Auto-Encoders

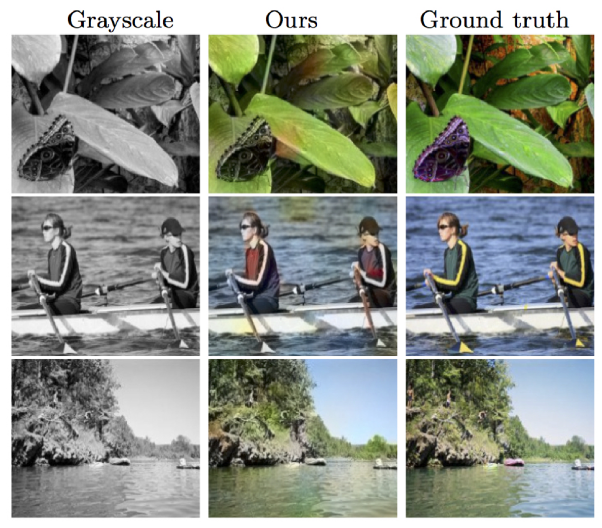
# Applicazioni

Gli auto-encoders sono degli strumenti non supervisionati. Uno degli utilizzi principali alla base del loro sviluppo è come metodo per migliorare le performance delle reti supervisionate: quando c’erano ancora difficoltà nell'addestrare le reti profonde, addestrare degli auto-encoders a livelli erano utilizzati come primo passo per addestrare le reti, era una tecnica che dava dei frutti e permetteva di superare queste limitazioni tecniche.

Gli auto-encoders hanno trovato applicazione in numerosi campi applicativi: colorazione delle immagini, super-resolution, image inpainting, traduzione automatica (machine translation) e molte altre.

## Colorazione delle immagini

Dato in input un immagine in scala di grigi, ne costruisce la sua versione colorata. Non si è ancora raggiunta la perfezione ma si ottengono dei risultati molto buoni. Questa tecnica è utilizzata molto nel cinema per riportare a colori immagini d’epoca.

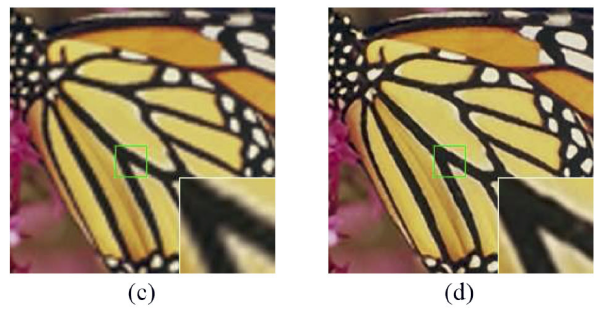
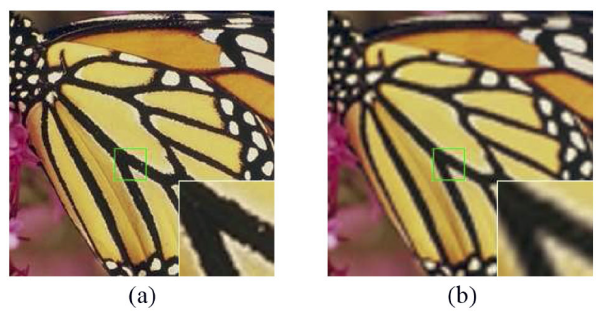


## Super-resolution

Tecniche per incrementare la risoluzione delle immagini senza perdere qualità dovuta ad artefatti, come lo smoothing gaussiano dei pixel.

E’ molto importante questa applicazione perché ci sono reti che permettono di inviare delle immagini a bassa risoluzione che poi dovranno essere ricostruite in alta risoluzione senza perdere qualità.

Un’altra importante applicazione è nei videogiochi che girano a 4K: problema di supportare un frame rate molto alto con così tanti pixel da muovere sullo schermo. Ciò che si fa è simulare tutto ad una risoluzione più alta e poi si fa upscaling soltanto nel momento in cui viene fatto il rendering a video dell’immagine.



## Image inpainting

Tecnica che permette di rimuovere artefatti dall’immagine che la rovinano. In altre parole, permette di recuperare l’immagine originale dopo essere stata alterata in qualche modo.

Esempio: l’immagine è ricoperta da tante scritte in bianco → la rete ha un risultato molto buono nel rimuovere le scritte.



# Definizione di Auto-Encoders

Un'auto-encoder è una rete neurale addestrata per cercare di copiare sul proprio output l’input presentato.

Si chiama auto-encoders perché è un modo per ricodificare il proprio input.

E’ composto da due parti

* **Encoder**: codifica l’input in una rappresentazione h
* **Decoder**: prende la rappresentazione h e ricostruisce l’input

L’auto-encoder ha successo quando r è ricostruito molto simile a x.

**Non ci aspettiamo che l’auto-encoder copi fedelmente l’input sull’output ma si vuole** **forzare l’auto-encoder, nel momento in cui costruisce la rappresentazione h, a cogliere gli aspetti salienti dell’input x ed eliminare gli aspetti non importanti**.

In molti auto-encoder, h è una versione compressa dell’input x che viene ricostruito tramite la funzione g.

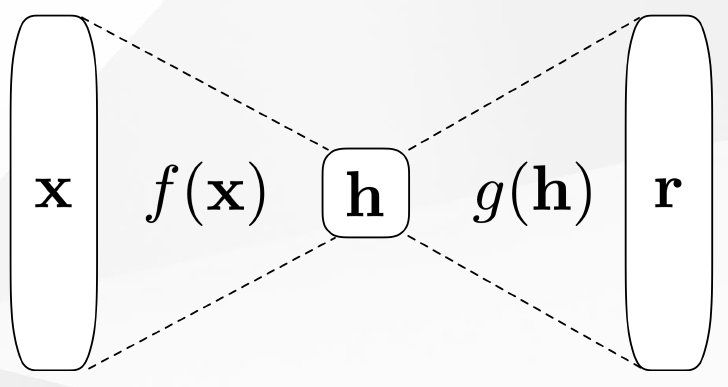
# Storia

Storicamente gli auto-encoders sono stati sviluppati per ridurre la dimensionalità dell’input oppure per fare apprendimento di features combinato con algoritmo di apprendimento supervisionato per migliorare le sue performance.

Oggi, gli auto-encoders sono usati come base dei modelli generativi. Vedremo che ci sono connessioni sostanziali con i modelli generativi in quanto h può essere usata per recuperare le variabili latenti del modello generativo (vedremo più avanti un cenno).

# Undercomplete Auto-Encoders

La rappresentazione h da apprendere ha una dimensionalità più piccola rispetto all’input x.

Fig.1: Undercomplete Autoencoder

Definizione

Un’auto-encoder è detto **undercomplete** (“sottodimensionato”) se la dimensione della rappresentazione h è più piccola della dimensione dell’input x.

L’idea di fondo è non copiare x in r ma forzare l’auto-encoder di apprendere le caratteristiche più importanti dell’input. Quando si usano gli auto-encoders undercomplete, questo viene fatto riducendo la dimensione di h in modo tale che l’autoencoder è forzato a comprimere l’informazione presente nell’input x, di conseguenza è costretto a tralasciare alcuni aspetti meno importanti di x.

Nel mondo migliore possibile, la distribuzione di x vive su un manifold a bassa dimensionalità e h riesce a cogliere tutto il manifold ignorando le direzioni nello spazio che non sono interessanti per modellare x.

## Apprendimento

Si addestra una rete feed-forward (Fig.1) che parte da una x larga e converge verso una dimensione h più piccola, mediante le tecniche di **propagazione dell’errore (backprop)** usando una **funzione loss**:



L è una loss che penalizza la differenza tra l’input x e la ricostruzione r mediante g(f(x)).

Tutte le volte che g(f(x)) è diverso da x, la loss penalizza l’auto-encoder e grazie alla backprop (discesa del gradiente) si apprendono i pesi che minimizzano la loss.

# Auto-Encoders e PCA

**Quando il decoder è lineare (funzione di attivazione lineare) e la funzione loss L è l’errore quadratico medio (MSE), allora un’auto-encoder undercomplete risolve lo stesso problema che risolve anche il PCA**.

Come effetto laterale del processo di apprendimento, l’auto-encoder prende in considerazione il sottospazio principale dei dati di apprendimento, trova il sottospazio in cui sono presenti le componenti principali dei dati.

## Principal Component Analysis (PCA)

<https://it.wikipedia.org/wiki/Analisi_delle_componenti_principali>

PCA può essere derivata come un algoritmo che proietta un esempio x in un vettore c a più basse dimensioni, minimizzando l’errore di ricostruzione.

Errore di ricostruzione:



La funzione f(x) può essere pensata come la funzione loss L usata negli auto-encoders.

La funzione di ricostruzione g è soggetta ai seguenti vincoli:

* **g è lineare**



l’input è moltiplicato per una matrice D

* **le colonne di D sono ortogonali le une con le altre** → **per semplificare il problema**. senza questo vincolo, il numero di soluzioni cresce e non si riesce a trovare una soluzione in forma chiusa
* **le colonne di D hanno norma unitaria** → **per rendere la soluzione unica**. Senza questo vincolo, moltiplicando le colonne per tante costanti diverse, si ottengono gli stessi vettori ma di lunghezza leggermente diversa. I vettori descrivono lo stesso spazio nello stesso modo dei vettori unitari.

Sotto le condizioni appena descritte, si dimostra che la soluzione è in forma chiusa, in particolare:



ed essendo e allora



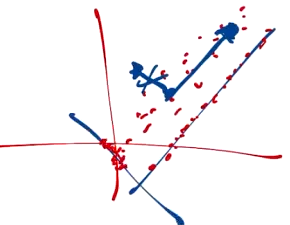
e che la matrice D è data dagli autovettori che corrispondono ai più grandi autovalori della matrice 

**Nota bene**:

Ritroviamo la PCA. Di seguito è illustrata brevemente la procedura PCA:

* Data la matrice X, si calcola la matrice di covarianza come assumendo che a X ho già tolto la media dei vettori.
* Si calcolano i valori degli autovettori.
* Si costruisce la funzione una volta che sono stati selezionati i vettori con gli autovalori più grandi, ovvero i vettori lungo cui i dati sono più dispersi

Se si sta cercando di rinunciare a una delle direzioni per ridurre la dimensionalità, bisogna rinunciare alla direzione più “corta” in quanto si riesce ad ottenere un’approssimazione migliore.



Riassumendo:

L’auto-encoder di tipo undercomplete minimizza la funzione loss



La PCA fa la stessa cosa usando



e imponendo che g è lineare.

Se si setta la funzione di loss alla norma 2



e il livello di ricostruzione sia composto da unità lineari, allora si hanno due approcci che risolvono lo stesso problema.

Non è detto che si arrivi alla stessa soluzione, però si analizza il problema considerando lo stesso spazio della PCA.

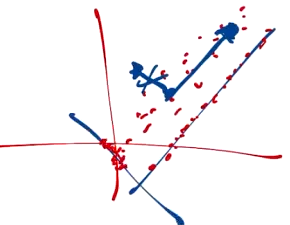
## Domanda

Si assume che i dati da ricostruire vivono su un manifold lineare.

Conviene usare gli Auto-Encoders oppure PCA e per quale motivo?

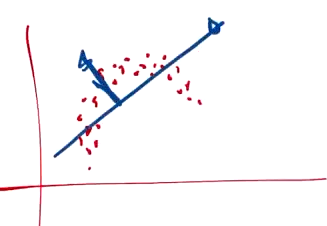
E se lo spazio fosse altamente non lineare?

Nel caso di un manifold lineare, gli auto-encoder vanno bene perché come visto nel seguente caso

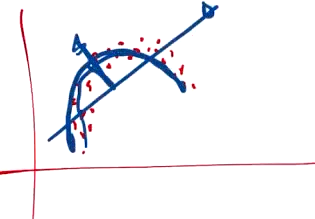


tutta la variabilità è spiegata lungo direzioni lineari (non curvano). Questo è anche il caso ideale per la PCA. Gli Auto-Encoders con la giusta configurazione approssimano la PCA.

Nel caso invece di manifold altamente non lineare, la PCA fallisce perché apprenderebbe

 e non riuscirebbe ad apprendere la curva.

Vogliamo che apprenda questo

 approssimando la curva

La PCA è definita da vincoli più stringenti: loss quadratica e funzione di ricostruzione lineare.

Nel caso degli Auto-Encoders, si ha una definizione più generale: le funzioni loss e di ricostruzione possono essere una qualsiasi.

Gli auto-encoders se usati bene con la giusta architettura e se addestrati bene, permettono di risolvere casi che non possono essere risolti da PCA, come il problema della non linearità.

**Qualità del risultato**

Nel caso lineare sono equivalenti PCA e Auto-Encoders.

Nel caso di manifold non lineare, la PCA non è in grado di cogliere la curva del manifold. Invece gli Auto-Encoders si adattano molto bene.

**Performance**

Dipende dal problema.

La PCA ha il vantaggio di avere una soluzione in forma chiusa: algoritmi molto ottimizzati per cercare la soluzione. Quindi si preferisce se il problema è piccolo, PCA è molto veloce e trova soluzione ottimale.

Se il problema è molto grosso, calcolare gli autovettori di matrici molto grandi potrebbe essere molto difficoltoso, quindi l’uso di Auto-Encoders addestrati con backprop che approssimano numericamente il risultato della PCA (discesa del gradiente della loss che minimizza la PCA), gli auto-encoders sono molto vantaggiosi in quanto **non** costruiscono  e **non** calcolano gli autovettori/autovalori in modo preciso.

# Regularized Auto-Encoders

La differenza in dimensione tra la rappresentazione h (c in PCA) e l’input x determina quanto l’auto-encoder è forzato per apprendere solo la parte più rilevante dell’input.

Se la dimensione di h è troppo grande, l’effetto di regolarizzazione (forzatura) si viene a perdere quindi l’auto-encoder potrebbe semplicemente apprendere la funzione identità.

|  |
| --- |
| **Domanda**  **Se la dimensione di h è molto grande, l’auto-encoder può apprendere la funzione identità.**  **Cosa succede se la dimensione di h è pari a 1?**  **(quindi h è rappresentato con un unico valore numerico).**  h deve essere piccolo per forzare l’auto-encoder ad imparare qualcosa di utile su x.  In teoria, se si ha una funzione altamente non lineare, posso far passare la funzione su quanti punti voglio. Soltanto con un indice numerico posso ricostruire ciò che voglio.    Nell’esempio, associo ad ogni numero una immagine. Questa associazione deve essere memorizzata da qualche parte.  In pratica questo non succede, è molto difficile riprodurre una cosa di questo tipo. Un valore piccolo di h è sufficiente per imparare qualcosa di utile.  **Per costruire un buon auto-encoder, non è sufficiente pensare solo alle dimensioni di h, ma bisogna pensare anche in modo attento a cosa metto nella rete neurale che forma un auto-encoder. Se rete profonda con tante non linearità, ottengo overfitting**. Devo calibrare molto bene queste scelte.    Un input x è mappato su un unico neurone e poi rimappato sulla ricostruzione r. Tra il neurone e la ricostruzione r, posso avere una rete neurale qualsiasi. Ad esempio, una rete neurale che parte dal neurone e usa milioni di pesi per arrivare alla ricostruzione. Per memorizzare il dataset si usa tutta questa capacità e per indicizzare si usa il valore del neurone come indice del dataset.  Se l’input è la Monnalisa e il neurone ha valore 1, imparo ad attivare le parti della rete neurale che corrispondono alla Monnalisa.  Nella parte di ricostruzione ho molta capacità in termini di neuroni e non linearità per riuscire a memorizzare l’input, il fatto che h è molto piccolo è indifferente. |

Se l’auto-encoder ha troppa capacità nella ricostruzione (o perché encoder e decoder sono troppo potenti o perché h è troppo grande), non riesce ad apprendere qualcosa di interessante.

In questi casi, volendo mantenere un'architettura in cui h non è più piccola dell’input x, un modo per ovviare a questo fatto è introdurre un **termine di regolarizzazione :** al posto di limitare la capacità del modello rendendo h più piccolo di x, si utilizza una funzione di loss che incoraggia il modello ad avere altre proprietà. Il termine di normalizzazione **vincola l’auto-encoder ad apprendere qualcosa di interessante**.

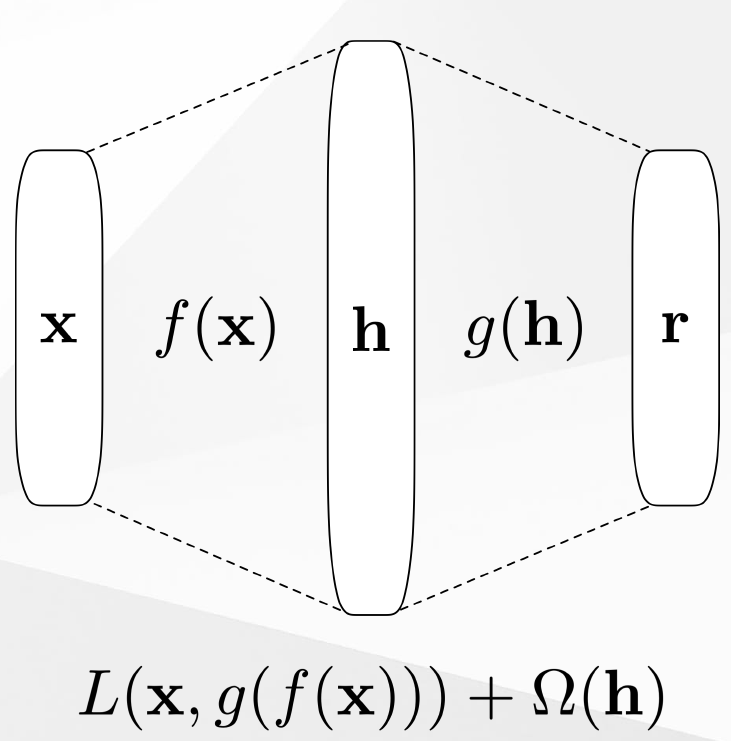
Esistono 3 varietà di auto-encoders regolarizzati

* **Sparse Auto-encoders**: inducono una sparsità nella rappresentazione. La rappresentazione h ha un numero di componenti molto alta ma si induce l’auto-encoder a tante parti di questo vettore a 0.
* **Denoising Auto-encoders**: inducono una robustezza rispetto al rumore o agli input mancanti
* **Contractive Auto-encoders**: vincolano la soluzione chiedendo che la derivata della funzione di costo sia piccola

## 

## Sparse Auto-encoders

Uno Sparse Auto-Encoder è semplicemente un auto-encoder che **non** vincola h più piccolo di x. Aggiunge un termine di normalizzazione alla funzione di loss **vincolando h a essere un vettore sparso** (h deve contenere un numero maggiore di componenti a 0).



### Sparse Auto-encoders come modelli generativi (GM), da un punto di vista probabilistico

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



dove  è la distribuzione a priori del modello rispetto alle variabili latenti h.

|  |
| --- |
| **Domanda**  Quale tra questi modelli corrisponde al modello generativo?    Il modello corretto è il terzo.  La distribuzione congiunta è governata dal seguente processo: si estrae h e sulla base di questo si estrae x.  Si supponga di avere due urne contenenti delle palline con distribuzioni diverse: in un’urna numeri pari e nell’altra urna numeri dispari. Dopodiché si sceglie l’urna da dove prelevare la pallina tirando una moneta. Viene generato h tirando la moneta e, a seconda del valore di h scelgo l’urna e prelevo la pallina.    probabilità del valore della moneta  probabilità di estrarre un certo numero a seconda del valore della moneta (che corrisponde alla scelta dell’urna da cui si preleva il numero). |

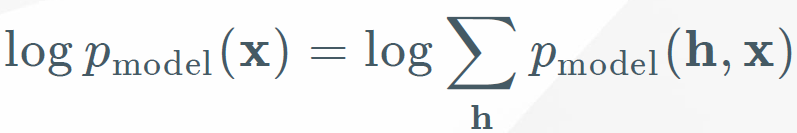
|  |
| --- |
| **IMPORTANTE: Significati di “distribuzione a priori”**  è la distribuzione a priori che il modello ha sulle variabili latenti h. Perché può creare confusione?  La stessa frase può essere utilizzata in un contesto diverso.  Nella **fase di apprendimento**, la probabilità a priori fa riferimento al bias dell’algoritmo di apprendimento rispetto ad un’ipotesi piuttosto che ad un’altra prima di osservare i dati.  Dato l’algoritmo di apprendimento, prima di osservare i dati, quali sono le ipotesi preferibili e quali non?  Ad esempio, un algoritmo che induce alberi di decisione minimizzando l’altezza (come criterio di regolarizzazione) ha come distribuzione a priori quella che favorisce alberi più corti. Poi se i dati implicano che l’albero deve essere molto alto, imparerà anche questo.  Nel caso degli **auto-encoders** di regolarizzazione, non si parla della stessa probabilità a priori come definita precedentemente ma della **preferenza del modello** (dopo averlo appreso) **per la rappresentazione latente h prima di osservare i dati x**.  Nell’esempio della moneta, è la mia intuizione rispetto a quanto alla moneta sia non truccata. |

Stiamo cercando di dare una prospettiva probabilistica alla regolarizzazione degli auto-encoders e questo ci darà un certo insights sul perché dovrebbe funzionare bene o male.

Dietro questa ipotesi:



Se calcoliamo il logaritmo della probabilità di x secondo il modello, si può scrivere indipendentemente dalle ipotesi che facciamo e su come sono collegati ad x.



L’auto-encoder: preso l’input, si costruisce un singolo valore per h e poi si cerca di ricostruire l’input mediante la ricostruzione r.

h è determinato dalla prima parte della rete (encoder, da x a h). La prima parte della rete serve per estrarre l’**h più probabile**, che più probabilmente permette una buona ricostruzione dato x. Dopodiché si usa h per ricostruire x.

Utilizzando la teoria delle probabilità dovrei farlo in modo perfetto usando questa forma guardando e mediando su tutti i possibili h.

Ciò che fa la rete, osservando un solo h, approssima il log della sommatoria usando un solo h scelto in modo sensato per essere quello che mi dà la maggior parte dell’informazione.

Allora, dato h generato, si massimizza:



prob a priori h dipende dal modello

Alcune scelte della funzione di regolarizzazione su h

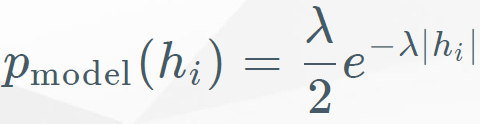


corrispondono esattamente al modello appena introdotto e sono totalmente giustificate dal punto di vista probabilistico.

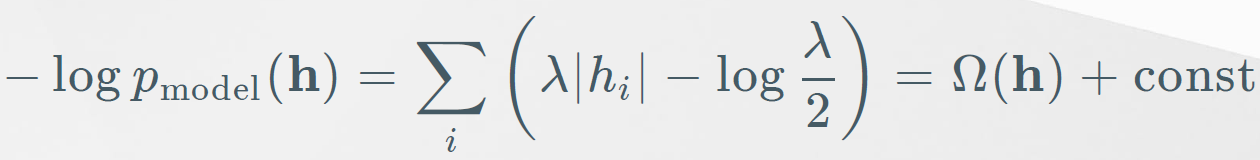
Ad esempio, se si vuole favorire un auto-encoder sparso, la funzione di regolarizzazione che induce la sparsità è la norma 1.

induce sparsità del vettore h.

La funzione di regolarizzazione si ottiene assumendo che la probabilità a priori su h è di tipo Laplace. Se le componenti del vettore h hanno una **distribuzione di Laplace** e sono **indipendenti tra di loro**, vale questa uguaglianza:



Allora



Se si assume che il modello ha una distribuzione di Laplace, si ottiene un modello dove la funzione di regolarizzazione è la norma 1 di h.

Si può affermare che alcuni termini di regolarizzazione corrispondono perfettamente ad **assunzioni a priori di come deve essere fatta h**.

Nel momento in cui il termine di regolarizzazione corrisponde alla norma 1 di h, si assume una probabilità di Laplace come probabilità a priori.

Da questo punto di vista, **la penalità della sparsità (norma 1 di h) non è un termine di regolarizzazione ma come conseguenza di come il modello pensa alle variabili latenti h**.

E’ interessante in quanto ci da ulteriori motivazioni per fare addestramento di un’auto-encoder:

* Si può pensare all’addestramento di un’auto-encoder come un modo per fare apprendimento approssimato di un modello generativo.
* La descrizione h può essere pensata come descrivere le variabili latenti del modello generativo: oggetto che parte da h e permette di generare l’input x. In questo modello, h spiega benissimo come generare x. Capire h permette di capire come generare x. Può avere molti vantaggi.

|  |
| --- |
| **Definizione**  Il modello generativo è un modello che permette di generare dei dati.  **Modello generativo vs Modello discriminativo**  I modelli visti finora sono discriminativi in cui dato un input x, si può decidere di ottenere in output una variabile y che rappresenta l’etichetta.  Il modello generativo modella la probabilità p(x,y) permettendo di fissare y e generare l’input x più probabile per y.  Ciò non è possibile in un modello discriminativo.  Con un modello discriminativo posso etichettare un quadro x come un quadro di Van Gogh.  Con un modello generativo, sia x un quadro di Van Gogh, restituisci l’immagine corrispondente. |

|  |
| --- |
| Chiarimenti  **Cosa sono gli autovalori e gli autovettori?**  **Principal Component Analysis?** |

### Profondità e potenza di rappresentazione

Una rete non profonda con un solo livello hidden [shallow learning] è un approssimatore universale (Teorema di approssimazione universale).

Nelle architetture feed-forward, aumentare la profondità dà un vantaggio rispetto al livello shallow mantiene la capacità di approssimare universalmente qualche funzione ma spesso si hanno vantaggi esponenziali nel **ridurre il costo computazionale nella rappresentazione delle funzioni**. A sua volta implica che si ha anche una **decrescita esponenziale nel numero di esempi che abbiamo bisogno per addestrare il modello**.

Queste due osservazioni sono in contrasto con la storia delle reti neurali: per tanto tempo le reti neurali si conoscevano ma non venivano utilizzate tanto perché si aveva difficoltà ad addestrare le reti profonde. I maggiori problemi erano derivanti dall’esplosione o la perdita del gradiente man mano che si scende lungo l’architettura.

Questi modelli sono tornati di moda quando sono state sviluppate tecniche per evitare questi problemi e quindi addestrare in modo corretto questi modelli.

La stessa cosa vale anche per gli auto-encoders.

Auto-encoders shallow (con un solo livello nascosto) possono fare tutto ciò che vogliamo ma dobbiamo avere abbastanza neuroni in quel livello. Ma gli Auto-encoders con un’architettura profonda ci offrono gli stessi vantaggi (in grassetto). In effetti auto-encoders profondi danno delle migliori performance in termini di **compressione del dato iniziale**: si ottengono h più piccoli e la ricostruzione è molto elevata.

Gli auto-encoders di tipo shallow (con un solo livello hidden) sono ancora molto usati in quanto fanno ancora parte di addestramento di alcune architetture deep. Una delle difficoltà delle architetture deep è riuscire a non perdere informazioni sul gradiente man mano che si scende lungo l’architettura. Una delle tecniche è l’uso di tanti auto-encoder shallow per pre-addestrare i livelli della rete. Quindi la rete è costruita a blocchi di auto-encoders shallow. Una volta costruiti i livelli, poi si addestra la rete in modo supervisionato a partire da quei pesi (pre-training).

## Auto-encoder dal punto di vista stocastico

Una strategia generale per progettare le unità di output e la funzione di perdita di una rete feedforward consiste nel definire una distribuzione di output p(y|x) e minimizzare .

Nel caso degli auto-encoders, x è anche il target oltre ad essere l’input, ma si possono usare le stesse idee.

Potremo vedere il decoder come un oggetto che modella la distribuzione condizionale



Assunzione:

Si assume che la rete è di tipo shallow (un solo livello hidden e fully-connected) con unità lineari. Si assume che x è valore reale e le unità di output sono lineari, si vuole caratterizzare la probabilità usando una distribuzione Gaussiana multivariata.



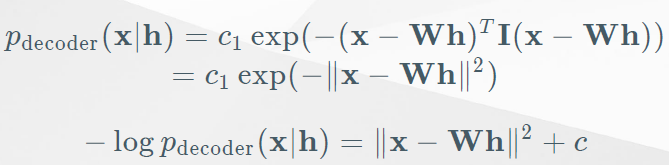
La probabilità che x sia tanto diverso da x^ (calcolato mediante Wh) decresce esponenzialmente man mano che x si allontani da x^.

x^: predizione della rete

Sotto queste ipotesi:



è una loss quadratica sull’output della rete → si può addestrare la rete con la discesa del gradiente utilizzando la loss quadratica. Si tratta di una rete che modella il decoder utilizzando una multivariata.



La probabilità è modellata usando una normale multivariata con densità



dove c1 è il termine di normalizzazione, serve per far sì che l’integrale su tutti i possibili valori di x è uguale a 1.

La matrice di covarianza è indicata con la matrice di identità I.

Se si addestra il modello con la discesa del gradiente, usando unità di output lineare e loss quadratica, si può interpretare un modello che modella mediante una distribuzione di probabilità Gaussiana multivariata.

Ricapitolando:

Nel momento in cui si addestra un modello in modo stocastico, una delle strategie è la **minimizzazione della log-likelihood** , ovvero la minimizzazione di



Quindi si può addestrare la rete con la discesa del gradiente usando la funzione di loss .

Siamo interessati all’insieme dei parametri W che rendono minima tutta la espressione.

c è una costante, non dipende dai parametri W, quindi non ci interessa.

Nel momento in cui si fa ottimizzazione, bisognerebbe massimizzare la p(x|h) ma è difficile da calcolare. Quindi si minimizza la -log… perché è più semplice da calcolare.

Se x è distribuito secondo una distribuzione di Bernoulli, le unità di output sono sigmoide.

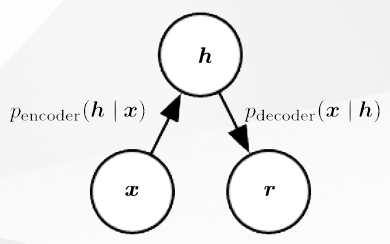
I valori x discreti corrispondono a una distribuzione softmax.

In generale, se si pensa agli auto-encoders utilizzando questo framework stocastico, si può determinare qual è il modo migliore per modellare le unità di output.

Quindi tutti gli auto-encoder possono essere visti come un modo di catturare un modello di variabili latenti

 probabilità congiunta di un modello sulle variabili x e h.

Modello pensato dal punto di vista stocastico:



x è usato per generare h

h è usato per generare r

La probabilità secondo il modello di h dato x è modellato mediante l’Encoder



La probabilità secondo il modello di x dato h è modellato mediante il Decoder



Deve esistere una probabilità congiunta p(x,h) del modello che sia compatibile con queste due probabilità condizionali in quanto le probabilità condizionali potrebbero non corrispondere ad alcuna distribuzione congiunta. Può succedere ma si può mostrare che se si addestra l’encoder e il decoder con un approccio di Denoising, questo tipo di addestramento tende a rendere compatibili asintoticamente queste due distribuzioni.

Quando usiamo il Denoising Auto-encoder, si convergono le due distribuzioni di probabilità per renderle compatibili con un’unica distribuzione congiunta → il modello pensato in modo stocastico è un modello sensato.

## Denoising Auto-Encoders (DAE)

I Denoising Auto-Encoders (DAE) fanno parte dei Regularized Auto-Encoders.

Adottano una strategia differente per apprendere qualcosa di utile.

Anziché aggiungere una penalità alla funzione di costo (loss function), i DAE cambiano l’errore di ricostruzione della funzione costo. In particolare, si minimizza:



dove  è una copia di x che è stata corrotta da qualche forma di rumore.

Nella fase di apprendimento, l’esempio x viene corrotto con del rumore, lo si passa in input all’auto-encoder.

Nel momento in cui bisogna verificare se l’auto-encoder apprende bene, non si confronta l’input dell’auto-encoder con il suo output perché l’input è stato corrotto. Ma si confronta l’esempio x originale (senza rumore) con l’output dell’auto-encoder.

In questo modo l’auto-encoder è forzato a rimuovere il rumore dall’input x, anziché semplicemente copiare l’input nell’output.

Questo tipo di apprendimento forza f e g ad imparare la distribuzione degli input x.

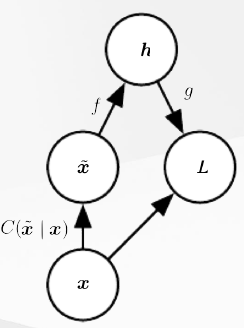
### DAE dal punto di vista stocastico

Un modo per capire i Denoising Auto-Encoders (DAE) è introdurre un processo di corruzione  che produce rumore nell’input.

Mediante la funzione f, si genera la rappresentazione h (Encoder).

La funzione g è usata per costruire la rappresentazione r (Decoder).

L’esempio x originale è confrontato con r da una loss function utile per fare propagazione all’indietro dell’errore e addestrare il modello.



L’auto-encoder impara una **distribuzione di ricostruzione**



con



Si può semplicemente applicare la discesa del gradiente (discesa del gradiente mini batch) per minimizzare il negative log likelihood



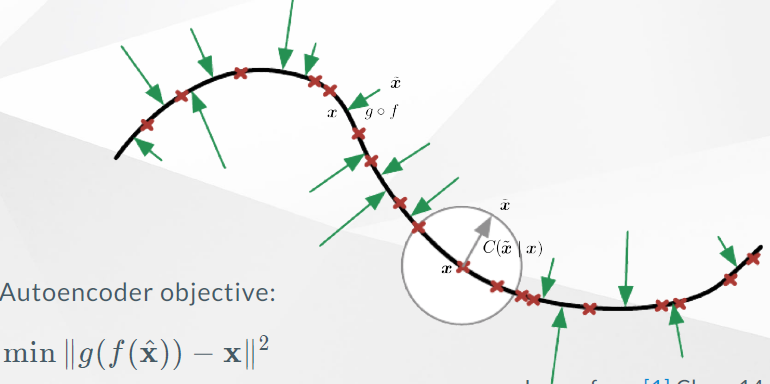
Se la parte di encoding è deterministica, tutto l’auto-encoder si può addestrare end-to-end usando la discesa del gradiente stocastico e possiamo quindi vedere l’auto-encoder DAE come una discesa stocastica del gradiente su:



ovvero si cerca di minimizzare la log likelihood sulla distribuzione dei dati secondo la distribuzione che introduce il rumore della probabilità di decodificare correttamente x dato l’esempio corrotto.

### DAE come approssimazione di un campo di vettori

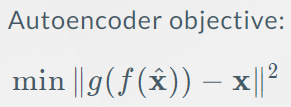
Si può pensare di vedere i DAE come oggetti che approssimano un campo di vettori.



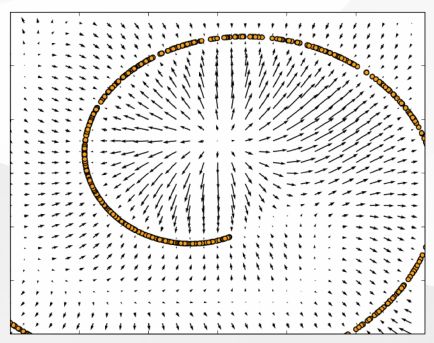
(Monodimensionale)

Idea: in tutto lo spazio, la funzione restituisce un vettore che indica una direzione.

Il fatto di minimizzare:



vuol dire che g(f(x^)) deve essere in grado di prendere l’input corrotto e riportarlo sul manifold dove giaceva inizialmente.

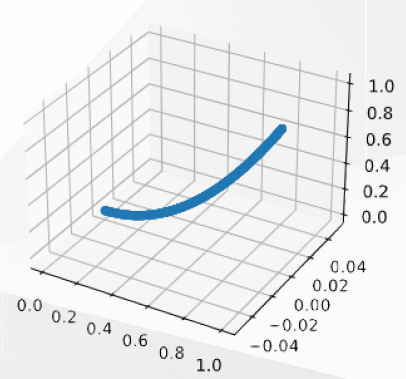


(Bidimensionale)

### Apprendimento Manifolds

Gli auto-encoders sono usati per apprendere la struttura del manifold dove vivono i dati.

Un manifold è uno spazio di dimensioni ridotte su cui vivono i dati che sono immersi in uno spazio più ampio.



L’idea dei manifold è usata in molti algoritmi di apprendimento automatico che fanno questa assunzione per semplificare l’apprendimento.

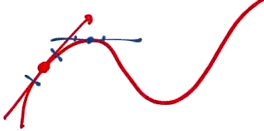
In figura, il manifold è monodimensionale ed immerso in uno spazio tridimensionale.

Se si fa apprendimento tenendo in considerazione tutte le possibili direzioni nello spazio, ho bisogno di molta più complessità e quindi di molte più informazioni rispetto a fare apprendimento sull’oggetto (curva) che può essere mappato su una retta.

Gli Auto-Encoders apprendono tutta la struttura del manifold in modo tale che l’apprendimento venga fatto sul manifold e non più sullo spazio originale e quindi apprendere più facilmente.

Un modo di caratterizzare un manifold è attraverso l’**insieme dei suoi piani tangenti**.

In un punto x su un manifold d-dimensionale, il piano tangente è dato da d vettori di base che coprono le direzioni locali di variazione consentite sul manifold.



Si pensi ad un manifold come alla curva (in figura) immersa in uno spazio tridimensionale.

Per ogni punto della curva, si ha un piano tangente al punto.

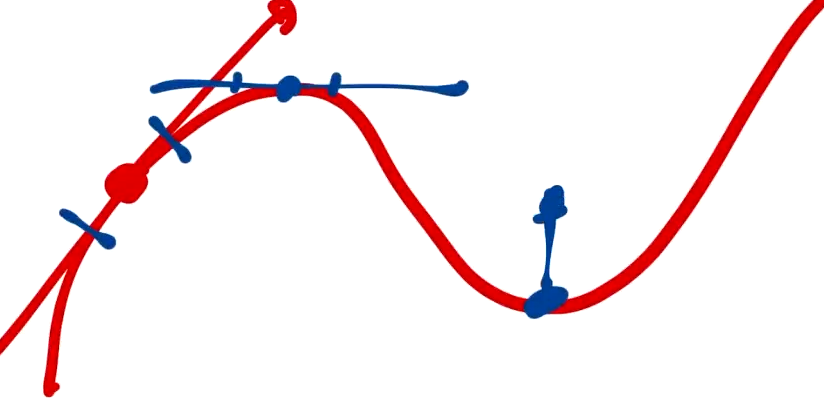
Nell’intorno del punto considerato, il manifold si comporta come se fosse un piano.

Quindi si può approssimare la curva usando questi piani.

Si può assumere che tutti i punti su questo piano nell’intorno del punto abbiano la stessa etichetta.

I piani tangenti indicano come cambiare x in modo infinitesimo rimanendo sul manifold.

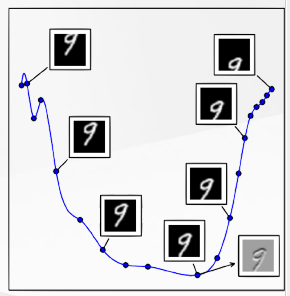
Se prendo un qualsiasi punto della curva e mi sposto in una direzione a caso (come in figura), mi sposto nel manifold, non c'entra più niente con i dati...



Se invece mi sposto lungo il piano tangente, rimango sul manifold.



Si consideri la seguente figura:



Data un’immagine del numero 9, viene traslata verso il basso.

La curva blu è la caratterizzazione del manifold dove vivono i dati: si tratta di un manifold molto più ampio di 2D proiettato in 2D.

Lungo questo manifold, tutti i punti sono oggetti che potrebbero essere dentro i dati.

Se si prende un punto al di fuori del manifold, avremo un esempio che non c'entra assolutamente niente con i nostri dati.

Mentre lungo tutti i punti del manifold abbiamo un oggetto che è dentro ai nostri dati.

Nel momento in cui ci si sposta lungo una delle tangenti del manifold, si ottiene un'immagine che ancora fa parte del manifold.

In colori più chiari sono mostrati i punti della nuova immagine e in più scuro punti della vecchia immagine.

L’immagine del 9 grigia è quella generata a seguito dello spostamento sulla tangente del manifold.

### Auto-Encoders e Manifolds

Tutte le procedure di apprendimento degli auto-encoders coinvolgono un compromesso tra le due forze:

1. Apprendere una rappresentazione h tale per cui a partire da essa si può recuperare in modo approssimativo il valore di input
2. Soddisfare qualche vincolo di regolarizzazione

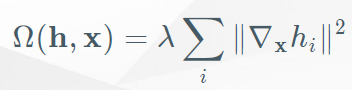
Queste due forze insieme sono utili in quanto l’auto-encoder non si può permettere di rappresentare tutte le possibili variazioni che potrebbero capitare nei dati, ma è forzato a ricostruire solo le possibili variazioni che hanno a che fare con i dati visti in input. Quindi deve tralasciare le variazioni dei dati che non hanno a che fare con i dati.

Se la distribuzione che genera i dati si concentra su un manifold a basse dimensioni, l’auto-encoder tende a apprendere il manifold in cui vivono i dati. In altre parole, apprende una rappresentazione che proietta i dati sul manifold, perdendo tutte le direzioni che porterebbero i punti lontani dal manifold.

## Contractive Auto-Encoders (CAE)

Auto-encoder che fanno contrazione.

Questo tipo di auto-encoder aggiunge un termine di regolarizzazione della forma:



L’auto-encoder impara un vettore h, ciò significa che impara una serie di neuroni in cui ciascun neurone può essere visto come una funzione h\_i dell’input.



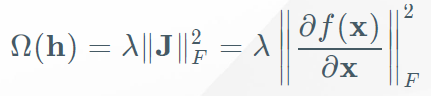
Su ognuno di questi si può calcolare il gradiente secondo il suo input x.

Il termine di regolarizzazione calcola esattamente la lunghezza del gradiente, somma tutte queste lunghezze.

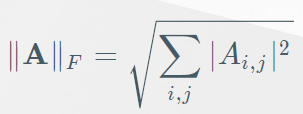
Il termine di regolarizzazione rende tutte le derivate prime delle funzioni h\_i piccole, ovvero forza il modello ad imparare una funzione che non cambia molto quando cambia x: per piccole variazioni di x, non sono permesse grosse variazioni dell’output della funzione.

|  |
| --- |
| **Domanda: Quali sono le implicazioni di un valore molto alto della costante ?**  L’auto-encoder apprende una funzione costante in quanto forzando la derivata prima ad essere molto piccola, in particolare ponendo la costante molto alta e quindi si forza la derivata prima ad essere pari a 0, allora la funzione è costante.  Inoltre, con molto alto,    viene ottimizzato solo il termine di regolarizzazione e viene ignorato il termine di ricostruzione, quindi non si riesce più a ricostruire x. |

Lo stesso tipo di termine di regolarizzazione può essere scritto in questo modo:



 è la **matrice Jacobiana** e



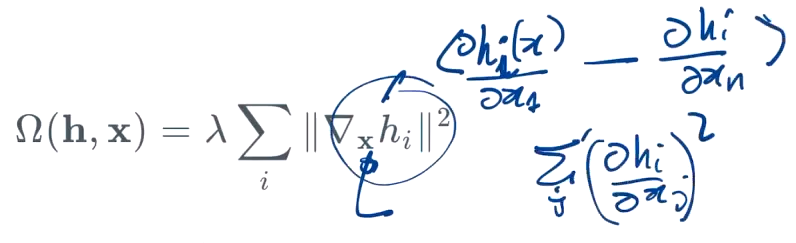
|  |
| --- |
| La **norma di Frobenius** è legata alle matrici (non ai vettori), corrisponde alla norma euclidea ma relativa alle matrici.  Data una matrice  A = [[a b]  [c d]] |

J è detta matrice Jacobiana

|  |
| --- |
| Si parla di una funzione multivariata.  Esistono molti output e ognuno di questi può essere pensato come una funzione f(x) dove x è un vettore.  matrice delle funzioni  Essendo funzioni multivariate, per ognuna di queste si può calcolare il gradiente rispetto alle variabili x e ottenere quindi la seguente matrice Jacobiana. |

Calcolare la norma Frobenius della matrice Jacobiana J significa prendere ciascuna componente della matrice, elevare al quadrato e fare la somma.

Quindi



questa è la stessa espressione che si ottiene calcolando la norma di Frobenius della matrice Jacobiana J.

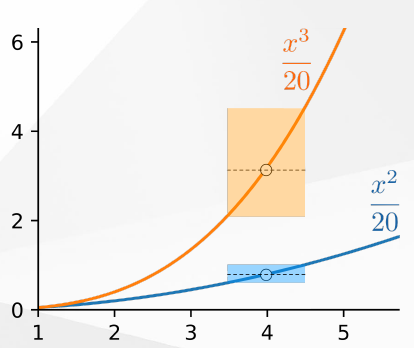
La matrice Jacobiana J indica quali sono le direzioni dove varia di più la funzione multivariata.

Moltiplicando J per il vettore x:

se si ottiene un vettore lungo → lungo quella direzione la funzione varia molto

se si ottiene un vettore corto → lungo quella direzione la funzione varia poco.

**Perché il nome “Contractive”?**



Come effetto laterale della penalità appena vista, l’auto-encoder impara a mappare un intorno del punto x in un punto in uno spazio ridotto rispetto a quello di partenza.

In figura sono raffigurate due funzioni.

Le derivate delle due funzioni sono diverse:

quindi la derivata della funzione y1 cresce più velocemente rispetto alla derivata della funzione y2.

Preso un punto sulla curva y2, si considera un suo intorno, osservare cosa viene mappato sulle y questo intorno x del punto scelto, questi punti di arrivo sono molto più piccoli rispetto a ciò che succede dove la derivata è maggiore.

L’intorno relativo alla curva y2 è mappato su uno spazio minore.

L’intorno relativo alla curva y1 è mappato su uno spazio maggiore, in quanto la derivata della funzione cresce più velocemente.

Si chiede che la derivata prima sia piccola in modo tale che non apprenderemo una funzione dove l’intorno è mappato su uno spazio minore.

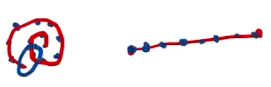
Caso estremo: cosa succede se abbiamo un lambda molto alto oppure non avessimo il termine di regolarizzazione? (vedere la domanda)

* si apprende una funzione costante che mappa l’intero spazio di input in un unico punto che è la costante lambda

In tutti gli altri casi si forza il mapping a ridurre le dimensioni nello spazio.

La proprietà contrattiva degli Auto-Encoder è vera solo localmente: se prendo un intorno di x, in genere si riduce.

Se prendo due punti x1 e x2 che sono distanti tra di loro, potrebbero essere mappati in due punti distanti.



manifold mapping

originale

### CAE e DAE a confronto

Si suppone un input con rumore gaussiano piccolo.

Con gli auto-encoder DAE, l’errore di ricostruzione basato sulla rimozione del rumore è equivalente a una penalità di tipo costruttivo sulla funzione di ricostruzione che mappa x su r=g(f(x)).

I DAE fanno sì che la funzione di ricostruzione g sia in grado di sopportare piccole variazioni all’input.

Mentre i CAE fanno la stessa cosa ma sulla funzione f → f(x) è la funzione che mappa in h. Quindi f(x) è robusto a piccole variazioni del suo input.

### Apprendimento Manifolds

…………………….