

UNIVERSITÀ DEGLI STUDI DI TORINO

SCUOLA DI SCIENZE DELLA NATURA

Corso di Laurea Magistrale in Informatica



Appunti del corso di Reti neurali e Deep Learning

Autori:

Falchi Lorenzo
Picone Corrado (come il comico)

Anno Accademico 2024/2025

Questi appunti sono basati sulle slide del corso di Reti neurali e deep learning tenuto dai professori Valentina Gliozzi e Roberto Esposito per l'anno accademico 2024/2025. Questo documento è il risultato dell'integrazione di slide e appunti presi a lezione. Ogni capitolo corrisponde a circa una lezione, quando possibile le lezioni sono accorpate in un singolo capitolo.

Indice

| | | |
|----------|--|-----------|
| 1 | Introduzione | 1 |
| 1.1 | Machine learning | 1 |
| 1.1.1 | Task | 1 |
| 1.1.2 | No free launch theorem | 1 |
| 1.1.3 | Training di una rete neurale | 1 |
| 1.2 | (Matrici e vettori) Calcolo | 3 |
| 1.2.1 | Derivate | 3 |
| 1.2.2 | Integrali | 4 |
| 1.2.3 | Derivate parziali e Gradienti | 5 |
| 1.2.4 | Derivate direzionali | 6 |
| 1.2.5 | Matrice Jacobiana | 8 |
| 1.2.6 | Derivate seconde | 8 |
| 1.3 | Probabilità | 10 |
| 1.3.1 | Probabilità marginale | 11 |
| 1.3.2 | Probabilità condizionata | 12 |
| 1.3.3 | Chain rule della probabilità condizionata | 12 |
| 1.3.4 | Indipendenza | 12 |
| 1.3.5 | Expectation (Valore atteso) | 13 |
| 1.3.6 | Varianza e Deviazione standard | 13 |
| 1.3.7 | Distribuzioni di Probabilità Comuni | 15 |
| 1.3.8 | Proprietà utili delle funzioni comuni | 20 |
| 1.3.9 | Measure Theory | 22 |
| 1.4 | Information theory | 24 |
| 1.4.1 | Quantificare l'informazione | 24 |
| 1.4.2 | Modelli grafici (modelli probabilistici strutturati) | 29 |
| 2 | Autoencoders | 31 |
| 2.1 | Definizione | 32 |
| 2.2 | Undercomplete Autoencoders | 33 |
| 2.2.1 | Autoencoders e PCA | 34 |
| 2.2.2 | Manyfold | 35 |
| 2.3 | Regularized Autoencoders | 36 |
| 2.3.1 | Sparse Autoencoders | 36 |
| 2.3.2 | Generative Models (GM) | 37 |

Capitolo 1

Introduzione

1.1 Machine learning

1.1.1 Task

Le **reti neurali** sono una delle modalità con cui si possono risolvere problemi di **machine learning**. Il machine learning è una branca dell'informatica che si occupa di come i computer apprendono e migliorano i propri risultati "imparando" dai loro stessi errori. I computer vengono in particolare applicati a **task**, i quali possono essere di diversi tipi (classificazione, regressione, clustering, ecc..).

Breve descrizione di alcuni tipi di task:

- **classification**, il task di classificazione è un task **supervisionato**, il che vuol dire che a partire dalla descrizione e soluzione al problema (training set etichettato), vogliamo imparare un algoritmo di machine learning che crei un modello che risolva task di classificazione del tipo desiderato, in maniera autonoma. In particolare **il modello deve essere in grado di generalizzare** e predire correttamente la classe di esempi che non ha visto nella fase di training (esempi: spam detection, sentiment analysis ecc..).
- **regression**, il task di regressione è simile a quello di classificazione, ma invece di predire un set di lable l'output sarà un numero reale (esempi: house price prediction, stock price prediction ecc..). Si può usare un regressore per fare classificazione.

1.1.2 No free lunch theorem

In sostanza, questo teorema afferma che a meno di non considerare i bias, gli algoritmi di machine learning sono sostanzialmente uguali in termini di prestazioni.

1.1.3 Training di una rete neurale

Le reti neurali sono una famiglia di incredibilmente flessibili modelli i quali riescono ad approssimare sostanzialmente qualsiasi funzione. Il prezzo da pagare è l'elevatissimo numero di iper parametri grazie ai quali si possono raggiungere ottime performance. Alcuni tipi di iper parametri sono:

- numero di layer;
- numero di neuroni in ogni layer;
- learning rate;
- optimizer;

-

Importantissimo è imparare a settare questi parametri e validare la qualità del modello. Ora ci concentreremo su questi due argomenti.

Il **generalization error** è l'errore che la nostra funzione \mathbf{g} commette quando vede nuovi esempi, è scritta così:

$$R = E_{(x,y) \sim p^*}[L(y, g(x; \Theta))] \quad (1.1)$$

dove p^* è la **reale distribuzione dei dati** e

$$L(y, g(\cdot; \Theta)) \quad (1.2)$$

è la **loss function** usata per misurare quanto vale l'errore quando y è predetto come $g(x; \Theta)$.

Il problema è che qui R non è veramente calcolabile, non si può fare praticamente. Questo perchè non abbiamo accesso a p^* e anche se ce l'avessimo dovremmo valutare un numero infinito di punti, cosa che non è possibile ovviamente.

Esempi di **loss function** sono la **0 – 1 loss**:

$$L(y, y') = \mathbb{I}_{y \neq y'} = \begin{cases} 1 & \text{if } y \neq y' \\ 0 & \text{otherwise} \end{cases} \quad (1.3)$$

e la **quadratic loss**:

$$L(y, y') = (y - y')^2. \quad (1.4)$$

Tornando a noi, vogliamo misurare il **generalization error** ma come abbiamo detto ciò richiede un sampling dalla distribuzione che noi non abbiamo. Per ovviare a questo problema adotteremo diverse soluzioni, la prima è l'utilizzo di un errore diverso dal generalization, un errore più generale detto **empirical error**:

$$\hat{R}_T = \frac{1}{|T|} \sum_{(x,y) \in T} L(y, g(x; \Theta)), \quad (1.5)$$

dove T è un campione finito estratto da p^* .

Arrivati a questo punto:

- se T denota il **training set**, allora $T \equiv Tr$, \hat{R}_{Tr} denota il **training error** di g ,
- se T denota il **test set**, allora $T \equiv Te$, \hat{R}_{Te} denota il **test error** di g .

Siccome il **training error** è ottimizzato semplicemente con l'algoritmo di apprendimento, ha un **bias ottimistico** (tende ad essere minore del generalization error). Quindi è sostanzialmente un estimatore ottimistico del generalization error.

Il **test error** è invece un estimatore **unbiased** del generalization error R , anche se a determinate condizioni il bias può essere **pessimistico**, per esempio nel caso in cui si tenga da parte una parte del training set (si immagina che trainare il modello una seconda volta utilizzando tutto il training set produca un modello con un errore minore).

Quando si verifica che l'errore sul test set è maggiore di quello sul training set, cioè

$$\hat{R}_{Te} - \hat{R}_{Tr} > 0 \quad (1.6)$$

si parla di **overfitting**. A parole, un modello soffre di overfitting quando le prestazioni sul training set sono molto migliori di quelle sul test set. Un leggero overfitting è invece abbastanza normale e tollerabile ma si tende a renderlo più piccolo possibile.

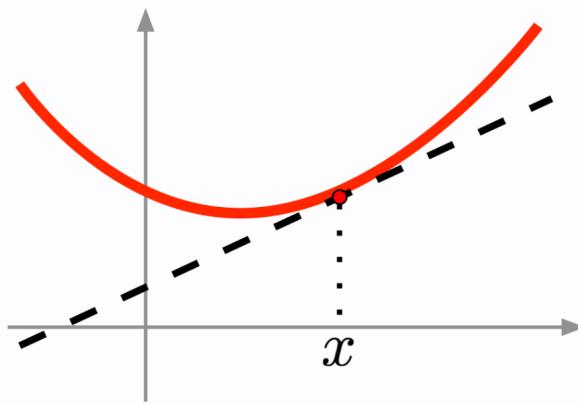
Tornando invece all'altro dei quesiti iniziali, cioè **come stimare gli iperparametri**; con quello che abbiamo imparato fino ad ora, vogliamo implementare questa procedura: redqui ci va il codice python

Sfortunatamente non funziona, per sostanzialmente lo stesso motivo per cui il training è un estimatore ottimistico del generalization erorre, stiamo rendendo anche il test error un estimatore ottimistico. Così facendo stiamo portando il modello a fare overfitting sul test set.

1.2 (Matrici e vettori) Calcolo

1.2.1 Derivate

Data la funzione $y = f(x)$, dove x e y sono numeri reali. La **derivata** di f nel punto x , denotata da $f'(x)$ or $\frac{df}{dx}(x)$ è la **pendenza della tangente** (o **coefficiente angolare**) ad f nel punto x . Tra le cose interessanti della derivata c'è il fatto che se si vuole calcolare il valore di f in un



punto vicino ad x , diciamo ϵ , si può fare in questo modo:

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x). \quad (1.7)$$

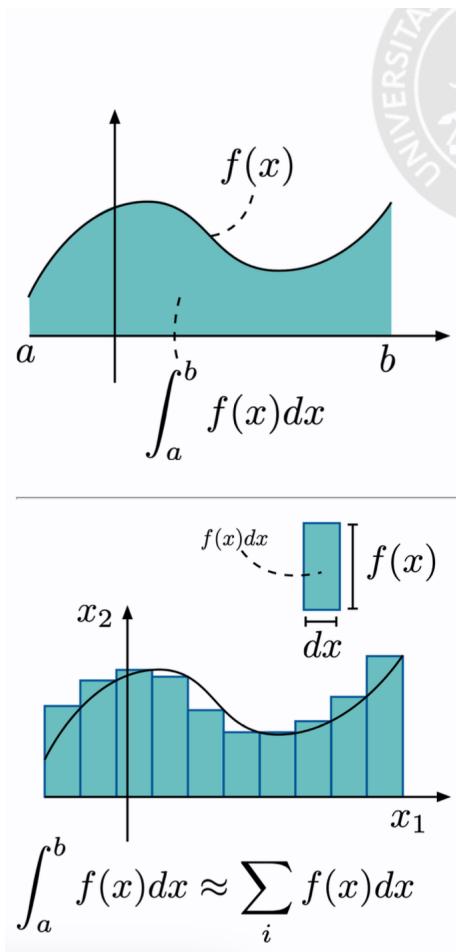
Proprietà importanti delle derivate:

- **linearity** $(\alpha f(x) + \beta g(x))' = \alpha f'(x) + \beta g'(x)$
- **chain rule** $(f(g(x)))' = f'(g(x))g'(x)$
- **product rule** $(g(x)h(x))' = g'(x)h(x) + g(x)h'(x)$
- **quotient rule** $\left(\frac{f(x)}{g(x)}\right)' = \frac{f(x)'g(x) - f(x)g'(x)}{(g(x))^2}$
- **power rule** $(x^r)' = rx^{r-1}$

1.2.2 Integrali

Data la funzione $f : \mathbb{R} \rightarrow \mathbb{R}$ e un intervallo $[a, b]$ sulla retta reale:

l'integrale di f tra a e b rappresenta l'area sotto f nella regione delimitata dagli **estremi dell'intervallo** (quando la funzione si trova sotto lo 0, l'area contribuisce negativamente). Ciò che è interessante è che esiste un teorema, il **teorema fondamentale dell'analisi**,



che afferma che **esiste una relazione stretta tra integrali e derivate**. In particolare, se f ammette un'**antiderivata** F , se esiste F tale che $F'(x) = f(x)$ allora:

$$\int f(x)dx = F(x) + C \quad (1.8)$$

e

$$\int_a^b f(x)dx = F(x)|_a^b = F(b) - F(a) \quad (1.9)$$

N.B.

- gli **integrali indefiniti** sono quelli per cui **non è specificato l'intervallo di integrazione** e per cui la soluzione, per convenzione, è l'antiderivata F ;
- gli **integrali definiti** sono quelli per cui **è specificato l'intervallo di integrazione**.

Proprietà importanti degli integrali:

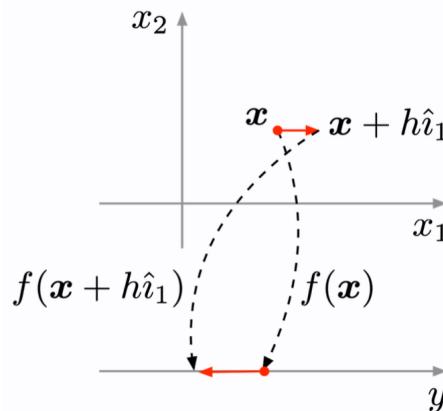
- **linearity** $\int \alpha f(x) + \beta g(x) dx = \alpha \int f(x) dx + \beta \int g(x) dx$
- **constant rule** $\int k dx = kx + C$
- **power rule** $\int x^n dx = \frac{x^{n+1}}{n+1} + C, n \neq -1$
- **log rule** $\int \frac{1}{x} dx = \ln(|x|) + C$
- **exponential rule** $\int a^{kx} dx = \frac{a^{kx}}{k \ln a} + C, x \neq 0$
- **sin rule** $\int \sin(x) dx = -\cos(x) + C$
- **cosin rule** $\int \cos(x) dx = \sin(x) + C$

1.2.3 Derivate parziali e Gradienti

Sia data la **funzione multivariata** $y = f(x_1, \dots, x_n) = f(\mathbf{x})$, dove $y \in \mathbb{R}, \mathbf{x} \in \mathbb{R}^n$.

La **derivata parziale** $\frac{\delta}{\delta x_j} f(\mathbf{x})$ misura come f varia al variare della sola variabile x_j , tutte le altre rimangono invariate:

$$\frac{\delta}{\delta x_j} f(\mathbf{x}) = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\hat{i}_j) - f(\mathbf{x})}{h} = \lim_{h \rightarrow 0} \frac{f(x_1, \dots, x_j + h, \dots, x_n) - f(x_1, \dots, x_n)}{h} \quad (1.10)$$



Il **gradiente** di f , denotato con $\nabla_x f$ (o più semplicemente ∇f) è il vettore che contiene tutte le derivate parziali della funzione f :

$$\nabla f = \left[\frac{\delta f}{\delta x_1}, \dots, \frac{\delta f}{\delta x_n} \right]^T. \quad (1.11)$$

Il gradiente è un vettore con una proprietà molto particolare, per parlare della quale bisogna introdurre prima un concetto: la **regola della catena per il calcolo multivariato**.

Assumiamo $z = f(x, y)$ e siano x, y variabili dipendenti da una variabile addizionale t (gli input di f sono a loro volta funzioni della variabile t , cioè abbiamo $f(x(t), y(t))$), quindi:

$$\frac{dz}{dt} = \frac{\delta z}{\delta x} \frac{dx}{dt} + \frac{\delta z}{\delta y} \frac{dy}{dt}. \quad (1.12)$$

Quello appena visto vale per 2 variabili, più in generale per $f : \mathbb{R}^n \rightarrow \mathbb{R}$, quando x_1, \dots, x_n dipendono da una variabile t :

$$\frac{df}{dt} = \sum_{i=1}^n \frac{\delta f}{\delta x_i} \frac{dx_i}{dt}. \quad (1.13)$$

Facciamo un esempio, date:

- $(x, y) = (t^2, t)$, perciò $x(t) = t^2$ e $y(t) = t$;
- $z = f(x, y) = x^2y^2$

calcolare le derivata di z rispetto a t . Cioè:

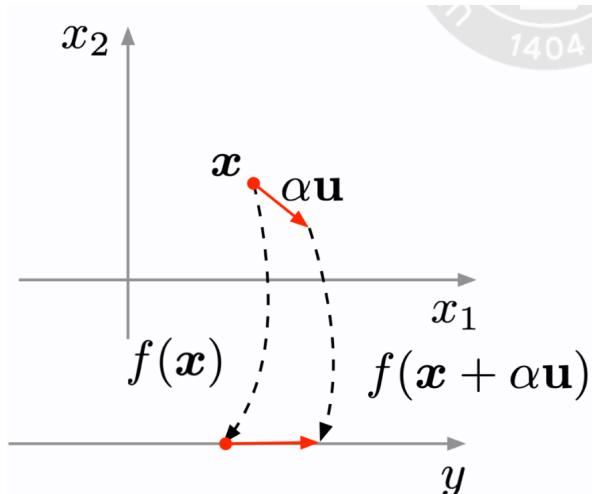
$$\frac{\delta z}{\delta t} = \frac{\delta z}{\delta x} \frac{dx}{dt} + \frac{\delta z}{\delta y} \frac{dy}{dt} = 2xy^2 \cdot 2t + 2x^2 \cdot 1 = 4t^5 + 2t^5 = 6t^5. \quad (1.14)$$

1.2.4 Derivate direzionali

Vediamo ora un'applicazione immediata della regola della catena nel caso multivariato. Fino ad ora abbiamo calcolato la derivata soltanto lungo gli assi ma cosa succede se invece scegliamo di calcolarla lungo un vettore qualsiasi? Qual è il tasso di variazione della funzione rispetto al movimento in una direzione data da un vettore?

Sia u un vettore unità. La derivata direzionale di f nel punto x nella direzione di u è esattamente il **tasso di cambiamento nella direzione indicata dal vettore u** .

$$D_u f(x) = \lim_{h \rightarrow 0} \frac{f(x + hu) - f(x)}{h} \quad (1.15)$$



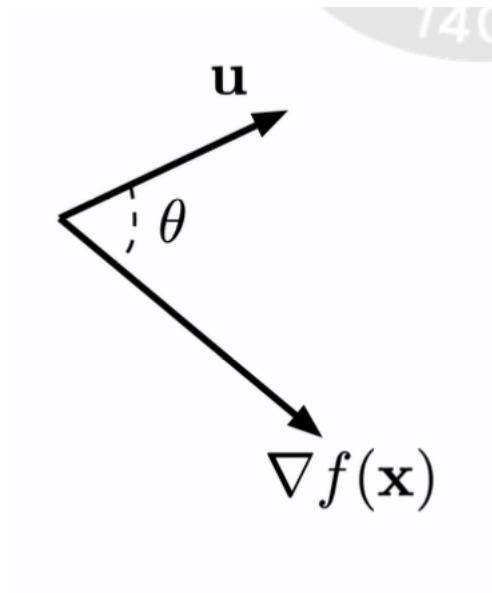
In altre parole, la derivata direzionale è la derivata di $f(x + \alpha u)$ rispetto ad α calcolata quando $\alpha = 0$.

Utilizzando la **chain rule**, possiamo facilmente calcolare un'espressione per $D_u f(x)$:

$$D_u f(x) = \frac{d}{d\alpha} (x + \alpha u) \Big|_{\alpha=0} = \sum_{i=1}^n \frac{\partial f(x + \alpha u)}{\partial x_i} \Big|_{\alpha=0} \frac{dx_i}{d\alpha} = \nabla f(x) \cdot u = u^T \nabla f(x). \quad (1.16)$$

Ci interessa ora **trovare la direzione nella quale la funzione cresce maggiormente**, ovvero trovare u tale che $D_u f$ è maggiore. Vogliamo risolvere:

$$\max_{u, u^T u=1} D_u f(x) = \max_{u, u^T u=1} u^T \nabla f(x) = \max_{u, u^T u=1} |u| |\nabla f(x)| \cos(\theta). \quad (1.17)$$



Ora, visto che $|u| = 1$ e $\nabla f(x)$ non dipende da u , ciò che ci rimane è cercare u tale che massimizzi il $\cos(\theta)$. Questo implica però che **il massimo viene raggiunto quando u ha la stessa direzione di $\nabla f(x)$** .

IMPORTANTE: il gradiente punta nella direzione nella quale f cresce di più.

1.2.5 Matrice Jacobiana

Quando abbiamo una funzione che oltre ad essere multivariata **ha anche molti output diversi**:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m, f(x) = [f(x)_1, \dots, f(x)_m]^T \quad (1.18)$$

se calcoliamo tutte le derivate di un oggetto di questo tipo, stiamo calcolando lo **Jacobiano della funzione** f . Più formalmente essa è la matrice $J \in \mathbb{R}^n \rightarrow \mathbb{R}^m$ che contiene tutte le derivate parziali di $f(x_i)$, ($1 \leq i \leq m$) per tutte le variabili x_j , ($1 \leq j \leq n$):

$$J_{i,j} = \frac{\partial}{\partial x_j} f(x)_i \quad (1.19)$$

o, equivalentemente, lo Jacobiano è la matrice contenente $\nabla[f(x)_i]$ nella riga $i - esima$:

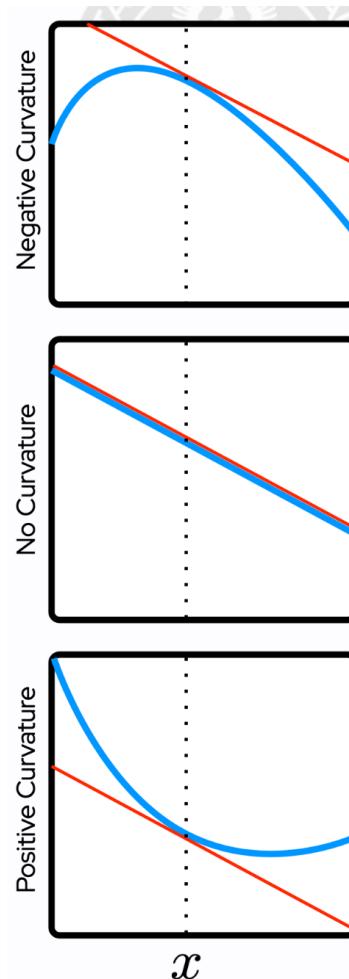
$$J = [\nabla[f(x)_i]^T]_{i=1}^m. \quad (1.20)$$

1.2.6 Derivate seconde

La **derivata seconda** è la derivata di una derivata. Per esempio, sia $\mathbb{R}^n \rightarrow \mathbb{R}$, possiamo calcolare n^2 derivate seconde:

$$\frac{\partial^2}{\partial x_i \partial x_j} f(x). \quad (1.21)$$

La derivata seconda ci dice **come cambia la derivata prima al variare dell'input**. Può anche essere vista come una **misura della curvatura**.



La matrice contenente tutte queste derivate parziali è chiamata **Hessiana** della funzione f .
N.B.: $H(f) = J(\nabla f)$.

Proprietà della matrice Hessiana.

- in tutti i punti in cui le derivate seconde sono continue, gli operatori differenziali sono commutativi, cioè il loro ordine può essere scambiato:

$$\frac{\partial^2}{\partial x_i \partial x_j} f(x) = \frac{\partial^2}{\partial x_j \partial x_i} f(x) \quad (1.22)$$

il che implica che **in quei punti la matrice Hessiana è simmetrica**.

- in maniera intuitiva, possiamo dire che, come la derivata seconda ci aiuta a capire se siamo in un punto di minimo o di massimo (indica la curvatura, quindi se la curvatura è verso l'alto siamo in un minimo, se è verso il basso siamo in un massimo), anche l'hessiano ci aiuta in questo senso. In particolare ci dà questa informazione nel caso di **una funzione multi variata**.

In particolare, quando abbiamo che $\nabla f(x_0) = 0$, **l'hessiano ci aiuta a capire se siamo in un minimo** (hessiano definito positivo¹) o **in un massimo** (hessiano definito negativo). Se l'hessiano non è né definito positivo né definito negativo (abbiamo almeno un autovalore che vale 0):

- **siamo in un punto di sella**, se c'è almeno 1 autovalore positivo e 1 autovalore negativo;
- **il test non è conclusivo**, altrimenti.

¹una matrice si definisce **positiva** quando $\forall z : z^T M z > 0$

1.3 Probabilità

La **probabilità** può essere vista come un'estensione della logica che tratta l'incertezza.

La **logica** fornisce una serie di regole formali per determinare quali proposizioni devono essere vere e quali false, data l'assunzione che un'altro set di proposizioni siano vere o false.

La **teoria delle probabilità** fornisce invece una serie di regole formali per determinare la **likelihood** di una proposizione data la likelihood di altre proposizioni.

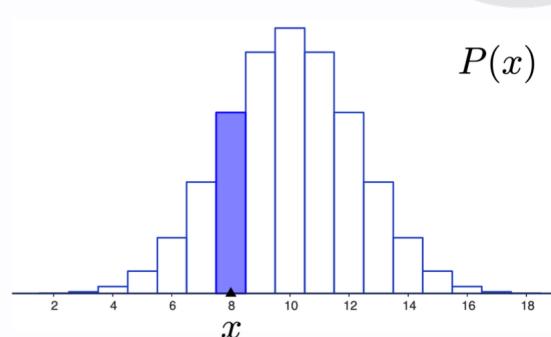
Una **variabile casuale** è una variabile che può assumere randomicamente diversi valori. Possono essere **discrete** o **continue**.

Notazione:

- variabili casuali sono indicate da lettere minuscole semplici come x, t, \dots ;
- i valori assunti dalle variabili sono indicati da lettere minuscole in corsivo come x, y, \dots ;
- il set di tutti i possibili valori assunti dalla variabile casuale x è denotato da Ω_x ;
- a volte scriveremo $x \in x$ per indicare $x \in \Omega_x$;
- per valori che riguardano vettori e loro valori usere il grassetto quindi \mathbf{x}, \mathbf{y} e \mathbf{x}, \mathbf{y} .

Distribuzione di probabilità (Caso discreto) Sia data una **variabile discreta** $x = x_1, \dots, x_n$. La **distribuzione di probabilità di una variabile discreta** può essere indicata usando la **probability mass function (PMF)**. Normalmente essa è indicata con $P(x)$.

La PMF è un mapping che assegna ad ogni valore che può assumere la variabile casuale una probabilità.



Notazione: la probabilità che $P(x = x)$ è denotata normalmente come $P(x)$. La notazione $x \sim P(x)$ è utilizzata per indicare che la variabile casuale x segue la distribuzione descritta da $P(x)$.

Proprietà di una PMF:

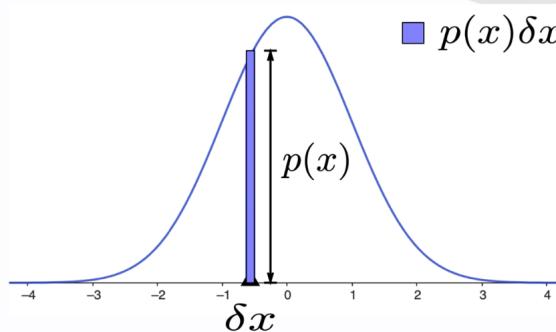
- il dominio di P deve essere l'insieme di tutti i possibili stati di x ;
- $\forall x \in x : 0 \leq P(x) \leq 1$;
- $\sum_{x \in x} P(x) = 1$.

Altre proprietà utili sono:

- $P(S) = \sum_{x \in S} P(x)$;
- $P(S_1 \cup S_2) = P(S_1) + P(S_2) - P(S_1 \cap S_2)$;
- $P(\Omega \setminus S) = 1 - P(S)$

con S, S_1, S_2 insiemi di possibili output; $P(S)$ è una shortcut per $P(x \in S)$ e Ω è l'insieme di tutti i possibili valori.

Distribuzione di probabilità (Caso continuo) In questo caso utilizziamo variabili **continue** e la distribuzione di probabilità non è più descritta dalla PMF ma dalla **probability density function (PDF)**.



La PDF è indicata con p e deve soddisfare le seguenti proprietà:

- il dominio di p deve essere l'insieme di tutti i possibili stati di \mathbf{x} ;
- $\forall x \in \mathbf{x} : p(x) \geq 0$, **notare che non richiediamo** $p(x) \leq 1$;
- $\int p(x) dx = 1$.

1.3.1 Probabilità marginale

A volte si conosce la distribuzione di probabilità di un insieme di variabili ma a noi interessa sapere la distribuzione di probabilità di un sottoinsieme di esse. In questo caso la distribuzione di probabilità è detta **distribuzione marginale** di probabilità.

Le probabilità marginali sono calcolate sommando tutti i valori delle variabili.

Per esempio, immaginiamo di avere 2 variabili casuali discrete x e y con una distribuzione congiunta $P(x, y)$. La **distribuzione marginale** $P(x)$ sarebbe:

$$\forall x \in x P(x) = \sum_y P(x = x, y = y); \quad (1.23)$$

per le variabili continue:

$$p(x) = \int p(x, y) dy. \quad (1.24)$$

| | | y | | | | |
|-----|---|------|------|------|------|------|
| | | A | B | C | D | |
| x | 1 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 0.01 | 0.03 | 0.06 | 0.1 |
| 3 | 0 | 0.03 | 0.13 | 0.14 | 0.3 | |
| 4 | 0 | 0.06 | 0.14 | 0.15 | 0.5 | |
| | | 0 | 0.1 | 0.3 | 0.5 | 1.00 |

$P(x = 2, y = C)$

$P(x = 4)$

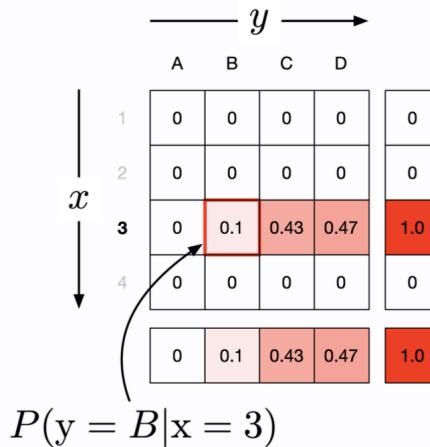
1.3.2 Probabilità condizionata

In molti casi, siamo interessati alla probabilità di un evento dato un altro evento accaduto. Questa è la **probabilità condizionata**. La probabilità condizionata che $y = y$ dato $x = x$ è indicata come

$$P(y = y|x = x). \quad (1.25)$$

Può essere calcolata con la formula:

$$P(y = y|x = x) = \frac{P(y = y, x = x)}{P(x = x)}. \quad (1.26)$$



La cosa interessante da notare è che questa è **una distribuzione completamente nuova su y** .

1.3.3 Chain rule della probabilità condizionata

Qualsiasi distribuzione di probabilità congiunta su più variabili casuali può essere decomposta in prodotti di distribuzioni condizioni di una sola variabile:

$$P(x^{(1)}, \dots, x^{(n)}) = P(x^{(1)}) \prod_{i=2}^n P(x^{(i)}|x^{(1)}, \dots, x^{(i-1)}). \quad (1.27)$$

Esempi:

- $P(a|b,c) = \frac{P(a,b,c)}{P(b,c)} \Rightarrow P(a,b,c) = P(a|b,c)P(b,c);$
- similmente avremo che $P(b,c) = P(b|c)P(c).$

1.3.4 Indipendenza

Due variabili casuali x e y sono **indipendenti** (indicate con $x \perp y$) se la loro distribuzione di probabilità può essere espressa come prodotto delle loro probabilità marginali:

$$\forall x \in X, y \in Y : p(x = x, y = y) = p(x = x)p(y = y). \quad (1.28)$$

Nota:

$$x \perp y \iff \forall x \in X, y \in Y : p(x|y) = p(x) \wedge p(y|x) = p(y). \quad (1.29)$$

Se è vero che le due variabili sono indipendenti valgono anche che:

- $P(x|y) = P(x);$

- $P(y|x) = P(y)$

il che è abbastanza intuitivo, se le due variabili sono indipendenti le loro probabilità non hanno nessun impatto l'una sull'altra. Inoltre, due variabili casuali x e y sono **condizionatamente indipendenti** data una variabile casuale z (scriviamo $x \perp y \mid z$) se la distribuzione di probabilità condizionale su x e y fattorizza in questo modo per ogni valore di z :

$$\forall x \in X, y \in Y, z \in Z : p(x=x, y=y|z=z) = p(x=x|z=z)p(y=y|z=z). \quad (1.30)$$

1.3.5 Expectation (Valore atteso)

Il **valore atteso** (spesso denotato da μ) di una funzione $f(x)$ rispetto alla distribuzione di probabilità $P(x)$ è la media o il valore medio che f assume quando x è tratto da P .

Variabili discrete:

$$\mathbb{E}_{x \sim P}[f(x)] = \sum_x P(x)f(x). \quad (1.31)$$

Variabili continue:

$$\mathbb{E}_{x \sim p}[f(x)] = \int p(x)f(x)dx. \quad (1.32)$$

Nota: il valore atteso è un **operatore lineare**:

$$\mathbb{E}[\alpha f(x) + \beta g(x)] = \alpha \mathbb{E}[f(x)] + \beta \mathbb{E}[g(x)]. \quad (1.33)$$

1.3.6 Varianza e Deviazione standard

Varianza La varianza (spesso denotata con σ^2) restituisce la misura di **quando i valori di una funzione di una variabile casuale x variano rispetto al campionamento di diversi valori di x dalla sua distribuzione di probabilità**:

$$\text{Var}[f(x)] = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2]. \quad (1.34)$$

Esempio: La varianza può anche essere calcolata come $\text{Var}[f(x)] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2$:

$$\text{Var}[f(x)] = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] \quad (1.35)$$

$$= \mathbb{E}[(f(x)^2 - 2f(x)\mathbb{E}[f(x)] + \mathbb{E}[f(x)])^2] \quad (1.36)$$

$$= \mathbb{E}[f(x)^2] - 2\mathbb{E}[f(x)\mathbb{E}[f(x)]] + \mathbb{E}[\mathbb{E}[f(x)]^2] \quad (1.37)$$

$$= \mathbb{E}[f(x)^2] - 2\mathbb{E}[f(x)]\mathbb{E}[f(x)] + \mathbb{E}[f(x)]^2 \quad (1.38)$$

$$= \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2. \quad (1.39)$$

Deviazione Standard. La **deviazione standard** (denotata da σ) è data dalla radice quadrata della varianza.

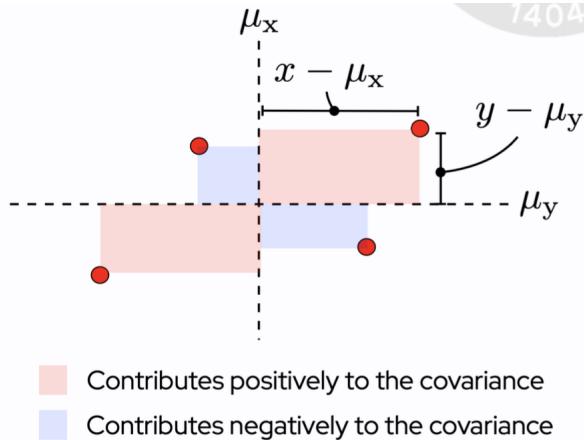
Utile da sapere: per una variabile descritta da una distribuzione normale, circa il 95% dei punti cade in un range di $\mu \pm 2\sigma$.

Covarianza. La **covarianza** è una misura di quanto due variabili casuali siano relazionate l'una con l'altra:

$$\text{Cov}(x,y) = \mathbb{E}_{x,y \sim P(x,y)}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])] \quad (1.40)$$

$$= \mathbb{E}_{x,y \sim P(x,y)}[(x - \mu_x)(y - \mu_y)] \quad (1.41)$$

L'idea è che, se μ_x e μ_y rappresentano gli assi (come in figura), prendendo un punto a caso (per esempio (x,y) , punto rosso in alto a destra) mi posso chiedere in che maniera contribuisca



al calcolo della covarianza. La risposta è che il contributo corrisponde all'area (rossa, nel caso di (x,y)) che sta sotto il rettangolo di cui il nostro punto rappresenta uno dei vertici. Quest'area contribuisce positivamente al calcolo se (x,y) sono entrambe maggiori delle rispettive medie, negativamente altrimenti.

Nota: la covarianza è affetta dalla scala delle variabili.

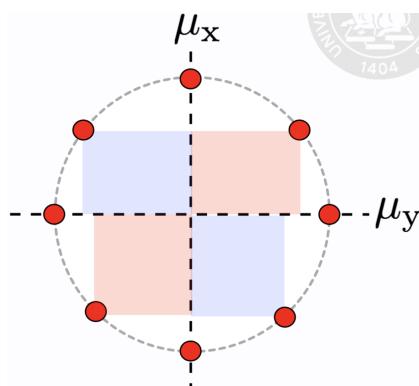
Correlazione. La **correlation** risolve esattamente il problema appena descritto, cioè il fatto che la covarianza sia affetta dalla scala. Essa garantisce che la relazione tra le variabili sia misurata senza che sia influenzata dalle loro magnitudini individuali:

$$\text{Corr}(x, y) = \frac{\text{Cov}(x, y)}{\sigma_x \sigma_y}. \quad (1.42)$$

Il risultato di questa formula restituisce un numero compreso in $[-1, 1]$ ($+1$ le varabili assumono sempre gli stessi valori, -1 le varabili assumono sempre valori opposti) e una correlazione vicina a ± 1 indica una relazione forte tra le variabili mentre una correlazione vicina alle 0 indica che le varibaili **potrebbero essere indipendenti**.

Covarianza vs Dipendenza. Le nozioni di **covarianza** e **dipendenza** sono correlate ma sono **concetti distinti**. Infatti:

- due variabili che sono indipendenti hanno zero covarianza;
- due variabili che hanno covarianza diversa da zero sono dipendenti;
- due variabili possono avere covarianza pari a zero ed essere comunque dipendenti (come mostrato nella figura).



1.3.7 Distribuzioni di Probabilità Comuni

Bernoulli distribution. Si tratta di una distribuzione di una singola variabile casuale **binaria** (semplicemente la distribuzione che modella il lancio di una moneta). E' controllata da un singolo parametro $\phi \in [0, 1]$, il quale restituisce la probabilità che la variabile casuale abbia valore uguale ad 1. Ha le seguenti proprietà:

- $P(x = 1) = \phi;$
- $P(x = 0) = 1 - \phi;$
- $P(x = x) = \phi^x(1 - \phi)^{1-x};$
- $\mathbb{E} = \phi;$
- $\text{Var}(x) = \phi(1 - \phi).$

Multinoulli distribution. E' un'estensione della distribuzione di Bernoulli a più di un risultato. La distribuzione **multinormale** o **categorica** tratta una singola variabile discreta con k differenti stati, dove k è finito.

E' parametrizzata da un vettore $\mathbf{p} \in [0, 1]^k$, con $\mathbf{1}^T \mathbf{p} = 1$ (che indica che gli elementi del vettore p devono sommare ad 1) dove p_i restituisce la probabilità dell' i -esimo stato.

Questo tipo di distribuzione è spesso usata per descrivere valori categorici, quindi normalmente non assumiamo che lo stato 1 ha valore numerico 1. Per questa ragione, **normalmente non abbiamo bisogno di calcolare l'expectation o la varianza** di variabili casuali di distribuzione multinormale.

Binomial distribution. La distribuzione binomiale restituisce **la probabilità di osservare un dato numero di successi ripetendo l'esperimento di Bernoulli**. E' parametrizzata da:

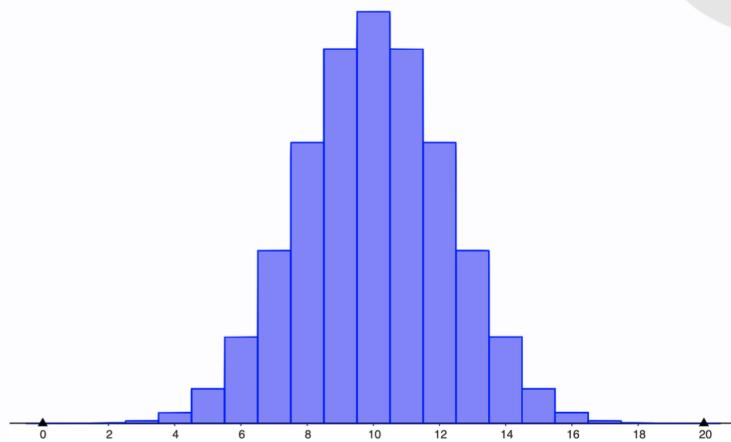
- p : la probabilità di successo dell'esperimento di Bernoulli;
- N : il numero totale di ripetizioni dell'esperimento di Bernoulli.

Se $x \sim Bi(p, N)$, allora:

$$P(x = k) = \binom{N}{k} p^k (1 - p)^{N-k} \quad (1.43)$$

$$\mathbb{E}[x] = Np \quad (1.44)$$

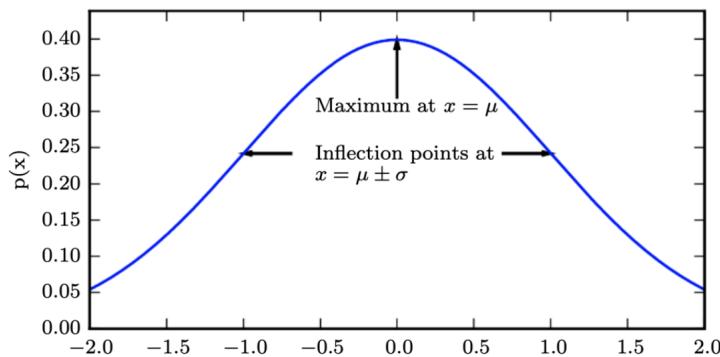
$$\text{Var}[x] = Np(1 - p) \quad (1.45)$$



Gaussian distribution. La distribuzione più utilizzata per i numeri reali è la **distribuzione normale**, anche detta **distribuzione Gaussiana**.

Una Gaussiana è parametrizzata da una media μ e da una varianza σ^2 . Se $x \sim \mathcal{N}(\mu, \sigma^2)$, allora la PDF di x è data da:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \quad (1.46)$$



In realtà molte delle distribuzioni che vogliamo modellare sono molto vicine ad una distribuzione normale.

Teorema del limite centrale. Sia X_1, \dots, X_n una sequenza di variabili casuali indipendenti identicamente distribuite con media μ e varianza σ^2 . La distribuzione della somma $S = \sum_{i=1}^n X_i$ tende ad una distribuzione normale $\mathcal{N}(n\mu, n\sigma^2)$ per n che tende all'infinito. Anche la media $\frac{1}{n}S$ tende ad una distribuzione normale $\mathcal{N}(\mu, \frac{\sigma^2}{n})$ se n tende all'infinito.

Tra tutte le possibili distribuzioni di probabilità con la stessa media e varianza, la distribuzione normale **codifica la massima quantità di incertezza** (cioè è la distribuzione che ha entropia massima). Ciò significa che se dobbiamo studiare un fenomeno di cui non abbiamo informazioni, l'ipotesi che si predilige, facendo meno assunzioni addizionali possibili riguardo il mondo esterno, è che segua la distribuzione normale.

La distribuzione normale si generalizza al caso multivariato \mathbb{R}^n :

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})\right) \quad (1.47)$$

- $\boldsymbol{\mu}$: vettore che denota la media della distribuzione;

- Σ : la matrice di covarianza della distribuzione.

Leggiamo l'equazione:

- $\frac{1}{\sqrt{(2\pi)^n \det(\Sigma)}}$ è in realtà lo stesso fattore che troviamo nella Gaussiana ma che ha al posto di σ^2 il determinante della matrice di covarianza. Come nel caso precedente il fattore serve per riuscire a gestire meglio l'integrale;
- $(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)$ non è altro che un'espressione quadratica, la cui formula generale è $z^T M z$.

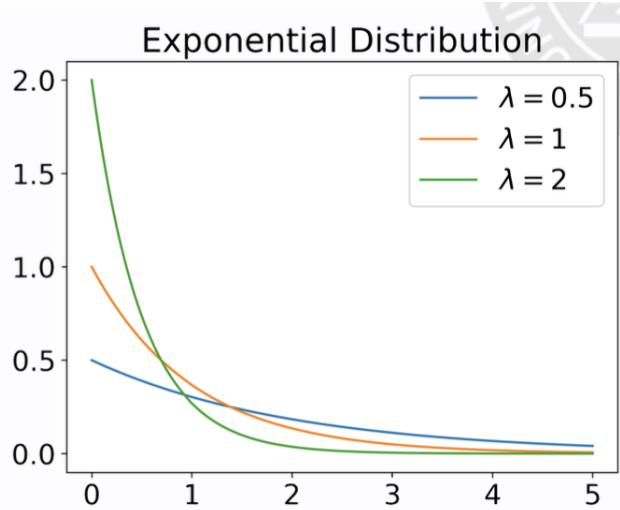
Quindi nella sostanza è abbastanza simile alla formual della Gaussiana. Le matrici di covarianza sono **simmetriche e semi-definite positive** e la loro diagonale principale contiene le varianze.

Se $\Sigma = \sigma^2 \mathbf{I}$, la varianza è la stessa in ogni direzione: **la distribuzione è definita isotropica**.

Exponential e Laplace distributions. Nel contesto del deep learning, spesso vogliamo avere una distribuzione di probabilità con un picco in $x = 0$. Per ottenere questo, possiamo usare la **distribuzione esponenziale**:

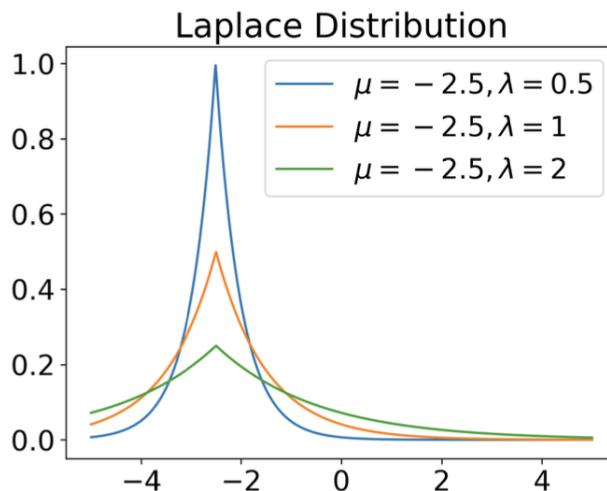
$$p(x; \lambda) = \lambda \exp(-\lambda x), x \geq 0. \quad (1.48)$$

Il parametro λ definisce quanto ripido è il picco.



Una distribuzione di probabilità correlata alla precedente, che ci permette di piazzare un picco di massa di probabilità in un punto arbitrario μ è la **distribuzione di Laplace**:

$$\text{Laplace}(x; \mu, \gamma) = \frac{1}{2\gamma} \exp\left(-\frac{|x - \mu|}{\gamma}\right) \quad (1.49)$$

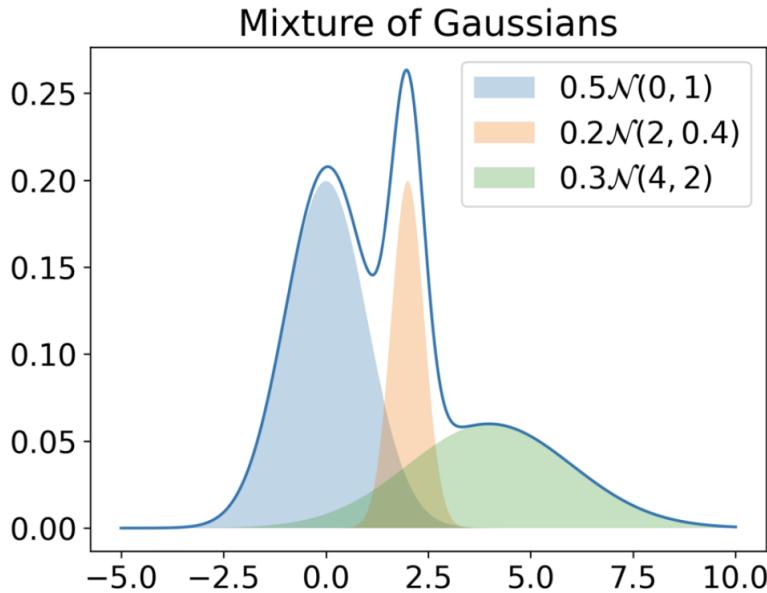


In questo caso γ ha lo stesso ruolo che aveva λ nell'esponenziale mentre, come detto μ definisce in quale punto si trova il picco.

Distribuzioni di tipo Mixture. E' comune anche definire distribuzioni di probabilità a partire dalla combinazione di diverse semplici distribuzioni. Un modo comune di combinare distribuzioni è quello di costruire una **mixture distribution**:

$$P(x) = \sum_i P(c=i)P(x|c=i), \quad (1.50)$$

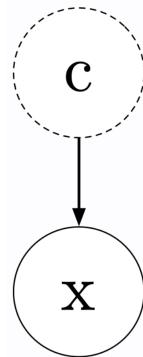
dove $P(c)$ è una distribuzione multinormale sui componenti della mistura.



Regola di Bayes. Ci troviamo spesso in situazioni dove conosciamo $P(y|x)$ e ci interessa conoscere $P(x|y)$. Fortunatamente, se conosciamo anche $P(x)$ e $P(y)$, possiamo calcolare $P(x|y)$ usando la **regola di Bayes**:

$$P(x|y) = \frac{P(x)P(y|x)}{P(y)}. \quad (1.51)$$

Variabili latenti. I mixture model ci permettono di accennare un concetto che avrà molta importanza in futuro: le **variabili latenti**.



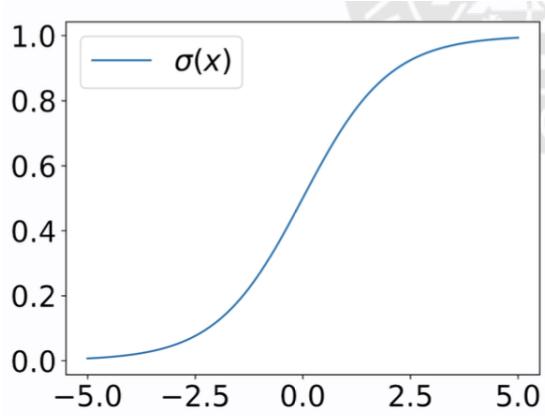
Una variabile latente è una variabile casuale la quale **non può essere osservata direttamente** e che governa la distribuzione che stiamo osservando. Spesso assumere che ci sia una variabile latente ci permette di modellare e gestire meglio i problemi. La variabile identità della componente **c** del mixture model ne fornisce un esempio.

Le variabili latenti potrebbero essere relazionate ad **x** attraverso la distribuzione congiunta, in questo caso, $P(x,c) = P(x|c)P(c)$.

1.3.8 Proprietà utili delle funzioni comuni

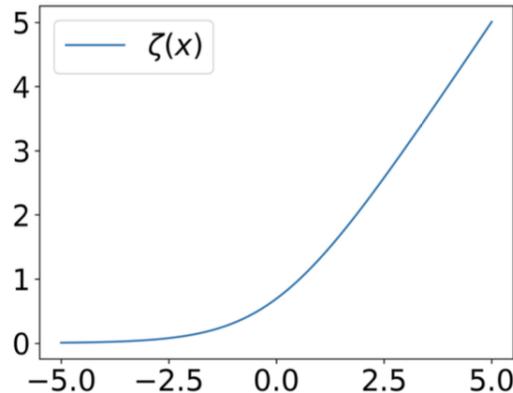
Sigmoid. E' spesso usata per produrre il parametro ϕ della distribuzione di Bernoulli. E' denotata con σ e definita come:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \quad (1.52)$$



Softplus. Una versione **smooth** di $x^+ = \max(0, x)$, è denotata con ζ e definita come:

$$\zeta(x) = \log(1 + \exp(x)). \quad (1.53)$$



Proprietà utili delle funzioni comuni.

- la sigmoide **si satura** per valori grandi (positivi e negativi) di x ;
- $\sigma(x) = \frac{\exp(x)}{\exp(x)+\exp(0)}$;
- $\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$;
- $1 - \sigma(x) = \sigma(-x)$;
- $\log \sigma(x) = -\zeta(-x)$;
- $\frac{d}{dx}\zeta(x) = \sigma(x)$;
- $\forall x \in (0, 1) : \sigma^{-1}(x) = \log(\frac{x}{1-x})$;

- $\forall x > 0 : \zeta^{-1}(x) = \log(\exp(x) - 1);$
- $\zeta(x) = \int_{-\infty}^x \sigma(y)dy;$
- $\zeta(x) - \zeta(-x) = x.$

1.3.9 Measure Theory

Una corretta comprensione formale delle variabili casuali continue e della probability density function richiede uno sviluppo della teoria delle probabilità attraverso un ramo della matematica conosciuto come **measure theory** o teoria della misura.

Senza di essa, potremmo ritrovarci in situazioni paradossali come:

è possibile costruire due insiemi S_1 e S_2 , con $S_1 \cap S_2 = \emptyset$ tali che $p(x \in S_1) + p(x \in S_2) > 1$.

Questi paradossi spesso riguardano la costruzione di insiemi *molto esotici*, come insiemi frattali o che derivano da trasformazioni di numeri razionali, ma la possibilità esiste.

Uno dei contributi chiave di questa è il fatto che fornisca un framework per caratterizzare insiemi in cui le probabilità possono essere calcolate in maniera consistente, evitando paradossi.

La measure theory fornisce un modo rigoroso per descrivere che **un insieme di punti è trascurabilmente piccolo**. In questo caso diremo che l'insieme ha **measure zero**, o misura zero.

Esempio: Una linea in \mathbb{R}^2 ha measure zero.

Qualsiasi **unione di insiemi numerabili** che abbiano measure zero ha anch'essa measure zero (quindi l'insieme di tutti i numeri razionali ha measure zero).

Quando una proprietà vale in tutto lo spazio tranne che per un punto in un insieme di misura zero, diremo che quella proprietà vale **quasi ovunque**.

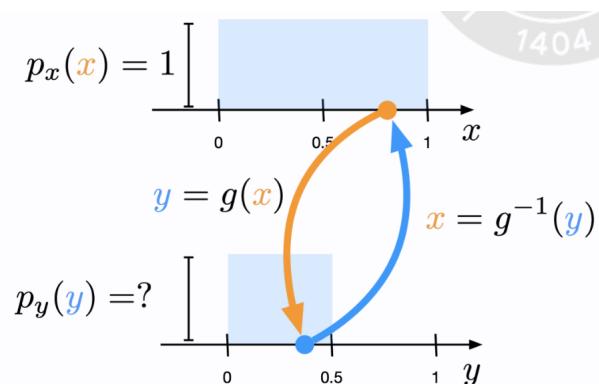
Dettagli tecnici delle variabili continue. Trattiamo ora le variabili casuali continue che sono **funzioni deterministiche di altre variabili**.

Supponiamo di avere due variabili casuali, x e y , tali che $y=g(x)$, dove g è una trasformazione differenziabile, invertibile e continua.

Sfortunatamente: $p_y(y) \neq p_x(g^{-1}(y))$.

Sia $y = \frac{x}{2}$ e $x \sim U(0, 1)$. In questo caso abbiamo:

- $y = g(x) = \frac{x}{2}$;
- $x = g^{-1}(y) = 2y$.

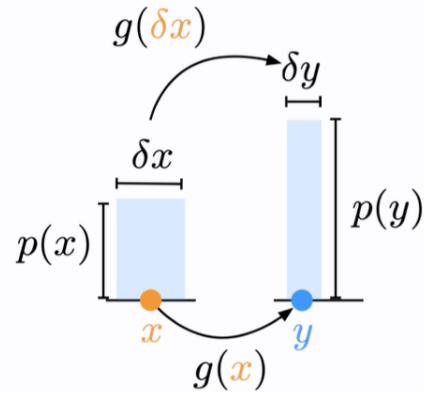


Ci aspetteremo che $p_y(y) = p_x(2y)$, ma non è questo il caso. Infatti, se così fosse, $p_y(y)$ sarebbe uguale a zero dappertutto ad eccezione dell'intervallo $[0, \frac{1}{2}]$ dove sarebbe 1. Invece:

$$\int_{-\infty}^{\infty} p_y(y) dy = \int_0^{0.5} 1 dy = x \Big|_0^{0.5} = 0.5 - 0 = 0.5 \quad (1.54)$$

Il problema con questo approccio è che fallisce nel tenere conto della distorsione dello spazio introdotta dalla funzione g .

La probabilità che x si trovi in una regione infinitesimalmente piccola con volume δx è data da $p(x)\delta x$.



Siccome g può espandere o contrarre lo spazio, il volume infinitesimale che circonda x nello spazio x potrebbe avere un volume differente nello spazio y .

Per correggere il problema abbiamo bisogno di preservare la proprietà:

$$|p_y(g(x))dy| = |p_x(x)dx| \quad (1.55)$$

che porta a:

$$p_y(g(x))|dy| = p_x(x)|dx| \Rightarrow p_x(x) = p_y(g(x)) \left| \frac{d}{dx}g(x) \right| \quad (1.56)$$

o, equivalentemente:

$$p_y(y)|dy| = p_x(g^{-1}(y))|dx| \Rightarrow p_y(y) = p_x(g^{-1}(y)) \left| \frac{d}{dy}g(y) \right|. \quad (1.57)$$

Nel nostro esempio:

$$p_y(y) = p_x(g^{-1}(y)) \left| \frac{\partial g^{-1}(y)}{\partial y} \right| = p_x(2y) \left| \frac{d}{dy}2y \right| = 1 \cdot 2 = 2 \quad (1.58)$$

e

$$\int_0^{0.5} p_y(y)dy = \int_0^{0.5} 2dy = 1. \quad (1.59)$$

In **dimensioni maggiori** $g : \mathbb{R}^m \rightarrow \mathbb{R}^n$, la derivata si generalizza alla matrice jacobiana e il valore assoluto al valore assoluto del determinante:

$$p_x(x) = p_y(g(x))|\det(J)| \quad (1.60)$$

dove la jacobiana J è tale è $J_{ij} = \frac{\partial g(x)_i}{\partial x_j}$.

1.4 Information theory

L'**information theory** è una branca della matematica applicata che ruota attorno alla quantificazione della quantità di informazioni presenti in un segnale.

In questo corso, useremo alcune delle idee chiave della information theory per **caratterizzare distribuzioni di probabilità o misurare la similarità tra distribuzioni di probabilità**.

1.4.1 Quantificare l'informazione

L'intuizione base dietro l'information theory è il fatto che la quantità di informazione trasportata da un messaggio dipende da quanto esso sia probabile: sapere che un **evento improbabile** sia accaduto è **più informativo** di sapere che un evento probabile sia accaduto.

Esempio:

Sapere che oggi ha piovuto nel deserto del Sahara è più informativo di sapere che oggi ha piovuto a Londra.

Volendo formalizzare questa intuizione:

- un evento con probabilità del 100% è **assolutamente non sorprendente e non produce informazioni**;
- **meno l'evento è probabile** più è sorprendente e **più produce informazioni**;
- **eventi indipendenti dovrebbero produrre informazioni aggiuntive**. Per esempio, scoprire che una moneta lanciata ha dato testa due volte dovrebbe fornire il doppio delle informazioni rispetto a scoprire che una moneta lanciata ha dato testa una volta.

Misura dell'entropia di Shannon. Possiamo quantificare l'incertezza di un evento utilizzando il concetto di *self-information measure*:

$$I(x) = -\log P(x) \quad (1.61)$$

la quale equazione gode delle proprietà descritte precedentemente:

- se un evento ha probabilità del 100% allora $\log 1 = 0$ e non produce informazioni;
- più la probabilità dell'evento si abbassa, più produce informazione. Infatti il logaritmo di un numero vicino allo 0 tende a $-\infty$.

L'**entropia di Shannon** è definita come il **valore atteso della self-information measure**, in maniera più formale cattura la quantità media di "informazioni" su tutti i possibili risultati di una variabile casuale:

$$H(x) = E_{x \sim P}[I(x)] = - \sum_x P(x)[\log(P(x))]. \quad (1.62)$$

Il **Shannon's Source Coding Theorem** afferma che $H(x)$ fornisce un confine inferiore per la lunghezza media della parola da utilizzare per ottenere una codifica ottimale di tutti i possibili valori di x .

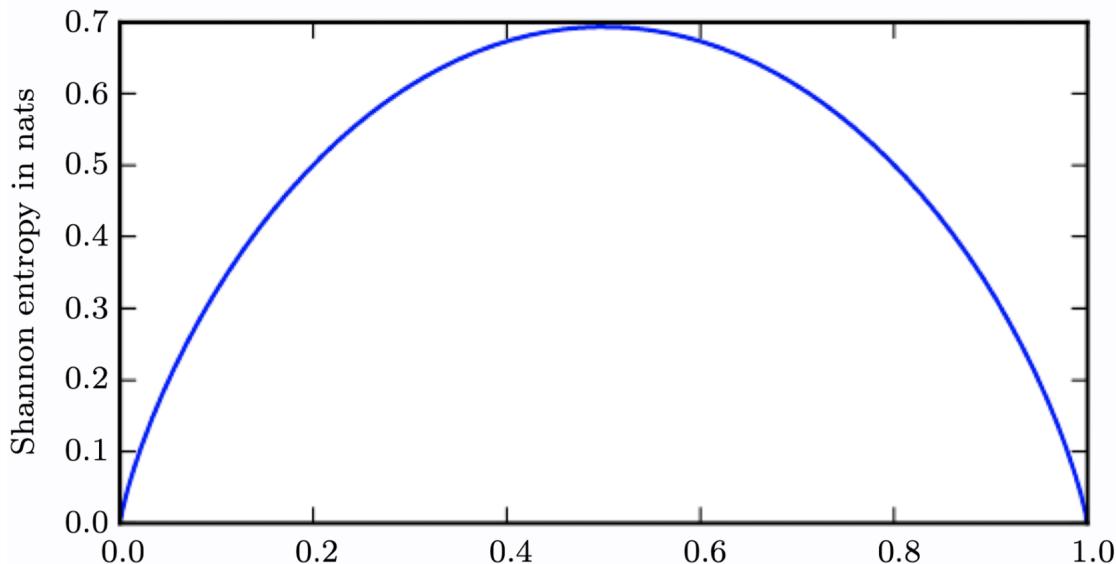


Figura 1.1: L'entropia di una variabile casuale $x \sim \text{Bernoulli}(\phi)$ al variare di ϕ tra 0 e 1.

La divergenza Kullback-Leibler(KL). Se abbiamo due distribuzioni separate $P(x)$ e $Q(x)$ della stessa variabile x , possiamo misurare quanto siano differenti l'una dall'altra utilizzando la **Kullback-Leibler divergence**:

$$D_{\text{KL}}(P \| Q) = E_{x \sim P} \left[\log \frac{P(x)}{Q(x)} \right] = E_{x \sim P} [\log P(x) - \log Q(x)]. \quad (1.63)$$

Nel caso di variabili discrete, rappresenta la **quantità extra di informazioni** di cui si necessita per inviare un messaggio contenente simboli estratti dalla distribuzione di probabilità P , quando usiamo codice il cui design è pensato per minimizzare la lunghezza di messaggi estratti dalla distribuzione di probabilità Q .

Proprietà della KL divergence.

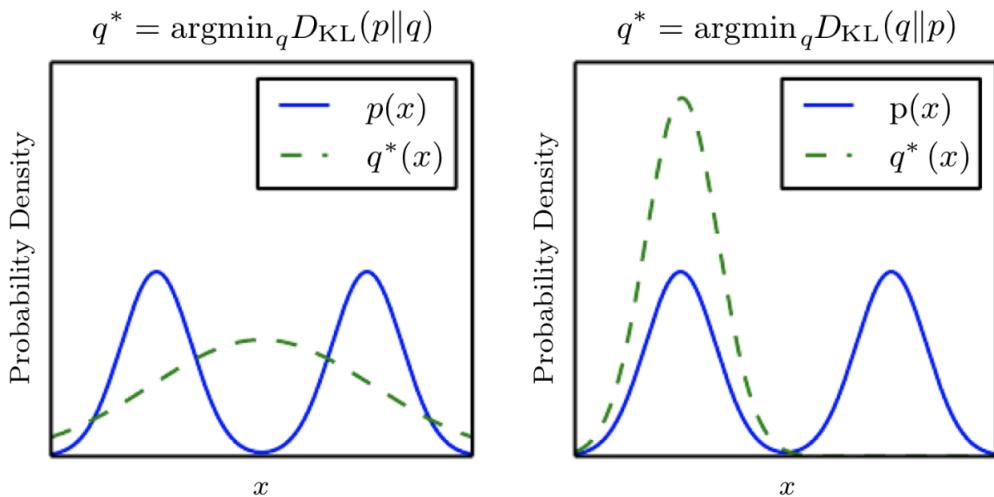
- la KL divergence è sempre **non negativa**;
- la KL divergence è $0 \iff P$ e Q sono la stessa distribuzione (o sono uguali *quasi in ogni punto* nel caso di variabili continue);
- la KL divergence **non è una misura di distanza**: una distanza dovrebbe essere simmetrica e soddisfare la disegualanza triangola, ma la KL divergence non lo fa.

Quest'ultimo punto, in particolare il fatto che non sia simmetrica, ha importanti conseguenze quando bisogna minimizzare la **distanza** tra le due distribuzioni P e Q .

Minimizzare la KL divergence. Minimizzare $D_{\text{KL}}(p\|q)$ può essere molto diverso da minimizzare $D_{\text{KL}}(q\|p)$. Assumiamo che:

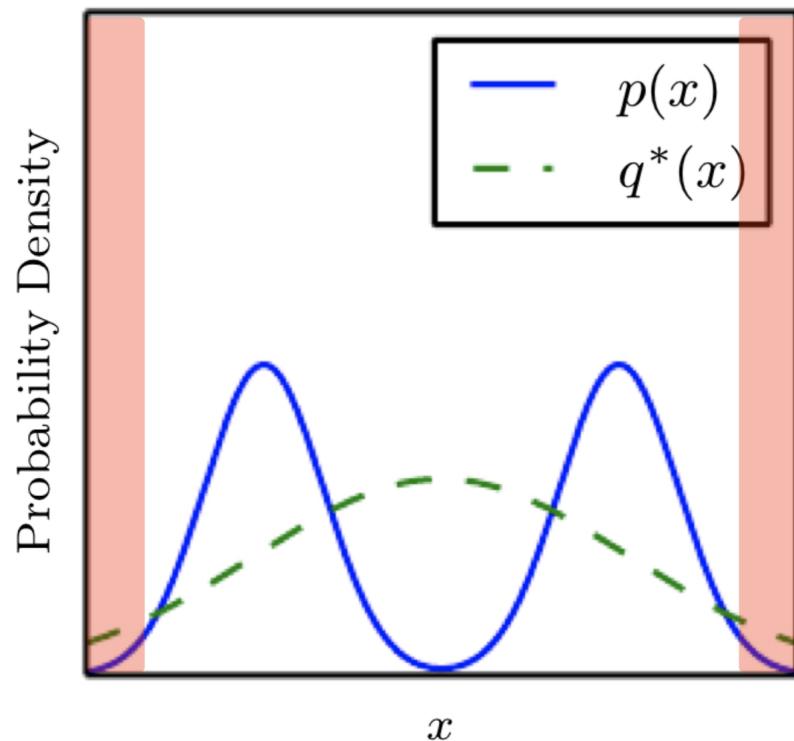
- p sia una mistura di due Gaussiane con 2 diverse mode;
- q sia una singola Gaussiana che vogliamo **ottimizzare** in modo che matchi p nel modo migliore possibile.

$$D_{\text{KL}}(p\|q) = E_{x \sim p} \left[\log \frac{p(x)}{q(x)} \right]. \quad (1.64)$$

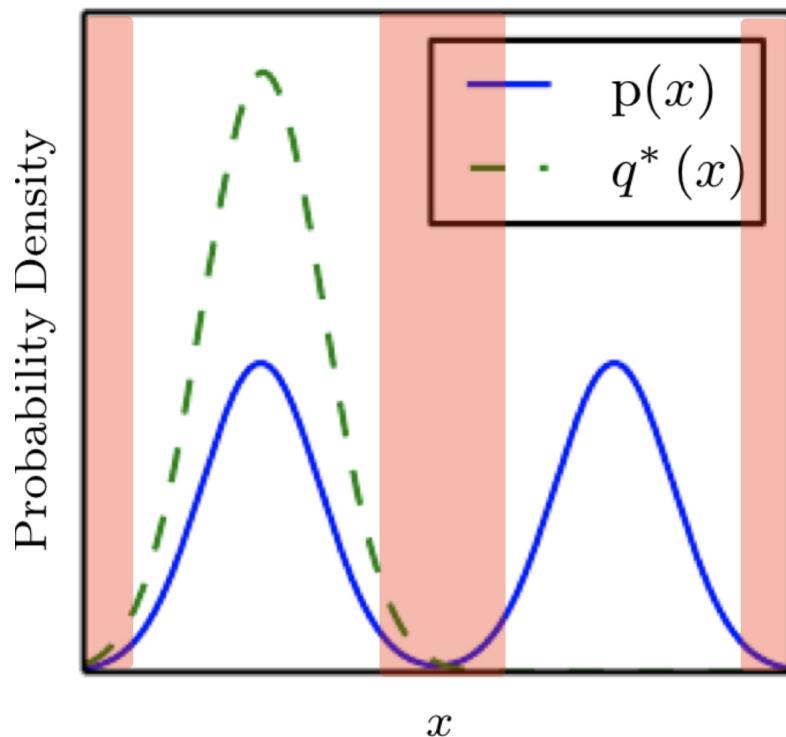


$$D_{\text{KL}}(q\|p) = E_{x \sim q} \left[\log \frac{q(x)}{p(x)} \right]. \quad (1.65)$$

$$\min_q [D_{\text{KL}}(p\|q)] = \min_q \left[\mathbb{E}_{x \sim p} \left[\log \frac{p(x)}{q(x)} \right] \right] \quad (1.66)$$



$$\min_q [D_{\text{KL}}(q\|p)] = \min_q \left[\mathbb{E}_{x \sim q} \left[\log \frac{q(x)}{p(x)} \right] \right] \quad (1.67)$$



Cross Entropy. Una quantità che è strettamente correlata alla KL divergence è la **cross-entropy**:

$$H(P, Q) = H(P) + D_{\text{KL}}(P\|Q) = -E_{x \sim P}[\log Q(x)] \quad (1.68)$$

cioè la cross entropy è il numero medio di bit necessari per codicare un messaggio del codice Q con un codice pensato per P .

Note:

- è simile a $D_{\text{KL}}(P\|Q) = E_{x \sim P}[\log P(x) - \log Q(x)]$:
 - espressioni simili ma manca il termine $\log Q(x)$;
 - concettualmente differente è il fatto che D_{KL} misuri il numero di bit extra previsto mentre H misuri il numero totale di bit.
- minimizzare $H(P, Q)$ rispetto a Q è la stessa cosa di minimizzare $D_{\text{KL}}(P\|Q)$.

Il motivo si vede bene guardando il fattore $H(P) + D_{\text{KL}}(P\|Q)$; infatti quando minimizziamo P e Q lo facciamo rispetto alla distribuzione Q , in questo fattore si vede bene che c'è un termine, $H(P)$, che non dipende dalla distribuzione Q , e il secondo termine, che è esattamente la KL, che invece dipende da Q . Perciò sostanzialmente stiamo andando a minimizzare la KL anche quando minimizziamo la cross-entropy. **Il punto è che spesso è più semplice minimizzare la cross-entropy perchè ha una forma più semplice rispetto alla KL.**

1.4.2 Modelli grafici (modelli probabilistici strutturati)

Spesso le distribuzioni di probabilità possono essere divise in molti fattori. Per esempio, assumiamo che un variabile casuale a influenzi il valore di un'altra variabile b e che questa a sua volta influenzi c , ma sono indipendenti l'una rispetto all'altra. Possiamo rappresentare l'intera distribuzione come segue:

$$p(a, b, c) = p(a)p(b|a)p(c|b). \quad (1.69)$$

Questi tipi di fattorizzazione possono **ridurre enormemente il numero di parametri** necessari per descrivere una distribuzione.

Esempio:

Assumiamo che a, b, c assumano tutti valori in $\{1, 2, 3, 4, 5\}$. Per descrivere completamente le probabilità coinvolte dovremmo specificare $5^3 = 125$ valori di probabilità.

Se la distribuzione fattorizza come sopra abbiamo bisogno solo di:

- 5 parametri per descrivere $p(a)$;
- 25 parametri per descrivere $p(b|a)$;
- 25 parametri per descrivere $p(c|b)$.

Modelli grafici. Le fattorizzazioni sulle distribuzioni possono essere visivamente descritte usando i grafi.

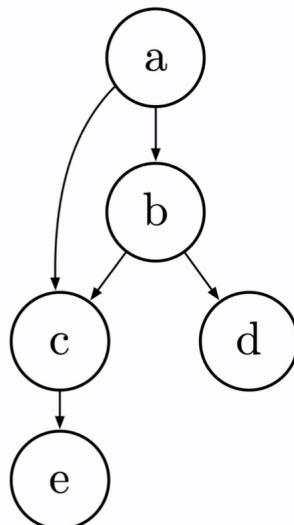
Modelli diretti. Utilizzano grafi con archi diretti. Rappresentano le fattorizzazioni in distribuzioni di probabilità condizionate:

- un fattore per ogni variabile casuale x_i ;
- il fattore consiste nella distribuzione condizionata di x_i dato dai genitori.

$$p(\mathbf{x}) = \prod_i p(x_i | Pa_{\mathcal{G}}(x_i)) \quad (1.70)$$

dove $Pa_{\mathcal{G}}(x_i)$ è l'insieme dei genitori di x_i .

Example



$$p(a, b, c, d, e) = p(a)p(b|a)p(c|a, b)p(d|b)p(e|c)$$

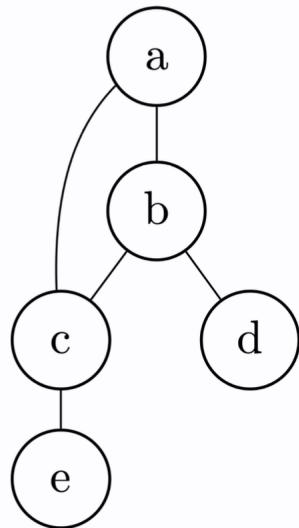
Modelli indiretti. Utilizzano grafi con archi indiretti. Rappresentano la fattorizzazione utilizzando un insieme di funzioni (non necessario né comune che siano distribuzioni probabilistiche):

- c'è un fattore ϕ^i per ogni clique² $\mathcal{C}^{(i)}$ del grafo;
- la fattorizzazione è il prodotto normalizzato di tutti i fattori.

$$p(\mathbf{x}) = \frac{1}{Z} \prod_i \phi^{(i)}(\mathcal{C}^{(i)}), \quad (1.71)$$

con $Z = \sum_{x \in \mathbf{x}} \prod_i \phi^{(i)}(\mathcal{C}^{(i)})$ fattore di normalizzazione.

Example



$$p(a, b, c, d, e) = \frac{1}{Z} \phi^{(1)}(a, b, c) \phi^{(2)}(b, d) \phi^{(3)}(c, e)$$

²sottografo completamente connesso.

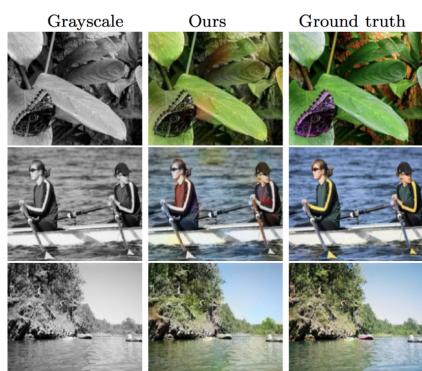
Capitolo 2

Autoencoders

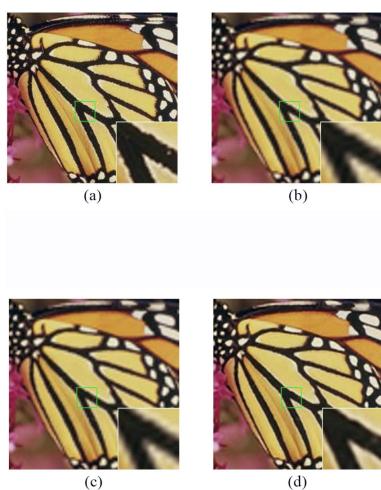
Gli autoencoder (AE) sono strumenti di apprendimento non supervisionato **utilizzati per migliorare le reti supervisionate** (ad esempio, utilizzando livelli di rappresentazione AE appresi per inizializzare la rete supervisionata) e per **risolvere molti compiti interessanti**:

- image colorization;
- increase resolution;
- image inpainting;
- machine translation.

Storicamente, sono state utilizzate, per esempio, per **ricolorare immagini**. L'input è la grayscale, e l'output è l'immagine ricolorata.



Un altro esempio di applicazione è dato dalla **increase resolution**, nel seguito applicata ad immagini.



Oppure, ancora, **image inpainting**: l'input è un'immagine con background noise e l'obiettivo è rimuoverlo.



2.1 Definizione

Un autoencoder è una rete naturale addestrata a tentare di copiare il suo input nel suo output tramite una rappresentazione h costruita dai suoi hidden layers. Se chiamiamo h la **rappresentazione** e x l'input, le componenti principali di un autoencoder sono:

- **encoder** $h = f(x)$;
- **decoder** $r = g(x)$.

L'obiettivo dell'encoder è di costruire una nuova rappresentazione dell'input x , quello del decoder è invece riprodurre l'input x . Può sembrare un processo stupido, miriamo a riprodurre l'input, ma ciò che è importante è la rappresentazione. Nella nostra trattazione chiamiamo f la parte dell'encoder, g la parte del decoder, h la rappresentazione di x e \hat{x} la rappresentazione ricostruita di x . Non è previsto che un autoencoder copi fedelmente ogni input nel suo output. Sono invece costretti a dare la priorità a quali aspetti dell'input dovrebbero essere preservati.

Tradizionalmente gli autoencoder sono stati utilizzati per la **riduzione della dimensionalità** o **l'apprendimento delle funzionalità**. Per quanto riguarda la dimensionality reduction, ad esempio si parte con un encoder che riceve una x di 1000 feature e si costruisce una rappresentazione che ne ha 100. Il punto è che l'autoencoder è pensato per essere in grado di riprodurre l'input, quindi la nuova rappresentazione, molto più piccola dell'originale, contiene comunque tutte le informazioni dell'originale, perchè è stata costruita a partire dalle informazioni utili di essa.

Oggi sono la prima linea della ricerca sui **modelli generativi** grazie alle connessioni stabilite con i **latent variable models**.

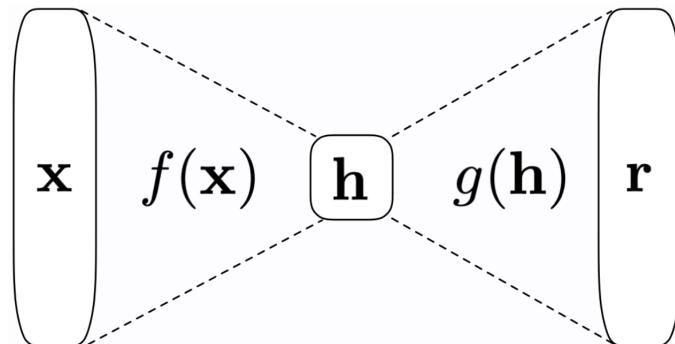
Nota: un **latent variable models** è un modello che ha sia variabili osservate x che variabili latenti h e il cui obiettivo è apprendere la distribuzione $p(x, h)$. Abbiamo introdotto brevemente queste idee alla fine delle lezioni sull'apprendimento della rappresentazione.

2.2 Undercomplete Autoencoders

Copiare semplicemente l'input nell'output corrisponde alla costruzione della funzione identità, cosa che evidentemente non è utile. Quindi la speranza è che durante il training si riesca ad imparare una qualche proprietà della hidden representation h che sia utile.

Uno dei modi per ottenere una h interessante è avere una h con dimensioni più piccole di x .

Definizione. Un autoencoder la cui dimensione della codifica è inferiore alla dimensione dell'input è chiamato **Undercomplete Autoencoders**.



L'idea di questo primo tipo di autoencoder, l'undercomplete autoencoder è di utilizzare un solo modo di regolarizzare il risultato, cioè avere una hidden rappresentazione più piccola rispetto ai dati originali. La ricostruzione di x è chiamata a volte \hat{x} , a volte r .

Normalmente, l'idea alla base del learning process di un undercomplete autoencoder è la minimizzazione di una loss function:

$$L(x, g(f(x))), \quad (2.1)$$

dove L è una loss function che penalizza $g(f(x))$ se è troppo diversa da x (ad esempio, la MSE). In altre parole, usiamo una loss che prova a minimizzare la distanza tra l'input x e l'output $g(f(x))$, cioè l'esempio ricostruito dall'autoencoder.

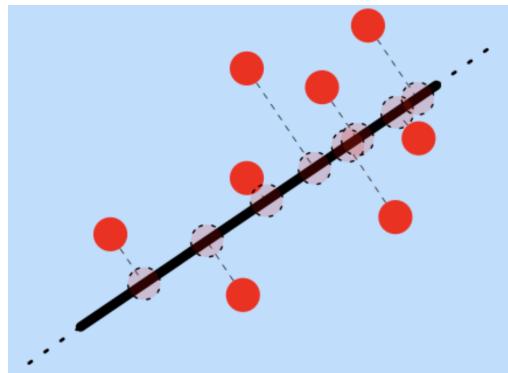
2.2.1 Autoencoders e PCA

Esiste una connessione interessante tra undercomplete autoencoders e la PCA (tecnica di dimensionality reduction). Quando il decoder è lineare e L è la MSE, un undercomplete autoencoder impara ad estendersi nello stesso sottospazio della PCA. Per capire meglio questo concetto, torniamo ricordiamo cosa è la PCA.

Sommario della PCA. La Principal Component Analysis può essere derivata da un algoritmo che proietta ogni esempio x in uno spazio vettoriale di dimensione inferiore con errore di ricostruzione minimo, cioè, per la PCA:

$$f(x) = \arg \min_h \|x - g(h)\|_2, \quad (2.2)$$

dove g è una funzione di ricostruzione soggetta a **qualche costante**.



In altre parole, l'idea è prendere i punti del dimensional space e proiettarli in uno spazio di dimensionalità ridotta rispetto al primo. L'obiettivo della PCA è costruire una funzione f che prenda in input x e restituisca la sua rappresentazione nel nuovo spazio. $f(x)$ esplora tutte le possibili rappresentazioni di h e sceglie quella che si avvicina di più ad x . Nel processo, viene anche fissata qualche costante.

Nello specifico, g deve essere tale che :

- g sia **lineare**, vogliamo avere ad esempio $g(h) = Dh$, per qualche D , cioè che $g(h)$ sia il prodotto matriciale Dh ;
- le colonne di D siano **ortogonali l'una con l'altra** (per semplificare il problema);
- le colonne di D abbiano **norma unitaria** (per avere soluzioni uniche).

Con queste costanti, possiamo dimostrare che:

- $f(x) = \arg \min_h \|x - g(h)\|_2 = D^T x = h_x$;
- $g(h) = Dh$,

che portano a:

- $r(x) = g(f(x)) = g(h_x) = DD^T x$,

con D dato dagli autovalori corrispondenti all'**autovettore più grande** della matrice $X^T X$.

In sintesi:

- un undercomplete autoencoder minimizza una loss tale che $L(x, g(f(x)))$;
- la PCA trova il mapping $f(x) = \arg \min_h \|x - g(h)\|_2$, imponendo che g sia un modello lineare.

Dovrebbe essere evidente che quando $L = \|\cdot\|_2$ e il layer di ricostruzione è modellato da unità lineari, abbiamo due approcci per risolvere lo stesso problema.

2.2.2 Manyfold

Manyfold è traducibile come **varietà**: consideriamo uno spazio con tante dimensioni, diciamo tridimensionale; se consideriamo solo due dimensioni, quei punti sono ancora nello spazio tridimensionale ma sono anche in un sottospazio che ha una dimensione di meno. Pensiamo ad una sfera e ad un piano, in questo caso il piano è un manyfold bidimensionale immerso in uno spazio tridimensionale (la sfera).

2.3 Regularized Autoencoders

Anche in questo caso, viene utilizzata la size di h come regularization factor. Ora però, la differenza tra h e l'input x determina quanto l'autoencoder è forzato ad imparare solo le **parti più rilevanti dell'input**.

Se h è troppo grande, l'autoencoder imparerà semplicemente la funzione identità.

E se invece h ha dimensione 1? Siamo sicuri di non stare costruendo semplice la funzione identità?

Infatti se l'autoencoder ha troppa capacità (o perché l'encoder e il decoder sono troppo potenti, o perché il codice è troppo grande), non sarà in grado di apprendere nulla di utile.

Esamineremo ora gli **regularized autoencoders**: invece di limitare la capacità del modello mantenendo l'encoder e il decoder poco profondi e la dimensione degli hidden layer piccola, **si utilizza una loss function che incoraggia il modello ad avere altre proprietà oltre a quella di copiare l'input nell'output**.

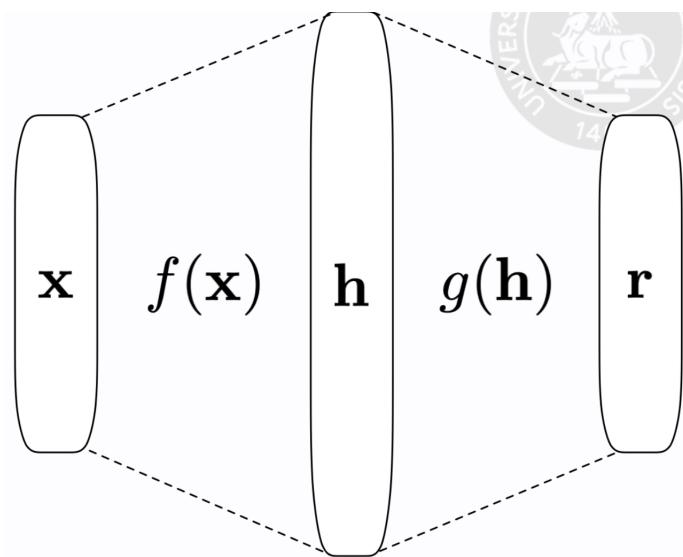
Proprietà fruttate per la regolarizzazione.

- sparsità della rappresentazione \Rightarrow **sparse autoencoders**;
- robustezza al rumore o mancanza di input \Rightarrow **denoising autoencoders**;
- derivata piccola \Rightarrow **contractive autoencoders**.

2.3.1 Sparse Autoencoders

Definizione. Uno **sparse autoencoder** è semplicemente un autoencoder i cui criteri di allenamento includono una **penalità di scarsità** $\Omega(h)$ su layer di coding h , in aggiunta al reconstruction error:

$$L(x, g(f(x))) + \Omega(h). \quad (2.3)$$



2.3.2 Generative Models (GM)

Esiste una connessione tra sparse autoencoders e generative models. Un **generative model** è un modello che cerca di apprendere la distribuzione congiunta $p(x, y)$ dei dati. Dopo che il modello è stato addestrato, può generare nuovi punti di una determinata classe, campionando da essa. A volte y non è una variabile osservabile. In questi casi, y è chiamata **variabile latente** e solitamente è indicata con h .

I generative models differiscono dai **modelli discriminativi**, poiché essi imparano ad approssimare la probabilità condizionata $p(y|x)$ e non possono quindi generare nuovi punti dati.

Supponiamo di avere un modello generativo con x variabili visibili e h variabili latenti, con una distribuzione:

$$p_{model}(h, x) = p_{model}(h)p_{model}(x|h), \quad (2.4)$$

dove $p_{model}(h)$ è la **distribuzione a priori del modello sulle variabili latenti**.

Sotto questi presupposti, la **likelihood di x dato il modello** può essere calcolata come la marginalizzazione delle variabili latenti h :

$$p_{model}(x) = \sum_h p_{model}(h, x). \quad (2.5)$$

Importante: $p_{model}(h)$ è un **prior differente** rispetto a quello normalmente assunto nei modelli probabilistici. Di solito il termine **prior** indica la preferenza di un algoritmo di apprendimento per una data soluzione (cioè per un dato modello) (**prior**) prima di vedere i dati. Qui stiamo usando questo termine per **riferirci alla preferenza del modello per la rappresentazione latente h prima di vedere x** .

Prendendo il logaritmo di $p_{model}(h)$ otteniamo la **log-likelihood** di x nel modello:

$$\log p_{model}(x) = \log \sum_h p_{model}(h, x). \quad (2.6)$$

Possiamo pensare all'autoencoder come un'**approssimazione di tale somma con una stima puntuale per un solo valore altamente probabile di h** .

Poi data una \tilde{h} generata dall'encoder:

$$\log p_{model}(x) = \log \sum_h p_{model}(h, x) \approx \log p(\tilde{h}, x) = \log p_{model}(\tilde{h}) + \log p_{model}(x|\tilde{h}). \quad (2.7)$$

Ricordando ora che **gli sparse autoencoder sono allenati a minimizzare la loss**:

$$L(x, g(f(x))) + \Omega(h), \quad (2.8)$$

(dove il primo termine è la reconstruction loss e il secondo è la sparsity loss), possiamo vedere che, minimizzando la **reconstruction loss**, l'autoencoder sta imparando ad **approssimare la log-likelihood dei dati nel modello**.

Infatti, minimizzando la reconstruction loss, l'autoencoder sta imparando ad approssimare la probabilità condizionata $p_{model}(x|h)$ poiché sta cercando di trovare x che è la migliore ricostruzione dato h , cioè sta effettivamente risolvendo:

$$\arg \max_p p_{model}(x|h) = \arg \min_x -\log p_{model}(x|h). \quad (2.9)$$

Se $\Omega(h) = \lambda \sum_i |h_i|$ (cioè usando la norma L_1 di h , **la quale incoraggia la sparsità**), la minimizzazione del termine di sparsità corrisponde alla massimizzazione della log-likelihood del termine $p(h)$ assumendo una Laplace prior su ogni componente di h , indipendentemente. Formalmente, se:

$$p_{model}(h_i) = \frac{\lambda}{2} e^{-\lambda|h_i|}, \quad (2.10)$$

allora

$$-\log p_{model}(h) = \sum_i \left(\lambda|h_i| - \log \frac{\lambda}{2} \right) = \Omega(h) + \text{const}, \quad (2.11)$$

dove λ può essere trattata come un iper parametro e il termine costante, dipendente solo da λ , non viene ottimizzato durante l'apprendimento e può essere ignorato.

Mettendo tutto insieme, allenare la rete a minimizzare il reconstruction error e la sparsity penalty equivale ad allenare la rete a minimizzare la log-likelihood negativa dei dati del modello:

$$\min L(x, g(f(x))) + \Omega(h) \approx \min -\log p_{model}(x). \quad (2.12)$$

Da questo punto di vista, la sparsity penalty non è un fattore regolarizzante: è solo la conseguenza della modellazione della joint distribution tenendo conto delle variabili latenti e assumendo che esse abbiano la Laplace prior.

Questo punto di vista fornisce diverse motivazioni per il training di sparse autoencoder:

- è un modo per addestrare approssimativamente un modello generativo;
- le feature imparate sono utili perché descrivono le variabili latenti che spiegano l'input.

Designing output units Progettare le output units ci fornisce una connessione per giustificare la definizione dei prossimi tipi di autoencoders. Nel seguito giustificheremo la scelta di utilizzare particolari output units.

Una strategia generale per progettare le **output units e la loss function** di una rete feed-forward è di interpretare la rete come il calcolo di una distribuzione $p(y|x)$ e la minimizzazione della log-likelihood negativa $-\log p(y|x)$.

Nel caso degli autoencoder, x è anche il target, oltre ad essere l'input. In ogni caso, possiamo sfruttare la stessa idea.

Potremmo vedere il decoder come il fornitore della distribuzione condizionata $p_{decoder}(x|h)$.

Nel caso in cui x sia un valore reale, **adottare linear output units e una loss quadratica** corrisponderebbe a massimizzare $p_{decoder}(x|h)$ sotto l'assunzione che la probabilità si distribuisce come $\mathcal{N}(x; \hat{x}, \mathbf{I})$, con \hat{x} fornita dal linear layer output: $\hat{x} = Wh$. Cominciamo ricordando che la probabilità di una distribuzione normale multivariata, di parametri μ e Σ , è data da:

$$p(x) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp \left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right), \quad (2.13)$$

che porta a

$$p_{decoder}(x|h) = c_1 \exp \left(- (x - Wh)^T \mathbf{I} (x - Wh) \right) = c_1 \exp -\|x - Wh\|^2, \quad (2.14)$$

implicando che minimizzare la loss quadratica equivale a massimizzare la log-likelihood dei dati del modello, siccome:

$$-\log p_{decoder}(x|h) = \|x - Wh\|^2 + c. \quad (2.15)$$

Nota: visto che le linear units non saturano, non creano nessun problema agli algoritmi di ottimizzazione basati su gradienti.

slide 30: x è un vettore di valori reali, se adottiamo linear units e quadratic loss è uguale ad assumere che la prob distr di $x-h$ è una normal distr. dove mean is e variance is id matrix

slide 31: qui parliamo di x contenenti valori booleani 0 1, possiamo mostrare che possiamo assumere che

slide 32: possiamo guardare l'Autoencoders come, l'encoder come modelling a distribution di $h-x$ e l'encoder come $x-h$. questo esempio mostra che fare questo tipo di modelling da qualche suggerimento. l'encoder può essere pensato come abbiamo detto e il decoder uguale, il punto è che non c'è motivo

slide 33: una ragione per parlare di questo tipo è che si può dimostrare che l'encoding e il decoder corrispondono alle prob di prima ma sono coerenti tra loro e sono due facce dello stesso modello, cosa che prima non era garantita ma ora lo è. è un risultato teorico che non dimostriamo.

slide 35: l'idea è che l'unica cosa che cambia da prima

slide 36:

slide 37:

slide 38:

slide 39:

slide 40:

slide 41:

slide 42: possono essere completamente descritte da.. se pensiamo ad un punto nel manifold è un punto tangente al piano. il punto è che una proprietà interessante teoricamente tangenti ai punti ti danno la direzione nell'higher dimensional space in cui si può viaggiare di più senza lasciare il manifold

slide 43:

slide 44:

slide 45:

slide 46:

slide 47: l'idea dietro i denoising Autoencoders è intuitiva, qua no. a contractive Autoencoders è un regularised Autoencoder che provoca a minimizzare una loss function di questo tipo (guarda forma nella slide) la forma particolare di questa regularisation term è dove c'è lambda. quello che dice è che puoi pensare ad x come una funzione da x a h dove h è un vettore di componenti h_1, h_2, \dots, h_k e f è un output visto come una funzione che ha molti input e molti output. f è composta quindi dall'unione di k differenti funzioni che costituiscono h_1, h_2, \dots, h_k . abbiamo il gradient rispetto ad x di ogni termine h. il regularization term ci dice che vuole tenere tutte le derivate parziali piccole.

slide 48: se abbiamo un lambda molto alto, qual è la soluzione? quale tipo di funzione andremo ad imparare? è un piano (?) costante

slide 49: qua diciamo che possiamo riscrivere lo stesso oggetto omega in un altro modo

slide 50: se guardiamo la variazione nello spazio originale e in quello embedded il modello cercherà di mappare le variazioni nello spazio originale in piccole variazioni in quello embedded. per capire perché guardiamo le due funzioni nella slide. il denoising serve per avere funzioni da plottare semplicemente. se prendiamo un range di valori nel space, quando la derivata è piccola, come nella quadratica funzione, per lo stesso range se guardiamo l'output, nel caso della quadratica il range di valori è molto più piccolo di quello della cubica.

slide 51: menzioniamo solo il fatto che queste proprietà contrattive valgono solo localmente. quindi se si fanno un certo numero di passaggi possiamo avere due punti distanti di più nell'embedding space che nello spazio originale.

slide 52: c'è un paper che mostra che se assumiamo uno small gaussian input noise, allora la reconstruction è equivalente ad una constraint penalty on the reconstructive function. la differenza è che vedere la slide.

slide 53: già detto

slide 54: da qualsiasi direzione dello spazio calcoliamo il prodotto tra il jacobiano e la direzione abbiamo un prodotto grande quando la direzione è allineata alla direzione in cui la h cambia di più. queste direzioni sono quelle che vengono penalizzate dalla penalità contrattiva. quindi le direzioni che sopravvivono sono quelle lungo il manifolds. rimangono quelle in cui cresce di più perchè sono quelle che più descrivono i dati.

slide 55: un variational Autoencoders regolarizzano la encoding distribution e ciò consiglia rendere un Autoencoders buono come generative model. se proviamo a ottimizzare così, è troppo difficile da fare quindi assumiamo che la distribuzione abbia una particolare forma facile da ottimizzare. a quel punto cerchiamo di approssimare quanto meglio possibile.

slide 56: partiamo da perché abbiamo bisogno di vari Autoencoders, focus sui simple Autoencoders, e proviamo ad usarlo come generative model. durante il training facciamo tutto come al solito, x , costruisamo h e ricostruiamo x nel last layer del decoder. per usare questa architettura buttiamo via l'encoding part e cominciamo a generare ogni vettore randomicamente. poi cerchiamo di ricostruire l'input da questi random vector.

slide 57: questo succede. a sinistra abbiamo le ricostruzioni dei mnist digit fatti da un Autoencoders quando le vengono forniti gli h . a destra abbiamo il risultato dell'estrazione dei h vettori random. perchè succede?

slide 58: il punto è che non c'è niente nel Autoencoders definition data fino ad ora che forzi l'embedding space ad essere compatto e ad avere un significato. l'idea è di regolarizzare la distribuzione dei punti nell'embedding space così da compattarli.

slide 59: l'idea principale è che abbiamo gli usual Autoencoders, quindi metà Autoencoders function, g che funziona come decoder function. la novità è che f non vuole direttamente h , builda un vettore di 2 pezzi che specifica la media e la varianza di una distribuzione normale. il punto è che startiamo con l'immagine, abbiamo una mu e una sigma associata ad ogni image e l'idea è che nell'embedding space tutti i punti vicini a questa media e varianza corrispondano a qualche tipo di.. quindi sampling da una normal distribution.

slide 60:

slide 61: rimane un problema intrattabile anche con queste assunzioni.

slide 62:

slide 63: dobbiamo ancora fare qualche operazione. non sappiamo ancora come ottimizzare g , la decoder function. per ottimizzare g proviamo ad ottimizzare $p(x|h)$, quindi trovare argmax di expectation rispetto alle h distribuzioni come $q(x|h)$ del log di $p(x|h)$. il log di $p()$ è il termine usato prima, possiamo quindi mettere tutto insieme nel problema di ottimizzazione. k non dipende da g , ma il primo termine sì. cambiamo il meno in più così abbiamo un problema di minimizzazione. è una regularize network con il reconstruct term e il regularization term.

slide 64: abbiamo ancora il problema che nella nostra idea originale dobbiamo fare sampling per generare h . il problema è come differenziarli. la soluzione è la reparametrization trick. a gaussiana con parametri μ e σ + è esattamente la stessa di una random normal da cui facciamo sampling e poi riscaliamo. non estraiamo dalla prima normale, che dipendono da σ e μ che dipendono dalla nostra rete, ma dalla seconda normale 0 e 1, riscalando e ottengo una serie di dipendenze che permette di derivare h e μ e σ continuando la backpropagation. non riesco a calcolare le derivate della parte che contiene il sampling ma non ci interessa quella parte della rete.

slide 65:

slide 66:

slide 67:

slide 68:

slide 69:

slide 70: