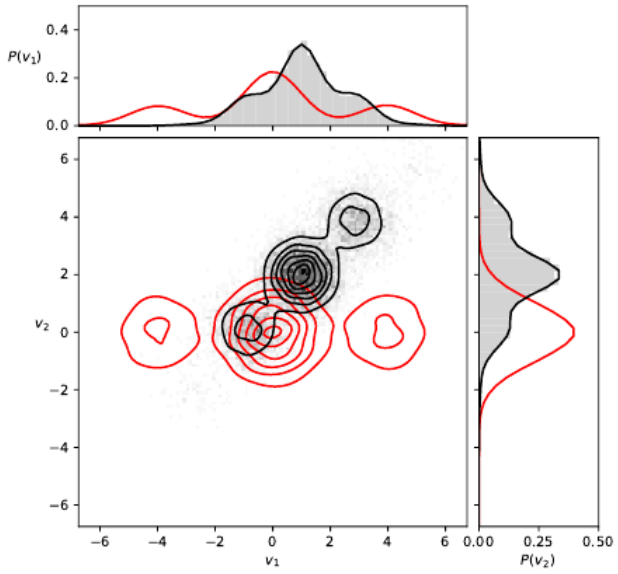


Figura 1.8: *La RTBM usata per generare il sample bidimensionale di figura 1.5, quì è stata ruotata, scalata e traslata, fino ad ottenere la curva riportata in nero. Successivamente è poi stato generato un nuovo sample a partire dalla RTBM trasformata. Per completezza sono riportate anche le proiezioni sui due assi. Immagine presa da [6].*

applicata la trasformazione affine e la macchina trasformata.

Queste che ho appena descritto sono le principali applicazioni, o perlomeno quelle più interessanti, per cui è stata utilizzata la RTBM fino ad oggi. Nel prossimo capitolo invece introdurrò due nuove proprietà, che come nel caso dell'invarianza sotto trasformazioni affini, permettono di introdurre due nuovi strumenti a disposizione delle RTBM.
Questo è proprio l'argomento che ha impegnato la prima metà del mio lavoro di tesi e come vedremo lascia trasparire le grandi potenzialità di questo modello, ma inevitabilmente anche le complicazioni che insorgono nell'allenamento, a cui già ho accennato in questo capitolo e che discuterò più approfonditamente nei prossimi.

Capitolo 2

Due Nuove Proprietà

Come anticipato, nella prima parte del lavoro di tesi mi sono occupato di ricavare e sviluppare due nuove utili proprietà della RTBM. Proprietà che poi ovviamente si traducono in utili strumenti a sua disposizione.
La prima che tratterò sarà la probabilità condizionata, introducendo inizialmente il problema che ci proponiamo di risolvere contestuallizzato all'ambito delle RTBM. A seguire darò una breve derivazione della nuova proprietà in oggetto e infine ne mostrerò qualche esempio di utilizzo.

La seconda sezione sarà invece dedicata alla "Cumulative Distribution Function", in breve CDF. Come per la condizionata, prima presenterò il problema in generale, mostrando per quale motivo possa essere utile la RTBM. Subito dopo presenterò la deduzione dell'espressione della CDF per una RTBM, e infine ne mostrerò di nuovo qualche esempio.

2.1 Probabilità Condizionata

2.1.1 Motivazione e Problema

Il problema di modellizzare una densità di probabilità di un certo set di dati è già di per se non banale, anche a basse dimensioni. Inoltre spesso è utile estrarre altre quantità legate alla funzione densità, come per esempio delle probabilità condizionate o la CDF.
Come abbiamo visto nella sezione 1.2.2 le RTBM sono in grado di modellizzare densità di probabilità efficacemente, perciò ora ci chiediamo se una

volta imparata una certa distribuzione, sia possibile ricavare dalla RTBM altre quantità, come ad esempio una probabilità condizionata.
Questo è interessante, perchè fino ad oggi il metodo usato per ricavare le condizionate è quello delle copulae gaussiane, che funziona bene entro certi limiti ma diventa difficile da utilizzare appena si considerano delle densità più articolate. Perciò le RTBM rappresenterebbero un nuovo modo, potenzialmente più efficace vista la loro capacità di modellizzazione, per generare tali condizionate. Vediamo quindi come formulare il problema nel contesto di una RTBM.

Immaginiamo di avere una RTBM che modellizzi una distribuzione di nostro interesse con il suo $P(v)$ dato dalla (1.12). Se fattorizziamo v nel modo seguente:

$$v = \begin{pmatrix} y \\ d \end{pmatrix}$$

con y e d vettori colonna di dimensioni arbitrarie, ma pur sempre pari ad N_v in totale, è possibile ricavare in qualche modo la probabilità condizionata $P(y|d)$? O in altre parole, possiamo ricavare la proiezione multidimensionale normalizzata della distribuzione che abbiamo imparato?
La risposta è sì, e nella prossima sezione mostrerò non solo che è possibile, ma anche che tale probabilità è modellizzata da un'altra RTBM riparametrizzata a partire da quella originale, ovvero che le RTBM sono chiuse rispetto a questo tipo di proiezione.

2.1.2 Deduzione Matematica

Consideriamo il caso di y con dimensione $k > 1$ e d con dimensione n, in totale abbiamo v di dimensione $n + k$ e perciò stiamo considerando una RTBM con $N_v = n + k$ ed N_h arbitrario.
Per ricavare la proiezione, ovvero la probabilità dei primi k nodi condizionata ai secondi n, ci fa comodo dividere i bias e le matrici W e T in una parte relativa ad y ed una relativa a d.

Quindi preso:

$$v = \begin{pmatrix} y_0 \\ . \\ . \\ y_{k-1} \\ d_0 \\ . \\ . \\ d_{n-1} \end{pmatrix},$$

viene naturale dividere T nel modo seguente:

$$T = \left(\begin{array}{c|c} \bar{T}_0 & \bar{T}_1^t \\ \hline \bar{T}_1 & \tilde{T} \end{array}\right),$$

dove $\bar{T}_0$ è una matrice quadrata $k \times k$, $\bar{T}_1$ è una matrice rettangolare $n \times k$ e $\tilde{T}$ è una matrice quadrata $n \times n$.
Seguendo questa linea scomponiamo anche la matrice W e il vettore dei bias B_v in modo simile:

$$W = \left(\begin{array}{c} W_0 \\ \hline \\ W_1 \\ \\ \end{array}\right),$$

$$B_v = \begin{pmatrix} B_{v,0} \\ B_{v,1} \end{pmatrix},$$

con W_0 matrice rettangolare $k \times N_h$, W_1 matrice rettangolare $n \times N_h$, $B_{v,0}$ vettore colonna di dimensione k e $B_{v,1}$ vettore colonna di dimensione n.

Ora che abbiamo diviso tutto il necessario in una parte relativa ad y ed una relativa a d, possiamo andare a riscrivere la (1.12) che a sua volta sarà separabile in una parte dipendente da y ed una indipendente.
Se infatti sfruttiamo la nuova definizione di T possiamo ricalcolare il termine $v^t T v$, trovando:

$$v^t T v = y^t \bar{T}_0 y + 2 d^t \bar{T}_1 y + d^t \tilde{T} d\,, \tag{2.1}$$

dove abbiamo usato che $d^t\bar{T}_1 y = y^t\bar{T}_1^t d$, poiché essendo semplicemente degli scalari coincidono con il loro trasposto. In modo simile possiamo riscrivere i termini $B_v^t v$ e $v^t W$ ottenendo:

$$B_v^t v = B_{v,0}^t y + B_{v,1}^t d\,, \tag{2.2}$$

$$v^t W = y^t W_0 + d^t W_1\,. \tag{2.3}$$

A questo punto abbiamo riscritto tutti i termini di (1.12) in cui compare v, in linea di principio bisognerebbe riscrivere anche i termini $B_v^t T^{-1} B_v$, $B_v^t T^{-1} W$ e $W^t T^{-1} W$ ma come vedremo più avanti si semplificano nel rapporto $P(y,d)/P(d)$ e perciò non ci interessano ai fini del calcolo della $P(y|d)$. Quindi sostituendo le relazioni (2.1), (2.2) e (2.3) nella (1.12) si trova:

$$P(y,d) = \sqrt{\frac{\det T}{(2\pi)^{Nv}}} \quad e^{-\frac{1}{2}y^t\bar{T}_0 y - d^t\bar{T}_1 y - \frac{1}{2}d^t\tilde{T}d - B_{v,0}^t y - B_{v,1}^t d - \frac{1}{2}B_v^t T^{-1} B_v}\cdot \\ \cdot \frac{\tilde{\theta}(B_h^t + y^t W_0 + d^t W_1 | Q)}{\tilde{\theta}(B_h^t - B_v^t T^{-1} W | Q - W^t T^{-1} W)}. \tag{2.4}$$

Per ottenere la marginale $P(d)$ ora è sufficiente integrare su tutto lo spazio relativo al vettore y:

$$P(d) = \int d^k y P(y,d) = \int dy_1 \int dy_2 \cdots \int dy_k P(y,d)\,,$$

che una volta esplicitate le funzioni $\tilde{\theta}$:

$$\tilde{\theta}(z^t|\Omega) = \sum_{n\in\mathbb{Z}^{Nh}} e^{n^t z - \frac{1}{2}n^t\Omega n}$$

e tolta la parte indipendente da y, si riduce al calcolo del seguente integrale:

$$I = \int d^k y \quad e^{-\frac{1}{2}y^t\bar{T}_0 y - d^t\bar{T}_1 y - B_{v,0}^t y + n^t W_0^t y}.$$

Questo integrale è noto ed è semplicemente la generalizzazione dell'integrale gaussiano, con soluzione:

$$\int d^n x \quad e^{-\frac{1}{2}x^t A x + b^t x} = \sqrt{\frac{(2\pi)^n}{\det A}} \quad e^{\frac{1}{2}b^t A^{-1} b}.$$

Nel nostro caso quindi quello che troviamo è:

$$I = \sqrt{\frac{(2\pi)^k}{\det \bar{T}_0}} \quad e^{\frac{1}{2}n^t W_0^t \bar{T}_0^{-1} W_0 n - n^t W_0^t \bar{T}_0^{-1}(B_{v,0} + \bar{T}_1^t d) + (B_{v,0} + \bar{T}_1^t d)^t \bar{T}_0^{-1}(B_{v,0} + \bar{T}_1^t d)}\,,$$

e possiamo fare un ulteriore step.
Infatti raggruppando i termini in n sotto la sommatoria, si nota che è possibile scrivere tutto nuovamente sotto forma di una funzione $\tilde{\theta}$, ottenendo finalmente:

$$P(d) = \sqrt{\frac{\det T}{(2\pi)^{Nv}}} \quad e^{-\frac{1}{2}d^t\tilde{T}d - B_{v,1}^t d - \frac{1}{2}B_v^t T^{-1}B_v} \sqrt{\frac{(2\pi)^k}{\det \bar{T}_0}} \quad e^{(B_{v,0}+\bar{T}_1^t d)^t \bar{T}_0^{-1}(B_{v,0}+\bar{T}_1^t d)} \cdot \\ \cdot \frac{\tilde{\theta}(B_h^t + d^t W_1 - (B_{v,0}+\bar{T}_1^t d)^t \bar{T}_0^{-1} W_0 | Q - W_0^t \bar{T}_0^{-1} W_0)}{\tilde{\theta}(B_h^t - B_v^t T^{-1} W | Q - W^t T^{-1} W)} \cdot \tag{2.5}$$

L'ultima cosa che ci resta da fare è dividere la (2.4) per la (2.5). Come preannunciato alcuni termini si semplificano ed otteniamo come risultato la probabilità condizionata che stavamo cercando:

$$P(y|d) = \sqrt{\frac{\det \bar{T}_0}{(2\pi)^k}} \quad e^{-\frac{1}{2}y^t\bar{T}_0 y - (B_{v,0}+\bar{T}_1^t d)^t y - \frac{1}{2}(B_{v,0}+\bar{T}_1^t d)^t \bar{T}_o^{-1}(B_{v,0}+\bar{T}_1^t d)} \cdot \\ \cdot \frac{\tilde{\theta}(B_h^t + d^t W_1 + y^t W_0 | Q)}{\tilde{\theta}(B_h^t + d^t W_1 - (B_{v,0}+\bar{T}_1^t d)^t \bar{T}_0^{-1} W_0 | Q - W_0^t \bar{T}_0^{-1} W_0)}. \tag{2.6}$$

Inoltre possiamo fare un ulteriore osservazione. Infatti si nota che la $P(y|d)$ scritta in questo modo, corrisponde alla $P(v)$ di una RTBM con $N_v = k$, a patto di fare la seguente riparametrizzazione:

$$\begin{aligned} T &\to \bar{T}_0\,, \\ W &\to W_0\,, \\ B_v &\to B_{v,0} + \bar{T}_1^t d\,, \\ B_h &\to B_h + W_1^t d\,. \end{aligned} \tag{2.7}$$

Questo significa che una volta allenata una RTBM su di una distribuzione multidimensionale, possiamo immediatamente generare qualsiasi probabilità condizionata semplicemente creando una nuova RTBM ed inizializzandola secondo (2.7).

2.1.3 Esempi

In questa sezione riporterò qualche esempio della proprietà appena ricavata. Sebbene in linea di principio quello che abbiamo trovato vale per qualsiasi

dimensione n arbitraria, per ovvi motivi il caso $n > 3$ sarebbe difficile da visualizzare. Perciò mi sono limitato principalmente ad esempi di distribuzioni bidimensionali e per completezza ho aggiunto anche un esempio di distribuzione tridimensionale, già non semplicissimo da interpretare dato che avremmo bisogno di quattro dimensioni per visualizzare la funzione densità di probabilità.

Un altro problema risiede nel fatto che non sono molte le distribuzioni di cui è nota un'espressione analitica della condizionata, proprio per questo inoltre è molto utile questa proprietà. Ciò però implica che le condizionate generate dalle RTBM, nella maggior parte dei casi possono essere confrontate esclusivamente con la condizionata empirica. Fortunamente però, per il primo esempio che prendiamo in considerazione, è stata trovata l'espressione analitica della condizionata [7] [8].

Multivariate Student t Distribution

La definizione di distribuzione t di dimensione p è data dalla seguente espressione:

$$f(x) = \frac{\Gamma((v+p)/2)}{\Gamma(v/2)\,(v\pi)^{\frac{p}{2}}\,|\Sigma|^{\frac{1}{2}}}\left[1 + \frac{1}{v}\,(x-\mu)^t\Sigma^{-1}(x-\mu)\right]^{-\frac{v+p}{2}}, \tag{2.8}$$

dove il vettore μ rappresenta il centro della distribuzione, Σ è la matrice che ne modifica la shape e v è semplicemente un numero che indica quanto essa sia vicina ad una multivariata. Per $v \to \infty$ infatti, si ha che $f \sim N_p(\mu, \Sigma)$. Inoltre si può dimostrare [7] [8] che la condizionata è ancora una distribuzione t, ovvero definito $x = (x_1, x_2)$ si trova:

$$x_2 \,|\, x_1 \;\sim\; t_{p_2}\left(\mu_{2|1}, \frac{v+d_1}{v+p_1}\,\Sigma_{22|1}, v+p_1\right), \tag{2.9}$$

dove abbiamo fatto uso delle seguenti definizioni:

$$\begin{aligned}
\Sigma &= \left(\begin{array}{c|c} \Sigma_{11} & \Sigma_{12} \\ \hline \Sigma_{21} & \Sigma_{22} \end{array}\right), \\
\mu_{2|1} &= \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(x_1 - \mu_1)\,, \\
d_1 &= (x_1-\mu_1)^t\Sigma_{11}^{-1}(x_1-\mu_1)\,, \\
\Sigma_{22|1} &= \Sigma_{22} - \sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}\,.
\end{aligned} \tag{2.10}$$

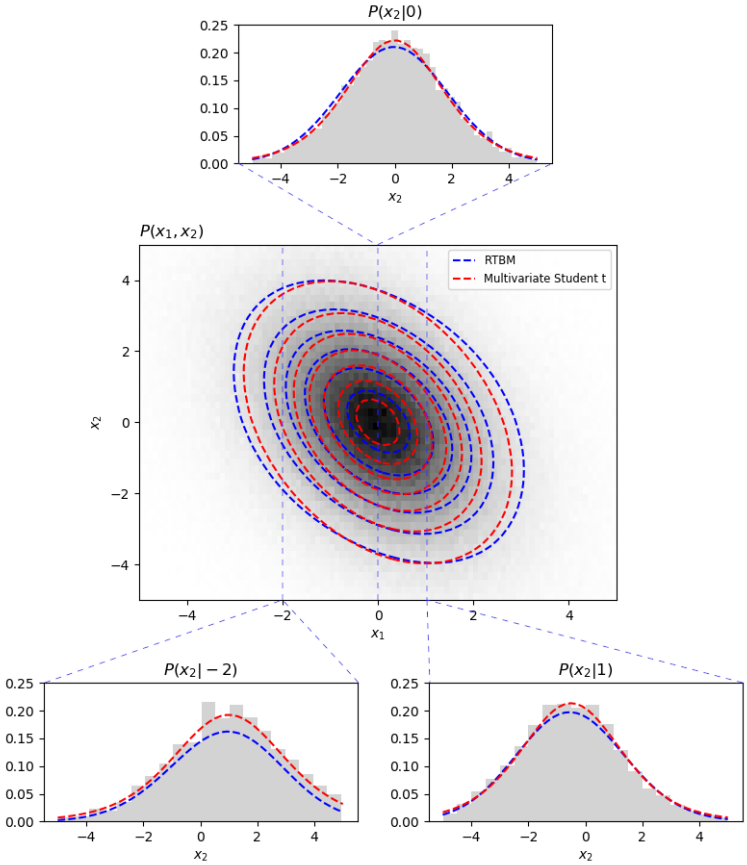


Figura 2.1: *Al centro l'istogramma bidimensionale di una distribuzione Multivariate Student t definita dalla (2.8), con parametri scelti secondo la (2.11). Sovrapposti rispettivamente: in blu il contour plot della t analitica e in rosso quello della RTBM (2.12). Sopra e sotto il confronto tra la condizionata analitica (2.9) e quella generata usando la (2.7).*

Multivariate Student t		2D Example		3D Example	
Conditional	MSE	Conditional	MSE	Conditional	MSE
$P(x_2\|-2)$	$3.255 \cdot 10^{-4}$	$P(y\|2)$	$1.176 \cdot 10^{-4}$	$P(y_1, y_2\|-0.4)$	$4.953 \cdot 10^{-5}$
$P(x_2\|0)$	$4.083 \cdot 10^{-5}$	$P(y\|1.3)$	$6.888 \cdot 10^{-5}$	$P(y_1, y_2\|-0.6)$	$6.775 \cdot 10^{-5}$
$P(x_2\|1)$	$4.433 \cdot 10^{-5}$	$P(y\|0.4)$	$2.538 \cdot 10^{-4}$	$P(y_1, y_2\|-0.8)$	$5.304 \cdot 10^{-5}$

Tabella 2.1: *MSE calcolato per ciascuna delle condizionate riportate nell'esempio della student t, in quello bidimensionale e in quello tridimensionale.*

Questo significa che abbiamo a disposizione la forma analitica della condizionata e possiamo perciò confrontarla con quella generata usando la (2.7), come viene mostrato in figura 2.1.
Per la student t ho scelto i parametri:

$$\mu = (0, 0), \quad \Sigma = \begin{pmatrix} 2 & -1 \\ -1 & 4 \end{pmatrix}, \quad v = 6, \tag{2.11}$$

e per modellizzarla ho allenato una RTBM $N_v = 2$, $N_h = 2$ utilizzando l'algoritmo genetico *Covariance Matrix Approximation*, in breve CMA-ES. Utilizzando come funzione costo la loglikelyhood ho trovato la migliore soluzione ad un valore $\sim 1.3 \cdot 10^4$, corrispondente ai seguenti valori dei parametri:

$$\begin{aligned} W &= \begin{pmatrix} -1.11 & 1.02 \\ -0.66 & 0.60 \end{pmatrix}, \quad T = \begin{pmatrix} 0.56 & 0.18 \\ 0.18 & 0.30 \end{pmatrix}, \quad B_v = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \\ B_h &= \begin{pmatrix} 8.22 \\ 17.40 \end{pmatrix}, \quad Q = \begin{pmatrix} 24.15 & -0.44 \\ -0.44 & 41.57 \end{pmatrix}. \end{aligned} \tag{2.12}$$

Inoltre per quantificare l'errore che si compie nel modellizzare le distribuzioni condizionate con la RTBM, ho calcolato l'MSE, "Mean Squared Error", per i tre esempi riportati in figura 2.1. I valori sono riportati in tabella 2.1.

Distribuzione 2D

Quella che mi accingo a mostrare ora invece, è una distribuzione con shape molto più complicata rispetto alla Student t, perciò ovviamente non abbiamo a disposizione l'espressione analitica delle condizionate, ma in realtà

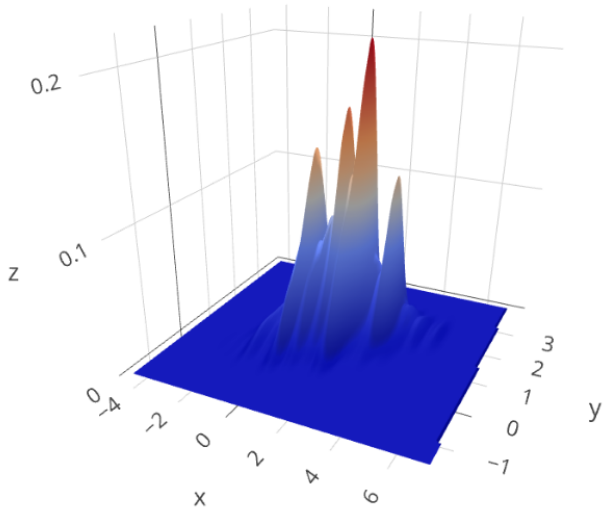


Figura 2.2: *Rappresentazione tridimensionale della distribuzione corrispondente ad una RTBM $N_v = 2$, $N_h = 4$ e T diagonale, con la scelta di parametri (2.13).*

nemmeno quella della distribuzione in se.
Infatti questo esempio è stato generato inizializzando a mano una RTBM di architettura $N_v = 2$, $N_h = 4$ e T diagonale, con la seguente scelta per i parametri:

$$W = \begin{pmatrix} 18.54 & 3.02 & -12.89 & -5.45 \\ 0.46 & 1.01 & -1.32 & -5.54 \end{pmatrix}, \quad T = \begin{pmatrix} 28.77 & 0 \\ 0 & 6.3 \end{pmatrix}, \quad B_v = \begin{pmatrix} -1.76 \\ -2.69 \end{pmatrix},$$
$$B_h = \begin{pmatrix} -0.31 \\ 2.29 \\ 1.65 \\ -2.73 \end{pmatrix}, \quad Q = \begin{pmatrix} 15.48 & 8.82 & -3.19 & -3.67 \\ 8.82 & 17.99 & 8.94 & -4.04 \\ -3.19 & 8.94 & 15.74 & 4.14 \\ -3.67 & -4.04 & 4.14 & -5.54 \end{pmatrix}. \tag{2.13}$$

Una rappresentazione tridimensionale della distribuzione è data in figura 2.2.

A partire dalla RTBM così inizializzata, ho generato un sample di dati come spiegato nella sezione 1.2.3 [6], di cui è mostrato l'istogramma bidimensionale in figura 2.3. Successivamente ho ricavato le condizionate rispetto all'una e all'altra variabile in diversi punti usando la (2.7) e le ho confrontate

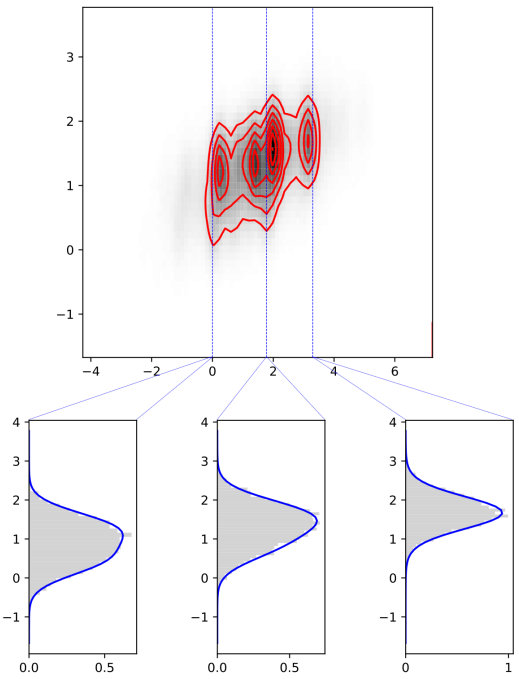
con il relativo istogramma monodimensionale.
Da notare che la (2.7) fa riferimento al caso in cui si voglia generare la condizionata rispetto alle ultime n variabili, ovvero rispetto alla y in questo caso. Quindi per ricavare le condizionate rispetto alla x mostrate in figura 2.3, c'è stato prima bisogno di ruotare la RTBM di un angolo di $\pi/2$. Fortunatamente come spiegato nella sezione 1.2.5, ciò è possibile grazie alla invarianza delle RTBM rispetto alle trasformazioni affini [6].

Distribuzione 3D

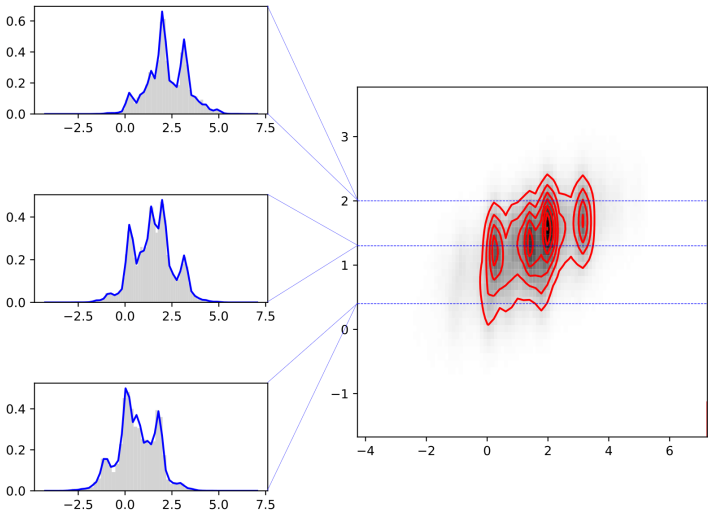
Come ultimo esempio, riporto quello di una distribuzione tridimensionale rappresentata da una RTBM con $N_v = 3$, $N_h = 1$ e parametri inizializzati nel modo seguente:

$$W = \begin{pmatrix} -15.76 \\ 2.29 \\ 2.09 \end{pmatrix}, \quad T = \begin{pmatrix} 16.02 & -6.52 & -6.76 \\ -6.52 & 29.04 & -2.56 \\ -6.76 & -2.56 & 42.16 \end{pmatrix}, \quad B_v = \begin{pmatrix} 1.08 \\ -0.67 \\ 4.86 \end{pmatrix},$$
$$B_h = \begin{pmatrix} 3.17 \end{pmatrix}, \quad Q = \begin{pmatrix} 19.18 \end{pmatrix}. \tag{2.14}$$

Come nel caso precedente è stato generato un sample di dati seguendo quanto descritto in [6], purtroppo per visualizzare la distribuzione avremmo bisogno di quattro dimensioni, ma ciononostante uno scatter plot del sampling si può vedere nella figura 2.4 a destra.
Le condizionate sono state ricavate sempre dalla (2.7), ma in questo caso si tratta di densità bidimensionali e non più monodimensionali come nei casi precedenti. Infatti chiamate (x, y, d) le coordinate di questa distribuzione, ho deciso di generare le condizionate $P(x, y|d)$, ovvero le proiezioni sul piano (x, y) rispetto ad un certo d fissato.
A sinistra in figura 2.4, sono rappresentati i contour plot delle condizionate sovrapposti ai relativi istogrammi bidimensionali estratti dalla distribuzione tridimensionale.



(a) *Distribuzioni condizionate a tre diversi valori di x fissato, generate per mezzo della (2.7).*



(b) *Distribuzioni condizionate a tre diversi valori di y fissato, generate per mezzo della (2.7).*

Figura 2.3: *Istogramma bidimensionale con 10^6 punti della distribuzione mostrata in figura 2.2, sovrapposto in rosso il contour plot della RTBM inizializzata con i parametri (2.13). Sono rappresentate in blu anche le distribuzioni condizionate a tre diversi valori rispettivamente della x e della y.*

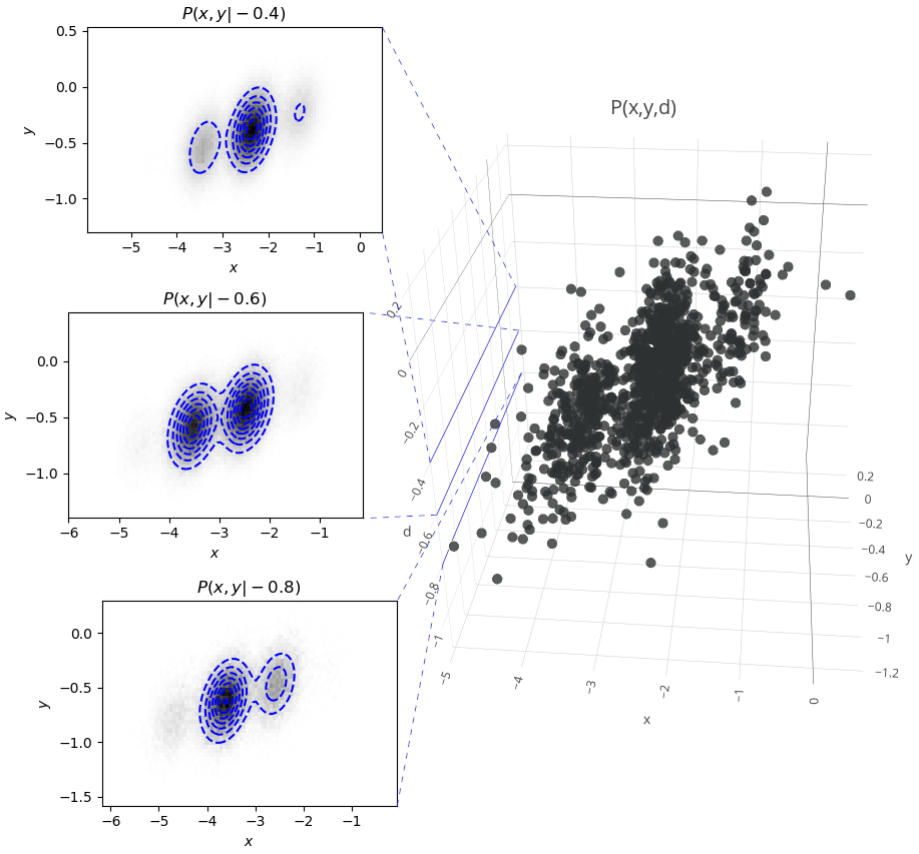


Figura 2.4: *Sulla destra, il sampling della distribuzione tridimensionale data dalla RTBM (2.14). Sulla sinistra invece, il confronto tra i contour plot delle probabilità condizionate* $P(x, y|d)$ *e i corrispondenti istogrammi bidimensionali, a tre diversi valori di* d*.*

2.2 Cumulative Distribution Function

2.2.1 Motivazione e Problema

Un'altra quantità interessante legata alle densità di probabilità è senza dubbio la Cumulative Distribution Function, in breve CDF.
Data una distribuzione con densità di probabilità $P(x)$, si definisce la sua CDF come:

$$F(y) = \int_{-\infty}^{y} dx\, P(x)\,, \tag{2.15}$$

da cui segue ovviamente che $F(\infty) = 1$. Questa funzione è molto utilizzata nell'ambito dell'analisi statistica, quando per esempio si vuole stabilire se un sample di dati segua una distribuzione nota, oppure se due diversi sample siano stati generati dalla stessa distribuzione, anche ignota.
Sono stati ideati infatti due test proprio per assolvere a questo compito: il test di Kolmogorov-Smirnov e quello di Kuiper. Quest'ultimo in particolare fa riferimento al caso in cui il dominio della distribuzione sia periodico e perciò viene utilizzato per studiare la distribuzione di eventi periodici, come la variazione del numero di tornadi in un anno o la variazione delle vendite di un determinato prodotto all'interno di un mese.
Nel nostro caso, analogamente a quanto fatto per la probabilità condizionata, ci chiediamo se una volta allenata una RTBM per modellizzare una distribuzione ignota sia possibile generarne la CDF in maniera semplice, senza dover ricorrere all'integrazione numerica che in generale può essere piuttosto costosa computazionalmente. La risposta ovviamente è sì e, come vedremo nella prossima sezione, ancora una volta otteniamo questo risultato grazie alla parentela delle RTBM con la multivariata.

2.2.2 Deduzione Matematica

Stiamo considerando il caso di una RTBM con $N_v = 1$ che modellizzi con la sua $P(v)$ una densità monodmensionale, perciò possiamo riscrivere la (2.15) come:

$$F(y) = \int_{-\infty}^{y} dv\, P(v)\,, \tag{2.16}$$

che sostituendo la definizione di $P(v)$ (1.12) ed esplicitando la $\tilde{\theta}$ (1.8) diventa:

$$F(y) = \frac{\sqrt{\frac{t}{2\pi}}\, e^{-\frac{B_v^2}{2t}}}{\tilde{\theta}(B_h^t - \frac{1}{t}B_v W | Q - \frac{1}{t}W^t W)} \sum_{n \in \mathbb{Z}^{N_h}} e^{-\frac{1}{2}n^t Q n + n^t B_h} \int_{-\infty}^{y} dv\, e^{-\frac{1}{2}tv^2 + (n^t W^t - B_v)v}\,. \tag{2.17}$$

Da notare che, dato che $N_v = 1$, sia la matrice T che il vettore colonna B_v sono semplicemente degli scalari, perciò li abbiamo chiamati per semplicità t e B_v.

Ora ci concentriamo sull'integrale. Non è difficile verificare che aggiungendo un termine $-\frac{1}{2t}(n^t W^t - B_v)^2$ all'esponenziale, possiamo completare il quadrato ed ottenere:

$$\sqrt{\frac{t}{2\pi}} \int_{-\infty}^{y} dv\, e^{-\frac{t}{2}[v - \frac{1}{t}(n^t W^t - B_v)]^2}\,,$$

che corrisponde esattamente alla CDF di una gaussiana con media $\mu = \frac{1}{t}(n^t W^t - B_v)$ e varianza $\sigma^2 = 1/t$. La soluzione è nota ed è data dalla funzione $\mathrm{erf}(x)$:

$$F(y) = \sqrt{\frac{1}{2\pi\sigma^2}} \int_{-\infty}^{y} dx\, e^{-\frac{1}{2\sigma^2}(x-\mu)^2} = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{y-\mu}{\sigma\sqrt{2}}\right)\right]. \tag{2.18}$$

Otteniamo quindi l'espressione esplicita della CDF per la nostra RTBM:

$$F(y) = \sum_{n \in \mathbb{Z}^{N_h}} \frac{e^{-\frac{1}{2}n^t(Q - \frac{1}{t}W^t W)n + n^t(B_h - \frac{1}{t}B_v W^t)}}{2\,\tilde{\theta}(B_h^t - \frac{1}{t}B_v W | Q - \frac{1}{t}W^t W)} \left[1 + \mathrm{erf}\left(\frac{ty - n^t W^t + B_v}{\sqrt{2t}}\right)\right], \tag{2.19}$$

dove abbiamo già riarrangiato il termine $\frac{1}{2t}(n^t W^t - B_v)^2$ necessario per bilanciare il completamento del quadrato.

Inoltre il primo dei due termini da luogo semplicemente alla stessa $\tilde{\theta}$ presente al denominatore e possiamo quindi semplificare l'espressione ulteriormente ottenendo:

$$F(y) = \frac{1}{2} + \frac{\sum_{n \in \mathbb{Z}^{N_h}} e^{-\frac{1}{2}n^t(Q - \frac{1}{t}W^t W)n + n^t(B_h - \frac{1}{t}B_v W^t)}\, \mathrm{erf}\left(\frac{ty - n^t W^t + B_v}{\sqrt{2t}}\right)}{2\,\tilde{\theta}(B_h^t - \frac{1}{t}B_v W | Q - \frac{1}{t}W^t W)}\,. \tag{2.20}$$

Abbiamo così ricavato l'espressione chiusa della CDF per la RTBM. Questa richiede unicamente il calcolo della funzione $\mathrm{erf}(x)$ e perciò porta enormi vantaggi rispetto al calcolo dell'integrale richiesto dalla definizione della CDF.

Ovviamente va fatto notare che la somma in linea di principio coinvolge infiniti termini, ma come vedremo nella possima sezione, fissata la precisione desiderata è sufficiente fermarsi ad un numero finito di termini.

2.2.3 Calcolo della CDF

Abbiamo appena trovato l'espressione della CDF per la RTBM, ma resta ancora da capire come dobbiamo comportarci con la somma su tutto $\mathbb{Z}^{N_h}$. In realtà nonostante non mi ci sia mai soffermato fino ad ora, anche il calcolo della $\tilde{\theta}$ coinvolge la stessa somma, e per far fronte a ciò abbiamo seguito quanto descritto in [9].
Infatti partendo dalla definizione della $\theta(z|\Omega)$ (1.8), si può rielaborare la somma in questione con l'introduzione di un reticolo Λ, formato da dei vettori $v(n) \in \mathbb{Z}^{N_h}$ con una particolare forma dipendente da z e Ω. Se a questo punto consideriamo l'ellisoide S_R definito da:

$$S_R = \left\{ v(n) \in \Lambda : \|v(n)\| < R \right\}, \tag{2.21}$$

è possibile riscrivere la somma:

$$\sum_{n \in \mathbb{Z}^{N_h}} \quad \Longrightarrow \quad \lim_{R \to \infty} \sum_{S_R} \quad ,$$

ma soprattutto quello che ci interessa è che si ricava:

$$\epsilon(R) = |\theta(z|\Omega) - \theta_R(z|\Omega)| \leq \sum_{\Lambda \setminus S_R} e^{-\|v(n)\|^2} . \tag{2.22}$$

Questo significa che l'errore ϵ che si commette fermando la somma all'ellissoide di raggio R, è limitato superiormente, o vedendola da un altro punto di vista, che fissata la precisione desiderata ϵ possiamo ricavare il raggio R a cui fermare la somma.

Lo stesso tipo di ragionamento ci viene in soccorso anche per la CDF, ovvero procedendo analogamente a quanto appena descritto, si trova il seguente limite superiore per l'errore di approssimazione della CDF:

$$\epsilon(R) = |F(y) - F_R(y)| \leq \frac{1}{2\,|\tilde{\theta}(B_h^t - \frac{1}{t} B_v W | Q - \frac{1}{t} W^t W)|} \sum_{\Lambda \setminus S_R} e^{-\|v(n)\|^2} , \tag{2.23}$$

dove S_R è sempre dato dalla (2.21), ma con una differente definizione di $v(n)$ e Λ. In particolare per la CDF abbiamo:

$$v(n) = M\left(n + \left[\left[\left(Q - \frac{1}{t}W^tW\right)^{-1}\left(\frac{1}{t}B_vW^t - B_h\right)\right]\right]\right), \tag{2.24}$$

dove ho definito $[[x]] = x - [x]$ e M è la matrice tale che $M^tM = Q - \frac{1}{t}W^tW$.

Quindi come per il calcolo della θ, dalla (2.23) abbiamo un modo per stabilire a che punto della somma fermarci per mantenere l'errore entro una certo limite. Negli esempi che mostrerò nella prossima sezione, il numero di punti necessari per mantenere un errore dell'ordine di 10^{-8} varia tra i 10 e i 30, appare perciò evidente l'enorme efficienza nel rappresentare le CDF con le RTBM in confronto all'integrazione numerica.

2.2.4 Esempi

Per dimostrare la validità dell'espressione per la CDF (2.20) appena trovata, riporto qualche esempio tratto da distribuzioni note. Prendo in considerazione distribuzioni con CDF analitica nota, in modo da poter misurare quantitativamente l'errore commesso nel modellizzare la CDF con la RTBM. In teoria avendo a disposizione anche solo l'espressione analitica della PDF si potrebbe ricavare la CDF dalla definizione (2.15), ma in questo modo nel confronto con la RTBM contribuirebbe anche l'errore legato all'integrazione.

Cauchy

Perciò iniziamo con la distribuzione di Cauchy, con PDF definita da:

$$P(x) = \frac{\gamma/\pi}{(x - x_0)^2 + \gamma^2} \tag{2.25}$$

e CDF:

$$F(x) = \frac{1}{2} + \frac{1}{\pi}\arctan\frac{x_0 - x}{\gamma}\,. \tag{2.26}$$

Nell'esempio considerato ho preso $x_0 = 0$ e $\gamma = 1$. Per modellizzare tale distribuzione ho usato una RTBM con $N_v = 1$ ed $N_h = 2$, allenata con l'algoritmo genetico CMA-ES su un sample di 5000 dati. La soluzione trovata

è quella corrispondente alla scelta di parametri seguente:

$$W = \begin{pmatrix} 2.98 & -2.66 \end{pmatrix}, \quad T = \begin{pmatrix} 0.42 \end{pmatrix}, \quad B_v = \begin{pmatrix} -0.01 \end{pmatrix}, \\ B_h = \begin{pmatrix} -0.23 \\ 0.08 \end{pmatrix}, \quad Q = \begin{pmatrix} 39.85 & -10.92 \\ -10.92 & 22.82 \end{pmatrix}, \tag{2.27}$$

il plot della macchina corrispondente a questa scelta e la soluzione analitica sono mostrati in figura 2.5 a sinistra.
In seguito ho generato la CDF usando la (2.20) con 27 punti nella somma e l'ho confrontata con quella analitica (2.26), come si può apprezzare a destra in figura 2.5. Inoltre per quantificare l'errore commesso ho calcolato l'MSE tra le due, che è riportato nella tabella 2.2.

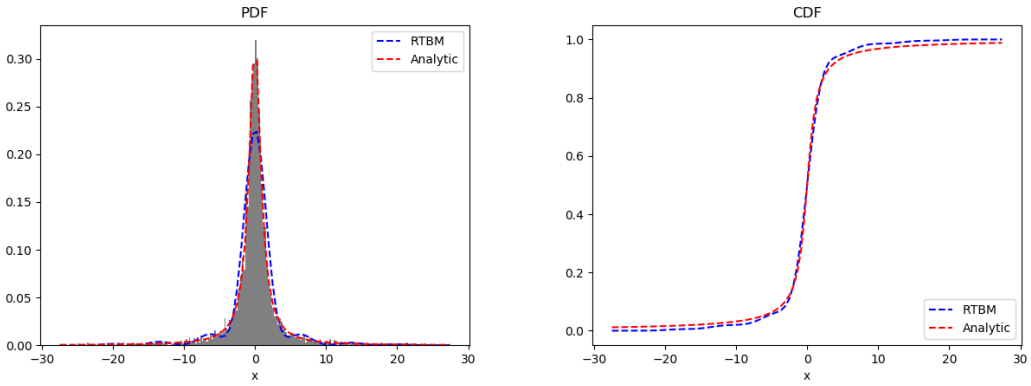


Figura 2.5: *Sulla sinistra una rappresentazione della distribuzione di Cauchy (2.25) con parametri $x_0 = 0$ e $\gamma = 1$, sono mostrate anche la curva analitica in rosso e la RTBM allenata (2.27). Sulla destra invece sono mostrate le relative CDF, in rosso quella analitica (2.26) ed in blu quella ricavata usando la (2.20) con 27 punti nella somma. L'MSE realtivo a questo esempio è visibile in tabella 2.2.*

Gamma

Come secondo esempio consideriamo la Gamma con PDF data da:

$$P(x) = \frac{1}{\Gamma(k)\,\theta^k}\, x^{k-1}\, e^{-\frac{x}{\theta}} \tag{2.28}$$

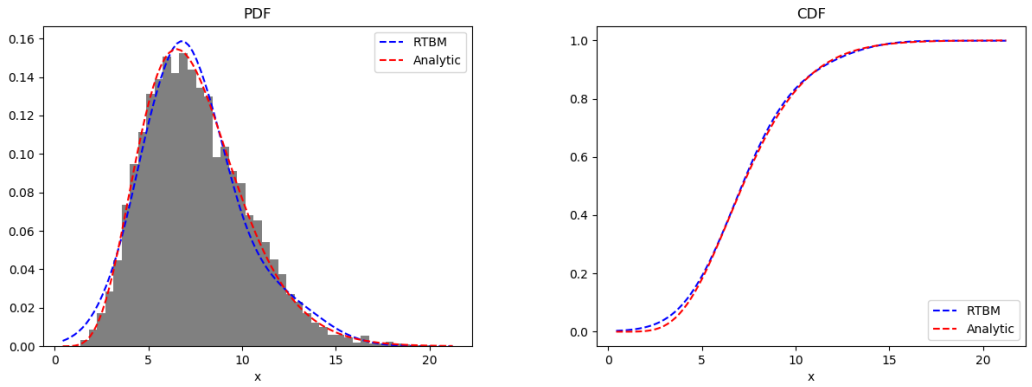


Figura 2.6: *Sulla sinistra una rappresentazione della distribuzione Gamma (2.28) con parametri* $k = 7.5$ *e* $\theta = 1$*, sono mostrate anche la curva analitica in rosso e la RTBM allenata (2.32). Sulla destra invece sono mostrate le relative CDF, in rosso quella analitica (2.29) ed in blu quella ricavata usando la (2.20) con 21 punti nella somma. L'MSE relativo a questo esempio è visibile in tabella 2.2.*

e CDF:

$$F(x) = \frac{1}{\Gamma(k)}\,\gamma\left(k, \frac{x}{\theta}\right), \tag{2.29}$$

dove $\Gamma(z)$ è la funzione gamma di Eulero:

$$\Gamma(z) = \int_0^\infty dt\, t^{z-1}\, e^{-t}, \tag{2.30}$$

mentre $\gamma(s, x)$ viene chiamata *Lower Incomplete Gamma Function* ed è definita da:

$$\gamma(s, x) = \int_0^x dt\, t^{s-1}\, e^{-t}. \tag{2.31}$$

Per l'esempio in questione ho preso una Gamma con parametri $k = 7.5$ e $\theta = 1$, e come nell'esempio precedente per modellizzarla ho allenato una RTBM $N_v = 1$, $N_h = 2$ con il CMA-ES su un sample di 5000 dati. La migliore soluzione è quella corrispondente a questa scelta dei parametri:

$$\begin{aligned} W &= \begin{pmatrix} 1.40 & -2.8 \end{pmatrix}, \quad T = \begin{pmatrix} 0.82 \end{pmatrix}, \quad B_v = \begin{pmatrix} 0.01 \end{pmatrix}, \\ B_h &= \begin{pmatrix} 1.88 \\ -1.24 \end{pmatrix}, \quad Q = \begin{pmatrix} 18.76 & 19.52 \\ 19.52 & 23.44 \end{pmatrix}. \end{aligned} \tag{2.32}$$

I plot della distribuzione originale e della macchina allenata si possono vedere sulla sinistra in figura 2.6 . Sulla destra sono riportate invece le CDF, analitica (2.29) e generata con la (2.20), usando 21 punti nella somma.
L'MSE calcolato per questo esempio si può sempre trovare in tabella 2.2.

Triangolare

Infine prendiamo la distribuzione triangolare, che date le ascisse dei tre vertici del triangolo a, c e b, ha PDF definita da:

$$P(x) = \begin{cases} \frac{2(x-a)}{(b-a)(c-a)} & a \leq x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)} & c < x \leq b \\ 0 & \text{altrove} \end{cases} , \tag{2.33}$$

in questo caso l'integrale per ricavare la CDF è banale e si trova:

$$F(x) = \begin{cases} 0 & x < a \\ \frac{(x-a)^2}{(b-a)(c-a)} & a \leq x \leq c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & c < x \leq b \\ 1 & x > b \end{cases} . \tag{2.34}$$

In figura 2.7 a sinistra, si può osservare la distribuzione corrispondente alla scelta $a = -3$, $b = 8$ e $c = 0$. Come nei casi precedenti una RTBM $N_v = 1$, $N_h = 2$ è stata allenata con il CMA-ES su un sample di 5000 dati, fino a trovare la migliore soluzione coincidente con la seguenta scelta dei parametri:

$$\begin{aligned} &W = \begin{pmatrix} 1.77 & -1.64 \end{pmatrix} , \quad T = \begin{pmatrix} 0.68 \end{pmatrix} , \quad B_v = \begin{pmatrix} 0.01 \end{pmatrix} , \\ &B_h = \begin{pmatrix} 7.48 \\ -4.31 \end{pmatrix} , \quad Q = \begin{pmatrix} 20.80 & -0.49 \\ -0.49 & 21.00 \end{pmatrix} . \end{aligned} \tag{2.35}$$

Come al solito in figura 2.7 sulla destra è mostrato il confronto tra la CDF analitica (2.34) e quella ricavata con (2.20), usando 9 punti nella somma. Per un confronto più oggettivo si può sempre trovare l'MSE in tabella 2.2.

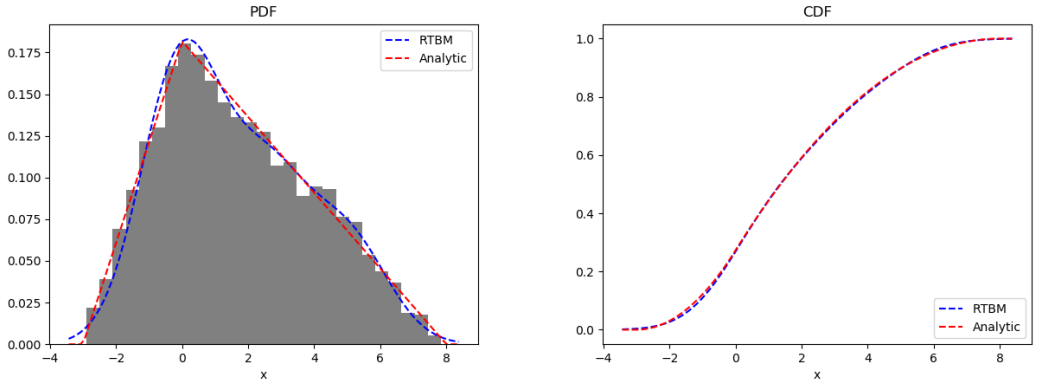


Figura 2.7: *Sulla sinistra una rappresentazione della distribuzione triangolare (2.33) con parametri* $a = -3$, $c = 0$ *e* $b = 8$, *sono mostrate anche la curva analitica in rosso e la RTBM allenata (2.35). Sulla destra invece sono mostrate le relative CDF, in rosso quella analitica (2.34) ed in blu quella ricavata usando la (2.20) con 9 punti nella somma. L'MSE realtivo a questo esempio è visibile in tabella 2.2.*

	Cauchy	Gamma	Triangolare
Punti	27	21	9
MSE	$2.291 \cdot 10^{-4}$	$6.047 \cdot 10^{-5}$	$2.242 \cdot 10^{-5}$

Tabella 2.2: *MSE calcolato tra la CDF analitica e quella ricavata usando la (2.20) per i tre esempi mostrati in questa sezione. È riportato inoltre il numero di punti usati nella somma in (2.20) per ciascun esempio.*

Capitolo 3

Allenamento e Ottimizzazione su Manifold

In questo ultimo capitolo parlerò invece di quello che ha impegnato la seconda parte del mio lavoro di tesi, ovvero la ricerca di un nuovo metodo per allenare le RTBM.
Come avevo anticipato nel capitolo 1, la RTBM presenta una complicazione non indifferente dal punto di vista dell'allenamento. Infatti abbiamo visto che i parametri di un modello sono raccolti all'interno di una matrice definita positiva, ciò significa che non tutte le scelte dei parametri sono accettabili, potremo accettare solo quelle combinazioni che danno luogo a matrici positive.
Questo è in contrasto con quanto si trova normalmente per le reti neurali feed forward. In genere infatti, non si ha nessun particolare vincolo e si possono scegliere i parametri in tutto $\mathbb{R}^n$, in altre parole si tratta di ottimizzazione libera. Tutti i principali e più usati algoritmi per l'allenamento di reti neurali, come Adam, RMSprop, AdaDelta, SGD, ecc..., sono pensati per uno spazio di ricerca coincidente con $\mathbb{R}^n$ e non possono essere quindi applicati alle RTBM in maniera diretta.

A questo punto ci restano solo due alternative. La prima è quella di usare semplicemente gli algoritmi sopracitati, aggiungendo però una routine di controllo che rifiuti le mosse non accettabili e faccia ripetere lo step in tal caso. È chiaro però che questa strategia è piuttosto costosa computazio-

nalmente, soprattutto ricordando che già di per se il calcolo della funzione theta e delle sue derivate ci costa risorse. La seconda possibilità è invece quella un pò più intelligente, si potrebbe pensare infatti di generalizzare gli algoritmi a spazi curvi e non semplicemente piatti come $\mathbb{R}^n$. In questo modo le mosse sarebbero vincolate allo spazio considerato e non rischieremmo più di uscirne.
Nel caso delle RTBM per esempio dovremmo considerare uno spazio fatto in questo modo:

$$\mathbb{R}^{N_h+N_v} \times \mathcal{P}(N_h + N_v)\,, \tag{3.1}$$

ovvero il prodotto cartesiano tra lo spazio piatto, dove vivono i bias, e la manifold delle matrici definite positive, che è quella che interessa la matrice dei parametri A.

Come si può intuire la strada che ho deciso di percorrere è la seconda, ora dobbiamo quindi capire come generalizzare un qualsiasi algoritmo allo spazio (3.1). Prima dobbiamo però introdurre un pò di nozioni sulle manifold che ci serviranno per la generalizzazione.

3.1 Manifold: un po' di Teoria

In questa sezione introdurrò brevemente la definizione di manifold, riportando principalmente le nozioni che mi servirannò poi per la generalizzazione, per una trattazione molto più completa rimando a [10].

Sia $\mathcal{M}$ un insieme, si definisce *carta* la biezione ϕ tra un sottoinsieme $\mathcal{U}$ di $\mathcal{M}$ e un insieme aperto di $\mathbb{R}^d$. Per completezza in genere la carta viene individuata proprio dal binomio $(\mathcal{U}, \phi)$. Inoltre preso $x \in \mathcal{U}$, gli elementi $\phi(x) \in \mathbb{R}^d$ sono dette coordinate di x nella carta $(\mathcal{U}, \phi)$.
Le carte sono utili perché ci permettono di passare dalla rappresentazione sulla manifold ad una rappresentazione in $\mathbb{R}^d$, per esempio se abbiamo una funzione definita sulla manifold $f : \mathcal{U} \to \mathbb{R}$, possiamo considerare invece la funzione $f \circ \phi^{-1} : \mathbb{R}^d \to \mathbb{R}$ che è reale e possiamo quindi usare tutti gli strumenti di analisi reale.

A partire dalle carte possiamo introdurre un nuovo concetto, quello di *atlante*.

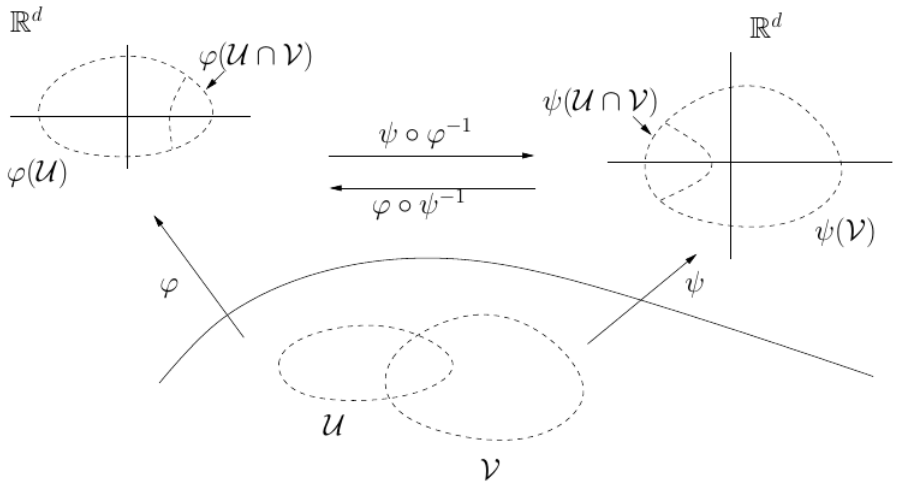


Figura 3.1: *Rappresentazione schematica di due carte. Immagine presa da [10].*

Definizione 3.1.1. Si definisce $\mathcal{A}$, atlante $\mathcal{C}^\infty$ di $\mathcal{M}$ su $\mathbb{R}^d$, l'insieme delle carte $(\mathcal{U}_\alpha, \phi_\alpha)$ di $\mathcal{M}$ tali che:

(*i*) $\bigcup_\alpha \mathcal{U}_\alpha = \mathcal{M}$

(*ii*) Per ogni coppia α, β con $\mathcal{U}_\alpha \cap \mathcal{U}_\beta \neq \emptyset$, gli insiemi $\phi_\alpha(\mathcal{U}_\alpha \cap \mathcal{U}_\beta)$ e $\phi_\beta(\mathcal{U}_\alpha \cap \mathcal{U}_\beta)$ sono aperti di $\mathbb{R}^d$ e il cambio di coordinate:

$$\phi_\beta \circ \phi_\alpha^{-1} : \mathbb{R}^d \to \mathbb{R}^d$$

è $\mathcal{C}^\infty$ su $\phi_\alpha(\mathcal{U}_\alpha \cap \mathcal{U}_\beta)$.

Finalmente possiamo quindi introdurre la definizione di manifold.

Definizione 3.1.2. Si definisce manifold la coppia $(\mathcal{M}, \mathcal{A})$, con $\mathcal{M}$ insieme e $\mathcal{A}$ atlante di $\mathcal{M}$ su $\mathbb{R}^d$, tale che la topologia indotta da $\mathcal{A}$ sia di Haussdorf e completamente separabile.

Inoltre data una carta ϕ su $\mathcal{M}$, si chiama ϕ^{-1} la parametrizzazione locale della manifold $\mathcal{M}$.

Abbiamo appena visto come rappresentare le funzioni reali sulla manifold, quello che manca è capire come sono fatte le derivate e per farlo avremo

bisogno di introdurre il concetto di spazio tangente.
Cominciamo considerando una curva sulla manifold $\gamma : \mathbb{R} \to \mathcal{M}$, verrebbe naturale definire la derivata lungo la curva come:

$$\gamma'(t) := \lim_{\tau \to 0} \frac{\gamma(t+\tau) - \gamma(t)}{\tau}\,, \tag{3.2}$$

il problema però è che la differenza $\gamma(t+\tau) - \gamma(t)$ necessita della struttura di spazio vettoriale, che non abbiamo, e perciò non ha nessun significato. Se consideriamo però la funzione reale f su $\mathcal{M}$, abbiamo che $f \circ \gamma : t \to f(\gamma(t))$ è una funzione da $\mathbb{R}$ in $\mathbb{R}$ con derivate ben definite.
Perciò preso $x \in \mathcal{M}$ e γ curva in $\mathcal{M}$ tale che $\gamma(0) = x$, possiamo definire il vettore tangente alla curva γ in $t = 0$ come:

$$\dot{\gamma}(0)\, f = \frac{d(f(\gamma(t)))}{dt}\bigg|_{t=0}, \qquad f \in \mathscr{F}_x(\mathcal{M})\,, \tag{3.3}$$

dove $\mathscr{F}_x(\mathcal{M})$ è l'insieme delle funzioni reali liscie definite in un intorno di x. Formalmente abbiamo la seguente definizione di vettore tangente.

Definizione 3.1.3. Un vettore tangente ξ_x alla manifold $\mathcal{M}$ nel punto x è una mappa $\xi_x : \mathscr{F}_x(\mathcal{M}) \to \mathbb{R}$, tale che esista una curva γ in $\mathcal{M}$ con $\gamma(0) = x$, che soddisfi:

$$\xi_x\, f = \dot{\gamma}(0)\, f = \frac{d(f(\gamma(t)))}{dt}\bigg|_{t=0}$$

per ogni $f \in \mathscr{F}_x(\mathcal{M})$.

Con questa definizione possiamo introdurre il concetto di spazio tangente.

Definizione 3.1.4. Si chiama spazio tangente ad $\mathcal{M}$ in x, in breve $T_x(\mathcal{M})$, l'insieme di tutti i vettori ξ_x tangenti ad $\mathcal{M}$ in x.

È importante notare che questo insieme ha la struttura di spazio vettoriale, ovvero dati due vettori tangenti $\dot{\gamma}_1(0)$ e $\dot{\gamma}_2(0)$ si ha che:

$$(a\dot{\gamma}_1(0) + b\dot{\gamma}_2(0))f = a\dot{\gamma}_1(0)f + b\dot{\gamma}_1(0)f\,, \qquad \forall a, b \in \mathbb{R}\,.$$

Questo fatto è di fondamentale importanza perché lo spazio tangente $T_x\mathcal{M}$ approssima la manifold in un intorno di x. Ciò significa che il problema di ottimizzazione sulla manifold può essere trattato localmente come un

problema di ottimizzazione canonica in $T_x\mathcal{M}$, è proprio questo che sta alla base della generalizzazione che vogliamo fare. Rimane però da capire come tornare sulla manifold una volta trovata la soluzione nello spazio tangente.

Un'altra questione di cui non abbiamo ancora parlato, è il come misurare le distanze sulla manifold. Per fare ciò cominciamo col fornire ogni spazio tangente di un prodotto interno:

$$\|\xi_x\|_x = \sqrt{g_x(\xi_x, \xi_x)}\,, \tag{3.4}$$

dove il pedice x ci ricorda che tale prodotto dipende dallo spazio tangente che stiamo considerando. Se tale prodotto interno è liscio, la manifold prende il nome di manifold Riemanniana, e viene indicata dalla coppia $(\mathcal{M}, g)$, mentre g viene detta metrica Riemanniana.
Con l'introduzione di questa metrica possiamo andare a definire la lunghezza di una curva $\gamma : [a, b] \to \mathcal{M}$ sulla manifold:

$$L(\gamma) = \int_a^b dt\, \sqrt{g(\dot\gamma(t), \dot\gamma(t))}\,, \tag{3.5}$$

e ciò induce naturalmente la seguente definizione di distanza sulla manifold:

$$d(x, y) = \inf_{\gamma\in\Gamma} L(\gamma) \tag{3.6}$$

con Γ insieme delle curve che congiungono x e y. Io non lo mostro, ma si può verificare che d ha tutte le proprietà di una distanza.

Abbiamo già visto come sono fatte le derivate direzionali sulla manifold, ovvero quelli che abbiamo chiamato vettori tangenti, resta da vedere però se si riesca a dare una definizione di gradiente analogamente a quanto si fa nello spazio piatto. Questo passaggio è molto importante, perchè la quasi totalità degli algoritmi usati per l'ottimizzazione fa uso del gradiente per trovare la direzione di massima discesa della funzione costo, e ci chiediamo quindi se anche nel nostro caso potremo sfruttarlo.
La risposta è sí, preso infatti un campo scalare $f : \mathcal{M} \to \mathbb{R}$, si definisce il gradiente Riemanniano di f in x come quell'unico elemento di $T_x\mathcal{M}$ per cui valga:

$$g_x(\text{grad}\, f(x), \xi_x) = D\, f(x)[\xi_x] \qquad \forall \xi_x \in T_x\mathcal{M} \tag{3.7}$$

dove $D\,f(x)[\xi_x]$ denota proprio la derivata di f lungo la direzione ξ_x nel punto x. In particolare quindi grad $f(x)$ punta nella direzione di massima salita di f sulla manifold, come volevamo.

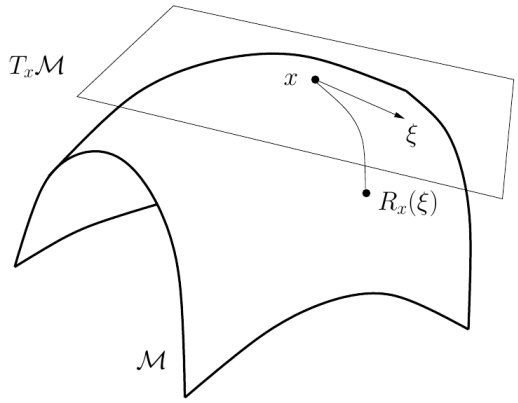


Figura 3.2: *Retrazione sulla manifold $\mathcal{M}$. Immagine presa da [10].*

Ora abbiamo praticamente tutto quello che ci serve, ci resta ancora da capire come passare dallo spazio tangente alla manifold, e volendo il viceversa. Lo strumento che ci permette di fare ciò è una particolare mappa $R_x : T_x(\mathcal{M}) \to \mathcal{M}$ che viene chiamata retrazione, di seguito la definizione formale.

Definizione 3.1.5. Una retrazione su una manifold $\mathcal{M}$ è una mappa liscia $R_x : T_x(\mathcal{M}) \to \mathcal{M}$ con le proprietà:

(*i*) $R_x(0_x) = x$ dove 0_x è l'elemento 0 di $T_x\mathcal{M}$.

(*ii*) Identificando $T_{0_x}T_x\mathcal{M} \simeq T_x\mathcal{M}$, R_x soddisfa:

$$D\,R_x(0_x) = \mathrm{Id}_{T_x\mathcal{M}}$$

con $\mathrm{Id}_{T_x\mathcal{M}}$ mappa identica su $T_x\mathcal{M}$.

In particolare tutte le manifold che ammettono metrica Riemanniana, ammettono anche una retrazione che è identificata dalla mappa esponenziale Riemanniana. Purtroppo però il calcolo di tale mappa è tipicamente computazionalemente costoso e si ricorre quindi ad approssimazioni, prestando attenzione ovviamente a non inficiare le proprietà di convergenza dell'algoritmo che si sta considerando.

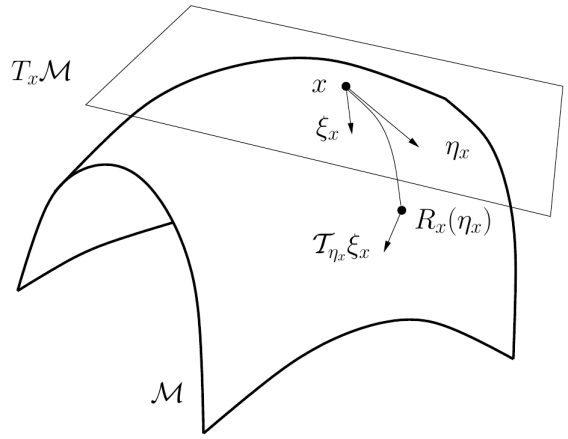


Figura 3.3: *Trasporto vettoriale sulla manifold $\mathcal{M}$. Immagine presa da [10].*

Infine è utile domandarsi come confrontare due vettori $v_x \in T_x\mathcal{M}$ e $v_y \in T_y\mathcal{M}$. Come vedremo nelle prossime sezioni questo capita molto spesso negli algoritmi iterativi, in cui ad ogni step vengono aggiornate alcune quantità sulla base del loro valore allo step precedente. Nel caso delle manifold ad ogni step ci troviamo su un diverso spazio tangente e perciò qualsiasi operazione tra due quantità appartenenti a spazi tangenti differenti non avrebbe alcun senso, per questo ora introduciamo il concetto di trasporto parallelo.
Sia $T\mathcal{M} = \cup_{x\in\mathcal{M}} T_x\mathcal{M}$ e $T\mathcal{M} \oplus T\mathcal{M}$ l'insieme:

$$T\mathcal{M} \oplus T\mathcal{M} = \{(\eta_x, \xi_x) : \eta_x, \xi_x \in T_x\mathcal{M}, x \in \mathcal{M}\},$$

allora abbiamo la seguente definizione di trasporto vettoriale.

Definizione 3.1.6. Il trasporto parallelo $\mathcal{T}$ su una manifold $\mathcal{M}$ è una mappa liscia:

$$\mathcal{T} : T\mathcal{M} \oplus T\mathcal{M} \to T\mathcal{M} : (\eta_x, \xi_x) \to \mathcal{T}_{\eta_x}(\xi_x),$$

che soddisfa le seguenti proprietà $\forall x \in \mathcal{M}$:

(*i*) $\mathcal{T}_{\eta_x}(\xi_x)$ è un vettore tangente in $T_{R_x(\xi_x)}\mathcal{M}$, con R retrazione associata a $\mathcal{T}$.

(*ii*) $\mathcal{T}_{0_x}(\xi_x) = \xi_x$ per ogni $\xi_x \in T_x\mathcal{M}$.

(*iii*) $\mathcal{T}_{\eta_x}(a\xi_x + b\gamma_x) = a\mathcal{T}_{\eta_x}(\xi_x) + b\mathcal{T}_{\eta_x}(\gamma_x)$.

Finalmente ora abbiamo tutto ciò che ci serve e possiamo concentrarci sul problema di trovare il minimo di una funzione definita su una manifold.

3.2 Ottimizzazione su Manifold

Come viene descritto in [10], [11], [12], [13], in realtà esistono già degli algoritmi per l'ottimizzazione su manifold, principalmente il *Linesearch* con *Armijo Backtracking* e il metodo di Newton. Il primo è anche già implementato in python all'interno della libreria *Pymanopt* [14] ed è proprio la prima cosa che ho provato. L'algoritmo di Newton, essendo di secondo ordine, necessita del calcolo dell'Hessiana ad ogni step e richiederebbe perciò troppe risorse, per questo non l'ho considerato.
La libreria permette di allocare diversi tipi di manifold per l'ottimizzazione, per la soluzione poi sono implementati alcuni algoritmi tra cui quello che viene chiamato *Steepestdescent*. Essenzialmente è un algoritmo di linesearch in cui la direzione di ricerca ad ogni step è il gradiente riemanniano cambiato di segno. In altre parole si tratta dell'equivalente del gradient descent però in un generico spazio curvo.
A partire da questo metodo ho scritto quindi un algoritmo di allenamento per la RTBM che riesce effettivamente ad abbassare il costo, ma si blocca sempre in un minimo locale, risultando in una rappresentazione banale della distribuzione. Il problema è che la strategia di seguire la direzione di massima discesa del costo è troppo banale, per allenare efficacemente la macchina abbiamo bisogno di algoritmi più sofisticati come i già nominati Adam o AMSGrad.

Perciò siamo finalmente arrivati al punto cruciale del discorso: come si generalizzano questi algoritmi a generici spazi curvi? Per spiegare l'idea di fondo consideriamo prima un algoritmo più semplice, una volta capito il meccanismo la generalizzazione degli altri verrà da se.
Prendiamo per esempio l'SGD, *Stochastic Gradient Descent*, con momento. Esso differisce poco dal normale gradient descent, infatti l'unica differenza è che per l'update dei parametri si fa uso anche del gradiente allo step precedente, oltre che a quello corrente, in questo modo si ha una sorta di momento residuo che aiuta a superare i minimi locali. Quindi chiamato p_t

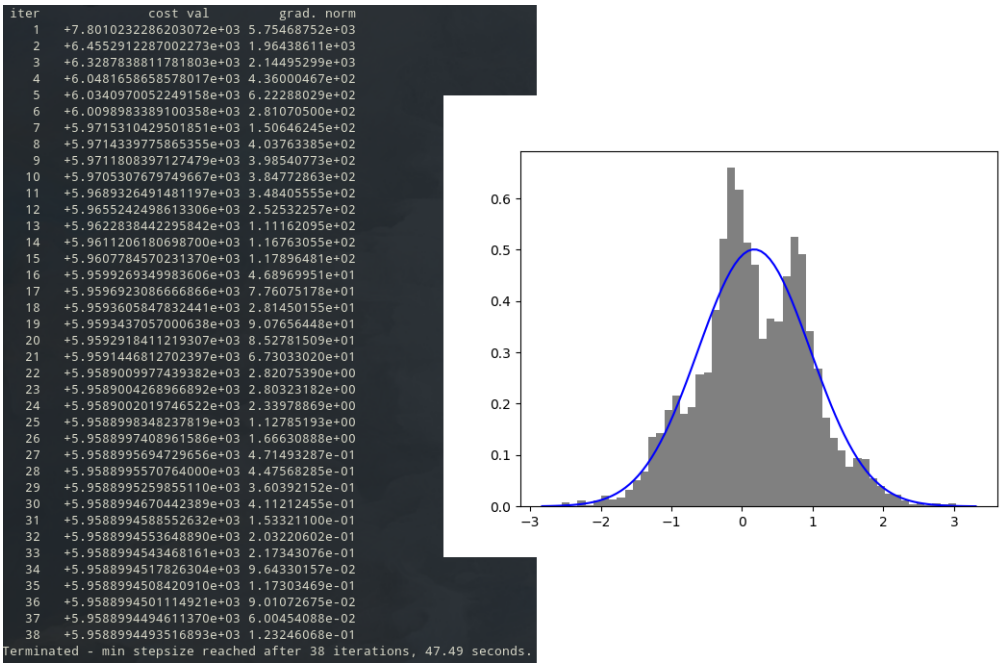


Figura 3.4: *Risultato dell'allenamento di una RTBM* $N_v = 1$, $N_h = 2$ *con l'algoritmo Steepestdescent di pymanopt [14]. A sinistra il valore della funzione costo, la loglikelyhood in questo caso, trovato per le diverse iterazioni. Come si vede l'algoritmo rimane bloccato in un minimo locale, trovando una soluzione evidentemente banale e che non rappresenta bene i dati.*

il vettore dei miei parametri allo step t, definiremo la nuova direzione di ricerca come:

$$v_t = \gamma\, v_{t-1} - \eta\, \nabla f(p_t)\,, \tag{3.8}$$

dove γ è il coefficiente di momento, tipicamente impostato a $\gamma = 0.9$, ed f è la funzione costo che vogliamo minimizzare. Una volta ottenuto v_t, l'update dei parametri viene fatto nel solito modo:

$$p_{t+1} = p_t - v_t\,. \tag{3.9}$$

Si ripete poi il tutto fino a convergenza.
Tutto questo vale però fintanto che siamo nello spazio piatto $\mathbb{R}^n$, vediamo ora il caso di una generica manifold Riemanniana.
Il vettore dei parametri p_t sarà ora un punto sulla nostra manifold $\mathcal{M}$, a cui

possiamo associare lo spazio tangente $T_{p_t}\mathcal{M}$. Perciò dobbiamo capire quale direzione, ovvero quale vettore in $T_{p_t}\mathcal{M}$, prendere per spostarci. Come ci si poteva aspettare la scelta che facciamo è quella di prendere il gradiente Riemanniano, rispetto allo spazio piatto operiamo quindi la sostituzione:

$$\nabla f(p_t) \quad \longrightarrow \quad \text{grad}\, f(p_t)\,.$$

Sorge però un problema, nella regola (3.8) compare una differenza tra due vettori, la direzione presa allo step precedente e il gradiente corrente, ma questi due vettori in generale non apparterranno allo stesso spazio tangente e di conseguenza tale differenza non ha alcun senso.
Questo significa che prima di andare a calcolare la nuova direzione di ricerca, dovremo trasportare parallelamente il vettore v_{t-1} dallo spazio tangente $T_{p_{t-1}}\mathcal{M}$ a quello $T_{p_t}\mathcal{M}$. Una volta fatto, la differenza in (3.8) sarà ben definita e potremo quindi calcolare la nuova direzione di ricerca come:

$$v_t = \gamma\, \mathcal{T}(v_{t-1}) - \eta\ \text{grad}\, f(p_t)\,, \tag{3.10}$$

dove $\mathcal{T}$ è proprio il trasporto parallelo dallo spazio $T_{p_{t-1}}\mathcal{M}$ a quello $T_{p_t}\mathcal{M}$. Abbiamo trovato il modo per definire v_t, bisogna però ricordare che chiaramente v_t sarà ancora nello spazio tangente, per ritornare su $\mathcal{M}$ perciò ci affidiamo alla retrazione. Finalmente possiamo scrivere quindi la regola di update dei nostri parametri come:

$$p_{t+1} = R_{p_t}(v_t) = \exp_{p_t}(v_t)\,, \tag{3.11}$$

con v_t dato dalla (3.10) ed $\exp_{p_t}$ mappa esponenziale.

3.2.1 AMSGrad

Ora che abbiamo capito l'idea di fondo possiamo trattare algoritmi più articolati, gli step da seguire saranno sempre i medesimi, quello che cambierà invece saranno le regole di update della direzione e dei parametri.
Partiamo dall'AMSGrad, algoritmo introdotto in [15] che rappresenta essenzialmente un evoluzione dell'Adam. In maniera simile a quanto viene fatto per il SGD con momento, si tiene conto delle direzioni prese agli step precedenti definendo la successione:

$$m_t = \beta_1\, m_{t-1} + (1-\beta_1)\, \nabla f(p_t)\,, \tag{3.12}$$

che andrà rappresentare la media smorzata, con coefficiente di decadimento β_1, dei gradienti calcolati lungo tutto il percorso. In questo caso però si tiene traccia anche del quadrato del gradiente, si definisce infatti un'altra successione:

$$v_t = \beta_2 \, v_{t-1} + (1 - \beta_2) \, (\nabla f(p_t))^2 \, , \tag{3.13}$$

che fornirà una stima del secondo momento. Prima di passare all'update dei parametri c'è però un ulteriore step, quello che fondamentalmente differenzia l'AMSGrad dall'Adam, ovvero viene confrontato il precedente v con quello calcolato allo step attuale e se ne prende poi il massimo, quindi:

$$v_t = \max(v_{t-1}, v_t) \, . \tag{3.14}$$

Infine bisogna procedere con l'update dei parametri, dato dalla seguente regola:

$$p_{t+1} = p_t - \frac{\alpha}{\sqrt{v_t} + \epsilon} \, m_t \, , \tag{3.15}$$

da notare che l'ϵ al denominatore viene aggiunto semplicemente per evitare che nell'implementazione sul calcolatore possano insorgere problemi numerici legati alla divisione per zero, un valore tipico è $\epsilon = 10^{-8}$.
Faccio notare inoltre che le quatità m, v e p sono vettori, perciò tutte le operazioni che sono riportate in (3.12), (3.13), (3.14) e (3.15) sono da intendersi elemento per elemento, questo significa inoltre che l'aggiornamento di ogni parametro avverà con rate diverso.

Ora usando quanto abbiamo discusso nella sezione precedente possiamo costruire l'AMSGrad sulla manifold. Per riscrivere le regole (3.12) e (3.13) dovremo sostiuire i gradienti euclidei con quelli Riemanniani e ricordarci di trasportare parallelamente tutto quello che ci serve nello spazio tangente al nuovo punto della manifold. Le nuove regole che otteniamo saranno quindi:

$$m_t = \beta_1 \, \mathcal{T}(m_{t-1}) + (1 - \beta_1) \, \operatorname{grad} f(p_t) \, , \tag{3.16}$$

$$v_t = \beta_2 \, \mathcal{T}(v_{t-1}) + (1 - \beta_2) \, (\operatorname{grad} f(p_t))^2 \, . \tag{3.17}$$

Come nel caso precedente sceglieremo poi il massimo tra v_t e v_{t-1}:

$$v_t = \max(v_{t-1}, v_t) \, , \tag{3.18}$$

AMSGrad Euclideo	AMSGrad Riemanniano
• Inizializzo $t = 0$, $m_0 = 0$, $v_0 = 0$ • Scelgo punto di partenza $p_1 \in \mathbb{R}^n$ • **for** $t = 1$ **to** T **do**: • Calcolo $\nabla f(p_t)$ • $m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla f(p_t)$ • $v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla f(p_t))^2$ • $v_t = \max(v_{t-1}, v_t)$ • $p_{t+1} = p_t - \frac{\eta}{\sqrt{v_t+\epsilon}} m_t$ • **end for**	• Inizializzo $t = 0$, $m_0 = 0$, $v_0 = 0$ • Scelgo punto di partenza $p_1 \in \mathcal{M}$ • **for** $t = 1$ **to** T **do**: • Calcolo grad $f(p_t)$ • $m_t = \beta_1 m_{t-1} + (1-\beta_1)$ grad $f(p_t)$ • $v_t = \beta_2 v_{t-1} + (1-\beta_2)(\text{grad } f(p_t))^2$ • $v_t = \max(v_{t-1}, v_t)$ • $p_{t+1} = R_{p_t}\left(-\frac{\eta}{\sqrt{v_t+\epsilon}} m_t\right)$ • Trasporto $m_t = \mathcal{T}(m_t)$, $v_t = \mathcal{T}(v_t)$ • **end for**

Tabella 3.1: *Schema riassuntivo dell'AMSGrad nello spazio piatto e sulla manifold.*

e infine l'aggiornamento dei parametri corrisponderà allo spostamento ad un nuovo punto sulla manifold per mezzo della retrazione:

$$p_{t+1} = R_{p_t}\left(-\frac{\alpha}{\sqrt{v_t}+\epsilon} m_t\right). \tag{3.19}$$

Prima di mostrare i risultati ottenuti con questo algoritmo ne introduciamo un secondo per poterli poi confrontare.

3.2.2 CMA-ES

Il CMA-ES [16] è un algoritmo di carattere completamente diverso rispetto all'AMSGrad che abbiamo appena visto, si tratta infatti di un algoritmo genetico e non fa uso dei gradienti per la minimizzazione. In realtà è possibile inserirli per migliorare le performance in alcuni casi, ma noi non ce ne occuperemo.

L'idea di questo algoritmo è quella di trattare i vettori dei parametri come individui di una popolazione. A ciascuno di essi è anche associato un grado di *fitness* che indica quanto esso rappresenti una buona soluzione al problema di ottimizzazione. In pratica nel nostro caso la *fitness* sarà proprio la funzione costo, perciò gli individui con costo minore saranno quelli con

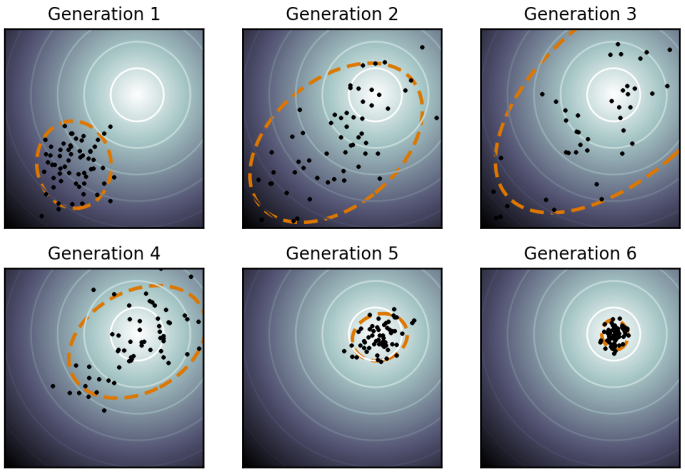


Figura 3.5: *Rappresentazione schemetica del funzionamento del CMA-ES.*

fitness più alta.
L'algoritmo poi può essere diviso in tre fasi, la prima è la fase di generazione della popolazione. Se come prima chiamiamo p_t il valore dei parametri allo step t, che in questo caso rappresenterà l'individuo genitore allo step t, possiamo andare a generare la nuova popolazione in questo modo:

$$p_i^{(t)} = p_t + \sigma_t \mathcal{N}(0, C_t)\,, \tag{3.20}$$

dove σ_t è il $t-$esimo *stepsize* mentre C_t è la matrice di covarianza dei parametri sempre allo step t. In pratica stiamo generando un sample gaussiano nei dintorni del genitore p_t.

Ci ritroviamo quindi con una nuova popolazione $\{p_i^{(t)}\}_{i=1,\dots,n}$ in cui ogni individuo avrà una diversa *fitness*, qualcuno più bassa del genitore qualcun'altro più alta. Possiamo quindi passare alla seconda fase che viene detta ricombinazione.
Come prima cosa ordiniamo gli individui appena generati con *fitness* descrescente, ovvero partendo da quello con costo più basso fino ad arrivare a quello più alto. Una volta ordinati, selezionamo i primi $m < n$ che saranno i migliori tra quelli generati, e li facciamo partecipare alla ricombinazione:

$$p_{t+1} = \sum_{i=1}^{m} \omega_i \, p_i^{(t)} \tag{3.21}$$

dove ω_i è una successione di pesi decrescenti, in questo modo gli individui con *fitness* più alta avranno peso maggiore.
Abbiamo così generato l'individuo genitore dello step $t+1$, resta da capire però come aggiornare la matrice di covarianza C e lo stepsize σ. Questo fa parte della terza fase, che essendo la più tecnica non tratterò in dettaglio, per una descrizione approfondita rimando a [16]. In pratica l'aggiornamento di C e σ è controllato rispettivamente da due vettori v_c e v_σ, che vengono detti *evolution paths*. Sono inizializzati a zero al primo step e successivamente seguono le seguenti regole di aggiornamento:

$$v_c^{t+1} = (1-c_c)v_c^t + h_\sigma \sqrt{c_c(2-c_c)\mu}\, p_{t+1}\,, \tag{3.22}$$

$$v_\sigma^{t+1} = (1-c_\sigma)v_\sigma^t + \sqrt{c_\sigma(2-c_\sigma)\mu}\, C_t^{-\frac{1}{2}} p_{t+1}\,, \tag{3.23}$$

a partire da questi si ricavano poi le regole di aggiornamento per C e σ:

$$C_{t+1} = \left(1 + c_1\delta(h_\sigma) - c_1 - c_\mu \sum_i \omega_i\right) C_t + c_1 v_c^{t+1}(v_c^{t+1})^t + c_\mu \sum_{i=1}^{m} \omega_i^o p_i^{(t)} (p_i^{(t)})^t\,, \tag{3.24}$$

$$\sigma_{t+1} = \sigma_t \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|v_\sigma^{t+1}\|}{E\|\mathcal{N}(0,1)\|}\right) - 1\right). \tag{3.25}$$

Come si nota sono regole piuttosto complicate che comprendono anche molti coefficienti di cui non riporto il valore esplicito, le ho riportate per completezza ma preferisco non approfondire oltre per non appesantire il discorso.

Veniamo ora al caso della manifold, per il quale ho seguito principalmente [17]. Non essendo questo un algoritmo *gradient-based* il discorso cambia leggermente dal caso precedente, ma l'idea è sempre la solita, ovvero vogliamo lavorare nello spazio tangente e poi trasportare tutto sulla manifold con una retrazione.
Partiamo quindi dalla generazione di una nuova popolazione. Supponiamo come al solito di partire da un punto p_t sulla manifold, quello che faremo è generare un sample di vettori tangenti $u_i^{(t)} \in T_{p_t}\mathcal{M}$ distribuiti gaussianamente attorno a p_t:

$$u_i^{(t)} \sim \mathcal{N}(0, \sigma_t C_t)\,, \qquad i = 1, \dots, n\,. \tag{3.26}$$

Ora dovremmo ordinarli in base al loro costo, ma la nostra funzione costo sarà una $f : \mathcal{M} \to \mathbb{R}$ e perciò dobbiamo prima proiettarli su $\mathcal{M}$:

$$p_i^{(t)} = R_{p_t}(u_i^{(t)})\,, \tag{3.27}$$

e poi calcolarne il costo $f(p_i^{(t)})$.
A questo punto li possiamo ordinare e passare poi alla ricombinazione che avverà di nuovo nello spazio tangente:

$$\tilde{u}_t = \sum_{i=1}^{m} \omega_i u_i^{(t)} \,. \tag{3.28}$$

Il punto della manifold allo step successivo si ottiene poi come al solito:

$$p_{t+1} = R_{p_t}(\tilde{u}_t) \,. \tag{3.29}$$

L'ultima cosa che rimane da fare è aggiornare gli evolution paths, per poi calcolare le nuove C_{t+1} e σ_{t+1}. Analogamente alla (3.22) e (3.23) si trova:

$$v_c^{t+1} = (1 - c_c) v_c^t + \sqrt{c_c(2 - c_c)\mu}\, \frac{1}{\sigma_t}\, \tilde{u}_t \,, \tag{3.30}$$

$$v_\sigma^{t+1} = (1 - c_\sigma) v_\sigma^t + \sqrt{c_\sigma(2 - c_\sigma)\mu}\, \frac{1}{\sigma_t}\, C_t^{-\frac{1}{2}}\, \tilde{u}_t \,, \tag{3.31}$$

da notare che i vettori v_c^t, v_σ^t e $\tilde{u}_t$ appartengono tutti allo spazio tangente $T_{p_t}\mathcal{M}$, così come la matrice di covarianza C_t è quella relativa ai parametri p_t, perciò a loro volta v_c^{t+1} e v_σ^{t+1} saranno ancora in $T_{p_t}\mathcal{M}$. Questo significa che come ultimo step dobbiamo ricordarci di trasportarli parallelamente al nuovo spazio $T_{p_{t+1}}\mathcal{M}$.
Una volta ottenuti i nuovi evolution paths possiamo aggiornare anche matrice di covarianza e stepsize. Le regole sono essenzialmente le stesse del caso euclideo, le apparenti differenze sono semplicemente legate a differenti scelte di nomenclatura tra [16] e [17]. In ogni caso la versione riportata in [17] è la seguente:

$$C_{t+1} = c_{cov}\left(1 - \frac{1}{\mu_{cov}}\right) \frac{1}{\sigma_t^2} \sum_{i=1}^{m} \omega_i\, u_i^{(t)} (u_i^{(t)})^t + \left(1 - \frac{1}{\mu_{cov}}\right) C_t + \frac{c_{cov}}{\mu_{cov}} v_c^{t+1} (v_c^{t+1})^t \,, \tag{3.32}$$

$$\sigma_{t+1} = \sigma_t \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|v_\sigma^{t+1}\|_{p_t}}{E\|\mathcal{N}(0,1)\|}\right) - 1\right), \tag{3.33}$$

dove la norma $\|v_\sigma^{t+1}\|_{p_t}$ non sarà più quella euclidea in questo caso, ma piuttosto quella riemanniana $\|v_\sigma^{t+1}\|_{p_t} = \sqrt{g_{p_t}(v_\sigma^{t+1}, v_\sigma^{t+1})}$.
Infine, come avevo anticipato, dopo aver aggiornato tutto al nuovo step $t+1$ dobbiamo ricordarci di trasportare parallelamente v_c^{t+1}, v_σ^{t+1} e C_{t+1} al nuovo spazio $T_{p_{t+1}}\mathcal{M}$.

CMA-ES Euclideo	CMA-ES Riemanniano

• Inizializzo $t = 0$, $v_c^0 = 0$, $v_\sigma^0 = 0$, $C_1 = \mathbb{I}$
• Scelgo punto di partenza $p_1 \in \mathbb{R}^n$
• **for** $t = 1$ **to** T **do**:
  • Nuova Generazione $p_i^{(t)} \sim \sigma_t \mathcal{N}(0, C_t)$
  • Ricombinazione $p_{t+1} = \sum_{i=1}^m \omega_i p_i^{(t)}$
  • Aggiorno v_c^t e v_σ^t
  • Aggiorno C_t e σ_t
• **end for**
	• Inizializzo $t = 0$, $v_c^0 = 0$, $v_\sigma^0 = 0$, $C_1 = \mathbb{I}$
	• Scelgo punto di partenza $p_1 \in \mathcal{M}$
	• **for** $t = 1$ **to** T **do**:
	  • Nuova Generazione $u_i^{(t)} \sim \mathcal{N}(0, \sigma_t C_t)$
	  • Ricombinazione $\tilde{u}_t = \sum_{i=1}^m \omega_i u_i^{(t)}$
	  • Aggiorno v_c^t e v_σ^t
	  • Aggiorno C_t e σ_t
	  • $p_{t+1} = R_{p_t}(\tilde{u}_t)$
	  • Trasporto $\mathcal{T}(v_c^t)$, $\mathcal{T}(v_\sigma^t)$ e $\mathcal{T}(C_t)$
	• **end for**

Tabella 3.2: *Schema riassuntivo del CMA-ES nello spazio piatto e sulla manifold.*

Questo era il CMA-ES su manifold come presentato in [17], quello di cui presenterò i risultati nella prossima sezione in realtà non è esattamente lo stesso. Purtroppo infatti ho incontrato dei problemi numerici che portavano alla divergenza dello stepsize σ. Non essendo ancora riuscito a capirne l'origine, i risultati che presenterò faranno riferimento ad un CMA-ES con σ fissato, sebbene la matrice di covarianza e tutte le altre quantità vengano aggiornate come appena descritto.

3.3 Risultati

Ora che abbiamo visto come funzionano i nuovi algoritmi per l'ottimizzazione sulla manifold, è arrivato il momento di metterli alla prova.
Per farlo ho scelto come banco di prova un problema di density estimation, o in altre parole di apprendimento non supervisionato. Ovvero alla RTBM viene passato un set di dati con distribuzione ignota, e il suo compito consiste nell'approssimare la PDF il meglio possibile. Per allenare la macchina sui dati ho quindi usato i due algoritmi appena descritti.

Prima di mostrare i risultati però, voglio aprire una piccola parentesi sulla funzione costo. Per i problemi di learning non supervisionato come

questo, in genere vengono usati gli stimatori di massima verosimiglianza come il *maximum likelyhood* o la *loglikelyhood*, che essenzialmente richiedono alla macchina di fare una previsione sull'intero data set e massimizzano la probabilità che la macchina generi una previsione compatibile con la distribuzione di probabilità dei dati, appunto per questo vengono chiamati *maximum likelyhood*. Il problema di questi stimatori perciò è che richiedono una previsione sull'intero dataset, che nel nostro caso implica un grandissimo numero di chiamate della funzione $\tilde{\theta}$, rallentando parecchio l'esecuzione del singolo step.
Per questo ho deciso di implementare una nuove funzione costo per l'allenamento della RTBM, si tratta della *Kullback-Leibler divergence*:

$$D_{KL}(P|Q) = -\sum_x P(x) \log\left(\frac{Q(x)}{P(x)}\right), \tag{3.34}$$

dove P è la forma analitica della distribuzione di interesse mentre Q è la sua approssimazione, ovvero la previsione generata dalla RTBM nel nostro caso.
Ovviamente noi non siamo in possesso della forma analitica P, altrimenti si tratterebbe di un problema di learning supervisionato, però possiamo costruirne un'approssimazione a partire dall'istogramma dei dati. Questo significa che una volta scelti i bin, andremo a calcolare la (3.34) su ciascun bin e non più sull'intero dataset, risparmiandoci un enorme numero di chiamate della $\tilde{\theta}$. Senza dubbio perderemo dell'informazione con questa approssimazione, ma con una buona scelta di binning si può limitare l'errore commesso senza troppi problemi.

Veniamo ora al benchmark, ho generato un sample di 5000 dati a partire da una RTBM inizializzata a random, poi ho scelto sempre a caso un punto di partenza sulla manifold, comune ad entrambi e corrispondente ai seguenti parametri:

$$\begin{aligned} &W = \begin{pmatrix} 0.59 & 0.32 \end{pmatrix}, \quad T = \begin{pmatrix} 1.07 \end{pmatrix}, \quad B_v = \begin{pmatrix} -0.45 \end{pmatrix}, \\ &B_h = \begin{pmatrix} -0.29 \\ 0.01 \end{pmatrix}, \quad Q = \begin{pmatrix} 2.48 & -0.56 \\ -0.56 & 1..12 \end{pmatrix}, \end{aligned} \tag{3.35}$$

ed ho lasciato girare gli algoritmi per un numero fissato di 1000 iterazioni.
Prima ho provato ad allenare la RTBM con l'AMSGrad usando uno stepsize

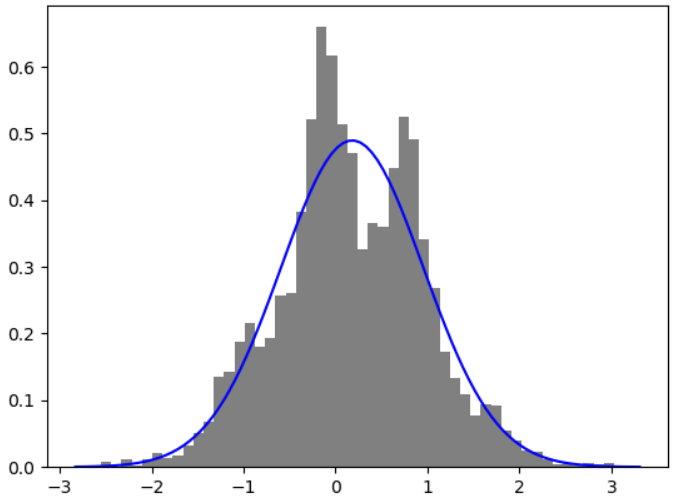


Figura 3.6: *Risultato dell'allenamento di una RTBM* $N_v = 1$, $N_h = 2$ *con parametri inziali dati dalla (3.35). La macchina è stata allenata con l'algoritmo AMSGrad riassunto nella tabella 3.1 per un numero fissato di* 1000 *iterazioni, trovando la soluzione data dalla (3.36).*

$\alpha = 0.01$ e prendendo $\beta_1 = 0.9$ e $\beta_2 = 0.999$. La soluzione trovata è visibile in figura 3.6 e corrisponde a questi parametri:

$$\begin{aligned} &W = \begin{pmatrix} 1.34 & 0.26 \end{pmatrix}, \quad T = \begin{pmatrix} 2.21 \end{pmatrix}, \quad B_v = \begin{pmatrix} -0.42 \end{pmatrix}, \\ &B_h = \begin{pmatrix} -0.30 \\ -0.06 \end{pmatrix}, \quad Q = \begin{pmatrix} 3.51 & -0.06 \\ -0.06 & 1.00 \end{pmatrix}. \end{aligned} \tag{3.36}$$

Come si vede purtroppo la soluzione non è diversa da quella trovata dallo Steepestdescent in figura 3.4. Ho provato anche a cambiare lo stepsize α e il valore di β_1 e β_2 ma il risultato non cambia. L'unico modo per ottenere un risultato più accettabile è scegliere accuratamente il punto di partenza, in qualche caso infatti si riescono a trovare soluzioni come quella mostrata in figura 3.7, tuttavia non è ancora chiaro quali siano le caratteristiche che rendano buono un punto di partenza.

Proviamo quindi col CMA-ES, in questo caso ho scelto uno stepsize $\sigma = 0.001$ e ho preso una popolazione di 20 individui ad ogni step, facen-

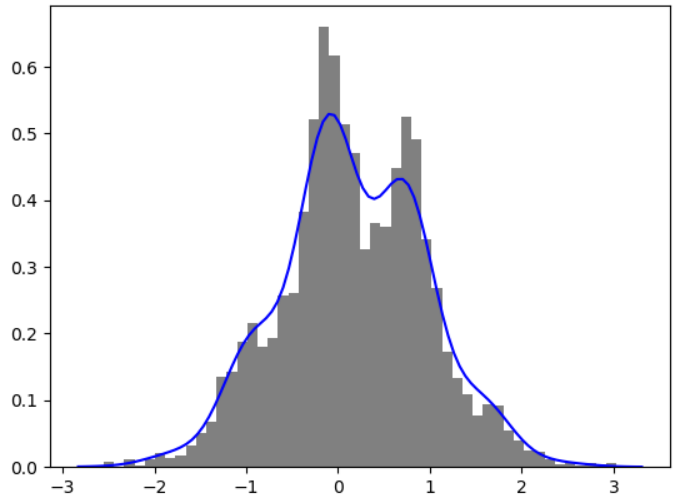


Figura 3.7: *Partendo da qualche punto particolare l'AMSGrad di tabella 3.1 riesce a trovare soluzioni migliori rispetto a quella mostrata in figura 3.6.*

do partecipare alla ricombinazione solo i migliori 10 tra loro. Dopo 1000 iterazioni la soluzione trovata corrisponde alla seguente scelta di parametri:

$$W = \begin{pmatrix} -17.6 & -9.68 \end{pmatrix}, \quad T = \begin{pmatrix} 11.38 \end{pmatrix}, \quad B_v = \begin{pmatrix} 0.88 \end{pmatrix}, \\ B_h = \begin{pmatrix} 2.67 \\ -1.17 \end{pmatrix}, \quad Q = \begin{pmatrix} 51.67 & 11.20 \\ 11.20 & 9.66 \end{pmatrix}. \tag{3.37}$$

Il risultato dell'allenamento col CMA-ES si può apprezzare in figura 3.8 e si vede subito che questa volta la RTBM modellizza molto bene la distribuzione dei dati.

La domanda che viene quindi da porsi è perché l'AMSGrad non riesca a trovare una soluzione altrettanto buona. Questa è una domanda a cui è difficile dare una risposta certa, però la figura 3.9 ci aiuta perlomeno a formulare un'ipotesi. Sono rappresentati infatti i valori della funzione costo, la Kullback-Leibler divergence in questo caso, al variare delle iterazioni sia per l'AMSGrad che per il CMA-ES. La cosa che si nota immediatamente è che in entrambi i casi, dopo una ripida discesa, il costo rimane circa costante, sintomo del fatto che si è raggiunto un minimo locale piuttosto ostico da

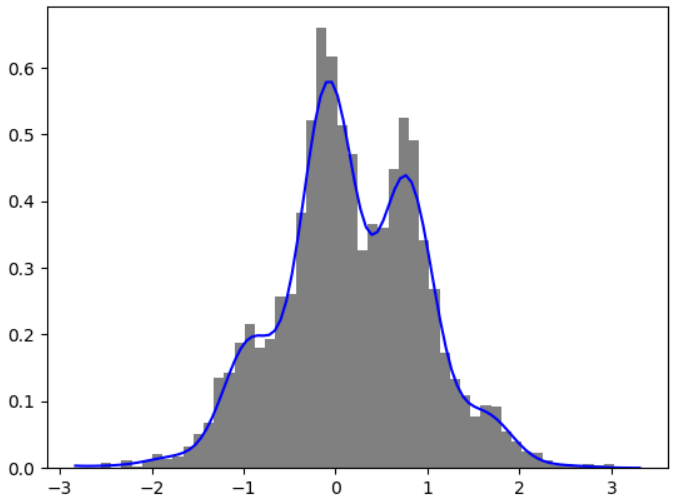


Figura 3.8: *Risultato dell'allenamento di una RTBM* $N_v = 1$, $N_h = 2$ *con parametri inziali dati dalla (3.35). La macchina è stata allenata con l'algoritmo CMA-ES riassunto nella tabella 3.2 per un numero fissato di* 1000 *iterazioni, trovando la soluzione data dalla (3.37).*

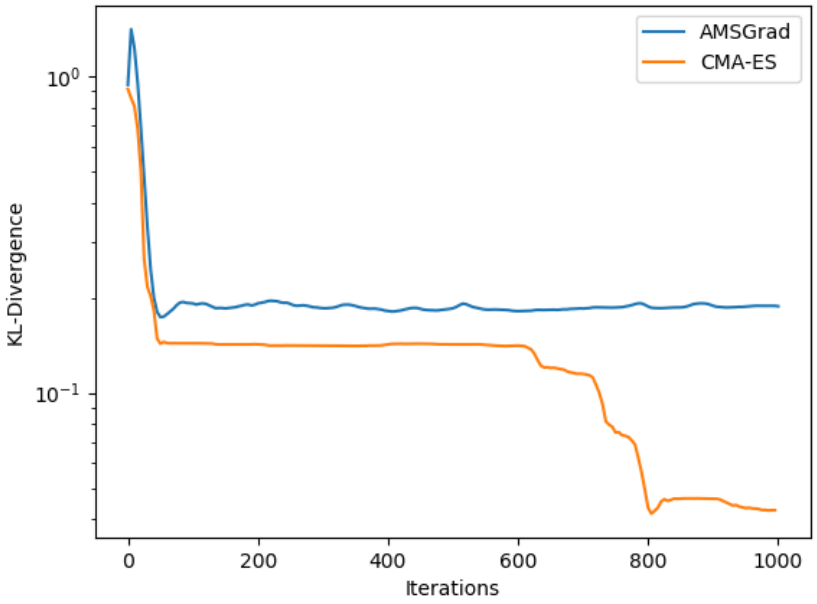


Figura 3.9: *Valore della Kullback-Leibler divergence (3.34) al variare delle iterazioni per l'AMSGrad 3.1 e il CMA-ES 3.2. Il primo rimane bloccato in un minimo locale trovando la soluzione (3.36), mentre il secondo riesce ad arrivare al minimo globale corrispondente ai parametri (3.37). Per una migliore facilità di lettura si è deciso di rappresentare l'asse delle ordinate in scala logaritmica.*

oltrepassare. Si nota appunto come l'AMSGrad non riuscirà più ad uscirne ed anche il CMA-ES spenderà un considerevole numero di iterazioni per superarlo. Questo probabilmente dipende dal fatto che l'AMSGrad sfrutta il valore del gradiente agli step precedenti per superare i minimi locali, se però come in questo caso rimane bloccato a lungo, sarà molto difficile che riesca a fuggire poiché tutti i gradienti di cui ha memoria saranno nulli o comunque molto piccoli. Al contrario invece il CMA-ES mantiene sempre un buon grado di casualità che gli permette di superare più facilmente i minimi locali.

L'ultima cosa che volevo discutere brevemente riguarda il tempo di ese-

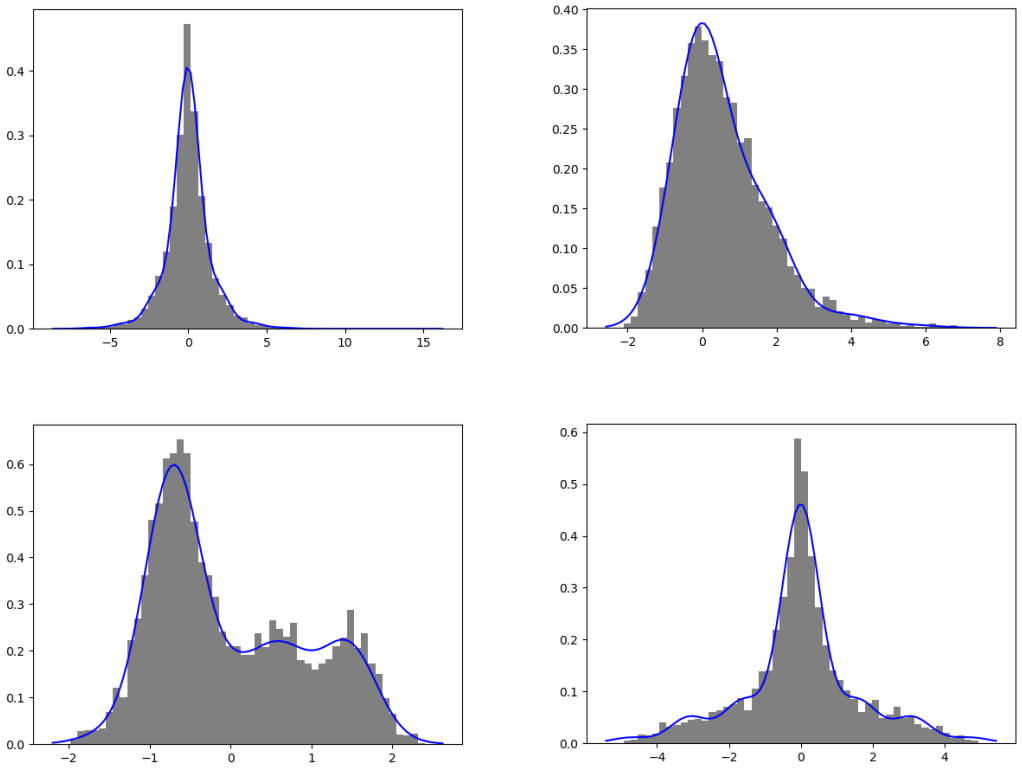


Figura 3.10: *Altri esempi di distribuzioni modellizzate allenando una RTBM* $N_v = 1$, $N_h = 2$ *con il CMA-ES (3.2). Partendo da in alto a sinistra fino in basso a destra abbiamo: una distribuzione di Laplace, una Gumbel, una mistura di due Gaussiane ai lati ed una triangolare al centro e infine sempre una Laplace ma con code triangolari.*

cuzione, ho parlato infatti di 1000 iterazioni per gli algoritmi sopra, ma non ho ancora detto esplicitamente a cosa corrispondano in termini di tempo macchina. È chiaro che se il tutto fosse più lento rispetto ad un algoritmo in $\mathbb{R}^n$ con rejection, sarebbe stato tutto inutile, forse più elegante ma all'atto pratico poco interessante. Fortunatamente anche sotto questo punto di vista troviamo un buon risultato, in questo esempio specifico per eseguire 1000 iterazione ho trovato un tempo medio di ~ 46 secondi per l'AMSGrad e ~ 89 secondi per il CMA-ES. Confrontati con i ~ 180 secondi del CMA-ES euclideo con rejection, rappresentano senza dubbio un buon traguardo.

Conclusioni

Siamo quindi giunti alla fine del mio lavoro. Avevamo iniziato nel capitolo 1 introducendo la macchina di Boltzmann e la sua nuova variante denominata Riemann-Theta Boltzman Machine, la quale ha l'enorme vantaggio di ammettere espressione analitica per la densità di probabilità del layer visibile $P(v)$. Abbiamo quindi discusso le sue proprietà e mostrato alcune applicazioni, come la generazione di samples e la density estimation, verificando quanto sia utile e versatile come modello.
Nel capitolo 2 abbiamo poi ricavato due nuove proprietà che arricchiscono le qualità delle RTBM, in particolare abbiamo visto come si possano modellizzare in maniera semplice e diretta distribuzioni condizionate e cumulative distribution functions con una minima spesa di risorse computazionali.
Infine nel capitolo 3 abbiamo parlato in generale di ottimizzazione vincolata su una manifold, per poi poter applicare quanto appreso al caso specifico delle RTBM. Infatti lo spazio dei parametri per questo tipo di modello corrisponde alla manifold $\mathbb{R}^n \times \mathcal{P}(n)$, prodotto di spazio piatto e manifold delle matrici definite positive, perciò avevamo bisogno di alcune nozioni della geometria differenziale per poter scrivere un nuovo algoritmo di allenamento della RTBM. Abbiamo quindi formulato una generalizzazione sulle manifold per due algoritmi di ottimizzazione noti: l'AMSGrad e il CMA-ES, testandoli poi su di un esempio costruito adhoc per poterli confrontare.

I buoni risultati di questa prima implementazione del CMA-ES su manifold ci spronano a continuare il lavoro iniziato sull'ottimizzazione vincolata. Prima di tutto completando l'algoritmo con l'implementazione di uno step size adattivo e in secondo luogo provando magari a parallelizzare l'esecuzione nei tanti passagi che lo permettono. Questo renderebbe così il tutto sem-

pre più competitivo, portando un vantaggio sia per le RTBM che ne fanno uso, sia in generale fornendo un utile strumento per affrontare un problema interessante come quello dell'ottimizzazione su manifold.

Oltre al perfezionamento dell'algoritmo di allenamento, il lavoro futuro verterà sulla ricerca di altre proprietà delle RTBM che non ho avuto il tempo di sviluppare nel lavoro tesi. Lo sviluppo di nuove qualità combinato con una sempre maggiore efficacia dell'allenamento renderanno senza dubbio la Riemann-Theta Boltzmann Machine un modello utile, interessante e competitivo.

Bibliografia

[1] David H. Ackley, Geoffrey E. Hinton and Terrence J. Sejnowski. *A Learning Algorithm for Boltzmann Machines.* 1985.

[2] Ruslan R. Salakhutdinov and Geoffrey E. Hinton. *An Efficient Learning Procedure for Deep Boltzmann Machines.* Neural Computation, 2012.

[3] J. J. Hopfield. *Neural networks and physical systems with emergent collective computational abilities.* Proceedings of the National Academy of Sciences of the USA, vol. 79 no. 8 pp. 2554-2558, April 1982.

[4] Daniel Krefl, Stefano Carrazza, Babak Haghighat and Jens Kahlen. *Riemann-Theta Boltzmann Machine.* arXiv:1712.07581 [stat.ML], 2018.

[5] *Gaussian-Bernoulli Deep Boltzmann Machine*

[6] Stefano Carrazza and Daniel Krefl, *Sampling the Riemann-Theta Boltzmann Machine.* arXiv:1804.07768 [stat.ML], 2018.

[7] Saralees Nadarajah and Samuel Kotz, *Mathematical Properties of the Multivariate t Distribution.* Acta Applicandae Mathematicae (2005) 89: 53-84, Springer, 2005.

[8] Peng Ding, *On the Conditional Distribution of the Multivariate t Distribution.* arXiv:1604.00561v1 [math.ST], 2016.

[9] Bernard Deconinck, Matthias Heil, Alexander Bobenko, Mark van Hoeij and Marcus Schmies, *Computing Riemann Theta Functions.* arXiv:nlin/0206009v2 [nlin.SI], 2002.

[10] P.-A. Absil, Robert Mahony and Rodolphe Sepulchre, *Optimization Algorithms on Matrix Manifolds.* Princeton University Press, 2007.

[11] Steven T. Smith, *Optimization Techniques on Riemannian Manifolds.* arXiv:1407.5965v1 [math.OC], 2014.

[12] Knut Huper and Jochen Trumpf, *Newton-like Methods for Numerical Optimization on Manifolds.* IEEE Xplore, DOI: 10.1109/ACSSC.2004.1399106, 2005.

[13] Reshad Hosseini and Suvrit Sra, *Manifold Optimization for Gaussian Mixture Models.* arXiv:1506.07677v1 [stat.ML], 2015.

[14] James Townsend, Niklas Koep and Sebastian Weichwald, *Pymanopt: A Python Toolbox for Optimization on Manifolds using Automatic Differentiation.* Journal of Machine Learning Research 17, 2016.

[15] Sashank J. Reddi, Satyen Kale and Sanjiv Kumar, *On the Convergence of Adam and Beyond.* arXiv::1904.09237 [cs.LG], 2019.

[16] Nikolaus Hansen, *The CMA Evolution Strategy: A Tutorial.* arXiv:1604.00772 [cs.LG], 2016.

[17] Sebastian Colutto, Florian Fruhauf, Matthias Fuchs and Otmar Scherzer, *The CMA-ES on Riemannian Manifolds to Reconstruct Shapes in 3-D Voxel Images.* IEEE Transactions on Evolutionary Computation, Vol. 14, 2010.

[18] Lutz Prechelt, *Early Stopping - but when?* In: Montavon G., Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg, 2012.

[19] Silvere Bonnabel, *Stochastic Gradient Descent on Riemannian Manifolds.* arXiv:1111.5280 [math.OC], 2011.

[20] Gary Bécigneul and Octavian-Eugen Ganea, *Riemannian Adaptive Ooptimization Methods.* arXiv:1810.00760 [cs.LG], 2018.

[21] Sebastian Ruder, *An Overview of Gradient Descent Optimization Algorithms.* arXiv:1609.04747 [cs.LG], 2016.

[22] Stefano Carrazza, Daniel Krefl and Andrea Papaluca, *Modelling Conditional Probabilities with Riemann-Theta Boltzmann Machines.* arXiv:1905.11313 [stat.ML], 2019.