# 1. SOM – Self Organazing maps

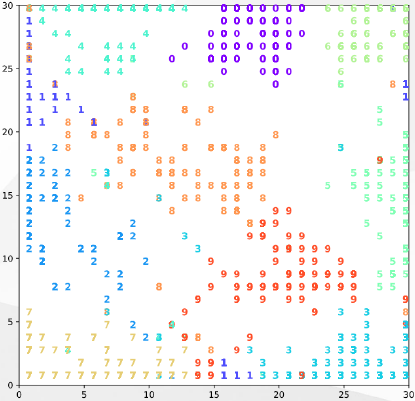
Tecnologia di NN di fine anni ’60 🡪 si ispira a come viene usato il cervello umano

Cervello umano = insieme di cortecce all’interno delle quali gli stimoli sono organizzati in maniera **topologica**: i neuroni vicini si eccitano/attivano in corrispondenza di stimoli vicini

Es. 2 img di gatti simili = attivazione cortecce vicine tra di loro

**Kohonen maps** 🡪 strumento computazionale (si discosta leggermente dalla rappresentazione umano) auto-organizzato

**Obiettivi e caratteristiche SOM:**

* **Modelli non supervisionati** (no classificazione, regressione) 🡪 descrizione dei dati
* **Mono o bi-dimensionali**
* **Una volta addestrata** 🡪 stimoli che arrivano in ingresso alla mappa organizzati **topologicamente** = stimoli simili mappati in neuroni vicini nella mappa 

**Principi dell’algoritmo delle SOM:**

1. Competizione tra neuroni: ad ogni nuovo stimolo tutti i neuroni competono per diventare la **BMU** (= Best Matching Unit) 🡪 l’unità che meglio rappresenta lo stimolo ricevuto
2. Cooperazione tra neuroni: il neurone vincente (ovvero la BMU) 🡪 **attiva una sorta di intorno**
3. Adattamento sinaptico: ad ogni iterazione, il neurone vincente fa si che i neuroni a lui vicino diventino più simili al neurone stesso

**Processo competitivo**

**x = vettore**

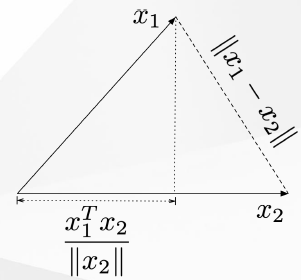
**colonna di input 🡪** [x1, x2 …, xm]

**SOM con *l* valori 🡪** ogni “cella” ha un neurone con un valore w

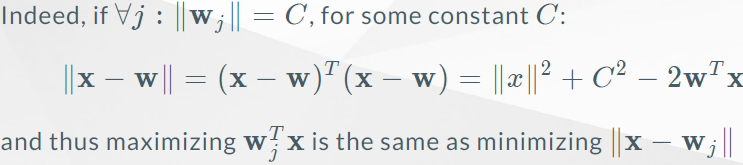
**Ogni unità *j* nella SOM** (= cella della mappa) possiede un **vettore di pesi wj con dimensione uguale a x**

**Neuroni disposti nella mappa secondo una certa topologia** (es. lineare [monodimensionale], a griglia [bidimensionale], ecc.)

**Topologia** = branca matematica che studia relazione di vicinanza degli oggetti 🡪 la cosa importante non è la forma, ma i rapporti di vicinanza

**Relazione tra prodotto scalare e norma euclidea** 🡪se i vettori sono normalizzati 🡪 forniscono la stessa informazione

Se si assume che i vettori *wj* hanno tutti stessa lunghezza 🡪 se si seleziona il vettore con il prodotto scalare massimo 🡪 si seleziona il w (tramite un indice) che minimizza la distanza euclidea

Quindi per trovare il minimo della distanza euclidea si deve fare:

**wTw = || w ||2**

**Cercando BMU ci sono due procedure diverse:**

1. Utilizzo della norma 🡪 minimizzare la differenza tra x e wj
2. Utilizzo prodotto scalare 🡪 massimizzare wTx

**Processo di cooperazione**

**Obiettivo =** far collaborare i neuroni una volta trovata la BMU che sono vicini ad essa

🡪 funzione della BMU identificata da i(x) e dai neuroni j ad esso vicino

**Tale funzione deve:**

* Avere il suo massimo nella BMU
* decresce (rapidamente) allontanandosi dalla BMU

**Distanza laterale** = di,j 🡪 funzione delle coordinate dei neuroni nella mappa

h spesso definita come segue (sigma indica quanto velocemente la funzione deve decrescere)

**Neighborhood function = h così 🡪 funzione a campana** 🡪 minore è sigma, più i vicini saranno meno considerati mentre ci si allontana dalla BMU

**Si deve diminuire sigma durante l’apprendimento** 🡪 perché all’inizio i pesi sono scelti a caso e quindi le relazioni topologiche saranno casuali, quando si è avanti nell’apprendimento va cambiato sigma per far sì che le nuove BMU identificate si creino dei vicini simili ad essa, altrimenti cambiano sempre gli stessi pesi andando ad interferire col lavoro degli altri neuroni

**Riduzione tramite la seguente formula 🡪** n e T sono parametri dell’utente 🡪 questa nuova sigma finisce al posto di quella nella formula di h di prima

**Processo adattivo**

**Regola di update** 🡪

🡪 learning rate al tempo n

Ovvio che se e = 1 🡪 si ottiene che

🡪 = learning rate iniziale T2 (teta) = indica con che velocità viene decrementato 🡪 diventerà a mano a mano vicino a zero. Entrambi i parametri sono settati dall’utente

2 fasi nel processo:

1. **fase di auto- organizzazione/ordinamento** 🡪 la struttura topologica viene inizializzata (che per forze di cose sarà sbagliata. Rapidamente, la struttura topologica verrà ripristinato mettendo vicini i neuroni con pesi vicini
2. **convergenza** 🡪 struttura viene raffinata andato a ridurre i difetti. Solitamente si sciolgono i “nodi” che si creano all’inizio

**Fase 1**: **auto organizzazione** 🡪 richiede almeno 1000 iterazioni (ma anche di più). Learning rate e sigma0 vanno scelti con molta cura (apprendimento sensibile a questi parametri).

Solitamente e

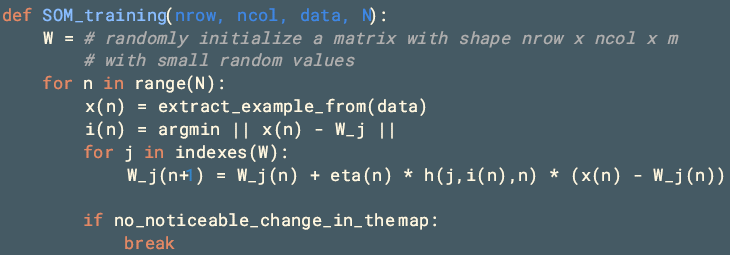
**Obiettivo**: la funzione di vicinanza (**neighborhood function**) deve raggiungere circa tutti i neuroni 🡪 questo perché all’inizio i neuroni sono settati a caso e avere una funzione adeguata permette di creare una corretta struttura topologica

**Suggerimento**:, inoltre settare al “**raggio**” **del lattice** = numero di neuroni al centro (considerando griglia bidimensionale)

**Fase 2: convergenza** 🡪 migliora di molto la quantizzazione dello spazio di input: si usano le SOM per far si che lo spazio di input, appunto, sia quantificato 🡪 ogni neurone rappresenta lo spazio di sua competenza

Per far sì che la convergenza avvenga correttamente 🡪 non deve scendere troppo, altrimenti SOM si blocca e rimane in uno stato metastabile (rimane stabile, ma non è nella sua posizione ottimale e basterebbe spostarla di poco per migliorare le performance) e presenta dei difetti topologici (= si hanno oggetti simili tra loro ma molto distanti tra loro nella mappa)

In questa fase 🡪 funzione di neighborhood non dovrebbe più abbracciare tanti pesi 🡪 si assume che la struttura topologica sia abbastanza buona 🡪 si vuole raffinare la descrizione dei neuroni

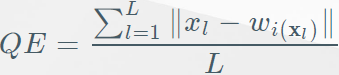
**Algoritmo di training della SOM**

**2 aspetti per misurare la qualità della SOM:**

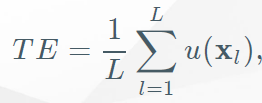
* qualità del learning (o della quantizzazione dei vettori) 🡪 quanto i neuroni nella SOM riescono ad approssimare bene i dati in input
* qualità della proiezione 🡪 qualità dell’ordinamento topologico trovato dalla SOM: alta quando oggetti simili sono in posizioni vicine e oggetti diversi sono in posizioni lontane

**3 misure per calcolare la qualità:**

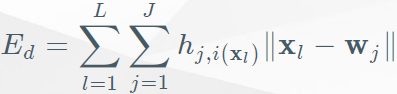
1. errore di quantizzazione
2. errore di topografia
3. misura di distorsione

**Errore di quantizzazione 🡪** misura quanto bene i dati vengono mappati sulla SOM. Computa la distanza media tra i vettori di esempio e le loro BMU

Misura solo la quantità della quantizzazione dello spazio di input 🡪 NO su spazio topologico

**Errore topografico** 🡪 topologia ben preservata se punti vicini in input (es per la loro distanza euclidea) sono vicini anche nello spazio di output 🡪 ovvero i(x1) vicino a i(x2) dove i sono gli indici assegnati nella SOM.  
Misura la percentuale dei dati in cui la prima e la seconda BMU **non** sono adiacenti nello spazio di output

La formula è la media aritmetica di , dove vale 1 se la prima e la seconda BMU sono vicine nello spazio di output, 0 altrimenti

**Misura di distorsione** 🡪 cerca di catturare entrambi gli errori prima citati. Se si assume di lavorare su un set di training finito (generalmente vero) e si assume che la funzione h (neighborhood) ha un’ampiezza fissa (non diminuisce come abbiamo detto) 🡪 allora la quantità usata nella regola di update è un’approssimazione efficiente del gradiente della misura di distorsione scritta sotto

Questa misura cattura in parte l’errore di quantizzazione e in parte quello di topologia

Questa misura di decompone in **3 parti:**

* qualità della quantizzazione dello spazio di input
* qualità di costruzione della topologia fatta dalla SOM
* una parte che combina/mette insieme/connette le prime due parti

**Applicazione della SOM** 🡪 in tanti campi diversi, per una vasta varietà di problemi. Tra i principali:

* analisi processi
* geoscienza
* design di sistemi esperti
* design di sistemi intelligenti
* data mining
* …

Tante applicazioni per vari **vantaggi nell’uso delle SOM**:

* clustering dei dati
* visualizzazione dei dati
* discretizzare il dataset in un numero piccolo di vettori (es. per la compressione dei dati)
* semplificazione per i learning task 🡪 permettono di fare una sorta di pre-processing

# 2. autoencoders – ae

**AE** 🡪 strumenti **non supervisionati** con vari utilizzi per risolvere diversi task (colorazione di immagini, aumento risoluzione img, machine translation, ecc.)

**Utilizzato principalmente per migliorare le performance delle reti supervisionate**

**AE 🡪** NN addestrata che cerca di copiare l’input sul proprio output

**2 parti:**

* Encoder: codifica l’input 🡪
* Decoder: prende la rappresentazione h ricostruisce l’input r 🡪

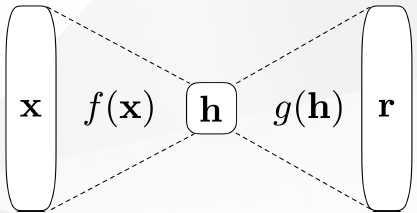
AE funziona bene se r è molto simile ad h

**Non** si vuole una copia esatta dell’input, ma si vuole dare priorità agli aspetti importanti e da preservare dell’input 🡪 rappresenta una sorta di compressione dell’input, in cui vengono eliminati gli aspetti irrilevanti

**principali tipi di AE:**

1. **Undercomplete AE** 🡪 la rappresentazione h che si apprende ha una dimensione più piccola rispetto a quella data in input.

AE forzato a comprimere l’input: tralascia alcuni aspetti dell’input (non considera alcune dimensioni nello spazio)

**Apprendimento (sfrutta la backpropagation)**: minimizzazione di una funzione di perdita 🡪 penalizza la differenza tra input x e la ricostruzione fatta da g dell’encoding fatto da f.

**AE e PCA**

Quando il decoder è lineare (non ci sono funzioni di attivazioni non lineare) e L è il mean squared error 🡪 AE sta risolvendo lo stesso probelma che cerca di risolvere la PCA

Effetto del processo di apprendimento 🡪 AE impara/prende in considerazione il sottospazio dei dati di apprendimento

**PCA** 🡪 può essere derivata come un algoritmo che proietta un esempio x in vettore c *con una dimensione minore* formalizzando: 🡪 dove **g rispetta alcuni vincoli:**

* È lineare 🡪
* Colonne di D sono ortogonali l’una con le altre (per semplificare il problema)
* Colonne di D devono avere una norma unitaria (per soluzione unica)

Dati questi vincoli 🡪 🡪 la matrice D è data dagli autovettori che corrispondono ai più grandi autovalori della matrice

**PCA 🡪** considerando una matrice di dati X a cui ciascun valore è stato sottratto il valore della media della colonna a cui appartiene 🡪 la covarianza dei vettori in X si può calcolare con 🡪 la covarianza dice quanto ognuno delle coordinate in X varia quando confrontate con altre coordinate in x.

**Quindi PCA 🡪** trova gli autovalori e autovettori di e trova gli autovettori della matrice a seconda del loro autovalore 🡪 prima quelli con autovalore più grande 🡪 questi autovettori vanno in D che è la matrice che serve nella PCA per fare la trasformazione = ottenere la rappresentazione in sole 3 dimensioni

**In parole povere:**

* AE minimizza la funzione di loss
* PCA 🡪 trova il mapping , imponendo g come modello lineare

Avendo due approcci simili, ma differenti (AE vs PCA) 🡪 ci sono due approcci per risolvere lo stesso problema

AE su manifold lineare 🡪 qualità risultato uguale a PCA

Su manifold non lineare 🡪 PCA non lavora bene

**Performance AE vs PCA**: dipende dal problema. PCA fornisce una soluzione chiusa (con problema piccolo 🡪 soluzione trovata rapidamente). Con problema grosso AE va meglio perché non devono costruire grosse matrice

1. **Regularized AE** 🡪 se la dimensione di **h è troppo grande** 🡪 AE imparerebbe solo la funzione identità

Quindi, quando AE ha troppa capacità c’è il rischio che non impari nulla di utile

**Regularided AE** usa una funzione di loss che incoraggia il modello ad avere qualche proprietà che ci interessa (oltre all’abilità di copiare l’input nell’output)

**Si regolarizza la funzione di loss**

3 varietà:

* Inducono una sparsità della rappresentazione 🡪 **sparse autoencoders**
* Indurre robustezza del rumore o input mancanti 🡪 **denoising AE**
* Derivata della funzione di costo piccola 🡪 **Contractive AE**

**Sarse AE**

Aggiunge una funzione di penalizzazione 🡪 vincola h ad essere un vettore sparso (= deve contenere tanti zeri) 🡪 

Sparse AE come **Generative Model (GM)** che graficamente si rappresenta come

No, va beh, sta parte la balzo... fanculo! Non ha senso di esistere

Da slides 22 a 29, ma mi rifiuto di capire

Una rete non profonda con un solo livello hidden è un approssimatore universale.

1. Sdfs

**Profondità AE**

AE profondi 🡪 migliori performance di compressione del dato iniziale 🡪 calcolano h di dimensioni molto più piccole, ma che permettono una ricostruzione adeguata dei dati.

**Visione stocastica degli AE**

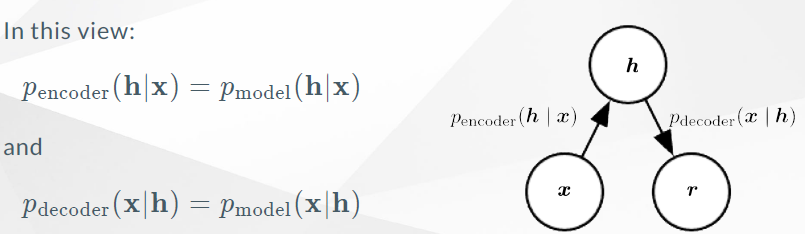
Spesso serve vedere l’output di una NN da un punto di vista stocastico 🡪 cercando di minimizzare la funzione

Per AE non c’è la y 🡪 si sta ricostruendo x 🡪 si immagina decoder come un oggetto che modella x dato h 🡪

**Partendo da questa concezione del decoder** 🡪 si può pensare di usare un output lineare caratterizzato attraverso una distribuzione gaussiana 🡪 si vuole che la probabilità che la x sia tanto diverso da Wh decresca esponenzialmente mentre ci si allontana da Wh

🡪 si tende a minimizzare questa funzione, 🡪 addestrare la rete con discesa del gradiente usando come loss proprio questa funzione

Tutto questo per dire che se addestro con la discesa del gradiente il modello usando delle unità di output lineari e una loss quadratica 🡪 si può interpretare il modello che modella quel tipo di distribuzione di probabilità (??) MAH

Con questa visione stocastica 🡪 encoder visti come modo di calcolare la congiunta di un modello sulle variabili sia x che h (BOH)

**Vista stocastica**

Affinchè le probabilità delle img sopra abbiano un senso 🡪 deve esistere una **probabilità congiunta** che sia compatibile con le due probabilità condizionali di encoder e decoder

Se si addestrano decoder e encoder come un denoising autoencoder 🡪 **rende compatibili asintoticamente le due distribuzioni nella img** 🡪 modello diventa sensato (BOH)

**Denoising AE – DAE**

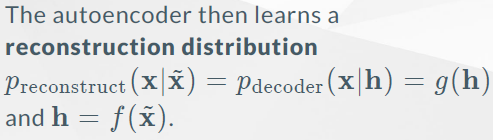
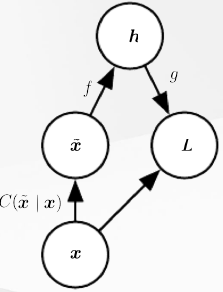
Altro modo per vincolare AE per apprendere qualcosa di interessante sugli input

**Si conta di minimizzare**  **🡪** rappresenta una copia di x corrotta da qualche forma di rumore

**Durante l’apprendimento** 🡪 si corrompe l’input x 🡪 lo si passa all’AE 🡪 per verificare se AE sta lavorando bene 🡪 **si confronta l’x senza rumore con l’output dell’AE 🡪** in questo modo l’AE è forzato ad apprendere come ridurre il rumore di x

**Buone performance = rimozione rumore dall’input**

Questo tipo di apprendimento forza le funzioni f e g ad imparare la struttura dei dati

**Vista stocastica DAE**

**C 🡪** processo che genera il rumore e restituisce

**f usata per costruire h**

**g usata per costruire r**

**r confrontata con x** insieme ad una loss function L che serve per fare retro-propagazione dell’errore 🡪 estrarre poi il modello

Si fa nuovamente la discesa del gradiente su

Se encoder deterministico 🡪 AE si può addestrare end-to-end (con discesa del gradiente e propagazione dell’errore) 🡪 cerca di minimizzare 

**AE e Manifold**

Gli AE apprendono tutta la struttura del manifold per far sì che sia più semplice fare apprendimento successivamente

**Manifold** 🡪 caratterizzato dai piani tangenti che dicono come si può cambiare un punto x infinitamente rimanendo comunque nel manifold

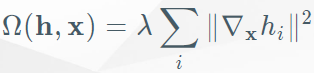
Procedure di **apprendimento** di **AE** coinvolgono un **compromesso tra 2 forze**:

1. Apprendere una rappresentazione h 🡪 a partire da essa si può recuperare l’input
2. Soddisfacimento di alcuni vincoli

Queste 2 forze insieme 🡪 AE forzato a ricostruire solo le possibili variazioni che hanno a che fare con i dati in input

SE la distribuzione che genera dati si concentra su un manifold con poche dimensioni 🡪 AE tende ad apprendere il manifold in cui vivono i dati 🡪 apprende una rappresentazione che cattura implicitamente il sistema di coordinate del manifold (proietta i dati su di esso) 🡪 con la proiezione, si perdono tutte le direzioni che porterebbero i punti lontano dal manifold stesso

**Contractive AE - CAE**

Idea 🡪 AE che aggiunge un termine di regolarizzazione alla loss function L con la seguente forma: 

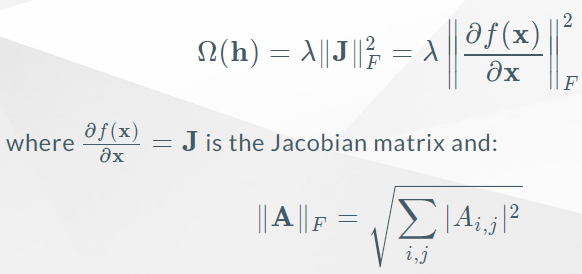
🡪 settato dall’utente

Delta al contario 🡪 gradiente su hi (? … forse)

hi 🡪 rappresenta le funzioni di ciascun neruone in h in funzione dell’input

obiettivo 🡪 rendere le derivate prime di hi molto piccole

**Per piccole variazioni di x 🡪 non si vogliono grandi variazioni dell’output**

**Altro modo per scrivere omega:**

**Norma di Frobenius =** norma applicata ad una matrice 🡪 prende tutti gli elementi della matrice al quadrato, li somma e li mette sotto la radice

**Va beh jacobiano sti cazzi** 🡪 è una roba strana, ma sticazzi

Perché **contractive**?🡪 effetto laterale di AE è quello che impara a mappare un intorno di un punto x in uno spazio ridotto rispetto a quello di partenza

**CAE vs DAE**

La ricostruzione con rimozione del rumore (denoising) è equivalente ad una penalità di tipo contrattivo (non su f) ma sulla funzione di ricostruzione mappando x a r = g(f(x)).

**DAE** fanno sì che la funzione di ricostruzione (g) sono in grado di sopportare piccole variazioni al suo input.

Mentre **CAE** fanno la stessa cosa ma sulla funzione f, fanno si che tutto il processo sia robusto perché f(x) è robusta a piccole variazione del suo input.

Nel caso di **CAE** le due forze sono reconstruction error (output = input) e la penalità contrattiva, il compromesso produce una soluzione con delle derivate più piccole.

# 3. representation learning

Capacità del modello di imparare una nuova rappresentazione dei dati rispetto a quella di input, facilitando l’uso successivo di tali dati

**Condivisione delle proprietà statistiche dei dati attraverso vari task**

Le **rappresentazioni condivise** sono utili per gestire domini e risolvere problemi differenti

Permettono di trasferire la conoscenza per risolvere problemi con pochi o nessun esempio

**Importanza di apprendere una rappresentazione** 🡪 passare da un problema difficile ad uno più facile

**Buona rappresentazione = semplificazione dei task successivi**

Apprendendo una buona rappresentazione 🡪 non sempre porta a buoni risultati, a meno che non si impongano dei vincoli sulla rappresentazione

**2 obiettivi principali, spesso in contrapposizione l’uno con l’altro**

* Preservare la maggior parta delle info dell’input
* Forzare delle proprietà sulle rappresentazioni

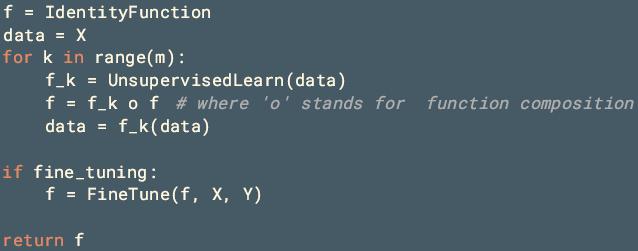
**Problemi supervisionati**: numero di esempi piccoli, solitamente divisi in due parti 🡪 da una parte ci sono le etichette, ma dall’altra c’è solo la descrizione dei dati

**SSL** = Semi Supervised Learning 🡪 ambito di ricerca in cui si costruisce uno strumento che sfrutta i dati privi di etichetta per creare una buona rappresentazione dei dati da sfruttare per completare l’apprendimento

**Gready Layer-Wise Unsupervised Pretraining**

Tecnica che ha messo in evidenza come fosse possibile addestrare reti anche molto profonde con pochi esempi

**Funziona grazie a qualche procedura shallow 🡪** architettura piatta a singolo livello

Ogni livello della struttura profonda di cui apprendere la rappresentazione 🡪 **addestrato indipendente per costruire nuovo output usato come input del livello precedente** (cercando di ricostruirlo, stile AE che fa learning non supervisionato)

**Fine tuning** 🡪 addestrare l’intera rete a partire dai pesi appresi con l’algoritmo, in modo che sia in grado di predire y

Questo algoritmo ha permesso di apprendere dei pesi iniziali per la rete e quindi andare a fare solamente fine tuning solamente sui pesi pre-addestrati

Perché ha questo nome?

* È un algoritmo **greedy** 🡪 nessun tentativo di arrivare ad un ottimo globale
* **Layer-wise** 🡪 procede livello per livello
* Livelli addestrati in modo **non supervisionato**
* Si suppone sia solamente un passo **precedente** all’operazione di fine tuning

**Ci sono alcune sue varianti**:

* Usato per passo di inizializzazione di modelli non supervisionati
* Può essere fatto come un pre-training **supervisionato** 🡪 usa l’output come input per un classificatore che si occupa di stimare le etichette

**C’è il rischio di peggiorare le performance**

**Perché e quando funziona?**

* **Scelta dei parametri iniziale** 🡪 può regolarizzare il modello. Permetteva di trovare minimi locali diversi, ma non è più un problema. Permette di considerare zone dello spazio che naviga la rete che non si sarebbero considerate inizializzando casualmente i pesi 🡪 zone solitamente molto disomogenee, difficili da trovare con inizializzazioni casuali
* **Apprendere la struttura di dati in input** 🡪 aiuta a imparare il mapping tra input e output. Permette di fare predizioni corrette successivamente.

Spesso ciò che si impara in un task supervisionato 🡪 riapplicabile in un altro task

In questo ambito 🡪 se imparo qualcosa in un modello non supervisionato, è possibile che si possa utilizzare in un task supervisionato

Es. img 🡪 apprendimento non supervisionato su img di macchine e moto 🡪 se siamo fortunati, l’apprendimento individua feature interessanti sulla img utili al task supervisionato. Una volta che si usa questa rappresentazione per fare classificazione 🡪 diventa più semplice classificarli grazie alle feature precedentemente individuate

Per ottenere questi benefici 🡪 **dare importanza ai dettagli della rete**

**Non vi è garanzia che le feature apprese in modo non supervisionato** abbiano proprietà interessanti da essere sfruttate

**A volte usando anche info sulle etichette** (supervisionato) **può aiutare a funzionare meglio**

**Quando è utile?**

* **Rappresentazione originale è povera:** dati in input difficile da capire che struttura abbiano 🡪 diventa utile apprendere nuove rappresentazioni più ricche aiuta molto per i task successivi

Es. apprendimento di word embedding del linguaggio naturale 🡪 parole rappresentate da un singolo vettore con tanti 0 e un 1 in una posizione specifica. Confrontando cane e gatto e cane e casa, si avrà una distanza sempre uguale perché tutti i vettori differiscono di una sola posizione 🡪 con la rappresentazione migliorata, invece, si può vedere come casa sia tanto diverso da gatto

* **Usato come** **regolarizzatore 🡪** utile quando il numero di esempi è piccolo o quando il numero di esempi senza etichette è molto grande
* **Funzioni da apprendere molto complicate** 🡪 a differenza di altri regolarizzatori, non forza la funzione da apprendere ad essere più semplice, ma aiuta a scoprire features utili per il task non supervisionato

**Se la funzione è complicata e la struttura dei dati permette di etichettare correttamente i dati 🡪** questo approccio diventa molto utile

**Svantaggi rispetto ad una tecnica di regolarizzazione standard**:

* Difficile da calibrare 🡪 nella maggior parte delle tecniche di regolarizzazione hanno un singolo parametro

**Pre-training non supervisionato largamente abbandonato**

**Pre-trainig supervisionato molto usato**

**Transfer Learning and Domain Adaption**

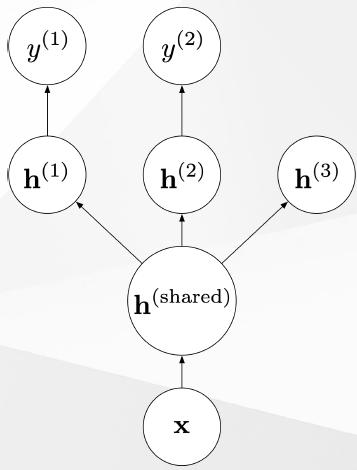
Tecniche che servono quando si ha una **distribuzione** **P1** e si cercano di usare le informazioni ricavate in P1 per migliorare la generalizzazione in un’altra **distribuzione** **P2**

**P1 e P2 devono essere collegate**

**È una generalizzazione del pre-training greedy** 🡪 p1 non supervisionato, p2 supervisionato

**2 task/compiti:**

**1. Transfer learning** 🡪 il learner (algo di apprendimento) si trova di fronte a 2 o più obiettivi di apprendimento 🡪 molti fattori che spiegano le variazioni di p1 sono rilevanti anche per p2 🡪 si assume che p1 e p2 abbiano fattori comuni

In ambiente supervisionato 🡪 input solitamente uguale, ma **etichette differenti**

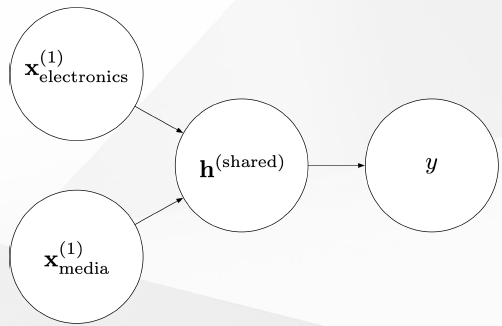
x = input

h(shared) 🡪 **rappresentazione condivisa**

h(1, 2, 3, …) 🡪 **specializzazioni della rappresentazione per risolvere i task y1 e y2**

A volte il **modello condiviso si trova vicino ad un singolo output** 🡪 utile quando gli input sono diversi tra di loro e serve eliminare gli elementi che fa sì che essi differiscano 🡪 crea una sintesi di tutte le rappresentazioni degli input 🡪 usato nello **speech recognition**

**2. Domain Adaption** 🡪 i task rimane sempre lo stesso, ma **cambiano le distribuzioni in input**

**Es.** sentiment analysis 🡪 estrarre dal testo il sentimento dell’autore mentre le scriveva

**Dove sfruttare Transfer Learning?**

**Concept Drift** 🡪 termine usato in apprendimento automatico 🡪 quando il concetto che sottende un certo task si sposta.

**Es.** interesso medio dei mutui in una regione 🡪 ma c’è un cambiamento delle condizioni dell’economia e tutte le variabili imparate dal modello non valgono più, perché non rappresentano lo stato veritiero dell’ambiente

In questo caso 🡪 **transfer learning** utile perché può sfruttare la rete pesata sul vecchio modello che coglie bene alcune caratteristiche del task e adattarlo alla nuova distribuzione

**One-Shot Learning** 🡪 caso estremo in cui si cerca di imparare da un singolo esempio etichettato. Funziona bene se i dati sono distribuiti in uno spazio latente in cui sono ben separate le classi

**Zero-shot learning** 🡪 non si ha nessun esempio etichettato 🡪 non porta da nessuna parte se non si ha una qualche info a latere che permetta di ricostruire l’etichetta

**In questo caso** 🡪 ci sono 3 variabili del task 🡪 x=input y=output (come al solito) e viene introdotta una variabile **T = descrizione del task** 🡪 il modello quindi è allenato sul *p(y|x,T)*

**Es.** riconoscimento delle immagini 🡪 T = descrizione dell’immagine da etichettare (es. gatto = animale a 4 zampe e le orecchie a punta)

**SSL e Causalità**

SSL = Semi Supervised Learning

Causalità 🡪 cercare di identificare le cause che determinano le etichette dei dati

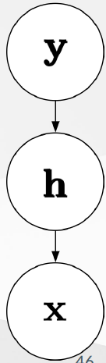
**Cosa rende una rappresentazione meglio di un’altra?**

L’ipotesi è che una rappresentazione ideale sia quella in cui le features della rappresentazione riescono a determinare le cause sottostanti ai dati osservati 🡪 **ipotesi di causalità**

**Riuscire a sbrogliare i fattori causali nella distribuzione in input 🡪 fondamentale per permettere di apprendere p(y|x)**

Quindi, se la rappresentazione **h rappresenta tutti i fattori che causano x** e si suppone che **y** sia collegato ad almeno uno di essi 🡪 p(h,x) = p(x|h)p(h)

I dati in questo caso hanno una **probabilità marginale** 🡪 🡪 E = media (boh)

Ricostruire p(x) può essere utile per scoprire quale sia la h giusta per predire y

**Per modellare h 🡪 si modella p(x)** 🡪 **si impara qualcosa di importante su h**

**Caso particolare** 🡪 y è collegato a una delle cause di h 🡪 imparare h diventa vantaggioso 🡪

**Sfrutta la regola di bayes:**

**Quindi 🡪** p(x) strettamente legato a p(y|x) e la conoscenza di p(x) può essere utile a predire y

**Quindi 🡪** imparare h serve ad apprenere p(y|x) 🡪 perché **h è importante per predire p(x) (dato che p(x) è uguale alla media scritta sopra) e p(x) è importante per predire y**

**Problemi nel trovare h per trovare le cause che sottendono i dati**

Ci sono tante cause e sono difficili da cogliere.

2 strategie seguite per raggirare il problema:

1. usare un segnale **supervisionato** per guidare il processo non supervisionato

2. usare **una rappresentazione h molto grande** a partire dall’apprendimento non supervisionato

Altra strategia 🡪 **cambiare la definizione di cosa sia importante per la rappresentazione**

Usando MSE (Mean Squared Error) può essere problematico 🡪 nel caso delle immagini, specifica che le cause rilevanti all’img sono solo quelle che hanno effetto su un grande numero di pixel 🡪 es. della pallina da ping pong che nella ricostruzione non viene ricostruita dall’AE – Es. dell’orecchio che scompare nella ricostruzione

**Fattori causali** sono **robusti al cambiamento del mondo** perché tendono ad essere costanti nel tempo

**Rappresentazioni distribuite – Apprendimento di rappresentazioni**

Può essere rilevante capire se la rappresentazione è distribuita o meno

**Rappr distribuita** 🡪 ogni oggetto descritto da tante feature, ciascuna delle quali è coinvolta nella rappresentazione di oggetti 🡪 **tante features condivise tra varie rappresentazioni**

**Esempio del quadrato blu a pag. 55**

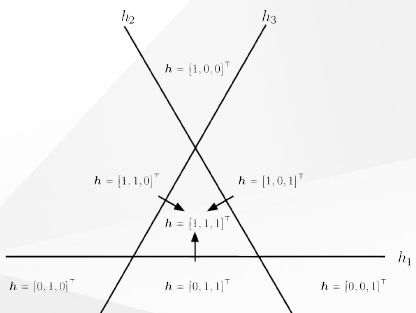
**Spesso 🡪** inducono similarità nello spazio 🡪 oggetti simili dal punto di vista semantico, sono anche vicini dal punto di vista della distanza euclidea dei vettori che li rappresentano 🡪 **caratteristica che manca nelle rappresentazioni puramente simboliche** (o one-hot rappresentation 🡪 distanza tra cane e gatto e tra casa e gatto)

**Vantaggio delle rappr distribuite** 🡪 grazie al fatto che si combinano tra loro 🡪 riescono a rappresentare un numero esponenziale di elementi

**Generalizzazione**

La cosa interessante è che grazie al fatto che possono essere combinate tra di loro riescono a rappresentare moltissimi oggetti (un numero esponenziale).

Senza questo tipo di proprietà per approssimare una distribuzione che non ha questo tipo di caratteristica io devo assegnare tanti esempi su ciascuna di queste zone, quindi sarebbe un numero esponenziale di esempi.

**Posso usare un numero minore di esempi per dare supporto statistico ad una certa zona contribuiscono esempi che condividono parte della rappresentazione 🡪** lo stesso esempio partecipa a più zone e quindi con meno esempi riesco a dare più supporto statistico con meno esempi. Sebbene questi modelli riescano a rappresentare bene molte zone differenti la capacità di rappresentazione è limitata. Non c’è flessibilità infinita in quanto si rischierebbe di fare overfitting. 

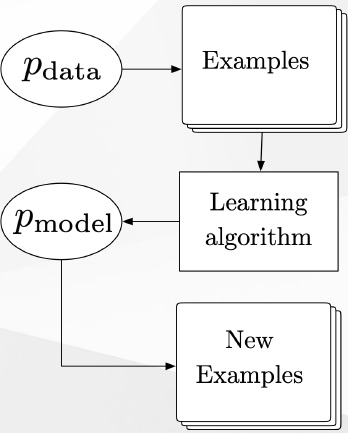
**VC dimension 🡪** modo formale per catturare l’idea che un modello è in grado di rappresentare tante possibili funzioni 🡪 **dimensione del più grande insieme che può essere “frantumato” (shattered) dal modello**

Un insieme si dice **shattered** da un a famiglia di funzioni F se per ognuna delle possibili etichettature dei punti della funzione 🡪 esiste un insieme di funzioni dentro ad F che può etichettare correttamente tutti i punti

**Rappresentazioni distribuite sono interpretabili** 🡪 non sempre vero

**Nell’apprendere distribuzioni distribuite** 🡪 guadagno esponenziale dovuto alla profondità dei sistemi 🡪 va beh fottesega

# 4. gan: Generative Adversarial networks

**Generative Models (GN)** = **Modelli Generativi 🡪** sistema che prende in input un insieme di esempi estratti da una distribuzione ed impara a rappresentare tale distribuzione, in particolare impara ad estrarre nuovi esempi da tale distribuzione

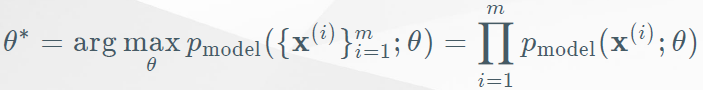
I GN si fermano una volta appreso e realizzato il model 🡪 da qui si possono etichettare gli esempi e estrarne di nuovi

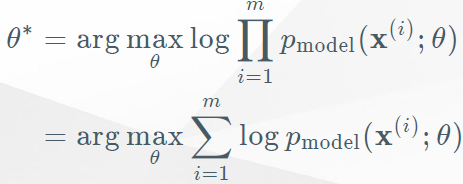
**Vantaggi teorici GN:**

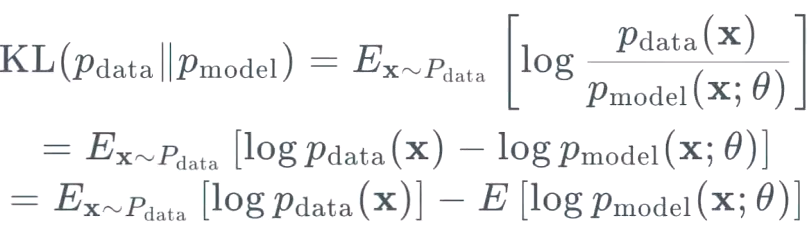
* Abilità di rappresentare distribuzioni di grandi dimensioni
* Nel campo del Reinforcement Leraning (RL) (es. far imparare ad un agente [robot] di muoversi in un ambiente che non conosce) 🡪 GN molto utili perché permetto di fare assunzioni su un modello creato dai GN 🡪 query fatte da sistemi RL
* Possono essere usati come base per i sistemi SSL

**Vantaggi applicativi GN:**

* GAN molto bravi a predire output multimodali (es. dell’orecchio di un umano ricostruita in malo modo da MSE, mentre GAN lo modella meglio)
* Super resolution 🡪 da img a bassa risoluzione a img con più alta risoluzione
* GAN costruiscono spazi latenti che hanno varie utilità 🡪 es. immagine che viene spesso deformata in modo poco legato alla realtà nei modelli classici 🡪 GAN immagina una forma simile a quella di partenza

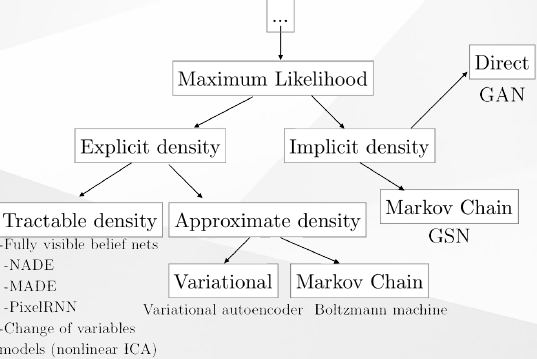
**Vantaggi rispetto ad altri modelli generativi basati su maximum likelihood:** modo di trovare parametri per un modello di apprendimento che massimizza la verosimiglianza del data set in base ai parametri che si stanno analizzando

Quasi sempre si l’ottimizzazione, però, è fatta in log-space:

**Divergenza KL** 🡪 usata perché minimizzare divergenza KL ( = massimizzare la maximum likelihood della distribuzione e 

**E = valore atteso**

L’ultimo termine della formula E[log pmodle(x;)] 🡪 maximum likelihood rispetto al modello

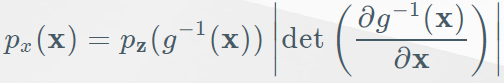
Quando si fa la massimizzazione di KL a partire dalla formula ottenuta nell’ultima riga 🡪 **il primo termine non dipende dai parametri**  che si stanno guardando durante la massimizzazione 🡪 **ci si concentra solo sul secondo termine 🡪 massimizzare la maximum likelihood che avendo un meno davanti = minimizzare la KL**

**Explicit Density Model** 🡪 Sottogruppo in cui la **densità è trattabile**

**1. Fully Visible Belief Nets (FVBN)** 🡪 fa ricorso alla **rule chain applicata n volte** per calcolare la probabilità di x a partire dal modello:

Questi modelli sono alla base di sofisticati GN realizzati da un gruppo di Google

**Problema di questi modelli:** la formula è molto costosa da calcolare 🡪 O(n) 🡪 ottimo nella generazione di risultati, **ma molto dispendioso**

**2. Nonliear Independent Components Analysis = Nonlinear ICA** 🡪 utilizzo di funzione non lineare per mappare x in un nuovo spazio per poi gestire la probabilità nel nuovo spazio

**Determinante di una matrice:** misura quanto la matrice distorce i volumi dello spazio su cui è applicata 🡪 come conseguenza si ottiene anche quando la matrice è invertibile

**Cosa serve tale determinante** nella formula: la probabilità di un singolo punto è sempre infinitesimale (sempre a zero in pratica) 🡪 si considera quindi un volume in cui è incluso il punto che mi interessa e si calcola la probabilità di quel volume ottenuto 🡪

**Se si pone x = g(z) 🡪** g(z) trasforma tutto il volume prima creato

Quindi **il determinante a che cazzo serve diocane?**

Si considera una singola variabile random x e si ottiene

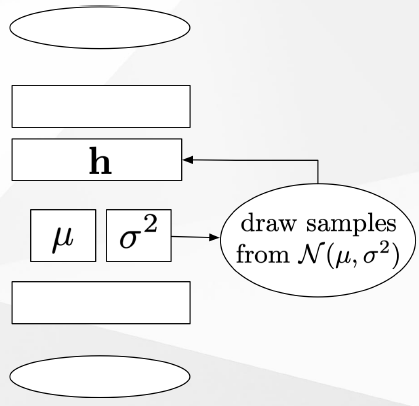
Da qui, con i calcoli, si arriva a

**Tutto questo per dire 🡪** nonlinear ICA cerca di risolvere il problema di star lavorando su una distribuzione complicata, utilizzando una **trasformazione che semplifica il lavorare su tale distribuzione**

**Problema 🡪 la funzione g deve essere invertibile!**

**Modelli espliciti che richiedono l’approssimazione**

**1. Approssimazione variazionale** 🡪 se non si riesce a trattare una distribuzione, si usa una sua versione trattabile che la approssima

**Variational AE 🡪** cambia la rappresentazione h dell’autoencoders 🡪 **h = insieme di numeri estratti da una particolare distribuzione usata per approssimare quella che non si riesce a trattare** 🡪 si usa spesso una gaussiana multivariata di cui si devono trovare i parametri 🡪 una volta trovati i parametri si genera h estraendo a caso dalla gaussiana e poi ci sarà la parte di rigenerazione dell’input a partire da h 

**Questi AE funzionano bene, MA** hanno performance peggiori rispetto alle GAN

**2. Approssimazione tramite catena di Markov** 🡪 trova un **operatore di transizione q** che estrae dalla distribuzione completa delle approssimazioni e dopo molte iterazioni si ottiene una buona approssimazione della distribuzione

**Problema: molto lento e molto peggiore delle GAN**

**Modelli impliciti**

**1. Generative Stochastic Networks** 🡪 non cerca di modellare una distribuzione x1,.., xn 🡪 usa le markov chain imparando a costruire **q** in maniera ottimale permette di lavorare con la distribuzione originale

**Problema: generazione degli esempi molto lenta**

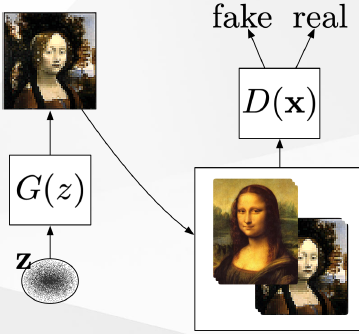
**2. GAN** 🡪 presentano svariati **vantaggi:**

* Modellazione implicita della distribuzione, estraendo dati con metodo veloce con un **singolo passo** 🡪 generazione di esempi in parallelo
* Funzione che genera gli esempi ha **pochissime restrizioni**
* Non necessitano di approssimazioni con le catene di Markov
* Non servono bound variazionali (come quelli usati nei variational AE)
* Dal punto di vista dei risultati 🡪 sembra produrne di migliori per quanto riguarda i test soggettivi (Es. questa immagine ti sembra realistica? Sì o no)

**Funzionamento GAN**

Ha un’idea di fondo basata sul “gioco” tra due attori:

* **Generatore** 🡪**G(z)**: prende in input un vettore solitamente di numeri casuali (= rumore bianco) che viene sfruttato per creare una nuova immagine passata al discriminatore insieme a delle immagini reali
* **Discriminatore** 🡪 **F(x)**: funzione che ha come scopo quello di determinare quali delle immagini che riceve in input (reali e quelle generate da G) sono reali e quali generate dal generatore 🡪 **fake o real**

**G(z)** è messo sotto dura prova perché deve creare img che non ha mai visto e sfrutta il discriminatore per capire quanto è lontano dalla realtà

**G e D = funzioni differenziabili** 🡪 solitamente sono NN addestrate con la discesa del gradiente

**G 🡪 prende in input un vettore z** 🡪 ed è definito dai parametri passati a G che è una NN, quindi sono i pesi della NN. **Output G** 🡪 valore estratto dallo stesso spazio di x

**D 🡪 prende in input un oggetto x** (es. un’immagine) ed è definita anch’essa in termine dei parametri 🡪 **Output D = {fake,real} 🡪** in realtà restituisce numeri tra 0 e 1: più vicino a 0 = fake

**G e D** 🡪 hanno funzioni di loss J definite in termine dei parametri e , ovvero:

**Ottimizzando le loss function 🡪** le due funzioni **non possono interferire sui parametri dell’altra funzione 🡪**  può intervenire solo su

**Il gioco tra G e D è a somma nulla** 🡪 ogni volta che uno dei due giocatori ha un tipo di vantaggio, questo si tramuta in uno svantaggio di uguale entità per l’altro giocatore

**Es.** degli scacchi 🡪 se faccio una mossa che mi avvantaggia, sto svantaggiando il mio avversario

**Soluzione gioco a somma nulla = punto di equilibrio di Nash** 🡪 punto nello spazio ( che è un minimo locale sia rispetto a che rispetto a

**Si ottiene una soluzione quando** 🡪 il primo giocatore non può fare nulla agendo sui suoi parametri per migliorare la situazione e il secondo giocatore non può far niente per migliorare la situazione su lui stesso

**Addestramento GAN 🡪** utilizza discesa del gradiente stocastico simultaneamente su e su Seguendo due fasi:

* **Estrazione di due mini-batch** (uno dai dati reali e uno richiesto al generatore)
* **Valutazione di**  e si **aggiorna**  usando i gradienti che arrivano da e usando i gradienti che arrivano da

A volte si fanno alcuni passi in più sul discriminatore prima di tornare dal generatore (es. uno step sul generatore ogni 5 sul discriminatore).

Durante l’apprendimento sia G che D **si muovono nel buio**, anche il discriminatore non sa distinguere il vero dal falso all’inizio 🡪 ci saranno errori grossolani all’inizio.

**Funzione di loss di G 🡪 usata da D per migliorare la distinzione**

**Funzione di costo del discriminatore**

Se D(x) è vicino a 1 🡪 logaritmo = 0 🡪 non si paga tanto 🡪 il discriminatore sta agendo bene

Prima componente, quindi, indica che 🡪 **si viene penalizzati** quando un esempio **reale viene etichettato come non reale**

Seconda componente 🡪 prende il valore medio del valore atteso secondo la distribuzione z 🡪 (1-D(G(x)) 🡪 quando il discriminatore pensa che G(x) è reale 🡪 valore vicino a 1 🡪 logaritmo vicino a 0, quindi -infinito 🡪 **si paga pegno quando il discriminatore dice che l’esempio è reale: ovvero vicino a 1**

**Questa formula = cross-entropy applicata a questo problema**

**Cross-entropy** 🡪 misura quanti bit in media si devono mandare quando il codice che si sta inviando è ottimizzato per una distribuzione q, mentre i dati seguono una distribuzione p. 

**In genere misura la lunghezza media del messaggio che è più lunga di quella ottimale**

La differenza tra la cross entropy e l’entropia ottimale è la **divergenza KL**

**Nel caso ideale** 🡪 cross entropy si riduce all’entropia H(p), succede quando **p diventa uguale a q**

**Minimizzazione cross-entropy = ottimizzazione modello per la distribuzione p**

**Cross-entropy per classificazione binaria** (che è il caso delle GAN) ha la seguente formula

Per le GAN 🡪 quando **y = 1** 🡪 esempio **reale** e l’y segnato corrispondente è D(x) dove x è l’esempio reale di cui si sta parlando

Con y = 1🡪 secondo termine scompare 🡪 primo termine riamne solamente Ex[log ]

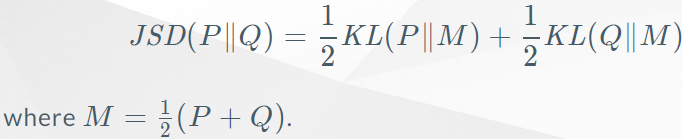
Con **y = 0** 🡪 si ha a che fare con un esempio **fake**

Y = 0 🡪 primo termine scompare 🡪 la predizione è data da dove z fa parte del rumore casuale dato in input a G 🡪 **uguale alla funzione di costo scritta sopra**

**Funzione di costo del Generatore**

È semplicemente l’inverso della lost del discriminatore 🡪 questo per il fatto di trovarsi in un **gioco a somma nulla**

**Tutto il gioco può essere riassunto in una funzione minimax:**

**Questo setup per il gioco è facilmente analizzabile teoricamente 🡪** il gioco corrisponde a minimizzare la **divergenza di Jenesn-Shannon**:

La **il gradiente della loss è basso quando il discriminatore rifiuta gli esempi del generatore con un’alta confidenza 🡪** il generatore non potrà più migliorare

**Per risolvere il problema della loss** 🡪 **loss euristica non-saturating del generatore**

**Penalizza il generatore quando il discriminatore dirà 0 per un’immagine generata dal generatore**

**Quando si è molto bravi a generare img che il generatore non sa più distinguere 🡪** il gradiente è zero, ma tanto si è già molto bravi.   
**Quando il discriminatore capisce esattamente quali sono le img generati dal generatore 🡪** la loss è alta e si riesce a migliorare rapidamente

**Cambiando la loss del generatore** (quella del discriminatore rimane uguale a J(D)) 🡪 **non si ha più un gioco a somma nulla**

**Perché i GAN funzionano?**

Buone performance grazie alla minimizzazione della divergenza Jensens-Shannon e non della divergenza KL 🡪 **KL non è simmetrica** 🡪 maxiumum likelihood minimizza KL(pdata | pmodel)

**Minimizzare JS 🡪 simile a minimizzare KL(pmodel | pdata)**

Dice altre cose che però non ho ben capito, ma sostanzialemente parla della differenza tra KL e reverse KL

**Negli ultimi anni** 🡪 non può essere la ragione giusta quella della divergenza JS:

* Sono state implementata delle GAN che utilizzano il maximum likelihood giusto (non KL invertito) come funzione da ottimizzare 🡪 **non hanno avuto problemi nello scegliere un numero ridotto di mode quando generano nuovi esempi**
* Comunque si generino le GAN 🡪 si concentrano su poche mode diverse 🡪 meno mode di quante sarebbero permesse dalla capacità del modello

**La vera ragione per cui funzionino non si sa ancora**

**Si ipotizza 🡪** il fatto che le GAN si concentrino su poche mode sia dettato da un **difetto nella procedura di training** (non un problema della divergenza, quindi)

**Ci sono varie ragioni per cui succede**

**DI CERTO C’E’ CHE LE GAN FUNZIONANO BENE**

**DCGAN** 🡪 Deep Convolutional GAN

Fa la batch normalization in quasi tutti i livelli di G e D

L’ultimo livello di G e il primo di D non hanno la normalizzazione 🡪 per permettere al discriminatore di imparare la distribuzione dei dati nei primi livelli

**Batch normalization** 🡪 procedimento che dà ottimi risultati nelle NN 🡪 solitamente nelle NN si fa una normalizzazione dei dati prima di darli in pasto alla NN 🡪 la batch norm calcola la normalizzazione **per ogni singolo batch di dati che arriva e in più i fattori di normalizzazione vengono calcolati in tutti i livelli**

**WGAN** 🡪 **sono più facili da addestrare delle GAN**

Con distribuzioni reali 🡪 si hanno problemi di convergenza

Quindi proposta di una **nuova funzione di loss**:

Date due distribuzioni (considerate come cumuli di terra da spostare) 🡪 la distanza tra esse è data dalla minima quantità di terra che posso spostare da una all’altra per far sì che diventino uguali.

L’infimum è altamente non trattabile

**VA BEH MA CHE CAZZO E’ STA MERDA 🡪** fa cose con formule e cose varie

Tutto sto schifo per dire che WGAN hanno i seguenti **vantaggi**:

* La nuova metrica di loss 🡪 è più interpretabile 🡪 correla la loss con il fatto che il generatore stia convergendo meno 🡪 correla con la qualità delle img del generatore
* **Semplifica processo di addestramento** 🡪 addestrando il generatore sotto una particolare loss quando D è vicina all’ottimalità
* **Permettono al discriminatore di essere addestrato fino all’ottimalità** 🡪 quando è vero WGAN rende molto più robusto l’addestramento del generatore