



UNIVERSITÀ DEGLI STUDI DI UDINE

Dipartimento Politecnico di Ingegneria e Architettura
Corso di Laurea magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

**VALUTAZIONE DELLA COMPRESSIONE
DELL'INFORMAZIONE NELLE RETI NEURALI
FEED-FORWARD**

Relatore:
Prof. Mirko Loghi

Laureando:
Stefano Pota

Correlatore:
Ing. Riccardo Fontanini

Anno Accademico 2018/2019

*È impossibile vivere senza fallire in qualcosa,
a meno che non viviate in modo così prudente
da non vivere del tutto – in quel caso,
avrete fallito in partenza.*

J.K. Rowling.

Ringraziamenti

Un ringraziamento particolare va alla mia famiglia per avermi consentito di raggiungere questo traguardo, senza la quale non sarebbe stato possibile raggiungere, e per avermi supportato in questi anni di duro lavoro.

Ringrazio inoltre i miei amici e tutte le persone che mi sono state vicino in questo periodo.

Udine, Marzo 2020

Indice

Introduzione	1
1 Reti neurali	7
1.1 Neurone	7
1.2 Altre funzioni di attivazione	8
1.3 Reti neurali artificiali	10
1.4 L'algoritmo di Backpropagation	11
1.4.1 Le equazioni fondamentali	11
1.4.2 L'algoritmo	12
2 Teoria dell'Informazione	13
2.1 Mutua Informazione	13
2.1.1 Distribuzione di probabilità discreta, probabilità congiunta, probabilità condizionale	14
2.1.2 Entropia	15
2.1.3 Entropia congiunta, entropia condizionale ed entropia relativa	15
2.2 Il metodo dell'Information Bottleneck	20
2.2.1 Soluzione dell'IB framework	21
2.2.2 L'algoritmo di Blahut-Arimoto	22
2.2.3 IB Curve	24
2.2.4 IB framework utilizzato nelle DNN	25
2.2.5 Information Plane	26
2.3 Stima dell'Informazione Mutua	27
2.3.1 Problema ill-posed	27
2.4 Binning	28
2.4.1 Binning statico	29
3 Prove Sperimentali	31
3.1 Dataset	31
3.1.1 Dummy Dataset	31
3.1.2 IB Dataset	32
3.1.3 Dataset con una Struttura	33
3.1.4 Dataset MNIST	34
3.1.5 Dataset CIFAR10	35
3.2 Simulazioni	36
3.3 Simulazione con IB Dataset	37
3.3.1 Gradiente della funzione costo	51

3.3.2	Benefici hidden layers	55
3.3.3	Variazione dei parametri	58
3.4	Simulazione con Dummy Dataset	67
3.5	Simulazione con Struttura Dataset	68
3.6	Simulazione con MNIST Dataset	69
3.7	Simulazione con CIFAR-10 Dataset	70
	Applicazioni e lavoro futuro	71
	Conclusioni	73
A	Appendice terzo capitolo	77
A.1	Mutua Informazione	77
A.1.1	Interpolazione dei dati	78
A.1.2	Calcolo online dell'informazione mutua	79
A.1.3	Rimescolamento dataset	80
A.2	Requirements	80
A.3	Compressione e generalizzazione	81
A.4	Andamento entropia	83
A.5	Gradiente funzione costo calcolato rispetto il bias	84
A.6	Variazione numero Hidden layer	85
A.7	Variazione grandezza dataset	91
	Acronimi	93
	Bibliografia	95

Elenco delle figure

1.1	Percettrone	7
1.2	Rete di Percetroni	8
1.3	Andamento funzioni di attivazione tanh, sigmoide, ReLU e LeakyReLU	9
1.4	Esempio di rete neurale	10
2.1	Diagramma di Venn che mostra la relazione che c'è tra l'informazione mutua tra due variabili e la loro entropia.	17
2.2	I layer di una Deep Neural Network (DNN) formano alla fine della fase di addestramento una Catena di Markov, che viene attraversata dall'ingresso X , fornendo in uscita una predizione dell'etichetta \hat{Y} . Ogni rappresentazione T dell'informazione compressa dell'ingresso X , può essere quantificata mediante la coppia di punti $I(X, T)$ e $I(T, Y)$ nel piano dell'informazione. Allo stesso modo al punto può essere associato una distribuzione di probabilità per la codifica $P(T X)$ ed una di decodifica $P(\hat{Y}, T)$. Si è scelto di inserire l'etichetta vera Y a sinistra della rete, poiché questo è in generale diverso dall'etichetta predetta \hat{Y}	18
2.3	Quando i dati entrano dentro la rete, ogni neurone che va in 'fire' (uscita alta), trasmette un impulso ai neuroni a cui è collegato, facendoli così risultare più propensi a loro volta ad attivarsi. Il processo filtra il rumore ed estrae solo le caratteristiche principali.	20
2.4	Procedura di compressione dell'insieme X , nella quale i vari sottosets vengono mappati in sottosets di dimensione minore, nell'insieme T	23
2.5	IB Curve al variare del coefficiente di Lagrange β	24
2.6	Fino al punto C l'uscita del layer, la quale viene interpretata come essere una unica variabile, acquisisce informazioni sull'ingresso e sull'uscita migliorando la predizione. Tra la fase C ed E la rete entra nella fase di compressione andando a scartare le informazioni irrilevanti dell'ingresso, raggiungendo così l'ottimo tra accuratezza e compressione.	26
2.7	Esempio di grafici	29
3.1	Rappresentazione di un sample dei primi 15 elementi del dataset MNIST con le rispettive etichette.	34
3.2	Rappresentazione di un sample dei primi 15 elementi del dataset CIFAR10 con le rispettive etichette.	35

3.3	Esempio di architettura utilizzata, rete <i>Fully Connected</i> 12-10-7-5-4-3-2.	36
3.4	Traiettorie della mutua informazione per ogni layer T_i . Le traiettorie dei layer più vicini all'ingresso si trovano sul lato destro della figura, mentre le traiettorie dei layer più profondi si trovano sulla sinistra. È stato evidenziato mediante una croce rossa, per gli ultimi hidden layer, il punto in cui si passa dalla fase di apprendimento a quella di compressione. Learning rate= 0.001, batch size = 512, funzione di attivazione=tanh, n.bins=30.	39
3.5	Grafici relativi alla variazione in percentuale del train dataset. Learning rate=0.001, batch size=1, funzione di attivazione=tanh, n.bins=30.	42
3.6	Grafici della mutua informazione utilizzando funzioni di attivazioni diverse dalla tanh. Learning rate=0.001, batch size=64, n.bins=30.	43
3.7	Esempio di come il valore dell'uscita del neurone (l'activity), viene divisa in bin equamente spaziati. Le zone evidenziate sono i valori quantizzati nei quali l'activity tende a convergere durante l'addestramento. Come si può vedere la funzione ReLU e la LeakyReLU non limitano automaticamente l'uscita del neurone.	44
3.8	Andamento medio activity digitalizzata degli hidden layer durante l'addestramento. La funzione <i>tanh</i> fà convergere le uscite verso ± 1 , la funzione sigmoide verso 0, 1. Learning rate=0.1, batch size=512, n.bins=30.	46
3.9	Andamento medio activity digitalizzata degli hidden layer durante l'addestramento, utilizzando la funzione <i>ReLU</i> . Learning rate=0.1, batch size=512, n.bins=30.	47
3.10	Andamento medio activity digitalizzata degli hidden layer durante l'addestramento, utilizzando la funzione <i>LeakyReLU</i> . Learning rate=0.1, batch size=512, n.bins=30.	48
3.11	Andamento funzione costo con diverse funzioni di attivazione per gli hidden layer. Learning rate=0.001, batch size=1.	49
3.12	Andamento media, std ed SNR del gradiente durante l'addestramento. Learning rate=0.001, batch size=512, n.bins=30.	53
3.13	Andamento informazione al variare del numero di layer. Learning rate=0.001, batch size=512, n.bins=30.	57
3.14	Andamento informazione con dataset di train al variare del batch size. Learning rate=0.001, n.bins=30.	60
3.15	Andamento informazione con dataset di train al variare del learning rate. Batch size=512, n.bins=30.	63
3.16	Analisi sperimentali al variare dei parametri alla ricerca di eventuali linearità.	64
3.17	Andamento Mutua Informazione sul dataset di test al variare del learning rate e batch size, n.bins=30.	66
3.18	Andamento Mutua Informazione e gradiente al variare della funzione di attivazione per il Dummy Dataset. Learning rate=0.005, batch size=512, n.bins=30.	67
3.19	Andamento Mutua Informazione al variare della funzione di attivazione per il Struttura Dataset. Learning rate=0.005, batch size=512, n.bins=30.	68

3.20 Andamento Mutua Informazione al variare della funzione di attivazione per il MNIST Dataset. Learning rate=0.001, batch size=512, n.bins=30, n.samples=4000. Configurazione Hidden Layers: 32-32-32-16-16.	69
3.21 Andamento Informazione al variare della funzione di attivazione per il CIFAR-10 Dataset. Learning rate=0.001, batch size=256, n.bins=30, n.samples=100. Configurazione Hidden Layers: 32-32-32-16-16.	70
A.1 Addestramento online con funzione di attivazione tanh. n.train samples=3096, learning=0.001, batch size=512.	79
A.2 Informazione mutua nel caso in cui si facesse il rimescolamento del dataset. n. campioni per l'addestramento=3096, learning=0.001, batch size=512.	80
A.3 I punti evidenziati in rosso e celeste sono quelli in cui si raggiunge il massimo valore dell'accuratezza durante l'addestramento, sia per il train dataset, che per il test dataset. Questo allo scopo di poter mostrare come accuratezza e generalizzazione sono collegate, ma non indicano la stessa cosa. N.train samples=5000, n.test samples=1000, dataset MNIST, n.bins=30.	82
A.4 Rappresentazione durante l'addestramento di come variano i diversi contributi dell'entropia all'interno della mutua informazione. Learning rate = 0.001, batch size=512, n.bins=30.	83
A.5 Andamento media, std ed SNR del gradiente della funzione costo rispetto b. Learning rate=0.001, Batch size=512.	84
A.6 Andamento funzione costo al variare dei layer. Learning rate=0.001, batch size=512, n.bins=30.	86
A.7 Andamento media e varianza del gradiente calcolato rispetto W al variare degli Hidden layer. Learning rate=0.001, batch size=512, n.bins=30.	88
A.8 Andamento SNR gradiente rispetto W al variare degli Hidden layers. Learning rate=0.001, batch size=512, n.bins=30.	90
A.9 Traiettoria dell'informazione del test dataset al variare del train dataset. Learning rate=0.01, batch size=128, funzione di attivazione=tanh, n.bins=30. In questo caso di è utilizzato il dataset originale fornito da Tishby, allo scopo di confrontare i risultati.	91

Elenco delle tabelle

3.1	Caratteristiche Dummy Dataset.	32
3.2	Caratteristiche IB Dataset.	32
3.3	Caratteristiche Struttura Dataset.	33
3.4	Caratteristiche MNIST Dataset.	34
3.5	Caratteristiche CIFAR Dataset.	35
3.6	Numero di neuroni per layer della prima architettura realizzata. . .	36

Elenco dei codici

A.1 File requirements.txt contenente la lista dei pacchetti utilizzati e le rispettive versioni.	80
--	----

Sommario

Le Deep Neural Network ([DNN](#)) vengono utilizzate per svolgere compiti complessi in diversi campi, come per esempio quello del riconoscimento delle immagini. Allo stato attuale non è ben chiaro però, come queste siano in grado di raggiungere certe performance di accuratezza, e vengono per questo considerate tuttora delle “Black Box”, per quanto riguarda il flusso di informazioni attraverso la rete.

In recenti studi Tishby e Schwartz-Ziv hanno proposto un metodo innovativo per analizzare le reti neurali, attraverso l'utilizzo della Teoria dell'Informazione. In particolare, essi affermano che le reti neurali raggiungono una buona generalizzazione, poiché sono in grado di comprimere le informazioni in ingresso, scartando i dati irrilevanti [\[45\]](#). Seguendo questa analisi, è possibile quindi visualizzare, l'ammontare dell'informazione che ogni layer possiede rispetto l'ingresso e l'uscita. Sono state riscontrate due fasi distinte durante l'addestramento di una rete, quella di apprendimento e quella di compressione. Nella prima, la rete si adatta al dataset in ingresso, facendo diminuire la funzione costo molto velocemente. Nella seconda, c'è una compressione molto lenta dell'informazione. Viene ulteriormente affermato che la [DNN](#), se allenata tramite l'algoritmo di Stochastic Gradient Descent ([SGD](#)), tende automaticamente a convergere in prossimità della Information Bottleneck Curve ([IB Curve](#)). Tale curva è la soluzione dell'Information Bottleneck ([IB](#)), e rappresenta i punti di massima compressione dell'ingresso, mantenendo allo stesso tempo sufficienti informazioni sull'uscita. In questa tesi si andrà principalmente a verificare e replicare il lavoro svolto da Tishby, svolgendo un'analisi sulla validità delle sue affermazioni, e sulle problematiche derivanti da una stima corretta dell'informazione. Successivamente, si valuterà se tale teoria ha validità generale anche per altri dataset, quali per esempio il *MNIST* [\[28\]](#) ed il *CIFAR10* [\[26\]](#), indipendentemente dalle caratteristiche della rete e dai suoi parametri.

Parole chiave: Information Bottleneck, Reti Neurali, Deep Neural Network, Teoria dell'informazione.

Introduzione

Le **DNN** negli ultimi anni hanno raggiunto dei livelli di prestazioni notevoli, tanto da poter svolgere diversi tipi di compiti più complessi: battere il campione mondiale di *Go* [49], essere capaci di riconoscere diversi tipi di immagini [14, 21, 48], fino ad arrivare ad essere utilizzati nella guida autonoma. Sebbene vengano utilizzate da diversi anni, esse vengono ancora considerate delle “*Black Box*”, e non c’è ancora una spiegazione che possa dire con certezza, il perché riescano a raggiungere certe performance in termini di generalizzazione [2]. Inoltre non sono disponibili degli algoritmi per determinare l’architettura ed i parametri ottimi da usare, continuando a rimanere un approccio empirico “*Try and Error*”.

Una proprietà interessante delle **DNN** come dimostrato da Bengio et al., è quella di essere in grado di memorizzare una grande quantità di dati, anche nel caso in cui non ci fosse nessun *pattern* presente all’interno del dataset in ingresso [7]. Questi risultati ci fanno capire quanto ci sia ancora da imparare sulla dinamica di questi modelli matematici, che riescono a risolvere problemi tipicamente affrontati dall’essere umano.

Le **DNN** sono delle reti profonde composte da un elevato numero di **hidden layer**. Per motivi ancora non del tutto chiari, nonostante l’alto numero di parametri, queste reti tendono a generalizzare molto bene, violando così la regola convenzionale dell’*apprendimento statistico*. Bengio et al. hanno svolto una serie di esperimenti, per riuscire a capire quali sono i limiti che una rete ha nel memorizzare un dataset in ingresso [7]. Si è scoperto che anche nel caso in cui le etichette in uscita venissero sostituite con una sequenza casuale, la rete a regime tende a memorizzare completamente il dataset in ingresso, ottenendo così l’**overfitting**.

In una fase successiva dell’esperimento, oltre alle etichette, è stato sostituito anche il dataset in ingresso con una sequenza di pixel casuali. Si è visto come un’architettura di rete come **Inception** [43], che normalmente possiede un numero di parametri maggiore della **AlexNet** [48] o di un Multi Layer Perceptron (**MLP**) classico, riesca a raggiungere un’accuratezza ed una generalizzazione migliore rispetto a queste ultime. Tale comportamento è presente anche nel caso in cui si utilizzi un dataset di dimensione maggiore, come l’**ImageNet** [16]. Bengio et al. sono arrivati alla conclusione che le Deep Neural Networks:

Easily fit random labels.

Questo permette di addestrare una rete su un certo dataset ed utilizzarle successivamente su un altro che non hanno mai visto prima. Per vedere questo fenomeno non è necessario utilizzare delle regolarizzazioni durante l'addestramento, come il ***dropout*** o la ***normalizzazione L2***, che vanno a penalizzare la memorizzazione del dataset a discapito della generalizzazione. Evidenze sperimentali derivanti dal lavoro svolto da Arpit et al., hanno portato alla conclusione che durante l'addestramento sono presenti due fasi distinte: nella prima la rete cerca di trovare i pattern più semplici all'interno del dataset in ingresso, mentre nella seconda fase si occupa di memorizzare il dataset portandolo così all'overfitting. Viene inoltre affermato che con dei dataset reali si possono ottenere dei buoni risultati anche con delle reti di piccole dimensioni, e questo grazie al fatto che i pattern più semplici vengono trovati più facilmente in questo caso, rispetto a quando, sia il dataset in ingresso che le etichette, vengono generate in modo casuale [6].

In conclusione possiamo affermare grazie al lavoro di Arpit et al., che il processo di apprendimento di una rete si avvantaggia del pattern condiviso dai campioni in ingresso, andando a trovare prima quelli più semplici nel dataset, e solamente in un secondo momento si occupa della memorizzazione dei dettagli.

Per poter capire meglio cosa succede all'interno di una rete profonda durante il processo di apprendimento, Tishby e Schwartz-Ziv, suggeriscono di osservare il processo di addestramento della rete utilizzando le nozioni della Teoria dell'Informazione. Questo allo scopo di poter eventualmente ottimizzare il processo di apprendimento se si trovassero caratteristiche simili anche in altre architetture. La loro intera analisi si basa sull'utilizzo della mutua informazione che, come si vedrà in maggior dettaglio in seguito, è una grandezza statistica che misura la quantità di informazioni che una variabile casuale fornisce su un'altra, e viene misurata solitamente in bit. La mutua informazione si può pensare come alla riduzione di incertezza che ha una variabile quando viene a conoscenza del valore assunta da un'altra. In questo modo quando questa è grande, si ha una grande riduzione dell'incertezza, viceversa quando assume un valore piccolo se ne indica una diminuzione ridotta, questa assume valore nullo quando due variabili sono indipendenti. Nel nostro caso quello che si andrà a valutare è la mutua informazione tra ogni hidden layer rispetto l'ingresso **X** e l'etichetta target **Y** [37, 44–46].

Durante la fase di addestramento, ogni layer ottiene informazioni dal layer precedente, facendo attraversare all'informazione quella che viene chiamata Catena di Markov, questa trasforma in modo progressivo lo spazio dei dati in ingresso nello spazio delle etichette, mentre attraversa l'***Information Path***. Si parla di Catena di Markov perché ogni rappresentazione può essere calcolata solamente quando il layer precedente ha finito la sua computazione.

Tishby e Schwartz-Ziv hanno affermato che ci sono due fasi distinte durante l’addestramento della rete. Nella prima, la rete impara molto velocemente ad estrarre e a memorizzare informazioni sulle etichette target \mathbf{Y} , e sull’ingresso \mathbf{X} (*fase di apprendimento*), dove sia l’informazione sull’ingresso che sull’uscita aumenta. Nella seconda, l’informazione viene compressa lentamente, scartando le informazioni irrilevanti che la rete ha sull’ingresso (*fase di compressione*), raggiungendo un minimo teorico dettato dall’[IB Curve](#). Alla fase di compressione viene associata la capacità di generalizzazione della rete, e che questo comportamento deve essere osservato anche in altri costrutti, come per esempio nelle *Bayesian Networks* o *Random Forests* [45].

Un’ulteriore affermazione che viene fatta, è che durante la fase di apprendimento, la media del gradiente della funzione costo rispetto la matrice dei pesi \mathbf{W} , assume un valore molto più grande rispetto alla varianza della medesima grandezza, la quale rappresenta il rumore. In questa fase si ha un alto rapporto segnale rumore ([SNR](#)), e si suppone essere questo il motivo per cui la fase di memorizzazione è molto più veloce rispetto a quella di compressione, a cui è associato un [SNR](#) più piccolo. Questo permette ai pesi di cambiare come un *Processo di Weiner o di diffusione caotica*. Tale processo di diffusione è responsabile nel massimizzare l’entropia condizionale, la quale come si vedrà meglio in seguito, comporta una compressione dell’informazione rispetto l’ingresso. Viene infine affermato che il vantaggio portato dall’aggiunta degli hidden layer è un vantaggio di tipo computazionale, portando l’informazione a convergere in minor tempo [45].

Una rete, che può avere anche milioni di parametri, riesce a convergere in un *minimo locale* generalizzando molto bene anche su dati mai visti. Alcune ricerche recenti affermano che una componente fondamentale per ottenere una buona generalizzazione, è portata dal rumore introdotto dal processo di aggiornamento dei pesi, seguendo l’algoritmo del gradiente stocastico [11, 39, 51].

Scopo della Tesi

Questa tesi ha molteplici scopi:

- Replicare il lavoro svolto da Tishby e determinare se la Teoria dell'Informazione può essere un metodo corretto per comprendere la dinamica delle **DNN** [45].
- Determinare se la fase di compressione può essere causata da una particolare scelta della funzione di attivazione, come affermato da Saxe et al. [33], e se è possibile ottenere una compressione anche quando si utilizzano altre funzioni di attivazione con andamento non a saturazione, ed in quel caso come.
- Visualizzare l'andamento delle traiettorie dell'informazione al variare dei parametri e dell'architettura di rete, verificando inoltre che queste convergono verso la **IB Curve**.
- Vedere se lo stesso comportamento della traiettoria dell'informazione descritto da Tishby nel suo dataset d'esempio, ha carattere generale e può essere visualizzato anche in altri dataset, per esempio nel MNIST [28] e nel CIFAR [26], che non sono simmetrici come il dataset utilizzato da Tishby [45].
- In base a quanto affermato da Tishby, ogni layer della rete cerca di convergere verso l'ottimo definito dalla teoria dell'**IB**, facendo così soddisfare alle variabili di codifica e decodifica, le sue equazioni autoconsistenti, ottimizzando il tradeoff tra compressione e predizione per ogni layer. Verificare che tale affermazione valga per tutte le variazioni della rete, effettuate durante le sperimentazioni.
- Verificare se c'è un legame tra compressione e generalizzazione, e se le reti che non comprimono nel piano dell'informazione sono ancora in grado di generalizzare e viceversa.
- Verificare i benefici dell'aggiunta di ulteriori layer nella **DNN**, e che questi riducano il tempo di rilassamento del processo di diffusione del **SGD**. Si andrà ad analizzare se troppi hidden layer possano peggiorare le performance di generalizzazione di una rete, come affermato dalla letteratura classica.

Essendo il metodo di analisi delle **DNN**, attraverso l'**IB**, ancora in fase di discussione in diverse ricerche, si cercherà di dare il proprio contributo sui punti precedenti.

Struttura della Tesi

Il testo della tesi è così strutturato:

Nel primo capitolo si andrà a fare una breve introduzione sulle reti neurali, partendo dal Neurone per arrivare fino alle **DNN**. Si parlerà inoltre della funzione costo e di come questa può essere portata in un suo minimo locale, attraverso l'algoritmo di *Backpropagation*, utilizzando il metodo del gradiente stocastico per modificare i parametri della rete.

Nel secondo capitolo si andranno a porre le basi teoriche che stanno dietro la Teoria dell'Informazione, e quali sono le soluzioni proposte dalla letteratura per una sua corretta stima. Infine si andranno ad introdurre i risultati di maggior rilievo della teoria dell'Information Bottleneck (**IB**).

Il terzo capitolo è il lavoro principale di questa Tesi e serve per verificare se l'**IB** può essere uno strumento adeguato per analizzare la dinamica delle **DNN**. Tale capitolo andrà a replicare i risultati sperimentali del lavoro di Tishby, utilizzando il metodo del *binning statico* per stimare la mutua informazione. Si vedrà come cambiano le traiettorie dell'informazione al variare dei parametri di rete, e se si ottiene lo stesso andamento con dataset "reali" (per esempio con il MNIST ed il CIFAR10), verificando di conseguenza che le affermazioni di Tishby abbiano carattere generale [45]. Si determinerà inoltre la variazione della traiettoria dell'informazione al variare delle funzioni di attivazione.

Nelle conclusioni si farà un breve riassunto sulle attuali ricerche che utilizzano la teoria dell'**IB**. Si andranno a fare dei commenti finali in base ai risultati ottenuti nei capitoli precedenti, sui problemi derivanti da una corretta stima della mutua informazione, ed infine su come procedere per un eventuale lavoro futuro.

Capitolo 1

Reti neurali

Nel seguente capitolo e nel successivo, si introdurrà la teoria necessaria per comprendere al meglio questa tesi, rendendo così il lavoro autoconsistente. Verranno introdotte le reti neurali, partendo dal singolo neurone ed arrivando fino alle **DNN**. Verrà spiegato come funziona l'algoritmo di **Backpropagation**, utilizzato dalle reti **MLP** per poter ridurre la **funzione costo** e farla convergere così in un minimo locale.

Nel nostro particolare caso, verrà utilizzato un apprendimento di tipo *supervisionato* su un problema di *classificazione*, escludendo così quello di *regressione*.

1.1 Neurone

Il neurone, componente fondamentale di un hidden layer, può essere visto in Figura 1.1.

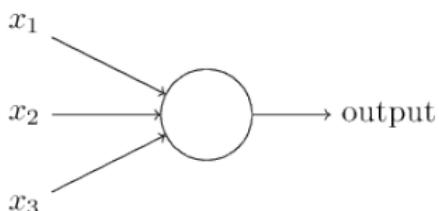


Figura 1.1: Esempio di neurone con tre ingressi.

Il neurone è composto da dalle sinapsi pesate tramite i coefficienti w_j , l'uscita è determinata in base al valore assunto dai suoi ingressi, come mostrato dall'equazione seguente. Si assume inizialmente che la funzione di attivazione f sia una funzione a gradino:

$$\begin{cases} \text{output} = f(\text{input}) = 0 & \text{se } \text{input} = \sum_j w_j x_j \leq \text{soglia} \\ \text{output} = f(\text{input}) = 1 & \text{se } \text{input} = \sum_j w_j x_j > \text{soglia} \end{cases} \quad (1.1)$$

Variando il numero di neuroni ed il tipo di connessioni, si possono ottenere dei modelli di decisione più complessi (Figura 1.2).

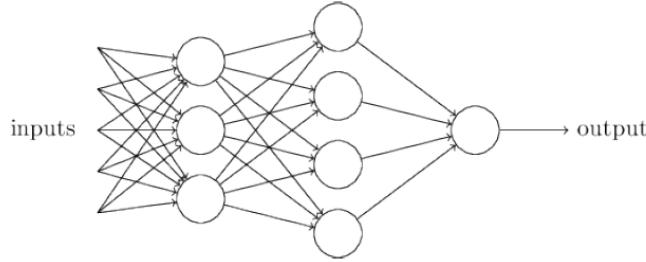


Figura 1.2: Esempio di rete di neuroni.

Utilizzando la notazione vettoriale ed introducendo l'ingresso di bias b , in modo da cambiare il riferimento della funzione di attivazione f , si può riscrivere l'Equazione (1.1) nel seguente modo:

$$\begin{cases} \text{output} = f(\text{input}) = 0 & \text{se } \text{input} = w \cdot x + b \leq 0 \\ \text{output} = f(\text{input}) = 1 & \text{se } \text{input} = w \cdot x + b > 0 \end{cases} \quad (1.2)$$

1.2 Altre funzioni di attivazione

La funzione di attivazione a gradino soffre di un problema importante, infatti un piccolo cambiamento in ingresso potrebbe causare un grande cambiamento in uscita della rete, rendendo quasi impossibile qualsiasi processo di apprendimento. Per questo motivo la funzione f a gradino, è stata sostituita da altri tipi di funzioni, potendo riscrivere l'equazione precedente in generale come:

$$Y_j = f\left(\sum_{i=1}^M X_i W_{ij} + b_j\right) \quad (1.3)$$

Dove la funzione f è la **funzione di attivazione**, X_i è l'uscita del i -esimo neurone precedente a j , W_{ij} il peso che lega l'uscita del neurone precedente i con il neurone che si sta considerano j , mentre b_j è il bias del neurone j .

Le performance di generalizzazione di una rete sono anche legate dalla scelta della funzione di attivazione, di seguito si possono visualizzare quelle che verranno utilizzate nel [capitolo 3](#):

$$f_1 = \text{Tanh}(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad (1.4)$$

$$f_2 = \text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (1.5)$$

$$f_3 = \text{ReLU}(x) = \max(0, x) \quad (1.6)$$

$$f_4 = \text{LeakyReLU}(x) = \max(\alpha x, x) \text{ con } \alpha \leq 1 \quad (1.7)$$

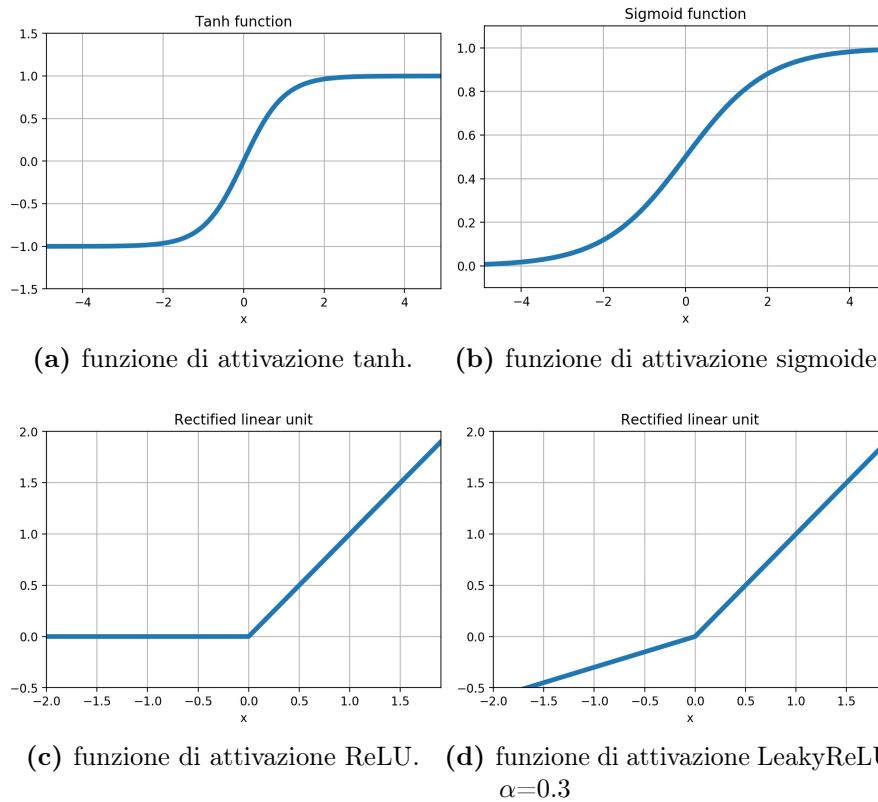


Figura 1.3: Andamento funzioni di attivazione tanh, sigmoide, ReLU e LeakyReLU.

Come si può notare, una prima differenza tra le varie funzioni di attivazione utilizzate, sono i limiti superiori ed inferiori.

Nel caso della funzione *Tanh* e *Sigmoide* queste sono limitate, mentre nel caso della funzione *ReLU* e *LeakyReLU* queste non lo sono, rendendo la stima della mutua informazione, come si vedrà meglio in seguito, più difficile. Funzioni come la *Tanh* che limitano i valori dell'uscita in un certo range, verranno definite da qui in avanti *funzioni a saturazione*.

1.3 Reti neurali artificiali

Le reti neurali artificiali sono organizzate in hidden layer, ed ognuno di questi sono composti da più neuroni. La rete si occupa di mappare gli stati in ingresso (*l'inputspace*), in uno stato d'uscita (*l'outspace*) [8, 13, 19].

Una regola generale indica che se il numero di hidden layer è superiore a tre allora la rete neurale diventa una **DNN**, cioè una Deep Neural Network. Oltre a questa categoria, c'è un'altra struttura interessante presente in letteratura, che si differenzia dalla **DNN** da un diverso *dataflow*, queste vengono chiamate Recurrent Neural Network (**RNN**) (spesso queste vengono utilizzate nel riconoscimento dei linguaggi naturali e sono dotate di un loop che riporta l'informazione di un nodo, sul nodo stesso). Una rete **DNN** è tipicamente organizzata in una struttura di tipo ***Feedforward*** dove il flusso di dati viaggia dall'ingresso verso l'uscita senza nessun ulteriore loop. Le reti **DNN** vengono chiamate ***Fully connected*** quando hanno i neuroni di ogni layer completamente connessi con i neuroni del layer successivo

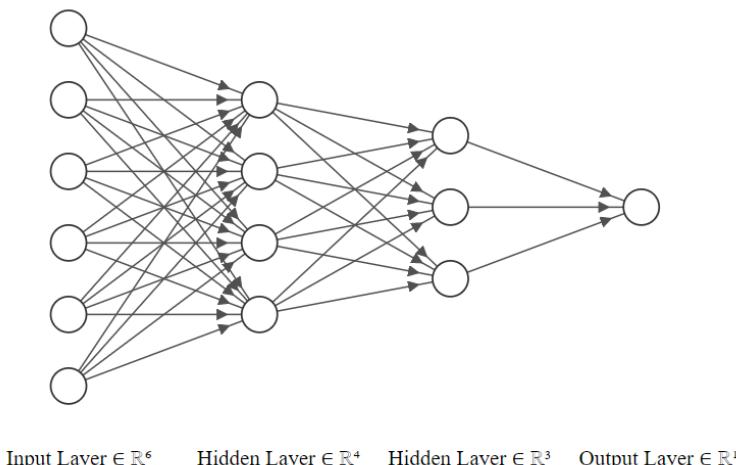


Figura 1.4: Esempio di rete neurale Feedforward Fullyconnected.

1.4 L'algoritmo di Backpropagation

L'algoritmo di Backpropagation, viene utilizzato per far convergere la rete, durante l'addestramento, al minimo locale della *funzione costo*. Quest'ultima funzione misura quanto lavora bene la rete, ed il suo valore dipende dalla differenza tra la predizione \hat{Y} della rete, dato un particolare ingresso X, e l'uscita desiderata Y. Il Gradient Descent (**GD**) è un algoritmo di ottimizzazione del primo ordine, e trova il minimo locale della funzione costo, modificando i pesi W ed i bias b, muovendosi ad ogni *step* in direzione opposta al valore trovato del gradiente della funzione costo, calcolato utilizzando l'intero dataset [8, 13, 19, 31].

In questo elaborato si utilizzeranno le seguenti definizioni: l'approssimazione del **GD**, quando viene stimato attraverso un unico campione, prelevato dal dataset in modo casuale, verrà chiamato **SGD**. Nel caso in cui si utilizzasse un **mini-batch** di sample per la stima del gradiente, il metodo prenderà il nome di Mini-batch Gradient Descent (**MGD**).

1.4.1 Le equazioni fondamentali

Si riportano di seguito solamente le equazioni fondamentali per poter applicare l'algoritmo di Backpropagation correttamente, essendo tale argomento coperto in modo più che sufficiente dal riferimento in bibliografia [31].

Il valore di attivazione del layer l viene definito in forma compatta come:

$$\begin{cases} a^l = \sigma(w^l a^{l-1} + b^l) = \sigma(z^l) \\ z^l = w^l a^{l-1} + b^l \end{cases} \quad (1.8)$$

Data una funzione costo C allora:

$$\delta^L = \nabla_a \odot \sigma'(z^L) \quad (1.9)$$

$$\delta^l = \left((w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l) \right) \quad (1.10)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (1.11)$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (1.12)$$

Mentre per l'aggiornamento dei parametri della rete si utilizzano le equazioni seguenti:

$$w_l \rightarrow w_l - \frac{\eta}{m} \sum_x a_x^{l-1} \delta_x^l \quad (1.13)$$

$$b_l \rightarrow b_l - \frac{\eta}{m} \sum_x \delta_x^l \quad (1.14)$$

1.4.2 L'algoritmo

Algoritmo 1: Algoritmo di Backpropagation

Input: Dataset in ingresso
for *Per tutti gli epoch* **do**
 /* Input: Utilizzando la (1.8) con $l = 1$ */
 calcolo del valore di attivazione del layer di ingresso a^1 ;
 /* Feedforward: Per ogni $l = 2, 3 \dots, L$ calcolare a^l, z^l con la (1.8)*/
 Calcolo Feedforward ;
 /* Errore d'uscita: Trovare il vettore δ^L con la (1.9)*/
 Calcolo errore d'uscita ;
 /* Backpropagate dell'errore: Per ogni $l = L - 1, L - 2 \dots, 2$ calcolare δ^l
 con la (1.10)*/
 Backpropagate dell'errore ;
 /* Output: Il gradiente della funzione costo lo si ricava utilizzando
 (1.11, 1.12)*/
 Calcolo gradiente della funzione costo;
 /* Aggiornamento: tramite l'equazione (1.13, 1.14) */
 Aggiornamento parametri ;
end

Capitolo 2

Teoria dell'Informazione

Nel seguente capitolo, verrà introdotta la teoria necessaria per poter stimare la mutua informazione, analizzando le problematiche legate ad un suo corretto calcolo.

2.1 Mutua Informazione

Claude Shannon è stato il padre fondatore della teoria dell'informazione alla base dei sistemi di Telecomunicazioni. Parte del suo lavoro è stata quella di collegare l'informazione al concetto di entropia, già a quei tempi conosciuta dai processi termodinamici della meccanica statistica, dove veniva usata per misurare il disordine e la complessità di un sistema [34]. Nel caso della teoria dell'informazione invece, l'entropia descrive l'informazione media prodotta da un dato stocastico.

Definizione 2.1. L'informazione, o per essere più precisi la mutua informazione, è una quantità tipicamente espressa in bit che viene fornita da una variabile rispetto un'altra, quando la conoscenza della prima, in media, riduce l'incertezza sulla seconda [15].

Molto simile alla correlazione tale concetto può essere applicato a diversi tipi di dati, catturando iterazioni non lineari, risultando indipendente dal modello usato. In altre parole non è necessario fare delle ipotesi specifiche sulla struttura tra le variabili, e non è inoltre necessario avere tipi di dati particolari. Può essere utilizzata quindi in qualsiasi ambito, anche a seguito di una riduzione della dimensionalità dei dati in ingresso (Principal component analysis ([PCA](#))). Un ulteriore vantaggio della teoria dell'informazione è quello di potere essere applicata ad un sistema multi-variabili.

Sia X il set di tutti i possibili messaggi scambiati tra un trasmettitore ed un ricevitore, il fatto di poter comunicare un'informazione è equivalente a ridurre l'incertezza, ed inoltre, il contenuto informativo di un messaggio dipende dalla probabilità dell'intero set di messaggi.

Per questo motivo si andrà a misurare il contenuto informativo di un messaggio, come il numero di bit medio usato durante la comunicazione dei messaggi $x_i \in X$, se si usasse il codice ottimo per la codifica/decodifica. L'informazione viene in generale calcolata in *nat*, ma può essere convertita in *bit* tramite un fattore moltiplicativo.

2.1.1 Distribuzione di probabilità discreta, probabilità congiunta, probabilità condizionale

Si ricorda al lettore, che la distribuzione di probabilità p , nel caso in cui questa sia discreta, viene definita *distribuzione di probabilità discreta* o *distribuzione marginale*.

Dati due set, $x \subseteq X, y \subseteq Y$, la *probabilità congiunta* $P(x, y)$ e la *probabilità condizionale* $P(x|y)$ sono legate dalla formula di **Bayes**:

$$P(x|y) = \frac{P(x \cap y)}{P(y)} \quad (2.1)$$

La probabilità condizionale è la probabilità di un evento data la conoscenza del verificarsi di un altro.

Si ricorda inoltre, che in caso di indipendenza tra le variabili, vale la seguente relazione:

$$p(x, y) = p(x)p(y) \quad (2.2)$$

Ed in generale che:

$$p(x) = \sum_y p(x, y) \quad (2.3)$$

Attraverso queste formule è possibile costruire i concetti fondamentali della teoria dell'informazione.

Nel seguente elaborato si utilizzeranno le ipotesi fatte da Tishby, il quale tratta l'intero hidden layer T come una singola variabile casuale, caratterizzata da delle distribuzioni di codifica $P(T|X)$ e di decodifica $P(Y|T)$.

2.1.2 Entropia

L'entropia di una variabile discreta casuale X, i cui stati siano x, è definita nel seguente modo [35]:

$$\begin{cases} H(X) = -E(\log(X)) = -\sum_{x \in X} p(x) \log_2 p(x) \geq 0 \\ Hyp. : 0 \cdot \log_2 0 \equiv 0 \end{cases} \quad (2.4)$$

Essa rappresenta il contenuto medio informativo dei messaggi $x \in X$, ognuno dei quali è caratterizzato da una probabilità $p(x)$.

Vale inoltre la proprietà:

$$H(X) \leq \log_2 |X| \quad (2.5)$$

Cioè l'entropia è al più pari al valore della cardinalità dell'insieme X in cui appartengono gli stati, questo valore lo assume nel caso in cui la $p(x) = |X|^{-1}$ per ogni $x \in X$.

Si può pensare all'entropia come al numero di risposte binarie di tipo "si/no", alle domande che possono venir chieste su Y in modo da venir a conoscenza del suo valore.

2.1.3 Entropia congiunta, entropia condizionale ed entropia relativa

L'entropia congiunta $H(X, Y)$ di due variabili casuali discrete (X, Y) , i cui stati $x \in X, y \in Y$, è definita come:

$$0 \leq H(X, Y) = -\sum_{(x \in X, y \in Y)} p(x, y) \log_2 p(x, y) \leq \log_2 |X| \quad (2.6)$$

Essa rappresenta il contenuto informativo medio dei messaggi (x, y) con probabilità $p(x, y)$.

Nel caso in cui due variabili fossero indipendenti vale:

$$H_{ind}(X, Y) = H(X) + H(Y) \quad (2.7)$$

L'entropia condizionale $H(Y|X = x)$ è il numero di "si/no", alle varie domande che si possono fare su Y, in modo da venire a conoscenza del suo valore se si conoscesse il valore di $x \in X$. D'altro canto $H(Y|X)$ è la media su tutti i possibili eventi $x \in X$, $H(Y|X) = \mathbb{E}[H(Y|X = x)]$.

L'entropia condizionale $H(Y|X)$ viene formalmente definita come:

$$H(Y|X) = - \sum_{(x,y) \in A} p(x,y) \log_2 p(y|x) \geq 0 \quad (2.8)$$

ed è definita come l'incertezza media di una variabile, dato lo stato di un'altra.

Valgono inoltre le seguenti relazioni:

$$H(Y|X) \geq 0 \quad (2.9)$$

$$H(Y|X) \leq H(Y) \quad (2.10)$$

$$H(X, Y) = H(X) + H(Y|X) \quad (\text{chain rule}) \quad (2.11)$$

$$H(X|Y) \geq H(X) \quad (2.12)$$

$$H(X|Y) \neq H(Y|X) \quad (\text{in generale}) \quad (2.13)$$

Un teorema interessante che Shannon ha fornito è il seguente:

Teorema 2.2. *L'Entropia rappresenta il contenuto informativo di una variabile casuale, ed essa è uguale al numero medio di bit usati per messaggio, nel caso in cui venisse usato il codice ottimo per rappresentare i possibili valori di una variabile casuale. Il codice ottimo è un codice di codifica che usa in media la più corta lunghezza per la sua versione codificata.*

Nel caso in cui la variabile X fosse continua, il segno di sommatoria viene sostituito da un integrale nel calcolo dell'entropia.

Per due variabili discrete, casuali, descritte dalle distribuzioni di probabilità $p(x)$ e $q(y)$ con $x, y \in A$, si definisce l'entropia relativa (anche chiamata distanza di Kullback-Leibler), nel seguente modo:

$$KL(p||q) = \sum_{x \in A} p(x) \log_2 \left[\frac{p(x)}{q(x)} \right] \geq 0 \quad (2.14)$$

L'informazione mutua è una misura nel campo della Teoria dell'Informazione, che quantifica la quantità di informazione che una variabile casuale contiene su un'altra. In modo equivalente è la riduzione di incertezza di una variabile, quando si è a conoscenza del valore assunto da un'altra. Se conoscere lo stato di una variabile riduce l'incertezza di un'altra, allora in media la prima variabile fornisce informazioni sulla seconda.

L'entropia relativa viene calcolata come il rapporto tra la **probabilità congiunta** $p(x,y)$ ed il prodotto tra le due distribuzioni di probabilità $p(x)$, $p(y)$, dove X ed Y sono due variabili discrete casuali [52].

$$\begin{aligned} I(X, Y) &= KL(p(x, y) \| p(x)p(y)) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right) \quad (2.15) \\ &= \sum_{x \in X} \sum_{y \in Y} p(x, y) \log \left(\frac{p(x|y)}{p(x)} \right) = H(X) - H(X|Y) \geq 0 \end{aligned}$$

Inoltre vale che:

$$I(X, Y) = I(Y, X) \quad (2.16)$$

Quando X ed Y sono indipendenti l'informazione mutua è nulla, indicando che X non contiene informazioni su Y.

È possibile dimostrare inoltre che vale la seguente relazione:

$$I(X, Y) = H(X) - H(X|Y) \geq 0 \quad (2.17)$$

dove $H(X)$ come detto prima è l'entropia della variabile X, mentre $H(X|Y)$ è l'entropia condizionale di X dato Y. Tale equazione conferisce un significato conciso alla mutua informazione $I(X, Y)$. Essa può essere interpretata infatti come la differenza tra l'ammontare medio di informazione di un messaggio x, ed il residuo di informazione che rimane dopo che è noto il valore di y. Di conseguenza $I(X, Y)$ è l'ammontare medio di informazione che y rivela di x e viceversa.

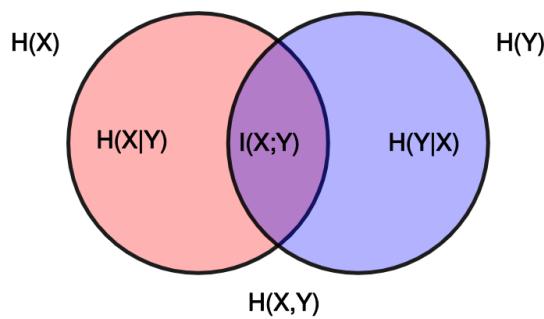


Figura 2.1: Diagramma di Venn che mostra la relazione che c'è tra l'informazione mutua tra due variabili e la loro entropia.

Per simmetria inoltre vale che:

$$I(X, Y) = H(X) + H(Y) - H(X, Y) \geq 0 \quad (2.18)$$

Se tre variabili casuali $(x, y, z) \in A$ sono collegate da una Catena di Markov $X \rightarrow Y \rightarrow Z$, allora si ha la cosiddetta Data Processing Inequality (DPI):

$$I(X, Y) \geq I(X, Z) \quad (2.19)$$

La DPI mostra esplicitamente come l'informazione all'interno di una Catena di Markov può solo che diminuire quando le variabili sono casuali. Da precisare che si ha una Catena di Markov solamente quando si è giunti a regime dopo la fase di addestramento.

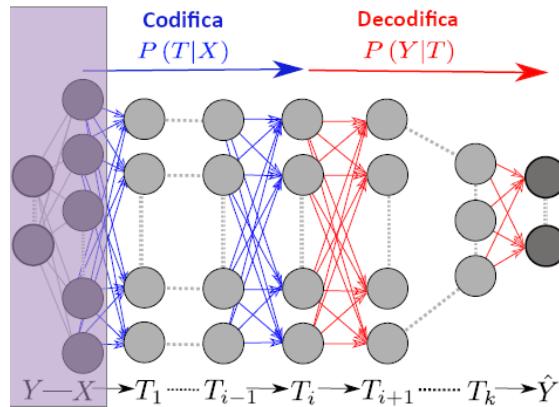


Figura 2.2: I layer di una DNN formano alla fine della fase di addestramento una Catena di Markov, che viene attraversata dall'ingresso X, fornendo in uscita una predizione dell'etichetta \hat{Y} . Ogni rappresentazione T dell'informazione compressa dell'ingresso X, può essere quantificata mediante la coppia di punti $I(X, T)$ e $I(T, Y)$ nel piano dell'informazione. Allo stesso modo al punto può essere associato una distribuzione di probabilità per la codifica $P(T|X)$ ed una di decodifica $P(\hat{Y}, T)$. Si è scelto di inserire l'etichetta vera Y a sinistra della rete, poiché questo è in generale diverso dall'etichetta predetta \hat{Y} .

Riassumendo, si è visto come l'entropia rappresenta il contenuto informativo di una variabile casuale, e che se si usasse il codice ottimo per rappresentare i suoi stati, il numero medio di bit usati per messaggio sarebbe uguale all'entropia H. La variabile Y è stata inserita a sinistra della Figura 2.2, poiché si vuole sottolineare che la predizione dell'etichetta fatta dalla rete (\hat{Y}), è in generale diversa dalla etichetta vera Y.

Si può inoltre vedere dalla Figura 2.2, come per ogni hidden layer T_i , c'è una mappa di codifica dall'ingresso fino al layer considerato e di decodifica dal layer considerato fino all'uscita.

Definizione 2.3. Un codice binario $C : X \rightarrow \bigcup_{L \geq 1} \{0, 1\}^L$ mappa tutti i messaggi del set X ad un set di stringhe binarie, associando una parola codificata $C(x)$ ad ogni messaggio $x \in X$.

Definizione 2.4. La lunghezza $l(x)$ di una codifica $C(x)$ viene definita come il numero di simboli nella stringa $C(x)$, i.e. $l(x) = 1$ se $C(x) \in \{0, 1\}^l$.

Definizione 2.5. La lunghezza $L[C]$ del codice binario C è la lunghezza media delle sue parole codificate $L[C] = \sum_{x \in A} p(x)l(x)$.

Definizione 2.6. Il codice ottimo C per una data variabile casuale, è quello che codifica in maniera univoca la variabile in maniera tale da avere la lunghezza più piccola $L[C]$.

Teorema 2.7. *Ogni codice C usato per codificare un messaggio $x \in X$ rispetta la seguente diseguaglianza: $L[C] \geq H(X)$, con l'uguaglianza se $l(x) = \log_2[\frac{1}{p(x)}]$ per ogni $x \in X$.*

Questo teorema indica che non si può utilizzare un codice C che permetta di comunicare un messaggio $x \in X$ in maniera tale che il numero medio di bit usati sia minore dell'entropia: $L[C] \geq H(X)$.

Teorema 2.8. *Per ogni set di messaggi descritti dalla variabile casuale $x \in X$, esiste un codice C con la proprietà $L[C] < H(X) + 1$.*

Il secondo teorema d'altra parte indica che per ogni variabile casuale $x \in X$, esiste un codice unico che arriva molto vicino a $L[C] < H(X) + 1$.

Si può ora riassumere la relazione che c'è tra codici ottimi ed entropia in un modo più compatto:

Teorema 2.9. *Il codice ottimo C che codifica un messaggio descritto dalla variabile casuale $x \in X$, implica un numero medio di bit per messaggio $L[C]$ che rispetta la seguente relazione:*

$$H(X) \leq L[C] \leq H(X) + 1. \quad (2.20)$$

Da questo teorema deriva automaticamente che nel caso in cui la cardinalità del set X sia finita, si può approssimare l'equazione (2.20), ottenendo un numero medio di bit per messaggio molto vicino ad $H(X)$.

2.2 Il metodo dell'Information Bottleneck

Il metodo dell'Information Bottleneck (**IB**), presentato da Tishby, Pereira e Bialek nel 2000, era stato inizialmente sviluppato per essere utilizzato in un sistema di telecomunicazioni, trovando però poi successivamente impiego anche nelle **DNN** [44].

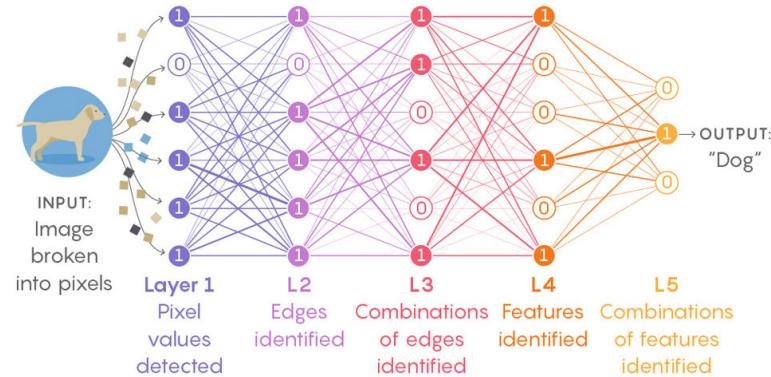


Figura 2.3: Quando i dati entrano dentro la rete, ogni neurone che vede in 'fire' (uscita alta), trasmette un impulso ai neuroni a cui è collegato, facendoli così risultare più propensi a loro volta ad attivarsi. Il processo filtra il rumore ed estrae solo le caratteristiche principali.

Siano X e Y due variabili casuali discrete, dipendenti con una certa *distribuzione congiunta* $p(x,y)$, con $x \in X$, $y \in Y$.

Il problema dell'**IB** consiste nel trovare un codice corto per X che ne comprima l'informazione sotto un certo limite ($I(X, T) \leq r$) mantenendo informazioni rilevanti sull'uscita ($I(T, Y) \geq i$), concentrando le informazioni che X ha di Y attraverso un "collo di bottiglia", formata da un set limitato di parole codificate T . Questo problema di ottimizzazione può essere pensato come trovare un codificatore ottimo delle informazioni rilevanti, con una rappresentazione efficiente.

Una proprietà importante da accennare dell'informazione mutua, è l'invarianza rispetto ad una trasformazione invertibile:

$$I(X, Y) = I(\Theta(X), \Psi(Y)) \quad (2.21)$$

Come detto precedentemente l'**IB** introduce un'ulteriore variabile T , che è una rappresentazione compressa di X e soddisfa la **DPI**, dove T non può contenere più informazioni riguardo ad Y di quelle che contiene rispetto a X : $I(X, T) \geq I(T, Y)$. Per trovare una rappresentazione ottima di X rispetto Y , si utilizza la classica nozione di Minimal sufficient statistics (**MSS**), la quale verrà successivamente rilassata per poter ricavare una famiglia di soluzioni dell'**IB** framework, che prenderanno il nome di **IB Curve**. La **MSS** di X rispetto Y in questo caso assume il ruolo del codice ottimo.

Questo problema di ottimizzazione può essere visto come una generalizzazione del problema del *Rate Distortion*, in cui la misura della distorsione vale $d(x, \hat{x})$. Si otterranno così un insieme di equazioni autoconsistenti, la cui soluzione è una generalizzazione dell'algoritmo di *Blahut-Arimoto*. Tale procedura consente di ottenere un framework utilizzabile per lo studio della fase di apprendimento delle reti neurali.

Teorema 2.10 (Teorema del piano dell'informazione). *Data una rete, la complessità dei campioni di una DNN sono completamente determinati dalla mutua informazione della codifica $I(X, T)$, dell'ultimo hidden layer. L'accuratezza invece, la quale và a definire l'errore di generalizzazione, è determinata dall'informazione di decodifica $I(T, Y)$ dell'ultimo hidden layer.*

2.2.1 Soluzione dell'IB framework

Per poter estrarre solamente le informazioni rilevanti da un set di dati è necessario definire per prima cosa, “cosa è rilevante”. Nel problema di *Rate Distortion*, da cui l'**IB** prende spunto, viene caratterizzato il tradeoff, tra la rappresentazione del segnale e la media di distorsione del segnale ricostruito. In questo modo si trova la *Rate Distortion Function* $R(D)$, la quale indica le caratteristiche rilevanti del segnale in ingresso. Essendo quest’ultima funzione difficile da determinare, si sfrutta l’informazione portata dall’etichetta nell’apprendimento supervisionato, che indica cosa è rilevante dell’ingresso per poter predire l’uscita.

Sia X lo spazio dei messaggi in ingresso ed \hat{X} la sua rappresentazione compressa (Figura 2.4). Si assume che tale spazi siano finiti, nel caso in cui non lo fossero si procede alla quantizzazione, come descritto nella sezione relativa al binning statico (Sezione 2.4).

Ogni valore $x \in X$ si mappa in un altro insieme \hat{X} , caratterizzato da una distribuzione condizionale $p(\hat{x}|x)$, tale mappa induce una partizione di X in blocchi, dove ad ognuno di questi è associato un elemento $\hat{x} \in \hat{X}$, con una probabilità data da:

$$p(\hat{x}) = \sum_x p(x)p(\hat{x}|x) \quad (2.22)$$

Questo deriva dalla classica nozione di *minimal sufficient statistics*, la quale fornisce dei buoni candidati per trovare una rappresentazione ottima di X in Y .

Definizione 2.11. Le *Sufficient statistics* sono mappe o partizioni di X , $S(X)$, che catturano tutte le informazioni che X ha di Y , ottenendo $I(S(X), Y)$.

Definizione 2.12. La *Minimal Sufficient statistics* $T(X)$, è la più semplice *sufficient statistics* che induce la più grezza partizione di X . Questa condizione esiste solamente per certe distribuzioni particolari, per esempio esponenziali, e deve essere rilassata, richiedendo di catturare più informazioni possibili sull’uscita ma non tutte.

Il volume medio degli elementi in X mappati sulla stessa parola codificata è $2^{H(X|\hat{X})}$, dove l'entropia condizionale X dato \hat{X} vale:

$$H(X|\hat{X}) = - \sum_{x \in X} \sum_{\hat{x} \in \hat{X}} p(\hat{x}|x) \log(p(\hat{x}|x)) \quad (2.23)$$

La qualità della quantizzazione è caratterizzata dal limite inferiore $I(\hat{X}, X)$, imposto attraverso delle ulteriori condizioni al contorno fornite dalla funzione di distorsione d , tale funzione specifica quali sono le informazioni più rilevanti di X . Il partizionamento di X implica una distorsione attesa:

$$\langle d(x, \hat{x}) \rangle_{p(x, \hat{x})} = - \sum_{x \in X} \sum_{\hat{x} \in \hat{X}} p(x, \hat{x}) d(x, \hat{x}) \quad (2.24)$$

In base al teorema di *Shannon e Kolmogorov*, viene caratterizzato il tradeoff tra il rate di quantizzazione e la distorsione attesa, attraverso la *Rate Distortion Function* $R(D)$, definita come il minimo rate sotto alcune condizioni della distorsione attesa [15]:

$$R(D) \equiv \min_{p(\hat{x}, x): \langle d(x, \hat{x}) \rangle \leq D} I(X, \hat{X}) \quad (2.25)$$

Trovare la $R(D)$ è un problema variazionale, risolvibile tramite i moltiplicatori di Lagrange β , la cui minimizzazione porta alla soluzione:

$$p(\hat{x}|x) = \frac{p(\hat{x})}{Z(x, \beta)} e^{-\beta d(x, \hat{x})} \quad (2.26)$$

2.2.2 L'algoritmo di Blahut-Arimoto

Le equazioni (2.22, 2.26) devono essere risolte iterativamente, come descritto dall'*Algoritmo di Blahut-Arimoto* per il calcolo della *Rate distortion function*, iterando in t fino la convergenza [5, 9]:

$$\begin{cases} p_{t+1}(\hat{x}) = \sum_x p(x) p_t(\hat{x}|x) \\ p_t(\hat{x}|x) = \frac{p_t(\hat{x})}{Z_t(x, \beta)} e^{-\beta d(x, \hat{x})} \end{cases} \quad (2.27)$$

Il sistema di Equazioni (2.27) non mantengono però informazioni rilevanti sull'uscita Y . Come visto precedentemente si vuole che \hat{X} comprima X il più possibile, ma anche che catturi più informazioni possibili su Y . L'ammontare di informazioni di Y in \hat{X} è dato da:

$$I(\hat{X}, Y) = \sum_y \hat{x} p(y, \hat{x}) \log \frac{p(y, \hat{x})}{p(y)p(\hat{x})} \leq I(X, Y) \quad (2.28)$$

Si può dimostrare che la soluzione del problema appena esposto non è unica [44]. Per trovare questo valore bisogna minimizzare la funzione di Lagrange:

$$\mathcal{L}[p(\hat{x}|x)] = I(X, \hat{X}) - \beta I(\hat{X}, Y) \quad (2.29)$$

La variabile β è il moltiplicatore di Lagrange il quale decide il livello di tradeoff tra il livello di compressione $I(X, \hat{X})$ e l'ammontare dell'informazione mantenuta $I(Y, \hat{X})$. Un valore di $\beta=0$ indica una quantizzazione il più possibile grossolana, mentre se $\beta \rightarrow \infty$ vengono catturate tutte le informazioni, in questo modo si può controllare il tradeoff tra informazioni preservate e livello di compressione con risoluzioni differenti [44]. Facendo la minimizzazione del funzionale (Equazione 2.29) e mettendo insieme i risultati precedenti si giunge al seguente set di equazioni:

$$p(\hat{x}|x) = \frac{p(\hat{x})}{Z(x, \beta)} e^{-\beta \sum_y p(y|x) \log \frac{p(y|x)}{p(y|\hat{x})}} \quad (2.30)$$

$$= \frac{p(\hat{x})}{Z(x, \beta)} e^{-\beta D_{KL}[p(y|x) \| p(y|\hat{x})]} \\ p(y|\hat{x}) = \frac{1}{p(\hat{x})} \sum_x p(y|x)p(\hat{x}|x)p(x) \quad (2.31)$$

$$p(\hat{x}) = \sum_x p(\hat{x}|x)p(x) \quad (2.32)$$

$$Z(x, \beta) = \sum_{\hat{x}} p(\hat{x}) e^{-\beta D_{KL}[p(y|x) \| p(y|\hat{x})]} \quad (2.33)$$

Dove $p(\hat{x})$ è definito nell'Equazione (2.22). Mentre D_{KL} è la divergenza di Kullback-Leibler e misura l'effettiva distorsione, potendo infatti assumere che: $d(x, \hat{x}) = D_{KL}[p(y|x) \| p(y|\hat{x})]$.

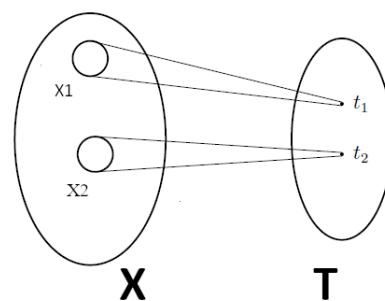


Figura 2.4: Procedura di compressione dell'insieme X , nella quale i vari sottoinsiemi vengono mappati in sottoinsiemi di dimensione minore, nell'insieme T .

2.2.3 IB Curve

Come è stato fatto precedentemente, dopo la minimizzazione del funzionale si giunge ad un insieme di Equazioni (2.35) da iterare in s fino la convergenza [44].

Sia $t \in T$ la rappresentazione compressa di $x \in X$, e siano date $p(t|x)$, $p(t)$, $p(y|t)$ e la Catena di Markov $Y \rightarrow X \rightarrow T$. Formalmente l'Information bottleneck minimizza la Lagrangiana:

$$\min_{p(t|x), p(t), p(y|t)} \{I(X, T) - \beta I(T, Y)\} \quad (2.34)$$

Definizione 2.13. La parte rilevante che X ha di Y , che verrà chiamata T , è la *minimal sufficient statistics* di X rispetto Y , ed è l'associazione più semplice di X che cattura l'informazione mutua $I(X, Y)$. Considerando la Catena di Markov $Y \rightarrow X \rightarrow T$, si minimizza l'Equazione (2.34) per minimizzare l'informazione mutua $I(X, T)$ con una condizione al contorno per $I(T, Y)$.

La soluzione implicita a questo problema è data dal seguente set di equazioni autoconsistenti da iterare ad ogni step s :

$$\begin{cases} p_s(t|x) = \frac{p_s(t)}{Z_s(x, \beta)} e^{-\beta D_{KL}[p(y|x) || p(y|t)]} \\ p_{s+1}(y|t) = \sum_x p(y|x)p_s(x|t) \\ p_{s+1}(t) = \sum_x p(x)p_s(t|x) \end{cases} \quad (2.35)$$

Dove l'espressione di D_{KL} è presente nell'Equazione (2.15). Queste equazioni sono soddisfatte sull'**IB Curve**.

All'aumentare del moltiplicatore di Lagrange β , si ottiene una famiglia di soluzioni diverse all'interno del piano dell'informazione (I_X, I_Y) . Per distribuzioni regolari di $P(X, Y)$, per esempio quando Y non è completamente deterministica in X , si ha un'unica pendenza della curva.

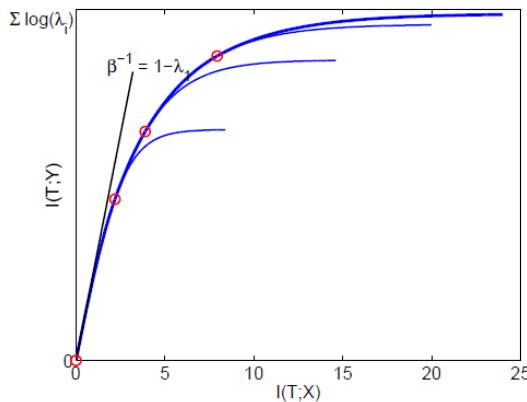


Figura 2.5: IB Curve al variare del coefficiente di Lagrange β .

2.2.4 IB framework utilizzato nelle DNN

La capacità di una [DNN](#) in un processo di apprendimento supervisionato di generalizzare, è correlata alla sua capacità di comprimere l'informazione, mantenendo informazioni rilevanti sull'uscita. In modo equivalente si può dire che il processo di apprendimento cerca di catturare una rappresentazione efficiente delle informazioni, trovando un'approssimazione di *minimal sufficient statistics* dell'ingresso rispetto l'uscita. La teoria suggerisce che la compressione è un metodo per eliminare informazioni poco rilevanti e rumorose dall'ingresso, per ottenere una certa uscita. Solitamente l'ingresso X è composto da una quantità enorme di informazioni, per esempio pixel di un'immagine, mentre l'uscita Y è molto più semplice, indicando che la maggior parte dell'entropia di X non fornisce abbastanza informazioni su Y , e le caratteristiche presenti in X sono difficili da estrarre [45, 46].

Tishby e Schwartz-Ziv hanno utilizzato il framework dell'Information Bottleneck per analizzare le [DNN](#) e visualizzare la traiettoria della mutua informazione durante l'addestramento. In questo modo hanno quantificato la quantità di informazione che ogni hidden layer contiene rispetto l'ingresso X e l'uscita Y . In base alle loro osservazioni sperimentali, Tishby e Schwartz-Ziv hanno affermato che l'addestramento di una rete è composto da due fasi: la **fase di apprendimento** (l'hidden layer T acquisisce informazioni dall'ingresso e dall'uscita facendo aumentare sia $I(X,T)$ che $I(T,Y)$), e la **fase di compressione** dove la rete cerca di rimuovere informazioni inutili che non contribuiscono a predire Y (in questa fase $I(X,T)$ decresce mentre $I(T,Y)$ rimane alto) [45].

Si afferma inoltre che durante la fase di addestramento anche le [DNN](#) cercano automaticamente la rappresentazione ottima dell'ingresso rispetto l'uscita, per ogni hidden layer T , proprio come viene fatto nell'[IB](#) [45].

Tishby e Schwartz-Ziv fanno così le seguenti affermazioni:

- Il [SGD](#) se applicato ad una rete *Fully Connected*, trova la soluzione del metodo dell'[IB](#) durante la fase della compressione, di conseguenza tende a portare la traiettoria dell'informazione verso la [IB Curve](#).
- La fase di compressione migliora la generalizzazione perché rimuove le informazioni irrilevanti ed evita l'*overfitting*.

In questa tesi si vuole utilizzare il metodo dell'[IB](#) per studiare e visualizzare la dinamica di una rete [DNN](#) durante l'addestramento. Modificando i parametri, le funzioni di attivazione e utilizzando dataset differenti da quelli proposti da Tishby. L'informazione mutua verrà calcolata utilizzando algoritmi differenti per esserne certi di una sua corretta valutazione. Inoltre si cercherà di capire da dove deriva il fenomeno della compressione descritta da Tishby e Schwartz-Ziv [45], seguendo le idee di base di Saxe et al. [33].

2.2.5 Information Plane

Sia data una Catena di Markov di una DNN formata da k-layer, si definisce T_i l'uscita dell'i-esimo hidden layer, che può essere pensata come una singola ***multivariate variable***. Ognuna di queste multivariate variable è caratterizzata dalla sua distribuzione di codifica ($P(T_i|X)$) e di decodifica ($P(Y|T_i)$). I layer soddisfano la seguente catena di DPI:

$$I(X, Y) \geq I(T_1, Y) \geq I(T_2, Y) \geq I(T_3, Y) \cdots \geq I(T_k, Y) \geq I(\hat{Y}, Y) \quad (2.36)$$

$$H(X) \geq I(X, T_1) \geq I(X, T_2) \geq I(X, T_3) \cdots \geq I(X, T_k) \geq I(X, \hat{Y}) \quad (2.37)$$

L'Equazione (2.36) indica che l'informazione di Y che viene persa in ogni layer, non può essere recuperata dai layer successivi. Come visto precedentemente pure in questo caso ogni layer cerca di massimizzare $I(Y, h_i)$ e di minimizzare $I(h_{i-1}, h_i)$. Data una certa probabilità congiunta $P(X, Y)$, T_i è unicamente mappato in un punto dell'Information Plane con coordinate $I(X, T_i)$ e $I(T_i, Y)$.

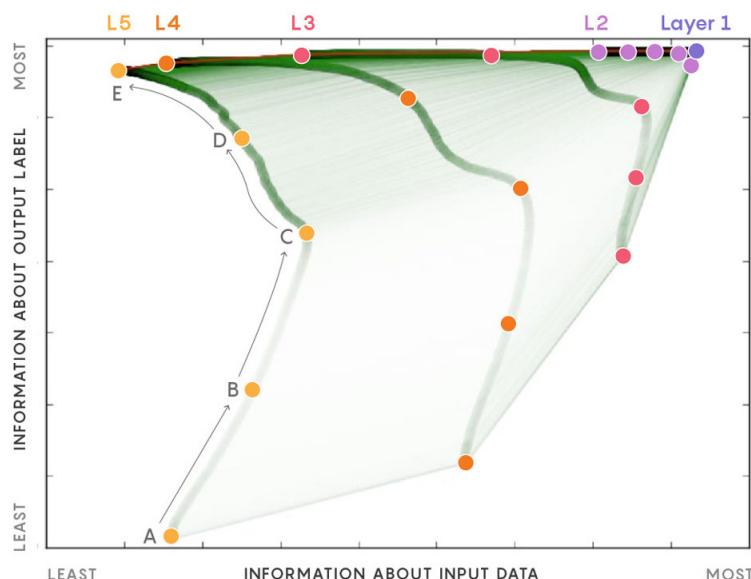


Figura 2.6: Fino al punto C l'uscita del layer, la quale viene interpretata come essere una unica variabile, acquisisce informazioni sull'ingresso e sull'uscita migliorando la predizione. Tra la fase C ed E la rete entra nella fase di compressione andando a scartare le informazioni irrilevanti dell'ingresso, raggiungendo così l'ottimo tra accuratezza e compressione.

2.3 Stima dell'Informazione Mutua

In questa sezione si andrà ad argomentare come prima cosa che l'algoritmo di ottimizzazione dell'**IB** nelle **DNN** è un problema *ill-posed*¹, successivamente si andrà ad illustrare come stimare l'informazione mutua nel modo più corretto possibile tra gli hidden layer e l'ingresso, e tra gli hidden layer e l'uscita.

Per poter stimare le distribuzioni di probabilità, e quindi l'informazione, lo si fà attraverso dei campioni prelevati durante la fase di addestramento o facendo delle assunzioni sul modello.

2.3.1 Problema ill-posed

Si assumerà come prima cosa che ogni campione in ingresso è indipendente e distribuito in modo uniforme (*i.i.d.*), considerando l'uscita di un hidden layer una variabile T_i , allora questa assume un valore deterministico rispetto l'ingresso X (T_i è una *Kronecker Delta Function* in $f(X)$, dove $f(\bullet)$ è dettata dalle operazioni svolte dalla **DNN**). Si ottiene:

$$I(X, T_i) = H(T_i) - H(T_i|X) = H(T_i) = - \sum_{i=1}^N \frac{1}{N} \log \left(\frac{1}{N} \right) = \log(N) \quad (2.38)$$

Essendo $p(T = T_i) = \frac{1}{N}$, e poiché il processo di addestramento è deterministico, viene assunto che $H(T_i|X) = 0$ [33].

Si è inoltre assunto che la variabile T_i sia discreta, portando l'entropia ad essere pari ad:

$$H(T_i) = - \sum_{i=1}^N p_i \log p_i \quad (2.39)$$

Nel caso in cui T_i fosse continua si avrebbe avuto invece:

$$H(T) = - \int p_t(t) \log p_t(t) dt \quad (2.40)$$

Essendo nel caso considerato, p_Z una funzione Delta, si ottiene $H(T_i|X) = -\infty$ e l'informazione mutua, essendo questa legata all'entropia condizionale attraverso l'Equazione (2.17), è generalmente infinita fintanto che $H(T_i)$ assume un valore finito.

Per poter osservare la traiettoria dell'informazione si ha bisogno di un valore finito di $H(T_i|X)$, e per evitare la soluzione banale si aggiunge del rumore all'uscita dei neuroni causandone un valore finito dell'informazione.

¹Cioè non rispetta le seguenti tre condizioni: ha una soluzione, ha un'unica soluzione, ha una soluzione che dipende dall'ingresso

Aggiungere del rumore inoltre permette di poter studiare i fenomeni come dei processi stocastici e non deterministici.

Ci sono due modi principali per aggiungere del rumore, un modo è aggiungere del rumore Z (variabile indipendente da X) direttamente su T ed ottenere una variabile rumorosa $\hat{T} = T + Z$, in questo caso $H(T|X) = H(Z)$ e l'informazione mutua diventa $I(\hat{T}, X) = H(\hat{T}) - H(Z)$. Quando il rumore additivo è Gaussiano, l'informazione mutua può essere approssimata usando il Kernel Density Estimation ([KDE](#)) con l'assunzione che Z sia distribuita come una *Gaussian mixture* [24]. Il secondo modo per aggiungere del rumore è discretizzare la variabile continua in bin, in questo modo il rumore nasce poiché si approssima la distribuzione di probabilità di una variabile casuale [33, 45].

Quando si lavora con l'informazione nelle [DNN](#), il valore analizzato è il risultato di un processo di stima molto sensibile, nel caso in cui la funzione di attivazione sia di tipo a “saturazione”, non c'è nessun problema essendo la funzione limitata. Quando invece si usano funzioni non limitate, il livello di rumore portato dalla stima deve essere proporzionale all'uscita dei neuroni, adattato per ogni layer e per ogni epoch (*binning dinamico*).

Da qui in avanti su assumerà, come fatto da Tishby, di trattare ogni hidden layer come una variabile scalare casuale.

2.4 Binning

Come si è potuto vedere precedentemente, le variabili in uscita dai neuroni assumono un valore continuo, per poterle discretizzare ci sono diverse tecniche, quali il *binning* o per esempio attraverso l'algoritmo *k-mean*. Questo passaggio è necessario perché lavorare con variabili continue può introdurre risultati poco chiari, infatti nel caso di variabili continue l'entropia può assumere anche un valore negativo.

Nel [capitolo 3](#) si utilizzerà il *binning statico* per andare ad approssimare la distribuzione di probabilità di una variabile continua. La larghezza del bin è un parametro fondamentale in questo caso, perché controlla il tradeoff tra catturare tante e poche informazioni rispetto la vera distribuzione [47]. Nella prossima sezione si andrà a vedere come determinare una possibile scelta per la grandezza del bin nel caso statico. Infine una volta trovata l'approssimazione della distribuzione di probabilità si può calcolare l'informazione mutua.

Nel binning dinamico, la grandezza del bin viene calcolato in modo indipendente per ogni layer e per ogni epoch, e verrà accennato successivamente, per essere studiato in maggior dettaglio in un eventuale lavoro futuro. Questa tecnica risulta infatti essere promettente per funzioni di attivazione con andamento non a saturazione (Sezione [3.7](#)).

2.4.1 Binning statico

Il binning *statico/uniforme* richiede come parametro il numero di bin N_{bins} , tutti i bin hanno la stessa lunghezza e per determinarla si utilizza la seguente formula:

$$d = \frac{L_{max} - L_{min}}{N_{bins}} \quad (2.41)$$

Dove L_{max} ed L_{min} sono i valori massimi e minimi dell'uscita dei neuroni durante l'addestramento della rete. L'uscita di questo processo è l'indice dell'istogramma nel quale il dato ricade. L'uscita vale n per il valore in ingresso x se soddisfa:

$$L_{min} + nd \leq x \leq L_{min} + (n + 1)d \quad (2.42)$$

La figura seguente mostra come funziona il binning uniforme sulle varie funzioni di attivazione introdotte nella Sezione 1.3.

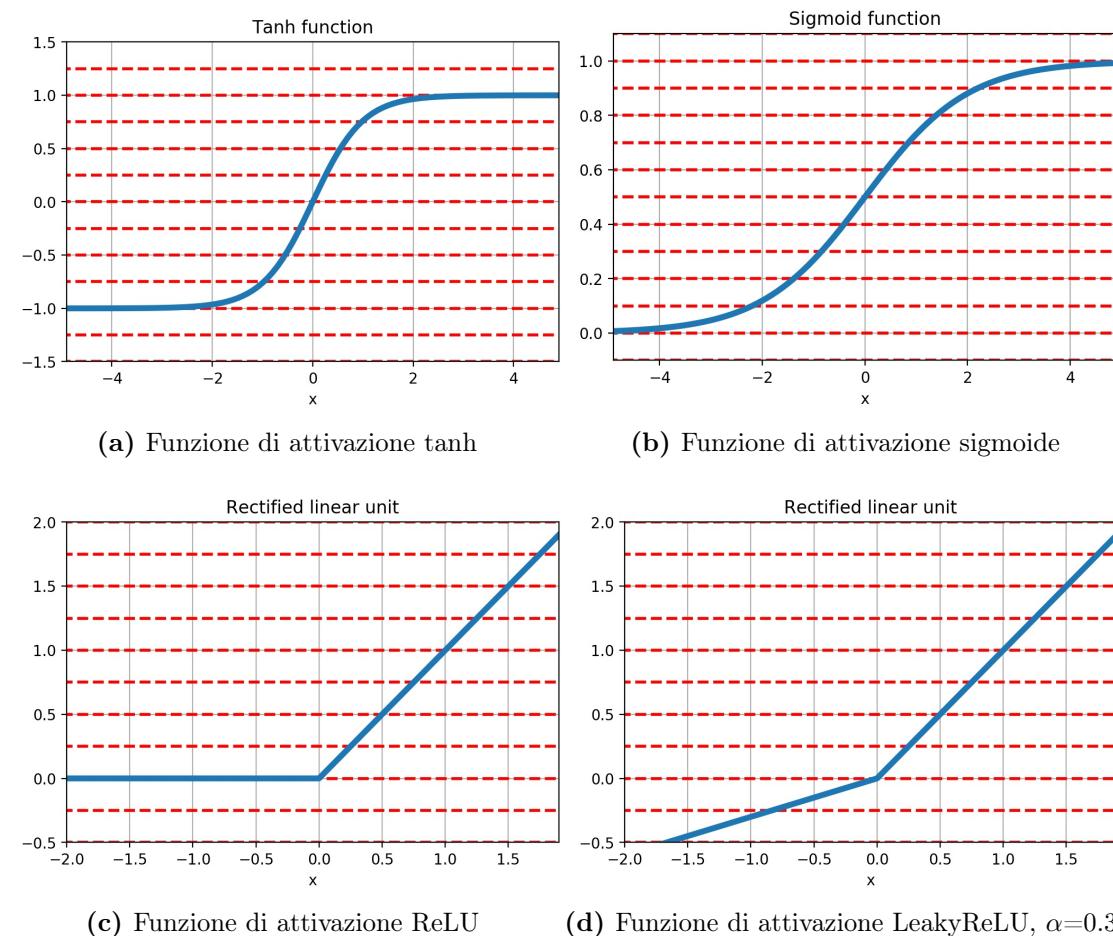


Figura 2.7: Grafici delle funzioni di attivazione utilizzate, discretizzate tramite binning statico.

Come si può notare, mentre per la funzione tanh e sigmoide L_{min} ed L_{max} sono limitate superiormente ed inferiormente, questo non vale per la funzione ReLU e LeakyReLU. In questo caso infatti bisogna impostare i valori di L_{max} ed L_{min} pari al valore massimo e minimo di attivazione, ottenuti in uscita dai neuroni durante l'addestramento. Questo richiede di conseguenza di effettuare il binning, e quindi il calcolo dell'informazione, in una fase successiva, non appena l'addestramento della rete ha terminato.

In letteratura sono presenti diverse metodologie per scegliere il numero di bin ottimali in base alla distribuzione dei dati. Nel [capitolo 3](#) si utilizzerà un numero di bin pari a 30, come suggerito da Tishby, ma nelle [conclusioni](#) si farà un'analisi più approfondita. Questo perché i risultati ottenuti possono variare in maniera significativa, quando si utilizza un numero di bin errato e le funzioni di attivazione non hanno un andamento a saturazione [50].

Capitolo 3

Prove Sperimentali

3.1 Dataset

In questa sezione si andranno a presentare gli algoritmi usati per la generazione dei vari dataset utilizzati all'interno dell'elaborato. La differenza tra i vari dataset realizzati è comunque esigua e non hanno portato a differenze significative per le traiettorie dell'informazione. Il test dataset, essendo stato realizzato in maniera praticamente casuale ed indipendente dal train dataset (ad esclusione del *MNIST* e del *CIFAR*), come ci si può aspettare, non porta ad una grande accuratezza.

3.1.1 Dummy Dataset

Algoritmo del primo dataset realizzato:

Algoritmo 2: Algoritmo creazione Dummy Dataset

Input: N. train samples, N. test samples
/* vettore get_f lungo 4096 con valori $\in [0,1]$ */
Data: get_f
/* vettore ix_train lungo N.train con valori int $\in [0,4096]$ casuali */
/* vettore ix_test lungo N.test con valori int $\in [0,4096]$ casuali */
Data: ix_train , ix_test
/* vettore train_binario lungo N.train con valori 12 bit di ix_train */
/* vettore test_binario lungo N.test con valori 12 bit di ix_test */
Output: train_binario, train_label, test_binario, test_label
inizializzazione casuale;
for Tutti i campioni train **do**
| label_train[campione_i] = get_f[ix_train[campione_i]];
end
for Tutti i campioni test **do**
| label_test[campione_i] = get_f[ix_test[campione_i]];
end

Tabella 3.1: Caratteristiche Dummy Dataset.

Dummy Dataset					
Training Samples	Test Samples	N. Classi	Dimensione	μ	σ
3096	1000	2	12	0.507	0.499

3.1.2 IB Dataset

Poiché non è stato spiegato in maniera chiara come è stato generato il dataset utilizzato da Tishby [45], si è deciso di creare una propria versione del dataset, che da qui in poi verrà chiamato *dataset IB*, anche se differisce da quello originale utilizzato nell'articolo.

Algoritmo 3: Algoritmo creazione IB Dataset

Input: N. train samples, N. test samples
/ vettore train_int lungo N.train con valori int $\in [0,4096]$ casuali*/*
/ vettore test_int lungo N.test con valori int $\in [0,4096]$ casuali*/*

Data: train_int $\in [0, 4096]$
Data: test_int $\in [0, 4096]$
/ vettore train_binario lungo N.train con valori 12 bit di train_int*/*
/ vettore test_binario lungo N.test con valori 12 bit di test_int*/*

Output: train_binario, train_label, test_binario, test_label
inizializzazione casuale;
for Tutti i campioni di train **do**
| assegna il campione i al gruppo_train [train_int[campione i] % 16];
end
for Tutti i campioni di test **do**
| assegna il campione i al gruppo_test [test_int[campione i] % 16];
end
for Tutti i gruppi di test e di train **do**
| Assegna un'etichetta casuale ad ognuno dei gruppi;
end

Tabella 3.2: Caratteristiche IB Dataset.

IB Dataset					
Training Samples	Test Samples	N. Classi	Dimensione	μ	σ
3096	1000	2	12	0.504	0.499

3.1.3 Dataset con una Struttura

Si è inoltre generato un dataset, a cui ad ogni valore viene associata un'etichetta con una certa probabilità, in base al valore assunto dai primi due bit del dato in ingresso. Più precisamente, quando uno dei due primi bit è unitario, si associa al dato in ingresso l'etichetta 1 con probabilità $p=0.9$, mentre si associa l'etichetta 0 con una probabilità $p=0.1$. Al contrario se i primi due bit sono entrambi a zero, si associa al dato in ingresso l'etichetta 0 con probabilità $p=0.9$, mentre si associa l'etichetta 1 con una probabilità $p=0.1$.

Algoritmo 4: Algoritmo creazione Struttura Dataset

```

Input: N. train samples, N. test samples
/* vettore ix_train lungo N.train con valori int ∈[0,4096] casuale*/
/* vettore ix_test lungo N.test con valori int ∈[0,4096] casuale*/
Data: ix_train , ix_test
/* vettore train_binario lungo N.train con valori 12 bit di ix_train */
/* vettore test_binario lungo N.test con valori 12 bit di ix_test */
Output: train_binario, train_label, test_binario, test_label
inizializzazione casuale;
for Tutti i campioni train do
    if uno dei due primi bit di train data =='1' then
        /* Assegna train l'etichettà '1' con probabilità p=0.9, '0' con p=0.1
        */
        Assegna l'etichettà al campione train i;
    else
        /* Assegna train l'etichettà '1' con probabilità p=0.1, '0' con p=0.9
        */
        Assegna l'etichettà al campione train i;
    end
end
for Tutti i campioni test do
    if uno dei due primi bit di test data =='1' then
        /* Assegna test l'etichettà '1' con probabilità p=0.9, '0' con p=0.1 */
        Assegna l'etichettà al campione test i;
    else
        /* Assegna test l'etichettà '1' con probabilità p=0.1, '0' con p=0.9 */
        Assegna l'etichettà al campione test i;
    end
end

```

Tabella 3.3: Caratteristiche Struttura Dataset.

Struttura Dataset					
Training Samples	Test Samples	N. Classi	Dimensione	μ	σ
3096	1000	2	12	0.501	0.499

3.1.4 Dataset MNIST

Il dataset del MNIST consiste in diverse immagini rappresentanti numeri scritti a mano, in bassa risoluzione, in bianco e nero, divisi in 10 classi differenti, mutualmente esclusive [28].

Tabella 3.4: Caratteristiche MNIST Dataset.

MNIST Dataset					
Training Samples	Test Samples	N. Classi	Dimensione	μ	σ
60000	10000	10	28X28	0.13	0.31

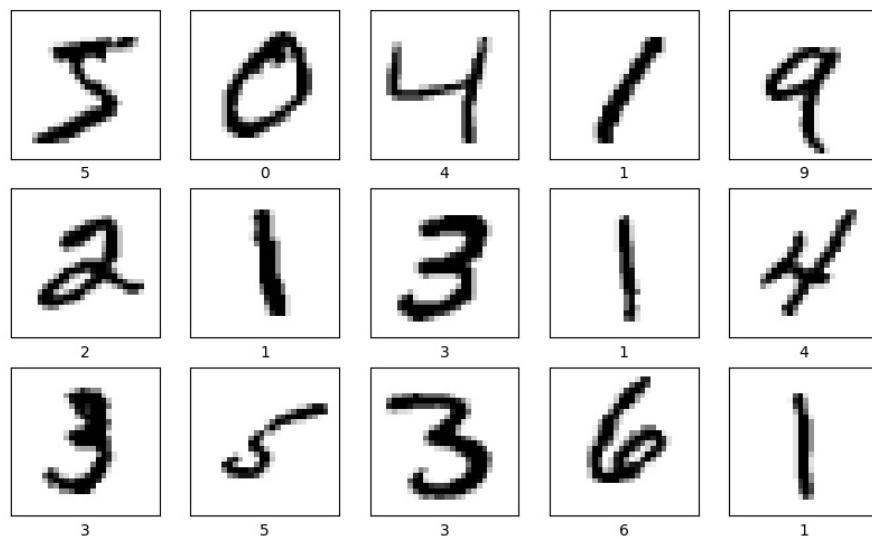


Figura 3.1: Rappresentazione di un sample dei primi 15 elementi del dataset MNIST con le rispettive etichette.

Il dataset *MNIST* risulta essere già prenormalizzato.

3.1.5 Dataset CIFAR10

Nel caso del CIFAR10, si è scelto di estrarre solamente la componente rossa dalle immagini per velocizzare l'apprendimento, non si ritiene che questa semplificazione possa falsare le varie prove. Il seguente dataset consiste in diverse immagini di animali ed oggetti in bassa risoluzione, mutualmente esclusive, colorate e divise in 10 classi differenti [26].

Tabella 3.5: Caratteristiche CIFAR Dataset.

CIFAR Dataset					
Training Samples	Test Samples	N. Classi	Dimensione	μ	σ
50000	10000	10	32X32	0.491	0.247



Figura 3.2: Rappresentazione di un sample dei primi 15 elementi del dataset CIFAR10 con le rispettive etichette.

3.2 Simulazioni

Una prima architettura sviluppata di rete è stata quella di Figura 3.3, essa consiste in una rete *Fully connected, Feedforward*, il cui numero di neuroni è presente in tabella 3.6:

Tabella 3.6: Numero di neuroni per layer della prima architettura realizzata.

Numero neuroni per Layer						
Input	Hidden 1	Hidden 2	Hidden 3	Hidden 4	Hidden 5	Output
12	10	7	5	4	3	2

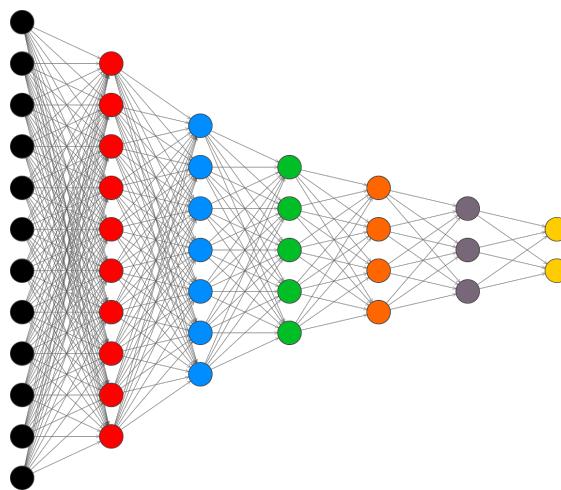


Figura 3.3: Esempio di architettura utilizzata, rete *Fully Connected* 12-10-7-5-4-3-2.

Per poter capire meglio la dinamica della traiettoria dell'informazione all'interno di una DNN, verranno costruite ulteriori architetture di rete, scegliendo funzioni di attivazione differenti per gli hidden layer, mentre la funzione di attivazione dell'output layer sarà fissata pari a quella di una *sigmoide* o *softmax* (in base al numero delle classi del dataset in ingresso). Verrà utilizzata come funzione costo *l'entropia mista categorica*, e per inizializzare la matrice W dei pesi della rete si userà l'algoritmo di *Glorot Uniform*, mentre il vettore di bias b della rete è inizializzato a zero. Verranno inoltre effettuate delle simulazioni dopo aver modificato la lunghezza e i parametri della rete, per vedere come variano le traiettorie dell'informazione al variare di questi.

Le simulazioni seguenti sono state realizzate utilizzando la versione 2.0.0 di *tensorflow*, in appendice viene riportato il file *requirements.txt* (Appendice A.2) contenente tutti i vari pacchetti utilizzati e le rispettive versioni, in maniera tale da permettere di creare un *ambiente virtuale* con le stesse versioni dei pacchetti utilizzati durante la stesura della seguente tesi.

Per creare la rete neurale, si è sfruttata principalmente la libreria fornita da *tensorflow*, andando a riscrivere il codice della funzione *fit* (la quale si occupa di fare l'addestramento della rete, calcolare la funzione costo, fare il calcolo del gradiente e l'aggiornamento di W e b oltre che ad occuparsi dell'attività per il debug), in modo

da avere un accesso a più basso livello delle variabili durante l’addestramento della rete ed un maggior controllo.

Si è inoltre sviluppata la funzione per il calcolo dell’informazione mutua e la si è comparata con quelle già presenti in rete, ottenendo risultati simili.

Si vuole sottolineare che durante l’addestramento non si è utilizzata nessuna tecnica ulteriore oltre a quella del [SGD](#). Vengono inoltre salvati durante l’addestramento della rete lo storico di tutti dati più rilevanti, per trovare le varie traiettorie all’interno del piano dell’informazione in maniera “offline”. Questo è stato fatto in maniera tale da effettuare diversi esperimenti sulla stima dell’informazione mutua, la quale risulta essere la parte più problematica, senza dover ripetere l’addestramento ogni volta.

Si ricorda inoltre che le varie distribuzioni di probabilità vengono calcolate, contando il numero di campioni quantizzati che cadono in un particolare bin, e dividendolo per il numero di bin totali.

Durante la fase di addestramento vengono spesso utilizzati, come dati di ingresso, dei campioni prelevati da un insieme X infinito, si pensi per esempio a tutte le possibili combinazioni che ci possono essere delle foto di un albero. In questo caso la distribuzione congiunta è sconosciuta $P(X,Y)$ e la si può stimare attraverso dei campioni prelevati dall’insieme, o facendo delle assunzioni sul modello.

Nel caso considerato per calcolare le distribuzioni di probabilità e successivamente la mutua informazione, viene utilizzato un binning statico. Non si è ritenuto necessario mediare i risultati su 50 esperimenti (con inizializzazione degli stati differenti come è stato fatto da Tishby [45]) per velocizzare la fase di sperimentazione, i risultati ottenuti sono comunque tutti simili.

3.3 Simulazione con IB Dataset

La prima simulazione effettuata è stata fatta utilizzando il dataset IB ed usando la funzione di attivazione *tanh* per gli hidden layer, per poter creare tale dataset è stato utilizzato l’algoritmo presente in Sezione 3.1.2, in modo che sia composto in ingresso da dei pattern simmetrici, mentre la distribuzione delle etichette in uscita è uniforme, in questo modo la mutua informazione vale:

$$\begin{aligned} I(X, Y) &= H(Y) - H(Y|X) = \\ &= - \sum_{i=0}^1 p(Y=i) \log_2 p(Y=i) - \sum_{i=1}^{4096} p(X=X_i) H(Y|X=X_i) \\ &\approx -0.5 \log_2(0.5) - 0.5 \log_2(0.5) - \sum_1^{4096} \frac{1}{4096} [-\log_2(1)] \approx 1 \end{aligned}$$

Possiamo dedurre quindi utilizzando la [DPI](#), che tale valore è il limite massimo della mutua informazione tra X e T in questo dataset.

Sia X l'ingresso, Y l'etichetta desiderata, ed \hat{Y} l'uscita della rete stimata, T_i l'uscita degli hidden layer, allora:

$$\begin{aligned} I(X, T_i) &= H(T_i) - H(T_i|X) = - \sum_{T_i} p(T_i) \log_2 p(T_i) - 0 \\ &= - \sum_{T_i} p(T_i) \log_2 p(T_i) \end{aligned}$$

$$\begin{aligned} I(Y, T_i) &= H(T_i) - H(T_i|Y) = - \sum_{T_i} p(T_i) \log_2 p(T_i) - \sum_{i \in (0,1)} p(Y = i) H(T_i|Y = i) \\ &= - \sum_{T_i} p(T_i) \log_2 p(T_i) + \sum_{i \in (0,1)} p(Y = i) \sum_{T_i, Y=i} p(T_i) \log_2 p(T_i) \end{aligned}$$

Dove si è utilizzato il fatto che T_i è deterministica in X , e questo comporta $H(T_i|X) = 0$. Si ricorda che T_i è l'activity del generico layer i , digitalizzato attraverso il binning, X è già binario e non serve digitalizzarlo essendo $p(X)$ distribuito in maniera binaria.

Come si può notare in Figura 3.4 ci sono due fasi distinte durante la fase di addestramento, in cui è stata inserita una croce rossa per gli ultimi due hidden layer per evidenziarne il punto di separazione. Nella prima fase l'informazione aumenta, nella seconda fase viene compressa. Questi risultati sono in accordo con quanto affermato da Tishby in [45], dove queste due fasi vengono chiamate **fase di apprendimento** e **fase di compressione** rispettivamente per la prima e seconda fase. Tali andature sono presenti pure in altri dataset come si vedrà in seguito (anche non simmetrici come è il dataset IB), si può supporre quindi essere una proprietà generale.

All'inizio della fase di addestramento, essendo stata la rete inizializzata in modo casuale, i layer più profondi non riescono a preservare informazioni rilevanti sull'ingresso. Questo comporta una diminuzione di $I(X, T)$ ed $I(T, Y)$ man mano che ci si muove sempre più in profondità all'interno della rete (Figura 3.4), rispettando così la DPI. Un'altra osservazione da fare è che un'andatura simile la si trova anche nel caso in cui si utilizzasse un'inizializzazione della rete differente, indicando che reti diverse tendono sempre a convergere verso punti vicini nel piano dell'informazione.

In queste sperimentazioni, si può notare come la maggior parte degli epoch vengano spesi a comprimere la rappresentazione interna degli hidden layer, e questa fase compare senza l'utilizzo di tecniche di regolarizzazione durante la fase di addestramento.

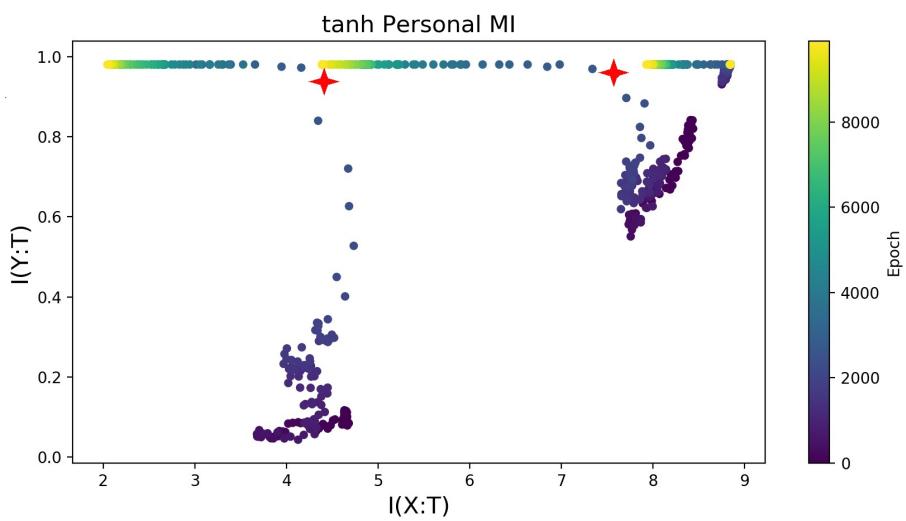


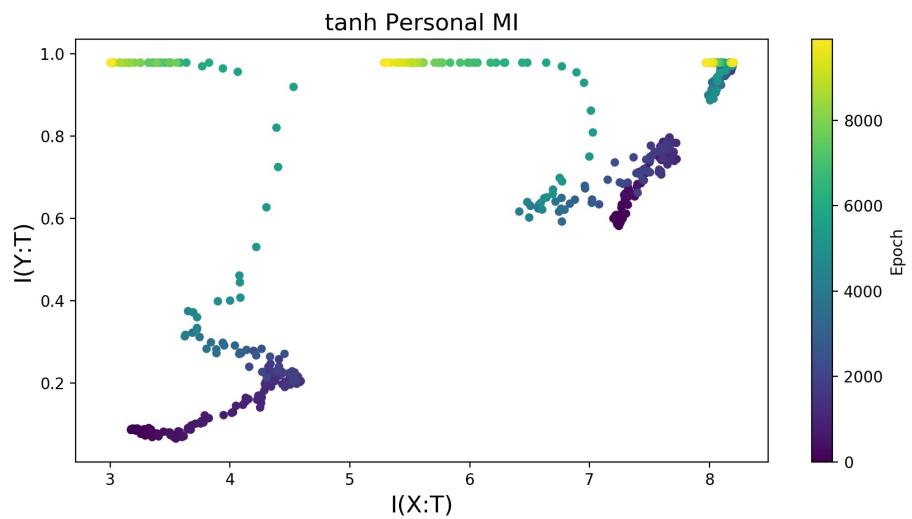
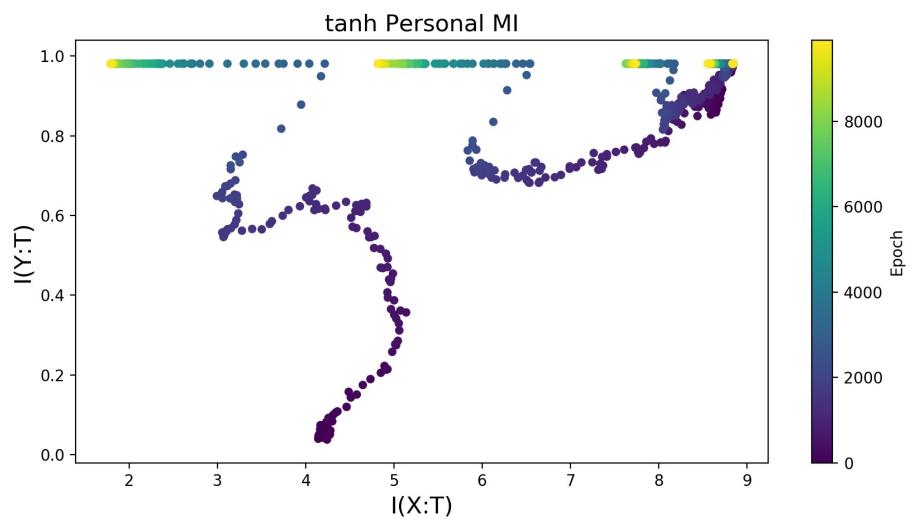
Figura 3.4: Traiettorie della mutua informazione per ogni layer T_i . Le traiettorie dei layer più vicini all’ingresso si trovano sul lato destro della figura, mentre le traiettorie dei layer più profondi si trovano sulla sinistra. È stato evidenziato mediante una croce rossa, per gli ultimi hidden layer, il punto in cui si passa dalla fase di apprendimento a quella di compressione. Learning rate= 0.001, batch size = 512, funzione di attivazione=tanh, n.bins=30.

Secondo quanto affermato da Tishby, la parte più importante dell'addestramento di una rete è la parte in cui la rete scarta le informazioni poco rilevanti dei dati in ingresso, realizzando così la compressione.

Durante la prima fase sia $I(X, T_i)$ che $I(Y, T_i)$ aumentano, nella fase di compressione invece gli hidden layer T_i iniziano ad eliminare le informazioni poco rilevanti per poter determinare l'etichetta Y, in questo caso $I(X, T_i)$ diminuisce, mentre $I(Y, T_i)$ rimane alto.

Un'altra osservazione da fare, è che durante l'addestramento, i primi layer (i quali si trovano sul lato destro nel piano dell'informazione) non acquisiscono informazioni sull'etichetta in uscita Y, e comprimono in maniera poco significativa i dati in ingresso, se comparati ai layer più profondi (Figura 3.5b). Acquisire informazioni sull'etichetta in uscita indica una migliore memorizzazione del dataset. Non si è ancora in grado di spiegare con certezza perché hidden layer diversi convergono in punti differenti. Tishby ha supposto che questo accade in quanto layer diversi hanno livelli di rumore differenti del gradiente durante la fase di compressione (Figura 3.12b), finendo in punti diversi di massima entropia condizionale $H(X|T_i)$, ma tale affermazione sarà contraddetta successivamente nel presente elaborato.

Nel caso in cui si mostri il grafico della mutua informazione, quando la rete viene addestrata su una percentuale del train dataset, si ottengono i risultati di Figura 3.5, dove si ha un comportamento simile al caso precedente. La mutua informazione del test dataset mentre si allena la rete con una data percentuale del train dataset, è presente invece in Appendice alla Figura A.9. In questo ultimo caso si nota come durante la fase di compressione si perdono delle informazioni rilevanti sulle etichette in uscita, quando viene utilizzato un train dataset più piccolo, causando una generalizzazione minore. All'aumentare della dimensione del train dataset l'informazione sulle etichette in uscita aumenta, tale andamento sembra essere collegato al comportamento della rete in caso di overfitting, evitabile per esempio mediante metodi come l'early stopping. Capire cosa determina i punti di convergenza dei layer nel piano, per diverse dimensioni del dataset in ingresso è ancora motivo di ricerca.

(a) $IB_{10\%}$ informazione mutua.(b) $IB_{30\%}$ informazione mutua.

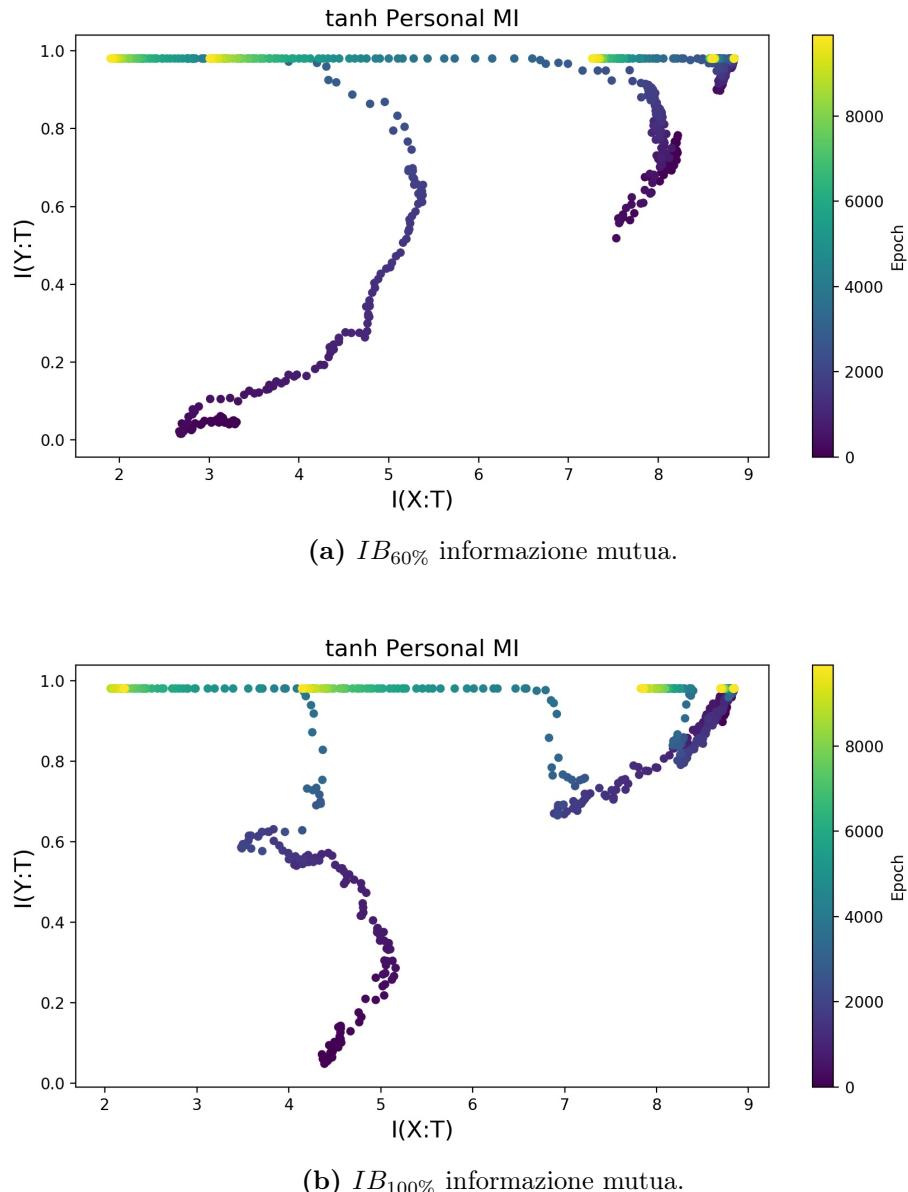


Figura 3.5: Grafici relativi alla variazione in percentuale del train dataset. Learning rate=0.001, batch size=1, funzione di attivazione=tanh, n.bins=30.

Nel caso in cui venisse utilizzata una funzione di attivazione per gli hidden layer, con andamento non a saturazione, per esempio *ReLU* o *LeakyReLU*, utilizzando sempre una funzione a *Sigmoide* per il layer d'uscita, si ha un comportamento diverso rispetto al caso precedente e poco spiegabile (Figura 3.6). In accordo con quanto affermato da Saxe et al., possiamo quindi dopo una prima analisi già giungere alla conclusione, che la fase di compressione dell'informazione deriva dalle due curve di saturazione presenti nella funzione *tanh* (Figura 3.7) [33]. Questa conclusione sarà approfondita ulteriormente nelle Conclusioni.

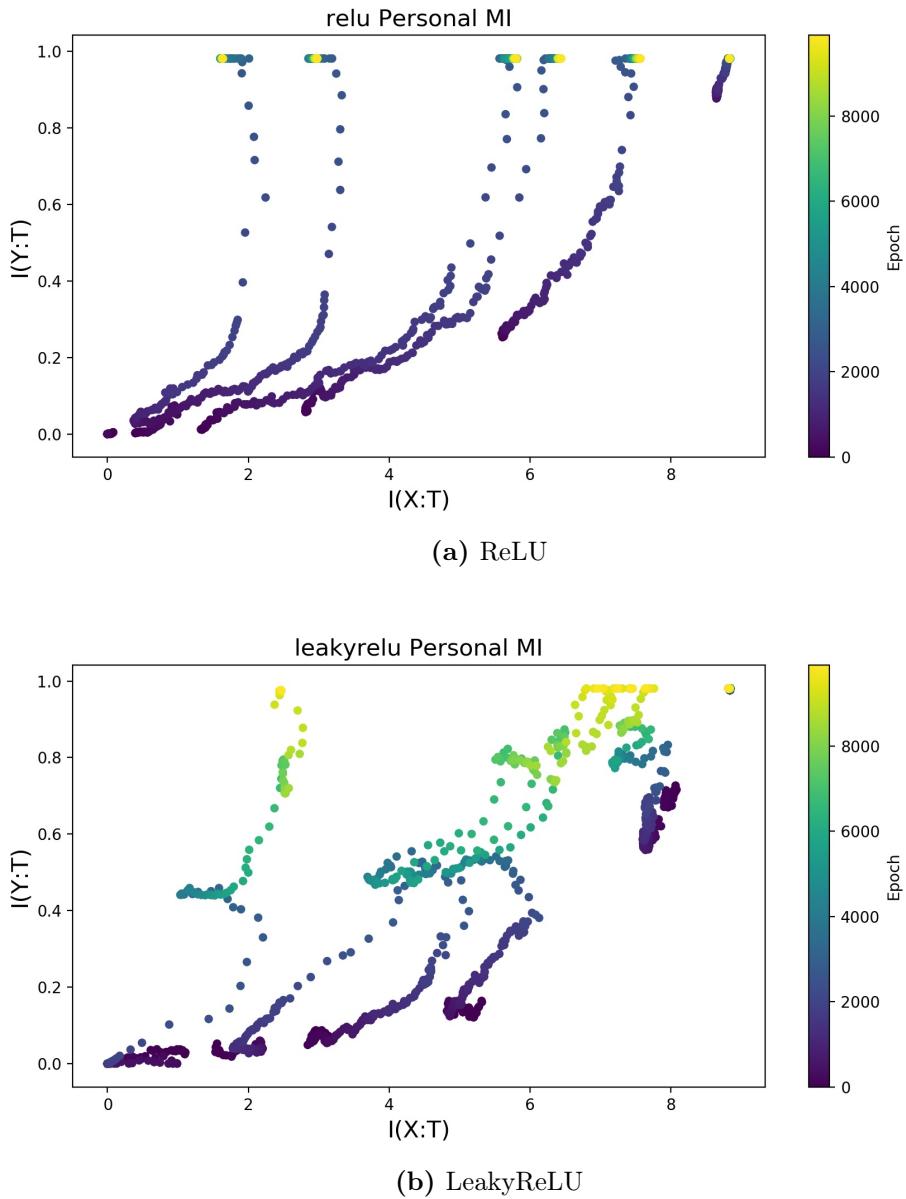


Figura 3.6: Grafici della mutua informazione utilizzando funzioni di attivazioni diverse dalla tanh. Learning rate=0.001, batch size=64, n.bins=30.

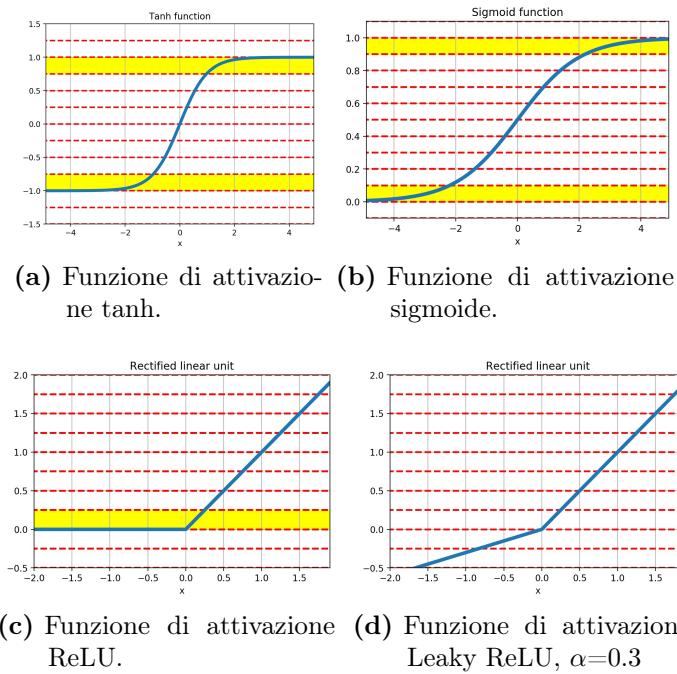


Figura 3.7: Esempio di come il valore dell’uscita del neurone (l’activity), viene divisa in bin equamente spaziati. Le zone evidenziate sono i valori quantizzati nei quali l’activity tende a convergere durante l’addestramento. Come si può vedere la funzione ReLU e la LeakyReLU non limitano automaticamente l’uscita del neurone.

Un’ulteriore prova di quanto riferito da Saxe et al., la si può vedere dal fatto che l’andamento medio dell’uscita digitalizzata dei neuroni (activity), degli hidden layer, a causa della funzione di saturazione *tanh*, tende a convergere verso i valori ± 1 , (Figura 3.8). Inoltre si vuole anche far notare una proprietà particolare delle reti DNN quando vengono addestrate mediante l’algoritmo di GD: la velocità di aggiornamento dei pesi è in generale maggiore sui layer più profondi rispetto a quelli vicini all’ingresso, facendo così convergere le uscite dei neuroni con velocità diverse, questo vale anche nel caso in cui si utilizzasse lo stesso numero di neuroni per ogni layer [31].

Ricapitolando, sembrerebbe dopo questa prima analisi che la compressione nasce a causa della forma a *doppia saturazione* presente nella funzione tanh e nella sigmoide. Per la tanh si è diviso l'intervallo $[-1,1]$ con un numero di bin pari a 30. Per la ReLU l'intervallo da considerare durante il binning vale $[0, L_{max}]$, dove L_{max} è il valore massimo dell'activity durante l'addestramento. Per la LeakyReLU si divide l'intervallo $[L_{min}, L_{max}]$, dove L_{min} è il valore minimo dell'activity durante l'addestramento mentre L_{max} è lo stesso di quello definito per la ReLU. Dopo aver digitalizzato l'activity e calcolato le varie distribuzioni di probabilità, se ne è calcolata la traiettoria all'interno del piano dell'informazione durante l'intera fase di addestramento. Per quanto riguarda le funzioni ReLU e LeakyReLU, come si può vedere in Figura 1.4, le due funzioni non fanno tendere la distribuzione ad essere una distribuzione binaria, e questo potrebbe essere il motivo della mancanza di compressione nel piano dell'informazione.

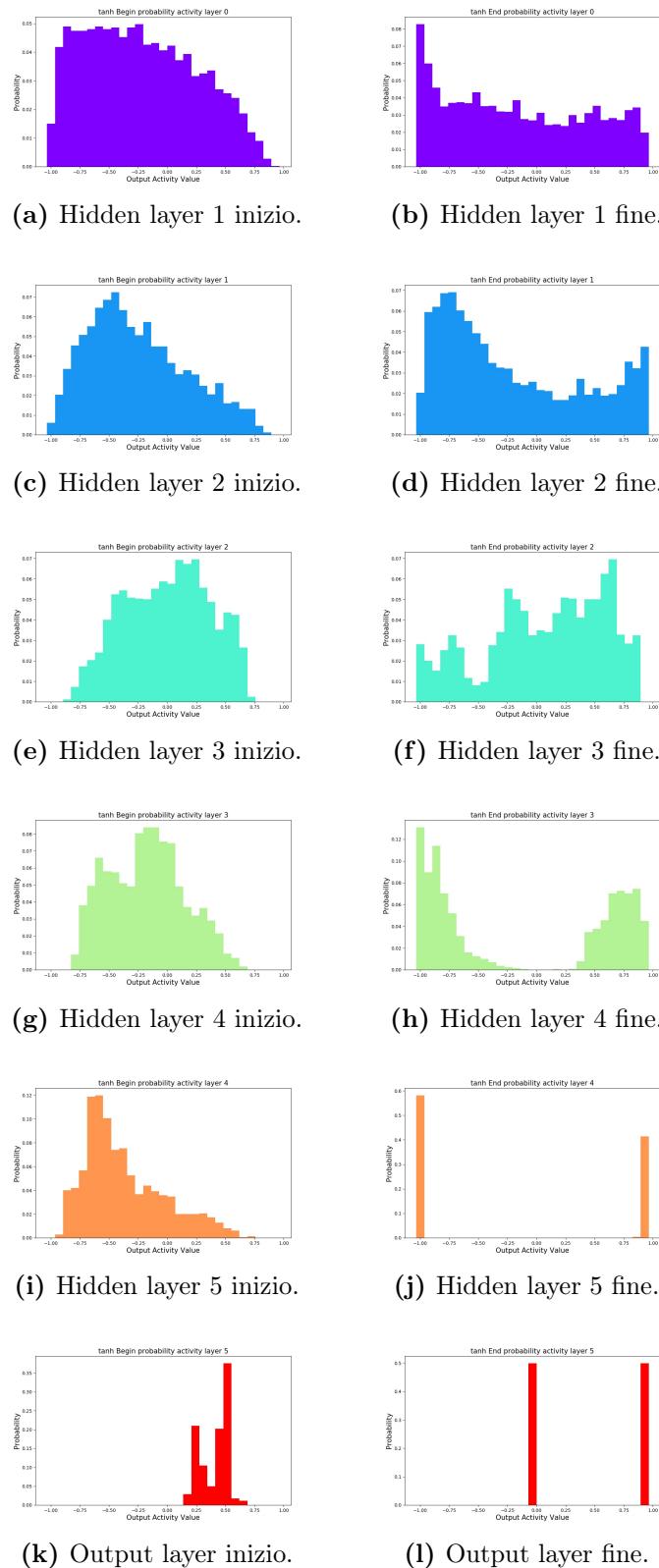


Figura 3.8: Andamento medio activity digitalizzata degli hidden layer durante l’addestramento. La funzione \tanh fà convergere le uscite verso ± 1 , la funzione sigmoide verso 0, 1. Learning rate=0.1, batch size=512, n.bins=30.

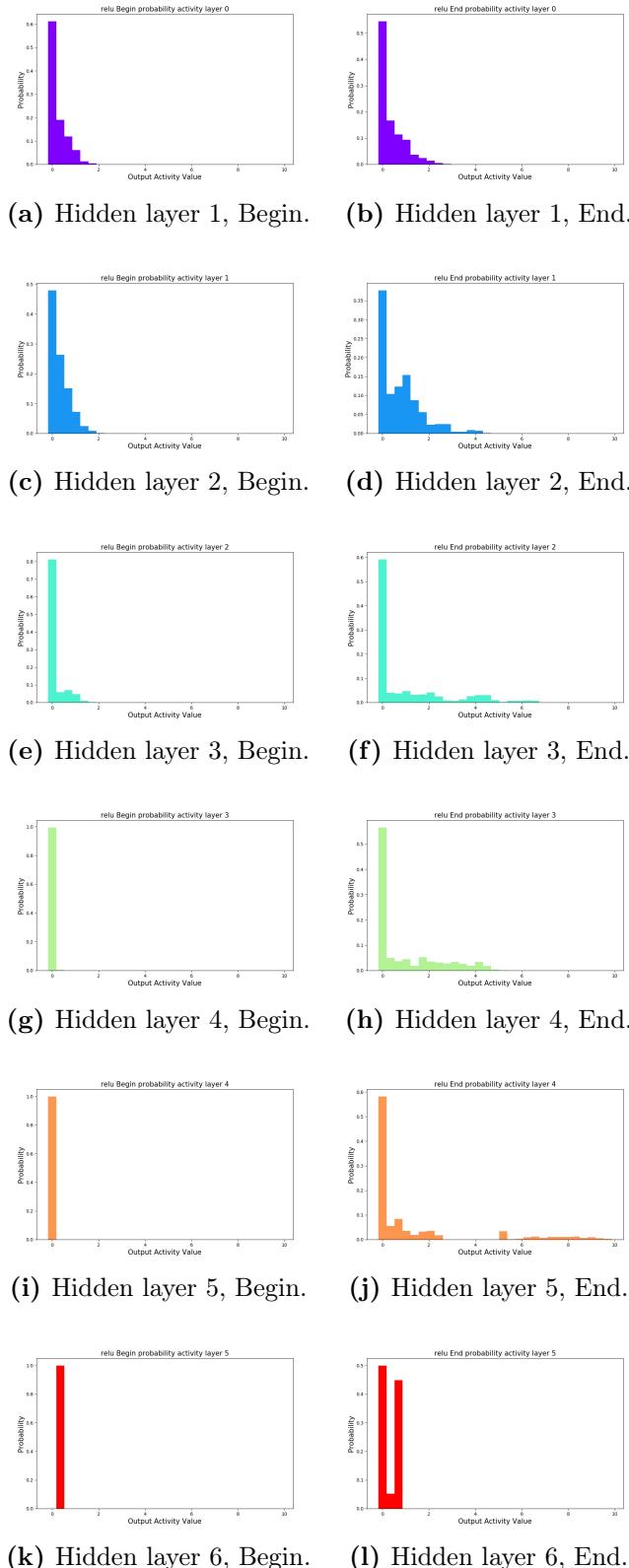


Figura 3.9: Andamento medio activity digitalizzata degli hidden layer durante l’addestramento, utilizzando la funzione *ReLU*. Learning rate=0.1, batch size=512, n.bins=30.

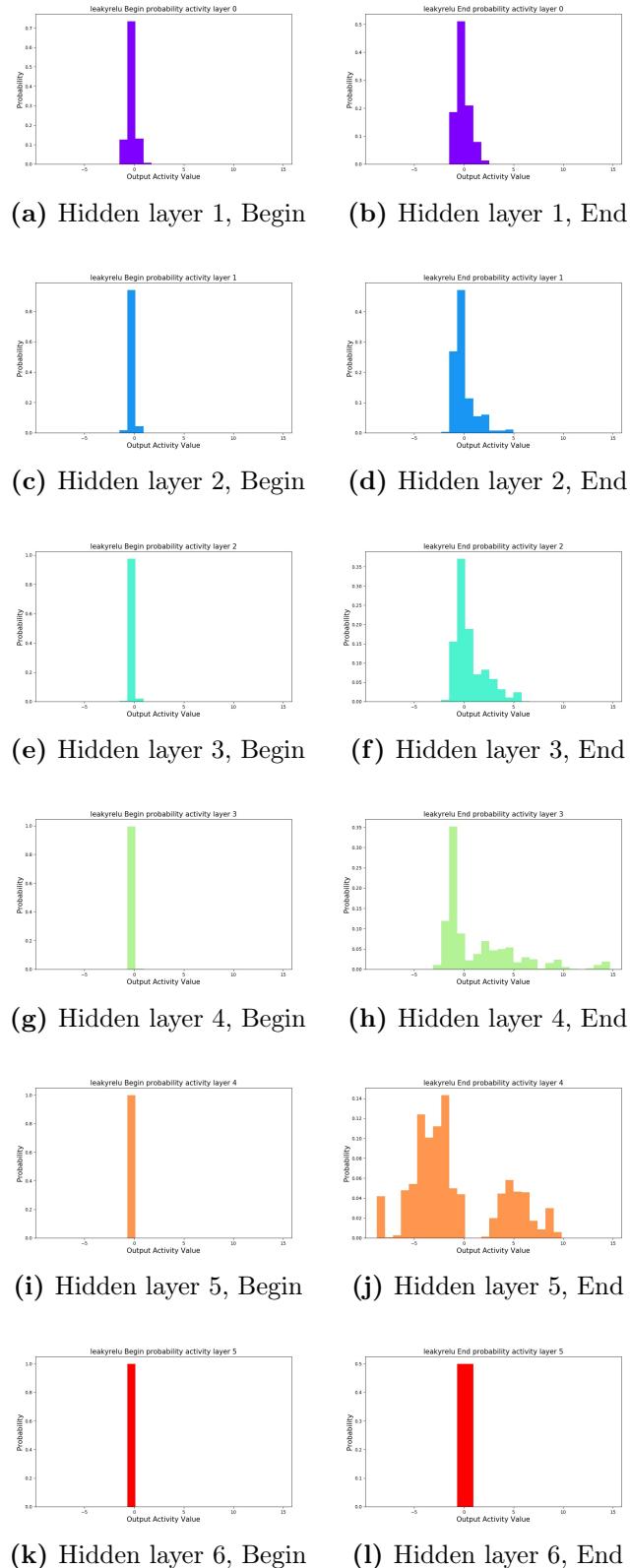
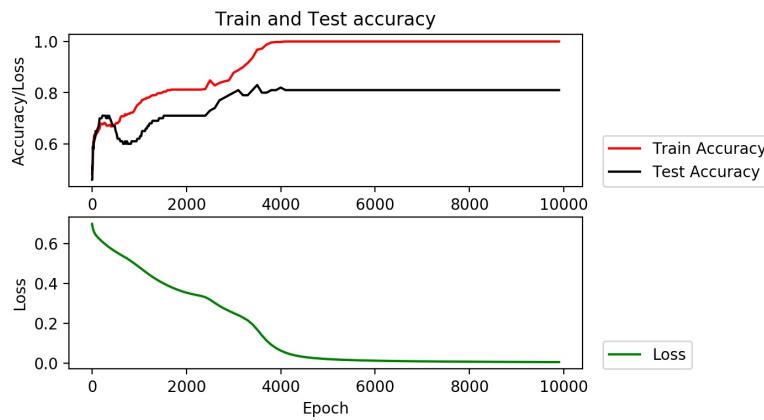
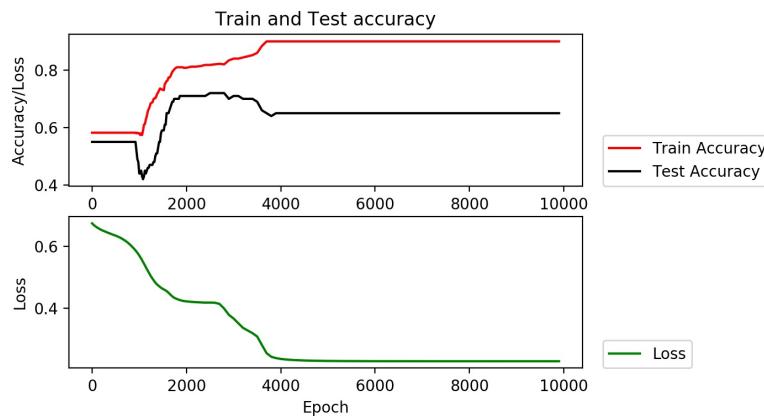


Figura 3.10: Andamento medio activity digitalizzata degli hidden layer durante l’addestramento, utilizzando la funzione *LeakyReLU*. Learning rate=0.1, batch size=512, n.bins=30.

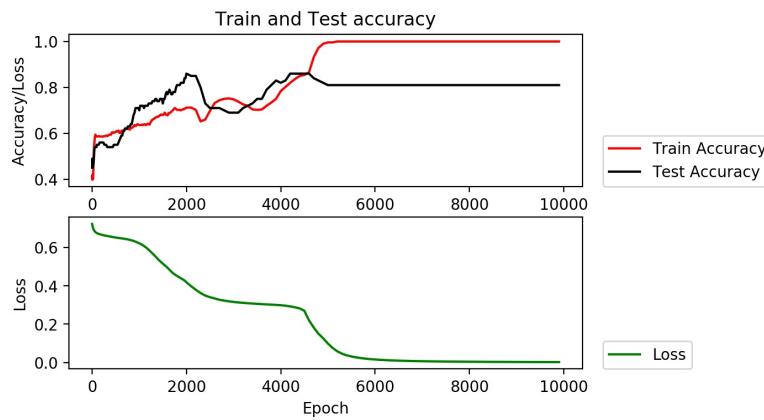
Come è naturale aspettarsi, la funzione costo diminuisce durante la fase di addestramento per tutte le funzioni di attivazione utilizzate (Figura 3.11):



(a) tanh funzione costo



(b) ReLU funzione costo



(c) LeakyReLU funzione costo

Figura 3.11: Andamento funzione costo con diverse funzioni di attivazione per gli hidden layer. Learning rate=0.001, batch size=1.

Aumentando il valore del modulo dei pesi, aumenta anche il modulo di $Wx + b$, andando a far convergere la funzione $f(Wx + b)$ verso ± 1 in virtù della caratteristica a doppia saturazione della \tanh , in questo modo \hat{T} tende ad avere una distribuzione binaria proprio come Y , di conseguenza $I(Y, T)$ si avvicina ad:

$$\begin{aligned} H(Y) &= - \sum_{i=0}^1 p(Y = i) \log_2 p(Y = i) \\ &= -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 1 \end{aligned}$$

poiché $I(T, Y) \leq H(Y)$ (l'informazione che T contiene su Y non può essere più di Y stessa) si ha anche che $I(T, Y)$ si avvicina ad 1, mentre durante la fase di compressione $I(X, T)$ diminuisce.

3.3.1 Gradiente della funzione costo

Per comprendere meglio il perché nascono le fasi di apprendimento (o chiamata anche fase di Empirical Error Minimization (**ERM**)) e di compressione, si esamina l'andamento della media normalizzata e della deviazione standard del gradiente stocastico, calcolato rispetto la matrice dei pesi W ed il vettore di bias b , durante l'addestramento della rete. Come si può notare dalla Figura 3.12 c'è una transizione tra due fasi distinte. Nella prima, definita da Tishby **fase di deriva**, la media del gradiente è molto più grande della sua fluttuazione stocastica, ottenendo un grande **SNR** che si suppone porti l'informazione mutua rispetto l'etichetta Y ad aumentare, e a far diminuire l'errore empirico molto velocemente. Nella seconda fase invece, la media del gradiente è più piccola rispetto alla fluttuazione della deviazione standard, e ciò comporta un'aggiunta di rumore casuale al processo di aggiornamento dei pesi. Il gradiente in questa fase si comporta come un processo di diffusione spiegabile tramite l'equazione di Focker-Planck, avendo così un basso **SNR**. Quest'ultima fase viene chiamata **fase di diffusione** e si suppone che porti ad una lenta fase di compressione dell'informazione rispetto l'ingresso X , ed un aumento dell'entropia condizionale $H(X|T_i)$ per ogni layer T_i .

Queste due fasi sono attese, in quanto mentre l'errore si trova in prossimità dello zero (Figura 3.12c), il **SGD** è dominato da dalle fluttuazioni, che lo portano a giungere in una fase di **rilassamento stocastico** non appena l'accuratezza è arrivata ad un valore unitario (Figura 3.12b).

Tishby suppone che questa variazione del **SNR** possa spiegare le fasi di *apprendimento* e quella di *compressione* trovate nell'analisi precedente, questo perché la transizione da una fase all'altra del **SNR**, compare in prossimità dell'istante temporale di separazione che è presente anche tra le due fasi nel piano dell'informazione. Tale affermazione sarà però analizzata in maggior dettaglio in seguito.

Dalla Figura 3.12b si può inoltre vedere che ogni layer ha un livello di rumore differente.

Poiché la rete tende a convergere negli stessi punti nel caso in cui si utilizzasse una inizializzazione della rete differente, si può affermare che ci sono N reti diverse con le stesse performance ottime sulla base dell'Equazione (2.21), la quale indica un'invarianza dell'informazione a fronte di trasformazioni invertibili. Si può affermare che ottimizzare il processo di apprendimento basandosi solamente sui singoli neuroni, anziché vedere l'intero layer nella sua completezza, è inutile. Le due fasi del gradiente sono state notate anche in un'altra ricerca, venendo definite fase *transitoria* e *stocastica* [12, 30].

Per trovare i grafici seguenti, per ogni layer si è calcolato media e deviazione standard del gradiente, rispetto la matrice dei pesi W ed i bias b , per poi normalizzare le grandezze attraverso la *norma di Frobenius*:

$$mean_l = \left\| \left\langle \frac{\partial E}{\partial W_l} \right\rangle \right\|_F \quad (3.1)$$

$$std_l = \left\| std \left(\frac{\partial E}{\partial W_l} \right) \right\|_F \quad (3.2)$$

$$b mean_l = \left\| \left\langle \frac{\partial E}{\partial b_l} \right\rangle \right\|_F \quad (3.3)$$

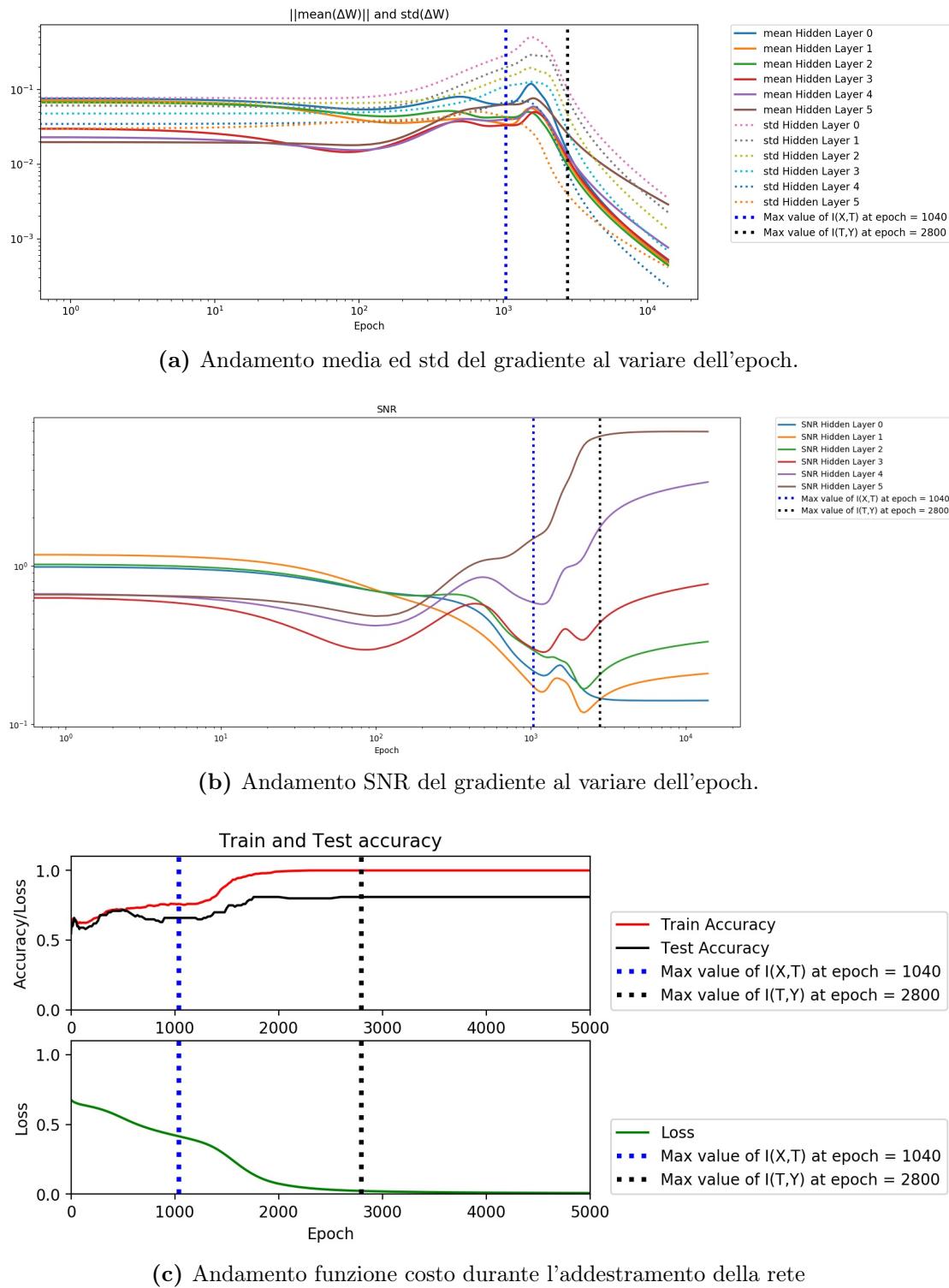
$$b std_l = \left\| std \left(\frac{\partial E}{\partial b_l} \right) \right\|_F \quad (3.4)$$

Il rapporto segnale rumore sarà invece:

$$SNR = 20 \log \left(\frac{mean_l}{std_l} \right) \quad (3.5)$$

$$SNR_b = 20 \log \left(\frac{b mean_l}{b std_l} \right) \quad (3.6)$$

Il grafico risulta essere leggermente diverso da quello di Tishby e Schwartz-Ziv, in quanto in quel caso si suppone sia stata utilizzata un'inizializzazione dei pesi particolare, che permette di amplificare la differenza tra le due fasi del SGD, ma non essendo stato specificato questo dettaglio non si ha modo di verificarlo [45].



All'incirca all'epoch 2000 della Figura 3.12a si può notare una sovraelongazione significativa del gradiente, tale andamento è sempre presente in tutte le simulazioni effettuate, e viene definito da Tishby ***Fall to flat minima point***.

Il valore della capacità del canale Gaussiano proposta da *Shannon* vale:

$$C_{canale} = \frac{1}{2} \log (1 + SNR) \quad (3.7)$$

E poiché a regime l'**SNR** raggiunge un valore costante per ogni layer, questo comporta un valore della capacità di canale degli hidden layer costante, tale valore di capacità è collegata al valore dell'informazione mutua. Valori costanti dell'**SNR** indicano che la rete è andata in convergenza in una configurazione che porta un flusso costante di informazioni rilevanti attraverso i layer dall'ingresso verso l'uscita.

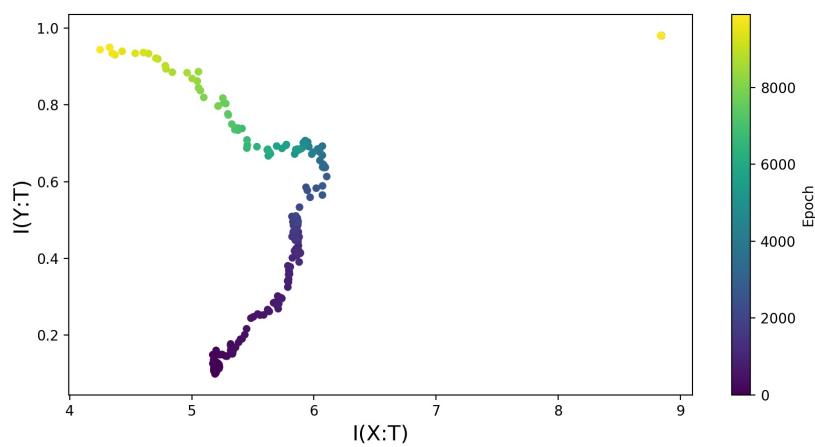
Secondo quanto riferito da Tishby, e come si può notare dalla Figura 3.12a, per come funziona l'algoritmo di Backpropagation, il rumore proveniente dagli ultimi hidden layer viene portato ed accumulato al primo layer, portando così la media e l'std del gradiente del primo layer ad un valore maggiore rispetto gli altri. Tale accumulo di rumore è responsabile della caduta del gradiente verso il suo punto di minimo.

3.3.2 Benefici hidden layers

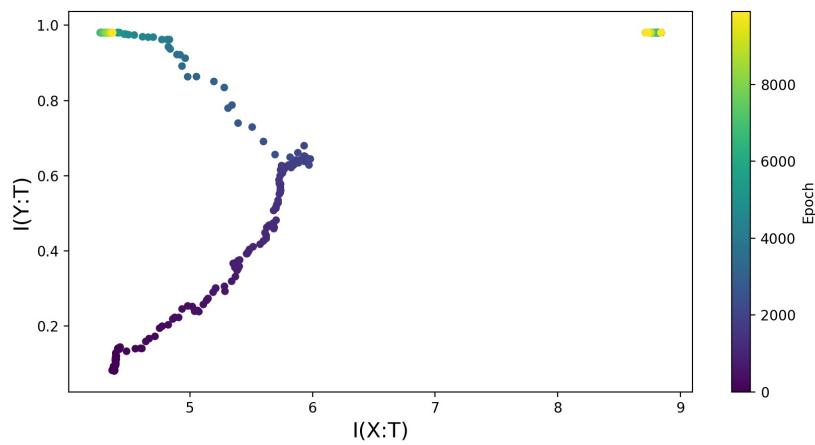
Cambiando la dimensione dell'architettura di rete, passando da un hidden layer a sei, si possono fare le seguenti osservazioni:

- Aggiungere hidden layer aiuta a memorizzare le etichette in minor tempo, per vederlo basta confrontare la Figura 3.13c con la Figura 3.13b. In questo caso la prima memorizza tutte le informazioni sulle etichette in uscita in 4600 epoch, mentre la seconda riesce a memorizzare tutte le etichette in 2600 epoch.
- Come afferma Tishby la compressione è più veloce quando questo processo inizia già nei layer precedenti, ma questo però lo si può notare solamente confrontando l'ultimo layer della Figura 3.13b con l'ultimo layer della 3.13c. In questo caso si ha una lunghezza differente del tratto in giallo, indicando una compressione più lunga nel primo caso rispetto al secondo, mentre non lo si nota nei restanti layer [45].
- Si può notare inoltre come in una rete più profonda (Figura 3.13b), si riesca a comprimere l'ingresso in maniera migliore rispetto a quando vengono utilizzati un numero minore di layer (Figura 3.13c). Raggiungendo nel primo caso, per l'ultimo layer, un bit di informazione rispetto l'ingresso, mentre nel secondo caso si raggiungono i due bit.
- Si nota inoltre che la fase di compressione e quella ERM sono più veloci per i layer più vicini all'ingresso, mentre i layer più profondi, prima imparano e dopo comprimono (Figura 3.13a e Figura 3.13b).

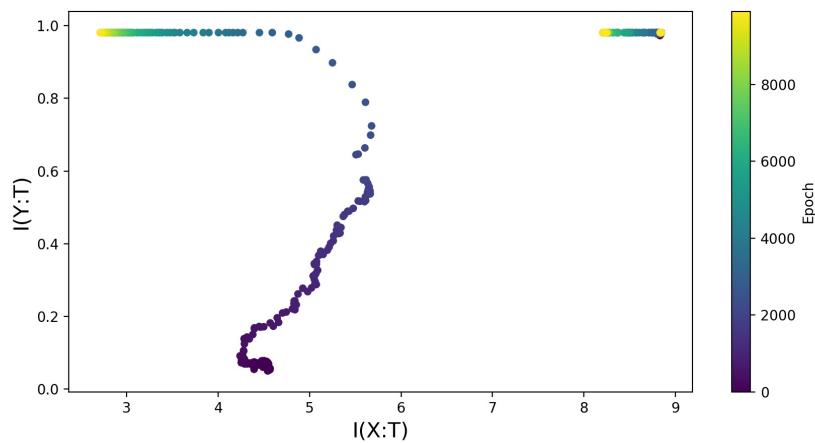
Il vantaggio derivante dall'aggiunta di hidden layer è un vantaggio di tipo computazionale, nel senso che si riduce il numero di epoch necessari per raggiungere una buona memorizzazione delle etichette in uscita, aumentando di conseguenza il tempo di rilassamento del processo stocastico, questa a discapito dell'aumento dei parametri utilizzati e della complessità della rete.



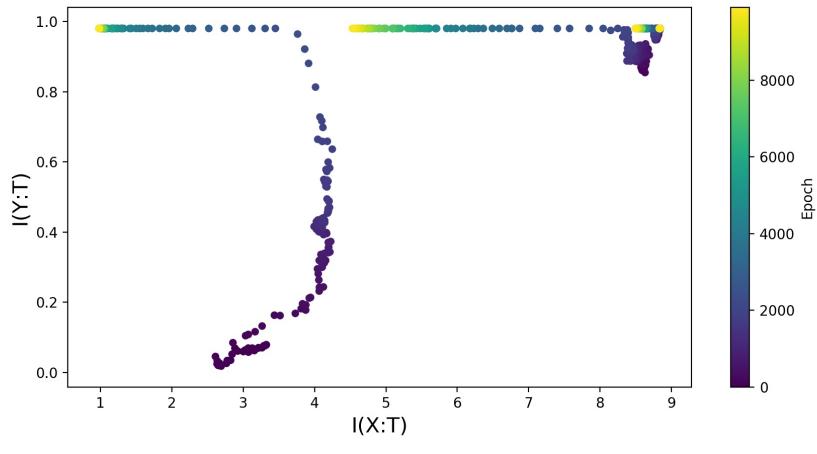
(a) Configurazione 1 Layer
Configurazione Neuroni: 10.



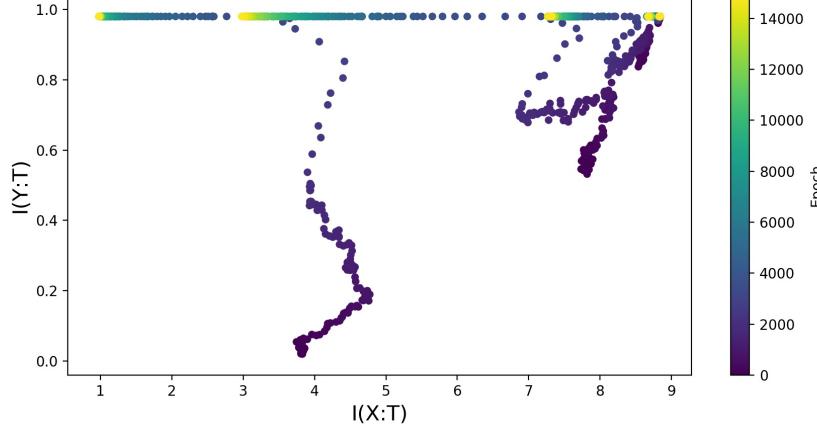
(b) Configurazione 2 Layers
Configurazione Neuroni: 10-7.



(c) Configurazione 3 Layers
Configurazione Neuroni: 10-7-5.



(a) Configurazione 4 Layers
Configurazione Neuroni: 10-7-5-4.

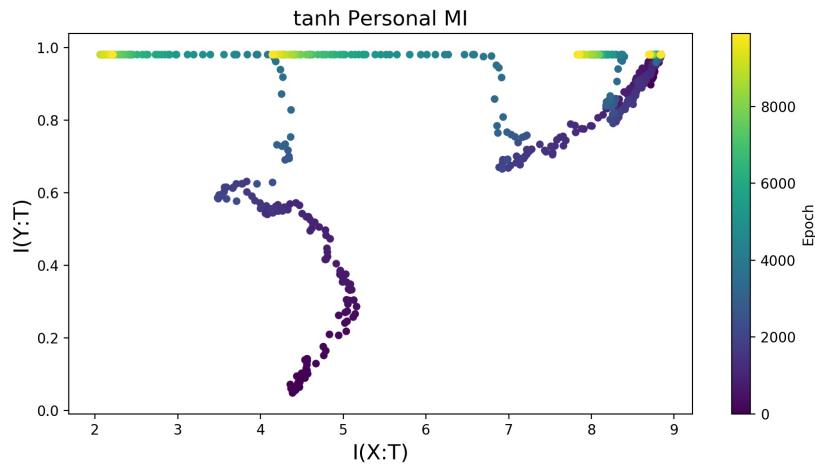


(b) Configurazione 4 Layers
Configurazione Neuroni: 10-7-5-4-3.

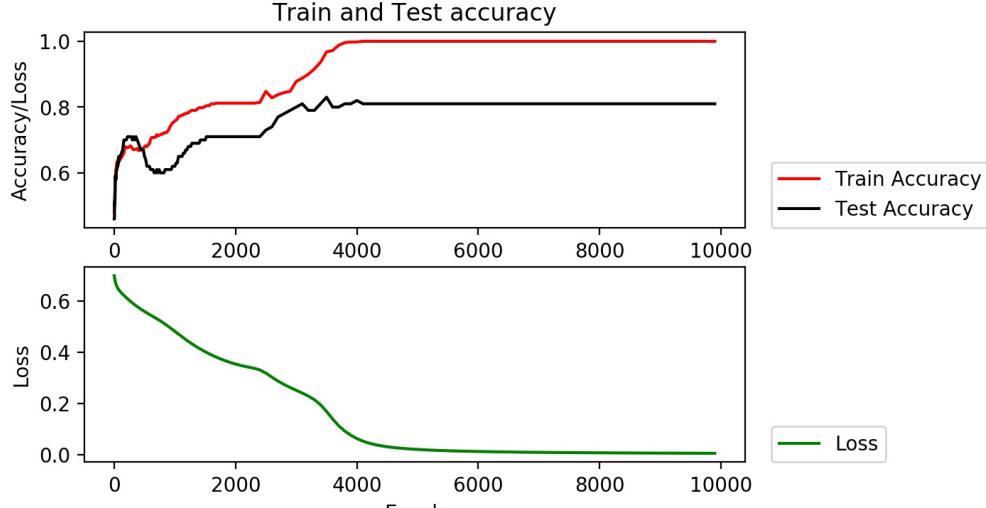
Figura 3.13: Andamento informazione al variare del numero di layer. Learning rate=0.001, batch size=512, n.bins=30.

3.3.3 Variazione dei parametri

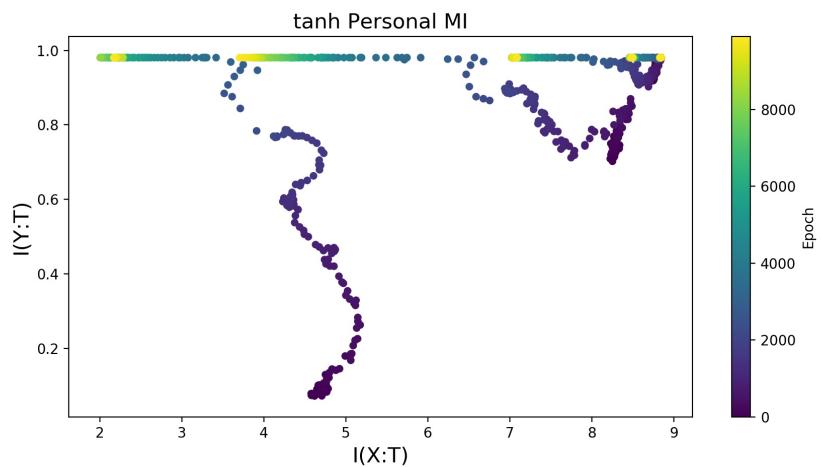
In questa sezione si andranno a variare i parametri della rete e a vedere come si modificano le traiettorie all'interno del piano dell'informazione. Come si può vedere dalle simulazioni di Figura 3.14, e di Figura 3.15, si conclude che il learning rate ed il batch size non modificano la traiettoria dell'informazione ma solo la velocità di convergenza:



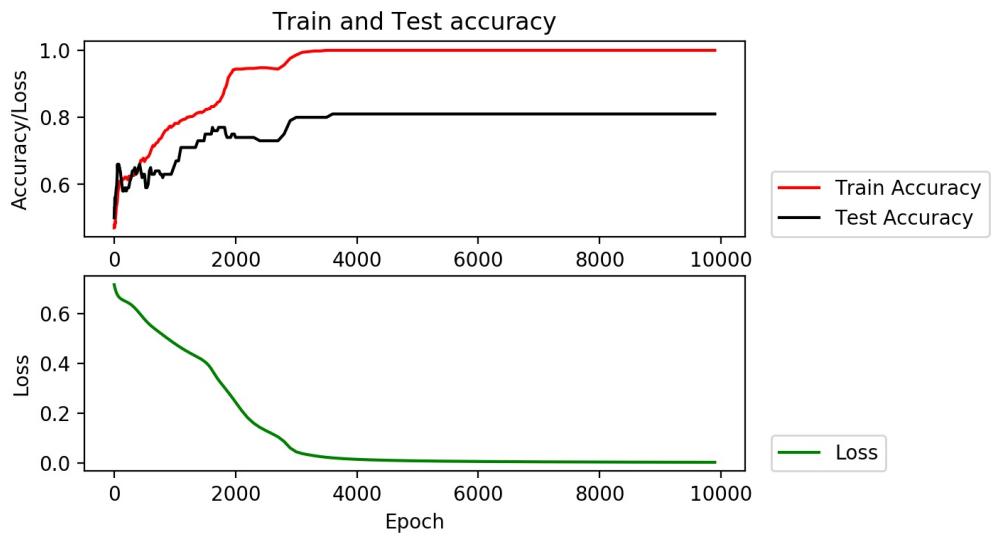
(a) Batch size = 1.

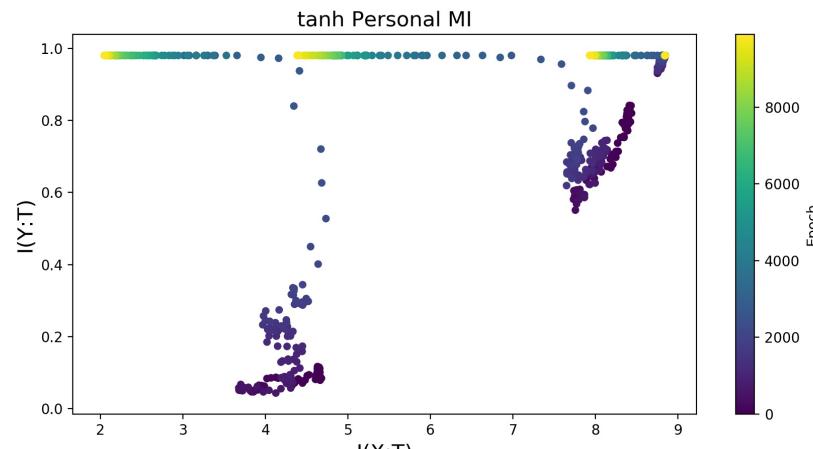


(b) Batch size = 1.

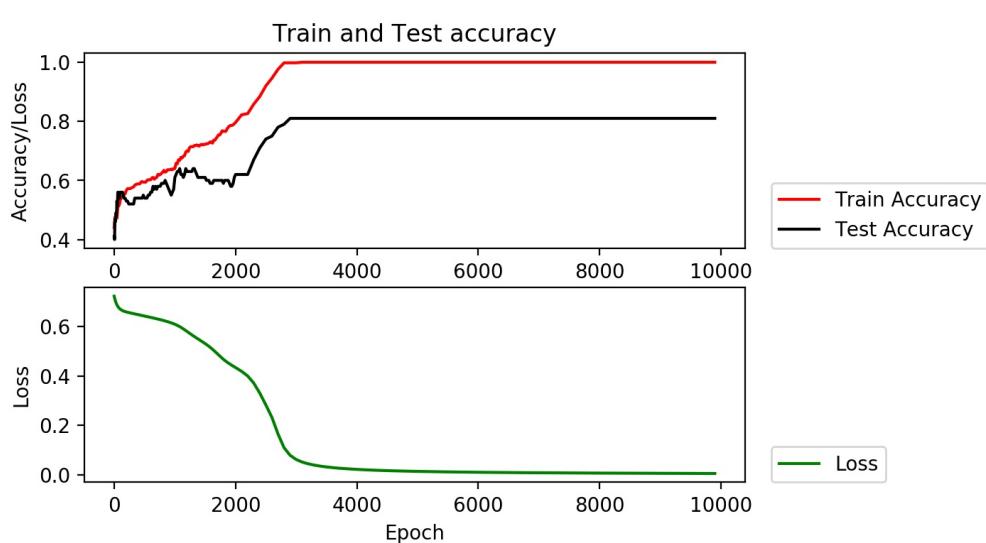


(a) Batch size = 64.



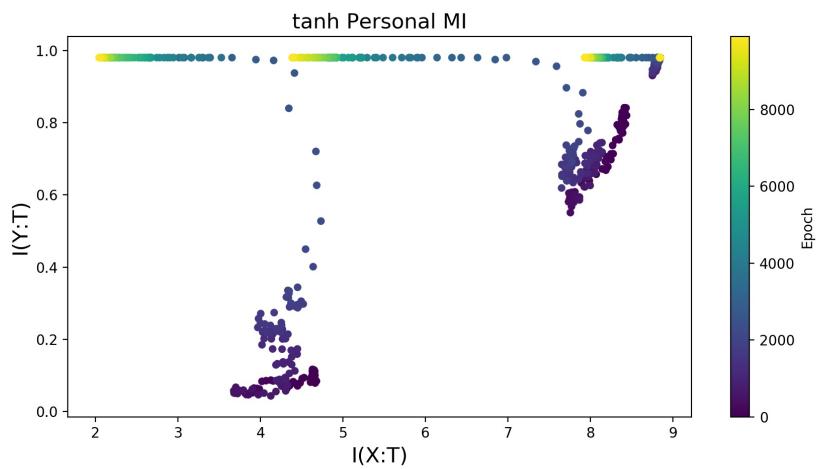


(a) Batch size = 512.

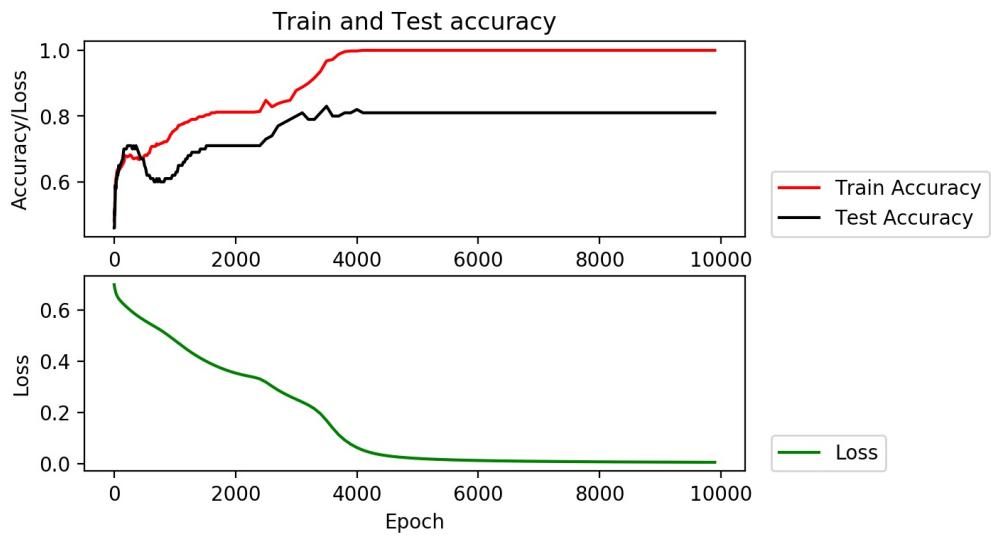


(b) Batch size = 512.

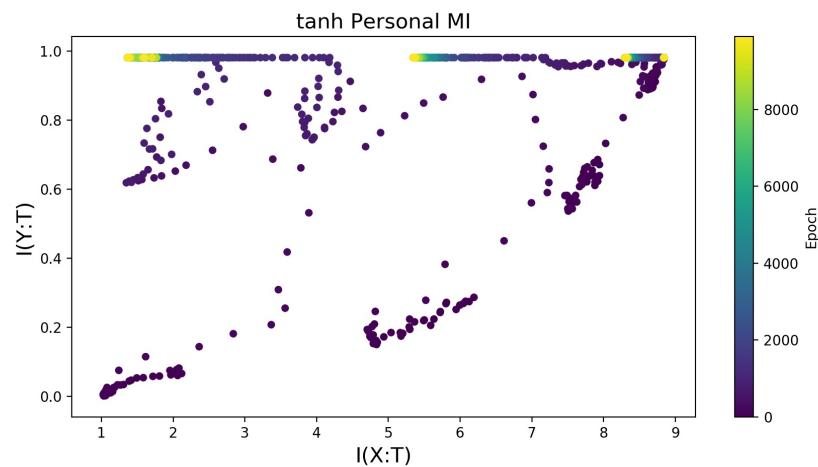
Figura 3.14: Andamento informazione con dataset di train al variare del batch size.
Learning rate=0.001, n.bins=30.



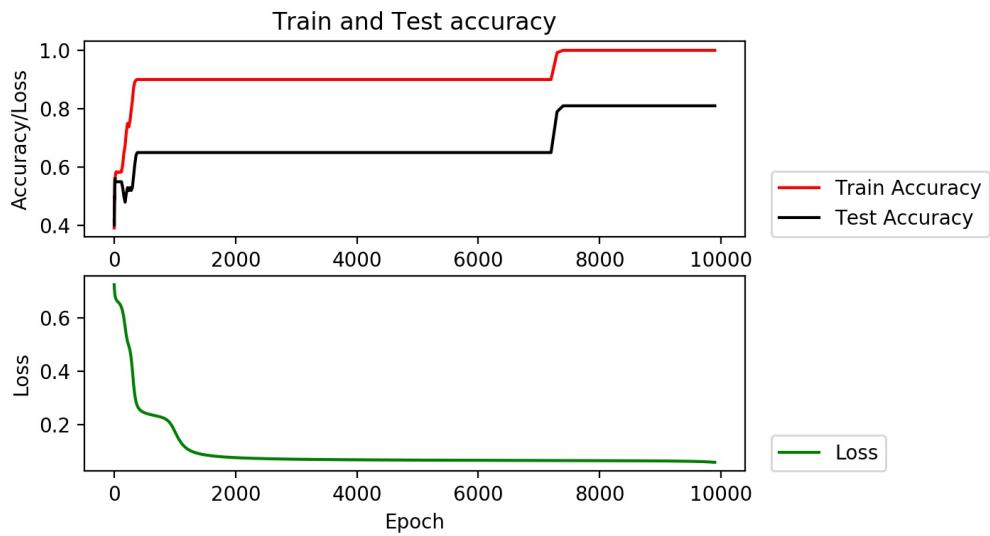
(a) Learning rate = 0.001.



(b) Learning rate = 0.001.



(a) Learning rate = 0.007.



(b) Learning rate = 0.007.

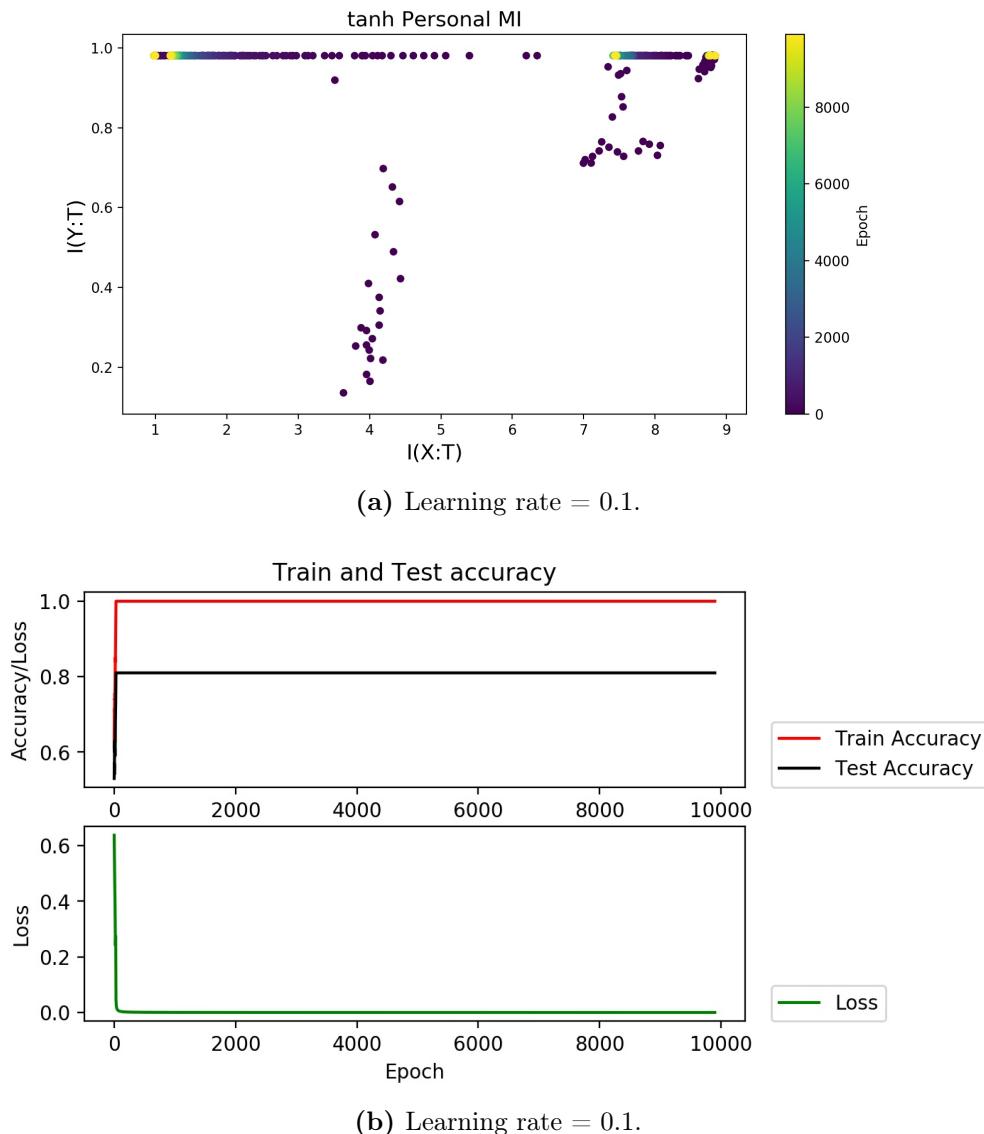
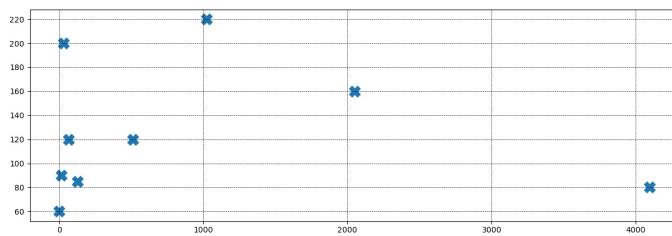


Figura 3.15: Andamento informazione con dataset di train al variare del learning rate.
Batch size=512, n.bins=30.

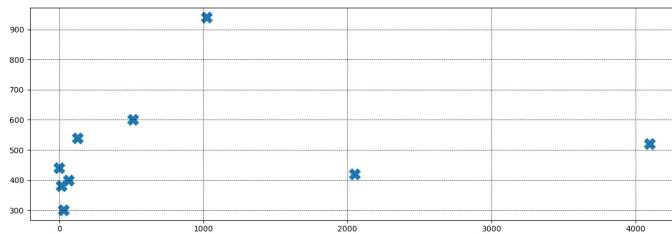
Anche variando i parametri di learning rate e batch size, e mostrando il grafico della mutua informazione calcolata utilizzando il dataset di test (Figura 3.17), la rete converge praticamente negli stessi punti, anche se in momenti diversi. Combinando questo, con l'**IB** framework, si può affermare che i parametri della rete non modificano il moltiplicatore di Lagrange β , e quindi non hanno un peso sulla rappresentazione ottima di ogni layer.

Dalla letteratura è noto che il rumore introdotto aiuta ad uscire dai minimi locali durante l'addestramento, e come supposto inizialmente in questa tesi, ad aiutare durante la compressione. Si può quindi affermare esserci un fattore di proporzionalità tra learning rate ed il rumore introdotto dal gradiente stocastico.

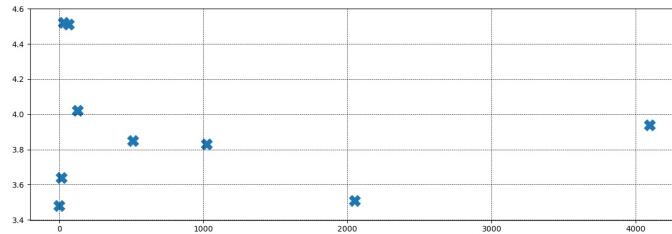
Le analisi sperimentali di Figura 3.16 hanno portato alla conclusione che non c'è nessun legame lineare tra batch size e la traiettoria dell'informazione.



(a) Relazione batchsize ed epoch in cui si raggiunge il max. valore di $I(X, T)$.



(b) Relazione batchsize ed epoch in cui si raggiunge il max. valore di $I(T, Y)$.



(c) Relazione batchsize e max. valore di $I(X, T)$ raggiunto durante il training.

Figura 3.16: Analisi sperimentali al variare dei parametri alla ricerca di eventuali linearità.

I layer tendono a convergere in prossimità dell'[IB Curve](#), e per diversi valori dei parametri, le distribuzioni di probabilità $P(X, T_i)$ per la codifica e $P(T_i, Y)$ per la decodifica, soddisfano le equazioni autoconsistenti dell'equazione [\(2.35\)](#).

Si vuole ricordare inoltre, come visto precedentemente, che la profondità della rete ha invece una relazione con β , infatti modificando il numero di hidden layer, la traiettoria dell'informazione converge in punti diversi all'interno del piano dell'informazione (Figura [3.13](#)).

Le fasi di [ERM](#) e di compressione, come si potrà vedere nelle prossime sezioni, sono presenti pure in altri dataset, possiamo quindi dire che tali fasi sono una caratteristica generale delle reti *Fully connected* se addestrate tramite il [SGD](#) e funzione di attivazione *tanh*.

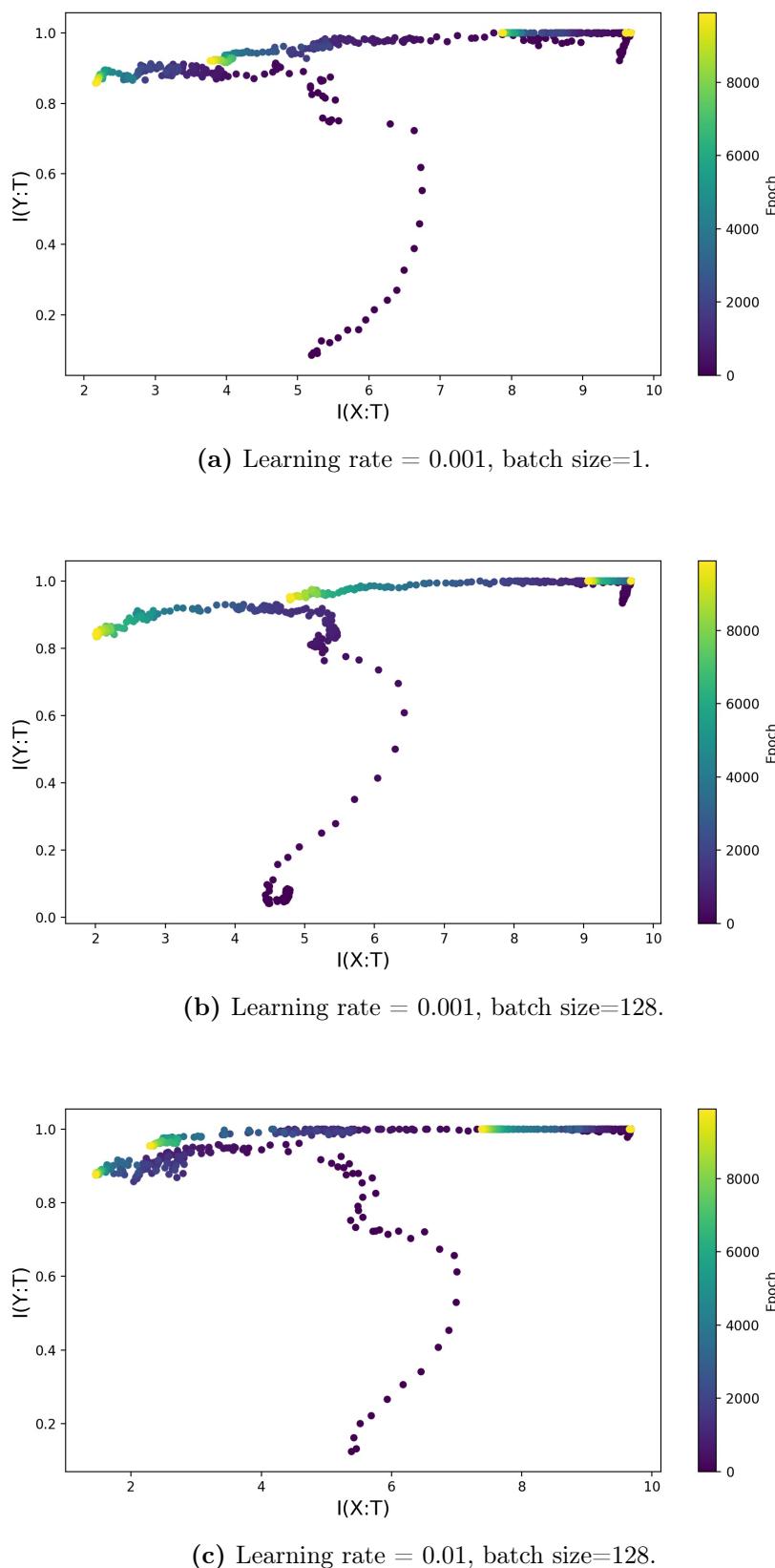


Figura 3.17: Andamento Mutua Informazione sul dataset di test al variare del learning rate e batch size, n.bins=30.

3.4 Simulazione con Dummy Dataset

Nel dataset *Dummy*, creato attraverso l'algoritmo 3.1.1, come era già successo nel dataset *IB*, si ha un comportamento poco spiegabile della traiettoria dell'informazione nel caso in cui venisse utilizzata come funzione di attivazione la funzione *ReLU* o la *LeakyReLU*. Nel caso in cui si utilizzi la funzione *tanh* si ottengono risultati simili a quelli ottenuti precedentemente per il dataset *IB*.

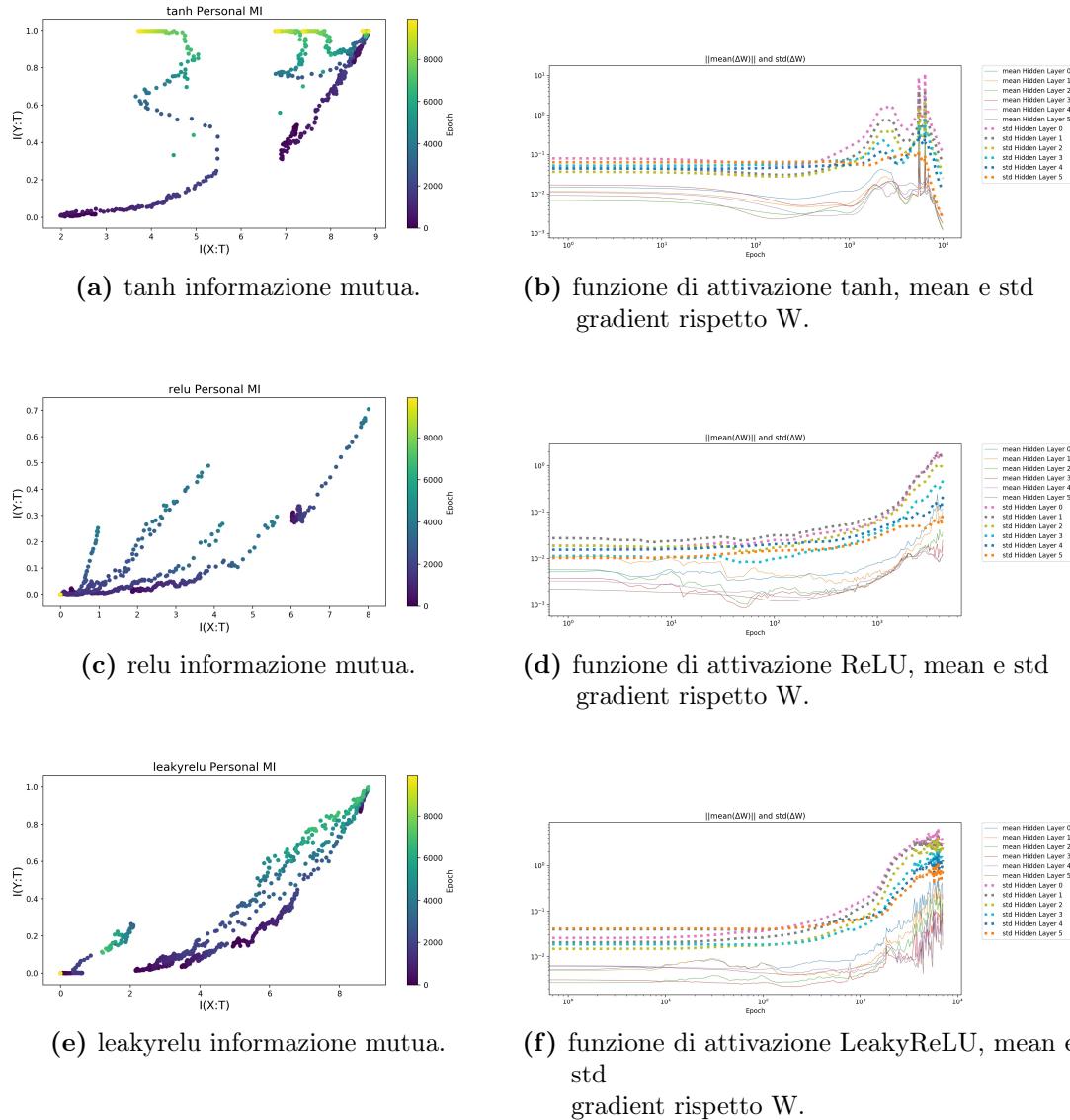


Figura 3.18: Andamento Mutua Informazione e gradiente al variare della funzione di attivazione per il Dummy Dataset. Learning rate=0.005, batch size=512, n.bins=30.

3.5 Simulazione con Struttura Dataset

Utilizzando il dataset *Struttura*, creato attraverso l'algoritmo 3.1.3, quello che cambia è solamente una difficoltà maggiore da parte della rete ad imparare le etichette in uscita. Mentre si può notare la fase di compressione quando viene utilizzata la funzione *tanh*, questa non la si vede invece negli altri casi.

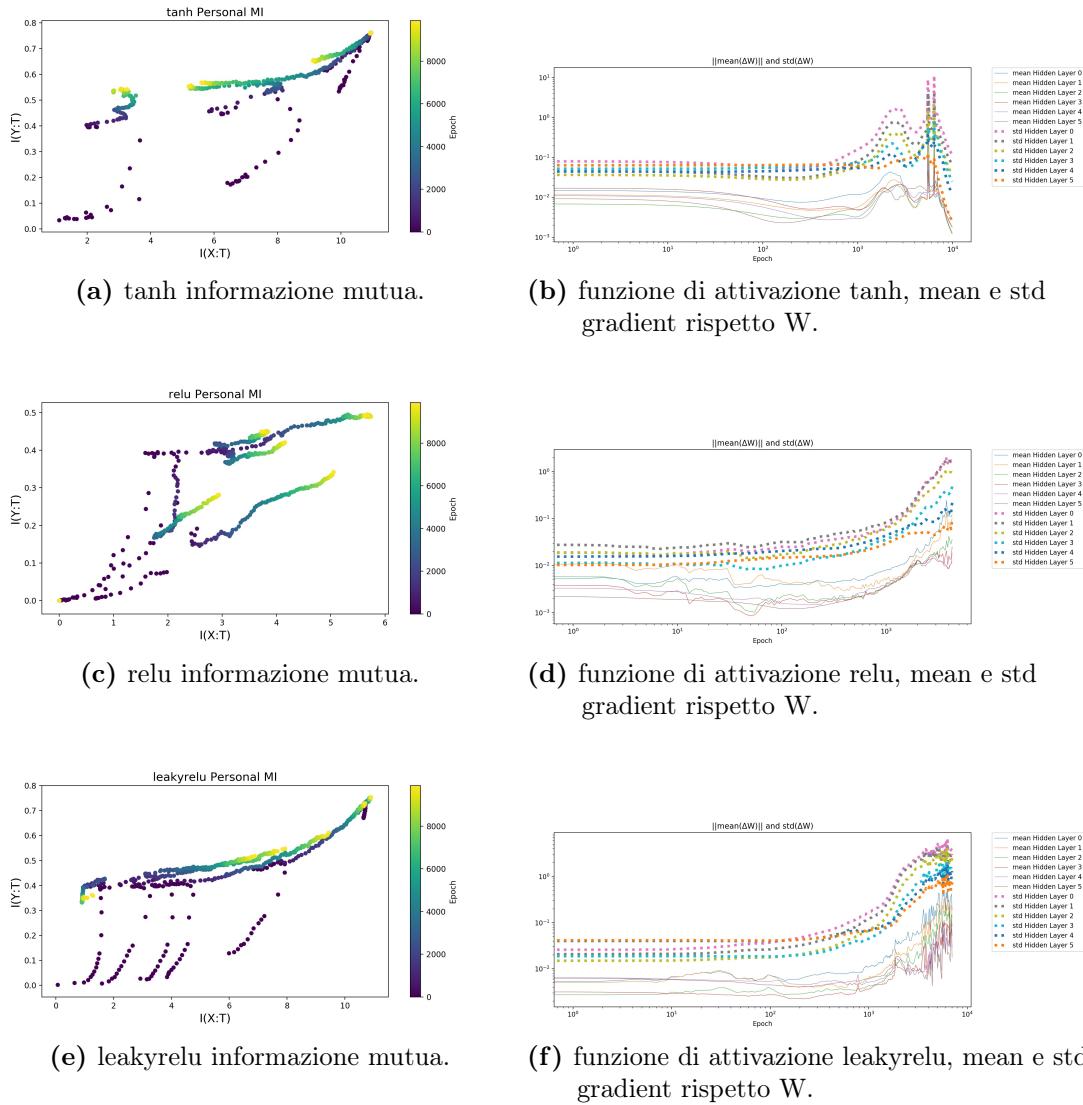


Figura 3.19: Andamento Mutua Informazione al variare della funzione di attivazione per il Struttura Dataset. Learning rate=0.005, batch size=512, n.bins=30.

3.6 Simulazione con MNIST Dataset

Stimare l'informazione quando si ha una grande dimensionalità dell'input in ingresso è difficoltoso, ma si sono lo stesso ottenuti risultati interessanti nel caso del *MNIST* [28], e del *CIFAR* nella Sezione 3.7. Anche in questo caso si può notare una fase di compressione quando viene utilizzata la funzione di attivazione *tanh*. Alcune delle traiettorie dell'informazione sono state compresse nello stesso punto, ma è possibile ottenere un andamento simile al caso del dataset *IB* nel caso in cui si scelga un miglior numero di neuroni per gli hidden layer.

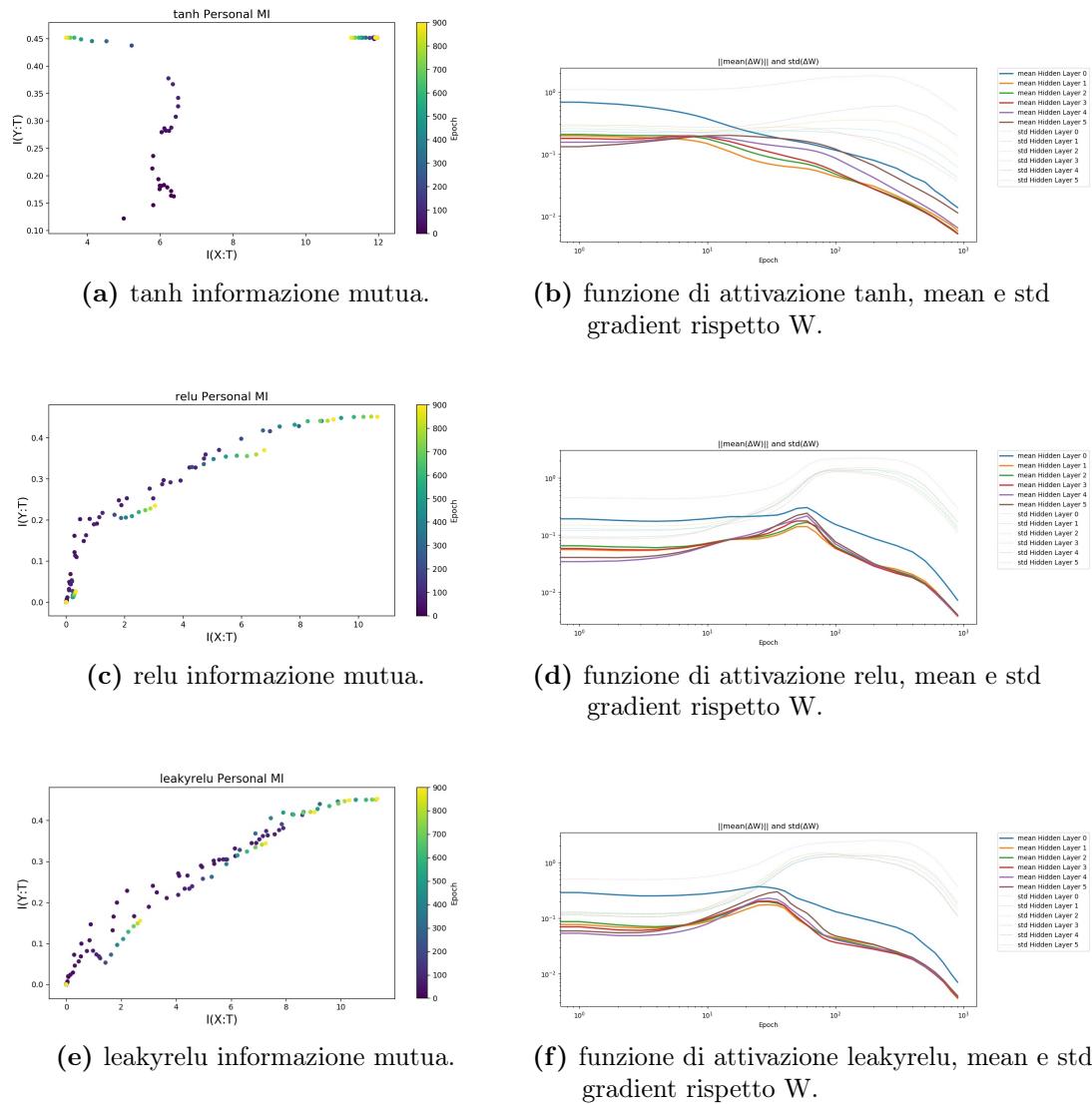


Figura 3.20: Andamento Mutua Informazione al variare della funzione di attivazione per il MNIST Dataset. Learning rate=0.001, batch size=512, n.bins=30, n.samples=4000. Configurazione Hidden Layers: 32-32-32-16-16.

3.7 Simulazione con CIFAR-10 Dataset

È possibile reperire maggiori informazioni sul dataset *CIFAR-10* dalla bibliografia [26]. Anche in questo caso si ha una compressione dell'informazione rispetto l'ingresso quando viene utilizzata la funzione di attivazione *tanh*, e non la si ottiene negli altri casi.

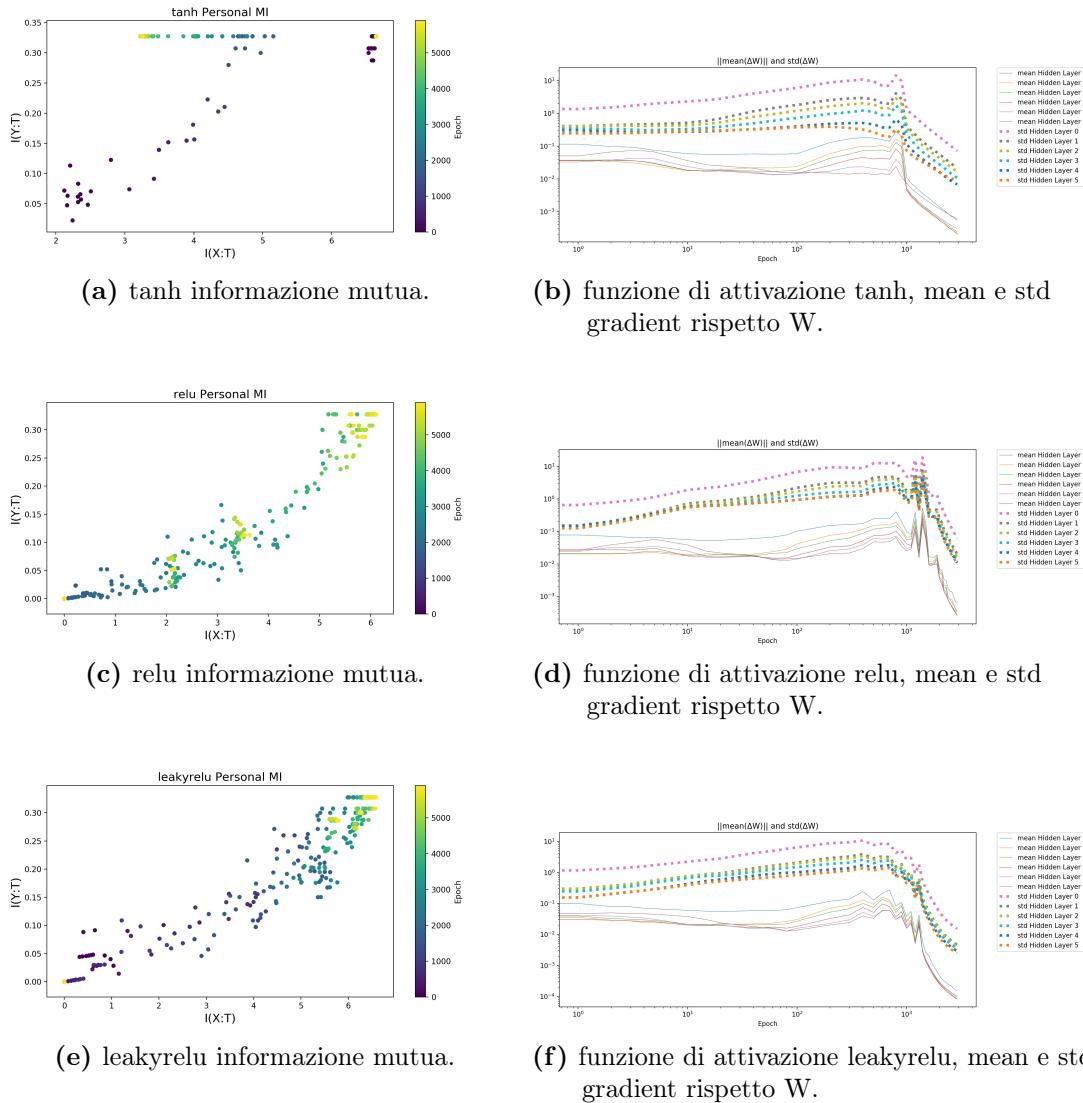


Figura 3.21: Andamento Informazione al variare della funzione di attivazione per il CIFAR-10 Dataset. Learning rate=0.001, batch size=256, n.bins=30, n.samples=100. Configurazione Hidden Layers: 32-32-32-16-16.

Applicazioni e lavoro futuro

Qui di seguito verranno esposte alcune ricerche attualmente in corso che utilizzano le nozioni derivanti dall'**IB**, e le quali risultano essere interessanti da essere approfondite per un eventuale lavoro futuro.

Attualmente il fatto che l'**IB** possa venir utilizzato per analizzare la capacità di generalizzazione di una **DNN**, le proprietà del **SGD** e i benefici derivanti dall'aggiunta di hidden layer, è ancora argomento di dibattito [32, 33, 45].

Il *Framework* dell'**IB** è stato applicato con dei risultati notevoli in campi differenti dalle **DNN**, come per esempio: nel campo delle telecomunicazioni e *signal processing* per le decodifiche dei codici Low Density Parity-check Codes (**LDPC**) [40], nell'analisi dei geni [23], nella bioinformatica [10], nel *clustering* e nella classificazione non supervisionata [36–38].

In particolare Amjad e Geiger eseguono un'analisi sulle difficoltà derivanti dall'addestramento delle **DNN**, andando a minimizzare il funzionale dell'**IB** (Equazione 2.34), sia nel caso che la **DNN** sia deterministica che stocastica. Analizzano inoltre il ruolo che assume il rumore, introdotto all'interno della rete durante la fase di addestramento, il quale porta gli stessi benefici derivanti dall'aumento della dimensione del dataset in ingresso [4]. Altre ricerche invece, vanno a sostituire il funzionale dell'**IB** con una funzione costo a questo dipendente, la quale verrà minimizzata durante l'addestramento, ottenendo così una rappresentazione più robusta [1, 3, 25].

Si è notata una grande sensibilità della traiettoria dell'informazione in base al numero di bin utilizzati, e scegliere il numero di bin ottimali per stimare l'informazione non è un problema banale. In alcune ricerche recenti è stato utilizzato un *binning dinamico*, per ottenere il fenomeno della compressione anche per le funzioni di attivazione ReLU e LeakyReLU [29]. In particolare si afferma che, poiché il massimo valore in uscita da un neurone può cambiare anche di due ordini di grandezza in base al layer o epoch considerato, usare un intervallo fisso durante la procedura di binning implica che alcuni layer, durante alcuni epoch, hanno tutti i loro valori che finiscono all'interno dello stesso bin. Ivan Chelombiev afferma che questa metodologia di analisi risulta essere errata, evitabile tramite un binning basato sull'Entropia. In questo caso, l'intervallo dei bin viene scelto in maniera tale che ogni bin contenga lo stesso numero di valori unici di attivazione, per un dato layer ed epoch [22]. Poiché in tali ricerche non si è fornita una motivazione teorica sulla validità di tale procedimento, si è deciso di lasciarlo come un possibile lavoro futuro.

Conclusioni

In questa sezione si riassumeranno le conclusioni a cui si è giunti dopo la fase di sperimentazione ed analisi.

1. Come affermato da Tishby, la maggior parte del tempo di addestramento viene speso per comprimere l'informazione del dataset in ingresso e non nella fase di apprendimento (Figura 3.4). In questo caso si potrebbero utilizzare algoritmi più efficienti, come quelli per la stima *MAP* [18], per velocizzare l'addestramento durante la fase di compressione.
2. Il successo delle **DNN** nel generalizzare, potrebbe essere dovuto al fatto che queste trovano in modo efficiente la rappresentazione minima dell'ingresso, date le etichette in uscita, proprio come fa il metodo dell'**IB**. Il valore di β all'interno del funzionale dell'Equazione 2.34 è però sconosciuto.
3. Il tempo di addestramento diminuisce quando si aggiungono più hidden layer, ottenendo un vantaggio di tipo computazionale (Appendice A.6), ma questo vale fino ad un determinato numero di layer, dopodiché se ne vengono aggiunti ulteriori, si ha una degradazione delle performance di generalizzazione del sistema. Inoltre, all'aumentare della profondità della **DNN** si ha una riduzione del tempo di rilassamento del **SGD** (Figura A.7).

I risultati sperimentali portano ad una conclusione diversa da quanto affermato da Tishby. Il tratto di colore nero presente in Figura A.8b, è il punto in cui si raggiunge il massimo valore di $I(T, Y)$ per l'ultimo hidden layer, ma come si può vedere invece dalla Figura 3.13b, dopo aver raggiunto il massimo valore di $I(T, Y)$, l'ultimo layer inizia a comprimere le informazioni dell'ingresso. In quest'ultima fase è però associato un alto **SNR**, giungendo così alla conclusione che non può essere il rumore, introdotto dal processo di aggiornamento dei pesi, il motivo della compressione.

Il processo di compressione non è causato dall'andamento diffusivo del **SGD** come affermato da Tishby, in quanto si ottengono grafici simili pure nel caso in cui si usi il puro **GD** o l'**MGD**.

4. Una fase di compressione non indica per forza una generalizzazione migliore [45]. Come affermato da Saxe et al. in [33]: “*Una rete che comprime potrebbe/non potrebbe generalizzare bene, una rete che non comprime potrebbe/non potrebbe generalizzare bene*”, o per meglio dire: generalizzazione e compressione non sono sempre legate tra di loro. In accordo con quanto si vede in Figura A.3a, i punti dove si ha il massimo valore della accuratezza calcolata sul test dataset, e su quella calcolata sul train dataset, si trovano in punti

antecedenti a quello di massima compressione dell'ingresso, ottenuto durante l'addestramento. Si può quindi supporre che compressione e generalizzazione sono si legate tra di loro, ma non sono la stessa cosa.

Da notare inoltre che nelle Figure A.3a, A.3b e A.3c, a fronte dello stesso punto di convergenza nel piano dell'informazione, si sono ottenuti valori di accuratezza sul test dataset, e quindi generalizzazioni, diverse.

Infine si vuole far presente, che è più corretto parlare di generalizzazione quando viene utilizzato il test dataset per visualizzare l'andamento della traiettoria della mutua informazione, altrimenti sarebbe meglio parlare di memorizzazione, all'aumentare di $I(T, Y)$ nel piano dell'informazione.

5. Si può giustificare il fenomeno della lenta compressione nel seguente modo: la funzione *tanh* attrae le informazioni nelle sue due zone di saturazione, da cui ne escono difficilmente (Figura 3.7a), e se ne venisse calcolato il gradiente per aggiornare i parametri della rete attraverso l'algoritmo di *Backpropagation*, si troverebbe un'infinitesima variazione della matrice dei pesi W e dei bias b , essendo le derivate in tali punti praticamente nulle. Questo potrebbe essere il motivo della lenta fase di compressione.
6. Layer diversi tendono a convergere verso punti diversi dell'**IB Curve** (Figura 3.4), si può quindi affermare che il *moltiplicatore di Lagrange* β è differente per ogni hidden layer, dipendendo esso dall'architettura e posizione del layer considerato.
7. Quando si è in condizione di *undertraining*, la traiettoria dell'informazione del test set tende ad essere più convessa, si può utilizzare tale andamento per implementare un *early stopping* (Appendice A.7).
8. Guardando l'istogramma della media dei bin nella Figura 3.8, si nota come le informazioni vengano spinte su entrambe le curve di saturazione positiva e negativa della *tanh*, durante la fase di addestramento. Si vengono così a formare due *cluster*, la cui distanza dei centroidi viene massimizzata dal **SGD**. Questo comportamento porta a perdere informazioni sull'ingresso X , facendo diminuire $I(X, T)$, mentre l'activity dei neuroni entra a far parte in uno dei due cluster.
9. In base alla forma della funzione ReLU, possiamo concludere che pure questa funzione comprime le informazioni in ingresso, ma meno rispetto la funzione *tanh*. L'andamento particolare della traiettoria dell'informazione quando vengono utilizzate funzioni con andamento non a saturazione (Figura 3.6), potrebbe essere dovuta ad una stima errata della mutua informazione attraverso il binning statico, essendo questa fortemente dipendente dal numero di bin. Una soluzione a questo problema potrebbe essere quello di utilizzare un binning di tipo dinamico, ma questa analisi la si lascia per un eventuale lavoro futuro.
10. Si è visto nella sezione 3.3.3, come i valori finali della mutua informazione non sono dipendenti dai parametri, ma dipendono invece solamente dalla tipologia della rete. Utilizzando le nozioni derivanti dall'**IB** si può affermare che quello

che cambia è il *moltiplicatore di Lagrange* β nell'equazione (2.34), facendo così convergere i diversi layer in punti differenti. Rimane comunque poco chiaro se β dipende ulteriormente da altri parametri. Possiamo affermare che la fase di compressione aumenta la distanza tra i cluster delle diverse classi. Quando però nel dataset in ingresso è presente del rumore, la compressione porta la DNN ad imparare dettagli errati, peggiorando così la generalizzazione.

11. Trovare la mutua informazione, e risolvere il funzionale dell'IB (Equazione 2.34) quando si utilizza una rete DNN deterministica, porta a dei problemi, e per questo motivo si utilizza una DNN di tipo stocastica. Uno studio più approfondito di queste problematiche la si lascia come un'eventuale lavoro futuro, così come il ruolo che ricopre il rumore, introdotto all'interno della rete durante la fase di addestramento, per migliorare le performance di generalizzazione e robustezza della rete [1, 4].

Appendice A

Appendice terzo capitolo

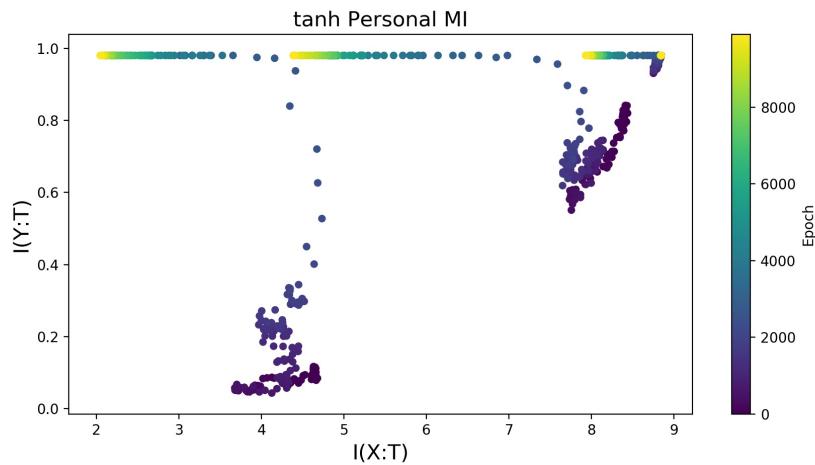
A.1 Mutua Informazione

Il programma scritto si basa su due step distinti implementati in due file separati: Nel primo (*fit.py*), viene allenata la rete e vengono salvati tutti i dati di maggior interesse (per esempio l'activity dei neuroni per il dataset in ingresso) per essere utilizzati nella seconda fase, questa procedura di salvataggio viene effettuata solo per determinati epoch, risparmiando così delle risorse di calcolo. Nel secondo step invece viene calcolata l'informazione mutua (*xt_ty.py*).

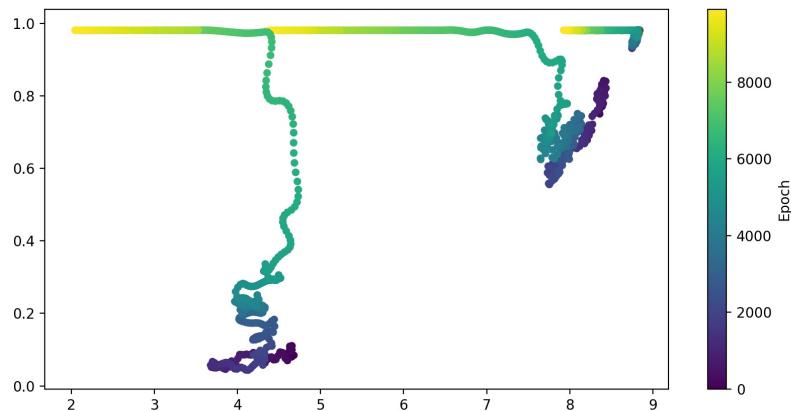
La scelta di utilizzare una risoluzione temporale non unitaria, è stata fatta per ridurre la quantità di dati salvati sullo spazio di archiviazione ed aumentare la velocità di computazione dell'informazione. Si è scelto di lavorare su due fasi distinte, per poter effettuare un'analisi “offline” della mutua informazione senza la necessità di allenare nuovamente la rete, ogni qual volta si voglia modificare il metodo per il calcolo dell'informazione. Nel caso in cui il lettore ne avesse bisogno, è stato sviluppato del codice che consente di ottenere una risoluzione maggiore della mutua informazione utilizzando delle funzioni interpolanti (Appendice A.1.1) o un metodo di addestramento “online” (Appendice A.1.2). Attraverso la versione *online* è possibile fare anche un rimescolamento dei dati in ingresso senza gravare sullo spazio di archiviazione (per maggiori dettagli si veda più avanti).

A.1.1 Interpolazione dei dati

Poiché si ha a disposizione una serie di punti per la traiettoria dell'informazione, per poter aumentare la risoluzione si può utilizzare una funzione interpolante, con il grado del polinomio di interpolazione a scelta dell'utente. Perdendo però l'informazione sulla velocità di convergenza essendo gli epoch interpolanti distanziati da una funzione lineare.



(a) Funzione senza interpolazione, n.train samples=3096, learning=0.001, batch size=512.



(b) Funzione con interpolazione, n.train samples=3096, learning=0.001, batch size=512.

A.1.2 Calcolo online dell'informazione mutua

Suddividere il programma su due step diversi porta a delle conseguenze. Nel caso in cui si volesse fare un rimescolamento del dataset durante la fase di addestramento, nasce la necessità di salvarsi il dataset “mischiato” che causa una determinata uscita dei neuroni, per poter così calcolare le distribuzioni di probabilità e le informazioni mutue in modo corretto. Come si può facilmente intuire, nel caso di *dataset* di dimensione consistenti, come per esempio il MNIST, ciò richiederebbe una grande quantità di spazio di archiviazione. Per poter ovviare a questo problema si è implementato del codice per effettuare un calcolo “online” della mutua informazione, durante la fase di addestramento, consentendo anche di utilizzare una risoluzione maggiore per la mutua informazione. Questo comporta l'impossibilità di utilizzare metodi come il binning dinamico senza poter conoscere a priori il valore massimo e minimo che assumono le uscite dei neuroni durante la fase di addestramento.

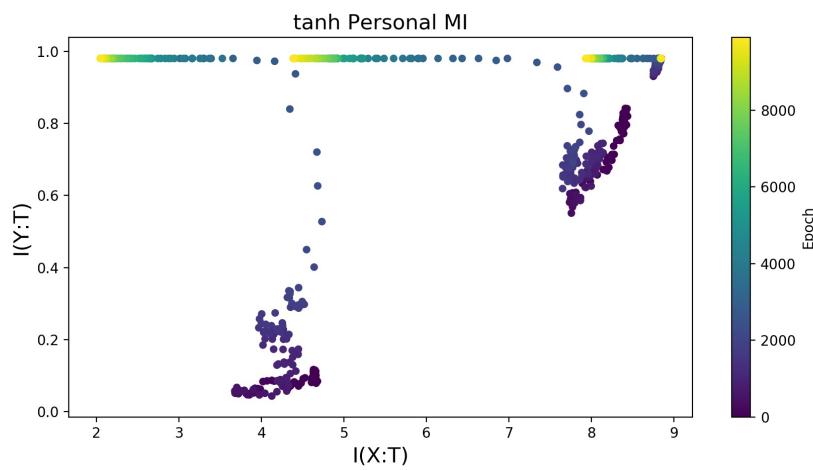


Figura A.1: Addestramento online con funzione di attivazione tanh.
n.train samples=3096, learning=0.001, batch size=512.

A.1.3 Rimescolamento dataset

Nel caso in cui il dataset venisse rimescolato durante l'addestramento, le curve che si ottengono sono molto simili:

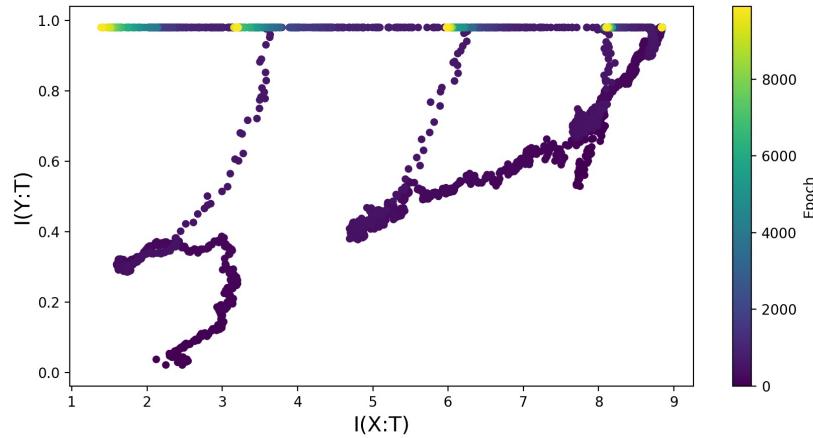


Figura A.2: Informazione mutua nel caso in cui si facesse il rimescolamento del dataset.
n. campioni per l'addestramento=3096, learning=0.001,batch size=512.

A.2 Requirements

File *Requirements.txt* contenente le versioni ed i pacchetti utilizzati durante la stesura della seguente tesi.

Codice A.1: File requirements.txt contenente la lista dei pacchetti utilizzati e le rispettive versioni.

```

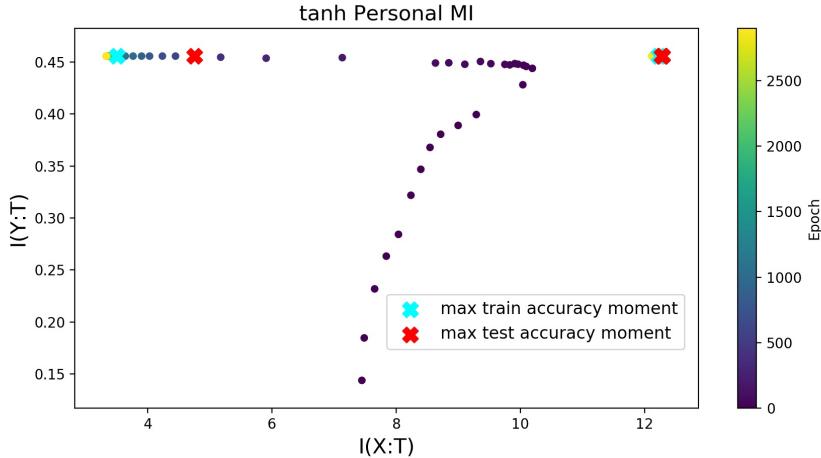
1 # Requirements automatically generated by pigar .
2 # https://github.com/damnever/pigar
3
4 joblib == 0.13.2
5 matplotlib == 3.1.1
6 numpy == 1.16.5
7 pandas == 0.25.1
8 pathlib2 == 2.3.5
9 scikit_learn == 0.21.3
10 scipy == 1.3.1
11 statsmodels == 0.10.1
12 tensorflow == 2.0.0

```

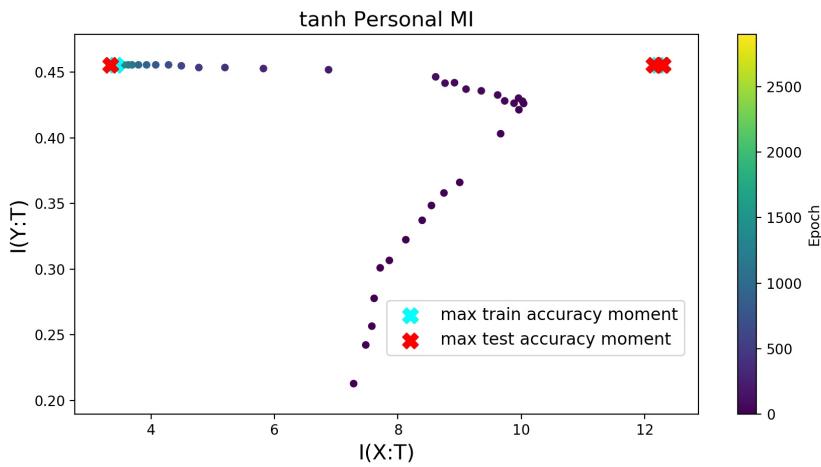
A.3 Compressione e generalizzazione

Per analizzare in maggior dettaglio la mutua informazione, si è scritto del codice per andare ad aggiungere sul grafico della traiettoria dell'informazione, i punti di massima accuratezza sul dataset di train e di test.

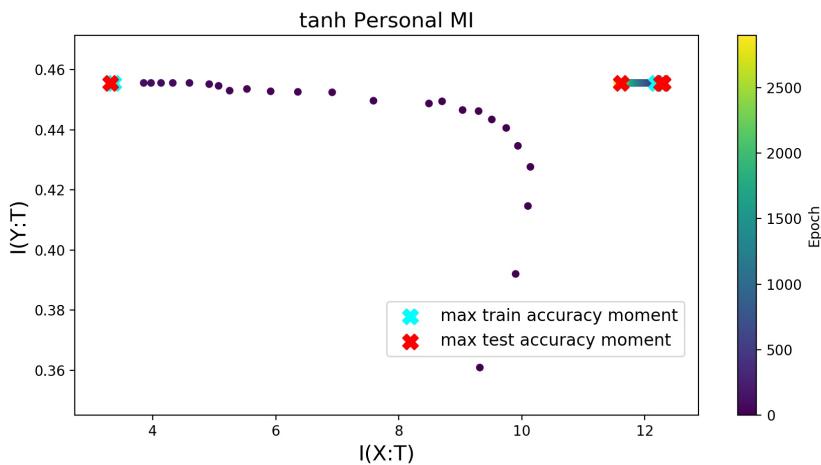
Osservando la Figura A.3 si può giungere ad una conclusione simile a quella fatta da Tishby, cioè che la compressione è il processo per cui la DNN è capace di generalizzare, ma bisogna però fare una precisazione. Mentre nella Figura A.3b, e nella Figura A.3c, si ottiene il massimo valore della test accuracy in corrispondenza del massimo livello di compressione, questo non vale per la Figura A.3a. Inoltre è da notare che a fronte dello stesso livello di compressione rispetto l'ingresso, nella Figura A.3b, e Figura A.3c si ha un valore di accuratezza più alto del 0.01 e 0.02 rispettivamente, rispetto la Figura A.3a. Si può quindi concludere che compressione e generalizzazione sono si collegate tra di loro, ma non sono per forza equivalenti.



(a) Learning=0.001, batch size=64. Max test accuracy=0.892.



(b) Learning=0.001, batch size=512. Max test accuracy=0.912.



(c) Learning=0.01, batch size=512. Max test accuracy=0.907.

Figura A.3: I punti evidenziati in rosso e celeste sono quelli in cui si raggiunge il massimo valore dell'accuratezza durante l'addestramento, sia per il train dataset, che per il test dataset. Questo allo scopo di poter mostrare come accuratezza e generalizzazione sono collegate, ma non indicano la stessa cosa.
 N.train samples=5000, n.test samples=1000, dataset MNIST, n.bins=30.

A.4 Andamento entropia

Per completezza si vuole inoltre mostrare il grafico dell'andamento dell'entropia durante l'addestramento.

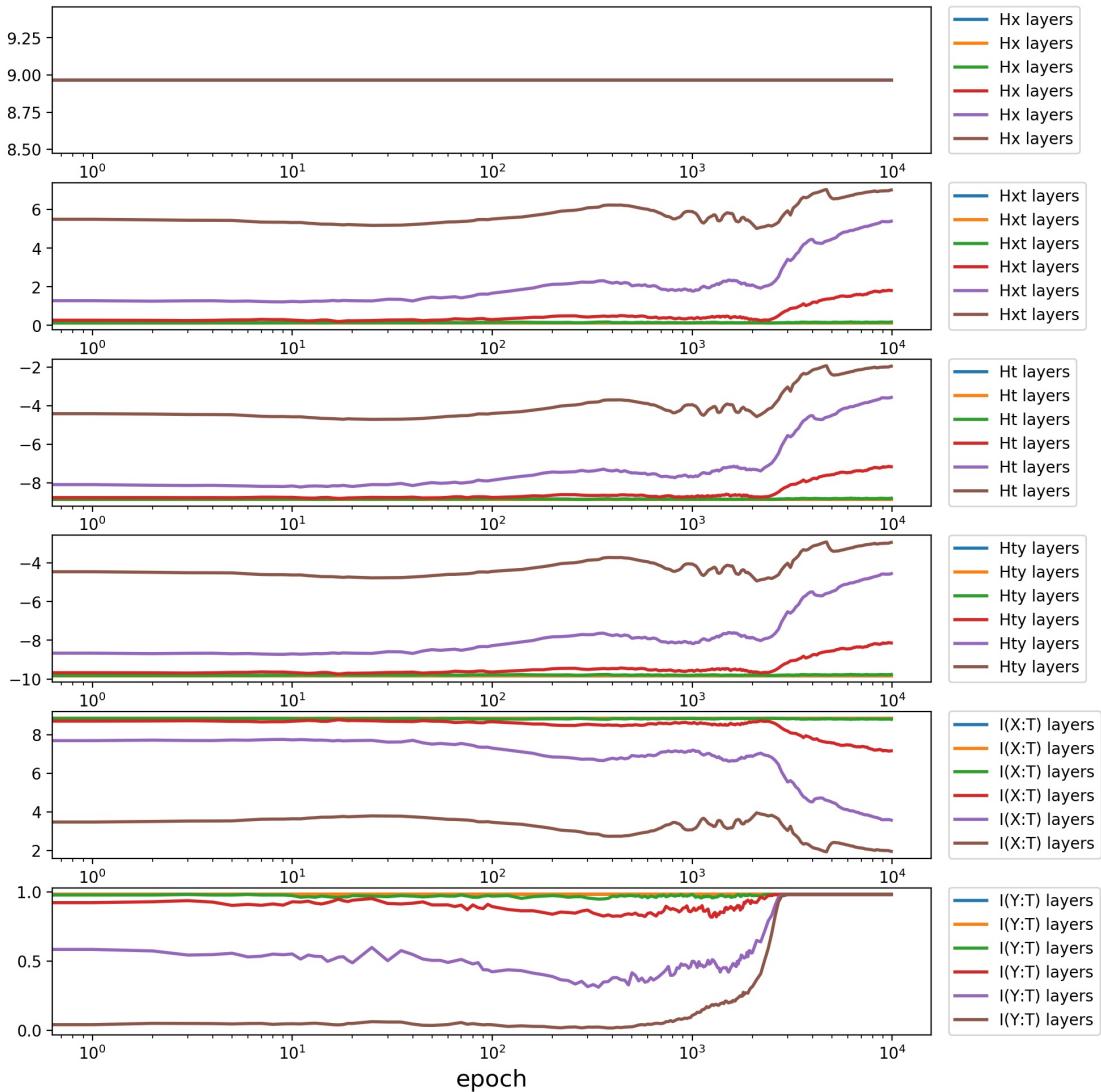
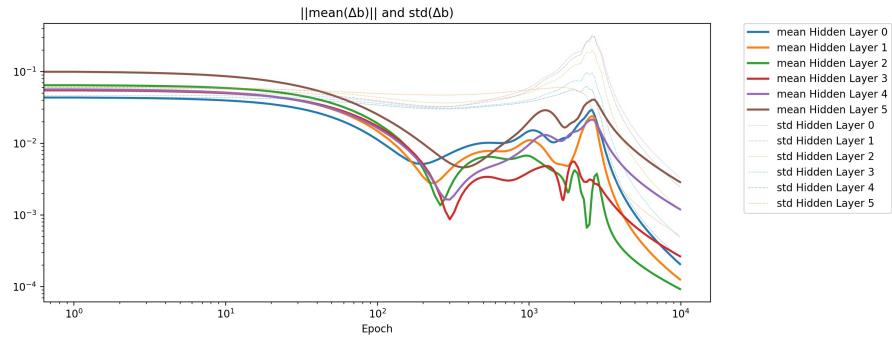


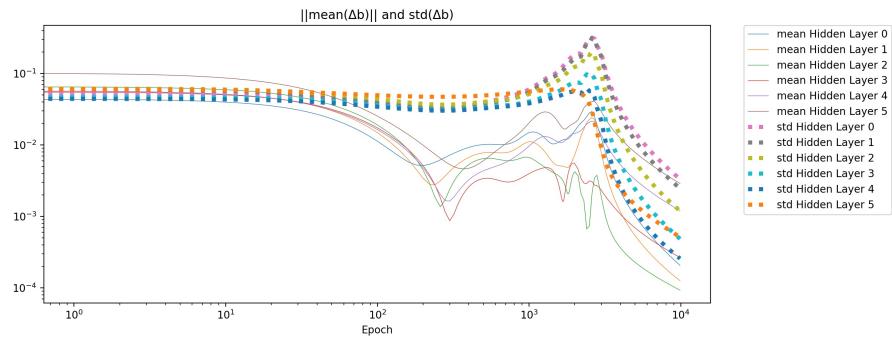
Figura A.4: Rappresentazione durante l'addestramento di come variano i diversi contributi dell'entropia all'interno della mutua informazione. Learning rate = 0.001, batch size=512, n.bins=30.

A.5 Gradiente funzione costo calcolato rispetto il bias

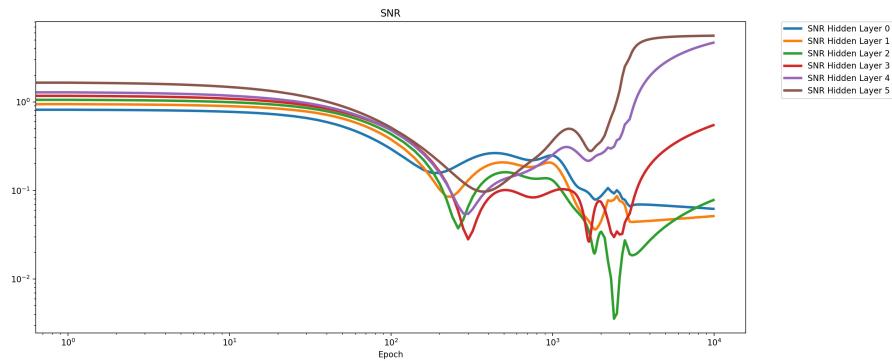
Come era stato anticipato precedentemente, nel caso in cui si calcolasse il gradiente della funzione costo rispetto il vettore di bias b , si ottiene un andamento molto simile a quello di Figura 3.12, dove in quel caso si era mostrato il grafico del gradiente calcolato rispetto la matrice dei pesi W . In entrambi i grafici la *sovraelongazione* è nel medesimo punto. Il vantaggio di calcolare il gradiente rispetto b e non rispetto W è quello di richiedere una minore capacità computazionale.



(a) Andamento media gradiente della funzione costo rispetto b.



(b) Andamento std gradiente della funzione costo rispetto b.

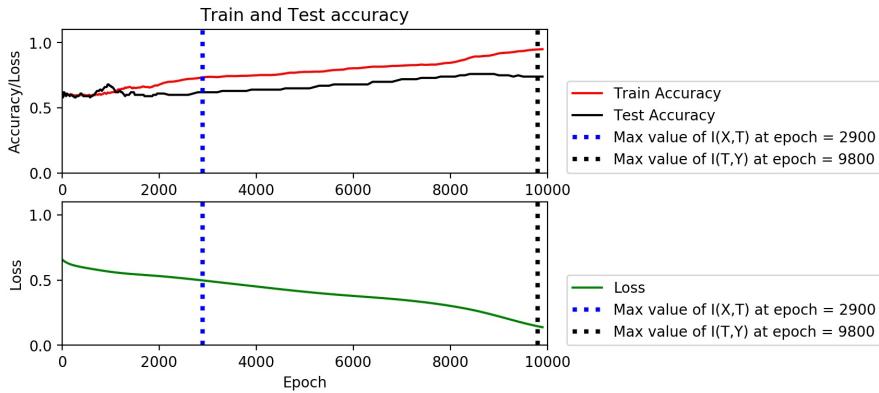


(c) Andamento SNR gradiente della funzione costo rispetto b.

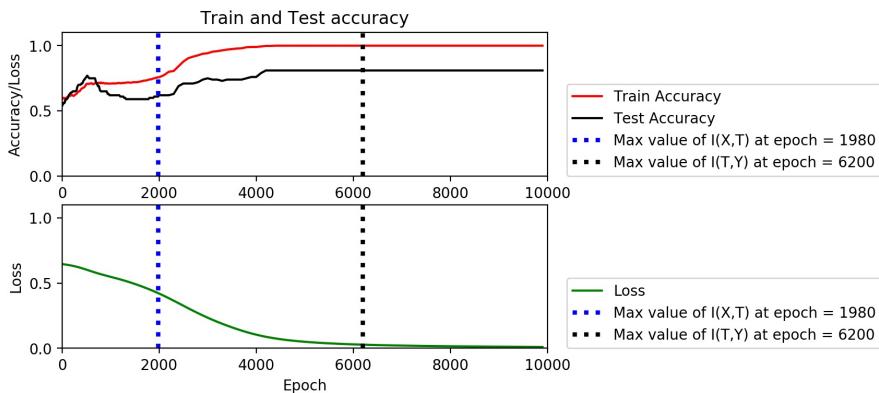
Figura A.5: Andamento media, std ed SNR del gradiente della funzione costo rispetto b. Learning rate=0.001, Batch size=512.

A.6 Variazione numero Hidden layer

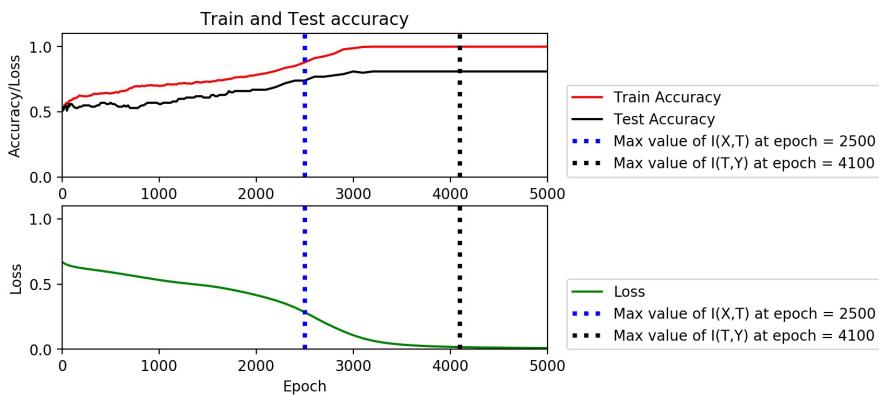
Come si può notare in Figura A.6, nel caso in cui si aumentino il numero di hidden layer, la funzione costo si riduce in un tempo inferiore. Questo vale però per un numero massimo di hidden layer, dopodiché i miglioramenti di velocità iniziano a degradarsi, come viene anche affermato dalla teoria [31].



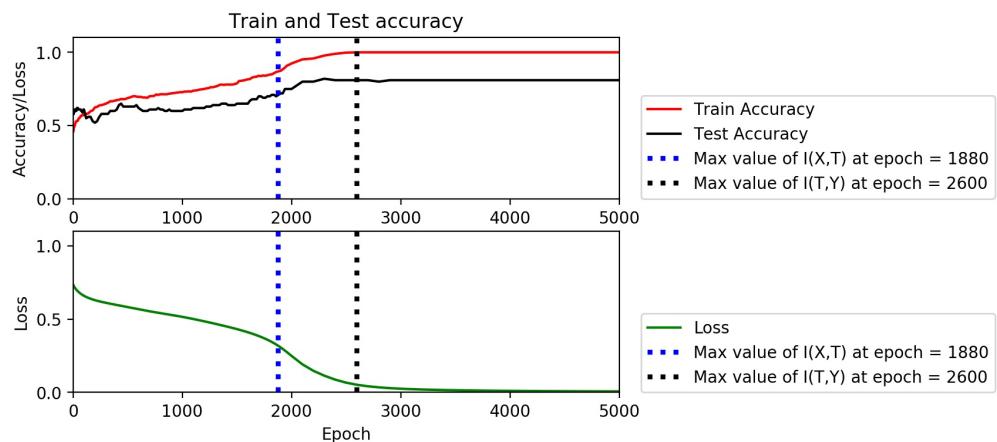
(a) Configurazione 1 Layer. Configurazione Neuroni: 10.



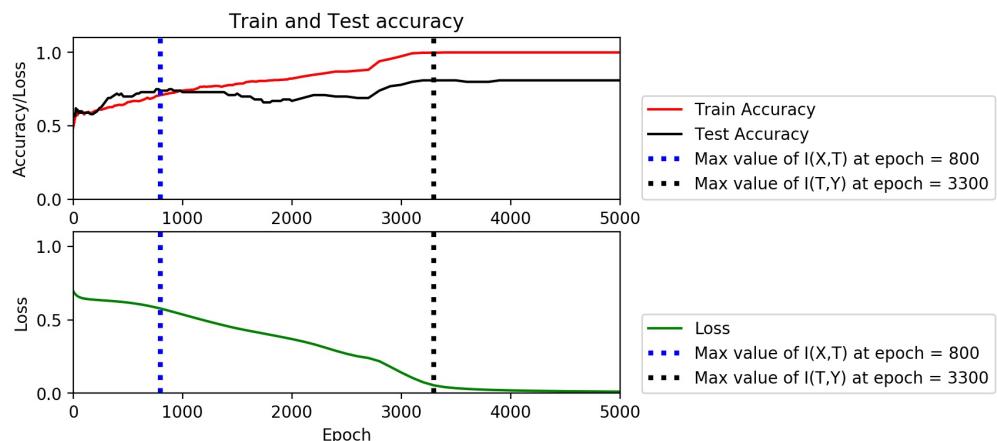
(b) Configurazione 2 Layers. Configurazione Neuroni: 10-7.



(c) Configurazione 3 Layers. Configurazione Neuroni: 10-7-5.



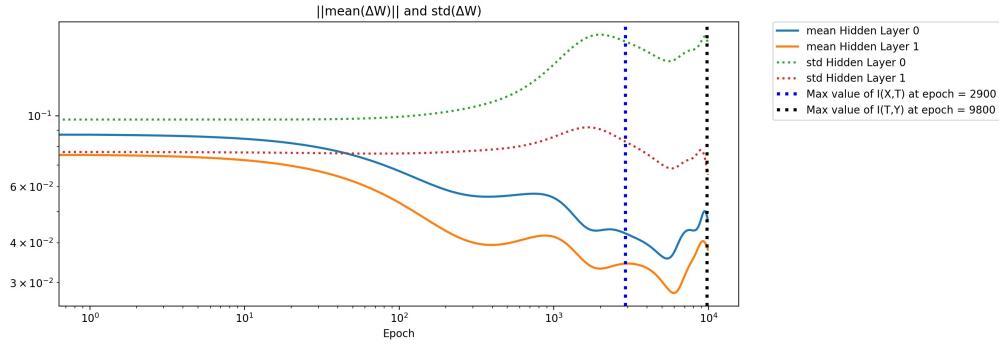
(a) Configurazione 4 Layers. Configurazione Neuroni: 10-7-5-4.



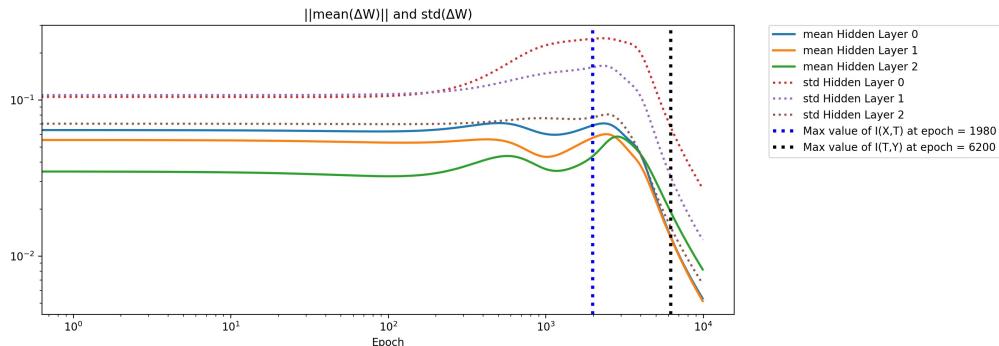
(b) Configurazione 4 Layers. Configurazione Neuroni: 10-7-5-4-3.

Figura A.6: Andamento funzione costo al variare dei layer. Learning rate=0.001, batch size=512, n.bins=30.

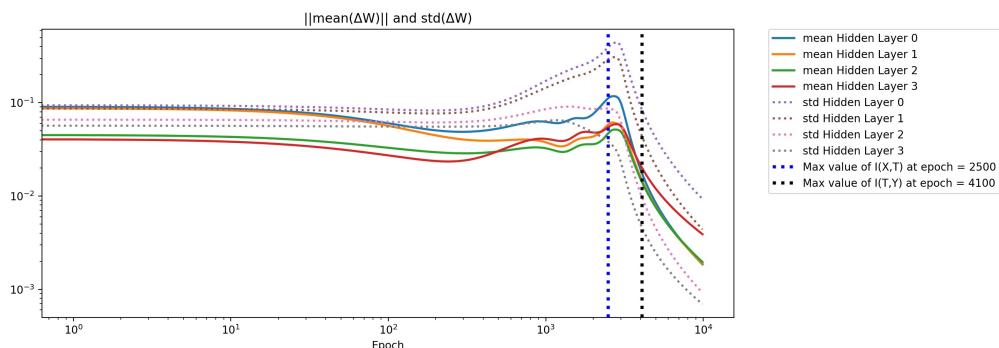
Si giunge ad una conclusione diversa da quanto affermato da Tishby [45]. Il tratto di colore nero presente in Figura A.8, è il punto in cui si raggiunge il massimo valore di $I(T,Y)$ per l'ultimo hidden layer. Come si può vedere dalla Figura 3.13, dopo aver raggiunto il massimo valore di $I(T,Y)$ l'ultimo layer inizia a comprimere le informazioni dell' ingresso, però in tale fase è associato un alto **SNR**, indicando che non può essere il rumore introdotto dal processo di aggiornamento dei pesi il motivo della compressione.



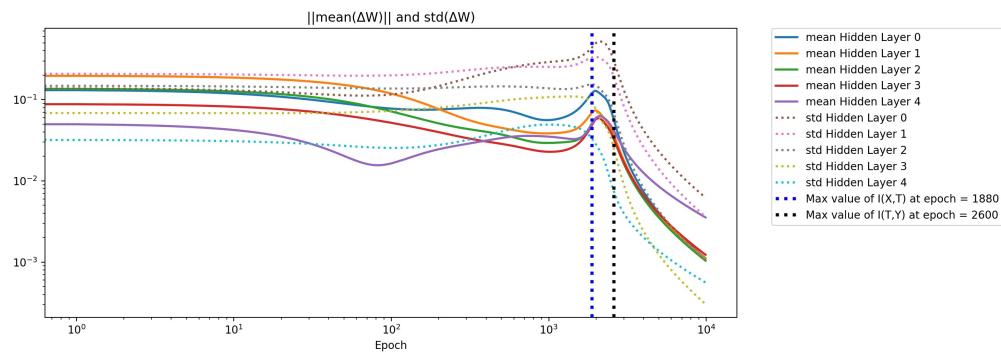
(a) Configurazione 1. Layer Configurazione Neuroni: 10.



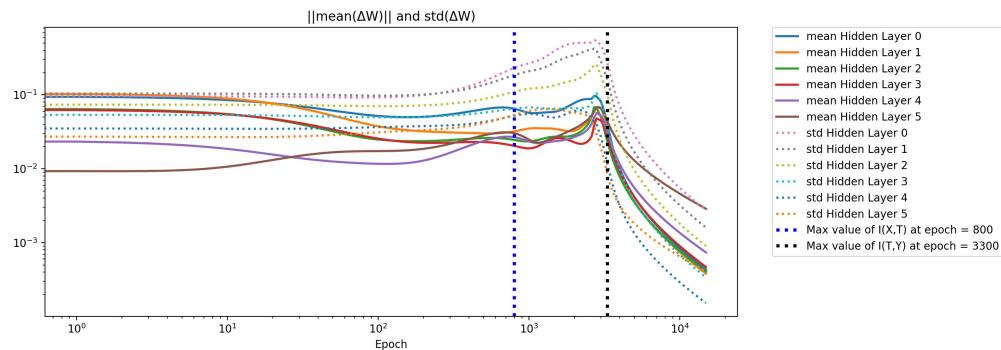
(b) Configurazione 2. Layers Configurazione Neuroni: 10-7.



(c) Configurazione 3. Layers Configurazione Neuroni: 10-7-5.

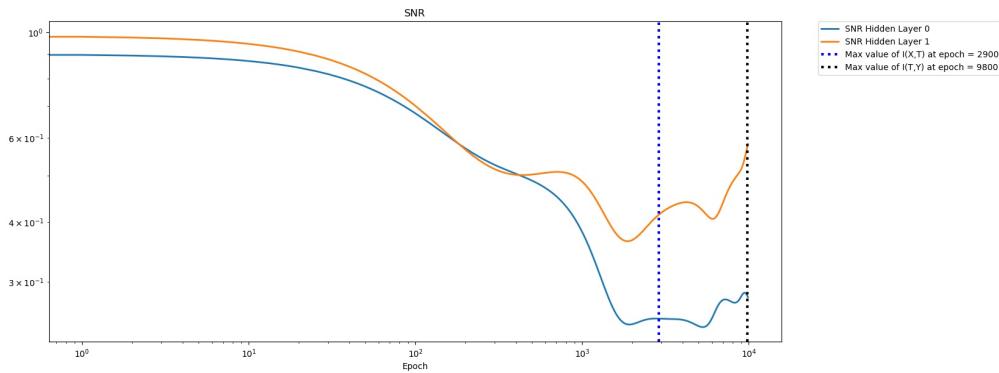


(a) Configurazione 4. Layers Configurazione Neuroni: 10-7-5-4.

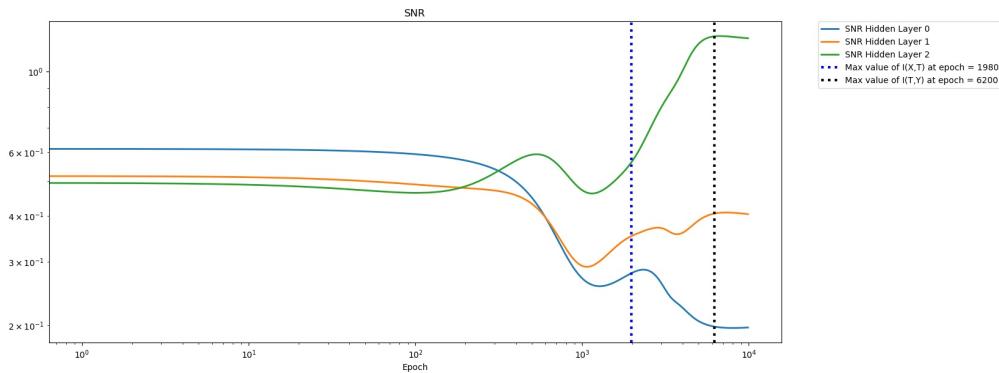


(b) Configurazione 4. Layers Configurazione Neuroni: 10-7-5-4-3.

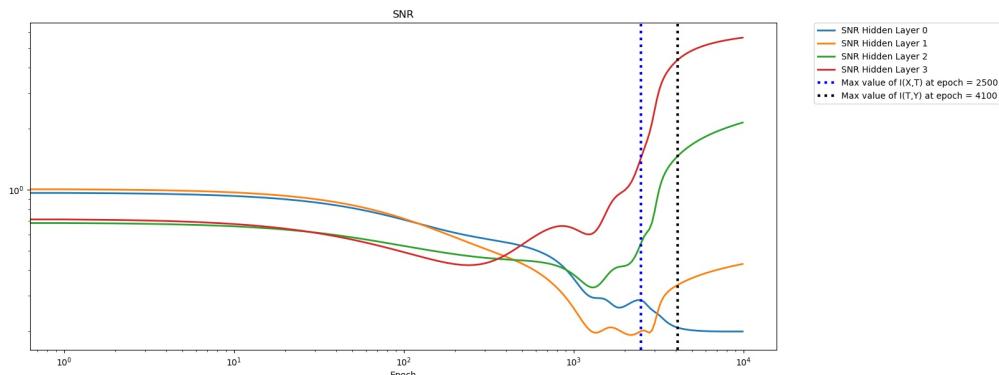
Figura A.7: Andamento media e varianza del gradiente calcolato rispetto W al variare degli Hidden layer. Learning rate=0.001, batch size=512, n.bins=30.



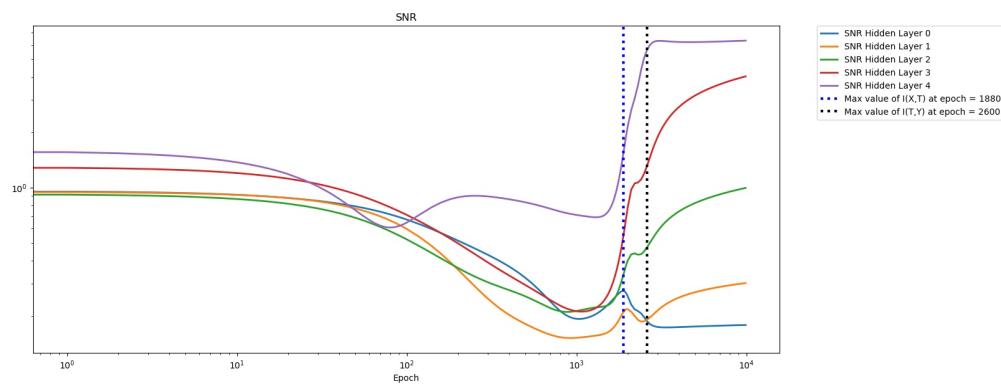
(a) Configurazione 1 Layer Configurazione Neuroni: 10.



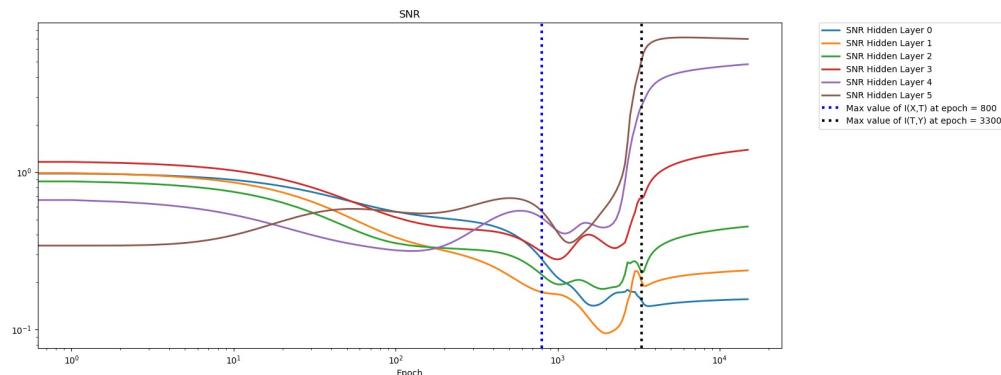
(b) Configurazione 2 Layers Configurazione Neuroni: 10-7.



(c) Configurazione 3 Layers Configurazione Neuroni: 10-7-5.



(a) Configurazione 4 Layers Configurazione Neuroni: 10-7-5-4.



(b) Configurazione 4 Layers Configurazione Neuroni: 10-7-5-4-3.

Figura A.8: Andamento SNR gradiente rispetto W al variare degli Hidden layers.
Learning rate=0.001, batch size=512, n.bins=30.

A.7 Variazione grandezza dataset

Per poter ottenere la Figura A.9, si è allenata la rete con una percentuale variabile del train dataset e si è andato a calcolare la traiettoria dell'informazione sul dataset di test. Si può notare come la traiettoria dell'informazione del test dataset aumenta quando si usa un dataset di train più grande, questo in accordo con quanto affermato da Tishby [45].

I punti in cui l'informazione converge, si trovano tutti in corrispondenza della medesima linea verticale, anche se in istanti temporali diversi, secondo quanto affermato da Tishby il motivo è dettato dall'**IB Curve**. Gli effetti sono diversi in base al layer considerato, infatti la traiettoria dei layer più vicini all'ingresso cambia in maniera poco significativa al variare della grandezza del dataset, a differenza dei layer più profondi. In quest'ultimo caso infatti all'aumentare del dataset di train, la rete memorizza più informazioni sulle etichette Y e impara a comprimere meglio le informazioni irrilevanti di X.

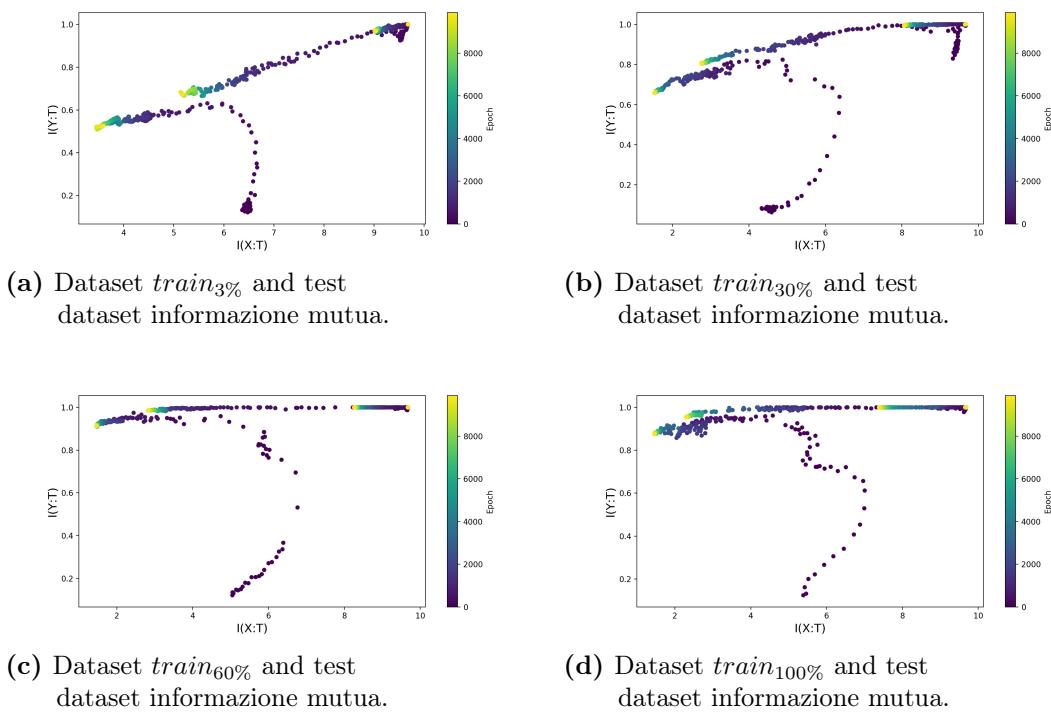


Figura A.9: Traiettoria dell'informazione del test dataset al variare del train dataset. Learning rate=0.01, batch size=128, funzione di attivazione=tanh, n.bins=30. In questo caso di è utilizzato il dataset originale fornito da Tishby, allo scopo di confrontare i risultati.

Si può quindi affermare che, utilizzando un dataset più grande, la rete generalizza in maniera migliore, ed impara a comprimere meglio le informazioni. Questo risulta essere in accordo con la necessità delle Neural Network (**NN**) di ampliare la dimensione del dataset in ingresso allo scopo di evitare l'overfitting (anche con algoritmi di elaborazione di immagini, come la rotazione [31]). All'overfitting può quindi essere associato una traiettoria convessa nel piano dell'informazione.

Acronimi

DNN	Deep Neural Network
MLP	Multi Layer Perceptron
SNR	rapporto segnale rumore
IB	Information Bottleneck
RNN	Recurrent Neural Network
CNN	Convolution Neural Network
GD	Gradient Descent
SGD	Stochastic Gradient Descent
MGD	Mini-batch Gradient Descent
PCA	Principal component analysis
DPI	Data Processing Inequality
SVM	Support Vector Machine
KDE	Kernel Density Estimation
ERM	Empirical Error Minimization
NN	Neural Network
MSS	Minimal sufficient statistics
IB Curve	Information Bottleneck Curve
LDPC	Low Density Parity-check Codes

Bibliografia

- [1] Alessandro Achille e Stefano Soatto. *Information Dropout: Learning Optimal Representations Through Noisy Computation*. 2016. arXiv: [1611.01353 \[stat.ML\]](#).
- [2] Guillaume Alain e Yoshua Bengio. *Understanding intermediate layers using linear classifier probes*. 2016. arXiv: [1610.01644 \[stat.ML\]](#).
- [3] Alexander A. Alemi et al. «Deep Variational Information Bottleneck». In: *CoRR* abs/1612.00410 (2016). arXiv: [1612.00410](#). URL: <http://arxiv.org/abs/1612.00410>.
- [4] Rana Ali Amjad e Bernhard Claus Geiger. «Learning Representations for Neural Network-Based Classification Using the Information Bottleneck Principle». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019), 1–1. ISSN: 1939-3539. DOI: [10.1109/tpami.2019.2909031](#). URL: <http://dx.doi.org/10.1109/TPAMI.2019.2909031>.
- [5] S. Arimoto. «An algorithm for computing the capacity of arbitrary discrete memoryless channels». In: *IEEE Transactions on Information Theory* 18.1 (1972), pp. 14–20. ISSN: 1557-9654. DOI: [10.1109/TIT.1972.1054753](#).
- [6] Devansh Arpit et al. *A Closer Look at Memorization in Deep Networks*. 2017. arXiv: [1706.05394 \[stat.ML\]](#).
- [7] Samy Bengio et al. *Understanding deep learning requires rethinking generalization*. 2016. arXiv: [1611.03530 \[cs.LG\]](#).
- [8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.
- [9] R. Blahut. «Computation of channel capacity and rate-distortion functions». In: *IEEE Transactions on Information Theory* 18.4 (1972), pp. 460–473. ISSN: 1557-9654. DOI: [10.1109/TIT.1972.1054855](#).
- [10] S.Kartik Buddha et al. «Function Identification in Neuron Populations via Information Bottleneck». In: *Entropy [electronic only]* 5 (mag. 2013). DOI: [10.3390/e15051587](#).
- [11] Yuan Cao e Quanquan Gu. «A Generalization Theory of Gradient Descent for Learning Over-parameterized Deep ReLU Networks». In: *CoRR* abs/1902.01384 (2019). arXiv: [1902.01384](#). URL: <http://arxiv.org/abs/1902.01384>.

- [12] Jerry Chee e Panos Toulis. *Convergence diagnostics for stochastic gradient descent with constant step size*. 2017. arXiv: [1710.06382 \[stat.ML\]](https://arxiv.org/abs/1710.06382).
- [13] Francois Chollet. *Deep Learning with Python*. 1st. USA: Manning Publications Co., 2017. ISBN: 1617294438.
- [14] D. Ciregan, U. Meier e J. Schmidhuber. «Multi-column deep neural networks for image classification». In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3642–3649. DOI: [10.1109/CVPR.2012.6248110](https://doi.org/10.1109/CVPR.2012.6248110).
- [15] Thomas M. Cover e Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. USA: Wiley-Interscience, 2006. ISBN: 0471241954.
- [16] J. Deng et al. «ImageNet: A Large-Scale Hierarchical Image Database». In: *CVPR09*. 2009.
- [17] Borja Rodriguez Galvez. *The Information Bottleneck, connections to other problems, learning and exploration of the IB curve*. KTH royal Insitute of technology school of electrical engineering e computer science, 2019.
- [18] Stuart Geman e Donald Geman. «Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-6* (1984), pp. 721–741.
- [19] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN: 0262035618.
- [20] Hassan Hafez-Kolahi e Shohreh Kasaei. *Information Bottleneck and its Applications in Deep Learning*.
- [21] Kaiming He et al. «Deep Residual Learning for Image Recognition». In: *CoRR* abs/1512.03385 (2015). arXiv: [1512.03385](https://arxiv.org/abs/1512.03385). URL: <http://arxiv.org/abs/1512.03385>.
- [22] Cian O'Donnell Ivan Chelombiev Conor Houghton. *Adaptive Estimators Show Information Compression in Deep Neural Networks*. 2019. arXiv: [1902.09037 \[cs.LG\]](https://arxiv.org/abs/1902.09037).
- [23] Bo Jin e Xinghua Lu. «Identifying informative subsets of the Gene Ontology with information bottleneck methods». In: *Bioinformatics (Oxford, England)* 26 (ott. 2010), pp. 2445–51. DOI: [10.1093/bioinformatics/btq449](https://doi.org/10.1093/bioinformatics/btq449).
- [24] Artemy Kolchinsky e Brendan Tracey. «Estimating Mixture Entropy with Pairwise Distances». In: *Entropy* 19.7 (2017), p. 361. ISSN: 1099-4300. DOI: [10.3390/e19070361](https://doi.org/10.3390/e19070361). URL: <http://dx.doi.org/10.3390/e19070361>.
- [25] Artemy Kolchinsky, Brendan D. Tracey e David H. Wolpert. «Nonlinear Information Bottleneck». In: *CoRR* abs/1705.02436 (2017). arXiv: [1705.02436](https://arxiv.org/abs/1705.02436). URL: <http://arxiv.org/abs/1705.02436>.
- [26] Alex Krizhevsky, Vinod Nair e Geoffrey Hinton. «CIFAR-10 (Canadian Institute for Advanced Research)». In: (2017). URL: <http://www.cs.toronto.edu/~kriz/cifar>.

- [27] Daniel Kunin, Mansheej Paul e Matthew Bull. «Message in a bottle: learning dynamics in the information plane». In: (2017). URL: <http://github.com/danielkunin/Message-in-a-Bottle>.
- [28] Yann LeCun e Corinna Cortes. «MNIST handwritten digit database». In: (2010). URL: <http://yann.lecun.com/exdb/mnist/>.
- [29] Feiyang Liu. *Implementation and verification of the Information Bottleneck interpretation of deep neural networks*. kth royal institue of technology, 2018.
- [30] Noboru Murata. *A Statistical Study on On-line Learning*. 1998, Waseda University, RIKEN Brain Science Institute.
- [31] Michael Nielsen. *neuralnetworksanddeeplearning*. URL: <http://neuralnetworksanddeeplearning.com/>.
- [32] OpenReview. *Openreview: On the Information Bottleneck Theory of Deep Learning*. URL: https://openreview.net/forum?id=ry_WPG-A-.
- [33] A. Saxe et al. «On the Information Bottleneck Theory of Deep Learning». In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=ry_WPG-A-.
- [34] Nicol Norbert Schraudolph. «Optimization of Entropy with Neural Networks». In: (1995, University of California, San Diego).
- [35] Claude Elwood Shannon. «A mathematical theory of communication». In: *Bell system technical journal* 27.3 (1948), pp. 379–423.
- [36] Noam Slonim. *The Information Bottleneck: Theory and Applications*. Hebrew University, 2002.
- [37] Noam Slonim e Naftali Tishby. «Agglomerative Information Bottleneck». In: *Neural Information Processing Systems* 12 (nov. 2000).
- [38] Noam Slonim e Naftali Tishby. «The Power of Word Clusters for Text Classification». In: 2006.
- [39] Samuel L. Smith e Quoc V. Le. *A Bayesian Perspective on Generalization and Stochastic Gradient Descent*. 2017. arXiv: [1710.06451 \[cs.LG\]](https://arxiv.org/abs/1710.06451).
- [40] Maximilian Stark. *Information Optimum Design of Discrete LDPC Decoders for Irregular Codes*. University of Hamburg-Harburg, 2017.
- [41] Susanne Still. «Information Bottleneck Approach to Predictive Inference». In: *Entropy* 16 (gen. 2014). DOI: [10.3390/e16020968](https://doi.org/10.3390/e16020968).
- [42] D J Strouse e David J Schwab. «The Deterministic Information Bottleneck». In: *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*. UAI'16. Jersey City, New Jersey, USA: AUAI Press, 2016, 696–705. ISBN: 9780996643115.
- [43] Christian Szegedy et al. «Going Deeper with Convolutions». In: *CoRR* abs/1409.4842 (2014). arXiv: [1409.4842](https://arxiv.org/abs/1409.4842). URL: <http://arxiv.org/abs/1409.4842>.
- [44] Naftali Tishby, Fernando C. Pereira e William Bialek. *The information bottleneck method*. 2000. arXiv: [physics/0004057 \[physics.data-an\]](https://arxiv.org/abs/physics/0004057).

- [45] Naftali Tishby e Ravid Schwartz-Ziv. «Opening the black box of Deep Neural Networks via Information». In: (2017). eprint: <https://arxiv.org/pdf/1703.00810.pdf>.
- [46] Naftali Tishby e Noga Zaslavsky. *Deep Learning and the Information Bottleneck Principle*. 2015. arXiv: [1503.02406 \[cs.LG\]](https://arxiv.org/abs/1503.02406).
- [47] M. P. Wand. «Data-Based Choice of Histogram Bin Width». In: *The American Statistician* 51.1 (1997), pp. 59–64. ISSN: 00031305. URL: <http://www.jstor.org/stable/2684697>.
- [48] Wikipedia. *AlexNet*. URL: <https://en.wikipedia.org/wiki/AlexNet>.
- [49] Wikipedia. *AlphaGo*. URL: <https://it.wikipedia.org/wiki/AlphaGo>.
- [50] Wikipedia. *Histogram*. URL: <https://en.wikipedia.org/wiki/Histogram>.
- [51] Lei Wu, Zhanxing Zhu e Weinan E. «Towards Understanding Generalization of Deep Learning: Perspective of Loss Landscapes». In: *CoRR* abs/1706.10239 (2017). arXiv: [1706.10239](https://arxiv.org/abs/1706.10239). URL: [http://arxiv.org/abs/1706.10239](https://arxiv.org/abs/1706.10239).
- [52] Aston Zhang et al. *Dive into Deep Learning*. <http://www.d2l.ai>. 2019.
- [53] Tianchen Zhao. «Information Theoretic Interpretation of Deep learning». In: *CoRR* abs/1803.07980 (2018). arXiv: [1803.07980](https://arxiv.org/abs/1803.07980). URL: [http://arxiv.org/abs/1803.07980](https://arxiv.org/abs/1803.07980).