Università degli Studi di Padova
Dipartimento di Scienze Statistiche
Corso di Laurea in

Statistica per l'Economia e l'Impresa

Stima della funzione di densità mediante reti neurali

Relatrice: Prof.ssa Giovanna Menardi
Dipartimento di Scienze Statistiche

Laureando: Riccardo Santi
Matricola n. 2067220

Anno Accademico 2024/2025

Indice

Introduzione

Ogni volta che, per effetto del caso, della mancanza di informazioni o della complessità di un fenomeno, non è possibile prevederne con certezza l'esito, ci troviamo di fronte a quello che, in termini probabilistici, viene definito esperimento casuale. L'esito numerico associato a un esperimento casuale prende il nome di *variabile casuale*: si tratta dello strumento statistico fondamentale per descrivere e analizzare l'incertezza. Conoscere una variabile casuale significa conoscerne la legge di probabilità, ovvero la distribuzione che ne regola il comportamento. Questa legge può assumere forme diverse a seconda della natura della variabile: se la variabile è discreta, ovvero assume valori in uno spazio finito o numerabile, la legge è descritta da una funzione di probabilità, che assegna a ogni valore possibile una probabilità specifica. Se la variabile è continua, ovvero assume valori in uno spazio continuo, la legge è descritta da una densità di probabilità, che non fornisce direttamente la probabilità di un singolo valore, ma permette di calcolare la probabilità che la variabile cada in una regione.

La stima o la ricostruzione della legge di probabilità di una variabile casuale, a partire dall'osservazione di un insieme di dati, è uno degli obiettivi centrali dell'inferenza statistica, perché consente di fare previsioni, prendere decisioni e comprendere meglio i meccanismi che generano l'incertezza.

Di recente, l'alta accessibilità dei *big data* ha portato alla necessità di analizzare dati ad alta dimensionalità e numerosità; ciò ha reso il problema più complesso, sia dal punto di vista teorico che da quello computazionale. In questi casi, alcuni metodi tradizionali portano notoriamente a stime non soddisfacenti. Una possibilità per risolvere questo problema è usare modelli più elaborati, propri del *machine learning*, una branca dell'*intelligenza artificiale* che sviluppa algoritmi con l'obiettivo di riconoscere *pattern* nei dati, migliorando nel tempo le proprie prestazioni senza essere esplicitamente programmati per farlo.

Una tipologia di questi modelli sono le *reti neurali*, ossia funzioni complesse definite da una sequenza di trasformazioni lineari e non lineari, in grado di approssimare un'ampia gamma di funzioni in maniera arbitrariamente precisa. Questo tipo di modello ha avuto molto successo nei problemi di apprendimento supervisionato, come nei compiti di regressione o classificazione, ma può portare a risultati validi anche per problemi di natura non supervisionata, come la stima della densità.

L'obiettivo di questo elaborato è descrivere il funzionamento dettagliato di alcuni approcci per la stima della densità basati sulle reti neurali, analizzarne pro e contro e collocarli nel contesto della letteratura esistente. A tal fine, verranno presentate le principali architetture per le reti e le relative tecniche di stima e proposti esperimenti simulati, al fine di confrontare le prestazioni con metodi tradizionali e discuterne i

risultati.

Nello specifico, nel Capitolo 1 verrà definito formalmente il problema di stima, verrà offerta una panoramica di base sui metodi classici e su alcuni problemi ricorrenti e verranno definite le metriche per valutare le stime della densità.

Nel Capitolo 2 sarà descritta la struttura generale di una rete neurale, i metodi principali per la stima dei suoi parametri, i principali punti di forza e le sue criticità.

Nel Capitolo 3 si descriveranno le principali architetture di reti neurali ideate nello specifico per la stima della densità, come i modelli *autoregressivi* e i *flussi di normalizzazione*.

Nel Capitolo 4 verrà effettuato uno studio di simulazione, dove i metodi basati su reti neurali verranno confrontati con quelli tradizionali, per valutarne la qualità delle stime della densità al variare degli scenari di interesse.

Capitolo 1

Stima della densità

1.1 Impostazione del problema

Si supponga che un esperimento casuale dia vita al vettore casuale continuo $\mathbf{X} = (X_1, \ldots, X_D)^T$, definito sul supporto $\mathcal{S} \subseteq \mathbb{R}^D$ e con funzione di densità $p(\boldsymbol{x})$. Si assuma, inoltre, di essere in possesso di un campione $\mathcal{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^N$ costituito da N realizzazioni indipendenti e identicamente distribuite (*i.i.d.*) di $\mathbf{X}$.

Il problema in esame consiste nello stimare la funzione di densità $p(\boldsymbol{x})$ a partire dalle informazioni contenute nel campione, con l'attenzione che l'incertezza della stima sia la minore possibile e sia quantificabile.

In generale, la stima di p può essere qualsiasi funzione $\hat{p} : \mathcal{S} \to \mathbb{R}^+$ che rispetti la definizione di funzione di densità, ovvero sia non negativa su tutti i punti di $\mathcal{S}$ e integri a 1. Tuttavia, la natura del fenomeno di interesse e le assunzioni fatte su di esso restringono l'insieme delle stime possibili per $p(\boldsymbol{x})$; tale insieme costituisce il *modello statistico* e viene solitamente indicato con la lettera $\mathcal{F}$.

I modelli vengono comunemente distinti in due gruppi: quelli *parametrici* e quelli *non parametrici*.

1.2 Gli approcci classici

1.2.1 Modelli parametrici

Un modo solitamente usato per delimitare le funzioni ammesse dal modello è assumere che la funzione di densità di $\mathbf{X}$ appartenga a una specifica famiglia parametrica. Questo equivale ad affermare che la funzione di densità di $\mathbf{X}$ sia completamente determinata a meno di un parametro $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k) \in \boldsymbol{\Theta}$, dove $\boldsymbol{\Theta} \subseteq \mathbb{R}^k$ viene chiamato *spazio parametrico*. C'è una corrispondenza tra ogni valore che può assumere $\boldsymbol{\theta}$ e ogni funzione all'interno del modello $\mathcal{F} = \{p(\cdot; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \boldsymbol{\Theta}\}$.

Il problema della stima di p si riconduce pertanto a quello di stimare il parametro $\boldsymbol{\theta}$ associato alla vera funzione di densità p e ci si riferirà a tale stima con $\hat{\boldsymbol{\theta}}$.

Uno dei modelli più noti e usati in statistica è, ad esempio, il modello Normale o Gaussiano, caratterizzato dal parametro $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$, i cui elementi rappresentano, rispettivamente, la media e la matrice di covarianza della variabile casuale, e definito

da una funzione di densità del tipo

$$p(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp\Big(-\tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\Big).$$

Si scriverà, in simboli, $\mathbf{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Uno dei metodi più diffusi per la stima dei parametri di un modello statistico parametrico si basa sulla cosiddetta *funzione di verosimiglianza*, una funzione $L : \boldsymbol{\Theta} \to \mathbb{R}^+$, che fornisce la densità di probabilità dello specifico campione osservato, al variare del parametro $\boldsymbol{\theta}$. La *stima di massima verosimiglianza* $\hat{\boldsymbol{\theta}}$ di $\boldsymbol{\theta}$ L o, in maniera equivalente, la sua trasformata logaritmica ℓ chiamata *log-verosimiglianza*

$$L(\hat{\boldsymbol{\theta}}) \;=\; \sup_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} L(\boldsymbol{\theta}) \;=\; \sup_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} l(\boldsymbol{\theta}).$$

Lo stimatore di massima verosimiglianza gode di varie proprietà di ottimalità, e permette la costruzione immediata di strumenti per la valutazione dell'incertezza legata alla stima.

Un altro metodo di stima classico per i metodi parametrici è il *metodo dei momenti*. Nel caso in cui $D = 1$, tale metodo si basa sull'esistenza di k funzioni che legano $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$ ai primi k momenti di $\boldsymbol{X}$. In questo modo, sostituendo ai momenti teorici la loro controparte empirica, si ricava un sistema di equazioni con soluzione unica rispetto a $\boldsymbol{\theta}$; le soluzioni di questo sistema costituiscono la stima $\hat{\boldsymbol{\theta}}$.

Si rimanda ai capitoli 2 e 3 di Azzalini (2001), dove vengono descritte le proprietà e le criticità dei due metodi descritti.

In alcuni casi, come in presenza di comportamenti multimodali, asimmetrie o code particolarmente pesanti, risulta difficile specificare una famiglia distributiva rigida a priori, con il rischio che $\mathcal{F}$ non contenga un'approssimazione adeguata della vera densità p. Esistono dunque modelli con una struttura più elaborata, che permettono di approssimare meglio funzioni più complesse, ampliando l'insieme delle funzioni possibili. Ne sono un esempio i *modelli parametrici a mistura finita*, che assumono che un'osservazione $\boldsymbol{x^{(i)}}$ possa provenire da G distribuzioni differenti, ciascuna con probabilità π_g $(g = 1, ..., G)$ di essere selezionata. Questo restringe lo spazio delle densità possibili alle funzioni con forma

$$p(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{g=1}^{G} \pi_g \, p_g(\boldsymbol{x};\boldsymbol{\theta_g}), \tag{1.1}$$

dove $\sum_{g=1}^{G} \pi_g \;= 1, \pi_g \;\geq 0$ e le densità p_g sono a loro volta tipicamente specificate all'interno di una famiglia parametrica e definite a meno del parametro $\boldsymbol{\theta_g}$.

Un caso specifico particolarmente utilizzato è quello in cui si assume che le p_g seguano una distribuzione normale multivariata; il modello che ne consegue viene chiamato *modello mistura gaussiano* (*Gaussian Mixture Model*, GMM). Il parametro da stimare in questo caso è $\boldsymbol{\theta} = \{\pi_1, \ldots, \pi_G,\ \mu_1, \ldots, \mu_G,\ \boldsymbol{\Sigma_1}, \ldots, \boldsymbol{\Sigma_G}\}$, dove μ_g e $\boldsymbol{\Sigma_g}$ sono media e matrice di covarianza della g-esima componente della mistura. La stima avviene solitamente massimizzando la verosimiglianza tramite l'algoritmo iterativo di *Expectation–Maximization*(EM) (Dempster *et al.* (1977)), siccome in generale non è ottenibile una soluzione in forma chiusa.

Per una trattazione più dettagliata dei modelli mistura si rimanda a McLachlan e Peel (2000).

Assumendo la forma (1.1) per la densità, il modello $\mathcal{F}$ diventa progressivamente più ampio con l'aumentare del numero di componenti della mistura. Infatti, data una qualsiasi densità continua p, esistono un numero di componenti G e un insieme di parametri $\boldsymbol{\theta} = \{\pi_g, \mu_g, \boldsymbol{\Sigma_g}\}_{g=1}^{G}$ che definiscono un GMM in grado di approssimare con arbitraria precisione p, sotto alcuni vincoli (Titterington *et al.*, 1985). Tuttavia, nella pratica, la scelta di G è fatta in base anche ad altri fattori (come la numerosità campionaria e considerazioni sulla possibile forma di p), alcuni dei quali influiscono direttamente sulla bontà della stima (si veda la Sezione 1.3).

1.2.2 Modelli non parametrici

L'inferenza che si basa sui modelli parametrici è tanto valida quanto lo sono le assunzioni sottostanti ad essi, e a volte non è sufficientemente robusta all'errata specificazione del modello. Di contro, i modelli non parametrici non ipotizzano una famiglia distributiva a priori, non esiste dunque un numero finito di parametri che identifichi la densità p, che invece viene stimata interamente dai dati del campione.

Il più semplice stimatore non parametrico della densità è l'*istogramma*. Supponiamo che la densità p abbia supporto su un certo intervallo che, senza perdita di generalità, assumiamo essere $[0, 1]$. Sia m un intero positivo e definiamo gli intervalli $I_1 = \left[0, \frac{1}{m}\right), \quad I_2 = \left[\frac{1}{m}, \frac{2}{m}\right), \quad \ldots, \quad I_m = \left[\frac{m-1}{m}, 1\right]$, con $h = 1/m$ la larghezza degli intervalli. Sia n_j il numero di osservazioni contenute in I_j e poniamo $\hat{p}_j = \frac{n_j}{N}$ come stima di $p_j = \int_{I_j} p(u)\,\mathrm{d}u$. Lo *stimatore a istogramma* è dato da

$$\hat{p}(x) = \sum_{j=1}^{m} \frac{\hat{p}_j}{h}\, \mathbf{1}\{x \in I_j\}.$$

La giustificazione teorica per il metodo risiede nel fatto che, per $x \in I_j$ e h piccolo,

$$\mathbb{E}\big(\hat{p}(x)\big) = \frac{\mathbb{E}(\hat{p}_j)}{h} = \frac{p_j}{h} = \frac{1}{h}\int_{I_j} p(u)\,\mathrm{d}u \approx \frac{p(x)\,h}{h} = p(x).$$

Per una trattazione più approfondita del metodo, comprensiva del caso multidimensionale, si rimanda al Capitolo 3 di Scott (2015).

Un altro metodo molto utilizzato è lo stimatore *kernel* della densità (*Kernel Density Estimator*, KDE), che rispetto agli istogrammi ha il vantaggio di ottenere stime più lisce e, in molti casi, di convergere più rapidamente alla vera funzione da approssimare (si veda Wasserman, 2006, Capitolo 6).

Nella sua forma base, la stima ha forma

$$\hat{p}(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} K_H(\boldsymbol{x} - \boldsymbol{x^{(i)}}), \tag{1.2}$$

dove $K_H(\boldsymbol{x}) = |H|^{-\frac{1}{2}}\; K\big(H^{-\frac{1}{2}}\,\boldsymbol{x}\big)$ è la funzione *kernel* scalata, $H \in \mathbb{R}^{D\times D}$ è la matrice di lisciamento (o *bandwidth*) simmetrica e definita positiva e $K : \mathbb{R}^D \to \mathbb{R}$ è la funzione *kernel*, tipicamente positiva e tale che $\displaystyle\int_{\mathbb{R}^D} K(\mathbf{u})\, d\mathbf{u} = 1$.

Una scelta comune per K_H è la densità di una normale multivariata, dove la matrice di covarianza è definita dall'iperparametro H, da scegliere a priori o da calibrare empiricamente; la scelta di questo iperparametro determina la liscezza di $\hat{p}$ ed è cruciale per la qualità della stima.

Un altro metodo rilevante è quello dei k *vicini più prossimi* (*k-Nearest Neighbours*, KNN), che produce la stima

$$\hat{p}(\boldsymbol{x}) \;=\; \frac{k}{N\,\lambda\big(B(\boldsymbol{x},\, R_{(k)}(\boldsymbol{x}))\big)}, \tag{1.3}$$

dove $k \in \mathbb{N}$ è il numero di punti più vicini considerati, $\lambda()$ la misura di Lebesgue in $\mathbb{R}^d$, $B(\boldsymbol{x}, r) = \{\, \boldsymbol{a} \in \mathbb{R}^d : \|\boldsymbol{a} - \boldsymbol{x}\| \leq r\}$ e $R_{(k)}(\mathbf{x})$ è la distanza tra $\boldsymbol{x}$ e il suo k-esimo punto più vicino (per approfondimenti si veda ad esempio Biau e Devroye, 2015, Capitolo 3).

Si cita anche lo stimatore a *serie ortogonali*, che espande la densità in una base di funzioni ortonormali, come la serie di *Fourier* (si veda Scott, 2015, Sezione 6.1.3).

1.3 Misure di accuratezza

Data la natura casuale di $\mathbf{X}$, una ricostruzione perfetta di $p(\boldsymbol{x})$ è impossibile; dunque, è necessario quantificare l'incertezza della stima con delle misure di distanza tra $p(\boldsymbol{x})$ e $\hat{p}(\boldsymbol{x})$. Una possibilità è definire una *funzione di perdita* L e considerare il suo valore atteso puntuale:

$$\mathbb{E}\big[L\big(p(\boldsymbol{x}),\, \hat{p}(\boldsymbol{x})\big)\big]. \tag{1.4}$$

Per valutare la stima di una densità, invece che usare direttamente la (1.4), di solito si preferisce calcolare il valore atteso dell'integrale della funzione di perdita calcolato sull'intero supporto di $\boldsymbol{X}$, siccome è un valore più indicativo della qualità globale della stima:

$$\mathbb{E}\big[\int_{\mathcal{S}} L\big(p(\boldsymbol{x}),\, \hat{p}(\boldsymbol{x})\big)\, d\boldsymbol{x}\big]. \tag{1.5}$$

La bontà di un modello per una certa densità da approssimare sarà misurata in relazione alla (1.4) o alla (1.5): a valori bassi di queste metriche corrisponderanno modelli migliori per stimare la specifica densità $p(\boldsymbol{x})$.

Ad esempio, scegliendo come funzione di perdita l'*errore assoluto* (Absolute Error, AE) $\big|\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x})\big|$ si ottiene per la (1.4) l'*errore assoluto medio* (Mean Absolute Error, MAE)

$$\mathrm{MAE}\big(\hat{p}(\boldsymbol{x})\big) = \mathbb{E}\big[\big|\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x})\big|\big],$$

e per la (1.5) questo significa calcolare il valore atteso dell'*errore assoluto integrato* (Integrated Absolute Error, IAE) $\int_{\mathbb{R}^D} \big|\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x})\big|\, d\boldsymbol{x}$, ottenendo l'*errore assoluto medio integrato* (Mean Integrated Absolute Error, MIAE)

$$\mathrm{MIAE}\big(\hat{p}(\boldsymbol{x})\big) = \int_{\mathcal{S}} \mathbb{E}\big[\big|\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x})\big|\big]\, d\boldsymbol{x} = \mathbb{E}\int_{\mathcal{S}} \big|\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x})\big|\, d\boldsymbol{x}. \tag{1.6}$$

Un'altra funzione di perdita, più comunemente utilizzata, è quella quadratica, che attribuisce una misura di errore maggiore a differenze elevate tra $\hat{p}(\boldsymbol{x})$ e $p(\boldsymbol{x})$ e peso inferiore a scarti ridotti. La specifica forma risultante per la (1.4) data questa funzione di perdita è l'*errore quadratico medio*(*Mean Squared Error*, MSE)

$$\mathrm{MSE}\big(\hat{p}(\boldsymbol{x})\big) = \mathbb{E}\big\{(\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x}))^2\big\}. \tag{1.7}$$

Alcune metriche, tra cui la (1.7), sono scomponibili in due parti, una che è funzione del $\mathrm{Bias}(\hat{p}(\boldsymbol{x}), p(\boldsymbol{x})) = \mathbb{E}[\hat{p}(\boldsymbol{x})] - p(\boldsymbol{x})$, e un'altra che è funzione della *varianza* $\mathrm{Var}(\hat{p}(\boldsymbol{x})) = \mathbb{E}\big[\big(\hat{p}(\boldsymbol{x}) - \mathbb{E}[\hat{p}(\boldsymbol{x})]\big)^2\big]$. Ad esempio, l'MSE è scomponibile come

$$\begin{aligned}
\mathrm{MSE}\big(\hat{p}(\boldsymbol{x})\big) &= \mathbb{E}\big[(\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x}))^2\big] \\
&= \mathbb{E}\Big[\big(\hat{p}(\boldsymbol{x}) - \mathbb{E}[\hat{p}(\boldsymbol{x})] + \mathbb{E}[\hat{p}(\boldsymbol{x})] - p(\boldsymbol{x})\big)^2\Big] \\
&= \mathbb{E}\Big[\big(\hat{p}(\boldsymbol{x}) - \mathbb{E}[\hat{p}(\boldsymbol{x})]\big)^2\Big] + \big(\mathbb{E}[\hat{p}(\boldsymbol{x})] - p(\boldsymbol{x})\big)^2 + 0 \\
&= \mathrm{Var}\big(\hat{p}(\boldsymbol{x})\big) + \mathrm{Bias}^2\big(\hat{p}(\boldsymbol{x})\big).
\end{aligned}$$

e in maniera simile lo è anche la (1.5) con funzione di perdita quadratica, ovvero l'*errore quadratico medio integrato* (Mean Integrated Squared Error, MISE)

$$\mathrm{MISE}\big(\hat{p}(\boldsymbol{x})\big) = \int_{\mathcal{S}} \mathbb{E}\big[\big(\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x})\big)^2\big]\, d\boldsymbol{x} = \mathbb{E}\int_{\mathcal{S}} \big(\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x})\big)^2\, d\boldsymbol{x}. \tag{1.8}$$

Siccome *bias* e varianza sono dipendenti tra loro, questa scomposizione mette in evidenza la necessità di bilanciare queste due quantità, in modo che la misura di errore complessiva sia la minore possibile. Questo problema è noto in statistica come *compromesso bias-varianza*.

In generale, modelli più flessibili comportano stime con basso *bias* ma varianza più alta. Nel caso parametrico, una maggiore flessibilità si ottiene solitamente aumentando il numero di parametri e ampliando di conseguenza il modello parametrico $\mathcal{F}$, mentre nei modelli non parametrici le scelte sugli iperparametri sono spesso determinanti, come ad esempio lo è la scelta della matrice di lisciamento nel KDE, che viene solitamente definita per minimizzare l'*errore quadratico medio integrato asintotico* (*Asymptotic Mean Integrated Squared Error*, AMISE), un'approssimazione del MISE (si veda ad esempio Scott, 2015, Sezione 6.2.1).

Il compromesso *bias*-varianza è strettamente legato alla capacità di generalizzazione di un modello: se è molto flessibile, il modello è ottimizzato sui dati osservati, ma fornirà stime molto distanti da $p(\boldsymbol{x})$ per nuove osservazioni, siccome è molto variabile a seconda dello specifico campione usato per effettuare la stima. Per alcuni modelli, la media empirica dell'errore quadratico calcolata sul campione responsabile della stima

$$C = \frac{1}{N}\sum_{i=1}^{N} \big(p(\boldsymbol{x^{(i)}}) - \ \hat{p}(\boldsymbol{x^{(i)}})\big)^2 \tag{1.9}$$

diminuisce sistematicamente all'aumentare della complessità, senza riflettere la reale capacità di generalizzazione del modello.

A questo proposito, si definisce un secondo campione di dati generati da $\mathbf{X}$, che non hanno contribuito alla stima $\hat{p}$, noto come *insieme di verifica* (o *test set*),

indicato con $\mathcal{X}_{test}$ e di dimensione N_{test}. In alcuni casi, per evitare ambiguità, ci si riferirà al campione usato per la stima con il nome *insieme di addestramento* (o *training set*), che verrà indicato con $\mathcal{X}_{train}$ e la sua numerosità con N_{train}.

Solitamente si è interessati alla capacità di generalizzare di un modello, ovvero alla sua performance predittiva sul *test set*. La densità in corrispondenza dei punti del *test set* non è realisticamente nota a priori per nessuna applicazione reale. Una possibilità per valutare la precisione della stima è tramite un'approssimazione della (1.7) o della (1.8) tramite *cross-validation* (si veda ad esempio Wasserman, 2006, Sezione 6.1).

Si precisa che la terminologia utilizzata e la distinzione in *train* e *test set* non sono proprie della statistica, ma sono invece più comuni nell'affine campo dell' *apprendimento automatico* (*machine learning*), dove l'obiettivo è maggiormente predittivo.

In questo contesto ci si riferisce alla situazione di alto *bias* e bassa varianza con il termine *underfitting*, dove il modello non è abbastanza flessibile da cogliere la struttura dei dati; di contro si usa *overfitting* per la situazione opposta.

1.4 Maledizione della dimensionalità

In statistica, la *maledizione della dimensionalità* si riferisce al peggioramento della qualità delle stime che si riscontra all'aumentare del numero di dimensioni del supporto dei dati. Uno dei principali problemi che si riscontra all'aumentare della dimensionalità è che i dati diventano progressivamente più sparsi tra loro.

Per illustrare la natura del problema, si consideri un esempio tratto da Hastie *et al.* (2009, Sezione 2.5). Si immagini di avere dei dati distribuiti in modo uniforme dentro un ipercubo di lato unitario e dimensione D. Si vuole costruire, attorno a un punto x, un piccolo cubo che contenga una certa frazione f dei dati. Al crescere di D, per includere anche solo una piccola frazione dei dati, ad esempio, l'1%, bisogna considerare cubi sempre più grandi. Ad esempio, in una sola dimensione è sufficiente un intervallo di lunghezza 0.1, ma in dieci dimensioni è necessario un cubo che copra circa il 63% della lunghezza su ciascuna variabile.

I metodi non parametrici, a causa della loro elevata flessibilità, sono notoriamente sensibili alla maledizione della dimensionalità. Infatti, utilizzare in alta dimensione metodi non parametrici come il KNN non permette di cogliere strutture locali, siccome le stime dipendono da intorni di $\boldsymbol{x}$ troppo larghi. Ridurre drasticamente p diminuirebbe l'ampiezza dell'intorno, ma aumenterebbe la varianza della nostra stima, siccome $\lambda\big(B(\boldsymbol{x}, R_{(k)}(\boldsymbol{x}))\big)$ nella (1.3) dipenderebbe da meno osservazioni. Appare quindi evidente come, in questo caso, la maledizione della dimensionalità sia un altro modo di esprimere il fatto che il compromesso *bias*-varianza non possa essere realizzato in modo soddisfacente in dimensioni elevate. Considerazioni simili possono essere fatte per altri metodi non parametrici per i quali è dimostrabile che il tasso di convergenza dell'MSE risulta proporzionale a $N^{\frac{-4}{4+D}}$ (si veda ad esempio Wasserman, 2006, Sezione 4.5). Ne consegue che, tenendo fissa la numerosità campionaria, con l'aumento di D si ottengono stime che peggiorano esponenzialmente in termini di MSE. Si rimanda al Capitolo 7 di Scott (2015) per un approfondimento sulla maledizione della dimensionalità e per qualche esempio pratico.

Tuttavia, quasi sempre i dati ad alta dimensionalità appartenenti a $\mathbb{R}^D$ non si distribuiscono uniformemente in tutto lo spazio $\mathbb{R}^D$, ma si concentrano invece su un sottospazio di dimensione inferiore. Ciò avviene a causa delle dipendenze esistenti tra le variabili, che riducono la complessità effettiva della struttura dei dati. Appare quindi naturale pensare di riuscire a sfruttare queste relazioni per ottenere una migliore efficienza di stima. Esistono numerosi metodi che permettono di ricondursi a spazi latenti di numerosità inferiore, con la minima perdita di informazione possibile. Tra i più comuni si citano l' *analisi delle componenti principali*, l'*analisi fattoriale*, e la *Projection Pursuit*. Si rimanda ad Hastie *et al.* (2009, Sezioni 14.5 e 14.7) per un approfondimento.

Capitolo 2

Reti neurali artificiali

2.1 Generalità sulle reti neurali

Le reti neurali (*neural networks*) sono un modello di apprendimento automatico sempre più diffuso nell'analisi dei dati complessi, utilizzato per affrontare problemi di regressione, classificazione, o in generale di stima di funzioni e riconoscimento di *pattern* interessanti nei dati.

Questi modelli, solitamente con una forma più complessa di quella delineata nella Sezione 2.1, sono risultati utili in svariati campi applicativi, come l'analisi di immagini, il riconoscimento e la sintesi vocale, l'elaborazione del linguaggio naturale, l'identificazione di valori anomali e la generazione di contenuti sintetici (si rimanda a Abiodun *et al.*, 2018, per approfondimenti). In questi e altri settori le reti neurali si sono diffuse grazie alla loro capacità di produrre risultati di notevole accuratezza rispetto a metodi più classici di analisi dei dati.

L'architettura di una rete neurale, ispirata al cervello biologico, è composta da unità elementari chiamate *neuroni* o *nodi*, connesse tra loro e responsabili dell'elaborazione dell'informazione. Il modo in cui queste unità elaborano i dati è semplice, ma i collegamenti tra nodi rendono tali trasformazioni molto complicate, in modo vagamente simile a come i neuroni nel cervello umano riescono a produrre pensieri complessi.

L'architettura e il funzionamento della rete dipendono dallo specifico obiettivo di analisi, ma ciò che accomuna le diverse reti è l'elaborazione dell'informazione attraverso l'applicazione successiva di trasformazioni lineari e non lineari, volte a minimizzare una precisa funzione di costo. Solitamente si è interessati allo scenario in cui si ha a disposizione un campione $\mathcal{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ di realizzazioni della variabile $\boldsymbol{X}$, e si vuole fornire una stima $\hat{\boldsymbol{y}}^{(i)}$ per una o più quantità di interesse $\boldsymbol{y}^{(i)}$ associate a ogni possibile valore assunto da $\boldsymbol{x}^{(i)}$.

Una rete è organizzata in insiemi di neuroni chiamati *strati*. Solitamente gli elementi del vettore $\boldsymbol{x}$ coincidono con i neuroni del primo strato, che per questo motivo viene chiamato *strato di input* e il vettore dei neuroni che lo compongono viene indicato con $\boldsymbol{a}^{(0)} = (x_1, \ldots, x_D)^T$, mentre gli elementi del vettore $\hat{\boldsymbol{y}}$ coincidono con i neuroni dell'ultimo strato, che per questo motivo viene chiamato *strato di output* e i suoi neuroni indicati con $\boldsymbol{a}^{(L)}$. Gli strati intermedi vengono chiamati *nascosti* e i neuroni che li compongono sono responsabili delle trasformazioni dei dati. In una classica rete neurale *feed-forward* (si veda, ad esempio, l'introduzione

di Haykin, 2009, e la Figura 2.1) in ogni strato nascosto vengono modificati i valori dei neuroni dello strato precedente, attraverso due trasformazioni:

1. Per ognuno dei n_l neuroni dello strato si calcola una combinazione lineare dei valori assunti dai neuroni nello strato precedente, definita da

$$z_j^{(l)} = \sum_{k=1}^{n_{l-1}} w_{jk}^{(l)}\, a_k^{(l-1)} \;+\; b_j^{(l)}, \qquad j = 1, \ldots, n_l.$$

   In maniera più compatta per l'intero strato si può scrivere

$$\boldsymbol{z^{(l)}} = \boldsymbol{W^{(l)}}\, \boldsymbol{a^{(l-1)}} + \boldsymbol{b^{(l)}},$$

   dove

   - $\boldsymbol{W^{(l)}} = \begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \cdots & w_{1,n_{l-1}}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \cdots & w_{2,n_{l-1}}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_l,1}^{(l)} & w_{n_l,2}^{(l)} & \cdots & w_{n_l,n_{l-1}}^{(l)} \end{pmatrix} \in \mathbb{R}^{n_l \times n_{l-1}}$
     viene detta *matrice dei pesi* (*weights*) dello strato l, i cui elementi sono denotati da $w_{jk}^{(l)}$, con $j = 1, \ldots, n_l$ e $k = 1, \ldots, n_{l-1}$;
   - $\boldsymbol{b^{(l)}} = \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{n_l}^{(l)} \end{pmatrix} \in \mathbb{R}^{n_l}$
     è chiamato *vettore dei bias* dello strato l, i cui elementi sono denotati da $b_j^{(l)}$ per $j = 1, \ldots, n_l$.
     Insieme questi due formano il vettore dei parametri

$$\boldsymbol{\theta} \;=\; \left\{ \boldsymbol{W^{(l)}}, \boldsymbol{b^{(l)}} \right\}_{l=1}^{(L)}, \tag{2.1}$$

2. Ad ognuno dei valori ottenuti al punto precedente viene applicata una funzione non lineare

$$a_j^{(l)} = \sigma^{(l)}\big(z_j^{(l)}\big), \qquad j = 1, \ldots, n_l,$$

   o in maniera più compatta

$$\boldsymbol{a^{(l)}} = \sigma^{(l)}\big(\boldsymbol{z^{(l)}}\big).$$

I vettori $\boldsymbol{z^{(l)}}$ e $\boldsymbol{a^{(l)}}$ vengono chiamati rispettivamente *pre-attivazione* e *attivazione* dello strato l. In maniera coerente, $\sigma^{(l)}$ è detta *funzione di attivazione* dello strato, viene applicata a ogni elemento di $\boldsymbol{z^{(l)}}$ e usata per modellare relazioni non lineari tra neuroni di strati adiacenti. Solitamente la funzione di attivazione è la stessa per tutti gli strati, fatta eccezione per l'ultimo, dove viene scelta una funzione con codominio coerente con $\boldsymbol{y}$.

Scelte comuni per $\sigma^{(l)}$, sono:

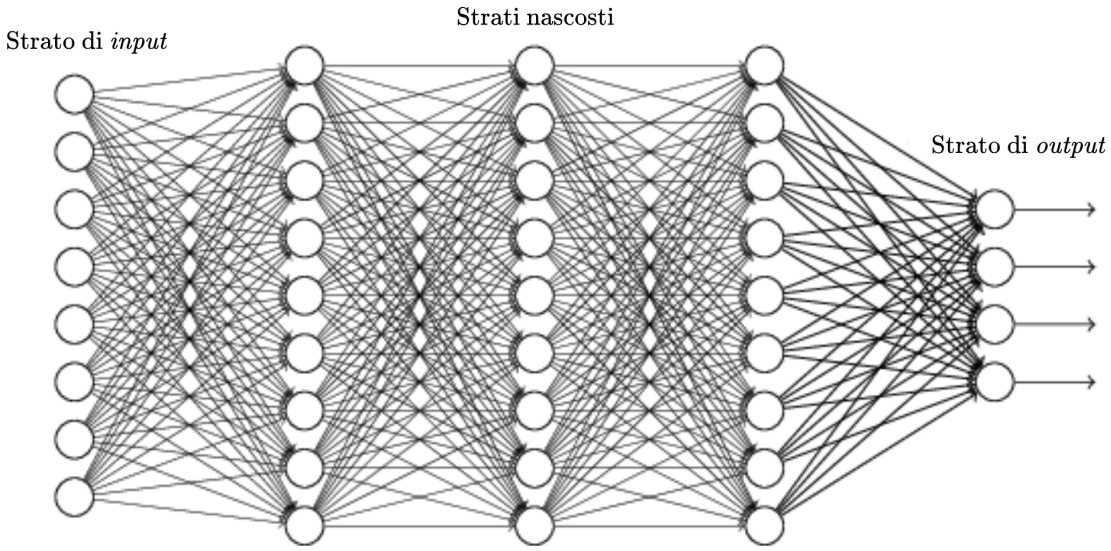

Figura 2.1: Schema classico di una rete neurale *feed-forward*, i cerchi sono i neuroni, le frecce indicano gli argomenti passati come *input* per lo strato successivo. I risultati di uno strato vengono passati solo allo strato successivo, mai a quelli precedenti.

- la *ReLU* (*Rectified Linear Unit*):

$$\sigma(z) = \max(0, z).$$

- la *Leaky ReLU*:

$$\sigma(z) = \begin{cases} z, & z \geq 0, \\ \alpha\, z, & z < 0, \end{cases}$$

- l'inversa della funzione logistica:

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

- la tangente iperbolica, *tanh*:

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}.$$

Per una trattazione più esaustiva delle funzioni di attivazione si rimanda a Apicella *et al.* (2021).

Quando un punto $\boldsymbol{x}^{(i)}$ viene trasformato nel modo appena descritto, si dice che è stato eseguito un *forward pass* nella rete.

Supponendo che $\boldsymbol{\theta}$ sia noto, una rete neurale è una funzione che prende in *input* una generica osservazione $\boldsymbol{x}^{(i)}$, applica una composizione di funzioni in successione e restituisce l'*output* $\hat{\boldsymbol{y}}^{(i)}$. Tuttavia, è irrealistico pensare di conoscere il valore del parametro $\boldsymbol{\theta}$, che andrà quindi stimato con i dati in possesso.

2.2 Addestramento della rete neurale

2.2.1 *Backpropagation*

La stima dei parametri, nota nella comunità del *machine learning* come *addestramento della rete*, consiste nell'ottimizzazione, rispetto al parametro $\boldsymbol{\theta}$ definito nella (2.1), di una funzione di costo C che misura l'errore commesso dalla rete.

Nei problemi di apprendimento supervisionato, con un *training set* $\{\mathcal{X}_{\text{train}}, \mathcal{Y}_{\text{train}}\} = \{\boldsymbol{x^{(i)}}, \boldsymbol{y^{(i)}}\}_{i=1}^{N}$, l'algoritmo di addestramento più utilizzato è l'*error backpropagation* (Linnainmaa, 1970). In questi casi, una scelta comune per la funzione di costo è l'errore sul *training set* con funzione di perdita L quadratica:

$$C \;=\; \frac{1}{N}\sum_{i=1}^{N} L_i \;=\; \frac{1}{N}\sum_{i=1}^{N} L\big(\boldsymbol{y^{(i)}}, \hat{\boldsymbol{y}}^{(\boldsymbol{i})}(\boldsymbol{x^{(i)}})\big) \;=\; \frac{1}{N}\sum_{i=1}^{N} \Big\|\boldsymbol{y^{(i)}} - \hat{\boldsymbol{y}}^{(\boldsymbol{i})}(\boldsymbol{x^{(i)}})\Big\|_2^2.$$

In generale la funzione di costo usata nell'*error backpropagation* può essere qualsiasi funzione che rispetti le seguenti assunzioni:

Assunzione 1 *Il costo complessivo C è esprimibile come media dei costi elementari $L\big(\boldsymbol{y^{(i)}}, \hat{\boldsymbol{y}}^{(\boldsymbol{i})}(\boldsymbol{x^{(i)}})\big)$ su tutte le N osservazioni del training set.*

Assunzione 2 *Per ciascuna coppia $\big(\boldsymbol{y^{(i)}}, \hat{\boldsymbol{y}}^{(\boldsymbol{i})}\big)$, il costo elementare $L\big(\boldsymbol{y^{(i)}}, \hat{\boldsymbol{y}}^{(\boldsymbol{i})}(\boldsymbol{x^{(i)}})\big)$ può essere considerato come funzione diretta dell'output della rete.*

Assunzione 3 *Il costo C è differenziabile rispetto agli output $\hat{\boldsymbol{y}}^{(\boldsymbol{i})}$ e, di conseguenza (con attivazioni adeguate), è differenziabile rispetto ai pesi $w_{jk}^{(l)}$ e ai bias $b_j^{(l)}$.*

L'*error backpropagation* è un algoritmo iterativo che permette di aggiornare $w_{jk}^{(l)}$ e $b_j^{(l)}(\forall j, k)$, sfruttando il metodo della discesa del gradiente e la regola della derivazione a catena. L'idea alla base è quella di modificare i parametri (all'inizio scelti casualmente) lungo le direzioni opposte a quelle indicate dal gradiente di C, in modo che questi cambiamenti portino a una diminuzione della funzione di costo.

Più precisamente, si definiscono le seguenti quantità:

- Per ogni neurone j dello strato L, definiamo

$$\delta_j^{(L)} \;=\; \frac{\partial C}{\partial z_j^{(L)}} \;=\; \frac{\partial C}{\partial a_j^{(L)}}\,\frac{\partial\, a_j^{(L)}}{\partial\, z_j^{(L)}}, \;=\; \frac{\partial C}{\partial a_j^{(L)}}\; a'\big(z_j^{(L)}\big), \tag{2.2}$$

  dove $\delta_j^{(L)}$ fornisce l'informazione su quanto e in quale direzione il costo complessivo C cambierebbe se modificassimo infinitesimamente il valore di $z_j^{(L)}$. Se $|\delta_j^{(L)}|$ è elevato, una piccola modifica di $z_j^{(L)}$ cambia di molto il costo: il valore del neurone è *errato* rispetto all'*output* desiderato, e un suo piccolo cambiamento porta una diminuzione sostanziale di C. Per questo motivo $\delta_j^{(L)}$ viene considerata una misura di errore nel neurone.

  Compattamente scriviamo

$$\boldsymbol{\delta}^{(L)} = (\delta_1^{(L)}, \ldots, \delta_j^{(L)}, \ldots, \delta_{n_L}^{(L)})^T.$$

Dove

$$\boldsymbol{\delta}^{(L)} \;=\; \nabla_a C \;\odot a'\big(\boldsymbol{z}^{(\boldsymbol{L})}\big), \quad \boldsymbol{\delta}^{(L)} \in \mathbb{R}^{n_L},$$

con $\nabla_a C = \big(\frac{\partial C}{\partial a_1^{(L)}}, \ldots, \frac{\partial C}{\partial a_{n_L}^{(L)}}\big)^T$, $a'(\boldsymbol{z}^{(\boldsymbol{L})})$ è il vettore delle derivate $\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$ e "$\odot$" denota il *prodotto di Hadamard*, ossia la moltiplicazione elemento per elemento tra vettori o, in generale, tra matrici.

- Per ogni $l = L-1, \ldots, 1$, e per ogni neurone in quello strato si definisce:

$$\delta_j^{(l)} \;=\; \frac{\partial C}{\partial z_j^{(l)}} \;=\; \sum_k w_{kj}^{(l+1)}\, \delta_k^{(l+1)}\, \sigma'\big(z_j^{(l)}\big). \tag{2.3}$$

  Più compattamente, per ogni strato:

$$\boldsymbol{\delta}^{(l)} \;=\; \frac{\partial C}{\partial \boldsymbol{z}^{(\boldsymbol{l})}} \;= \big(\boldsymbol{W}^{(\boldsymbol{l+1})}\big)^T \boldsymbol{\delta}^{(\boldsymbol{l+1})} \;\odot\; a'\big(\boldsymbol{z}^{(\boldsymbol{l})}\big).$$

  Si noti che $\boldsymbol{\delta}^{(\boldsymbol{l})}$ è una funzione dell'errore $\boldsymbol{\delta}^{(\boldsymbol{l+1})}$ nello strato successivo. Quindi, dato l'errore $\boldsymbol{\delta}^{(\boldsymbol{l+1})}$ al livello $(l+1)$, applicando la (2.3), possiamo pensare intuitivamente di fare *propagare all'indietro l'errore* attraverso la rete, ottenendo così una misura dell'errore in uscita dal livello l, da questo prende spunto il nome dell'algoritmo.

- Siccome è di interesse la stima di tutti i parametri $w_{jk}^{(l)}$ e $b_j^{(l)}$ si definisce nello specifico per i bias:

$$\frac{\partial C}{\partial b_j^{(l)}} = \delta_j^{(l)} \tag{2.4}$$

  e per i pesi:

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \delta_j^{(l)} \;, \tag{2.5}$$

  che sono funzioni della (2.2) e della (2.3).

Le Equazioni 2.2 - 2.5 vengono usate nell'algoritmo di *backpropagation* per il calcolo dei gradienti di C rispetto ai parametri da stimare; lo pseudocodice è fornito nell'Algoritmo 1, riferito al calcolo per una generica osservazione del *training set* (per ulteriori dettagli e per le dimostrazioni delle (2.2 - 2.5) si rimanda a Nielsen, 2015, Capitolo 2).

È pratica comune implementare la *backpropagation* con un algoritmo di apprendimento che combina il gradiente di più osservazioni: il più comune è la discesa del gradiente stocastica (*Stochastic Gradient Descent, SGD*) (Robbins e Monro, 1951), che sceglie casualmente m osservazioni e calcola la funzione di costo come $C = \frac{1}{m}\sum_{i=1}^m L_i$, il cui gradiente è facilmente calcolabile grazie all'Assunzione 1. Una volta calcolato, si aggiornano i parametri secondo le classiche regole della discesa del gradiente:

$$w_{jk}^{(l)} \;\longleftarrow\; w_{jk}^{(l)} \;-\; \eta\, \frac{\partial C}{\partial w_{jk}^{(l)}}\,,$$

$$b_j^{(l)} \;\longleftarrow\; b_j^{(l)} \;-\; \eta\, \frac{\partial C}{\partial b_j^{(l)}}\,,$$

Algorithm 1: *Backpropagation* per rete *feedforward*

Input : Osservazione $\big(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\big)$, pesi e bias $\{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\}_{l=2}^{L}$, funzione di costo C, attivazione σ

Output: $\dfrac{\partial C^{(i)}}{\partial \boldsymbol{W}^{(l)}}$, $\dfrac{\partial C^{(i)}}{\partial \boldsymbol{b}^{(l)}}$ per ogni livello $\boldsymbol{l}$

`// 1. Imposta attivazione ingresso`

$\boldsymbol{a^1} \leftarrow \boldsymbol{x}^{(i)}$;

`// 2. Forward pass`

for $\boldsymbol{l} \leftarrow \mathbf{2}$ **to** $\boldsymbol{L}$ **do**

  $\boldsymbol{z}^{(l)} \leftarrow \boldsymbol{W}^{(l)}\, \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$;

  $\boldsymbol{a}^{(l)} \leftarrow \sigma\big(\boldsymbol{z}^{(l)}\big)$;

`// 3. Errore nell'ultimo strato`

$\boldsymbol{\delta}^{(L)} \leftarrow \nabla_{\boldsymbol{a}} C^{(i)} \odot \sigma'\big(\boldsymbol{z}^{(L)}\big)$;

`// 4. Propagazione all'indietro dell'errore`

for $\boldsymbol{l} \leftarrow \boldsymbol{L-1}$ **down to** $\mathbf{2}$ **do**

  $\boldsymbol{\delta}^{(l)} \leftarrow \big(\boldsymbol{W}^{(l+1)}\big)^{\mathsf{T}} \boldsymbol{\delta}^{(l+1)} \odot \sigma'\big(\boldsymbol{z}^{(l)}\big)$;

`// 5. Calcolo dei gradienti`

for $l \leftarrow 2$ **to** L **do**

  for $j \leftarrow 1$ **to** n_l **do**

    $\dfrac{\partial C^{(i)}}{\partial b_j^{(l)}} = \delta_j^{(l)}$

    for $k \leftarrow 1$ **to** n_{l-1} **do**

      $\dfrac{\partial C^{(i)}}{\partial W_{jk}^{(l)}} = \delta_j^{(l)}\, a_k^{(l-1)}$

dove $\eta > 0$ è il *learning rate* e l'insieme delle m osservazioni viene chiamato *mini-batch*. Questo processo è iterativo e si ripete per un numero prefissato di *epoche*, ossia passaggi in cui l'intero dataset viene processato una volta, o fino al soddisfacimento di un criterio di arresto (si veda la Sezione 2.2.2).

Lo pseudo codice per questa procedura è fornito nell'Algoritmo 2.

L'*SGD* delinea le basi su cui si fondano algoritmi più complessi e più efficienti, tra cui il più utilizzato è *Adam* (Kingma, 2014).

2.2.2 Alcuni problemi nella stima del modello

Overfitting

Nelle reti neurali il numero di parametri aumenta velocemente al crescere della complessità della rete; più formalmente, il numero di parametri è uguale a

$$P \;=\; \sum_{\ell=2}^{L} \big(n_{\ell-1}\, n_\ell \;+\; n_\ell\big) \;=\; \sum_{\ell=2}^{L} n_{\ell-1}\, n_\ell \;+\; \sum_{\ell=2}^{L} n_\ell,$$

Algorithm 2: *Backpropagation* con *Mini-Batch SGD*

Input : Training set $\{(\boldsymbol{x_i}, \boldsymbol{y_i})\}_{i=1}^{N}$,
parametri iniziali $\{\boldsymbol{W^{(l)}}, \boldsymbol{b^{(l)}}\}_{l=2}^{L}$,
funzioni di attivazione $\{\sigma^{(l)}\}$,
learning rate η, *batch size* m, epoche $\boldsymbol{E}$

Output: Parametri aggiornati $\{\boldsymbol{W^{(l)}}, \boldsymbol{b^{(l)}}\}_{l=2}^{L}$

for $e \leftarrow 1$ **to** $\boldsymbol{E}$ **do**

`// 1. Mescola gli esempi per casualità`

Permuta casualmente l'ordine di $\{(\boldsymbol{x^{(i)}}, \boldsymbol{y^{(i)}})\}$;

`// 2. Numero di mini-batch`

$J \leftarrow \lceil N/m \rceil$;

for $j \leftarrow 1$ **to** J **do**

`// 3. Costruisci il` j`-esimo mini-batch`

$i_{\text{start}} \leftarrow (j-1)\,m + 1$;

$i_{\text{end}} \leftarrow \min(j\,m, N)$;

$B_j \leftarrow \{\, (\boldsymbol{x^{(i)}}, \boldsymbol{y^{(i)}} : i = i_{\text{start}}, \ldots, i_{\text{end}}\}$;

`// 4. Inizializzazione`

for $\boldsymbol{l} \leftarrow \boldsymbol{2}$ **to** $\boldsymbol{L}$ **do**

$\Delta \boldsymbol{W^{(l)}} \leftarrow \boldsymbol{0}, \quad \Delta \boldsymbol{b^{(l)}} \leftarrow \boldsymbol{0}$;

`// 5. Calcolo dei gradienti su` B_j

foreach $(\boldsymbol{x^{(i)}}, \boldsymbol{y^{(i)}}) \in B_j$ **do**

$\boldsymbol{a^{(1)}} \leftarrow \boldsymbol{x^{(i)}}$;

for $\boldsymbol{l} \leftarrow \boldsymbol{2}$ **to** $\boldsymbol{L}$ **do**

$\boldsymbol{z^{(l)}} \leftarrow \boldsymbol{W^{(l)}}\,\boldsymbol{a^{(l-1)}} + \boldsymbol{b^{(l)}}$;

$\boldsymbol{a^{(l)}} \leftarrow \sigma^{(l)}(\boldsymbol{z^{(l)}})$;

$\boldsymbol{\delta^{(l)}} \leftarrow \nabla_{\boldsymbol{a}} C^{(\boldsymbol{i})} \odot \sigma^{(l)\prime}(\boldsymbol{z^{(l)}})$;

for $\boldsymbol{l} \leftarrow \boldsymbol{L-1}$ **down to** $\boldsymbol{1}$ **do**

$\boldsymbol{\delta^{(l)}} \leftarrow (\boldsymbol{W^{(l+1)}})^{\mathsf{T}}\,\boldsymbol{\delta^{(l+1)}} \odot \sigma^{(l)\prime}(\boldsymbol{z^{(l)}})$;

for $\boldsymbol{l} \leftarrow \boldsymbol{2}$ **to** $\boldsymbol{L}$ **do**

$\Delta \boldsymbol{W^{(l)}} \mathrel{+}= \boldsymbol{\delta^{(l)}}\,(\boldsymbol{a^{(l-1)}})^{\mathsf{T}}$;

$\Delta \boldsymbol{b^{(l)}} \mathrel{+}= \boldsymbol{\delta^{(l)}}$;

`// 6. Aggiornamento`

for $\boldsymbol{l} \leftarrow \boldsymbol{2}$ **to** $\boldsymbol{L}$ **do**

$$\boldsymbol{W^{(l)}} \longleftarrow \boldsymbol{W^{(l)}} - \frac{\eta}{|B_j|}\,\Delta \boldsymbol{W^{(l)}}, \quad \boldsymbol{b^{(l)}} \longleftarrow \boldsymbol{b^{(l)}} - \frac{\eta}{|B_j|}\,\Delta \boldsymbol{b^{(l)}}$$

pertanto il rischio di *overfitting* è in linea di principio elevato e per evitare che il modello venga ottimizzato esclusivamente sui dati di training, e che non sia in grado di generalizzare, uno degli approcci più usati è noto come *early stopping* (Prechelt, 1998). L'idea alla base di questa procedura è quella di valutare la funzione di costo su un insieme di dati indipendenti da $\mathcal{X}_{train}$ e non utilizzati nell'addestramento, chiamato *insieme di convalida* (*validation set*) $\mathcal{X}_{val}$, e di procedere nella stima fino a quando la funzione di costo su questo insieme comincia a non mostrare miglioramenti sostanziali.

Al termine dell'addestramento, la capacità di generalizzazione della rete viene valutata su un terzo campione, l'insieme di verifica $\mathcal{X}_{test}$.

Lo pseudocodice è schematizzato nell'Algoritmo 3.

Siccome delle proprietà teoriche generali per questo metodo sono difficili da ricavare, la letteratura si è concentrata sull'analisi empirica di casi particolari. Per approfondimenti si rimanda, ad esempio, a Prechelt (2002) o Yao *et al.* (2007).

Un altro metodo utile per prevenire l'*overfitting* è il *weight decay* (Krogh e Hertz, 1991), che applica una penalizzazione L^2 direttamente alla funzione di costo, in modo analogo alla regolarizzazione della regressione *ridge* (si veda, ad esempio, Hastie *et al.*, 2009). La generica funzione di costo diventa:

$$C_p = C + \frac{\lambda}{2N} \sum w_{jk}^2,$$

dove C è la funzione di costo originale non regolarizzata. Le derivate sono $\frac{\partial C_p}{\partial w_{jk}} = \frac{\partial C}{\partial w_{jk}} + \frac{\lambda}{N} w_{jk}$ e $\frac{\partial C_p}{\partial b_j} = \frac{\partial C}{\partial b_j}$, che portano all'aggiornamento delle stime nel *SGD* a:

$$w_{jk} \to \left(1 - \frac{\eta\lambda}{N}\right) w_{jk} - \frac{\eta}{m} \sum_x \frac{\partial L_i}{\partial w_{jk}},$$

$$b_j \to b_j - \frac{\eta}{m} \sum_x \frac{\partial L_i}{\partial b_j}.$$

Di questo metodo si noti che la stima dei pesi w_{2k} diventa direttamente dipendente dalla scala degli input $\boldsymbol{x}$, siccome L_i è funzione di $\boldsymbol{x}$, e che il *bias* non viene penalizzato.

In termini del compromesso descritto nella (1.3), l'effetto del *weight decay* può essere descritto come una riduzione della varianza di stima siccome, penalizzando il valore dei pesi con un termine $\frac{\lambda}{2N} \sum w_{jk}^2$, il modello tende a mantenere i pesi w_{jk} piccoli, a meno che non contribuiscano in maniera elevata alla prestazione del modello. Di conseguenza, il modello è meno sensibile alle fluttuazioni dovute al rumore presente nel campione di training. Di contro, questo equivale a limitare la complessità del modello, siccome appiattire i parametri verso lo zero impedisce di catturare *pattern* più elaborati, aumentando il *bias*.

Un'altra tecnica di regolarizzazione molto comune è la *batch normalization*, che per ogni layer l sostituisce le pre-attivazioni $\boldsymbol{z^{(l)}} = \boldsymbol{W^{(l)}} \boldsymbol{a^{(l-1)}} + \boldsymbol{b^{(l)}}$ con

$$\hat{\boldsymbol{z}}^{(\boldsymbol{l})} = \frac{\mathbf{z}^{(l)} - \boldsymbol{\mu_B}}{\sqrt{\boldsymbol{\sigma_B}^2 + \varepsilon}}, \quad \boldsymbol{a^{(l)}} = \boldsymbol{\gamma} \odot \hat{\boldsymbol{z}}^{(\boldsymbol{l})} + \boldsymbol{\beta},$$

Algorithm 3: *Early stopping*

Input : numero di epoche massime $\boldsymbol{E}$, *patience* $\boldsymbol{p}$, $(\boldsymbol{\mathcal{X}}_{\text{tr}}, \boldsymbol{\mathcal{Y}}_{\text{tr}})$, $(\boldsymbol{\mathcal{X}}_{\text{val}}, \boldsymbol{\mathcal{Y}}_{\text{val}})$

Output: Parametri $\boldsymbol{\theta}^*$ con C_{val} minimo

$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}_0$, $C_{\text{val}}^{\min} \leftarrow +\infty$, $\boldsymbol{c} \leftarrow 0$;
for $\boldsymbol{e} = \mathbf{1}$ **to** $\boldsymbol{E}$ **do**
  aggiorna i parametri con il *backpropagation*;
  calcola $C_{\text{val}}(\boldsymbol{e})$;
  if $C_{\text{val}}(\boldsymbol{e}) < C_{\text{val}}^{\min}$ **then**
    $C_{\text{val}}^{\min} \leftarrow C_{\text{val}}(\boldsymbol{e})$;
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}_{\boldsymbol{e}}$;
    $\boldsymbol{c} \leftarrow \mathbf{0}$;
  else
    $\boldsymbol{c} \leftarrow \boldsymbol{c} + \mathbf{1}$;
    if $\boldsymbol{c} \geq \boldsymbol{p}$ **then**
      break;
return $\boldsymbol{\theta}^*$

dove $\boldsymbol{\mu}_B$ e $\boldsymbol{\sigma}_B^2$ sono rispettivamente il vettore delle medie e delle varianze delle pre-attivazioni calcolate sul *mini-batch* per ciascun neurone, ε è un piccolo scalare introdotto per stabilità numerica, e $\boldsymbol{\gamma}, \boldsymbol{\beta}$ sono vettori di parametri. Questo metodo permette a volte di accelerare la convergenza e introdurre un effetto regolarizzante (si veda Ioffe e Szegedy, 2015).

Come ultimo metodo si cita il *droput*, per cui si rimanda completamente a Srivastava *et al.* (2014).

Sensibilità agli iperparametri

Le reti neurali hanno un grande numero di parametri il cui valore viene scelto arbitrariamente a priori, senza essere stimato; ci si riferirà ad essi con il nome *iperparametri*, che si differenziano in quelli che riguardano l'architettura e specificano quindi il modello $\mathcal{F}$ della rete, come ad esempio il numero di strati L, e quelli che influenzano l'algoritmo di stima, come il *learning rate* η o il parametro di *weight decay* λ. L'uso di metodi di regolarizzazione aumenta la robustezza della rete neurale alle specifiche scelte degli iperparametri. Tuttavia, la prestazione e il tempo di convergenza per la stima di alcune reti, soprattutto quelle complesse, possono variare considerevolmente in relazione agli iperparametri scelti. Ad esempio, è noto come un valore troppo elevato di η non permetta di avvicinarsi con precisione al punto di minimo della funzione di costo C. Di contro, un valore troppo piccolo aumenta il rischio che l'algoritmo converga verso un minimo locale sub-ottimale.

Dunque, è pratica comune provare diverse combinazioni per allenare la rete. L'approccio più comune è quello di effettuare una ricerca a griglia per i principali iperparametri relativi al metodo di stima, sceglierne la combinazione migliore tramite l'uso di un *validation set* e allenare il modello finale su tutti i dati disponibili $\{\mathcal{X}_{train}, \mathcal{X}_{val}\}$. Un approccio automatizzato del genere è sicuramente più riproduci-

bile di uno manuale che procede a tentativi e questo può essere preferibile in alcune situazioni (si veda Bischl *et al.*, 2021; Bergstra e Bengio, 2012). Va fatto notare che tecniche del genere sono in alcuni casi computazionalmente molto onerose dal punto di vista computazionale; infatti, il tempo complessivo di stima aumenta esponenzialmente con il numero di iperparametri da validare. Per questo motivo nel Capitolo 4 si utilizzeranno iperparametri scelti a priori, secondo regole euristiche e pubblicazioni inerenti a metodi specifici che verranno presentati nel Capitolo 3.

Aspetti computazionali

L'elevato numero di parametri da stimare, la complessità del modello e il fatto che non esista una forma chiusa per il problema di stima comportano un elevato costo computazionale. Questo problema è particolarmente marcato in un sottoinsieme delle reti neurali, in cui le architetture presentano un numero particolarmente elevato di strati nascosti, chiamato *deep learning* (si veda Aggarwal, 2023). Il suo trattamento esaustivo non è oggetto di interesse principale dell'elaborato e viene semplicemente citato per completezza. In maniera analoga si menziona che un fattore importante per mitigare il problema del tempo computazionale è l'utilizzo della *Graphics Processing Unit* (*GPU*), un circuito elettronico progettato per eseguire calcoli in parallelo su larga scala, con numerosi *core* ottimizzati per elaborare grafica e dati complessi.

Interpretazione

Le reti neurali sono spesso considerate scatole nere (*black box*), con questo termine ci si riferisce al fatto che la loro alta complessità rende a volte impossibile tracciare in che modo gli input vengano trasformati per giungere agli *output* finali. Ciò rende l'utilizzo del modello vincolato ad un'ottica maggiormente predittiva, sebbene in alcuni casi esistano dei metodi per fornire un'interpretazione alla rete e ai suoi strati nascosti (si veda Zhang *et al.*, 2021).

2.3 Approssimatori universali

La principale motivazione teorica per l'utilizzo delle reti neurali risiede nel fatto che, sotto opportune ipotesi, sono in grado di approssimare con una precisione arbitraria una funzione obiettivo. La formalizzazione del concetto si può trovare nell'articolo di Hornik *et al.* (1989), dove viene dimostrato il *teorema di approssimazione universale*, enunciato di seguito.

Teorema 2.3.1 (Teorema di Approssimazione Universale) *Siano $K \subset \mathbb{R}^D$ un insieme compatto e $\sigma : \mathbb{R} \to \mathbb{R}$ una funzione di attivazione continua, limitata e non costante. Allora, per ogni funzione continua $y(\boldsymbol{x}) : K \to \mathbb{R}$, e per ogni $\varepsilon > 0$, esiste una rete neurale feedforward con un singolo strato nascosto, un numero finito di neuroni e funzione di attivazione σ, tale che la funzione $\hat{y}(\boldsymbol{x}; \boldsymbol{\theta})$ realizzata dalla rete soddisfa:*

$$\sup_{x \in K} |y(\boldsymbol{x}) - \hat{y}(\boldsymbol{x}; \boldsymbol{\theta})| < \varepsilon,$$

Il teorema 2.3.1 garantisce che, per una rete *feedforward* con un singolo strato nascosto, esista un parametro $\boldsymbol{\theta}$ in grado di riprodurre arbitrariamente bene qualsiasi funzione continua definita su un compatto $K \subset \mathbb{R}^D$. Ciò fornisce la motivazione principale per l'uso delle reti neurali: non vengono imposti vincoli restrittivi sulla forma della funzione da approssimare. Esistono estensioni del teorema più generali, che superano la limitazione su K, ad esempio (Hornik, 1991):

Teorema 2.3.2 (Approssimazione in $L^p(\mathbb{R}^D)$) *Sia $\sigma : \mathbb{R} \to \mathbb{R}$ una funzione di attivazione continua, limitata e non costante. Allora, per ogni funzione $y \in L^p(\mathbb{R}^D)$ con $1 \leq p < \infty$ e per ogni $\varepsilon > 0$, esiste una rete neurale feedforward con un singolo strato nascosto, un numero finito di neuroni e funzione di attivazione σ, tale che la funzione $\hat{y}$ realizzata dalla rete soddisfa:*

$$\|y(\boldsymbol{x}) - \hat{y}(\boldsymbol{x}; \boldsymbol{\theta})\|_{L^p} < \varepsilon.$$

Dove gli spazi $L^p(\mathbb{R}^D)$, per il parametro p ($1 \leq p < \infty$) che definisce la norma considerata, sono insiemi di funzioni misurabili $y : \mathbb{R}^D \to \mathbb{R}$ che soddisfano la seguente condizione di integrabilità:

$$\|y(\boldsymbol{x})\|_{L^p} = \left(\int_{\mathbb{R}^D} |y(\boldsymbol{x})|^p \, d\boldsymbol{x} \right)^{1/p} < \infty.$$

Per ulteriori varianti del teorema 2.3.1, e per un approfondimento sulla teoria dell'approssimazione delle reti neurali si rimanda, ad esempio, a Barron (1993), Pinkus (1999) o al lavoro più recente di Augustine (2024).

Dai teoremi si evince che, se $y(\boldsymbol{x})$ fosse nota, non servirebbe avere più strati nascosti: già con uno strato si ottiene un'approssimazione ottimale. Tuttavia, questo non comporta che un solo strato sia la migliore alternativa in termini di velocità di apprendimento, semplicità di implementazione o generalizzazione su dati nuovi.

Per illustrare le proprietà di una rete in un caso reale di stima, si consideri la situazione in cui i dati in *input* soddisfano, per un certo valore positivo r, la condizione

$$\boldsymbol{x} \in \left\{ \boldsymbol{x} \in \mathbb{R}^D : \|\boldsymbol{x}\| \leq r \right\}.$$

Questo consente di esprimere in modo semplice i limiti dell'errore per la stima $\hat{y}(\boldsymbol{x})$ prodotta da una rete con un singolo strato nascosto, coerente con i teoremi enunciati. In tal proposito si definisce il primo momento assoluto C_y:

- Sia $y : \mathbb{R}^D \to \mathbb{R}$ una funzione continua. Definiamo la sua trasformata di Fourier

  $$\tilde{y}(\omega) = \int_{\mathbb{R}^D} y(\boldsymbol{x}) \, e^{-j\,\omega^T \boldsymbol{x}} \, d\boldsymbol{x}, \qquad \omega \in \mathbb{R}^D.$$

  Il *primo momento assoluto* di $\tilde{y}$ è

  $$C_y = \int_{\mathbb{R}^D} \|\omega\| \, \left|\tilde{y}(\omega)\right| \, d\omega.$$

  Intuitivamente, C_y quantifica la regolarità di $y(x)$: se $y(x)$ è liscia, C_y resta piccolo.

Barron (1993) ha dimostrato che, se y è continua e $C_y < \infty$, allora esiste una rete con un solo strato nascosto, e con n_1 neuroni in tale strato che soddisfa

$$C \;\leq\; C_\psi \, \frac{C_y^2}{n_1}, \quad \text{dove} \quad C_\psi = \big(2\, r\, C_y\big)^2,$$

con C funzione di costo quadratica sul *training set*, definita nella 1.9.

Invece per la stessa rete

$$\mathrm{MSE}\big(\hat{y}(\boldsymbol{x})\big) \;\leq\; \mathcal{O}\left(\frac{C_y^2}{n_1}\right) \;+\; \mathcal{O}\left(\frac{D\, n_1}{N} \log N\right). \tag{2.6}$$

Dalla 2.6 emerge il compromesso *bias*-varianza di cui si è discusso nella Sezione 1.3: si nota come $\dfrac{C_y^2}{n_1}$ diminuisca all'aumentare del numero di parametri (sta diminuendo il *bias*), mentre $\dfrac{D\, n_1}{N}$ aumenta. Il numero ottimale di neuroni dipende dalla specifica funzione da approssimare.

Questo risultato evidenzia un aspetto teorico fondamentale: nonostante il teorema di approssimazione universale garantisca l'esistenza di una rete con un solo strato nascosto, capace di approssimare qualunque funzione che rispetti le condizioni del teorema 2.3.2, nella pratica non è possibile aumentare indefinitamente il numero di neuroni n_2 senza compromettere la capacità di generalizzazione. L'aumento della complessità della rete porta a un miglioramento dell'approssimazione solo se il numero di osservazioni N è sufficientemente grande, altrimenti prevale la componente $\dfrac{C_y^2}{n_2}$. Anche per questo motivo, nella maggior parte delle applicazioni si usano più strati nascosti, in grado di ottenere un miglior compromesso *bias*-varianza (si veda Haykin, 2009, Capitolo 4).

I risultati di Barron (1993) mostrano anche un'altra proprietà delle reti neurali, ovvero che, rispetto ad altri metodi di approssimazione classici, le reti neurali sono, in alcuni casi, relativamente robuste alla maledizione della dimensionalità. Infatti, per molte famiglie di funzioni lisce tradizionali (ad esempio polinomi o trigonometriche), il tasso di convergenza dell'errore totale peggiora rapidamente con l'aumento della dimensione D, ed è proporzionale a : $N^{\frac{-2s}{2s+D}}$, dove s rappresenta il numero di derivate continue della funzione $y(\boldsymbol{x})$ (per il nostro scopo lo si veda come un'altra misura della liscezza di $y(\boldsymbol{x})$. Al contrario, per reti neurali con un solo strato nascosto, scegliendo un opportuno n_2, l'errore può decrescere proporzionalmente a $C_y\sqrt{\frac{D \log N}{N}}$, un tasso di convergenza decisamente migliore. Per quest'ultima osservazione va fatto notare che la dimensionalità dell'input D influisce indirettamente anche su C_y, siccome è più probabile che una funzione sia complessa se ha più dimensioni e che i risultati sono vincolati allo specifico caso in esame. Nella pratica, la scelta del metodo ottimale è legata alla funzione da approssimare.

Capitolo 3

Reti neurali per la stima della densità

3.1 Introduzione

Uno degli ambiti applicativi in cui sta prendendo più piede, negli anni recenti, l'uso delle reti neurali, è quello della stima della funzione di probabilità o densità di una variabile casuale. A differenza dei problemi di regressione o classificazione, la stima del modello generatore dei dati rientra nell'ambito dei problemi di apprendimento non supervisionato, in genere di più difficile soluzione perché la funzione obiettivo non è nota né osservabile, neanche su un campione di osservazioni.

Indicato con $\boldsymbol{X} = (X_1, \ldots, X_D)^T$ un vettore casuale con supporto $\mathcal{S} \subseteq \mathbb{R}^D$, con $p(\boldsymbol{x}) = p(x_1, \ldots, x_D)$ la sua funzione di probabilità o densità e con $\mathcal{X} = \{\boldsymbol{x^{(i)}}\}_{i=1}^N$ un campione di realizzazioni *i.i.d.* da $\boldsymbol{X}$, i metodi di stima della densità basati sulle reti neurali sono tipicamente accomunati dal fatto di addestrare la rete che ottimizza, quale funzione di perdita, l'opposto della log-verosimiglianza dei dati

$$C(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} - \log p\big(\boldsymbol{x^{(i)}}\big), \tag{3.1}$$

che rende legittimo interpretare l'*output* della rete come una stima di probabilità o densità. Così come per il caso supervisionato, anche nel caso non supervisionato la letteratura è molto ricca di procedure designate per questo scopo.

Una tassonomia non esaustiva, ma utile al fine di dare una panoramica dei diversi metodi, distingue principalmente tre classi di modelli per la stima della densità basati sulle reti neurali:

1. Modelli autoregressivi: modellano la distribuzione mediante sequenza di distribuzioni condizionali. Definito un ordinamento $X_1, \ldots, X_D$ per le variabili del vettore casuale $\boldsymbol{X}$, ogni densità può essere scomposta nel modo

$$p(\boldsymbol{x}) = p(x_1) \prod_{d=2}^{D} p(x_d | x_1, \ldots, x_{d-1}), \tag{3.2}$$

   Ogni nodo dello strato di *output* è interpretato come la stima per la densità condizionata $p(x_d | \boldsymbol{x_{1:d-1}})$ $(\forall d = 1, \ldots, D)$ e successivamente si ottiene la stima per la densità $\hat{p}(\boldsymbol{x})$ come prodotto delle condizionate.

2. Modelli basati su variabili latenti: introducono nuove variabili non osservabili per descrivere la struttura sottostante ai dati. Tra questi modelli si distinguono i *Flussi di normalizzazione*, dove si assume che $\boldsymbol{X}$ sia una trasformata complessa di una struttura latente più semplice, e i modelli generativi basati su variabili latenti.

3. Modelli di approssimazione della verosimiglianza: approssimano la verosimiglianza dei dati minimizzando criteri di divergenza tra funzioni, quali quella di Kullback-Leibler, o mediante funzioni surrogate.

I modelli del terzo gruppo solitamente non producono una stima esplicita di $p(\boldsymbol{x})$, ma vengono invece usati per generare nuovi dati simili a quelli originali. Per questa ragione, nel seguito ci si concentrerà principalmente sui metodi appartenenti alle prime due classi, e si rimanda a Zammit-Mangion *et al.* (2024) per un approfondimento sui metodi della terza.

3.2 Modelli autoregressivi

3.2.1 Neural Autoregressive Distribution Estimation Model

Un esempio di approccio autoregressivo per la stima della densità di un vettore $\boldsymbol{X} \in \mathbb{R}^D$ è rappresentato dal modello *Real Neural Autoregressive Distribution Estimation Model* (RNADE Uria *et al.*, 2013), che generalizza al caso continuo il modello NADE (Larochelle e Murray, 2011) designato per dati binari.

Per semplicità espositiva, si consideri innanzitutti il caso in cui $\boldsymbol{X} \in \{0,1\}^D$. In NADE, ogni densità condizionata è modellata da una rete neurale *feed-forward* con un singolo strato nascosto. L'insieme di queste reti può essere visto come un'unica rete simile a quella nella Figura 2.1, con la differenza che i D strati nascosti sono paralleli tra loro invece che in successione, e ognuno di essi prende in *input* un sottoinsieme degli elementi di $\boldsymbol{x}$ e restituisce come *output* una probabilità condizionata. Lo schema per il modello è riportato nella Figura 3.1.

Analogamente al caso supervisionato, lo strato in *input* è composto da D neuroni, ciascuno per ogni x_d $d = 1, \dots, D$ e, coerentemente con la notazione usata nel capitolo precedente, si pone $\boldsymbol{a_0} = (x_1, \dots, x_D)^T$.

Per ogni d nello strato nascosto si determina un vettore latente con n_1 neuroni

$$\boldsymbol{z_d^{(1)}} = \boldsymbol{W^{(1)}} \boldsymbol{a_{(d-1)}^{(0)}} + \boldsymbol{b^{(1)}}, \tag{3.3}$$

dove, $\boldsymbol{W^{(1)}} \in \mathbb{R}^{n_1 \times D}$ e $\boldsymbol{a_{(d-1)}^{(0)}}$ è il vettore $\boldsymbol{a^{(0)}}$ i cui elementi di posizione successiva a $d-1$ sono posti pari a 0, e $\boldsymbol{b^{(1)}} \in \mathbb{R}^{n_1}$.

Una funzione di attivazione σ, non lineare, successivamente mappa i vettori $\boldsymbol{z_d^{(1)}}$ in $\boldsymbol{a_d^{(1)}} = \sigma(\boldsymbol{z_d^{(1)}})$.

Nello strato di *output*, infine, composto da D neuroni, una trasformata lineare determina i logit

$$z_d^{(2)} = \boldsymbol{W_d^{(2)}} \boldsymbol{a_d^{(1)}} + b_d^{(2)} \tag{3.4}$$

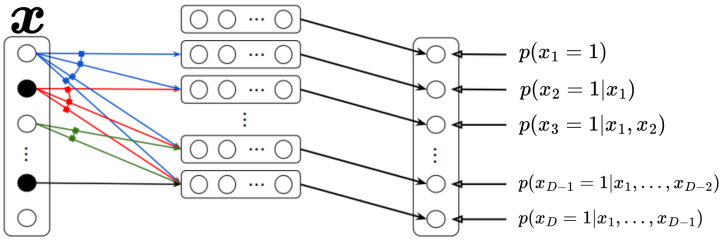

Figura 3.1: Schema per il modello NADE per dati binari. La condivisione dei parametri è rappresentata con il collegamento e i colori in comune tra le frecce. Immagine presa da Uria *et al.* (2016) con qualche lieve modifica.

dove $\boldsymbol{W_d^{(2)}} \in \mathbb{R}^D$ è un vettore riga di pesi specifico per ogni condizionata. Ad ogni $z_d^{(2)}$ viene poi applicata un'opportuna funzione di attivazione per ricondursi a valide probabilità condizionate:

$$p(x_d = 1|x_1, \ldots, x_{d-1}) = \sigma(z_d^{(2)}) = \frac{1}{1 + e^{-z_d^{(2)}}}.$$

La densità condizionata per la dimensione d viene modellata solamente dalle variabili precedenti nell'ordinamento, se così non fosse l'*output* corrisponderebbe a $p(x_d = 1|\boldsymbol{x})$ e la scomposizione nella 3.2 non sarebbe verificata.

La particolarità di NADE è che gli elementi di $\boldsymbol{W^{(1)}}$ e il bias $\boldsymbol{b^{(1)}}$ sono condivisi da ciascun vettore nel livello nascosto $\boldsymbol{z_d^{(1)}}$: ad esempio, la prima colonna di $\boldsymbol{W^{(1)}}$ modella lo strato nascosto per ogni probabilità condizionata $d = 2, \ldots, D$. Questo implica che NADE ha un numero di parametri proporzionale a $n_1 D$ anziché a $n_1 D^2$, che sarebbero invece richiesti se i vettori per ogni condizionata fossero indipendenti. La diminuzione del numero dei parametri può essere vista nell'ottica del compromesso *bias*-varianza, infatti stiamo diminuendo la flessibilità del modello per ridurre la varianza di stima. Un altro vantaggio è la diminuzione del tempo computazionale, siccome questo schema di condivisione di parametri permette il calcolo ricorsivo delle quantità $\boldsymbol{a_d^{(1)}}$ rispetto a quelle delle dimensioni precedenti:

$$\begin{aligned} \boldsymbol{a_d^{(1)}} &= \sigma(\boldsymbol{W^{(1)}} \boldsymbol{a_{(d-1)}^{(0)}} + \boldsymbol{b^{(1)}}), \\ &\text{con } \boldsymbol{W^{(1)}} \boldsymbol{a_{(d-1)}^{(0)}} + \boldsymbol{b^{(1)}} = \boldsymbol{W^{(1)}}_{\cdot, \boldsymbol{d-1}} \, x_{d-1} + \sigma^{-1}(\boldsymbol{a_{d-1}^{(1)}}), \end{aligned}$$

dove $\boldsymbol{W^{(1)}}_{\cdot, \boldsymbol{d-1}}$ indica solamente la colonna $d-1$ della matrice $\boldsymbol{W^{(1)}}$.

Nel caso di variabili binarie, la modellazione di ogni probabilità condizionata non è differente da un tipico problema di regressione per dati binari: date le variabili esplicative $x_1, \ldots, x_{d-1}$, vogliamo stimare la probabilità che la risposta x_d assuma valore 1.

Il modello RNADE, ovvero l'estensione del modello per vettori casuali definiti su $\mathbb{R}^D$, si ottiene modellando ogni densità condizionata come una mistura (1.1) di gaussiane unidimensionali

$$p(x_d \mid x_1, \dots, x_{d-1}) = \sum_{c=1}^{C} \pi_{d,c} \mathcal{N}(x_d; \mu_{d,c}, \sigma_{d,c}^2).$$

La trasformazione effettuata nello strato nascosto è analoga a quella definita in NADE, mentre nello strato di *output* vengono calcolate

- $z_{d,c}^{(2,\pi)} = \boldsymbol{W}_{\boldsymbol{d,c}}^{\boldsymbol{(2,\pi)}} \boldsymbol{a}_{\boldsymbol{d}}^{\boldsymbol{(1)}} + b_{d,c}^{(2,\pi)}$;
- $z_{d,c}^{(2,\mu)} = \boldsymbol{W}_{\boldsymbol{d,c}}^{\boldsymbol{(2,\mu)}} \boldsymbol{a}_{\boldsymbol{d}}^{\boldsymbol{(1)}} + b_{d,c}^{(2,\mu)}$;
- $z_{d,c}^{(2,\sigma)} = \boldsymbol{W}_{\boldsymbol{d,c}}^{\boldsymbol{(2,\sigma)}} \boldsymbol{a}_{\boldsymbol{d}}^{\boldsymbol{(1)}} + b_{d,c}^{(2,\sigma)}$,

che sono l'analogo della (3.4), ma per ogni parametro di ogni componente della mistura. A queste quantità vengono poi applicate opportune funzioni di attivazione per ottenere dei parametri validi per la mistura:

$$\pi_{d,c} = \frac{\exp\left\{z_{d,c}^{(2,\pi)}\right\}}{\sum_{c=1}^{C} \exp\left\{z_{d,c}^{(2,\pi)}\right\}}; \quad \mu_{d,c} = z_{d,c}^{(2,\mu)}; \quad \sigma_{d,c} = \exp\left\{z_{d,c}^{(2,\sigma)}\right\}.$$

Sia per NADE che per RNADE è necessario definire a priori un ordinamento delle variabili. Questa scelta arbitraria influenza la variabilità delle stime. In Uria *et al.* (2014) viene definita una procedura che limita parzialmente il problema, introducendo un'architettura che stima in parallelo più modelli RNADE con ordinamenti differenti per poi aggregare i risultati. Il modello è chiamato *DeepNADE* e una sua descrizione precisa può essere trovata in Uria *et al.* (2016), paragrafo 4.

3.2.2 Masked Autoencoder Distribution Estimator

Il *Masked Autoencoder Distribution Estimator* (MADE) (Germain *et al.*, 2015) segue un approccio simile a quello di NADE per dati binari, ma l'unico aspetto rilevante che lo differenzia da esso è che si basa su uno specifico tipo di rete chiamato *autoencoder* (si veda Goodfellow *et al.*, 2016, Capitolo 14).

L'obiettivo di un *autoencoder* è apprendere, tramite una rete neurale *feed-forward*, una rappresentazione latente $\boldsymbol{a}^{(1)} \in \mathbb{R}^{D'<D}$ di $\boldsymbol{x}$, da cui sia possibile ricostruire $\boldsymbol{x}$ con la minima perdita di informazione possibile. Il funzionamento della rete è analogo a quello descritto precedentemente, con la particolarità che lo strato nascosto dove risiede la rappresentazione latente di $\boldsymbol{x}$ è di dimensione molto inferiore a D.

Tuttavia, per usare gli *autoencoder* per stimare la densità dei dati è necessario apportare qualche modifica alla loro struttura, per non violare la scomposizione nell'Equazione 3.2. L'idea è di permettere a ciascun nodo di *output* $\hat{p}(x_d|x_1, \dots, x_{d-1})$ di dipendere soltanto dalle componenti precedenti $x_1, \dots, x_{d-1}$, escludendo ogni percorso computazionale che cominci dagli *input* successivi $x_d, x_{d+1}, \dots, x_D$, similmente a come veniva fatto in NADE ponendo gli ultimi valori di $\boldsymbol{a}_{\boldsymbol{d-1}}^{\boldsymbol{0}}$ nella (3.3) uguali a 0.

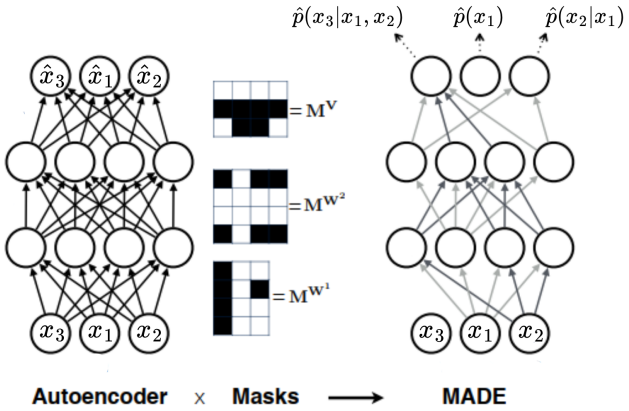

Figura 3.2: A sinistra: *Autoencoder* convenzionale. A destra: MADE. La rete ha la stessa struttura dell'*autoencoder*, ma un insieme di connessioni viene rimosso in modo tale che ogni unità di *input* sia prevista soltanto da quelle precedenti. Immagine presa da Germain *et al.* (2015) con qualche lieve modifica.

In maniera simile, in MADE vengono definite delle matrici a valori binari che vengono moltiplicate elemento per elemento alle matrici dei pesi del modello, rendendo nulli i contributi delle dimensioni precedenti alla probabilità condizionata considerata.

Se gli *autoencoder* utilizzati hanno più strati nascosti, sono necessarie più matrici e il modello viene chiamato *Deep MADE*, di cui è fornito un esempio dell'architettura nella Figura 3.2.

Come in NADE, questo modello è vincolato a uno specifico ordinamento delle variabili, definito dalle matrici binarie. In letteratura viene proposto un approccio che addestra la rete su più ordinamenti rimanendo comunque trattabile computazionalmente, mitigando il problema. Per un approfondimento si rimanda alle sezioni 4.2 e 4.3 di Germain *et al.* (2015).

Esistono altre architetture in cui le relazioni tra variabili vengono modellate da più modelli MADE, combinati con una variante autoregressiva dei flussi di normalizzazione. Tra questi si citano l'*Inverse Autoregressive Flow* (IAF) (Kingma *et al.*, 2016) e il *Masked Autoregressive Flow* (MAF) (Papamakarios *et al.*, 2017).

3.3 Modelli basati su variabili latenti

3.3.1 Flussi di normalizzazione

I modelli basati sui flussi di normalizzazione assumono che la variabile $\boldsymbol{X} \in \mathbb{R}^D$ sia la trasformazione di una variabile latente $\boldsymbol{Z} \in \mathbb{R}^D$, la cui funzione di densità $p_{\boldsymbol{Z}}(\boldsymbol{z})$ è nota e definita a priori. Una scelta comune per $p_{\boldsymbol{Z}}$ è la densità di una gaussiana sferica.

Questa classe di metodi sfrutta il fatto che, posto $x = g(\boldsymbol{z})$, con $p_{\boldsymbol{Z}}$ funzione di densità nota, e g funzione biettiva tale che $\boldsymbol{z} = g^{-1}(\boldsymbol{x}) = h(\boldsymbol{x})$, si ottiene facilmente

$$p(\boldsymbol{x}) = p_{\boldsymbol{Z}}\big(h(\boldsymbol{x})\big) \left|\det\left[\frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}^T}\right]\right|, \tag{3.5}$$

La trasformazione h viene approssimata da una rete neurale o una successione di reti, dove $\boldsymbol{x}$ è l'*input* iniziale della prima rete e $\boldsymbol{z}$ è l'*output* finale dell'ultima rete. Una volta approssimata h viene calcolata $p(\boldsymbol{x})$ tramite la 3.5.

Una delle più importanti architetture che usa i flussi di normalizzazione è RealNVP che, sebbene sia stato pensato perlopiù per la generazione sintetica di immagini, ha permesso di ottenere risultati di rilievo anche in ambiti più generali. Il modello prende il nome dalla classe di trasformazioni su cui si basa, ovvero le *Real-valued Non-Volume Preserving transformations*.

Per calcolare $\hat{p}(\boldsymbol{x})$ è necessario calcolare il determinante della matrice jacobiana dell'approssimazione di h, che in caso di funzioni ad alta dimensione ha in generale un alto costo computazionale. Per questo motivo, RealNVP modella h come una composizione di particolari funzioni biettive, che vengono applicate in successione e gli strati in cui vengono eseguite sono chiamati *affine coupling layers*. In questo modo si ottiene una trasformazione biunivoca e flessibile con Jacobiano velocemente calcolabile.

Più precisamente, ogni *affine coupling layer* è una funzione che trasforma il vettore $\boldsymbol{x}$ nella variabile $\boldsymbol{y}$, secondo le equazioni

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \\ y_{d+1} \\ \vdots \\ y_D \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ x_{d+1}\, \exp\big(s_{d+1}(x_1,\dots,x_d)\big) + t_{d+1}(x_1,\dots,x_d) \\ \vdots \\ x_D\, \exp\big(s_D(x_1,\dots,x_d)\big) + t_D(x_1,\dots,x_d) \end{pmatrix}, \tag{3.6}$$

dove, le *funzioni di scala* s_p e quelle di *traslazione* t_p $(p = d+1,\dots,D)$ sono definite da due reti neurali t e s con *input* in $\mathbb{R}^d$ e *output* in $\mathbb{R}^{D-d}$.

Di conseguenza, la matrice jacobiana di un *affine coupling layer* risulta essere

$$\frac{\boldsymbol{\partial y}}{\partial \boldsymbol{x}^T} = \begin{bmatrix} I_d & 0 \\ * & \operatorname{diag}\big(\exp[s(\boldsymbol{x_{(d)}})]\big) \end{bmatrix},$$

dove $\boldsymbol{x_{(d)}}$ è il sottovettore contenente solo i primi d valori di $\boldsymbol{x}$. La matrice jacobiana risulta essere triangolare inferiore, quindi ha determinante pari al prodotto degli elementi lungo la diagonale, ovvero $\prod_{p=1}^{d} \exp\big(s(x_p)\big) = \exp\big(\sum_{p=1}^{d} s(x_p)\big)$, calcolabile velocemente.

Si noti che, siccome il calcolo del determinante non comporta il calcolo delle derivate parziali di s o di t, queste possono essere arbitrariamente complesse. Per questo motivo usare reti neurali come s e t non rende il problema intrattabile.

Nelle Equazioni 3.6, per suddividere l'*input* in una parte che rimane invariata e una che viene trasformata, si implementa un vettore binario $\boldsymbol{m} \in \{0,1\}^D$ che maschera alcuni valori di $\boldsymbol{x}$ e conseguentemente calcolando l'*output* come

$$\boldsymbol{y} \;=\; \boldsymbol{m} \odot \boldsymbol{x} \;+\; (\boldsymbol{1} - \boldsymbol{m}) \odot \big(\boldsymbol{x} \odot \exp[s(\boldsymbol{m} \odot \boldsymbol{x})] + t(\boldsymbol{m} \odot \boldsymbol{x})\big).$$

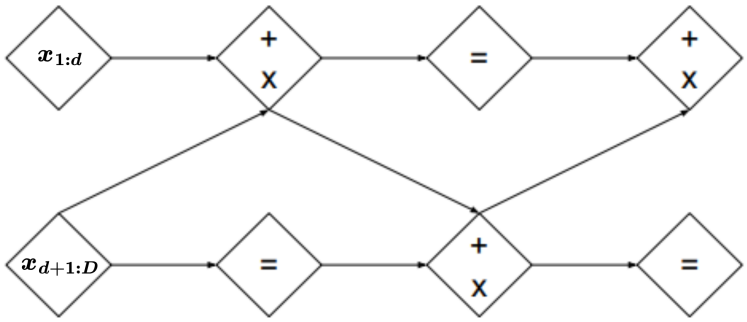

Figura 3.3: Esempio di schema di una successione di trasformazioni in una architettura RealNVP. Una possibilità è alternare le maschera $\boldsymbol{m}$ e quella $\mathbf{1} - \boldsymbol{m}$. Le variabili che non cambiano in una trasformazione vengono modificate nella successiva. Immagine presa da Dinh *et al.* (2016) con qualche lieve modifica.

Questa trasformazione lascia invariate alcune componenti, rendendo impossibile una mappatura da $\boldsymbol{x}$ a $\boldsymbol{z}$. Per questo motivo, si applicano diversi *coupling layers* in successione, modificando i valori assunti da $\boldsymbol{m}$ in ciascuno di essi, in modo che le componenti non modificate da una trasformazione vengano aggiornate da quella successiva (si veda la Figura 3.3). Il passaggio chiave di questa procedura consiste nel fatto che, se $\boldsymbol{y^{(a)}}$ e $\boldsymbol{y^{(b)}}$ sono due trasformazioni effettuate nel *coupling layers*, allora

$$\det\left[\frac{\partial(\boldsymbol{y^{(b)}}\circ\boldsymbol{y^{(a)}})(\boldsymbol{x})}{\partial\boldsymbol{x}^T}\right] = \det\left[\frac{\partial\boldsymbol{y^{(a)}}(\boldsymbol{x})}{\partial\boldsymbol{x}^T}\right] \det\left[\frac{\partial\boldsymbol{y^{(b)}}(\boldsymbol{y^{(a)}}(\boldsymbol{x}))}{\partial(\boldsymbol{y^{(a)}}(\boldsymbol{x}))^T}\right],$$

il problema rimane computazionalmente trattabile e i parametri delle reti che minimizzano la (3.1) possono essere trovati con gli algoritmi definiti nelle Sezioni 2.2.1 e 2.2.2.

3.3.2 Altri modelli e approfondimenti

Restricted Boltzmann Machines

Uno dei primi modelli ideati per dati binari, da cui hanno preso spunto i successivi, è la *macchina di Boltzmann* (*Boltzmann Machine*, BM), un *modello grafico probabilistico* (*Probabilistic Graphical Model*, PGM) (si veda Koller e Friedman, 2009) indiretto, basato sulle reti neurali. Ha avuto molto successo in letteratura un caso particolare di questi modelli, ovvero le *macchine di Boltzmann ristrette* (*Restricted Boltzmann Machines*, RBM), di cui una descrizione approfondita può essere trovata in Fischer e Igel (2012).

Un RBM è formato da due strati, uno *strato visibile* $\boldsymbol{x} \in \{0,1\}^D$, composto da D nodi che corrispondono alle variabili contenute in $\boldsymbol{X}$ e uno strato nascosto $\boldsymbol{z} \in \{0,1\}^F$, composto da F nodi associati al vettore latente $\boldsymbol{Z}$, che, al contrario dei flussi di normalizzazione, può avere dimensione inferiore a $\boldsymbol{X}$.

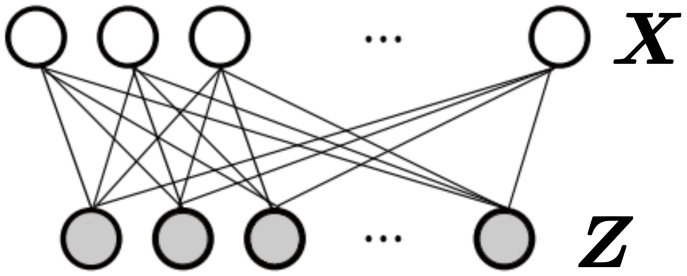

Figura 3.4: Struttura di una *restricted Boltzmann machine*, le unità nello stesso strato non sono connesse, in questo modo sono indipendenti condizionatamente ai nodi nell'altro strato.

In maniera differente da molti modelli grafici, ma più similmente alle reti neurali classiche, non sono presenti connessioni all'interno dello stesso strato, ma solo tra nodi in strati diversi (si veda la Figura 3.4).

Le connessioni tra unità visibili e nascoste sono descritte dalla matrice di pesi $\boldsymbol{W}$ e i vettori dei bias $\boldsymbol{b}$ e $\boldsymbol{c}$. Ogni elemento w_{dj} della matrice regola l'interazione tra la variabile visibile x_d e la variabile latente z_j, tramite la relazione $p(z_j = 1 \mid \boldsymbol{x}) = \sigma\Big(\sum_{d=1}^{D} w_{dj} x_d + c_j\Big)$ e $p(x_d = 1 \mid \boldsymbol{z}) = \sigma\Big(\sum_{j=1}^{F} w_{dj} z_j + b_d\Big)$. Senza entrare troppo nel dettaglio, lo stato della rete è definito dalla misura di *energia* della rete

$$E(\boldsymbol{x}, \boldsymbol{z}) = -\sum_{d=1}^{D}\sum_{j=1}^{F} x_d w_{dj} z_j - \sum_{d=1}^{D} b_d x_d - \sum_{j=1}^{F} c_j z_j.$$

che a sua volta permette di calcolare la funzione di probabilità congiunta

$$p(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{Z} e^{-E(\boldsymbol{x},\boldsymbol{z})}, \quad Z = \sum_{x,z} e^{-E(\boldsymbol{x},\boldsymbol{z})}$$

e la marginale

$$p(\boldsymbol{x}) = \frac{1}{Z} \sum_{z} e^{-E(\boldsymbol{x},\boldsymbol{z})}.$$

Una delle problematiche principali di questo modello è che richiede il calcolo della costante di normalizzazione Z, che diventa computazionalmente intrattabile per D troppo elevati.

Roundtrip

Roundtrip è un'architettura che si basa sulle *reti generative avversarie* (*Generative Adversial Network*, GAN) (si veda Goodfellow *et al.*, 2020) per il calcolo della funzione di densità. Similmente agli altri modelli, ipotizza l'esistenza di un vettore latente $\boldsymbol{Z} \in \mathbb{R}^F$, con F solitamente inferiore a D.

Un singolo GAN, in generale, si basa su due reti neurali contrapposte: il *generatore* $G(\boldsymbol{z}; \boldsymbol{\theta_G})$, che prende in input un vettore di rumore $\boldsymbol{z} \sim p_{\boldsymbol{z}}$ (con densità scelta a priori, solitamente gaussiana) e produce dei dati simulati $\tilde{\boldsymbol{x}} = G(\boldsymbol{z})$, con l'obiettivo di imitare i dati reali; invece, una seconda rete chiamata *discriminatore* $D(\boldsymbol{x}; \boldsymbol{\theta_D})$ prende in input un dato, reale o simulato da G, e restituisce una stima $D(\boldsymbol{x}) \in [0, 1]$ per la probabilità che $\boldsymbol{x}$ sia una vera osservazione del campione. Durante la fase di addestramento, G e D competono con due scopi opposti: G è allenata per generare osservazioni sempre più realistiche, mentre il discriminatore cerca di distinguere tra dati reali e generati.

Roundtrip approssima una mappa tra lo spazio latente e quello delle osservazioni, assumendo che $\boldsymbol{x} = \tilde{\boldsymbol{x}} + \boldsymbol{\epsilon}$, dove l'errore ha generico elemento $\epsilon_d \sim \mathcal{N}(0, \sigma)$. La trasformazione da $\boldsymbol{z} \in \mathbb{R}^F$ a $\boldsymbol{x} \in \mathbb{R}^D$ viene indicata con $G(\boldsymbol{z}) = \tilde{\boldsymbol{x}}$, mentre quella da $\boldsymbol{x}$ a $\boldsymbol{z}$ viene indicata con $H(\boldsymbol{x}) = \tilde{\boldsymbol{z}}$. Queste due trasformazioni vengono approssimate da due GAN distinti, come illustrato nella Figura 3.5. In maniera analoga si definiscono i discriminatori $D_{\boldsymbol{x}}$ e $D_{\boldsymbol{z}}$ per i due GAN.

I parametri dei generatori e dei discriminatori vengono stimati nella stessa fase di addestramento e le funzioni di costo per le reti sono i corrispondenti campionari di

$$\begin{cases} \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}\big[D_{\boldsymbol{x}}(G(\boldsymbol{z})) - 1\big]^2 & \text{per } G(\boldsymbol{z}), \\ \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}}\big[D_{\boldsymbol{x}}(\boldsymbol{x}) - 1\big]^2 + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}\big[D_{\boldsymbol{x}}(G(\boldsymbol{z}))\big]^2 & \text{per } D_{\boldsymbol{x}}, \\ \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}}\big[D_{\boldsymbol{z}}(H(\boldsymbol{x})) - 1\big]^2 & \text{per } H(\boldsymbol{x}), \\ \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}\big[D_{\boldsymbol{z}}(\boldsymbol{z}) - 1\big]^2 + \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}}\big[D_{\boldsymbol{z}}(H(\boldsymbol{x}))\big]^2 & \text{per } D_{\boldsymbol{z}}. \end{cases}$$

e della *roundtrip loss*, che è definita come:

$$\alpha\, \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}} \big\| \boldsymbol{x} - G(H(\boldsymbol{x})) \big\|^2 + \beta\, \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}} \big\| \boldsymbol{z} - H(G(\boldsymbol{z})) \big\|^2,$$

con α e β costanti scelte a priori. Questo termine penalizza la discrepanza tra le trasformazioni tra i due diversi GAN.

Una volta definito il modello, la densità marginale di $\boldsymbol{x}$ può essere ricavata tramite l'integrale

$$p_{\boldsymbol{x}}(\boldsymbol{x}) \;=\; \int p_{\boldsymbol{x}|\boldsymbol{z}}(\boldsymbol{x} \mid \boldsymbol{z})\, p_{\boldsymbol{z}}(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z} \;=\; \left(\frac{1}{\sqrt{2\pi}}\right)^{F+D} \sigma^{-D} \int \exp\!\Big(-\tfrac{v(\boldsymbol{x},\boldsymbol{z})}{2}\Big)\, \mathrm{d}\boldsymbol{z},$$

dove $v(\boldsymbol{x}, \boldsymbol{z}) \;=\; \|\boldsymbol{z}\|^2 + \sigma^{-2}\|\boldsymbol{x} - G(\boldsymbol{z})\|^2$ e $p_{\boldsymbol{x}|\boldsymbol{z}}(\boldsymbol{x} \mid \boldsymbol{z}) \;=\; \big(\sqrt{2\pi}\,\sigma\big)^{-D}$ sotto gaussianità di $\boldsymbol{z}$. L'integrale è approssimabile tramite campionamento per importanza o altri metodi di approssimazione (per maggiori dettagli si veda Liu *et al.*, 2021).

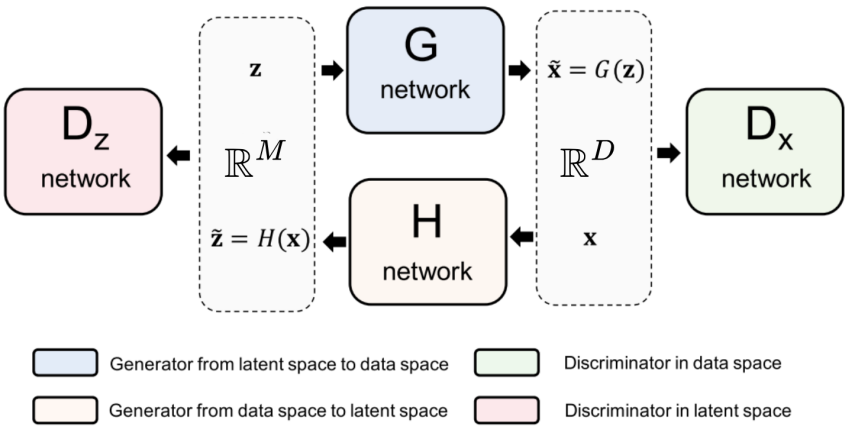

Figura 3.5: Architettura di *roundtrip*, i due generatori dei GAN approssimano le trasformazioni dallo spazio latente a quelle delle osservazioni e viceversa. Immagine presa da Liu *et al.* (2021).

Capitolo 4

Uno studio di simulazione

4.1 Descrizione dello studio

4.1.1 Obiettivi e impostazione dello studio

In questo capitolo si presentano i risultati di uno studio di simulazione progettato con l'obiettivo di valutare le prestazioni di alcuni metodi basati sulle reti neurali per la stima della densità, in termini assoluti e relativamente a stimatori della densità più classici, e al variare di determinate condizioni. I metodi considerati sono valutati su scenari di natura diversa, che differiscono per:

- struttura dei dati;
- numero di variabili;
- numerosità del *training set*.

In merito alla struttura dei dati, la difficoltà principale dello studio nasce dalla necessità di considerare scenari di simulazione che, oltre ad assumere caratteristiche diverse in modo da poter valutare sotto quali condizioni i modelli di stima a confronto sono messi maggiormente alla prova, siano facilmente generalizzabili a un numero arbitrario di dimensioni. Al fine di trarre considerazioni generali al variare degli scenari, è infatti necessario tenere sotto controllo la fonte di variabilità dei dati. Per questa ragione, e per mantenere sostenibili i tempi computazionali dello studio, sono considerati i due seguenti scenari:

(A) i dati sono generati da una distribuzione Normale D-dimensionale con matrice di varianza autoregressiva: $\mathbf{X}_A \sim \mathcal{N}(0, \Sigma_{AR})$, dove $(\Sigma_{AR})_{ij} = \rho^{|i-j|}; i, j = 1, \ldots, D,$ con $\rho = 0.9$. La Figura 4.1 riporta un'illustrazione per $D = 5$.

(B) i dati prevedono una forma di dipendenza non lineare e più complessa, con $\mathbf{X}_B = e^{0.7\mathbf{X}_A}$ e $\mathbf{X}_A$ definita nello scenario precedente. Si veda un'illustrazione nella Figura 4.2.

Per ognuno dei due scenari si fa variare il numero di dimensioni $D \in \{5, 10, 20\}$ e la numerosità campionaria $N \in \{5000, 10000\}$, definendo così un totale di 12 possibili combinazioni di modello generatore dei dati, numero di dimensioni e numerosità del campione.

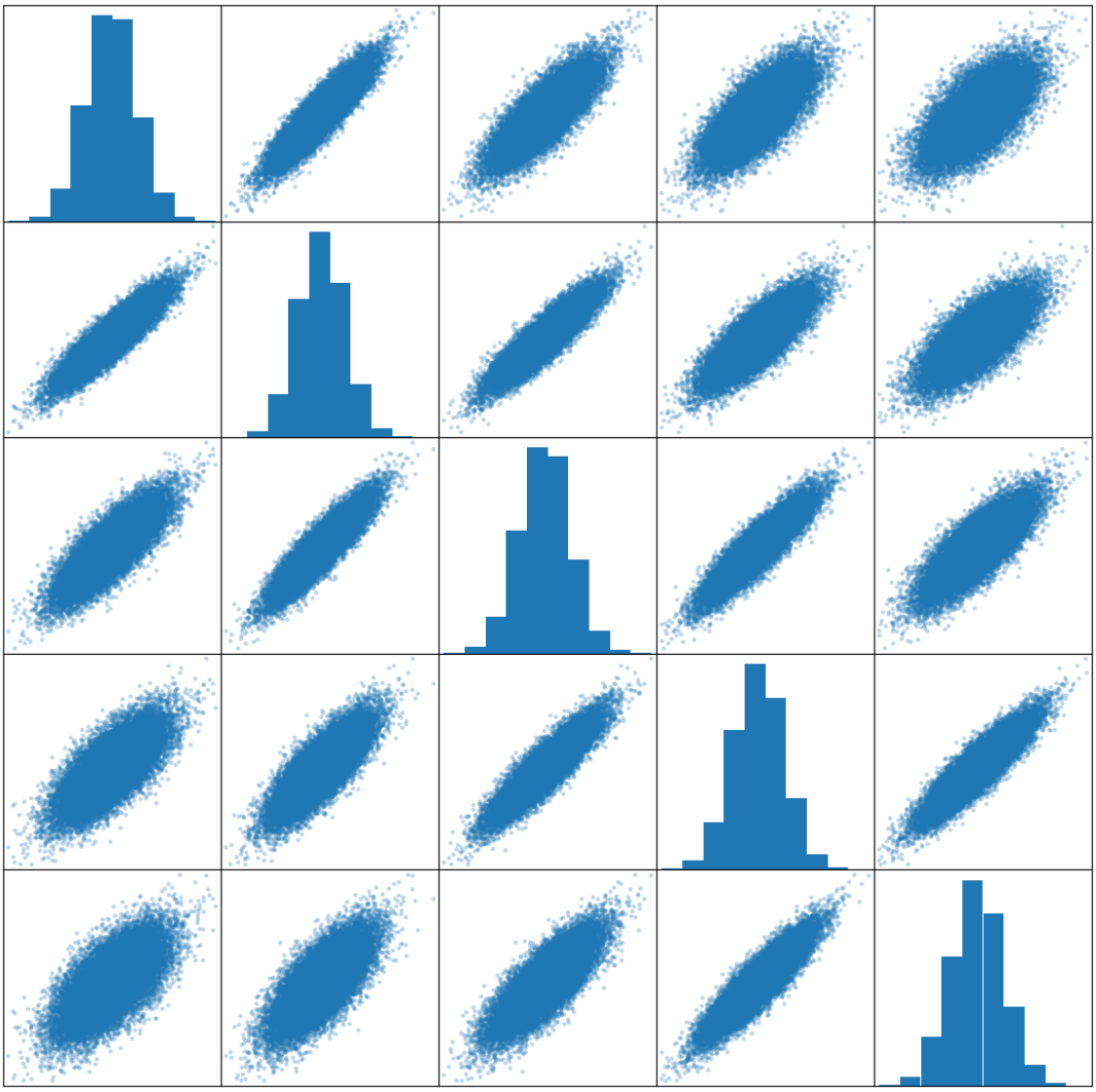

Figura 4.1: Un esempio di dati generati nello scenario A, qui $D = 5$. I dati seguono una distribuzione normale con struttura autoregressiva per la matrice di covarianza.

Nell'esperimento sono messi a confronto quattro modelli: due classici (uno parametrico e uno non parametrico) e due basati su reti neurali (un modello autoregressivo e un *normalizing flow*). Più precisamente, sono stati utilizzati:

- un modello mistura di densità gaussiane (GMM), come definito nella (1.1).
- uno stimatore *kernel*, come definito nella (1.2).
- un modello RNADE, come definito nella Sezione 3.2.1.
- un modello RealNVP, come definito nella Sezione 3.3.1.

Per ogni scenario considerato, vengono generati $M = 50$ campioni. La dimensione effettiva del training set usato per l'addestramento dei modelli è pari a $N_{train} = 0.8N \in \{4000, 8000\}$. Il restante 20% dei dati generati funge da *validation set* e viene utilizzato per la scelta degli iperparametri o per l'*early stopping*.

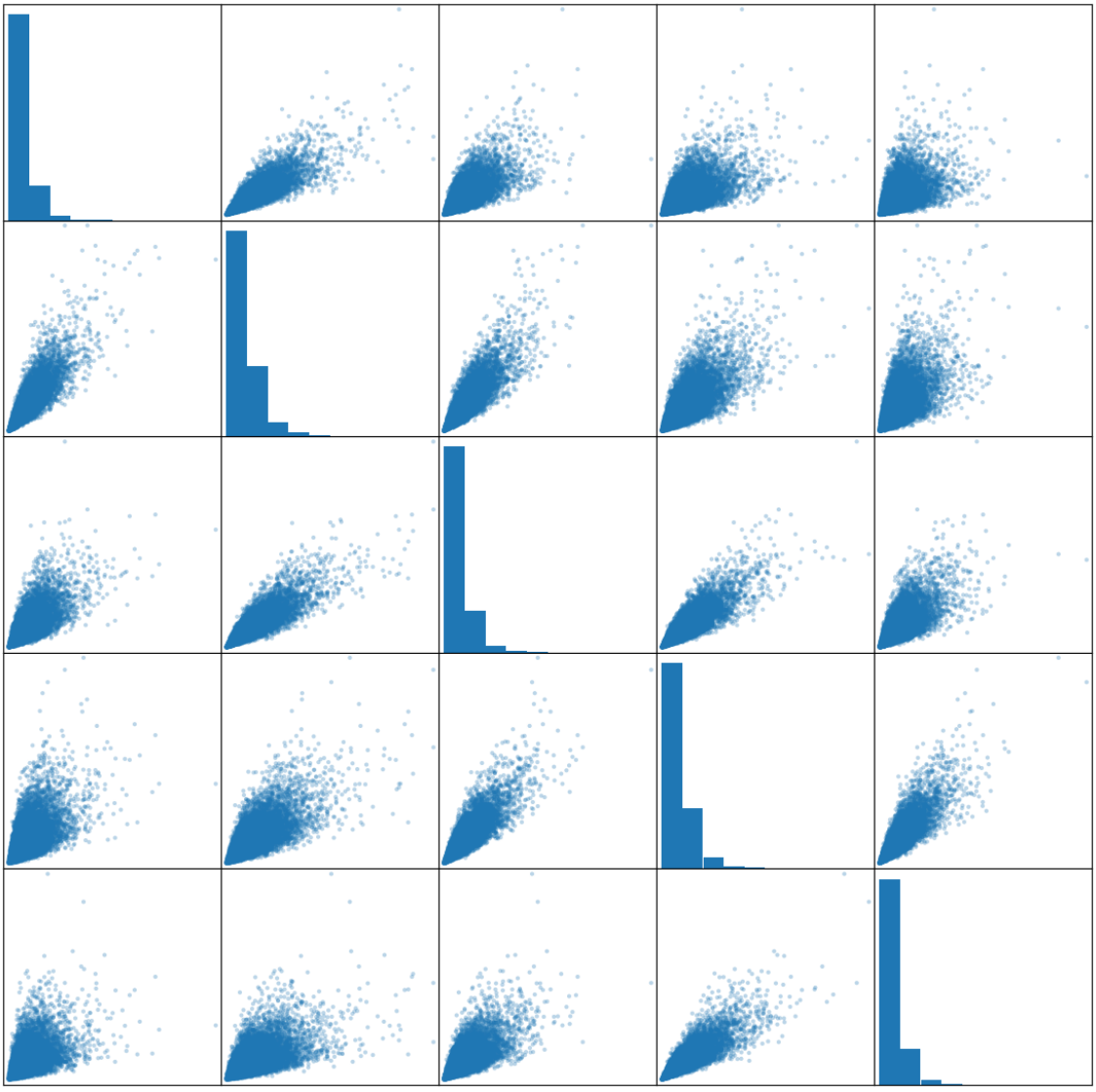

Figura 4.2: Un esempio di dati generati nello scenario B, qui $D = 5$. In ogni dimensione i dati assumono valori positivi, con maggiore massa di densità concentrata in punti vicino all'origine.

Le prestazioni dei modelli stimati vengono valutate su un *test set* di dimensione $N_{test} = 20000$. Poiché il valore analitico della funzione di densità in corrispondenza dei punti del *test set* è noto, può essere utilizzato direttamente per misurare la qualità delle stime. In particolare, l'errore di stima è calcolato mediante lo

$$\widehat{\mathrm{IAE}} \;=\; \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \frac{\left|\hat{p}(\boldsymbol{x^{(i)}}) - p(\boldsymbol{x^{(i)}})\right|}{p(\boldsymbol{x^{(i)}})}, \tag{4.1}$$

che è un'appropriata stima dello IAE, siccome è possibile approssimare l'integrale

$$\int \left|\hat{p}(\boldsymbol{x}) - p(\boldsymbol{x})\right| dx \;=\; \mathbb{E}_{\boldsymbol{X}\sim p}\Big[\tfrac{\left|\hat{p}(\boldsymbol{x})-p(\boldsymbol{x})\right|}{p(\boldsymbol{x})}\Big]$$

con la corrispettiva media campionaria.

Come sarà chiarito in seguito, a tale misura si affiancherà la variante

$$\widehat{\mathrm{IAE}}^{*} \;=\; \mathrm{median}\Big\{ \frac{\left|\hat{p}(\boldsymbol{x^{(i)}}) - p(\boldsymbol{x^{(i)}})\right|}{p(\boldsymbol{x^{(i)}})} \;\Big|\; i = 1, \ldots, N_{\mathrm{test}} \Big\} \tag{4.2}$$

che risulta meno sensibile a valori anomali di $\left|\hat{p}(\boldsymbol{x^{(i)}}) - p(\boldsymbol{x^{(i)}})\right|$.

Una volta calcolate la (4.1) e la (4.2) per ogni ripetizione $j = 1, \ldots, M$ dell'esperimento, è possibile farne una media, ottenendo

$$\widehat{\mathrm{MIAE}} = \frac{1}{M} \sum_{j=1}^{M} \widehat{\mathrm{IAE}}_{\mathrm{j}}, \tag{4.3}$$

che è uno stimatore corretto per la (1.6).

Se invece viene utilizzata la (4.2) si ottiene

$$\widehat{\mathrm{MIAE}}^{*} = \frac{1}{M} \sum_{j=1}^{M} \widehat{\mathrm{IAE}}_{\mathrm{j}}^{*}. \tag{4.4}$$

Una stima dello *standard error* della (4.3) viene fornita calcolando la quantità

$$\frac{\hat{\sigma}}{\sqrt{M}} \quad , \text{ dove } \hat{\sigma} = \frac{\sum_{j=1}^{M} \big\{\widehat{\mathrm{IAE}}_{\mathrm{j}} - \widehat{MIAE}\big\}^2}{M-1}, \tag{4.5}$$

o per la (4.2) calcolando

$$\frac{\hat{\sigma}}{\sqrt{M}} \quad , \text{ dove } \hat{\sigma} = \frac{\sum_{j=1}^{M} \big\{\widehat{\mathrm{IAE}}_{\mathrm{j}}^{*} - \widehat{MIAE}^{*}\big\}^2}{M-1}. \tag{4.6}$$

4.1.2 Dettagli di implementazione

Le analisi condotte sono state effettuate nell'ambiente di programmazione Python (Python Software Foundation, 2024). Il codice per riprodurre i risultati è disponibile sulla piattaforma *Github*[1] a cui si rimanda per maggiori dettagli e riferimenti alle

[1] `https://github.com/santiric/nn_density_estimation`

librerie utilizzate. Si segnala che a questo riferimento è possibile trovare delle funzioni utili in un contesto più generale di questa simulazione, come un'implementazione del modello RNADE che permette una ricerca a griglia per gli iperparametri (come suggerito nella Sezione 2.2.2).

Si riportano di seguito alcuni dettagli di implementazione.

- Il numero di componenti G della mistura di densità Gaussiane è stato selezionato massimizzando la verosimiglianza sul *validation set* nell'insieme nell'insieme $\{1, 3, 5, 7, 10, 15, 20, 25, 30, 35, 40\}$
- La matrice di lisciamento $H = cH_0$ utilizzata nello stimatore *kernel*, è stata determinata selezionando H_0 in base alla regola di Scott (si veda Wand e Jones, 1995). Il valore di c è stato scelto in modo da massimizzare la verosimiglianza sul *validation set*.
- Il numero di iperparametri nel modello RNADE è stato fissato a priori, ad esempio $H = 50$ e $C = 10$. Il *validation set* è stato usato per l'*early stopping*. Per ulteriori dettagli si rimanda alla cartella *github*.
- Il numero di iperparametri nel modello RealNVP è stato fissato a priori, ad esempio il numero di *coupling layers* è stato fissato a 10. Il *validation set* è stato usato per l'*early stopping*. Per ulteriori dettagli si rimanda alla cartella *github* e all'implementazione del modello reperibile al *link* `https://github.com/kamenbliznashki/normalizing_flows`.
- L'ordine delle variabili è stato scelto casualmente all'inizio di ogni ripetizione dell'esperimento, siccome influenza i due modelli basati sulle reti neurali.

4.2 I risultati

Una sintesi dei risultati ottenuti utilizzando la Metrica 4.3, insieme alla Stima 4.5 per lo *standard error*, è riportata nella Tabella 4.1.

La densità nello scenario A è, nella maggior parte dei casi, la più facile da stimare per tutti i metodi e, come prevedibile, in ogni variante dello scenario A, il GMM porta ai risultati migliori perché le assunzioni alla base di questo metodo, per $G = 1$, coincidono con la legge distributiva usata per generare i dati, rendendo così le sue stime più accurate ed efficienti.

Una considerazione simile può essere fatta per RNADE, nonostante il numero di componenti fosse stato fissato a 10. Infatti, nello scenario A questo modello risulta nettamente più accurato di RealNVP e del KDE. Quest'ultimo, invece, porta generalmente ai risultati peggiori e risente maggiormente della maledizione della dimensionalità, come previsto.

Invece, nello scenario B RealNVP porta sempre a risultati migliori, sebbene la significatività in alcuni di questi andrebbe confermata con opportuni test aggiustati per i confronti multipli, ma le considerazioni rimangono comunque valide. Si nota che in questo scenario $X \in \mathbb{R}^{+D}$ e invece modelli come il GMM o RNADE modellano la densità per tutti i reali, quindi una parte di essa viene attribuita a punti esterni al supporto, portando a una sottostima per i punti in $\mathbb{R}^{+D}$, siccome la densità ha il

Tabella 4.1: $\widehat{\text{MIAE}}$ (4.3) al variare della dimensione, della numerosità e dello scenario (la Stima 4.5 dello *standard error* è tra parentesi)

Dimensione	Numerosità	Scenario	GMM	KDE	Real NVP	RNADE
5	5000	A	0.0531 (0.00184)	0.2970 (0.00196)	0.2300 (0.00383)	0.1493 (0.00239)
5	10000	A	0.0364 (0.00126)	0.2580 (0.00095)	0.2190 (0.00361)	0.1120 (0.00188)
10	5000	A	0.0956 (0.00123)	0.6994 (0.00413)	0.4421 (0.00401)	0.3351 (0.00276)
10	10000	A	0.0654 (0.00080)	0.6380 (0.00305)	0.3980 (0.00416)	0.2370 (0.00231)
20	5000	A	0.1840 (0.00125)	1.3932 (0.10525)	0.8098 (0.00424)	0.7504 (0.00507)
20	10000	A	0.1280 (0.00076)	1.2100 (0.01700)	0.7400 (0.00450)	0.5290 (0.00296)
5	5000	B	0.3930 (0.00266)	0.6570 (0.00851)	0.3540 (0.00584)	0.5560 (0.02200)
5	10000	B	0.3330 (0.00276)	0.6260 (0.01040)	0.3310 (0.00589)	0.5850 (0.02870)
10	5000	B	0.7810 (0.00510)	0.9750 (0.00944)	0.6210 (0.00652)	0.8250 (0.02120)
10	10000	B	0.7270 (0.02300)	0.9930 (0.03890)	0.6020 (0.00702)	0.8760 (0.05360)
20	5000	B	1.1648 (0.00806)	1.0849 (0.00996)	1.0396 (0.01170)	1.1153 (0.03040)
20	10000	B	1.1860 (0.03300)	1.0745 (0.01170)	1.0089 (0.01270)	1.0876 (0.02060)

vincolo di integrabilità a 1 (si veda la Figura 4.3 per un esempio intuitivo). Dunque, esiste una distorsione sistematica delle stime per questi metodi. Invece RealNVP, sebbene sia anch'esso definito per tutti i reali, è in grado di gestire meglio questo problema, siccome mappa solamente le $\boldsymbol{x}$ possibili in $\boldsymbol{z}$. Inoltre, RealNVP assume che esista una relazione tra i dati e una variabile latente normale, che è molto simile a come vengono effettivamente generati i dati, conferendogli un vantaggio simile a quello che aveva il GMM nello scenario A.

Mentre nello scenario A si riscontra un miglioramento marcato dell'accuratezza della stima all'aumentare della numerosità campionaria per tutti i modelli considerati, lo stesso non può dirsi netto nello scenario B. È ragionevole ritenere che, per quanto riguarda il modello RNADE, questo comportamento sia dovuto a una scelta subottimale degli iperparametri usati nell'algoritmo di stima e nell'architettura della rete, che appaiono più rilevanti di N in termini di impatto sulla varianza della stima, nascondendo l'effetto della numerosità campionaria sulla (4.3) per un numero non sufficientemente alto di ripetizioni.

È utile precisare che un valore assunto dalla (1.6) pari a 1 corrisponda a stime molto vicine allo zero in ogni punto, siccome la (1.6) si riduce all'integrazione della densità stessa. Nel caso D=20 dello scenario B tutti i metodi ottengono una metrica stimata per la (1.6) leggermente peggiore al caso in cui $\hat{p}(\boldsymbol{x}) = 0$ per ogni $\boldsymbol{x}$. I motivi sono principalmente due: il primo è che, all'aumentare della dimensionalità, ci sono dei punti vicini alla frontiera del supporto in cui il valore assunto dalla densità cresce esponenzialmente (si veda la Figura 4.2) e, in corrispondenza di tali punti, nessun metodo è in grado di fornire stime sufficientemente vicine alla reale densità. Questo inflaziona particolarmente la metrica siccome, per definizione di densità, questi punti appaiono molto spesso nel *test set*. Il secondo motivo è che, per tutti i metodi, si assiste alla presenza di *outlier* per la quantità $\frac{\left|\hat{p}(\boldsymbol{x}^{(i)}) - p(\boldsymbol{x}^{(i)})\right|}{p(\boldsymbol{x}^{(i)})}$, per i quali la (4.1) è molto sensibile (si veda la Figura 4.4).

Per meglio comprendere questo comportamento, alla (4.1) è stato affiancato il calcolo della (4.2), più robusta a valori estremi. I risultati relativi a questa metrica per lo scenario B sono riportati nella Tabella 4.2 e un confronto visivo tra i metodi per il caso $D = 20$ è illustrato nelle Figure 4.5 e 4.6. La distribuzione empirica

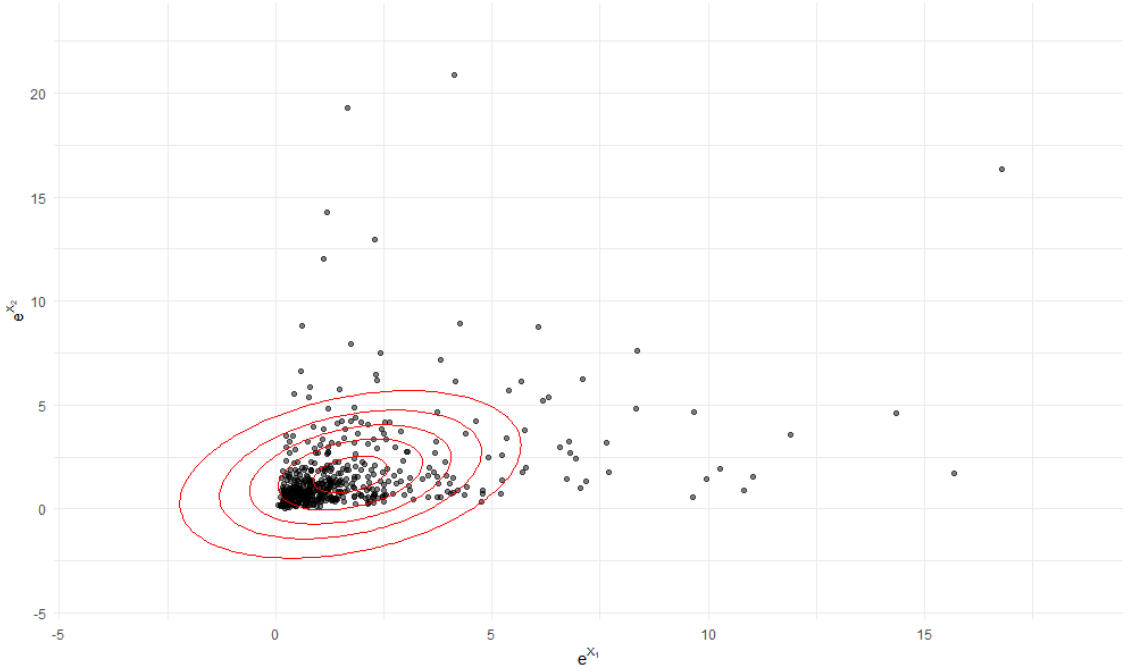

Figura 4.3: In rosso vengono rappresentate delle curve di equidensità. In questa immagine D=2 e viene stimata la densità assuemndo C=1 per un GMM. Si nota come si assegnino valori alti per la densità in corrispondenza di punti esterni al supporto, che dovrebbero avere densità 0. Considerazioni simili possono essere fatte in più dimensioni e per più componenti della mistura.

degli errori di stima evidenzia la scarsa performance del KDE in dimensioni elevate, che non era evidente nella tabella 4.1. La migliore qualità delle stime di RealNVP è chiara, mentre RNADE fornisce risultati molto variabili, verosimilmente a causa della scelta arbitraria degli iperparametri usati nell'algoritmo di stima, e in generale a una più elevata varianza di stima per le reti neurali. Il GMM porta a una minore varianza di stima a causa del suo minor numero di parametri, mentre la minore varianza riscontrata nel KDE è dovuta al fatto che questo metodo tende ad assegnare densità molto vicine allo 0 per molti punti, portando così a una metrica molto vicina a 1 in tutte le ripetizioni dell'esperimento (si veda la Tabella 4.3).

Nello scenario B si nota una maggiore robustezza alla maledizione della dimensionalità per i metodi che sfruttano le reti neurali, soprattutto in RealNVP; questo è coerente con le considerazioni fatte nella Sezione 2.3. Infatti, come illustrato nelle Figure 4.7 e 4.8, il GMM ottiene risultati simili a quelli di RealNVP e nettamente migliori di quelli di RNADE nel caso D=5, ma significativamente peggiori a entrambi nel caso D=20.

Per altre metriche e per i risultati relativi ad ogni ripetizione dell'esperimento si rimanda alla cartella *github* precedentemente citata.

Per riassumere, la coerenza tra l'assunzione sulla forma della densità di un modello e la distribuzione generativa effettiva emerge come elemento cruciale nella qualità delle stime: nel contesto di dati gaussiani (scenario A), il GMM risulta portare alle migliori prestazioni, mentre in presenza di trasformazioni non lineari che producono dati sul supporto $\mathbb{R}^{+D}$ (scenario B) i modelli basati su *normalizing flow* producono

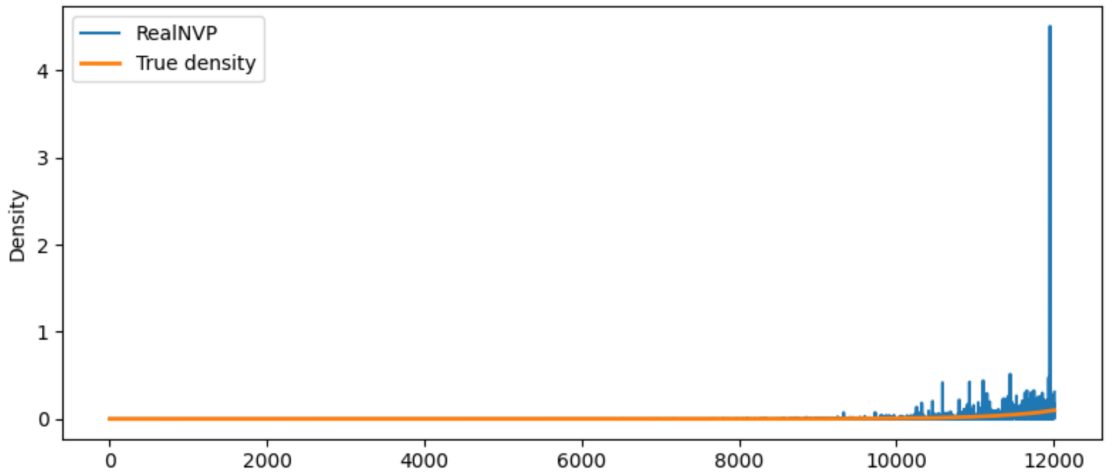

Figura 4.4: Confronto tra la densità teorica per alcune osservazioni e quella stimata da RealNVP in una ripetizione dello scenario B per D=20. La densità teorica è stata ordinata per motivi grafici. Si nota la presenza di una stima particolarmente distante dalla reale densità, che influenza di molto la (4.1).

stime migliori. Si osserva una varianza di stima più elevata nei modelli basati su reti neurali, ma una maggiore robustezza alla maledizione della dimensionalità.

Un possibile approfondimento futuro consisterebbe nel quantificare quanto di tale variabilità sia imputabile a scelte subottimali degli iperparametri e quanto, invece, derivi da fattori intrinseci al modello, siccome ciò non è chiaro con l'implementazione attuale.

In generale, si nota un vantaggio più ampio nell'uso delle reti neurali quando i dati presentano una complessità maggiore, sia in termini di struttura che di dipendenza tra variabili. Questo è ragionevole, siccome per questi modelli non è necessario specificare un modello rigido in cui ricercare la funzione di densità, permettendo una maggiore adattabilità a forme complesse presenti nei dati.

Tabella 4.2: $\widehat{\text{MIAE}}^*$ (4.4) per lo scenario B, al variare della dimensione e della numerosità (la Stima 4.6 dello *standard error* è tra parentesi)

Dimensione	Numerosità	Scenario	GMM	KDE	Real NVP	RNADE
5	5000	B	0.2897 (0.00257)	0.5118 (0.00444)	0.2398 (0.00470)	0.4089 (0.01994)
5	10000	B	0.2382 (0.00154)	0.4704 (0.00488)	0.2259 (0.00467)	0.4258 (0.02409)
10	5000	B	0.6180 (0.00164)	0.8902 (0.00129)	0.4464 (0.00444)	0.6471 (0.02153)
10	10000	B	0.5464 (0.00139)	0.8693 (0.00136)	0.4301 (0.00476)	0.6377 (0.02145)
20	5000	B	0.9374 (0.00065)	0.9968 (0.00008)	0.7753 (0.00231)	0.8470 (0.01350)
20	10000	B	0.9059 (0.00085)	0.9959 (0.00005)	0.7299 (0.00261)	0.8542 (0.01747)

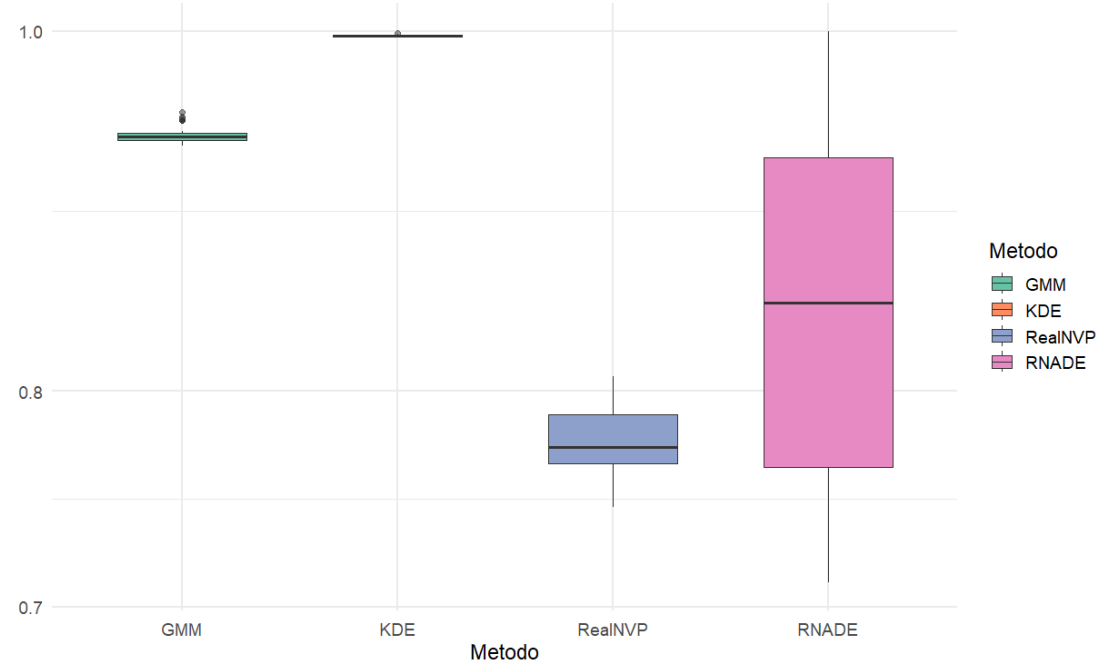

Figura 4.5: Valori di $\widehat{\text{IAE}}^*$ (4.2) per lo scenario B, D=20 e N=5000.

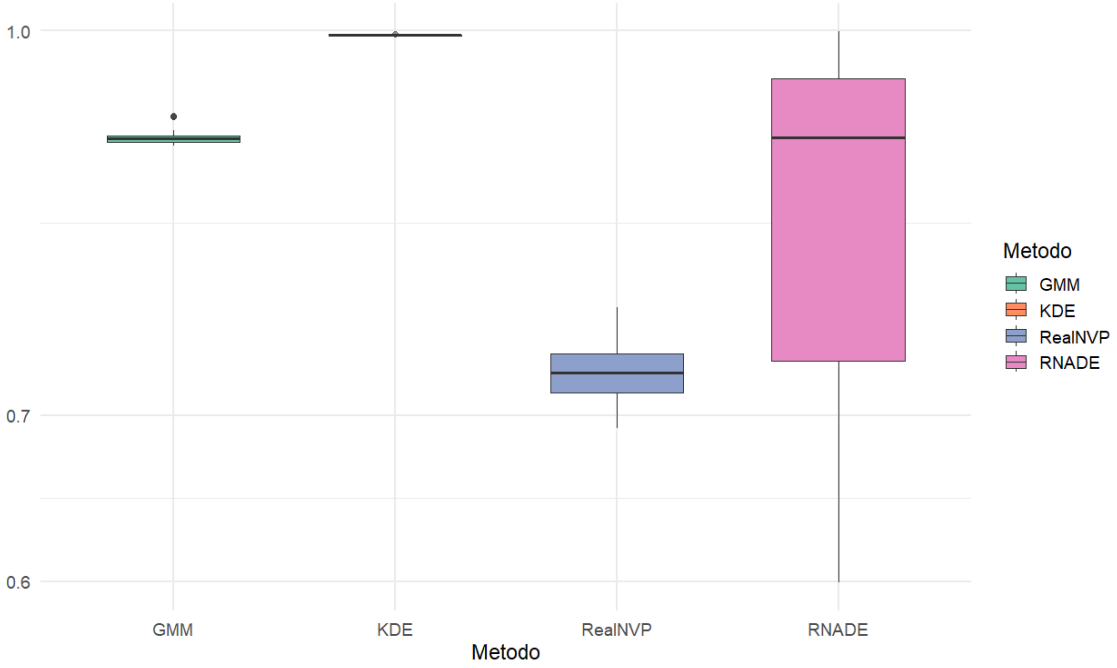

Figura 4.6: Valori di $\widehat{\text{IAE}}^*$ (4.2) per lo scenario B, D=20 e N=10000.

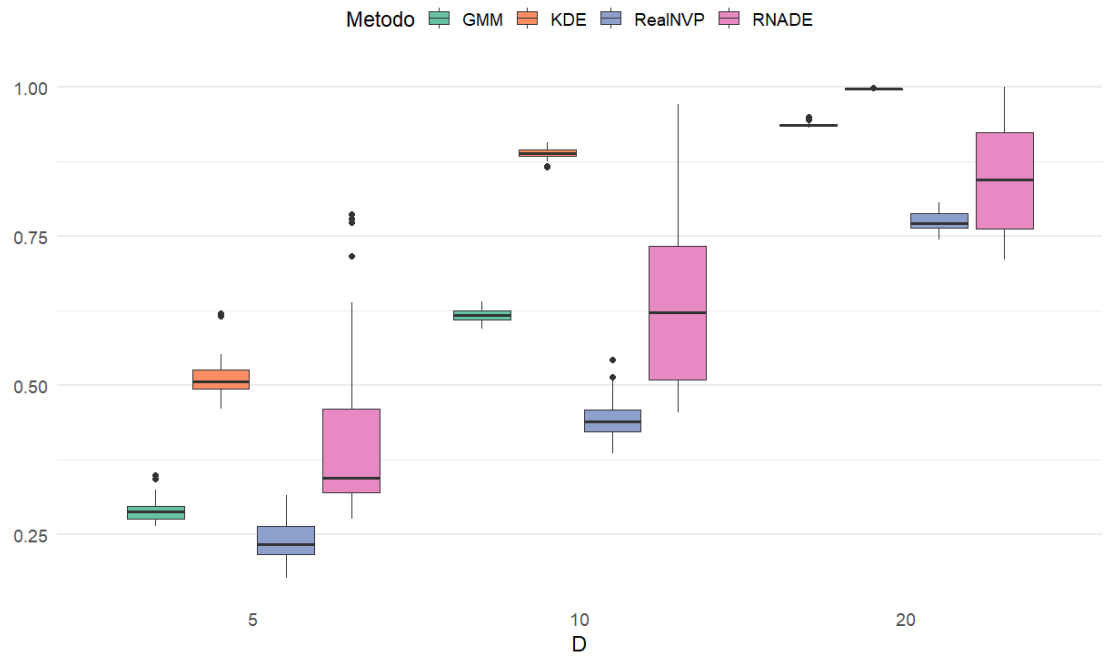

Figura 4.7: Valori di $\widehat{\text{IAE}}^*$ (4.2) per ogni metodo nello scenario B, al variare della dimensionalità e fissato N=5000.

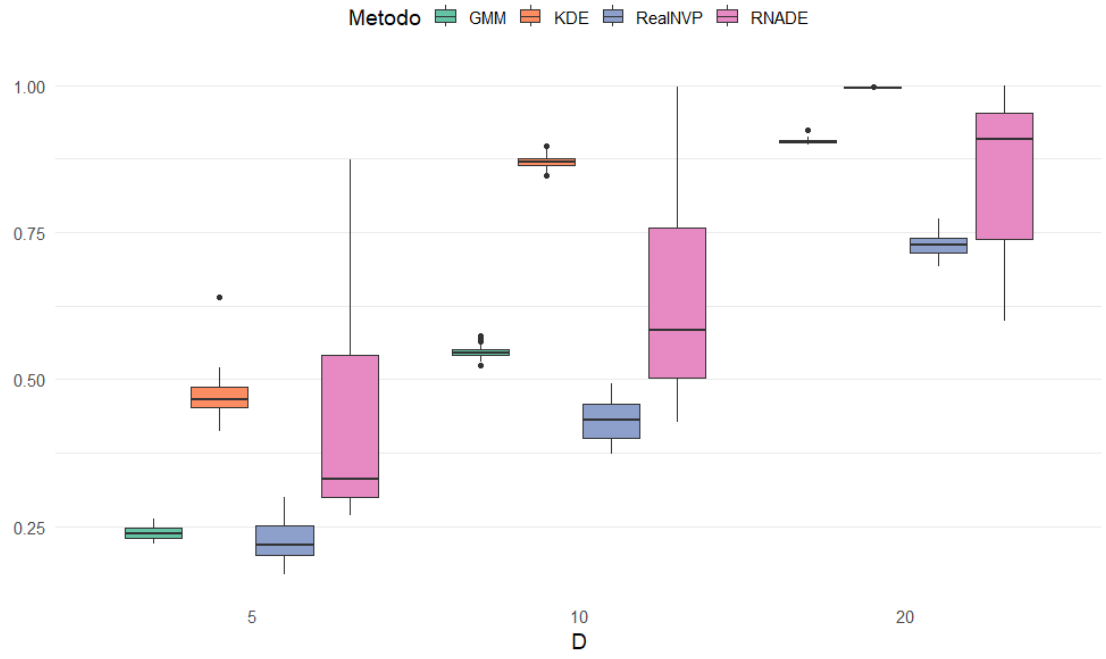

Figura 4.8: Valori di $\widehat{\text{IAE}}^*$ (4.2) per ogni metodo nello scenario B, al variare della dimensionalità e fissato N=10000.

Tabella 4.3: Un esempio di densità stimate con KDE per lo scenario B, $D = 20$ e $N = 10000$, in una ripetizione dell'esperimento. Vengono riportate la stima della densità $\hat{p}(\boldsymbol{x}^{(i)})$ in alcuni punti, il vero valore $p(\boldsymbol{x}^{(i)})$ corrispondente e la loro differenza assoluta relativa, che risulta essere spesso vicina a 1.

$\hat{p}(\boldsymbol{x}^{(i)})$	$p(\boldsymbol{x}^{(i)})$	$\frac{\lvert\hat{p}(\boldsymbol{x}^{(i)}) - p(\boldsymbol{x}^{(i)})\rvert}{p(\boldsymbol{x}^{(i)})}$
0.0202186	151.8781637	0.9998669
1.298195e-08	2.523267e-07	0.9485510
6.161817e-08	3.173274e-08	0.9417852
0.0008140	8.4898570	0.9999041
3.714187e-05	0.0013367	0.9722147
0.0614520	1.052517e+06	0.9999999
1.631818e-10	6.649577e-10	0.7545983
7.956630e-14	4.662750e-11	0.9982936
1.135306e-19	3.157425e-11	0.9999999
0.0047128	0.7992507	0.9941034

Conclusioni

In questo elaborato è stata innanzitutto offerta una panoramica generale sul problema non supervisionato della stima della funzione di densità, e una descrizione rapida degli approcci più tradizionali. Successivamente, sono state introdotte le reti neurali, prima in termini generali e poi nello specifico con l'obiettivo della stima della densità. Infine è stato condotto uno studio di simulazione, volto a indagare il comportamento delle reti neurali in confronto a metodi più tradizionali, e a verificare alcune caratteristiche delle reti attese sulla base di risultati teorici.

Come confermato da questa simulazione, questi modelli molto flessibili sono particolarmente utili per la stima di funzioni complesse, quando è più difficile pensare a una forma funzionale adeguata per la stima della densità.

Invece, se la struttura dei dati ha una forma imputabile a una classica famiglia distributiva, l'informazione che viene aggiunta facendo questa assunzione comporta un miglioramento netto delle stime rispetto a quelle fornite dalle reti neurali, che non riescono invece a ottenere un buon compromesso tra *bias* e varianza. In questo senso le reti neurali, pur essendo modelli parametrici, si comportano in modo più simile a un modello non parametrico, dato l'elevato numero di parametri che contengono.

Al contrario, in modo molto diverso dai modelli non parametrici, i metodi usati per le reti portano a stime di qualità anche in dimensioni elevate, poiché sembrano essere in grado di cogliere più facilmente la struttura di fondo dei dati, grazie a una modellazione diretta delle dipendenze tra variabili o tramite l'uso di variabili latenti.

Un altro aspetto importante da considerare è quello computazionale, siccome il processo di addestramento di una rete neurale può diventare particolarmente lungo all'aumentare del numero di strati. Tuttavia, come visto nella simulazione, reti di piccole dimensioni, che richiedono tempi moderati per la stima, possono portare in alcuni casi a risultati significativamente migliori di quelli dei metodi classici.

Invece, è emerso solo parzialmente nello studio del Capitolo 4 l'impatto che ha la numerosità campionaria sulle stime prodotte dalle reti. Tuttavia, questo è un fattore cruciale per scegliere se usare o meno questi modelli, siccome le reti neurali sono molto esposte al rischio di *overfitting* e l'aumento dei dati a disposizione è un modo immediato per abbassare l'alta varianza di stima che lo causa.

Nel complesso, le reti neurali sembrano un'alternativa valida ai metodi tradizionali, quando risulta difficile ipotizzare un modello parametrico semplice per la distribuzione dei dati o quando i limiti dei modelli non parametrici li rendono inutilizzabili, come nel caso di alta dimensionalità.

Bibliografia

Abiodun O. I.; Jantan A.; Omolara A. E.; Dada K. V.; Mohamed N. A.; Arshad H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, **4**(11).

Aggarwal C. C. (2023). *Neural Networks and Deep Learning: A Textbook*. Springer, 2nd edizione.

Apicella A.; Donnarumma F.; Isgrò F.; Prevete R. (2021). A survey on modern trainable activation functions. *Neural Networks*, **138**, 14–32.

Augustine M. T. (2024). A survey on universal approximation theorems.

Azzalini A. (2001). *Inferenza statistica, una presentazione basata sul concetto di verosimiglianza*. Springer-Verlag Italia.

Barron A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, **39**(3), 930–945.

Bergstra J.; Bengio Y. (2012). Random search for hyper-parameter optimization. *The Journal of machine learning research*, **13**(1), 281–305.

Biau G.; Devroye L. (2015). *Lectures on the nearest neighbor method*, volume 246. Springer.

Bischl B.; Binder M.; Lang M.; Pielok T.; Richter J.; Coors S.; Thomas J.; Ullmann T.; Becker M.; Boulesteix A.-L.; Deng D.; Lindauer M. (2021). Hyperparameter optimization: Foundations, algorithms, best practices and open challenges.

Dempster A. P.; Laird N. M.; Rubin D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, **39**(1), 1–22.

Dinh L.; Sohl-Dickstein J.; Bengio S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Fischer A.; Igel C. (2012). An introduction to restricted boltzmann machines. In *Iberoamerican congress on pattern recognition*, pp. 14–36. Springer.

Germain M.; Gregor K.; Murray I.; Larochelle H. (2015). Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pp. 881–889. PMLR.

Goodfellow I.; Bengio Y.; Courville A. (2016). *Deep Learning*. MIT Press.

Goodfellow I.; Pouget-Abadie J.; Mirza M.; Xu B.; Warde-Farley D.; Ozair S.; Courville A.; Bengio Y. (2020). Generative adversarial networks. *Communications of the ACM*, **63**(11), 139–144.

Hastie T.; Tibshirani R.; Friedman J. H.; Friedman J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer.

Haykin S. (2009). *Neural Networks and Learning Machines*. Pearson International Edition.

Hornik K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, **4**(2), 251–257.

Hornik K.; Stinchcombe M.; White H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, **2**(5), 359–366.

Ioffe S.; Szegedy C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr.

Kingma D. P. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma D. P.; Salimans T.; Jozefowicz R.; Chen X.; Sutskever I.; Welling M. (2016). Improved variational inference with inverse autoregressive flow In *Advances in Neural Information Processing Systems*. A cura di Lee D., Sugiyama M., Luxburg U., Guyon I., Garnett R., volume 29, pp. 4743–4751. Curran Associates, Inc.

Koller D.; Friedman N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Krogh A.; Hertz J. (1991). A simple weight decay can improve generalization In *Advances in Neural Information Processing Systems*. A cura di Moody J., Hanson S., Lippmann R., volume 4, pp. 950–957. Morgan-Kaufmann.

Larochelle H.; Murray I. (2011). The neural autoregressive distribution estimator. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 29–37. JMLR Workshop and Conference Proceedings.

Linnainmaa S. I. (1970). *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. Tesi per Master, University of Helsinki. M.Sc. thesis (in Finnish).

Liu Q.; Xu J.; Jiang R.; Wong W. H. (2021). Density estimation using deep generative neural networks. *Proceedings of the National Academy of Sciences*, **118**(15), e2101344118.

McLachlan G. J.; Peel D. (2000). *Finite mixture models*. John Wiley & Sons.

Nielsen M. A. (2015). *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA.

Papamakarios G.; Pavlakou T.; Murray I. (2017). Masked autoregressive flow for density estimation In *Advances in Neural Information Processing Systems*. A cura di Guyon I., Luxburg U. V., Bengio S., Wallach H., Fergus R., Vishwanathan S., Garnett R., volume 30, pp. 2339–2348. Curran Associates, Inc.

Pinkus A. (1999). Approximation theory of the mlp model in neural networks. *Acta Numerica*, **8**, 143–195.

Prechelt L. (1998). Automatic early stopping using cross validation: quantifying the criteria. *Neural networks*, **11**(4), 761–767.

Prechelt L. (2002). Early stopping-but when? In *Neural Networks: Tricks of the trade*, pp. 55–69. Springer.

Python Software Foundation (2024). Python language reference, version 3.x. `https://www.python.org`.

Robbins H.; Monro S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, pp. 400–407.

Scott D. W. (2015). *Multivariate density estimation : theory, practice, and visualization / David W. Scott, Rice University, Houston, Texas*. Wiley series in probability and statistics. Wiley, Hoboken NJ, 2nd ed. edizione.

Srivastava N.; Hinton G.; Krizhevsky A.; Sutskever I.; Salakhutdinov R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, **15**(1), 1929–1958.

Titterington D. M.; Smith A. F. M.; Makov U. E. (1985). *Statistical analysis of finite mixture distributions*. John Wiley & Sons, Chichester.

Uria B.; Murray I.; Larochelle H. (2013). Rnade: The real-valued neural autoregressive density-estimator. *Advances in Neural Information Processing Systems*, **26**.

Uria B.; Murray I.; Larochelle H. (2014). A deep and tractable density estimator. In *International Conference on Machine Learning*, pp. 467–475. PMLR.

Uria B.; Côté M.-A.; Gregor K.; Murray I.; Larochelle H. (2016). Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, **17**(205), 1–37.

Wand M. P.; Jones M. C. (1995). *Kernel smoothing / M. P. Wand, M. C. Jones*. Chapman & Hall, London.

Wasserman L. (2006). *All of nonparametric statistics*. Springer Science & Business Media.

Yao Y.; Rosasco L.; Caponnetto A. (2007). On early stopping in gradient descent learning. *Constructive approximation*, **26**(2), 289–315.

Zammit-Mangion A.; Sainsbury-Dale M.; Huser R. (2024). Neural methods for amortized inference. *Annual Review of Statistics and Its Application*, **12**, 311—-335.

Zhang Y.; Tiňo P.; Leonardis A.; Tang K. (2021). A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, **5**(5), 726–742.