Università degli Studi di Padova

Dipartimento di Scienze statistiche

Corso di Laurea Triennale in
Statistica per le Tecnologie e le Scienze

Deep Learning e compressione dati

Relatore:
Prof. Massimo Melucci

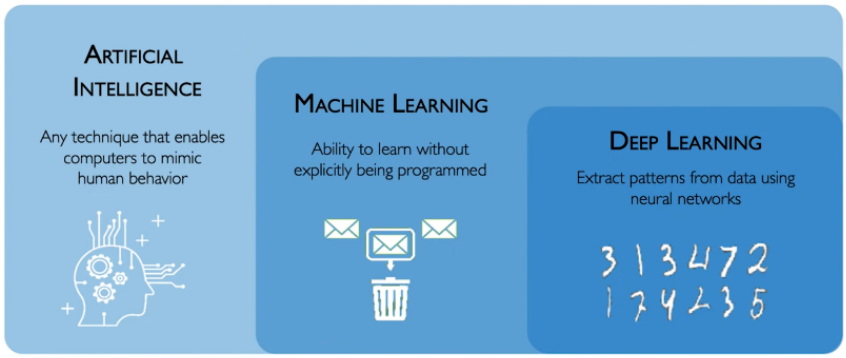
Laureando:
Federico Bottarelli
1198049

Anno Accademico 2021/2022

Indice

1 Introduzione

L'argomento di cui tratta la tesi è l'applicazione di una rete neurale artificiale (Artificial Neural Network, ANN) nella compressione dati di un testo scritto. Le tecniche di compressione dati nell'ambito informatico hanno avuto un ruolo molto importante nella rivoluzione digitale avvenuta negli ultimi decenni, senza di esse moltissimi servizi che vengono usati oggi sarebbero impossibili. Il modello oggetto della tesi è in grado, se allenato bene, di ottenere una compressione maggiore rispetto ai metodi tipicamente usati ora. Mediamente queste tecniche di compressione, applicate ad un file di testo, ottengono una dimensione di $\frac{1}{3}$ quella originale, con il modello utilizzato si ottiene quasi $\frac{1}{5}$.
La tesi è divisa in due parti, nella prima si spiega il funzionamento delle reti neurali artificiali mentre nella seconda parte viene prima introdotto il contesto della compressione dati in ambito informatico, per poi spiegare il metodo di compressione oggetto della tesi in modo da esporne potenzialità e aspetti negativi.



2 Intelligenze artificiali, machine learning e deep learning

Per deep learning (apprendimento profondo) si intende una sotto categoria di modelli del machine learning (apprendimento automatico), che a loro volta fanno parte dell'ampio mondo delle intelligenze artificiali (AI).
Queste tecnologie sono utilizzate da miliardi di persone. Chiunque abbia uno smartphone in tasca può sperimentare i progressi di queste tecniche, un esempio sono i sistemi di assistenza vocale, sistemi di raccomandazione di varie piattaforme che ci consigliano brani o video secondo i nostri gusti, riconoscimento facciale per lo sblocco di alcuni telefoni o per i filtri usati nei social media e molte altre applicazioni. L'intelligenza artificiale viene applicata in molti ambiti diversi: medicina, astronomia, scienza dei materiali, sicurezza informatica ecc. [1]

2.1 Contesto e storia

2.1.1 Intelligenza artificiale

Non esiste una definizione unica di intelligenza artificiale, tutte si basano sul concetto che una macchina possa agire o pensare in modo razionale o addirittura "umano". Due tra le definizione più conosciute sono:
"[L'automazione delle] attività che si associa con il pensiero umano, come il processo decisionale, la risoluzione di problemi, l'apprendimento . . ."
(Bellman, 1978) [2]
"L'intelligenza computazionale è lo studio della progettazione di agenti intelligenti." (Poole et al., 1998) [3]

Uno dei primi studiosi fu Alan Turing, tra i più grandi matematici del XX secolo, padre dell'informatica. Turing si domandò se le macchine come gli uomini non potessero prendere decisioni sulla base delle informazioni disponibili. Da questo quesito, nel 1950, scrisse un articolo, *Computing Machinery and Intelligence* [4] in cui, si parla del test di turing e delle macchine di Turing, ipotizzava una macchina di uso generale ma dovesse essere programmata manualmente in ogni suo passaggio; questo paradigma è stato il principale nel campo delle intelligenze artificiali fino agli anni 80. I progressi della teoria della computabilità, ispirarono una parte dei ricercatori a sviluppare modelli computazionali basati sulla cognizione umana, questo diede le basi già negli anni 40 alle tecniche di machine learning.

2.1.2 Machine learning

Nei modelli di machine learning il sistema è addestrato piuttosto che programmato in modo esplicito. Gli vengono presentati molti esempi rilevanti per un'attività e trova una struttura statistica che gli consente di elaborare regole per automatizzare il compito. Una delle definizioni più citate di machine learning è la seguente di Tom Mitchell:
Si dice che un programma per computer impari dall'esperienza E rispetto a qualche classe di compiti T e misura la prestazione P se la sua prestazione in compiti in T, misurata da P, migliora con l'esperienza E. [5]
Se, per esempio, si volesse avere una algoritmo in gradi di riconoscere se in una immagine il soggetto principale è un cane o un gatto, basterebbe dargli molti esempi di foto di cani o gatti con assegnata l'etichetta corretta (da una persona) e il sistema imparerebbe le regole statistiche per associare immagini specifiche alle etichette corrette.
L'allenamento del modello è possibile grazie ad una serie di parametri che vengono ottimizzati secondo una criterio (funzione di perdita) che ci dà un idea dell'andamento del modello e ci permette di vedere se è possibile migliorarlo.
Questi modelli possono essere classificati in varie categorie caratterizzate dal tipo di feedback che viene dato per l'allenamento: nell'allenamento supervisionato vengono passati al modello una serie di dati a cui è legato un etichetta che indica la corretta risposta per quell'input specifico, gli vengono così dati degli esempi da cui imparare e su ottimizzare i parametri.
Nell'allenamento non supervisionato vengono passati al modello i dati, senza etichetta corretta, si cerca di apprendere una qualche struttura o pattern già

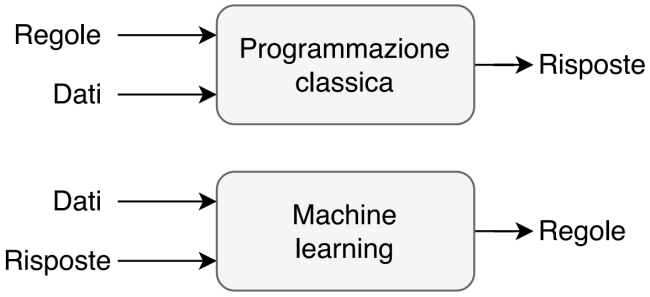


Figura 1: Differenza tra programmazione classica e machine learning

presente nei dati, utile modellare i dati o riderne la dimensionalità per poi modellare più facilmente. I modelli supervisionati da umani spesso riescono ad ottenere ottimi risultati in svariati compiti ma richiedono un grande importo di lavoro umano per avere i dati con cui allenare il modello.

2.1.3 Neuroni artificiali

Parallelamente allo studio delle intelligenze artificiali nell'ambito informatico vi furono parte di studiosi che si concentrarono sull'idea di imitare il funzionamento del cervello umano e di come apprende. Tra i primi Warren McCulloch e Walter Pitts, quest'ultimo, nel 1943 affermò che i neuroni biologici potessero essere descritti come dispositivi computazionali. In un certo senso, questa è un tentativo di descrivere un dispositivo di calcolo di uso generale (in grado di risolvere problemi di diversa natura). Ispirato al funzionamento dei neuroni biologici, e con un'architettura significativamente diversa dalle macchine di Turing. In un articolo pubblicato nello stesso anno si descrive per la prima volta il neurone artificiale. [6]
Il nostro sistema nervoso apprende tramite una rete di neuroni. Prima la rete riceve, in input, un segnale da elaborare generato dall'interazione della persona con il mondo esterno tramite i sensi. Il segnale poi viene passato ed elaborato nella rete di neuroni generando poi un output, cioè una risposta agli stimoli. Questo procedimento, avviene continuamente nei neuroni, ed è stato rappresentato in modo molto semplificato rispetto alla realtà, da McCulloch e Pitts con il loro neurone artificiale, in grado di elaborare delle informazioni binarie per costruire logiche booleane. Il singolo neurone consisteva in un semplice modello lineare che prende in input le varie informazioni sotto forma di dati binari :

$$x_1, x_2, \ldots, x_i, \ldots, x_n \quad \text{con} \quad x_i \in \{0, 1\}$$

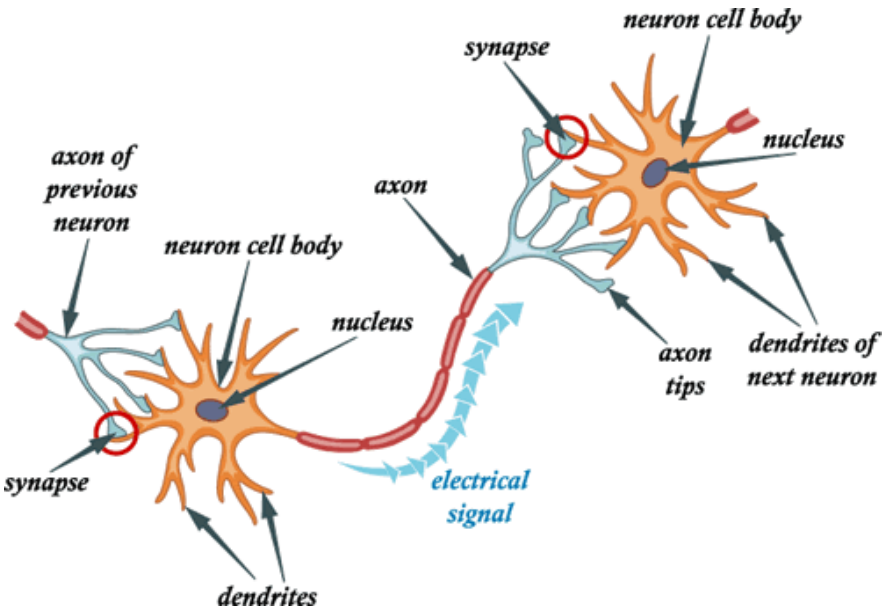


Figura 2: Neurone e sinapsi di un sistema nervoso umano

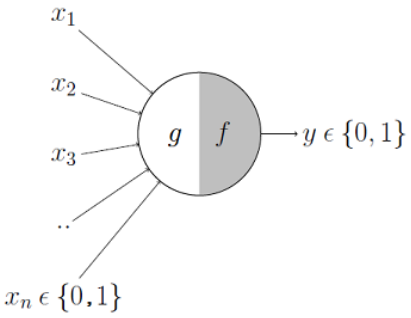


Figura 3: Neurone di McCulloch e Pitts

Vengono aggregati tramite una funzione g, per esempio la somma:

$$g(x_1, x_2, \ldots, x_i, \ldots, x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

Poi il valore risultante $g(\mathbf{x})$ viene passato ad un'altra funzione, detta di attivazione, che ha il compito di prendere la decisione sulla base del valore che riceve. Nel caso più semplice f trasforma l'input in una risposta binaria in base ad una soglia predefinita chiamata γ,

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad \text{if } g(\mathbf{x}) \geq \gamma \\ &= 0 \quad \text{if } g(\mathbf{x}) < \gamma \end{aligned}$$

Il neurone è quindi diviso in due parti e scegliendo il parametro di soglia

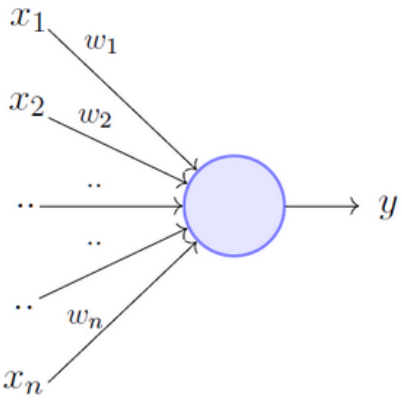


Figura 4: Perceptron di Rosenblatt

adeguato si può usare per rappresentare varie funzioni booleane. L'output del neurone può essere usato come input per altri neuroni in modo da costruire una logica più complessa, in grado di calcolare diverse funzioni logiche e per fare questo la rete deve essere specificata in tutte le sue parti per essere utilizzata. Non ci sono parametri da aggiustare per adattarla ad un problema diverso. L'apprendimento può essere fatto solo modificando la struttura stessa della rete e i collegamenti fra i nodi.

Perceptron Grazie agli studi sui neuroni artificiali di McCulloch e Pitts e alla teoria Hebbiana, nel 1957 Frank Rosenblatt nella sua pubblicazione "The Perceptron: A Perceiving and Recognizing Automaton" definisce il Perceptron [7], il primo vero esempio di machine learning ipotizzato.
Ha due principali differenze rispetto al neurone di MaCulloch e Pitts. La prima è che accetta in input valori reali, non solo binari; la seconda è che ogni input ha un peso per cui viene moltiplicato. Questi pesi vengono usati come parametri per ottenere un modello più flessibile.
Dati in input:

$$\{x_1, x_2, x_3, \ldots, x_n\}$$

Pesi:.

$$\{w_1, w_2, w_3, \ldots, w_n\}$$

b peso da sommare (bias)

$$g\left[(x_1 w_1, x_2 w_2, x_3 w_3, \ldots, x_n w_n)\right] + b) = g((\mathbf{x}, \mathbf{w}) + \mathbf{b}) = g(b + \sum_{i=1}^{n} x_i w_i)$$

$g(\mathbf{x}, \mathbf{w})$ viene passato ad un'altra funzione f che ha il compito di prendere una decisione basata sul valore che riceve; nel caso più semplice trasforma il valore finale in una risposta binaria in base ad una soglia scelta γ e viene definita funzione di attivazione.

$$\begin{aligned} y = f(g((\mathbf{x}, \mathbf{w}) + \mathbf{b})) = 1 \quad & \text{if } g((\mathbf{x}, \mathbf{w}) + \mathbf{b}) \geq \gamma \\ = 0 \quad & \text{if } g((\mathbf{x}, \mathbf{w}) + \mathbf{b}) < \gamma \end{aligned}$$

γ parametro di soglia
Nel contesto delmachine learning un algoritmo di questo tipo può essere usato come classificatore lineare. L'idea, già ipotizzata in passato, di una macchina computazionale in grado di risolvere problemi di diversa natura qui vedeva una sua implementazione teorica che influenzerà per decenni la ricerca in questo campo; bisognerà però aspettare più di quarant'anni e la diffusione delle unità di calcolo grafiche (GPU) per poterne vedere sfruttate di più le potenzialità.

2.1.4 Dal neurone alle reti neurali

Nel 1949 Donald Hebb, un neuroscienzato Canadese, nel suo libro *The Organization of Behavior* [8], espone la prima teoria in cui spiega che quando più neuroni sono collegati fra loro amplificano la propria capacità di elaborare le informazioni e apprendere. Questo non solo influenzò il modo di vedere l'apprendimento biologico umano ma influenzò anche lo sviluppo di modelli matematici e computazionali più complessi che imitavano i sistemi biologici. Ispirandosi alla teoria Hebbiana, collegando più neuroni artificiali fra loro, si possono ottenere macchine computazionali più complesse che risolvono problemi elaborati. Dal singolo Perceptron si svilupparono vari modelli matematici ed ottener le reti neurali artificiali. [9]

2.1.5 Primi modelli di deep learning

Con le pubblicazioni di Henry J. Kelley, *"Gradient Theory of Optimal Flight Paths"* [10] del 1960 e Stuart Dreyfus con *"The numerical solution of variational problems"* [11] nel 1962 viene definito l'algoritmo di allenamento chiamato "backpropagation" che sfrutta la chain rule, ancora oggi insieme alle sue varianti, è lo standard per ottimizzare la maggior parte di modelli ANN. Tre anni più tardi Alexey Grigoryevich Ivakhnenko insieme a Valentin Grigorevich Lapa, idearono la prima rappresentazione gerarchica di una rete neurale che

sfruttava anche delle funzioni di attivazione per i nodi. Oggi Ivakhnenko viene considerato il vero padre del Deep learning.
Negli anni a seguire vennero pubblicati molti articoli che svilupparono modelli rendendoli sempre più complessi. La scarsa potenza computazionale rendeva, però, ancora difficile e lontana la loro effettiva implementazione e utilità. Solo più recentemente, con lo sviluppo delle unità di calcolo grafiche e compresa la loro potenzialità, si ottennero i primi importanti risultati applicativi di questi modelli. Queste tempistiche si spiegano perché le i modelli di deep learning hanno un elevatissimo costo computazionale. [12] Le tecniche di deep learning incominciaro ad avere un utilizzo diffuso nelle gare di riconoscimento d'immagine, dove i partecipanti competono per classificare e rilevare correttamente oggetti e scene dentro immagini. ImageNet fu un importante database di immagini correttamente classificate e diede un grande contributo per lo sviluppo del deep learning organizzando queste competizioni. Nel 2012 Alex Krizhevsky proprio al concorso annuale di algoritmi di riconoscimento d'immagini organizzato da ImageNet, usando un modello di deep learning allenato tramite l'utilizzo di processori grafici, riuscì a vincere il contest con un top-5-score [1] del 15.3% più del 10% basso rispetto al secondo classificato, AlexNet, nome dato a questo modello, vinse la gara per tre anni di fila per poi essere superato nel 2015 da un modello sviluppato da Microsoft con più di 100 layer. [13] [14]
Questo importante risultato fece comprendere alla comunità scientifica che i modelli di deep learning potevano ottenere risultati mai raggiunti prima in determinati problemi. Nel 2015 Facebook introduce un algoritmo in grado di riconoscere le facce delle persone nelle foto all'interno del suo social. Nel 2016 l'algoritmo AlphaGo riesce a battere a Go (gioco da tavolo diffuso nei paesi asiatici) i giocatori professionisti, risultato che fino ad allora non era mai stato raggiunto data la complessità del gioco. [15]
Non ci si può aspettare nell'immediato futuro modelli in grado di imitare veramente le funzioni del cervello umano in tutta la sua complessità, però per determinate mansioni a condizione di avere molti dati pre-lavorati si possono ottenere applicazioni interessanti in molti ambiti
Alcuni esempi di applicazioni delle reti neurali sono:

- Riconoscimento d'immagini

[1] Si controlla se l'etichetta del target è tra le prime 5 previsioni (le 5 con le probabilità più alte).

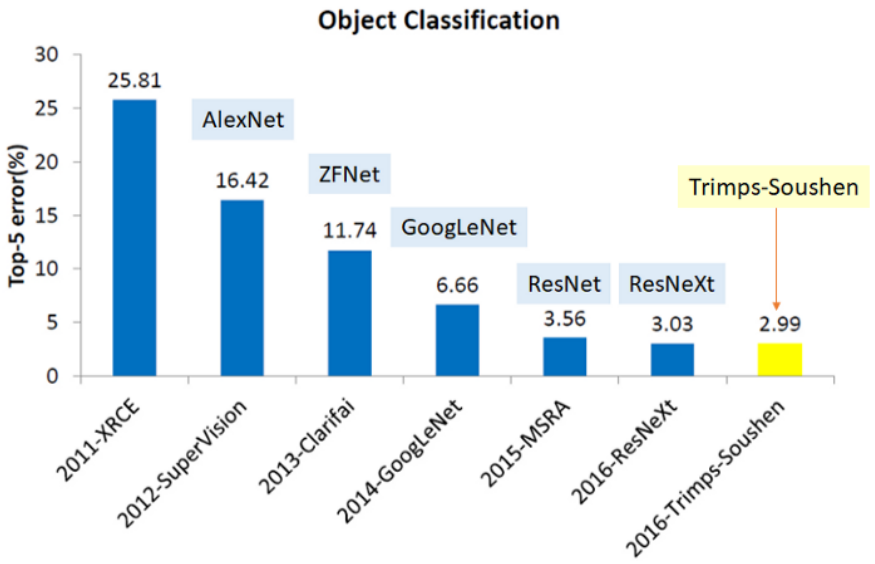


Figura 5: Errore di alcuni degli algoritmi di Deep Learning vintori di AlexNet tra il 2011 ed il 2016

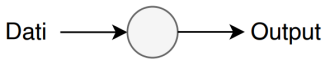


Figura 6: Rete più semplice possibile, corrispondente ad il Perceptron

- Verifiche di sicurezza e analisi
- Previsioni metereologiche
- Sistemi di guida automatica
- Compressione dati
- Molti altri...

2.2 Reti neurali Artificiali

2.2.1 Come funzionano le reti neurali?

Ora si mette in mostra più nel dettaglio il funzionamento delle strutture ANN. Il funzionamento è descrivibile graficamente tramite una struttura a grafo pesato dove ogni nodo rappresenta una unità di elaborazione dell'informazione e gli archi descrivono le operazioni fra esse. Nel caso più semplice (figura 6) si ha la rappresentazione già vista del Perceptron. Concatenando più neuroni tra loro si formano modelli più complessi, i dati in input passano prima per un nodo vengono trasformati e poi passano il nodo seguente. Ogni nodo riceve in ingresso l'output da uno o più nodi

Figura 7: Più nodi possono essere messi in sequenza, formando degli strati dall'input all'output

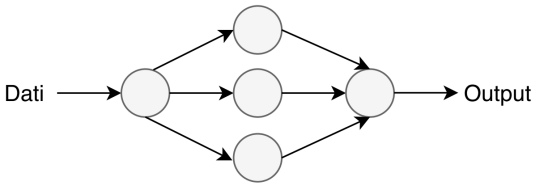


Figura 8: Si possono avere più nodi per strato

precedenti fatta eccezione per i nodi d'ingresso della rete per cui i dati vengono immessi dall'esterno della rete stessa. (figura 7) Si possono formare strati di neuroni dove l'output da un singolo neurone passa a molteplici neuroni messi in parallelo, ogni arco ha assegnato un peso differente e quindi aggiunge flessibilità al modello. (figura 8) . Possono esserci anche molteplici strati nascosti. (figura 9) Questa è la struttura generica di un modello di rete neurale artificiale. Un modello viene definito di Deep learning quando sono presenti un numero elevato di strati e di conseguenza un numero molto elevato di parametri. Si hanno usualmente tre tipologie di strati:

- Strati d'input: in cui ci sono i nodi dove i dati vengono immessi nella rete, ogni nodo rappresenta una feature della osservazione.

- Strati nascosti: dove le informazioni vengono elaborate e trasformate per permettere l'adattamento e l'apprendimento da parte del modello. Posso essere caratterizzati da varie funzioni e un numero variabile di strati e nodi.

- Strati d'output: nodi di uscita della rete che ci danno i risultati del processo di apprendimento. Nel caso di un solo nodo la risposta è binaria, con più nodi la risposta può essere più elaborata. Per esempio nel

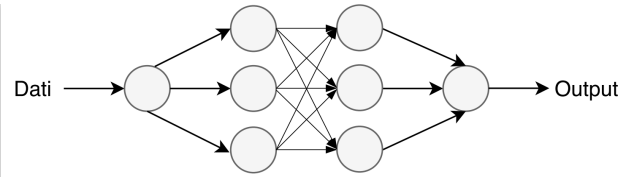


Figura 9: Si possono avere più nodi per strato

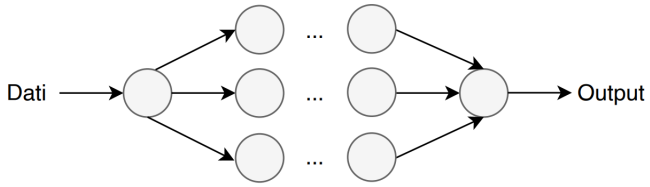


Figura 10: Struttura generica di un ANN

riconoscimento di un carattere, a partire da una immagine, si hanno più possibili risposte da valutare.

Le ANN possono essere di svariate forme diverse ed esistono molte tipologie diverse di architettura, per architettura della rete si intende semplicemente il modo in cui i neuroni della rete sono collegati fra loro. (in figura la 11 per vari esempi di architetture)

Nel caso delle reti neurali i parametri si trovano negli archi che formano i collegamenti fra i nodi, ad ogni passaggio tra due nodi al valore passato viene moltiplicato un peso e più un parametro di distorsione, questi due pesi per ogni arco del grafo sono i parametri che se modificati opportunamente permettono l'apprendimento tipico delle tecniche di machine learning. Le due tipologie di parametri usuali in una rete neurale sono:

Divisione del dataset in training, validation, test set Il dataset che si vuole usare per allenare deve essere opportunamente suddiviso in tre sottoinsiemi, uno per allenare (training set), uno per testare le performance del modello (testing set) per evitare di cadere nel "overfitting", un terzo sottoinsieme chiamato insieme di validazione (validation set) con lo scopo di verificare l'andamento del modello al variare degli iperparametri per scegliere il modello migliore da poi valutare nell'insieme test.

2.2.2 Architettura Feed-Forward

Si denota $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)\}$ il nostro dataset con i label composto di m instanze; la i-esima istanza è rappresentata da una tupla $(\mathbf{x}_i, y_i)$, dove:

$\mathbf{x}_i \in R^n$ è un vettore di caratteristiche n-dimensionale,

$$\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,n})^T$$

$y_i \in R$ y_1 ci indicano il risultato che ci si aspetta legato all'osservazione x_1

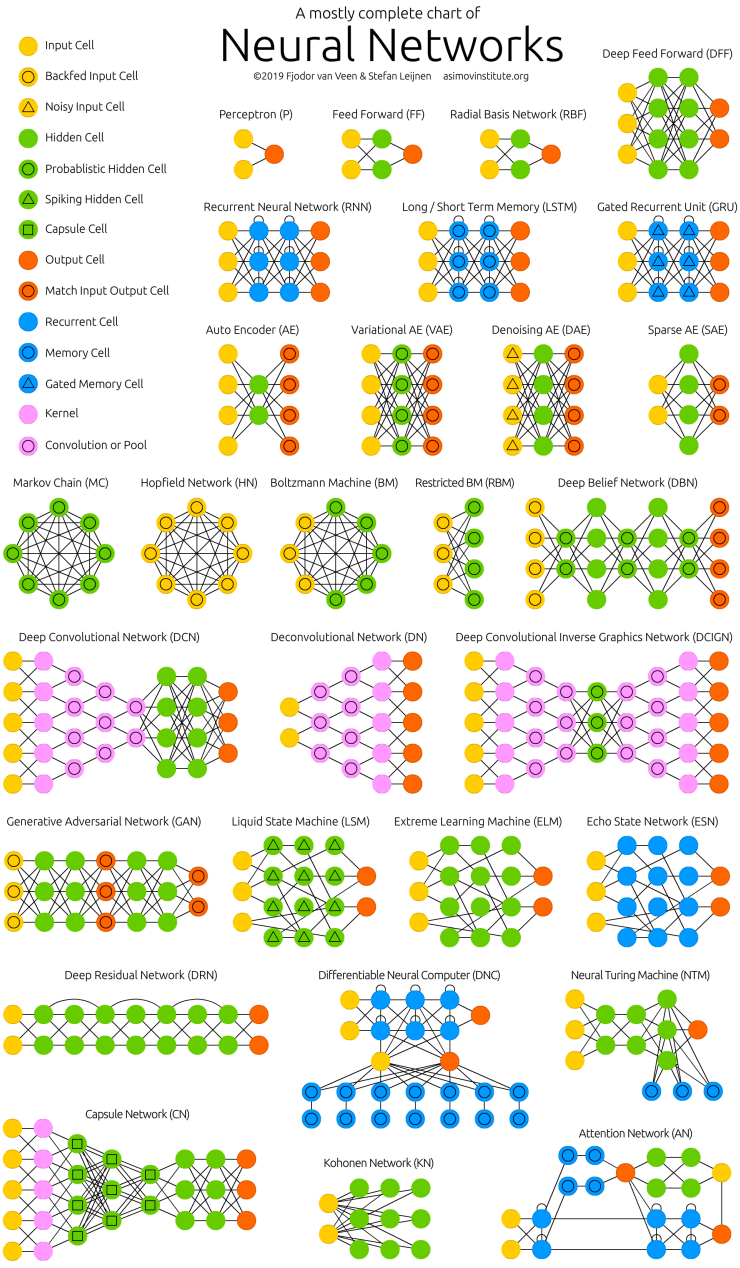


Figura 11: Diverse architetture di reti neurali, immagine di Fjodor van Veen "Neural Network Zoo"

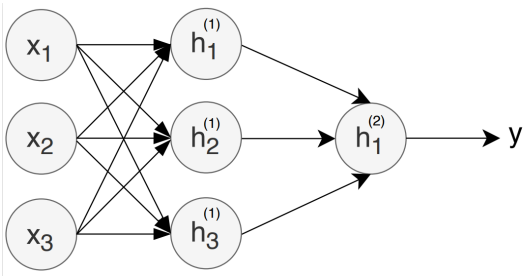


Figura 12: Rete neurale artificiale

Ad ogni iterazione dell'algoritmo gli input $\mathbf{x}_i$ entrano nella rete ed effettuano il passaggio chiamato propagazione in avanti (feed forward propagation), in cui vengono trasformati strato per strato così fino all'ultimo strato della rete: quello che genera gli output del modello. Ad ogni passaggio di strato i valori x_i in ingresso vengono moltiplicati e sommati con i pesi negli archi θ, entrano nei nodi successivi per poi ripetere il procedimento ad ogni strato proseguendo da sinistra verso destro nella lettura del grafo.
La procedura è scrivibile come combinazione lineare dei nodi in input:
$h_i^{(i)}$: Nodo nascosto i nello strato j
$\boldsymbol{\theta}^{(i)}$: matrice di pesi delle connessione tra lo strato j e lo strato $j+1$

$$
\begin{aligned}
h_l^{(2)} &= f\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right) \\
h_2^{(2)} &= f\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right) \\
h_3^{(2)} &= f\left(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3\right)
\end{aligned}
$$

$$
y_\theta(\mathbf{x}) = \gamma(h_1^{(3)}) = g\left(\theta_{10}^{(2)}h_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}h_2^{(2)} + \theta_{13}^{(2)}h_3^{(2)}\right)
$$

Con una notazione più leggera si può scrivere, generalizzando anche il caso di vettori e matrici al posto di singoli dati:

$$
\mathbf{h^{(1)}} = f\left(\mathbf{x}\right)
$$

$$
\mathbf{h^{(2)}} = g\left(\mathbf{h^{(1)}}\right)
$$

$$
\mathbf{y} = \gamma\left(\mathbf{h^{(2)}}\right) = \gamma\left(g(f(\mathbf{x})\right)
$$

Funzione di attivazione Le funzioni di attivazione che si vede scandisce gli strati della rete. Di solito sono funzioni non lineare con lo scopo di mappare i dati per avere un interpretazione sensata o per trasformali in modo da ottenere caratteristiche desiderate, per esempio l'outuput viene spesso adattato per

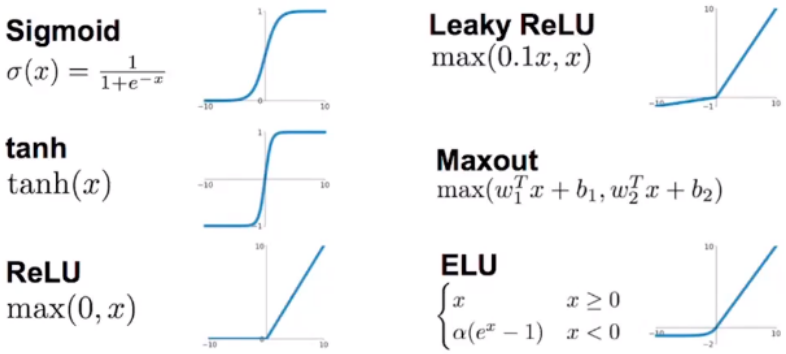


Figura 13: Alcune delle funzioni di attivazione più usate

poterne calcolare la funzione di perdita con esso. Le reti vengono ottimizzate una tecnica di discesa del gradiente e per questo bisogna scegliere funzioni derivabili e continue in ogni loro punto in modo da poter ripercorrere all'indietro con le derivate tutta la rete.
Ogni nodo ha una specifica funzione chiamata funzione di attivazione che viene applicata sulla somma pesata in input per generare l'output del nodo.

Progettazione di una rete neurale Per ogni modello ANN è possibile far variare molteplici iperparametri che ne modificano il funzionamento:

1. Quanti strati nascosti usare.
2. Numero nodi negli strati nascosti.
3. Come connettere i nodi tra gli strati della rete.
4. Come calcolare gli output finali del modello (in base al problema da risolvere).
5. Scegliere l'algoritmo con cui per ottimizzare i parametri della rete
6. Regolarizzazione per evitare overfitting.

2.2.3 Interconnessioni tra i nodi

Nel caso più generale possibile tutti i nodi di uno strato sono connessi a tutti i nodi dello strato successivo, questa struttura viene chiamata: fully connected
Altre opzioni sono:

- Random connected: si prendono m nodi dallo strato precedente

- Pooled: si dividono i nodi di un strato in cluster e poi per scegliere le connessioni nei nodi del elayer successivo si va a campionare nei vari cluster i nodi con cui fare il collegamento
- Convolutional : i nodi di un strato vengono messi su una griglia duo dimensionale ed un nodo indicizzato dai valori degli assi i e j andrà a collegarsi con i nodi dello strato precedente in un piccolo intorno di i e j, questo diminuisce drasticamente il costo computazionale delle iterazioni dell'algoritmo, necessario nel contesto di immagini.

2.2.4 Funzioni di perdita

La funzione di perdita dà una misura di quanto il modello stia sbagliando, in modo da derivare essa rispetto ai parametri del modello per modificarli nella direzione corretta. Le reti neurali artificiali possono svolgere molti compiti diversi e per questo possono avere diverse tipologie di output, in base ad esse si possono usare diverse funzioni. Nel caso della compressione di file di testo il problema può essere visto come una multi-classificazione, dove si hanno più classi $C_1, C_2, \ldots, C_n$ e si hanno n nodi nello strato d'output della rete. Questi nodi ci danno un vettore $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ dove p_i rappresenta la probabilità avendo x in input di avere come risultato la classe c_i, come per i parametri di una variabile multinomiale. Una funzione di perdita adatta può essere la cross-entropia che ci fornisce una distanza fra due distribuzioni di probabilità, quella stimata dalla rete p e quella prevista q.

$$H(\mathbf{p}, \mathbf{q}) = -\sum_{i=1}^{n} p_i \log q_i$$

L'entropia calcolata come:

$$H(\mathbf{p}) = -\sum_{i=1}^{n} p_i \log p_i$$

$$H(\mathbf{p}, \mathbf{p}) = H(\mathbf{p}), \text{ in generale } H(\mathbf{p}, \mathbf{q}) \geq H(\mathbf{p})$$

Il ruolo di questa misura è strettamente legata al contesto della compressione dati e più in generale con la teoria dell'informazione, per questo viene trattata più approfonditamente in seguito.

2.2.5 Back-propagation

L'algoritmo di ottimizzazione con cui si vuole minimizzare la funzione di perdita decisa attraverso la modifica dei parametri che si stanno stimando della rete: pesi e distorsioni. L'algoritmo più usato viene chiamato back-propagation e usa la tecnica del discesa del gradiente.

Discesa del gradiente Tecnica di ottimizzazione che modifica i parametri aggiornandoli nella direzione opposta al loro gradiente rispetto ad una funzione (di perdita) in modo da minimizzarla.

Si hanno 3 parametri:
θ parametro da ottimizzare
$\boldsymbol{\theta}_0 \in R^n$ il valore iniziale da cui parte
α step size, indica di quanto si muove nella direzione scelta il valore che ci sta ottimizzando
Algoritmo di discesa del gradiente

$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0$
for $k = 0, 1, 2, \ldots$ **do**
    Se θ_k soddisfa una condizione di stop specifica, allora STOP
    $\theta_{k+1} = \theta_k - \alpha_k \nabla f(\theta_k)$, con $\alpha_k > 0$ step-size
end for

Nel caso di più parametri, come nelle reti, al posto della singola derivata si ha la Jacobiana delle derivate e si aggiorna il vettore dei parametri, ognuno nella direzione che minimizza le derivate parziali rispetto ad esso.

Ad ogni iterazione dell'algoritmo: Viene calcolata la funzione di perdita su tutti i dati disponibili poi vengono calcolate le derivate parziali rispetto ai parametri (Jacobiana), si prende la direzione che le minimizza e si compie uno passo in questa direzione, si ripete finché non scatta un criterio per fermare l'algoritmo, dopo un numero predefinito di iterazioni o quando dopo alcune iterazioni la funzione di perdita non cambia più.
Gli aggiornamenti dei parametri ad ogni passaggio sono proporzionali all'iperparametro α è importante scegliere attentamente in modo che l'algoritmo ci impieghi un tempo adeguato a concludersi e ottenga risultati soddisfacenti. Spesso il lo step-size viene implementato in maniera che diminuisca con il

progredire dell'algoritmo di discesa del gradiente, in modo da aumentarne la velocità all'inizio dove probabilmente si è più lontani dai valori ottimi e aumentarne la precisione in seguito quando si è più vicini ai valori ottimi [16]. Il valore dei parametri iniziali da cui partire per l'algoritmo è scelto in base a quello si conosce del problema per cercare il punto più vicino possibile al ottimo, in caso contrario un punto casuale nel dominio.
All'inizio, prima dell'allenamento del modello, gli output saranno molto lontani da quelli corretti che ci si aspetta. Ma con ogni esempio che gli si pone durante la fase di allenamento il modello corregge leggermente i parametri nella direzione corretta per ottimizzare la precisione dei risultati. Con un numero adeguato di dati per l'allenamento è possibile ottenere un modello funzionante.

Back-propagation L'algoritmo di back-propagation è un algoritmo di ottimizzazione che permette di calcolare le derivate rispetto ai parametri del grafo in modo efficiente, grazie alla chain rule, questo fa si che si possa usare una tecnica di discesa del gradiente per allenare la rete neurale. Essendo la rete una sequenza di funzioni applicate una sull'altra fino ad ottenere la funzione di perdita, si può ripartire da essa e ripercorrere il grafo all'indietro usando la chain rule, trovando la direzione in cui modificare i pesi per minimizzare la funzione di perdita.
É importante inizializzare i parametri della rete con valori generati casualmente e diversi fra loro perché se partissero tutti con lo stesso valore causerebbe ai nodi di comportarsi nello stesso modo e quindi si perderebbe il vantaggio di avere più nodi.

Esempio Back-propagation Ipotizzando un semplice esempio dove gli input sono (a, b, c) si ha un strato nascosto con due nodi (d, e) ed un output Y. Come si vede in figura 14 in ogni nodo tranne per gli input ha un funzione di attivazione assegnata.

- $\nabla_{\mathbf{y}}(L)$ è il derivata di L funzione di perdita che in questo caso è l'errore quadratico medio.
$$g(\mathbf{y}) = \nabla_{\mathbf{y}}(L) = 2(\mathbf{y} - \hat{\mathbf{y}})$$
- $J_{\mathbf{d}}(\mathbf{y})$ indica derivata parziale di y rispetto a d

In questo semplice esempio per calcolare la derivata dei nodi d'intermezzo e e d si calcolano, tramite chain rule, come: $g(\mathbf{d}) = J_{\mathbf{d}}(\mathbf{y})g(\mathbf{y})$ e $g(\mathbf{e}) = J_{\mathbf{e}}(\mathbf{y})g(\mathbf{y})$ e si

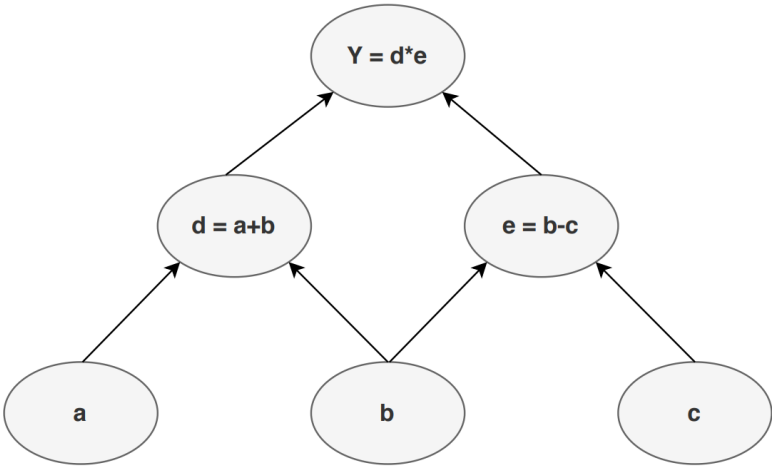


Figura 14: Rete con le varie trasformazioni che avvengono nella propagazione

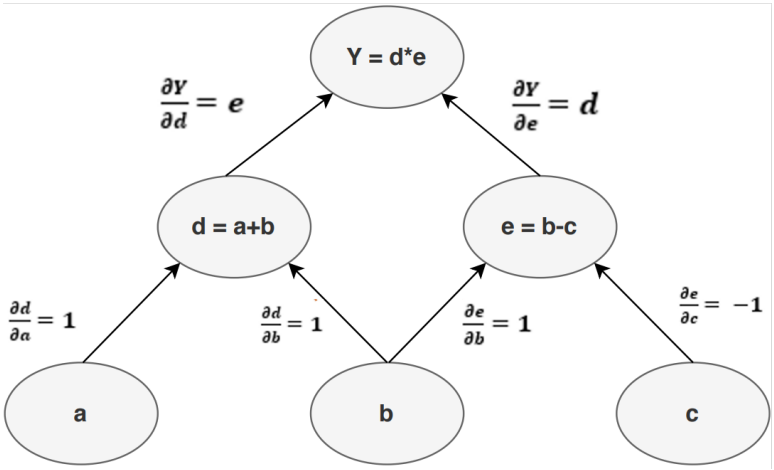


Figura 15: Derivate parziali della rete

arriva man mano così fino agli input: $g(\mathbf{a}) = J_{\mathbf{a}}(\mathbf{d})g(\mathbf{d})$, $g(\mathbf{c}) = J_{\mathbf{c}}(\mathbf{e})g(\mathbf{e})$ Per il valore b si hanno due nodi precedenti (guardando il grafo al contrario) quindi si sommano le due derivate: $g(\mathbf{b}) = J_{\mathbf{b}}(\mathbf{d})g(\mathbf{d}) + J_{\mathbf{b}}(\mathbf{e})g(\mathbf{e})$

$$\frac{\partial Y}{\partial a} = \frac{\partial Y}{\partial d} \times \frac{\partial d}{\partial a} = e \times 1 = e$$
$$\frac{\partial Y}{\partial b} = \frac{\partial Y}{\partial d} \times \frac{\partial d}{\partial b} = e \times 1 = e$$
$$\frac{\partial Y}{\partial c} = \frac{\partial Y}{\partial e} \times \frac{\partial e}{\partial c} = d \times -1 = -d$$

Calcolate le derivate, l'algoritmo sceglie la direzione in cui muovere i pesi per minimizzare la funzione di perdita, e li aggiorna sottraendo il gradiente moltiplicato per lo step-size, così facendo fa un passo nella direzione della discesa del gradiente. Ripetendo il procedimento ad ogni iterazione dell'algoritmo si

ottimizza il modello. Nella versione classica effettua un aggiornamento dei parametri dopo aver eseguito la rete su tutti i dati di allenamento ad ogni iterazione, questo spesso risulta molto lento e conviene aggiornare i parametri ad ogni singolo input o ad un gruppo di input. Procedendo in questo modo, in molti casi in cui il dataset è particolarmente grande, l'algoritmo riesce a convergere più velocemente rispetto alla normale discesa del gradiente. Si tratta di applicare la discesa del gradiente stocastico al caso di back-propagation.

2.2.6 Overfitting e regolarizzazione

Durante la fase di allenamento del modello è possibile controllare che il modello non stia overfittando i dati, per fare questo basta usare l'insieme di convalidazione. Dopo ogni tot iterazioni dell'algoritmo di ottimizzazione si va a valutare l'andamento del modello sia sul dataset di allenamento sia sul insieme di validazione. Studiando l'andamento sui due insieme è facile vedere se si sta overfittando: nel caso in cui il modello funziona meglio sul training significa che si sta adattando troppo ai pattern presenti nell'insieme specifico con cui viene allenato, riuscendo a generalizzare meno. Basterà fermare l'algoritmo quando si nota questo per avere un criterio per fermare l'ottimizzazione che regolarizza il risultato. [17] [**?**]

3 Compressione dati

Per compressione dati si intende l'insieme di tecniche che, attuate a mezzo di opportuni algoritmi, permettono la riduzione della quantità di bit necessari alla rappresentazione in forma digitale di un'informazione. Mentre per decompressione dati si chiama il processo inverso di riportare i dati dalla forma compressa a quella originale.
La compressione oggi viene usata ovunque nell'informatica. Internet è pieno di contenuti in formati compressi che ne permettono la sua fruibilità. Solitamente immagini nel formato JPEG, GIF o PNG, file audio come gli MP3 o in tutti i video che si guarda dove senza compressione non sarebbe possibile vederli in tempo reale anche con connessioni non estremamente preformanti che altrimenti sarebbero necessarie. Comprimere questi file permette una riduzione delle risorse richieste per inviarli tramite la connessione. Tutti i servizi in streaming che vengono usati oggi, come netflix o spotify, su internet non sarebbero possibili.
Dato l'aumento di persone che accedono ed usano internet nel mondo poter trovare nuovi modi, più efficienti, di comprimere i dati risulta molto utile per velocizzare i servizi. Per esempio, per rappresentare un secondo di video senza compressione (formato CCIR 6010) servono più di 20 megabyte, se si considera la lunghezza di un film avremmo video che peserebbero più di 150 gigabyte e non potrebbero essere visionati in streaming senza una compressione adeguata. Queste tecniche permettono anche l'ideazione di nuove possibili tecnologie che sfruttano l'alleggerimento dei dati nei loro processi. Un esempio recente sono le piattaforme di cloud-gaming dove è possibile giocare ad un videogame, in streaming, senza la necessita di avere una console hardware dove far andare i videogiochi stessi; basta una connessione, uno schermo ed un controller.
Questi metodi hanno un grande potenziale sfruttabile anche per la compressioni di dati più specializzati e complessi, come nel settore della fisica e biologia dove spesso si ha a che fare con grandi quantità di dati poco strutturati.

Le due principali tipologie di compressione dati sono:

- compressione dati con perdita d'Informazione in cui si comprimono i dati attraverso un processo con perdita in cui dal dato compresso non è più possibile ottenere la forma originale. Questo metodo sfrutta le ridondanze presenti nei dati;

- compressione dati senza perdita d'informazione in cui si comprimono i dati attraverso un processo senza perdita d'informazione in modo che dal dato compresso si possa ricostruire il dato originale. Questo metodo sfrutta le ridondanze nella codifica dei dati.[3]

Come è possibile intuire le tecniche con perdita d'informazione generalmente portano a livelli di compressione migliori ma non possono essere usate per tutte le tipologia di file.
Per esempio nell'ambito della trasmissione video o audio attraverso un canale di comunicazione come la TV o internet si usano algoritmi di compressione con perdita d'informazione, questo perché dopo la ricostruzione la differenza tra il file originale e quello compresso è lieve e difficile da notare ad occhio nudo, ma porta un grandissimo risparmio in termini di banda da usare per trasmettere il file.
Nel contesto invece di file di testo non va bene avere una perdita d'informazione perché potrebbe cambiare completamente il significato del testo stesso.

3.1 Tecniche di compressione nella codifica sorgente

Quando si parla di tecniche di compressione ci si riferisce principalmente a due tipologie di algoritmi: Un algoritmo che prende in input un file x e genera una sua rappresentazione x_c che occupa meno bits in memoria ed un secondo algoritmo che fa l'opposto, prende in input x_c e lo ricostruisce simile o uguale all'originale.
Ci sono molte tecniche di compressione che si differiscono in base alla tipologia di file che si vogliono comprimere, si cerca in tutti casi di eliminare ridondanze presenti nei dati da comprimere, il primo passo è quello di capire qual'è la fonte delle ridondanze nei dati per poterli eliminare in modo efficiente, per fare questo si sfrutta la modellazione probabilistica.
La teoria che sta dietro alla compressione dati è analizzata dalla teoria delle informazioni. Questi campi sono stati studiati inizialmente da Claude Shannon negli anni '40 e '50 del 900.

3.2 Comunicazione tramite un canale

Ipotizzando di trovarsi nel classico problema di comunicazione della teoria dell'informazione: dove si ha un mandante che vuole trasmettere un messaggio ad un mittente tramite un canale di comunicazione. Il messaggio è il nostro

Figura 16: Comunicazione tramite un canale

insieme di dati che si vogliono comprimere, in modo da risparmiare risorse nella trasmissione delle informazioni.
Il messaggio che deve essere trasmesso può avere varie forme. Immagine, testi, video o altro. Il canale presenta delle proprietà, può essere digitale o analogico e può essere senza rumore o con rumore in base se un messaggio passando in esso presenterà un aumento del rumore o meno. Alcuni esempi di canali di comunicazione sono la rete internet, onde radio o la fibra ottica; anche un sistema di archiviazione normale può essere inteso come canale di comunicazione perché il file può essere scritto e letto come se fosse mandato e ricevuto.
Il messaggio dopo aver attraversato il canale deve essere ricostruito per poter essere infine letto dal ricevitore. La ricostruzione dipende dal tipo di file e anch'essa può essere con o senza perdita d'informazione. In caso di una comunicazione continua anche la ricostruzione deve essere fatta in un flusso continuo, per esempio come quando si guardano servizi di streaming dove il video viene ricostruito e in tempo reale dopo essere stato mandato tramite internet.
Per permettere la trasmissibilità del messaggio serve aggiungere ulteriori due passaggi (figura 16): la fase di codifica e decodifica del messaggio. Il messaggio viene prima codificato, poi trasmesso tramite il canale di comunicazione per poi essere decodificato e ricevuto dal destinatario. Queste due fasi hanno lo scopo sia di comprimere e decomprimere il messaggio, ma anche di rendere il messaggio effettivamente trasmissibile sullo specifico canale. Questi due passaggi possono essere ulteriormente divisi in due sottopassaggi, uguali in entrambe le fasi: il primo sottopassaggio è la codifica sorgente dove vengono rimosse le ridondanze presenti nei dati ed avviene la compressione. Si specifica che le ridondanze che vengono rimosse sono proprie del messaggio. In seguito avviene la codifica di canale dove invece vengono aggiunte ridondanze, diverse dalle precedenti, queste si basano sulla tipologia di canale in cui il messaggio deve essere trasmesso e servono per rendere possibile il trasferimento.
Nella fase di decodificazione i passaggi sono gli stessi della codificazione. La codifica di canale serve a correggere eventuali errori dovuti al rumore introdotto dal canale stesso e nella fase di decodifica di sorgente viene fatto il processo inverso della codifica sorgente quindi decomprime il dato e lo rende leggibile.
Nella fase di codifica il file potrebbe diventare più piccolo come più grande

dipende da caso a caso, più piccolo quando il file presenta abbastanza ridondanze, se si hanno invece dati con molto rumore potrebbe diventare più pesanti da inviare rispetto all'originale.
Non sempre avvengono entrambi questi passaggi di codifica in tutte e due le fasi, dipende da come il canale di comunicazione è stato progettato.
Ipotizzando anche che il canale non aggiunga rumore ai nostri dati (noise-free), questo può essere assunto a livello teorico grazie alle due fasi di codifica di canale prima e dopo il canale che grazie alla ridondanze che aggiungono e tolgono nei dati per renderlo trasmissibile hanno proprio lo scopo di rendere la comunicazione libera dal rumore, almeno dal punto di vista teorico. Per il metodo di compressione oggetto della tesi quello d'interesse è la fase di codifica e decodifica di sorgente e si parla di un metodo senza perdita d'informazioni.
Per spiegare come avviene la compressione delle informazioni prima è necessario parlare del concetto di codifica in ambito informatico. Un codice, nella teoria delle comunicazioni e delle informazioni, consiste in un insieme di regole che trasformano delle informazioni dalla sua forma originale (immagine, testo, altro) ad una forma diversa che nel caso della compressione risulterà essere più compatta rispetto a quella originale.

Esempio: linguaggio naturale Ipotizzando quindi di voler inviare un messaggio scritto e volerlo comprimere senza nessuna perdita d'informazioni. In un messaggio scritto si ha un insieme di caratteri possibili che dipendono in parte dalla lingua, messi in sequenza formano i messaggi possibili. Nella codifica di essi si andrà a mappare ogni carattere ad un codice (di solito un numero o una sequenza di numeri) con lo scopo di facilitare la memorizzazione del testo in un sistema informatico.

3.3 Symbol code

Nella teoria dell'informazione i singoli caratteri che compongono il messaggio vengono chiamati simboli e i codici per mappare i simboli nella loro versione codificata vengono chiamati symbol Code. Si può assumere come simbolo il singolo carattere, tuttavia in alcuni casi è meglio considerare le parole come simboli; questo dipende da problema in problema, qui si considera come simbolo il carattere.
Il messaggio da inviare $\underline{x}$ è una sequenza di simboli x_i provenienti da una distribuzione discreta e finita X; lo spazio campionario della distribuzione di

simboli è chiamato alfabeto sorgente $\mathcal{X}$.

$$\underline{x} = x_1, x_2, .., x_k = (x_i)^k$$

con

$$x_1, x_2, x_3, \ldots, x_k \overset{\text{i.i.d.}}{\sim} X$$

con $X \sim Multinomiale(x, \pi)$ dove π è il vettore delle probabilità π_i che si osserva il carattere x_i indipendenti e identicamente distribuiti.
Ogni simbolo appare con una probabilità $P(X = x) = \rho(x)$. Queste probabilità possono essere stimate tramite la massima verosimiglianza che corrisponde alle frequenze relative dei caratteri nei messaggi del campione.
Dal messaggio originale $\underline{x}$ si vuole passare ad una sua rappresentazione compressa senza perdita d'informazioni $C^*(\underline{x})$ per fare ciò è necessario trovare un modo per mappare ogni elemento dell'alfabeto sorgente $\mathcal{X}$ in una forma codificata in modo da occupare meno spazio. Il code-book C è il dizionario che mappa ogni carattere dall'alfabeto sorgente $\mathcal{X}$ nella sua versione codificata $C(x)$ chiamata code-word per il simbolo x dell'alfabeto sorgente $\mathcal{X}$.

$$\mathrm{C} : \mathcal{X} \to C^*(x)$$

Si ha un secondo alfabeto di riferimento per i simboli codificati, questo alfabeto viene chiamato 'alfabeto codice', ed è l'insieme di elementi che formano tutti i simboli codificati. La sua cardinalità è definita come B
Nel caso in cui mappano i simboli in un codice binario si ha $B = 2$ e in generale si ha un codice B-nario C
Il messaggio codificato secondo il code-book scelto viene definito concatenando i vari simboli del messaggio codificati:

$$C^*(\underline{x}) = C\left(x_1\right) \| C\left(x_2\right) \| \ldots \| C\left(x_k\right)$$

Se si vuole evitare la perdita d'informazione è necessario che la funzione C che mappa i simboli sia univoca. Un codice C definisce un simbolo decodificabile in modo univoco se

$$\forall \underline{x}, \underline{x}' \in X^* \text{ with } \underline{x} \neq \underline{x}$$
$$C^*(\underline{x}) \neq C^*\left(\underline{x}'\right)$$

Due simboli differenti nell'alfabeto sorgente vengano codificate sempre in due simboli diversi con il code book. Se non fosse così sarebbe un problema nella

character	encoding	bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001
あ	UTF-8	11100011 10000001 10000010
あ	UTF-16	00110000 01000010
あ	UTF-32	00000000 00000000 00110000 01000010

Figura 17: Codifica unicode

compressione dati senza perdita d'Informazione perché rende impossibile la decodifica esatta alla parola originaria e si perderebbero informazioni.
Un esempio di codifica che rappresenta lo standard per quanto riguarda testi scritti in ambito informatico sono i sistemi Unicode, per cui ad ogni carattere possibile dell'alfabeto sorgente viene mappato ad un numero univoco sempre della stessa lunghezza (codice di lunghezza non variabile).
In questo caso si ha:

- **Alfabeto sorgente**: insieme di caratteri usati per la scrittura dei messaggi da inviare $\mathcal{X} := \{A, B, \ldots, Z, 0, 1, \ldots, 9\}$
- **code-book**: il dizionario di riferimento per mappare un carattere alla sua code-word $C = 00000011$
- **Alfabeto codice**: ogni carattere viene mappato ad una cifra formata da un codice binario, quindi l'alfabeto è formato da solo 0 e 1 $C^*(x) = \{0, 1\}$

Un altro esempio famoso di codifica di un messaggio scritto è l'alfabeto Morse. Questo è basato sulla trascrizione dei 36 simboli alfanumerici tramite combinazioni di punti e linee, insieme al telegrafo fu' fondamentale per il progresso delle comunicazioni a distanza perché rese possibile la trasmissione di un messaggio scritto tramite segnali elettrici (canale di comunicazione).
In questo caso si ha:

- **Alfabeto sorgente**: insieme di caratteri usati per la scrittura dei messaggi da inviare $\mathcal{X} := \{A, B, \ldots, Z, 0, 1, \ldots, 9\}$
- **code-book**: il dizionario di riferimento per mappare un carattere alla sua code-word

Figura 18: Dizionario per il codice Morse

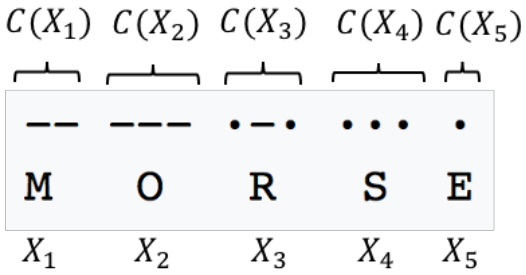


Figura 19: Parola Morse codificata

- **Alfabeto codice**: ogni carattere viene mappato ad una cifra formata da un codice binario quindi l'alfabeto è formato dai simboli tratto e punto. $C^*(x)$

Al contrario della codifica Unicode qui il codice è di lunghezza variabile e sfruttando le ridondanze presenti nel testo si può comprimere il messaggio: assegnando codeword corte a simboli che si ripetono più volte nel testo e le codeword lunghe ai simboli più rari. In questo caso Morse assegnò sulla base della frequenza dell'uso dei caratteri alfanumerici nella lingua inglese. Alle lettere più comuni (E,T) sono state assegnate le codeword più brevi (.,-) e così via dalle più frequenti a quelle meno, si assegnano codeword più brevi fino a quelle più lunghe. Il codice Morse ha un problema, i simboli, mappati in sequenza, essendo trasmessi uno di seguito all'altro e di lunghezza variabile $C^*(\underline{x}) = C(x_1) \| C(x_2) \| \ldots \| C(x_k)$ non permettono di capire quando inizia una code-word e quando finisce, rispetto alle altre. Questo perché alcune code-word sono il prefisso di altre e rende il codice non univocamente codificabile. Univocamente decodificabile significa quindi che ogni code-word è identificabile quando si trova all'interno di una sequenza di code-word. Per risolvere tale problema Morse aveva pensato agli "spazi": brevi pause tra una parola e l'altra, questa soluzione però aveva il difetto di rallentare, anche di molto, la velocità di trasmissione dei messaggi. Per risolvere questo problema si usano i codici

prefisso, in cui nessuna code-word è prefissa di un'altra. Questa proprietà, se rispettata, implica che i messaggi verranno codificati e decodificati in modo univoco. Se si ha un codice unicamente codificabile, però, non è detto che sia un codice prefisso.
Tutte le tecniche di compressione, come quella di Morse, si basano sul trovare un ottimo modello per la sorgente dei dati, più si riesce a modellare in modo preciso la fonte dei dati sorgente più si può ottenere una compressione effettivamente migliore, perché è possibile modellare la sorgente delle ridondanze nei messaggi ed eliminarle per diminuire il peso dei messaggi.

3.4 Misura di performance

Si può valutare un algoritmo di compressione guardando vari aspetti, si può misurarne la efficienza nel diminuire lo spazio occupato dal file, la velocità di compressione, o nel caso di compressione con perdita, quanto bene viene ricostruito il file alla sua forma originale.
La lunghezza del messaggio x codificato è definita come $\ell(x) = |C(x)|$
Si può calcolare la lunghezza media attesa per un simbolo x dato un code book (expected codeword length) C, corrispondente a

$$L = E[\ell(x)] = \sum_{x \in x} p(x)\ell(x)$$

Permette di avere una misura di quanta compressione in media si è raggiunto con l'uso del codice scelto per mappare i simboli originali. Cercare di minimizzarla nella scelta del code-book permette di ottenere una ottima compressione.
Per valutare la capacità di ridurre la dimensione un'altra misura logica e dalla facile interpretazione è il rapporto di compressione tra il file prima e dopo l'avvenuta compressione. Più il valore sarà piccolo più la compressione è maggiore.
Un esempio:

x	$p(x)$	$C(x)$	$\ell(x)$
a	1/2	0	1
b	1/4	10	2
c	1/8	110	3
d	1/8	111	3

$$
\begin{aligned}
L &= 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + 3 \cdot \frac{1}{8} \\
&= \frac{1}{2} + \frac{1}{2} + \frac{3}{8} + \frac{3}{8} \\
&= 1.75
\end{aligned}
$$

x	$p(x)$	$C(x)$	$\ell(x)$
a	$1/2$	101	3
b	$1/4$	00	2
c	$1/8$	0001	4
d	$1/8$	1	1

$$
\begin{aligned}
L &= 3 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 4 \cdot \frac{1}{8} + 1 \cdot \frac{1}{8} \\
&= 1.5 + \frac{1}{2} + \frac{1}{2} + \frac{1}{8} \\
&= 2.625
\end{aligned}
$$

Un fondamentale risultato della teoria dell'informazione è la disuguaglianza di Kraft MCmillan. Per ogni codice unicamente identificabile B-nario si ha:

$$\sum_{i=1}^{A} B^{-l_i} \leq 1$$

con B cardinalità dell'insieme alfabeto codice
da cui deriva l'assioma: Se un qualsiasi $\ell : X \to \{0, 1, 2, \ldots\}$ soddisfa la disguaglianza (1) allora esiste una codice prefisso B-nario $\mathbf{C}$ con

$$|C(x)| = \ell(x) \quad \forall x \in \mathcal{X}$$

Quindi guardando la cardinalità dell'alfabeto codice e le lunghezze dei vari simboli codificati è possibile capire se si può trovare un codice prefisso univocamente decodificabile. Per ottenere una buona compressione si deve cercare di minimizzare la media di bit per carattere L.

3.5 Informazione ed entropia

É utile comprendere il ruolo dell'informazione nella compressione dei dati. Il primo a darne una definizione matematica fu Claude Shannon che si ispirò molto dal telegrafo di Morse. Nel suo articolo del 1948 *The mathematical Theory of Communication* [18]. Shannon constatò una relazione inversa tra

l'informazione e la probabilità del messaggio e formulò una misurazione per l'informazione contenuta in un dato chiamata Entropia.

$$H(X) = \sum_{i=1}^{m} p_i \log_B \frac{1}{p_i}$$

L'entropia è in relazione con la probabilità che accada un evento in modo opposto al logaritmico: si vede infatti, se la probabilità che l'evento accada è pari ad 1 allora di conseguenza l'entropia è zero, aumenta con il diminuire della probabilità dell'evento. Ci da una misura dell'incertezza ed della informazione contenuta. Prendendo l'esempio del lancio di una moneta, dove si ha $p(Testa) = \pi$ e $p(Croce) = (1 - \pi)$

$$H = -((1 - \pi)\log(1 - \pi) + \pi \log \pi)$$

Immaginando che la moneta sia truccata e la probabilità che esca testa dopo un tiro di essa è $p(Testa) = 0.9$. Riuscire a prevedere cosa uscirà al prossimo lancio della moneta è molto semplice data l'elevata probabilità di testa. Più è prevedibile il prossimo lancio meno informazione che un campione ci può dare e minore entropia. L'entropia è la misura di questa incertezza e quindi è una misura della informazioni che viene convogliata dal messaggio. Più è prevedibile il messaggio che la sorgente produrrà, minore è l'incertezza, minore l'entropia e di conseguenza minore l'informazione. La distribuzione Gaussiana ha la massima entropia.

Consegue dal teorema source coding shannon [18] che il limite inferiore di questo problema di minimizzazione è uguale all'entropia della distribuzione che genera i simboli $H(p)$.

Theorem 1 *Dal teorema (Shannon's Source Coding Thoerem):*
Data una variabile casuale categoriale X campionata dall'alfabeto sorgente $\mathcal{X}$ con un alfabeto codice $\mathcal{A}$, allora per ogni univocamente codificabile $C : \mathcal{X} \to \mathcal{A}^*$, vale che $E[|C(X)|] \geq H(X)$

Il risultato può riscritto con la formulazione vista

$$H_B(p) \leq L$$

L'entropia ci da un limite inferiore a quanto si può comprimere un dato senza perdita d'informazione infatti e essa una misura stessa dell'informazione presente nel dato, almeno rispettivamente alla distribuzione sorgente che ha generato il dato. Più è "raro" il messaggio più contiene informazioni. Riprendendo l'esempio precedente

x	(fx)	$l(x)$	$p(x)$
a	0	1	1/2
b	10	2	1/4
c	110	3	1/8
d	111	3	1/8

$L = 1.75$

$$
\begin{aligned}
H(p) &= \sum p(x) \log_2 \frac{1}{p(x)} \\
&= \frac{1}{2} \log 2 + \frac{1}{4} \log 4 + 2\frac{1}{8} \log 8 \\
&= \frac{1}{2} + \frac{1}{2} + \frac{6}{8} = 1.75
\end{aligned}
$$

In questo caso molto semplice si vede come il codice usato è già ottimo e ha raggiunto il limite inferiore pari all'entropia della variabile generatrice dei caratteri. [**?**] Noi useremo come funziona di perdita per il modello della sorgente la cross-entropia:

$$H(\mathbf{p}, \mathbf{q}) = -\sum_{i=1}^{n} p_i \log q_i$$

misura il numero medio di bit necessari per identificare un simbolo estratto dal messaggio nel caso sia utilizzato uno schema ottimizzato per una distribuzione di probabilità q piuttosto che per la distribuzione vera p

3.6 Codifica di Huffman

Un metodo diffuso di codifica senza perdita d'informazione che si userà nel modello implementato è la codifica di Huffman, ideata nel 1952 da David A. Huffman. Studente di Claude Shannon, Huffman riusci a trovare un algoritmo in grado di produrre un ottimo symbol code dalla distribuzione di probabilità dei simboli sorgente; ottimo perché in caso si conosce la vera distribuzione dei simboli sorgenti x allora porta sempre ad una compressione parti all'entropia (entropy coding). Questo risultato è molto utile perché sposta il problema della

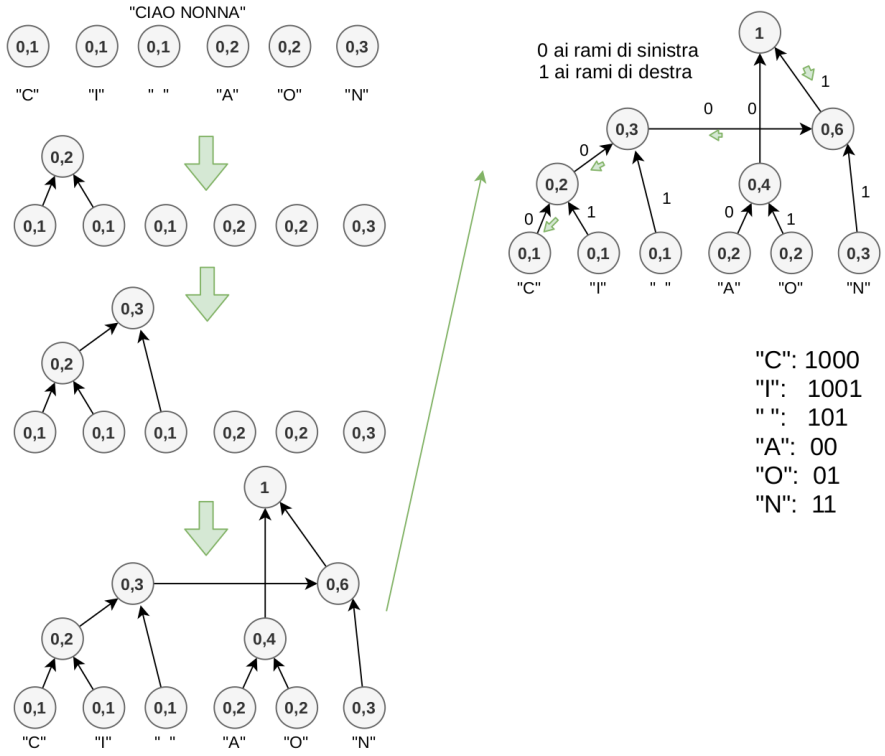


Figura 20: Esempio codifica Huffman

compressione dal trovare il codice ottimo per codificare i simboli ad un problema di modellazione della sorgente, se la si modello bene allora applicando la codifica Huffman si otterrà una buona compressione. Permette di tenere separate la fase di allenamento dal modello dalla fase effettiva di compressione dei dati. La codifica di Huffman viene ancora utilizzata nella compressione di testi, immagini, video e altri come JPEG, MPEG-2, MPEG-4, and H.263, etc.

Esempio codifica Huffman Si illustra brevemente il funzionamento della codifica di Huffman (figura 20): L'algoritmo sfrutta un albero binario per trovare il miglior codice ad ogni simbolo.
Si vuole comprimere il messaggio in un codice bi-nario. Si tratta di costruire un albero partendo dalle foglie fino ad arrivare alla radice. Le foglie iniziali rappresentano la distribuzione dei simboli con la loro probabilità, posizionate in ordine crescente o decrescente di probabilità. Si trovano le due probabilità più basse e le si somma per formare un nuovo nodo, si continuano a ripetere questo passaggio finché arrivati all'ultimo nodo si è completato l'albero, arrivando alla radice. Trovato l'albero si assegna ad ogni ramo il valore 0 se è un ramo di

sinistra ed 1 se è di destra; infine basterà seguire ogni percorso che porta dalla radice ai vari simboli e mettendo in sequenza i valori dei rami si trova il codice corrispondente al simbolo. [19]

3.7 Modellazione della sorgente generatrice dei caratteri

Usando la codifica di Huffman e grazie ai risultati teorici visti è possibile concentrarsi unicamente sulla modellazione della sorgente dei messaggi, per ottenere una buon metodo di compressione. Fino ad ora si è ipotizzato che il fenomeno generatore dei messaggi generasse i simboli in modo indipendente e con la stessa distribuzione di probabilità per ogni simbolo. L'assunzione d'indipendenza fra i simboli risulta non realistica nel caso di linguaggi naturali, nei testi scritti si ha quasi sempre una correlazione fra i simboli nella loro posizione all'interno del messaggio.

Modellare le correlazioni tra i simboli Si ha che ogni simbolo del nostro messaggio è dipendente in un certo modo dai simboli precedenti:

$$\underline{x} = (x_1, x_2, x_3, \cdots, x_k)$$
$$P(\underline{x}) = P(x_1)\, P(x_2, x_1)\, P(x_3 \mid x_1, x_2) \cdots P(x_k \mid x_1, x_2..x_{k-1})$$

Ci sono varie strategie possibili per modellare questa tipologia di relazione, in generale si assume che non tutte le correlazioni fra le variabili siano necessarie per descrivere le informazioni del messaggio.

Modello autoregressivo Nell'esempio del messaggio è che i caratteri in esso siano più correlate con i caratteri vicini rispetto a quelli più distanti. Se si pensa ad un testo è naturale ipotizzarlo.
Un esempio di modello è quello autoregressivo dove l'osservazione al tempo t funzione lineare di valori precedenti della stessa variabile più un errore.

$$Yt = \beta 0 + \beta 1 y(t-1) + \varepsilon t$$

Con i linguaggi naturali però questi modelli possono risultare limitanti dal punto di vista della flessibilità sul considerare dipendenze non lineari, per questo si introduce il modello usato in seguito nel metodo di compressione, una rete neurale ricorrente (Recurrent Neural Network, RNN).

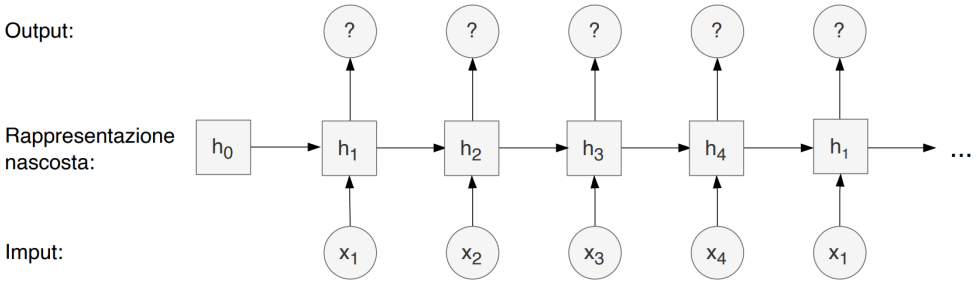


Figura 21: Struttura rete neurale ricorrente semplice

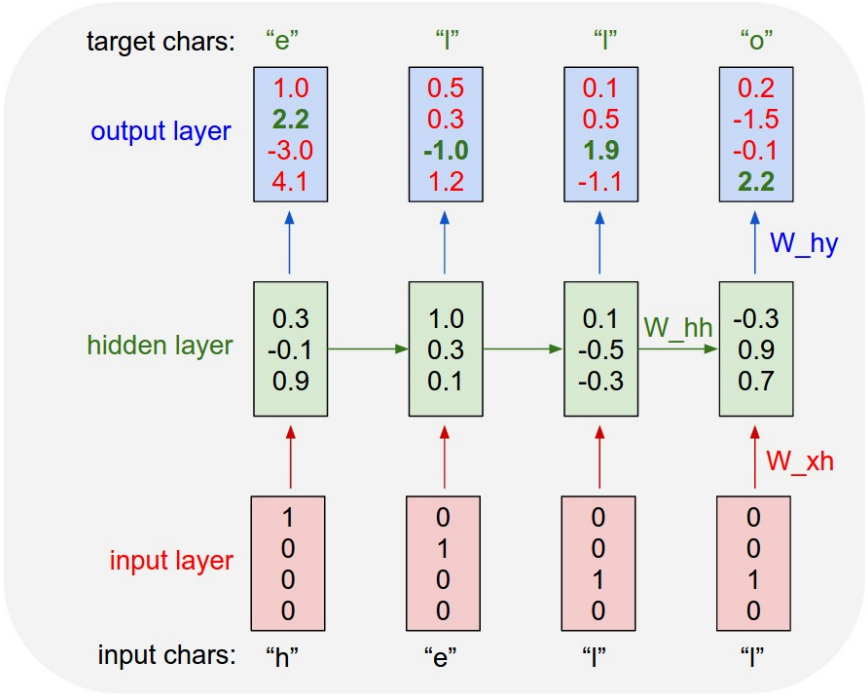


Figura 22: Rete neurale ricorrente semplice nella modellazione di caratteri

3.8 Modello rete neurale ricorrente sui caratteri di un testo

Sfruttare una rete neurale ricorrente (Recurrent Neural Network, RNN) permette di avere un modello flessibile per dati temporali. Le reti neurali artificiali classiche elaborano in modo separato i dati che ricevono e non tengono conto degli input precedenti come si vorrebbe nel caso di dati sequenziali, mentre la forma tipica di una RNN è come in in figura 21.
Al posto di una singola rete si ha una sequenza di reti, collegate fra loro dai nodi di elaborazioni (nodi nascosti), è una struttura molto flessibile che può avere forme diverse in cui variano le caratteristiche delle singole reti, il numero di input e output per ogni rete e i collegamenti fra i nodi.
Quello che si cerca di ottenere è un modello che riesca a prevedere un carattere

sulla base di quelli precedenti. Per fare ciò ogni carattere viene codificato in un vettore dove l'indice che corrisponde al carattere letto è uguale ad 1 e tutti gli altri sono 0, in modo da indicizzare il carattere letto. Questo vettore viene dato in input alla rete che ne genera la previsione per il carattere successivo, nella generazione dell'output, nel il calcolo dei nodi nascosti a tempo t, l'input x_i viene moltiplicato con il nodo precedente $t-1$ ed è in questo collegamento che viene modellata la struttura temporale fra i caratteri.

$$\mathbf{h}_t = f\left(\mathbf{h}_{t-1}, \mathbf{x}_t\right)$$

Poi viene eseguita la trasformazione che genera l'output nella forma di un vettore che rappresenta il vettore delle probabilità per il prossimo carattere. Spesso viene utilizzata la funzione softmax per restringere gli intervalli dei valori in uscita nel dominio $R \subseteq (0, 1)$ (funzione esponenziale normalizzata).

$$\mathbf{o}_t = f\left(\mathbf{h}_{t-1}, \mathbf{x}_t\right)$$

La previsione del prossimo carattere viene confrontata con il vero carattere successivo in modo da poter calcolare la funzione di perdita che ci permette sia di valutare l'andamento sia di ottimizzare i parametri del modello. Si tratta di una tecnica di apprendimento non supervisionata, dato che si usano i caratteri stessi del testo per calcolare la funzione di perdita.
L'algoritmo di ottimizzazione dei parametri si chiama back-propagation through time che funziona in modo molto simile alla back-propagation normale. Per calcolare le derivate parziali necessarie srotola la struttura ricorsiva all'indietro e ripercorre le derivate fino ad ogni peso per trovare la direzione ottima.
I parametri vengono aggiornati dopo aver calcolato la funzione di perdita su un insieme di porzione di testo. La dimensione della porzione dipende da quanto indietro si vogliono considerare le correlazioni ed il numero di porzioni da come si è implementata la discesa del gradiente, possono esserci più nodi nascosti, funzioni di attivazione diverse e altre modifiche.
Il modello può generare testo con una semplice modifica, si da in input un singolo carattere e dal secondo usa l'output generato a tempo y_{t-1} come input x_t, come si vede in figura 24.

Problema del gradiente che esplode/sparisce La struttura ricorsiva delle RNN porta ad due problemi molto comuni: l'esplosione o la scomparsa del

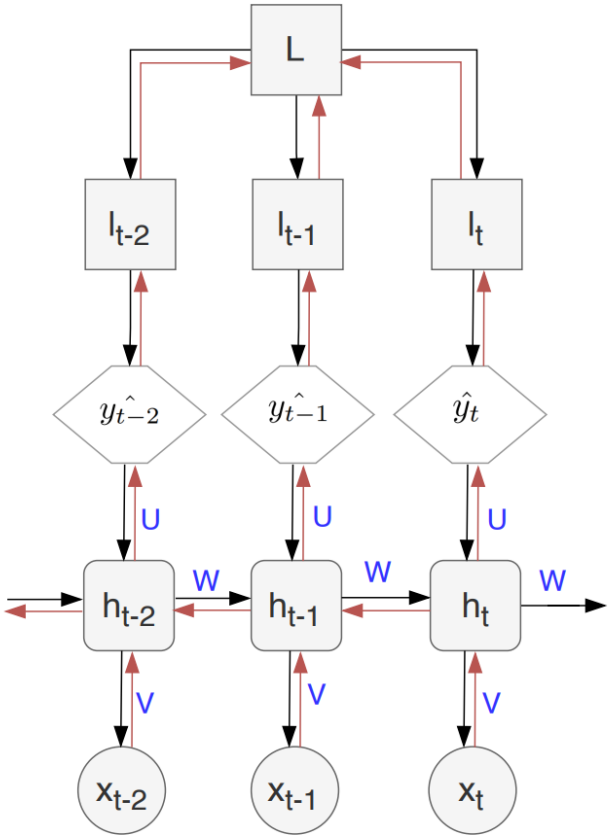


Figura 23: RNN con calcolo della funzione di perdita su 3 simboli

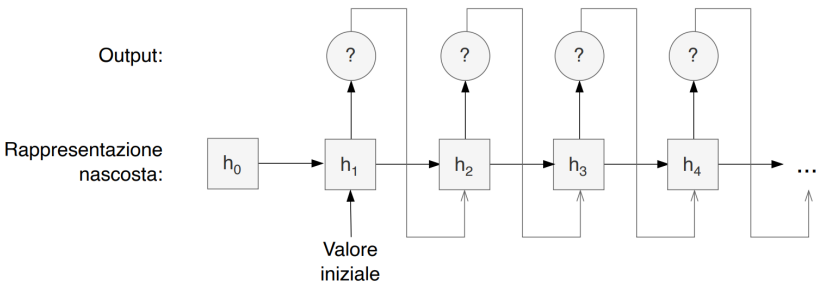


Figura 24: RNN per generazione di caratteri

gradiente: Ad ogni passaggio dell'algoritmo di ottimizzazione si calcolano le derivate parziali come

$$\frac{\partial \mathbf{L}}{\partial W} = \sum_{i=0}^{T} \frac{\partial \mathcal{L}_i}{\partial W} \propto \sum_{i=0}^{T} \left(\prod_{i=k+1}^{y} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

[17] la produttoria dei rapporti è quella in cui ricadono tutte le derivate ai tempi precedenti, si hanno due casi problematici in base a come è il rapporto:

1. Vanishing gradient $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$ il gradiente converge esponenzialmente a zero
2. Exploding gradient $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$ il gradiente converge verso l'infinito esponenzialmente

Come si può immaginare questo comporta dei problemi se si vuole applicare il modello su una sequenza di dati abbastanza lunga. Un primo modo per evitare questo è allenare il modello per sequenze più corte di testo alla volta, come avviene nella discesa del gradiente a batch, però spesso anche le porzioni di testo singole risultano troppo lunghe. Un altro modo è quello di scegliere adeguatamente le funzioni di attivazione e i pesi iniziali nella rete. Infine anche modificare la struttura della rete stessa può portare a una migliore gestione del calcolo delle derivate, è il caso dei blocchi strutturali chiamati gated cell.

Gated Cell: LSTM e GRU L'idea delle gated cell è quella di sostituire i semplici collegamenti fra i nodi di diverso tempo con delle strutture più complesse che hanno lo scopo di aggiungere o rimuovere informazione nel passaggio da un nodo al successivo, regolandolo. Le due gated cell tipicamente usati sono: Long short term memory (LSTM) e Gated recurrent unit (GRU), dalla figura 25 si può facilmente vedere come sono strutturate.

Pro e contro Questa tipologia di modelli permette di usare poca memoria e riesce a modellare bene correlazioni fino ad una discreta distanza, riscontra difficoltà per correlazioni particolarmente lontane (non così rare nel caso di testo scritto). Un' altro svantaggio è che non risulta parallelizzabile data la sua struttura di RNN. Essendo che gli output generati servono per la generazione degli output in seguito, non è possibile generare più di un carattere alla volta.

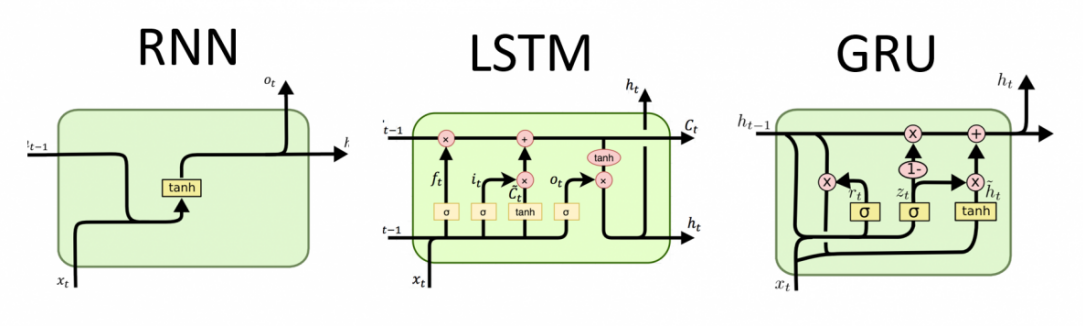


Figura 25: struttura delle celle LSTM e GRU [20]

4 Metodologia

L'implementazione del metodo di compressione è divisibile in due fasi principali, nella prima il modello RNN viene allenato su di un corpus di testo, possibilmente il più simile possibile alla tipologia di testo che si vuole comprimere; nella seconda fase si usa il modello pre-allenato insieme alla tecnica di codifica di Huffman per effettuare la compressione e la decompressione del testo.
Il modello RNN è stato allenato su due diversi dataset:

- il primo su l'intera bibliografia di Shakespeare [21]
- il secondo sulla trascrizione di tutti dialoghi di The Office, una serie TV americana comica. [22]

I due insiemi sono stati suddivisi nei tipici sottoinsieme: allenamento, validazione e test con le seguenti proporzioni: 70%, 20%, 10% e sono stati preprocessati per evitare che contenessero caratteri non alfanumerici. La rete implementata non differisce troppo da quella in figura 22, si trasformano i caratteri con una tecnica di embedding per avere un vettore che ci indica il carattere in input, il collegamento fra l'input, i nodi nascosti e i nodi nascosti dei tempi precedenti è tramite due strutture GRU ognuna formata da 100 nodi. Infine il risultato dei vari nodi interni viene trasformato linearmente per ottenere l'output.
La funzione di perdita valutata è la cross-entropia ottimizzata grazie all'algoritmo Adam, una implementazione della discesa del gradiente stocastica, dove ad ogni iterazione gli è stato dato un batch di testo casuale preso dal dataset di allenamento. L'algoritmo si ferma dopo 1000 iterazioni. Ogni 10 iterazioni viene valutata la cross-entropia sull'insieme di validazione ed è stata

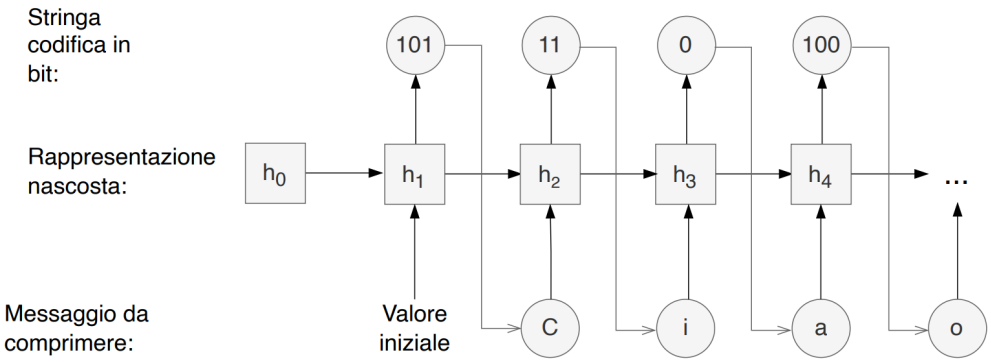


Figura 26: Codificatore

generata una stringa di caratteri dal modello utile per notare che con l'avanzare dell'allenamento la cross-entropia scenderà sempre di più e i testi generati di conseguenza saranno più simili a quelli presenti nell'insieme di allenamento. Inoltre sono stati fatti esperimenti su diversi iperparametri del modello tra i quali valutare porzioni di testo di diversa lunghezza, numero di nodi nascosti ad ogni strato, diverse misure di learning rate e diverse dimensioni del campione per l'ottimizzazione.

Il modello allenato ora viene usato per l'effettiva compressione, per fare ciò si usa la codifica di Huffman (sezione 3.5). Si hanno due strutture diverse, uno per la codifica del testo ed uno per la decodifica.

Codifica Nella fase di codifica si ha il messaggio da comprimere, si usa su di esso il modello RNN allenato precedentemente sulla sorgente dei caratteri per produrre una sequenza di distribuzioni di probabilità sull'alfabeto dei caratteri. Queste distribuzioni trovate carattere per carattere prima vengono normalizzate con la funzione logit poi usando la codifica di Huffman si ottiene il codice compresso. Si ha quindi una sequenza di codici Huffman, uno per ogni carattere. Più si riesce a prevedere meglio il carattere successivo meno lungo sarà il codice compresso risultante.

Decodifica Al momento della decodifica, non si conosce il vero messaggio che si vuole ricostruire, ma si utilizza il modello RNN allenato insieme alla sequenza di stringhe di codice compresso corrispondenti ai caratteri per decodificare. Similmente alla generazione di caratteri da parte del modello perché non si ha il vero messaggio da ricostruire, ora però si vuole generare in modo deterministico i caratteri usando il codice compresso. Grazie al modello RNN si può usare la

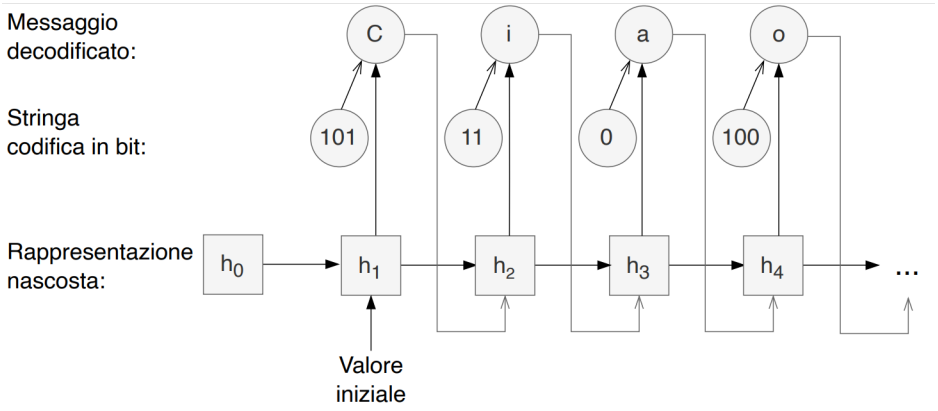


Figura 27: Decodificatore

sua capacità di prevedere il carattere per poter usare meno codice compresso possibile. Viene iterato il codice compresso ed per ogni elemento, grazie anche alle distribuzione generate dal modello è possibile ripercorrere all'indietro gli alberi di Huffman e decomprimere il carattere. Più il modello RNN si prevede meglio la sorgente dei dati, più si può risparmiare spazio nella versione compressa del messaggio perché quando si decodifica il messaggio servono meno informazioni se una parte può essere predetta dal modello probabilistico. Si ripete il processo, consumando una piccola parte della stringa di bit compressa per ogni carattere da decodificare fino a finire tutte le stringhe compresse.

Test Per ogni modello è stato valutato il metodo di compressione sia su del testo nell'insieme test corrispondente sia su testi diversi da quello in cui il modello è stato allenato per vederne meglio le capacità

- due articoli di wikipedia una in inglese ed una in tedesco [23] [24]
- una serie di trascrizioni da un tribunale canadese [25]

Come misura per valutare la compressione è stato calcolato il rapporto tra il file compresso rispetto a quello originale. Più il valore è piccolo più è compresso. I caratteri nella forma non compressa in con una stringa 8-bit quindi il rapporto può essere scritto come

$$\text{Rapporto di compressione} := \frac{\text{Bit per rappresentare la forma compressa}}{\text{Bit per rappresentare la forma originale (8 bit)}}$$

I dati su cui sono stati valutati i due modelli:

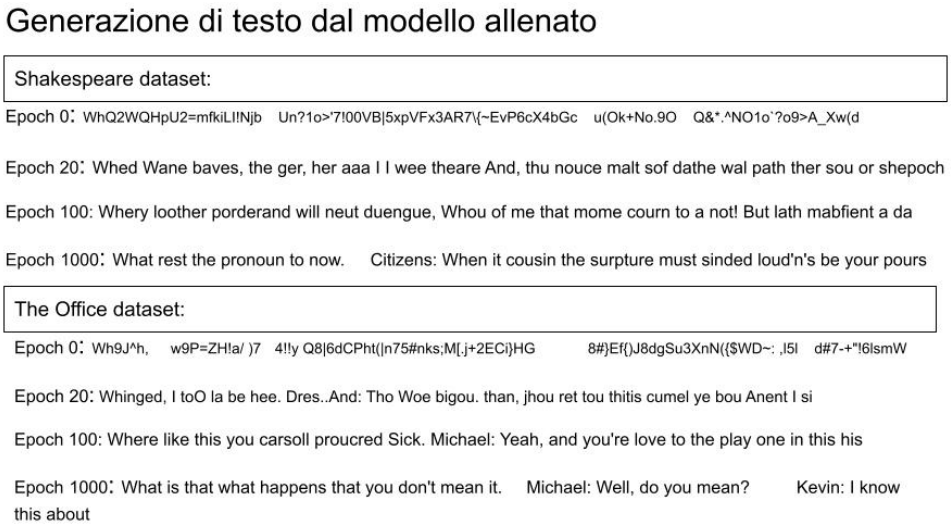


Figura 28: RNN durante la fase di allenamento

Implementazione del metodo di compressione. Il modello è stato testato con una implementazione in Python usando, come framework per la rete, Pytorch. L'implementazione è stata possibile grazie alla libreria Constriction [26] ed alla repositery [27].

5 Risultati

Come si vede dalla figura 29, usare porzioni di testo più ampie di 100 caratteri non porta significativi miglioramenti nel modello, grazie ad altri esperimenti si è scelto di usare due strati GRU, ognuno formato da 100 nodi, lo step-size migliore trovato è i di 0.01 e si aggiornano i parametri ogni 100 porzioni di testo lette. È stato valutato il rapporto di compressione paragonandolo a due metodi molto diffusi di compressione senza perdita d'informazione: gzip [28] e bzip2 [29]. Come si vede guardando l'insieme di test corrispettivo dei due modelli il metodo implementato è migliore anche dei due blasonati metodi di compressione, però quando si prova a comprimere file diversi incomincia a a dare problemi nel funzionamento e non riesce a competere con gzip e bzip2. Più i file di testo sono diversi dal testo in cui il modello è stato pre-allenato più le performance si degradano, questo significa che se si allena il modello su dei testi molto simili a quello che si vuole comprimere allora il metodo funziona bene. I risultati peggiori si vedono con la pagina di Wikipedia in lingua diversa, come ci si poteva aspettare. Essendo stato implementato in Python, la velocità non è

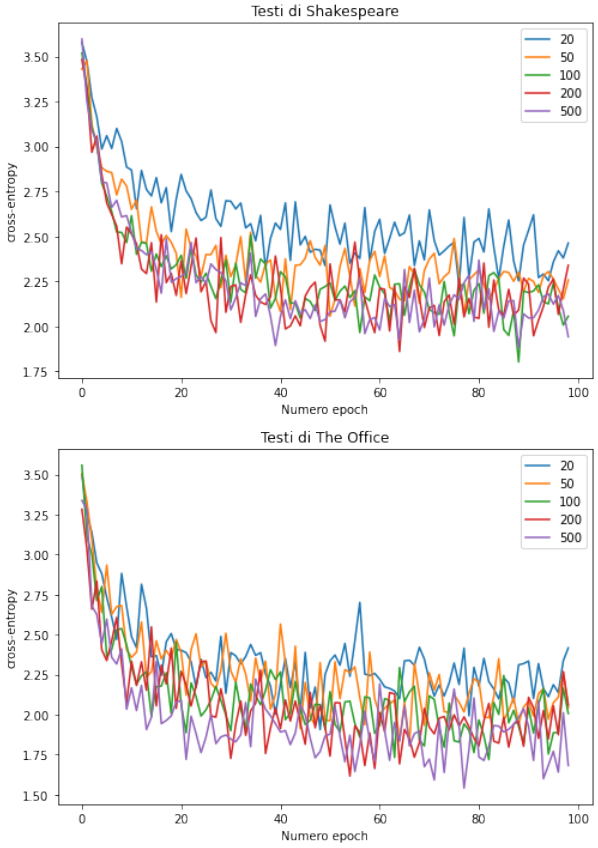


Figura 29: Cross entropy dei due modelli al variare di epoche per diverse lunghezze di testo

stata valutata perché il linguaggio presenta dei problemi a gestire la codifica bit per bit, sarebbe da implementare il metodo in un linguaggio più a basso livello.

	msg. len (chars)	rapporto di compressione Huffmann	gzip	bzip2
test set	133, 594	**0.27**	0.42	0.33
shakespeare	219, 561	0.47	0.38	0.30
wikipedia-en	24, 618	0.50	0.40	0.37
wikipedia-de	8, 426	0.70	0.50	0.48
lawsuit	207, 762	0.42	0.30	0.25

	msg. len (chars)	rapporto di compressione Huffmann	gzip	bzip2
test set	219, 561	**0.29**	0.42	0.33
The Office	133, 594	0.46	0.38	0.30
wikipedia-en	24, 618	0.65	0.40	0.37
wikipedia-de	8, 426	0.90	0.50	0.48
lawsuit	207, 762	0.57	0.30	0.25

Conclusioni Il modello analizzato è in grado, in condizioni ottimali, di ottenere un rapporto di compressione migliore dei metodi più diffusi. Questo al prezzo di dover essere allenato su un insieme di dati simile a quello che si vuole comprimere. Ha delle potenzialità se viene implementato nel contesto giusto, potrebbe velocizzare dei procedimenti che necessita di comprimere continuamente file molto simili. Bisogna tenere conto che si tratta di un modello molto semplice a scopo didattico per mostrare il funzionamento di metodi di compressione con reti neurali artificiali, per la previsione/generazione di caratteri esistono modelli molto più sofisticata come il GPT-2 [30] o GPT-3 [31] ma che spesso risultano troppo complessi, hanno milioni di parametri e ci vuole una notevole potenza di calcolo per usarli. Oltre a quello spiegato in questa tesi ci sono molte altri metodi di compressione che sfruttano reti neurali artificiali, sia senza e con perdita d'informazione, tra quelle senza perdita d'informazione un esempio sono le tecniche chiamate di bits-back coding che usano modelli a variabili latenti per modellare la sorgente dei messaggi. [32] [33]. Nel campo dei metodi con perdita d'informazione ci sono molti modelli, in particolare per file multimediali come immagini, video e audio,

in questi metodi si fa largo uso di encoder , autoencoder e variational autoencoder (VAE). [34] [9] Il modello mostrato qui è solo un esempio delle molteplici applicazioni possibili che hanno le reti neurali artificiali nell'ambito della compressione dati, già ora iniziano ad essere implementate in alcune tecnologie con buoni risultati, [35] ed in futuro se ne svilupperanno molte altre.

Riferimenti bibliografici

[1] European Parliament. What is artificial intelligence and how is it used?, 2017. https://www.europarl.europa.eu/news/en/headlines/society/20200827STO85804/what-is-artificial-intelligence-and-how-is-it-used, ultima visita 22-06-25.

[2] R. Bellman. *An Introduction to Artificial Intelligence: Can Computers Think?* Boyd & Fraser Publishing Company, 1978.

[3] D. Poole, A. Mackworth, and R. Goebel. *Computational Intelligence: A Logical Approach*. 01 1998.

[4] A.M. Turing. I.—computing machinery and intelligence. *Mind*, LIX(236):433–460, 10 1950.

[5] T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[6] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[7] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[8] D.O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949.

[9] A. Goltsev and V. Gritsenko. Modular neural networks with hebbian learning rule. *Neurocomputing*, 72(10):2477–2482, 2009. Lattice Computing and Natural Computing (JCIS 2007) / Neural Networks in Intelligent Systems Designn (ISDA 2007).

[10] H.J. Kelley. Gradient theory of optimal flight paths. *ARS Journal*, 30:947–954, 1960.

[11] S. Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962.

[12] K.D. Foote. A brief history of deep learning, 2022. https://www.dataversity.net/brief-history-deep-learning/, Last accessed on 2022-06-23.

[13] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[14] D. Gershgorn. The inside story of how AI got good enough to dominate Silicon Valley, 2018. https://qz.com/1307091/the-inside-story-of-how-ai-got-good-enough-to-dominate-silicon-valley/, ultima visita 22-06-22.

[15] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, Nal Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[16] G. Nakerst, J. Brennan, and M. Haque. Gradient descent with momentum - to accelerate or to super-accelerate? *CoRR*, abs/2001.06472, 2020.

[17] I.J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[18] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[19] D. Salomon, G. Motta, and D. Bryant. *Data Compression: The Complete Reference*. Molecular biology intelligence unit. Springer London, 2007.

[20] dProgrammer lopez. Recurrent neural network (RNN), long-short term memory (LSTM) gated recurrent unit (GRU). http://dprogrammer.org/rnn-lstm-gru, ultima visita 22-06-25.

[21] J. Hylton. The complete works of william shakespeare. http://shakespeare.mit.edu/, ultima visita 22-06-25.

[22] The entire transcript from the office in tidy format. https://github.com/bradlindblad/schrutepy, ultima visita 22-06-25.

[23] Claude shannon wikipedia-en. https://en.wikipedia.org/wiki/Claude_Shannon, ultima visita 22-06-25.

[24] Claude shannon wikipedia-de. https://de.wikipedia.org/wiki/Claude_Shannon, ultima visita 22-06-25.

[25] F. Galgani. Legal citation text classification. https://www.kaggle.com/datasets/shivamb/legal-citation-text-classification, ultima visita 22-06-24.

[26] R. Bamler. Understanding entropy coding with asymmetric numeral systems (ANS): a statistician's perspective. *arXiv preprint arXiv:2201.01741*, 2022.

[27] torch-rnn. https://github.com/jcjohnson/torch-rnn, ultima visita 22-06-25.

[28] GNU Gzip. https://www.gnu.org/software/gzip/, ultima visita 22-06-25.

[29] bzip2 and libbzip2. https://www.sourceware.org/bzip2/, ultima visita 22-06-25.

[30] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[31] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C.L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K.a Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. *arXiv*, 2022.

[32] J. Townsend, T. Bird, and D. Barber. Practical lossless compression with latent variables using bits back coding. *CoRR*, abs/1901.04866, 2019.

[33] Friso H. Kingma, P. Abbeel, and J. Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. *CoRR*, abs/1905.06845, 2019.

[34] D. Ding, Z. Ma, D. Chen, Q. Chen, Z. Liu, and F. Zhu. Advances in video compression system using deep neural network: A review and case studies. *arXiv*, 2021.

[35] Nvidia. Nvidia Deep Learning Super Sampling (DLSS). https://developer.nvidia.com/rtx/dlss, ultima visita 2022-06-25.