Information Dropout: Learning Optimal Representations Trhough Noisy Computation
- *Analisi Teorica* -

Nicolò Cavedoni

March 28, 2021

1 Introduzione

Gli autori Achille e Soatto rivisitano la teoria dell'Information Bottleneck di Tishby e compagni [1] sotto una luce deep learning, mostrandone le correlazioni con il rumore, il dropout e con le macchine variazionali. Con la nuova teoria dell'*Information Dropout* è possibile ottenere migliori rappresentazioni dei dati e quindi accelerare il processo di apprendimento, semplicemente aggiungendo rumore moltiplicativo alle funzioni di attivazione di una rete.
Studiando le prestazioni di una macchina dal punto di vista dell'Information Theory, si può notare come la più classica funzione di perdita cross-entropy sia strettamente correlata alle rappresentazioni dei dati che la rete apprende. Gli autori dimostrano che aggiungendo un termine di regolarizzazione alla suddetta funzione è possibile ottenere rappresentazioni migliori, che soddisfano delle proprietà desiderate, e di conseguenza dei miglioramenti nell'apprendimento. Questo termine regolatore è strettamente legato alla pratica del dropout, in quanto lo si può interpretare come un'iniezione di rumore nelle funzioni di attivazione dei neuroni. La nuova funzione di perdita può essere quindi minimizzata grazie all'Information Dropout presentato in questo paper.
Gli autori evidenziano la correlazione con l'inferenza variazionale e il relativo autoencoder, mostrando che l'Information Dropout è una generalizzazione del modello VAE, e sostenendo che le prestazioni possono migliorare se si adotta un priore fattorizzato da componenti indipendenti.
L'obiettivo di questa analisi è quello di spiegare tutti questi punti nelle teorie e i modelli che li definiscono, sviscerare i passaggi poco chiari del paper e mettere il lettore nelle condizioni di poter valutare criticamente l'elaborato di Achille e Soatto.
Disclaimer: Nella trattazione verranno dati per scontati i concetti base del machine learning e del deep learning, assieme alle nozioni principali di statistica che li definiscono.

2 Teoria dell'Informazione

La Teoria dell'Informazione è una disciplina che si occupa di studiare e misurare la quantità di informazione contenuta nei messaggi e tutto ciò che ne riguarda la trasmissione e comunicazione. L'informazione è principalmente misurata secondo la sua *Entropia*

$$\text{Entropia} \quad H(X) = \mathbb{E}_x[-\log p(x)] = -\sum_x p(x) \log p(x) \tag{1}$$

che ha quindi una formula logaritmica e una legata all'autoinformazione $I(x) = -\log p(x)$, equivalenti data la definizione di valore atteso $\mathbb{E}$. L'entropia misura la quantità di incertezza relativa alla variabile aleatoria x e si esprime di norma come "il numero di bit necessari a trasmettere un'informazione" (utilizzando un logaritmo in base 2). Nel momento in cui bisogna trasmettere un certo numero di bit, se alcuni di questi sono già noti prima della trasmissione ($p(x) = 1$) è logico pensare che non portino alcuna informazione a chi li riceve ($H(x) = 0$), esattamente come se il loro valore non potesse mai essere trasmesso. In quanto misura di incertezza, l'entropia segue le leggi della probabilità, come

$$\begin{aligned} \text{Entropia Congiunta} \quad H(X,Y) &= \mathbb{E}_{x,y}[-\log p(x,y)] \\ &= -\sum_{x,y} p(x,y) \log p(x,y) \end{aligned} \tag{2}$$

$$\begin{aligned} \text{Entropia Condizionata} \quad H(X|Y) &= \mathbb{E}_x[H(X|Y=y)] \\ &= -\sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(y)} \end{aligned} \tag{3}$$

$$\begin{aligned} \text{Cross-Entropy} \quad H_x(P;Q) &= \mathbb{E}_x[-\log q(x)] \\ &= -\sum_x p(x) \log q(x) \end{aligned} \tag{4}$$

Un concetto utilizzato spessissimo nel paper è quello di entropia relativa, più spesso chiamata *Divergenza di Kullback-Leibler*. È un modo di confrontare la distribuzione di probabilità $p(x)$ considerata "vera" con un'altra distribuzione approssimata $q(x)$ per la stessa variabile aleatoria. Si ottiene da una semplice differenza tra distribuzioni nella formula dell'entropia

$$D_{KL}(P||Q) = \mathbb{E}_x[\log p(x) - \log q(x)] = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{5}$$

e sempre ragionando in termini di bit e di $\log_2$ può essere interpretata come la quantità di bit che ci si aspetta di perdere dall'approssimazione della distribuzione $p(x)$ verso $q(x)$.

L'ultima misura di base che dobbiamo conoscere è l'*Informazione Mutua*, ovvero la quantità di informazione che possiamo ottenere di una v.a. osservandone una seconda. È una misura di interdipendenza di due v.a. e si calcola come

$$I(X;Y) = \sum_{x,y} \log \frac{p(x,y)}{p(x)p(y)}$$
$$I(X;Y) = D_{KL}(p(x,y)||p_x p_y) \tag{6}$$

Sebbene non lo sembri dalla definizione, la mutua informazione è simmetrica, quindi $I(X;Y) = I(Y;X)$, ed è strettamente connessa con l'entropia. La Figura 1 mostra la relazione che intercorre tra queste due quantità, rispettivamente con due e tre variabili. La dicitura $I(X;Y|Z) = I(Y;X|Z)$ rappresenta l'informazione mutua condizionata, ovvero l'interdipendenza fra due variabili osservabile dal punto di vista di una terza, quando realizzata. La sua formula non ci interessa, ma ci servirà sapere che quando le tre variabili sono in una catena di Markov $X \longrightarrow Y \longrightarrow Z$ allora $I(X;Z|Y) = 0$.

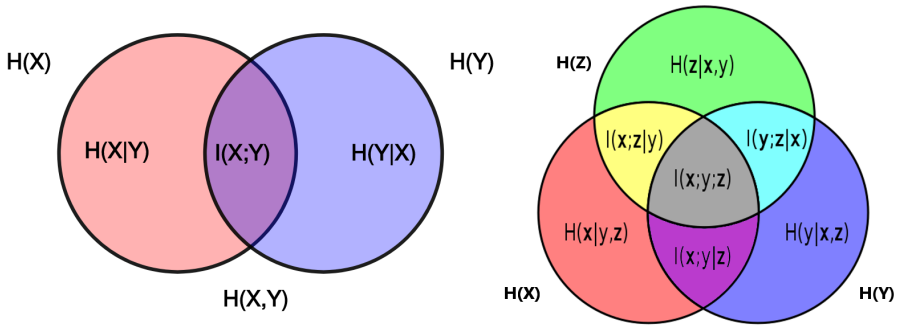

Figure 1: Relazione tra Entropia e Informazione Mutua

3 Inferenza Variazionale

L'inferenza bayesiana è un ottimo modo di calcolare l'incertezza nelle previsioni di un sistema di apprendimento automatico. Si basa tutta sulla definizione di probabilità a posteriori $p(z_i|x) = \frac{p(x|z_i)p(z)}{\int_i p(x|z_i)p(z)dz}$. L'ostacolo di questo tipo di calcolo è che il denominatore, equivalente alla probabilità marginale dei dati osservati $p(x)$ è spesso intrattabile, dunque ci sono alcuni approcci per girarci intorno e ottenere un'approssimazione verosimile. L'*inferenza variazionale* è uno di questi, e si occupa di costruire direttamente un modello approssimativo del posteriore, quindi $q(z) \approx p(z|x)$, dove x sono i dati in input a una macchina, nel caso del machine learning. Abbiamo già visto una misura efficace per calcolare la diversità fra due distribuzioni, ed è la divergenza KL, dunque minimizzare questa divergenza equivale a massimizzare la somiglianza tra la distribuzione approssimata e la vera distribuzione a posteriori. Poichè minimizzare D_{KL} è computazionalmente difficile, il metodo più utilizzato è chiamato ELBO (Evidence Lower BOund), la cui massimizzazione equivale alla minima divergenza KL.

$$\text{ELBO}(X) = H_z(Q) - H_z(Q; P(Z, X)) = -\sum_z q(z) \log q(z) + \sum_z q(z) \log p(z, x)$$

$$\text{ELBO}(X) = \log p(x) - D_{KL}(q(z)||p(z|x)) \tag{7}$$

ovvero la differenza fra l'entropia di $q(z)$ e la sua cross-entropy con la probabilità congiunta $p(z, x)$. La prima definizione si può ottenere partendo da $\log p(x)$ e sfruttando la disuguaglianza di Jensen, mentre la seconda si ottiene da una serie di trasformazioni di $D_{KL}(q(z)||p(z|x))$ che la conducono ad essere $\log p(x) - \text{ELBO}(x)$. Da notare che la massimizzazione dell'ELBO non equivale alla minimizzazione della funzione di perdita di una macchina, in quanto l'ELBO non è una misura di errore. Il modello che applica l'inferenza variazionale e quindi basato sul calcolo dell'ELBO e sulla stima del posteriore si chiama *Autoencoder Variazionale*.

3.1 Autoencoder Variazionale

Un VAE (Autoencoder Variazionale) è un modello piuttosto recente [2] di rete neurale generativa, cioè una macchina che apprende la distribuzione di probabilità sul suo spazio degli input, a differenza dei modelli discriminativi che invece apprendono il confine che separa le classi da discriminare. Una volta addestrata una rete generativa è infatti possibile estrarre dei campioni dalla distribuzione appresa, e quindi generare dei nuovi dati (spesso e volentieri sotto forma di immagine) a partire dal set di partenza. Le features apprese sono chiamate *variabili latenti* $z = f(x)$, in quanto non sono conosciute a priori, ma nella prossima sezione introdurremo il termine "rappresentazione" per definirle. Un VAE è composto da un encoder, che processa i dati in input riducendone la dimensionalità ed estrae le variabili latenti, e da un decoder che sostanzialmente effettua il processo inverso, generando nuovi dati $\tilde{x}$ dalla distribuzione di probabilità appresa. L'encoder è una rete neurale con pesi e bias θ che comprime l'input in uno

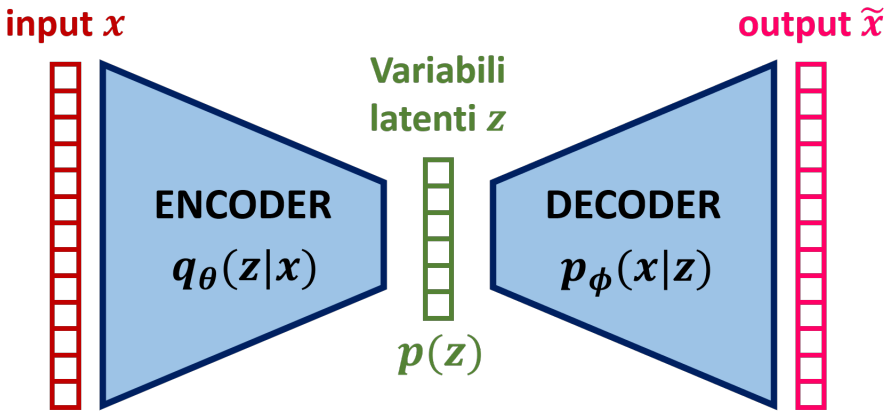

Figure 2: Struttura di un Autoencoder Variazionale

spazio latente di dimensione inferiore. Per motivi di overfitting, questo spazio latente necessita di essere regolarizzato, dunque $q_\theta(z|x)$ è modellata come una Gaussiana, peculiarità del VAE rispetto agli autoencoder generici. Poichè $p(z)$ non è nota, si assume che segua una distribuzione normale standard $\mathcal{N}(0,1)$, dunque l'addestramento dell'encoder è atto a minimizzare la differenza tra la distribuzione dei dati codificati $q_\theta(z|x)$ e la normale standard. Come sopra, il modo di affrontare questo calcolo è quello di massimizzare l'ELBO, ma poichè classicamente l'addestramento avviene tramite minimizzazione di una funzione di costo, si arriva a concludere che questa debba essere l'opposto dell'ELBO.

$$\text{ELBO}_i(\theta,\phi) = \mathbb{E}_{z\sim q_\theta(z|x_i)}[\log p_\phi(x_i|z)] - D_{KL}(q_\theta(z|x_i)||p(z)) \text{ dunque} \tag{8}$$

$$\text{loss } l_i(\theta,\phi) = \mathbb{E}_{z\sim q_\theta(z|x_i)}[-\log p_\phi(x_i|z)] + D_{KL}(q_\theta(z|x_i)||p(z)) \tag{9}$$

di cui riconosciamo entrambi i termini: Il secondo è proprio la divergenza tra la distribuzione ideale delle variabili latenti e quella prodotta dall'encoder, mentre il primo altro non è che la cross-entropy fra la distribuzione codificata e quella decodificata. Questa perdita è calcolata su un singolo datapoint i, ma ovviamente la loss totale è la somma delle perdite su tutti i datapoint x_i. L'entropia incrociata misura il numero di bit necessari per codificare un informazione proveniente da una distribuzione tramite una seconda distribuzione, quindi è l'ideale per addestrare il decoder a imparare a ricostruire i dati. Il decoder è una seconda rete neurale con pesi e bias φ che prende in ingresso le rappresentazioni e genera degli output campionando randomicamente lo spazio latente. I meccanismi di compressione e decompressione sono naturalmente soggetti a perdite di informazione, misurata come la differenza tra x e l'output del VAE $\tilde{x}$, dunque l'obiettivo di ogni autoencoder è quello di minimizzare questa perdita ed essere in grado di generare output decodificati casualmente che rispettino le caratteristiche apprese il più fedelmente possibile.

4 Dropout, ReLU, Softplus

Nell'ambiente del Deep Learning non sempre i dataset sono sufficientemente grandi e vari da addestrare una rete in maniera ottimale. Quando accade, il problema principale è dato dall'overfitting della macchina, ovvero l'apprendimento di features troppo dipendenti dall'input e poco generiche, che porterebbero la macchina a fallire il task non appena sottoposta a dati estranei al training set. Il *dropout* [3] è una tecnica utilizzata per prevenire questo problema, semplicemente "spegnendo" dei nodi casuali all'interno di uno strato e ignorando dunque i dati che prenderebbero in input e il loro output. Per ottenere un apprendimento generico infatti si può prendere la media dei risultati ottenuti dall'addestramento su tanti dataset o, alternativamente, quella dell'addestramento di macchine diverse sugli stessi dati. Il dropout simula questo secondo caso, scartando nodi casuali ogni volta e quindi riconfigurando la rete ad ogni iterazione. È un po' come inserire del rumore all'interno del processo di apprendimento, rendendolo più sporco ma anche più efficace. Il dropout viene effettivamente messo in pratica moltiplicando elemento per elemento tutti i nodi di uno strato con una matrice binaria (maschera). La probabilità di un bit nella maschera di essere acceso o spento è detta "dropout rate", che classicamente è settata a 0.5. Kingma propone un'interpretazione del dropout che viene sposata dagli autori, ovvero la tecnica di aggiungere rumore moltiplicativo ai dati in input di ogni layer, preso da una distribuzione di rumore (Gaussiana $\mathcal{N}(1, \alpha)$ nel caso di Kingma e di questo paper) [2].
Un altro modo di migliorare le prestazioni durante l'apprendimento è quello di adottare funzioni di attivazione particolari. Le più classiche funzioni sigmoide e tangente iperbolica infatti tendono a saturare, ovvero ridurre la maggior parte degli input ai due estremi del loro output, rimanendo sensibili attorno al valore medio. Questo rende il processo di bilanciamento dei pesi più complicato, oltre che prestarsi al problema della scomparsa del gradiente. Per questo motivo si possono utilizzare funzioni di attivazione "rettificanti" (ovvero di forma più simile a una retta) che aumentino la sensibilità, implementate su unità (neuroni) chiamate *ReLU*. Una ReLU ha quindi una funzione di attivazione del tipo $f_{ReL}(x) = \max(0, x)$, che la rende molto più simile a un neurone reale, che non può essere "meno che spento", e che mappa ogni input su una scala infinita di output, diventando molto più sensibile ai cambiamenti e risolvendo la scomparsa del gradiente. Tutte queste qualità hanno condotto la funzione di attivazione rettificante a soppiantare sigmoide e tanh nell'addestramento di reti neurali deep [4]. Dal concetto di funzione ReL sono nate diverse varianti negli anni, e nel testo si cita la funzione *Softplus*, la cui particolarità è di essere derivabile in una sigmoide

$$f_{Softplus}(x) = \ln(1 + e^x) f'(x) = \frac{e^x}{1 + e^x} = \text{Sigmoide} \tag{10}$$

Rispetto alla ReL, la Softplus è quindi molto più differenziabile e sensibile nei valori attorno a zero, come si evince dalla Figura 3. Tuttavia le ReLU sono comunque più utilizzate per la loro semplicità computazionale. È infatti molto più

immediato computare il massimo fra due numeri rispetto alla Softplus, e poichè i neuroni vengono attivati ripetutamente e in grandissime quantità, anche una differenza di millisecondi si ripercuote sulle prestazioni dell'intero addestramento.

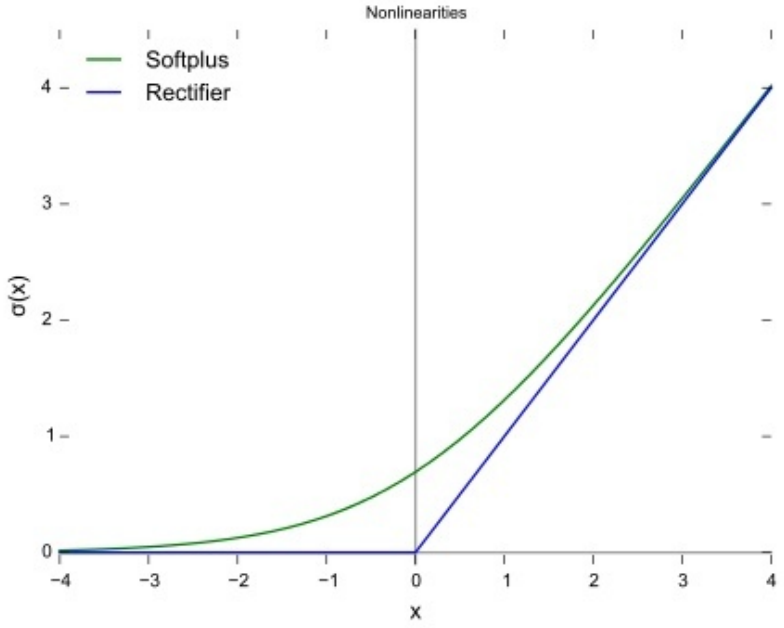

Figure 3: Funzioni di attivazione rettificatrice lineare (ReL) e Softplus a confronto

5 Rappresentazioni

Il termine "Representation Learning" nasce dalla necessità di migliorare le prestazioni degli algoritmi di machine learning. L'efficienza dell'apprendimento è infatti molto dipendente dal modo in cui i dati in ingresso vengono presentati alla macchina. Spesso e volentieri, l'input è molto grande e dispersivo rispetto quello che ci interessa che la macchina impari, e per questo motivo si cerca una *rappresentazione* dei dati in input che semplifichi il processo di apprendimento senza però disperdere informazioni importanti. L'apprendimento di una rappresentazione avviene in maniera non supervisionata nel momento in cui l'input x viene ridotto di dimensione negli strati nascosti di una rete, come abbiamo visto accadere per il VAE. Classificazione o regressione che sia, il task in una rete neurale è sempre svolto in maniera supervisionata, e in quanto tale ci interessa imparare la distribuzione $p(y|x)$ del task y a una specifica realizzazione di

x. Per semplificare l'operazione, vogliamo quindi trovare una variabile aleatoria z che sia una rappresentazione di x, e che sia una "buona" rappresentazione. Per essere tale, la distribuzione di z deve innanzitutto essere definita completamente dalla distribuzione condizionata $p(z|x)$, o comunque z deve essere in funzione solo di x. Questa definizione conduce naturalmente alla costruzione di una catena di Markov del tipo $y \longrightarrow x \longrightarrow z$. Ma non è sufficiente che z sia una rappresentazione, deve anche essere sufficiente e minimale per il task y.

$$z \text{ sufficiente per } y \longrightarrow I(z;y) = I(x;y) \tag{11}$$

$$z \text{ minimale per } y \longrightarrow I(z;x) < I(z_i;x), \forall z_i, z \in Z \text{ rappresentazioni di } x \tag{12}$$

Dunque vogliamo una rappresentazione dei dati che non abbia dispersione di informazioni rispetto all'input originale, ma anche che contenga solo le informazioni necessarie a svolgere il task, senza essere abbondante. La condizione di sufficienza può essere riscritta applicando le identità secondo la Figura 1, dunque $I(x;y) = H(y) - H(y|x)$ mentre $I(z;y) = H(y) - H(y|z)$. L'uguaglianza chiede che $H(y|x) = H(y|z)$ che è valida quando le intersezioni $I(x;y|z)$ e $I(y;z|x)$ coincidono. Tuttavia la catena di Markov $y \longrightarrow x \longrightarrow z$ che definisce z come rappresentazione di x equivale a dire che $I(y;z|x) = 0$, dunque la condizione di sufficienza si può esprimere come

$$z \text{ sufficiente per } y \longrightarrow I(x;y|z) = 0$$

Un'altra proprietà desiderata che si vorrebbe ottenere da una rappresentazione è il *disentanglement*, ovvero la minor dipendenza possibile fra tutte le feature apprese dalla rappresentazione z. Ad esempio, in un modello generativo (una macchina che crea dati ad alto livello, come musica o immagini) è preferibile quando le componenti dell'output sono identificate separatamente, come possono essere le parti del corpo di un essere umano o le figure in primo piano, secondo piano e lo sfondo in una foto. Una rappresentazione sufficientemente districata (disentangled) permette alla rete di apprendere le features separatamente su diversi neuroni, migliorandone le prestazioni generali. La misura del disentanglement è data dalla Correlazione Totale (o multi-informazione) tra le features che compongono z, ovvero

$$\mathrm{TC}(z) = D_{KL}(p(z)||\prod_i p(z_i)) \tag{13}$$

che non è altro che la divergenza fra la distribuzione di z e una costruita con le distribuzioni di tutte le features che la compongono. Idealmente, la rappresentazione è completamente districata quando $\mathrm{TC}(z) = 0$, ergo quando le componenti z_i sono tutte indipendenti fra loro.

6 Modellazione del Problema

Per studiare il miglior trade-off tra accuratezza e complessità, Tishby [1] propone una funzione chiamata *Information Bottleneck*. La formula generale vuole che, conosciuta $p(x, y)$, il problema di bilanciamento possa essere racchiuso in una funzione lagrangiana (uno dei possibili metodi di risoluzione di un problema di ottimizzazione) che prevede di

$$\text{minimizzare } \; I(X;Z) - \beta I(Z;Y) \; \text{ rispetto a } p(z|x) \tag{14}$$

Nata come modello di teoria dell'informazione, la IB ha trovato applicazione nel deep learning e negli algoritmi di compressione, se si considerano X e Y come gli strati di input e output rispettivamente e Z come un qualsiasi strato interno. Nel caso studiato, l'applicazione dell'Information Bottleneck considera Z come la rappresentazione di X, e considera i due membri della lagrangiana come sufficienza ($I(x; y|z)$) e minimalità ($I(z; x)$) da minimizzare nella lagrangiana $\mathcal{L}$. Questa lagrangiana può essere ulteriormente ridotta sfruttando le identità mostrate in Figura 1 e il fatto che $H(y|x)$ sia un valore costante, in

$$\mathcal{L} = H(y|z) + \beta I(z; x) \tag{15}$$

con β che serve a regolare il trade-off tra le condizioni di sufficienza e minimalità. Tishby propone anche un algoritmo per minimizzare questa funzione nel caso in cui tutte le variabili aleatorie siano discrete e z sia deterministica in funzione di x, ma non ce n'è ancora nessuno che risolva il problema nel continuo. Lo studio proporrà quindi un nuovo metodo di minimizzazione di questa funzione che viene nominato *Information Dropout*, in quanto risulterà essere una generalizzazione del concetto di layer di Dropout.
Innanzitutto indichiamo il training set $\{(x_i, y_i)\}_{i=1,\dots,N}$ da cui vengono presi i campioni di dati, la cui distribuzione considerata vera sarà $p(x, y)$. La distribuzione a posteriori $p_\theta(z|x)$ è già parte del problema di ottimizzazione, mentre $p_\theta(y|z)$ è un altro posteriore sconosciuto e sarà la seconda variabile del modello (entrambe parametrizzate su θ, che rappresenta i pesi e i bias della generica rete), dunque serve qualche trasformazione per metterle in evidenza nelle formule:

$$H(y|z) := \mathbb{E}_z[\mathbb{E}_{y\sim p(y|z)}[-\log p_\theta(y|z)]] = -\sum_{y,z} p(y, z) \log p_\theta(y|z)$$

Poichè z è rappresentazione di x vale $\sum_{y,z} p(y, z) \simeq \sum_{x,y,z} p(x, y, z)$ che può essere scomposto con la chain rule della probabilità condizionata in $\sum_{x,y,z} p(z|y, x)p(x, y)$. Il termine $p(z|y, x)$ tuttavia equivale a $p(z|x)$ secondo la definizione di catena di Markov $y \longrightarrow x \longrightarrow z$ che modella la rappresentazione, dunque abbiamo che

$$H(y|z) \simeq -\sum_z p_\theta(z|x) \sum_x \sum_y p(x, y) \log p_\theta(y|z) \simeq \mathbb{E}_{x,y}[\mathbb{E}_{z\sim p_\theta(z|x)}[-\log p_\theta(y|z)]$$

dove il simbolo $\mathbb{E}_{z\sim p_\theta(z|x)}$ indica che le z_i nella sommatoria del valore atteso sono prese dal posteriore $p_\theta(z|x)$ e non dalla distribuzione marginale $p(z)$ come

sarebbe lecito attendersi se fosse scritto solo $\mathbb{E}_z$.
La proprietà di minimalità viene anch'essa trasformata per mantenere il riferimento alla distribuzione $p_\theta(z|x)$.

$$I(z;x) = D_{KL}(p(z,x)||p_z p_x) = \mathbb{E}_x[D_{KL}(p_\theta(z|x)||p_\theta(z))] \text{ dato che}$$
$$\mathbb{E}_x[D_{KL}(p(z|x)||p(z))] = \mathbb{E}_x[\mathbb{E}_{z\sim p(z|x)}[\log \frac{p(z|x)}{p(z)}]] =$$
$$\mathbb{E}_{z,x}[\log \frac{\frac{p(z,x)}{p(x)}}{p(z)}] = \mathbb{E}_{z,x}[\log \frac{p(z,x)}{p(x)p(z)}] = D_{KL}(p(z,x)||p_z p_x)$$

La lagrangiana $\mathcal{L}$ si approssima quindi a

$$\mathcal{L}(\theta) \simeq \mathbb{E}_{x,y}[\mathbb{E}_{z\sim p_\theta(z|x)}[-\log p_\theta(y|z)] + \beta\, \mathbb{E}_x[D_{KL}(p_\theta(z|x)||p_\theta(z))] \tag{16}$$

Poichè però il numero di campioni N del training set è finito, è valida l'approssimazione $\mathbb{E}[f] \approx \frac{1}{N}\sum_n f(x_n)$ che segue la legge dei grandi numeri. La trasformazione finale della lagrangiana risulta quindi

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{z\sim p_\theta(z|x_i)}[-\log p_\theta(y_i|z)] + \beta D_{KL}(p_\theta(z|x_i)||p_\theta(z)) \tag{17}$$

Poichè le due distribuzioni da stimare sono $p(z|x)$ e $p(y|z)$, il primo termine corrisponde alla cross-entropy tra le due, funzione solitamente utilizzata per stimare l'errore del risultato di una macchina relativamente a quello desiderato. Osservando il problema da questo punto di vista, il secondo termine può essere allora interpretato come un regolatore. Achille e Soatto utilizzeranno quindi la D_{KL} per penalizzare il passaggio di informazione da x verso z e metteranno a punto un sistema basato sull'aggiunta di rumore nei dati per tenere sotto controllo questa perdita di informazione.
Alla lagrangiana che bilancia sufficienza e minimalità si vuole ora aggiungere la qualità di disentanglement, che abbiamo detto si misura come $\mathrm{TC}(z) = D_{KL}(p(z)||\prod_i p(z_i))$, quindi $\mathcal{L} + \gamma\mathrm{TC}(z)$ è la nuova funzione bilanciata da γ, che regola la penalità imposta da un disentanglement diverso da zero. Tuttavia, la distribuzione marginale di z non è nota a priori ed è difficilmente calcolabile, ma è necessaria sia nel computo della divergenza KL che nella correlazione totale. Per questo motivo gli autori propongono una soluzione equivalente non più basata su $p_\theta(z)$, possibile se si sceglie un valore per $\gamma = \beta$. Il problema diventa quindi una minimizzazione di

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{z\sim p_\theta(z|x_i)}[-\log p_\theta(y_i|z)] + \beta D_{KL}(p_\theta(z|x_i)||q(z)) \tag{18}$$

dove la distribuzione $q(z)$ equivale a $\prod_{i=1}^{z} q_i(z_i)$. In pratica imponendo il disentanglement, e dunque l'indipendenza delle componenti di z, si riesce a ridurre il problema a uno effettivamente minimizzabile.

7 Information Dropout

Finora abbiamo parlato di rappresentazioni dell'input e della loro distribuzione, ma nella pratica non è stato ancora definito formalmente cosa sia una rappresentazione z. Una rappresentazione è quindi il risultato di una qualche composizione di funzioni $f(x) = g(x) \circ h(x) \circ i(x) \cdots$ che rappresenta l'elaborazione dei dati negli strati di una rete neurale, alterata da un fattore ϵ preso casualmente da una distribuzione parametrica $p_{\alpha(x)}(\epsilon)$ di rumore basato sull'input x.

$$z = \epsilon \odot f(x) \tag{19}$$

dove il simbolo $\odot$ rappresenta il prodotto di Hadamard, ovvero la moltiplicazione elemento per elemento fra le celle di due matrici. Come viene fatto notare, se $p_{\alpha(x)}(\epsilon)$ è una distribuzione di Bernoulli, l'implementazione equivale al classico dropout layer con maschera binaria. La scelta per la distribuzione del rumore ricade però su

$$p_{\alpha(x)}(\epsilon) = \log \mathcal{N}(0, \alpha_\theta^2(x)) \tag{20}$$

ovvero una distribuzione lognormale. Una distribuzione lognormale è tale quando il logaritmo della v.a. in questione segue una distribuzione gaussiana. Non è spiegato il motivo della scelta, ma è intuibile: Statisticamente, infatti, la distribuzione lognormale è ottenibile dalla produttoria di variabili aleatorie positive tutte indipendenti fra loro, come conseguenza del Teorema Limite Centrale nel dominio dei logaritmi ($\sum \log = \log \prod$). Considerando le attivazioni dei neuroni come tutte indipendenti fra loro è logico pensare a una distribuzione lognormale. Definita questa distribuzione, è possibile ottenere $p_\theta(z|x)$, ma resta comunque il problema di $q_\theta(z)$ nella divergenza KL. La scelta per $q_\theta(z)$ dipende fondamentalmente dalle funzioni di attivazione prese in considerazione, quindi Achille e Soatto analizzano i casi in cui siano utilizzate ReLU e Softplus.
Una rete di ReLU possiede naturalmente una proprietà di invarianza di scala, ossia rimane uguale anche scalando i pesi e le attivazioni, per il semplice fatto che $\max(0, kx) = k \max(0, x)$. Per questo motivo si cerca una distribuzione a priori $q(z)$ che rispetti questa proprietà, e l'unica disponibile è la distribuzione log-uniforme, come dimostrato da Havil [5], dunque una distribuzione $q(\log(z)) = c$ o alternativamente $q(z) = c/z$. Poichè le ReLU sono spesso clampate a zero, non si vuole dover aggirare il denominatore $z = 0$ ogni volta, dunque si separa la funzione nei due casi $c/z + q_0\delta_0$, dove q_0 è il valore che assume la distribuzione in zero, mentre δ_0 è la funzione di Dirac sempre in zero. La δ_x di Dirac è una distribuzione che si annulla in tutti i punti fuorchè lo zero, dove ha un picco e tende all'infinito, e per cui $\int_{-\infty}^{+\infty} = 0$. Grazie a questa aggiunta, $q(z)$ è in grado di modellare efficacemente le attivazioni delle ReLU. Per quanto riguarda una rete attivata a Softplus, è spiegato abbastanza chiaramente il motivo della scelta di un priore lognormale. L'unico termine da spiegare è il metodo di normalizzazione batch, che consiste nello scalare tutti gli output dei layer nascosti di una rete in modo che abbiano $\mu = 0$ e $\sigma^2 = 1$, al fine di accelerare l'addestramento e semplificare le attivazioni.
Ritorniamo quindi alla lagrangiana, la cui divergenza KL è ora risolvibile, avendo

stabilito dei possibili candidati per il priore $p(z)$. Nell'appendice A del paper sono riportate le dimostrazioni che conducono ai risultati, ma per scorrevolezza sono riportati. Data la distribuzione di rumore come lognormale su parametro α_θ,

$$\text{Information Dropout ReLU} \longrightarrow \; D_{KL}(p_\theta(z|x)||p(z)) = -\log \alpha_\theta(x) + const \tag{21}$$

$$\text{ID Softplus} \longrightarrow \; D_{KL}(p_\theta(z|x)||p(z)) = \frac{1}{2\sigma^2}(\alpha^2(x) + \mu^2) - \log \frac{\alpha(x)}{\sigma} - \frac{1}{2} \tag{22}$$

ed entrambe possono essere sostituite nella lagrangiana per ricavare il costo di dropout in una rete con quelle determinate attivazioni. È quindi possibile ottimizzare la funzione di costo tramite la discesa stocastica del gradiente, utilizzando quello che Kingma chiama local reparametrization trick [2]. Non è problematico il calcolo del gradiente $\nabla_\theta \mathbb{E}_{p(z)}[f_\theta(z)]$, in quanto corrisponde al valore atteso del gradiente di f; se invece la distribuzione che verrà campionata è anch'essa parametrizzata $p_\theta(z)$, ci si trova con un valore atteso indifferenziabile. Il trick consiste in un cambio di parametro $\epsilon \sim p(\epsilon)$ tale che z sia in funzione di x e di questo nuovo parametro, ottenendo un valore atteso $\mathbb{E}_{p(\epsilon)}$ indipendente da θ e quindi computabile nella discesa stocastica del gradiente.

8 Correlazioni con altri Framework

Le similarità fra l'Information Dropout e la funzione di costo di un VAE è molto forte fin da subito. Il concetto di Information Bottleneck è applicabile allo schema del VAE nel momento in cui l'input viene compresso nello spazio latente. Inoltre, entrambe hanno un termine di cross-entropy fra due distribuzioni a posteriori e un termine regolatore dato da una D_{KL}. Consideriamo quindi che il task della macchina generica sia quello di ricostruire l'input, come farebbe un VAE, dunque poniamo $y = x$; poniamo inoltre il caso $\beta = \gamma$ che ci consente di semplificare la lagrangiana con disentanglement, ottenendo (i valori sostituiti sono in grassetto)

$$\text{VAE Loss su un datapoint } i \;\; l_i(\theta, \phi) = \mathbb{E}_{z\sim q_\theta(z|x_i)}[-\log p_\phi(x_i|z)] + D_{KL}(q_\theta(z|x_i)||p(z))$$

$$\text{VAE Loss generale } \; l(\theta, \phi) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{z\sim q_\theta(z|x_i)}[-\log p_\phi(x_i|z)] + D_{KL}(q_\theta(z|x_i)||p(z))$$

$$\text{IBL con Disentanglement } \; \mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{z\sim p_\theta(z|x_i)}[-\log \boldsymbol{p_\theta(x_i|z)}] + \beta D_{KL}(p_\theta(z|x_i)||q(z))$$

Le due funzioni di costo sono quindi equivalenti nel momento in cui $\beta = 1$ e i parametri $\theta = \phi$. La seconda condizione è chiara se pensiamo che un VAE è formato da due reti neurali (e quindi due set di pesi e bias), mentre l'Information

Dropout è una funzione genericamente parametrizzata per una singola rete neurale. Tenendo presente che $q(z) = \prod_{i=1}^{z} q_\theta(z_i)$, si ottiene che un VAE che sceglie un priore $p(z)$ fattorizzato in questo modo sarà in grado di estrarre delle features z che rispettano le qualità desiderate di sufficienza e minimalità. Oltre a semplificare il problema di ottimizzazione, il priore fattorizzato permette di avere una rappresentazione disentangled e di conseguenza una compressione dei dati meno lossy. Scegliendo valori di β più grandi si forza l'Information Dropout, ottenendo rappresentazioni sempre più districate, ricalcando la teoria alla base del modello β-VAE proposto da Higgins [6].
Per $\gamma = 0, \beta > 0$ si rientra nell'Information Bottleneck standard, che può essere applicata a un VAE per concentrarsi solo sulla compressione dei dati senza dedicare risorse al disentanglement delle variabili latenti. Nel caso invece di $\gamma > 0, \beta = 0$, si ottiene una funzione del tipo

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{z \sim p_\theta(z|x_i)}[-\log p_\theta(y_i|z)] + \gamma D_{KL}(p(z)||\prod_i p(z_i))$$

che secondo gli autori dovrebbe richiamare la "funzione di perdita standard dell'ICA". L'Analisi delle Componenti Indipendenti (ICA) è una tecnica che separa un segnale nei sotto-segnali che lo compongono additivamente, assumendo che questi siano indipendenti fra loro e non siano Gaussiani. Oltre all'analisi dei segnali è utilizzata anche in economia, per il riconoscimento di immagini e nei modelli generativi per la riduzione della dimensionalità. Tuttavia, non c'è una funzione obiettivo standard, ma semmai ci sono diversi metodi per stimarne il modello, come la massima verosimiglianza, la minima informazione mutua e alcune tecniche per massimizzare la non-Gaussianità delle componenti indipendenti. Non ho trovato riferimenti evidenti alla funzione di perdita di cui sopra, nemmeno nei testi citati dagli stessi autori.

9 Risultati Sperimentali

Le teorie sull'Information Dropout sono messe in pratica testandole su modelli già consolidati di reti e di dataset. La varianza del rumore viene vincolata ad $\alpha < 0.7$, perchè logicamente addestrare una rete con dati eccessivamente diversi tra loro non porta buoni risultati. Si sceglie inoltre di dividere la D_{KL} per il numero di campioni di training in modo che scali similmente a quella dell'esperimento di Kingma. Vediamo i risultati degli esperimenti, seguendo il paper.

9.0.1 Cluttered MNIST

Viene utilizzata una rete neurale di soli layer convoluzionali di ReLU e una versione "sporcata" del MNIST (un dataset pubblico di cifre disegnate a mano per addestrare le reti neurali), in cui le immagini in input presentano 21 "macchie" di rumore per distrarre la macchina. Per valori molto bassi di β, l'informazione

che passa da un layer al successivo comprende anche il rumore, mentre alzando il regolatore si forza la CNN a ignorare gli elementi meno importanti, quali i distrattori. Questo accade perchè gli strati di Information Dropout sono in grado di apprendere com'è strutturato il rumore nel dataset.

9.0.2 Occluded CIFAR

Simile all'esperimento precedente, ma cambiano la macchina e il dataset. La rete convoluzionale è meno complessa e processa immagini più piccole (32x32 dove prima erano 96x96), mentre i dati provengono dal CIFAR, un altro dataset pubblico che contiene immagini di animali e veicoli. Ancora una volta, le immagini sono sporcate inserendo le cifre del MNIST sopra di esse, che non devono essere classificate. Si addestra inoltre una seconda rete a classificare le cifre usate come rumore usando solo la rappresentazione ricavata dalla rete principale, per testare quanto questa sia effettivamente affetta dal rumore nel training set. I risultati confermano che per valori di β bassi, la rete secondaria è più abile nella classificazione dei numeri usati come rumore, segno che questo viene passato nella rappresentazione. Valori alti per β corrispondono a una maggiore insensibilità al rumore e mostrano un miglioramento nella performance del task principale (non eccezionale, ma comunque un miglioramento), laddove cala drasticamente per la rete basata sul rumore.

9.0.3 MNIST and CIFAR-10

In questo esperimento vengono messe a confronto due reti neurali differenti addestrate su dataset differenti, con l'obiettivo di studiare l'Information Dropout in ambienti diversi. La rete più piccola col MNIST e le ReLU viene testata con ben quattro tipi di dropout diversi: information Dropout, dropout binario (la maschera di dropout è di bit), dropout continuo (la maschera non spegne più i neuroni ma li divide in neuroni "molto attivi e poco attivi" [7]) e un dropout in cui il rumore è preso da una distribuzione lognormale costante. I risultati della rete più piccola sono riportati nel primo grafico della Figura 4, e mostrano che l'Information Dropout riduce la percentuale di errore, seppur si parli di valori già molto bassi in tutti e quattro i casi. Il secondo grafico spiega i risultati della CNN Softplus addestrata col CIFAR, ma per quanto vengano mostrati dei miglioramenti, sono poco convincenti: Come mai il paragone è solo con il dropout binario, invece che con tutti e tre i concorrenti? A cosa serve mostrare il rapporto tra dropout e nessun dropout? E perchè l'errore è calcolato sul validation set invece che sul test set? Gli ultimi due grafici in Figura 4 si riferiscono alla CNN con due strati di dropout e mostrano come varia la quantità di informazione trasmessa in base al rumore applicato.

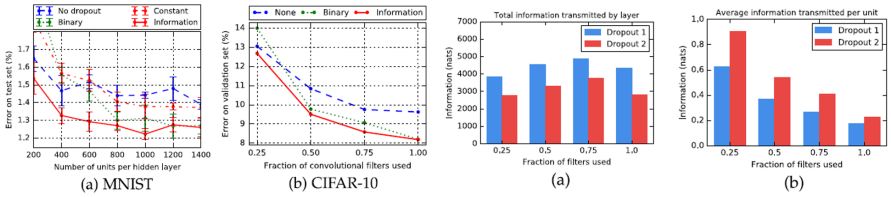

Figure 4: Risultati del confronto fra diversi dropout e diverse architetture

9.0.4 Disentangling

In caso di una distribuzione normale multivariata, è possibile utilizzare questa formula per la correlazione totale

$$\mathrm{TC}(X) = \frac{1}{2}\sum_i \log \sigma_{ii}^2 - \frac{1}{2}\log|\Sigma| \tag{23}$$

dove $\Sigma = \mathbb{E}[(x - \mathbb{E}[x])(x - \mathbb{E}[x])^T]$ è la matrice di covarianza associata alla distribuzione. Gli autori la approssimano a $-\log|\mathrm{diag}\Sigma^T\Sigma|$ e calcolano l'errore in relazione al disentanglement delle componenti al variare di β. Al crescere di β, le componenti sono più districate e l'errore medio cala, ma senza esagerare. Sebbene la curva della correlazione totale sia decrescente, l'errore torna a crescere di nuovo con $\beta > 1$, poichè la rete crea delle rappresentazioni sempre più sbilanciate verso la minimalità ignorandone la condizione di sufficienza.

9.1 VAE

Il VAE dell'esperimento di Kingma viene replicato e addestrato usando sia rappresentazioni gaussiane (come nel caso originale) che Information Dropout. La performance di entrambi i modelli va di pari passo, e ci consente di considerare il VAE come un caso particolare di Information Dropout.

References

[1] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[2] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28:2575–2583, 2015.

[3] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[4] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[5] Julian Havil. Gamma: exploring euler's constant. *The Australian Mathematical Society*, page 250, 2003.

[6] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[7] Xu Shen, Xinmei Tian, Tongliang Liu, Fang Xu, and Dacheng Tao. Continuous dropout. *IEEE transactions on neural networks and learning systems*, 29(9):3926–3937, 2017.