Quantum Deep Learning: il caso di studio delle Boltzmann Machines

Laureando
Saverio De Vito

Relatore
Prof. Fabio Antonio Bovino

Correlatore
Prof. Francesco Pugliese

SAPIENZA
UNIVERSITÀ DI ROMA

SAPIENZA
UNIVERSITÀ DI ROMA

Quantum Deep Learning: il caso di studio delle Boltzmann Machines

Dipartimento di Scienze di base ed applicate per l'Ingegneria

Facoltà di Ingegneria civile e industriale

Master di II° livello in Optics and Quantum Information

Saverio De Vito
Matricola 1931564

Relatore
Prof. Fabio Antonio Bovino

Correlatore
Prof. Francesco Pugliese

A.A. 2019-2020

Indice

Indice delle figure

1. Introduzione

Negli anni recenti, il Deep Learning ha avuto un profondo impatto e sul Machine Learning e sull'Intelligenza Artificiale. Il Deep Learning è una recente tecnica utilizzata nel Machine Learning che ha sostanzialmente condizionato la modalità con la quale le applicazioni di classificazione, inferenza ed Intelligenza Artificiale sono modellate. Questo si basa sul presupposto che, al fine di espletare sofisticate funzionalità riguardanti l'Intelligenza Artificiale, come il riconoscimento di oggetti all'interno di immagini oppure quello legato ai discorsi, possa talvolta essere necessario permettere alle macchine di addestrare un modello che contiene molteplici strati di astrazione in grado di processare il vettore dei valori di ingresso ad esse forniti. Per esempio, un modello allenato a rilevare un'automobile ammetterebbe come input un'immagine dotata di un certo numero di pixels. In uno strato della rete neurale immediatamente consecutivo, si comincerebbe ad estrarre l'oggetto "automobile" attraverso i suoi oggetti costitutivi di forma semplice. Nello strato successivo, queste forme elementari precedentemente captate sarebbero da aggregare a formare oggetti di maggiore complessità strutturale e dimensionale, quali ruote oppure fanali. Infine, gli strati ultimi della rete sarebbero preposti ad etichettare gli oggetti estrapolati dalla rete stessa. Dunque, le reti "Deep" automaticamente forniscono una complessa, innestata rappresentazione di un processamento di un quantitativo più o meno ingente di dati attraverso sistemi concepiti con innumerevoli strati, in maniera similare alla modalità con la quale il cervello umano, composto da sinapsi e neuroni, abilita la propagazione degli stimoli provenienti dal mondo esterno.

D'altra parte, occorre evidenziare che è stato dimostrato che talvolta gli algoritmi sviluppati per il Quantum Computing sono capaci a risolvere in modo più efficiente problemi che sarebbero intrattabili in un'ottica convenzionale, legata ai calcolatori tradizionali.

Scopo di questo elaborato è dimostrare che l'approccio "Quantum" non solo ridurebbe il tempo richiesto ad addestrare una cosiddetta "Restricted Boltzmann Machine", ma consentirebbe inoltre di condurre in modo più efficiente il processo di addestramento di "Boltzmann Machine" e di macchine multistrato con strati "fully connected", ovvero costituite da strati in cui ciascun neurone artificiale è collegato a tutti i neuroni dello strato precedente.

Le macchine di Boltzmann rappresentano una classe di reti neurali "Deep" e si ascrivono all'ambito delle reti ricorrenti. Da un punto di vista fisico, le macchine di Boltzmann elaborano i dati facenti parte del Training Dataset (usati per addestrare la rete stessa) mediante il cosiddetto modello di Ising all' equilibrio termico. Quest' ultimo è caratterizzato da due concetti importanti:

- i nodi, i quali sostanzialmente nella letteratura del Machine Learning codificano le varie "*features*", ossia l'insieme delle peculiarità oggetto di interesse nell'ambito del processo di riconoscimento; si dividono come tipologia in "*visible nodes*" in grado di codificare direttamente i valori in uscita della rete neurale e in "*hidden nodes*" rappresentanti lo spazio totale delle "*features*", processate in passi differenti dell'intero processo di riconoscimento;
- "*edges*", che nel modello di Ising indicano le dipendenze statistiche delle "*features*".

Fondamentalmente, sono note due importanti classi delle macchine di Boltzmann:

- "*Restricted Boltzmann Machine*" le quali hanno la particolarità di interpretare in forma bipartita il sopraccitato grafico delle dipendenze statistiche;
- "*Deep Restricted Boltzmann Machine*" concepite sotto il profilo architetturale come reti multi-strato.

Le macchine di Boltzmann sono impiegate per risolvere due tipologie abbastanza differenti di problemi computazionali. Per quanto riguarda la ricerca di informazioni, i pesi e le connessioni della rete sono fissati ed immutabili e vengono utilizzati per esprimere una funzione di costo. Le dinamiche stocastiche caratterizzanti il modello di Boltzmann consentono poi di campionare vettori di stato binari in grado di ottenere valori ridotti della funzione di costo.

Per un problema di apprendimento, invece, vengono prodotti dalla macchina vettori di dati binarizzati dovendo questi, come finalità ultima dell'applicazione, essere generati con elevate probabilità. Per poter espletare queste funzionalità, devono essere individuati pesi e connessioni della rete in modo tale che, relativamente ad ulteriori e differenti ingressi della rete intesi come vettori di dati binarizzati, i valori di uscita permettano il mantenimento della funzione di costo a determinati valori di soglia. Per risolvere un problema di apprendimento, le macchine di Boltzmann apportano continuamente piccole variazioni ai loro pesi e ciascun aggiornamento richiede la risoluzione di molteplici problemi di ricerca.

2. Intelligenza artificiale: dal Machine Learning al Deep Learning

La terminologia "Intelligenza Artificiale" fa riferimento ad agenti hardware e software che manifestano un comportamento intelligente da una prospettiva umana, e, in generale, possono o non possono essere addestrati su un Training Dataset. A titolo di esempio, si consideri a tal proposito il lancio di un vettore per missioni spaziali: questa situazione non prevede un addestramento del sistema attraverso esempi, il suo comportamento è basato dunque su algoritmi deterministici.

D'altra parte, Machine Learning e Deep Learning definiscono un paradigma tecnologico per il quale un dato sistema è in grado di acquisire intelligenza attraverso esempi (si parla di "apprendimento automatico"). Questo assunto comporta che nel Machine Learning e Deep Learning gli algoritmi siano stocastici, ovvero basati sul concetto di probabilità.

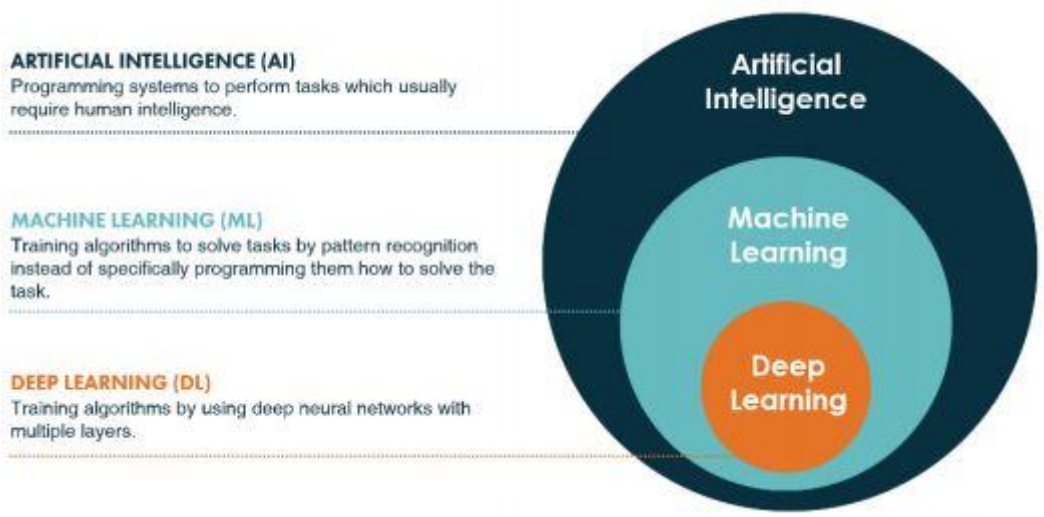

Figura 1:Artificial Intelligence, Machine Learning e Deep Learning.

La differenza fra Machine Learning e Deep Learning consiste nel fatto che, a livello teorico, *il Multi-Layer Perceptron* (percettrone dotato di almeno uno strato intermedio, *hidden*) è in grado potenzialmente di risolvere qualsiasi tipo di problema di classificazione (anche lo *XOR-Problem*) non linearmente separabile. Questa affermazione è, peraltro, confermata dal Teorema di Approssimazione Universale, secondo il quale una rete cosiddetta "*feed-forward*" con un singolo strato latente contenente un numero finito di neuroni può approssimare arbitrariamente funzioni continue su sottoinsiemi compatti facenti parte di R^n. In sostanza, questo teorema certifica che semplici reti neurali possono rappresentare una grande varietà di funzioni interessanti, qualora vengano assegnati parametri o pesi

appropriati. Tuttavia, esso garantisce solo l'esistenza di una rete che soddisfi determinate condizioni ma purtroppo non fornisce alcun metodo pratico per individuarne la struttura o i parametri corrispondenti. Per questo motivo si tratta di un Teorema non costruttivo, bensì di esistenza.

Passando dunque all'aspetto pratico della questione, gli algoritmi di addestramento delle reti neurali (i più comuni sono del tipo "*Gradient Descent*") sono limitati e si dimostra che tale tipologia non opera efficacemente quando si lavora con reti neurali dotate di strati "*fully-connected*" aventi immagini in input, per cui in questo caso l'esplosione combinatoria dei pesi non permette il corretto funzionamento degli algoritmi stessi di addestramento. Da qui nasce l'esigenza di adottare in applicazioni di *Computer Vision* come la classificazione di immagini reti neurali dotate di strati convoluzionali ("*Convolutional layers*") in grado di garantire la "*feature extraction*", cioè il rilevamento di opportune caratteristiche di una data immagine in maniera maggiormente dettagliata. Tale considerazione, al livello infrastrutturale, è supportata dal fatto che uno strato convoluzionale è tridimensionale, mentre il suo corrispondente strato di un *Multi-Layer Perceptron* è bidimensionale, a dimostrazione del fatto che la non adozione di reti "Deep" può implicare una perdita di informazione estratta. Pertanto, l'idea legata all'impiego di reti neurali convolutive si coniuga con la necessità di approntare sistemi di Intelligenza Artificiale più complessi da un punto di vista architetturale rispetto allo standard del *Multi-Layer Perceptron*, in cui, però, ogni strato della rete è composto da un numero limitato di parametri (dell'ordine, in ogni caso, di qualche milione) cosicché sia possibile ottenere rappresentazioni più accurate dei valori di uscita del sistema.

2.1 Apprendimento supervisionato vs non supervisionato

Le reti neurali possono essere addestrate utilizzando tre tecniche differenti. Nell'addestramento supervisionato i dati di input sono etichettati, ossia conosciamo il valore della variabile risposta. Gli algoritmi che vengono addestrati in questo modo richiedono un set di dati costituito da coppie input-output. Questo tipo di addestramento viene ad esempio utilizzato nei problemi di classificazione; la rete neurale, addestrata sui dati di training di cui conosciamo l'etichetta (*label*) associata, deve stimare l'output, ossia la *label* dei nuovi dati chiamati dati di test commettendo il minimo errore possibile. In questo caso l'algoritmo deve stimare la distribuzione di probabilità dell'output condizionata ai valori dell'input $p(\,|\,)$. I

principali problemi che vengono risolti utilizzando questo tipo di addestramento sono i problemi di classificazione che concernono ad esempio il riconoscimento di immagini, la *fraud detection* o la diagnostica di una malattia e problemi di regressione in cui prevediamo un valore numerico continuo dai dati di input come ad esempio la valutazione del prezzo di una casa, la previsione della temperatura atmosferica o il reddito medio annuo di un soggetto.

Se dovessimo risolvere problemi di clusterizzazione come ad esempio la *customer segmentation*, non necessariamente avremmo bisogno del *target* per i dati di *training*. In questo caso l'addestramento è detto non supervisionato. In questo tipo di apprendimento il modello dovrebbe stimare la distribuzione di probabilità dell'input $p(|)$ o cogliere delle caratteristiche di tale distribuzione senza avere il suggerimento aggiuntivo dato dalla conoscenza del *target*.

Una rete neurale potrebbe altresì essere addestrata con addestramenti semi-supervisionati in cui solo una parte dei dati utilizzati in fase di training sono etichettati. Questo permette alla rete neurale di avere maggiori gradi di libertà nello sfruttare le informazioni disponibili. Questo tipo di apprendimento può essere anche utilizzato per la riduzione della dimensionalità dei dati di input in modo da cogliere le discriminanti principali del fenomeno oggetto di studio.

Il *Reinforcement Learning* invece fornisce all'algoritmo degli strumenti necessari per migliorare il proprio addestramento.

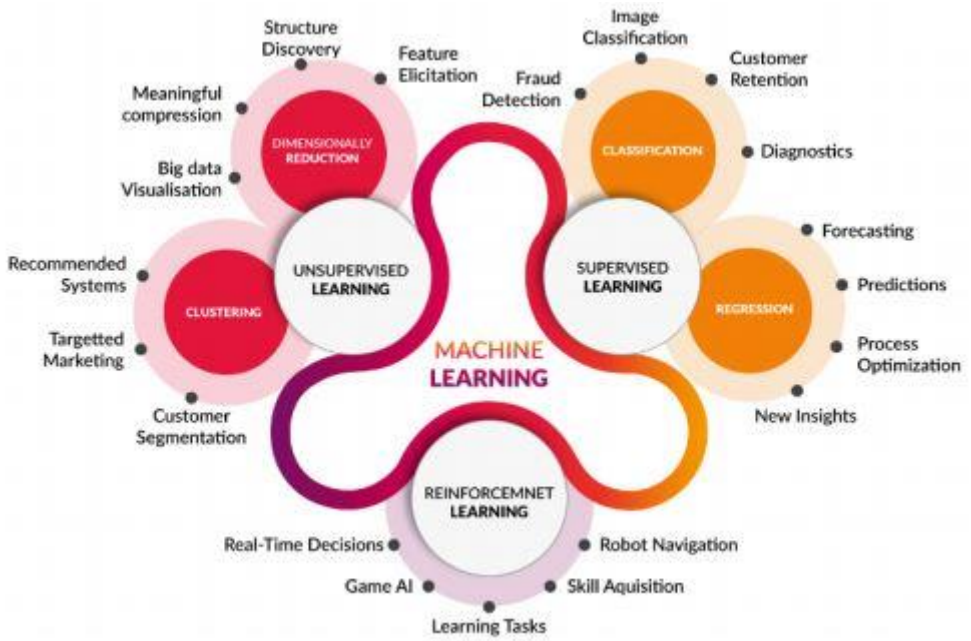

Figura 2: Supervised, Unsupervised Learning and Reinforcement Learning.

Prevedono la presenza di un agente in grado di ricevere informazioni relative all'ambiente circostante, rispetto alle quali reagisce ed esegue un'azione in totale autonomia. Le informazioni che vengono inviate all'agente sottoforma di dati numerici sono chiamate *states*. All'agente non viene detto quale azione intraprendere, deve scoprire invece quale azione porterà ad ottenere un *reward* maggiore, semplicemente esplorando l'insieme delle azioni possibili. Un agente riceve una ricompensa positiva o negativa utile per migliorare l'apprendimento. Il funzionamento degli algoritmi di *Reinforcement Learning* è schematizzato nella Figura 3.

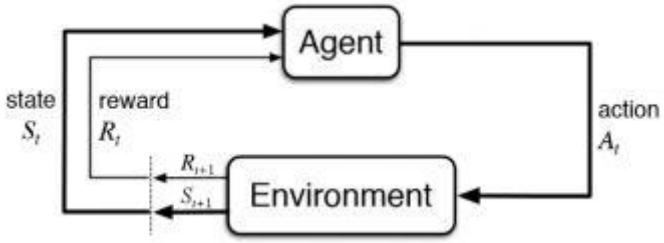

Figura 2: How to work a Reinforcement Learning algorithm.

Questo tipo di addestramento è stato utilizzato anche per progettare l'architettura di una rete neurale; un agente è addestrato a scegliere in sequenza gli strati di una *Convolutional Neural Network* (CNN). Esplorando un ampio ma limitato spazio di possibili architetture, è in grado di scoprire l'architettura con una prestazione migliore rispetto al problema da risolvere.

2.2 Nascita del Deep Learning e cenni storici salienti

Mentre gli algoritmi di *Machine Learning* sono in circolazione da molto tempo, è di recente sviluppo la possibilità di applicare automaticamente complessi calcoli matematici a dati su larga scala. Questo perché la maggiore potenza dei computer di oggi, in termini di velocità e memoria, ha aiutato le tecniche di apprendimento automatico ad evolversi per imparare da un ampio *Dataset* di addestramento. Ad esempio, con più potenza di calcolo e una memoria abbastanza grande, è possibile creare reti neurali di molti strati, che sono chiamate *Deep Neural network*, ovvero reti neurali profonde. Si può attribuire la nascita del *Deep Learning* a diversi momenti storici che hanno portato prima alla scoperta delle reti neurali profonde e poi alla risoluzione dei problemi grazie a questi modelli. Per quanto riguarda il trend delle ricerche effettuate su Google circa le parole *Machine Learning* e *Deep Learning*, il grafico della figura 2.4 mostra un andamento crescente da ottobre 2015, mese in cui il programma

Google AlphaGo sviluppato da *DeepMind* vinse al gioco strategico del *Go* contro il campione europeo *Mr Fan Hui*.

Figura 3: Trend mondiale del Machine Learning "rosso" e del Deep Learning "blu"

Da quel momento, il Deep Learning, è passato sotto gli occhi dei media. Questo ne ha consolidato la popolarità e come si può vedere dal grafico, le ricerche hanno iniziato ad avere un andamento sempre più crescente. Il secondo picco nel grafico può essere attribuito a DeepMind e in particolare alla vittoria 4-1 di AlphaGo a Seoul, in Corea del Sud, nel marzo 2016 che ha avuto un ascolto televisivo di oltre 200 milioni di persone in tutto il mondo. Anche in Italia queste notizie non sono passate inosservate. Infatti, nella figura 2.5, si possono notare gli stessi andamenti avvenuti a livello mondiale.

Figura 4: Trend mondiale del Machine Learning "rosso" e del Deep Learning "blu"

Il sistema di Google DeepMind ha combinato l'algoritmo "*Monte-Carlo Tree Search*" con le *Deep Neural Networks* che sono state addestrate in maniera supervisionata. In particolare, il termine "*Deep Learning*" è stato usato per descrivere il modo in cui DeepMind ha vinto.

Per concludere questa panoramica iniziale, il Deep Learning è una area del Machine Learning, che usa come base le reti neurali ma con la distinzione che le stesse presuppongono in questa circostanza l'aggiunta di diversi strati per migliorare l'apprendimento del modello. Le reti neurali sono bio-ispirate al funzionamento della corteccia cerebrale visiva. Ogni neurone può connettersi a qualsiasi altro neurone entro una certa distanza fisica. Le reti neurali artificiali hanno livelli discreti di connessioni e direzioni di propagazione dei dati. Fino a qualche anno

fa le reti neurali, come area del Machine Learning, erano state completamente ignorate dalla comunità di ricerca. Il problema era l'intensivo calcolo computazionale che richiedevano anche per i modelli più semplici. Fino a quando, il gruppo di ricerca guidato da Geoffrey Hinton, presso l'Università di Toronto, ha finalmente parallelizzato gli algoritmi per le reti neurali. La svolta finale è stata di Andrew Ng che ha aggiunto strati e neuroni artificiale di tipologie variegate alla rete per addestrare il sistema su enormi quantità di dati. Nel caso di Andrew Ng si trattava di immagini prese da 10 milioni di video su YouTube. Ng ha messo il termine “Deep" per descrivere l’enorme numero di livelli che hanno queste reti neurali.
Quindi, il primo requisito di un modello Deep è avere a disposizione un grande Dataset per fare l’addestramento. Questo perché ci sono un gran numero di parametri che devono essere compresi. Le ragioni per cui il Deep Learning è diventato così popolare e utilizzato sono: i *Big Data* e le *Graphic Processing Unit* (GPU). Ad esempio, un algoritmo di Apprendimento Profondo potrebbe essere istruito per "imparare" a riconoscere un cane. Per distinguere i dettagli che differenzino un cane da un lupo, c’è bisogno di un enorme Dataset su cui addestrare il modello, e di conseguenza potenza computazionale (GPU).

2.3 Perché si è affermato il Deep Learning

Negli ultimi anni le reti neurali profonde hanno raggiunto notevoli progressi nella ricerca di molti campi come l'Intelligenza Artificiale e il Machine Learning. Il Deep Learning è stato applicato con successo al riconoscimento vocale e in molti altri settori del Machine Learning. Finora, in tutti gli esperimenti, le prestazioni risultanti erano di gran lunga migliori di altre tecniche di apprendimento automatico disponibili. Una delle proprietà che conferiscono al Deep Learning un posto speciale è la capacità di estrarre concetti significativi (con una prospettiva umana) dai dati, combinando le caratteristiche basate sulla struttura dei dati. I concetti estratti a volte hanno una chiara interpretazione e sembra che le macchine abbiano effettivamente "appreso qualcosa". Le ragioni di questo successo derivano dai tre vantaggi dell'approccio del Deep Learning:

• Semplicità: anziché ritocchi specifici dei problemi e rilevatori di caratteristiche su misura, le reti profonde offrono blocchi architetturali di base, livelli di rete, che vengono ripetuti più volte per generare reti di grandi dimensioni;

• Scalabilità: i modelli di apprendimento approfondito sono facilmente scalabili in enormi Dataset. Altri metodi concorrenti, ad esempio, le *kernel machines*, incontrano gravi problemi computazionali se i Dataset sono enormi;

• Trasferimento del dominio: un modello appreso su un compito è applicabile ad altre attività correlate e le funzionalità apprese sono abbastanza generali da poter essere utilizzate su una serie di attività che potrebbero avere scarsi dati disponibili.

A causa dell'enorme successo nell'apprendimento delle reti neurali profonde, le tecniche di Deep Learning sono al momento all'avanguardia per il rilevamento, la segmentazione, la classificazione e il riconoscimento (cioè identificazione e verifica) degli oggetti nelle immagini. Ma, negli ultimi anni, gli avvenimenti che hanno determinato l'enorme interesse per il Deep Learning, derivano dai miglioramenti tecnologici. Possiamo identificare il più importante di loro come segue.

Innanzitutto, l'avvento di potenti *GPU* rende possibile l'addestramento di reti neurali molto grandi con oltre 60 milioni di parametri. Una *Graphical Processor Unit* (*GPU*) è un processore che esegue il *Rendering* di immagini, animazioni e video per lo schermo del computer. Le *GPU* si trovano solitamente nelle schede video dei computer ed eseguono operazioni parallele. Fondamentalmente le *GPU* sono architetture *SIMD* (*Single Instructions Multiple Data*) che rientrano nella categoria dei processori di Array. Grazie a questo parallelismo, le *GPU* hanno portato alla nascita del Computing accelerato dalla *GPU* che è l'uso di *GPU* insieme a una *CPU* per accelerare applicazioni scientifiche, di analisi, di ingegneria, di consumo e aziendali. Un modo semplice per comprendere la differenza tra una *CPU* e una *GPU* è confrontare il modo in cui elaborano le attività. Una *CPU* è costituita da pochi core ottimizzati per l'elaborazione seriale, mentre una *GPU* ha un'architettura massicciamente parallela composta da migliaia di core più piccoli ed efficienti progettati per gestire più attività contemporaneamente. Il Computing con accelerazione *GPU* offre prestazioni applicative senza precedenti, scaricando porzioni di elaborazione intensiva dall'applicazione alla *GPU*, mentre il resto del codice gira ancora sulla *CPU*. Dal punto di vista dell'utente, le applicazioni vengono eseguite molto più velocemente. Pioniere nel 2007 fu NVIDIA, adesso gli acceleratori *GPU* alimentano i Data Center ad alta efficienza energetica nei laboratori governativi, nelle università, nelle imprese e nelle piccole e medie imprese di tutto il mondo. Le *GPU* stanno accelerando le applicazioni su piattaforme che vanno dalle automobili, ai telefoni cellulari e ai tablet. Per questi motivi l'elaborazione accelerata da *GPU* è diventata la piattaforma di riferimento per una metodologia computazionalmente costosa come il Deep Learning.

2.4 Reti neurali convolutive: struttura generale e nozioni teoriche di base

La struttura generale di una non specificata rete neurale si articola come segue:

- L: numero di strati di una rete neurale;
- p_1: numero dei neuroni dello strato di input;
- p_l: numero di neuroni dello strato *l – esimo*;
- x_i: *i – esimo* input;
- a_j^l: valore dell'output del *j – esimo* neurone dello strato *l – esimo*;
- $w_{i_j}^{\boldsymbol{l}}$**:** peso della connessione tra *i − esimo* neurone dello strato *l − esimo* e il *j − esimo* neurone dello strato *l + 1 − esimo*;
- y_k: valore del *j − esimo* neurone dello strato di output.

Il primo strato di una rete neurale è lo *strato di input*, costituito da tanti neuroni quanti sono gli input che riceve. Il valore dei neuroni di questo strato eguagliano i valori degli input. Questa informazione viene propagata al primo *hidden layer;* ognuno dei neuroni di questo primo strato nascosto associa un peso alle connessioni con i neuroni dello strato di input generando la seguente relazione:

$$z_j^{(2)} = w_{0j}^{(1)} + \sum_{i=1}^{p_1} w_{ij}^{(1)} x_i$$

Si noti che alla combinazione lineare tra gli input e i pesi viene aggiunta una quantità, *bias* $w_{0_j}^{(\mathbf{1})}$ allo scopo di modellare un'eventuale distorsione. Successivamente il neurone sottopone questa quantità $z_j^{(2)}$ad una funzione di attivazione non lineare *g(·)* in questo modo:

$$a_j^{(2)} = g^{(2)}(z_j^{(2)})$$

La relazione tra lo strato *(l – 1) − esimo* e lo strato *l − esimo* è la seguente:

$$z_j^{(l)} = w_{0j}^{(l-1)} + \sum_{i=1}^{p_{L-1}} w_{ij}^{(l-1)} a_i^{(l-1)}$$

Allo stesso modo, il risultato viene sottoposto alla funzione di attivazione:

$$a_j^{(l)} = g^{(l)}(z_j^{(l)})$$

Per lo *strato di output* si ottiene:

$$z_k^{(L)} = w_{0k}^{(L-1)} + \sum_{i=1}^{p_{L-1}} w_{ik}^{(L-1)} a_i^{(L-1)}$$

$$y_k = g^{(L)}(z_k^{(L)})$$

Il numero k di neuroni che costituiscono lo *strato di output* e la sua funzione di attivazione dipendono dal problema che la rete neurale deve risolvere. Nel caso di un problema di *classificazione* k sarà uguale al numero delle classi e la funzione di attivazione sarà la *softmax* che esporremo più avanti. In un problema di *regressione lineare* lo strato di output sarà costituito da un unico neurone che potrebbe avere una funzione di attivazione lineare.

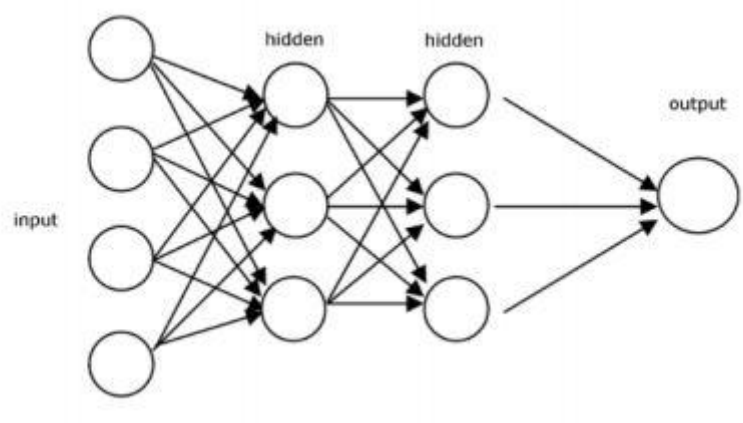

Figura 5: Architettura di una rete neurale costituita da uno strato di input, due strati nascosti e uno strato di output.

Pensando alla rete neurale come ad una funzione matematica che modelli i nostri dati di input, con l'addestramento di una rete neurale cerchiamo di scoprire le relazioni lineari e non lineari che legano le variabili di input all'output. In particolare, cerchiamo di stimare i parametri o l'insieme dei pesi di questa funzione minimizzando una funzione di perdita.

Se non ci fossero le funzioni di attivazioni non lineari, l'unica operazione che potremmo compiere è il prodotto scalare tra le variabili di input e la matrice dei pesi, il che è un'operazione lineare, ed in seguito propagare questa informazione da uno strato all'altro della rete neurale. In questo modo, potrebbe essere stimata solamente una relazione lineare tra le variabili esplicative e l'*outcome*, non riuscendo a stimare relazioni di tipo non lineari che intercorrono nella maggior parte dei casi. Una rete neurale senza funzione di attivazione è un modello di *regressione lineare*.

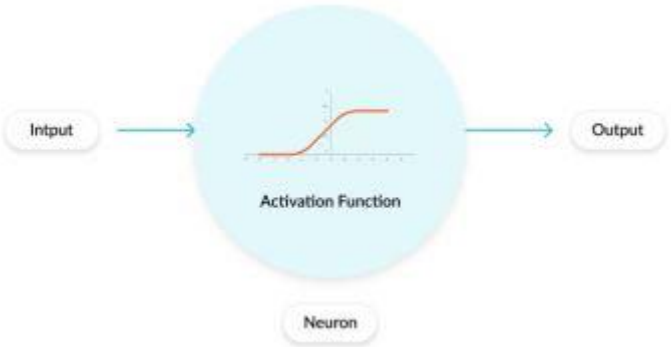

Figura 6: Un neurone riceve gli input, assegna i pesi alle connessioni e conferisce alla somma pesata la proprietà di non linearità attraverso la funzione di attivazione.

2.5 Funzioni di attivazione

Si può facilmente dedurre che esistono due tipi di funzione di attivazione, lineari e non lineari; quest'ultime sono quelle maggiormente utilizzate. Oltre a conferire la proprietà di non linearità per quanto concerne le attivazioni non lineari, decidono se un neurone verrà attivato o meno contribuendo al processo di addestramento e quindi se ci sarà un flusso di informazione. Per fare questo, bisogna innanzitutto definire una soglia di attivazione: se l'*output* della funzione di attivazione è più grande di una certa soglia il neurone sarà attivato, altrimenti sarà disattivato, di conseguenza il suo *output* non verrà propagato al successivo *hidden layer*, comportando l'esclusione di quel neurone dalla fase di training. Vengono di seguito illustrate alcune delle funzioni di attivazioni maggiormente impiegate nelle reti neurali:

- **Sigmoide :** $sigmoid\ (z) = \frac{e^z}{1+e^z}$

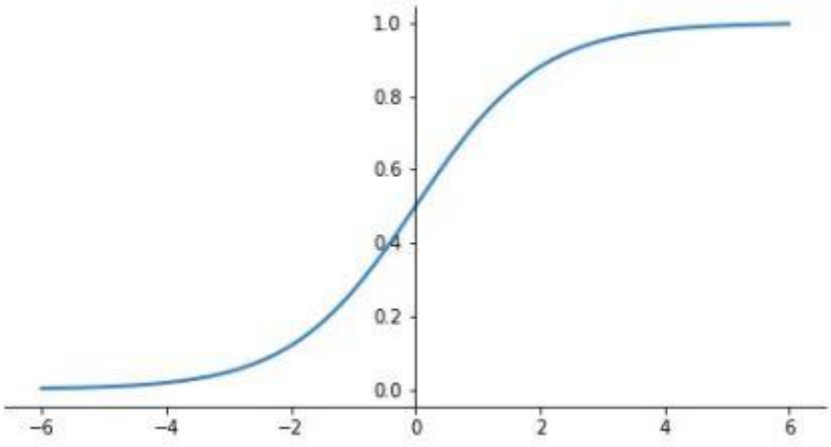

Figura 7: Funzione di attivazione Sigmoide.

- **Tangente Iperbolica :** $tanh\ (z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

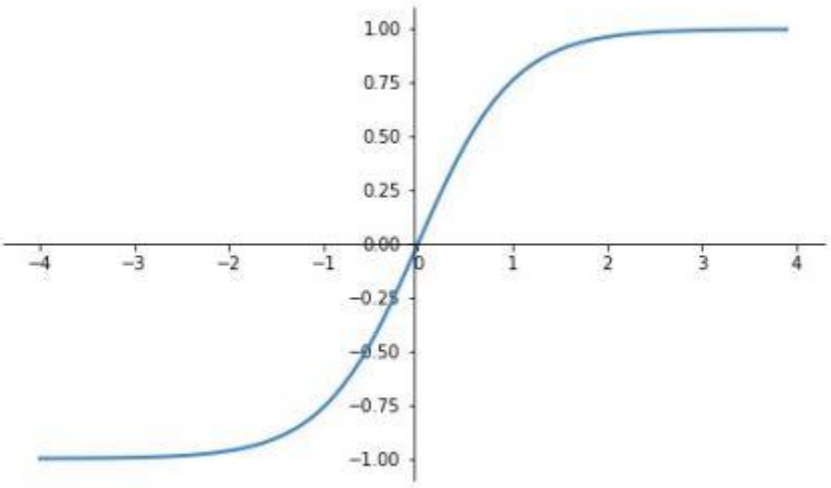

Figura 8: Funzione di attivazione tangente iperbolica.

- **ReLU (Rectified Linear Unit) :** $relu\ (z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$

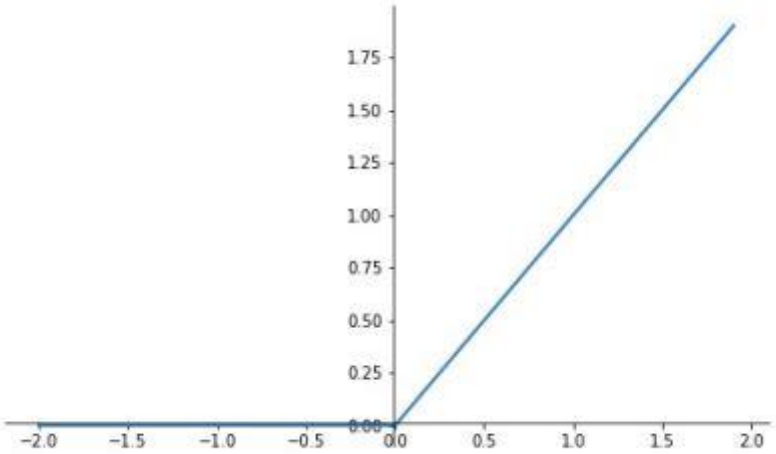

Figura 9: Funzione di attivazione ReLU.

- **Softmax**: la funzione di attivazione Softmax è un tipo di funzione sigmoide che viene utilizzata esclusivamente nello strato di output perché ridimensiona i neuroni tra 0 e 1 nel caso in cui l'obiettivo sia la stima delle probabilità di appartenenza alle classi nei problemi di classificazione. Quando risolviamo un problema di classificazione vogliamo che le probabilità di appartenenza sommate diano 1, in modo tale che la probabilità più alta si riferisca alla classe di appartenenza più probabile rispetto alle altre. Così facendo, lo strato di output è in grado di fornisci una vera e propria distribuzione di probabilità, garantita dal fatto che l'uscita di ogni neurone dello strato di output con funzione di attivazione softmax dipende da tutti i valori che riceve lo strato e non dal singolo valore che riceve il neurone, come accade in tutti gli altri strati con funzioni di attivazione diverse dalla softmax.

$$softmax\ (z_i) = \frac{e^{z_i}}{\sum_{i=0}^{m} e^{z_i}}$$

Step Function : $stepFunction\ (z)\ = \begin{cases} 1, & z > 0 \\ 0, & altri\ casi \end{cases}$

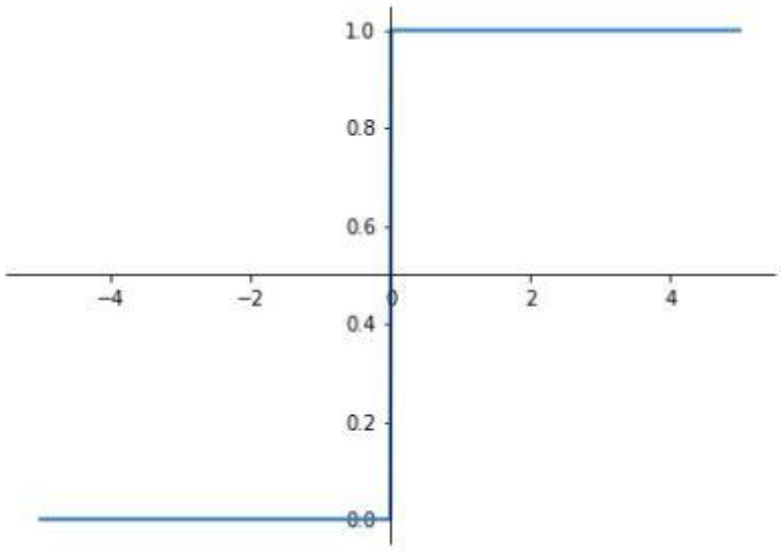

Figura 10: Funzione di attivazione stepFunction.

2.6 Tipologie di strati di una rete neurale artificiale

In una rete neurale si possono distinguere tre principali tipi di layer, ognuno con una specifica funzione: *Convolutional Layer*, *Pooling Layer*, *Fully-connected Layer*.

Convolutional layer

Se pensassimo al processo di apprendimento del cervello umano come ad un processo graduale, saremmo in grado di capire in che modo lavora un *layer* convoluzionale. Il compito degli strati convoluzionali è di estrarre le *features* dall'input con una sorta di scanner che tecnicamente prende il nome di filtro, kernel o *feature detector* del layer convoluzionale, ossia una matrice che scorre sull'input fino a scannerizzarlo integralmente. Il risultato di questa operazione è una nuova matrice detta *Feature Map* o *Convolved Feature*. Prendiamo in considerazione il problema della classificazione di un'immagine che non è altro che una matrice di numeri compresi tra 0 e 255 ognuno corrispondente all'intensità di un pixel. Le immagini RGB, in particolare, possono essere pensate come uno *stack* di 3 matrici, corrispondenti rispettivamente alle scale del rosso, del verde e del blu. Il numero di queste matrici, 3 nel caso di un'immagine RGB e 1 nel caso di un'immagine in *grayscale*, viene chiamato *channel*.

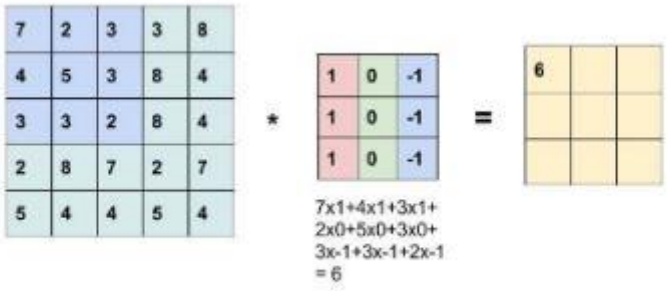

Figura 11: Convolutional layer.

Tra l'input di partenza, e il kernel viene calcolato il prodotto elemento per elemento. Questi numeri successivamente verranno sommati per ottenere il primo elemento della *Feature Maps*. Tale operazione dovrà essere effettuata per ogni matrice del canale nel caso di immagini RGB. Kernel differenti possono produrre diverse *Feature Maps* dello stesso input, non a caso quando viene costruita una rete neurale, per i filtri convoluzionali, bisognerà anche definire la profondità, ossia quanti filtri vogliamo utilizzare nell'operazione di convoluzione. Ogni filtro produrrà una *Feature Map* che possiamo pensare anche in questo caso come uno *stack* di matrici. Inoltre, bisognerà decidere lo *stride* del filtro; con stride pari a 1 il kernel si sposterà di un pixel per volta, con stride pari a 2 di due pixel per volta e così via. Ne consegue che più è grande lo stride, più sarà piccola la Feature Map. Un numero maggiore di kernel e uno stride più piccolo potrebbero migliorare l'apprendimento a discapito del tempo computazionale.

Pooling layer

Al fine di ridurre la potenza computazionale degli algoritmi richiesta per processare l'input, vengono introdotti gli strati di *pooling* con la funzione di ridurre la dimensione spaziale ed estrarre solamente le *features* dominanti. Questa operazione permette alla rete di dover calcolare un numero inferiore di parametri ed evitare l'*overfitting*. Anche il *pooling layer* è dotato di un filtro o kernel che comporta una distinzione tra i diversi strati di *pooling*. Gli strati di *max pooling* calcolano il più grande valore della porzione di input processata dal kernel, diversamente dagli strati di *average pooling* che calcolano la media dei valori di questa porzione. Sarà necessario definire lo *stride* del filtro, come per gli strati convoluzionali. La funzione dello strato di *pooling* è legata al concetto di *spatial invariance*; ad esempio nel

caso della classificazione delle immagini, la differenza di luminosità, o differenti angolazioni non dovrebbero compromettere l'apprendimento.

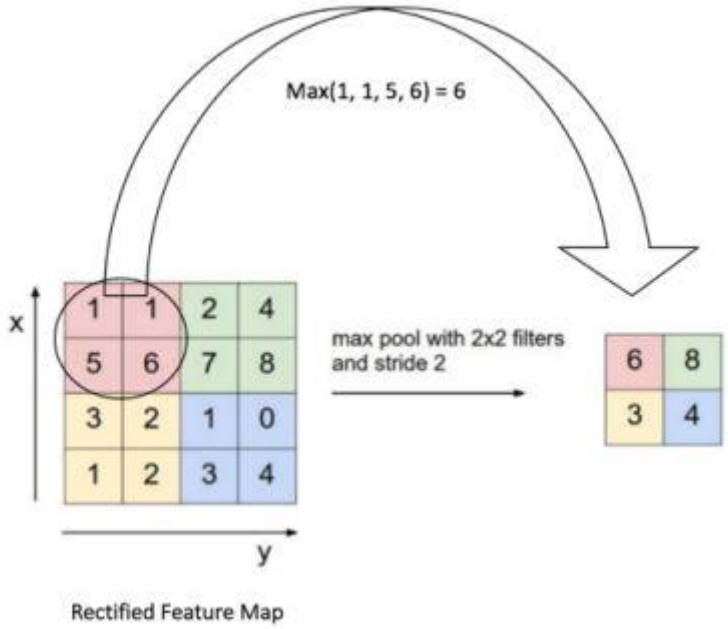

Figura 12: Max Pooling Layer.

Fully-Connected layer

Sono rappresentati da uno *stack* di strati completamente connessi come indicato dalla stessa denominazione. Dopo aver appreso le caratteristiche di alto livello dell'input dalla *Feature Maps* generate dagli strati precedenti, questi strati cercano le caratteristiche maggiormente correlate con una particolare classe, nel caso stessimo risolvendo un problema di classificazione. L'ultimo strato di questo *stack* è lo strato di *output* che ha il compito di fornire la predizione della classe, stimando una vera e propria distribuzione di probabilità. Considerando al solito la classificazione di un'immagine, l'output con un valore più alto e quindi la probabilità più alta, sarà associato alla classe che più delle altre rappresenta la stima del vero target dell'immagine, date le sue caratteristiche.

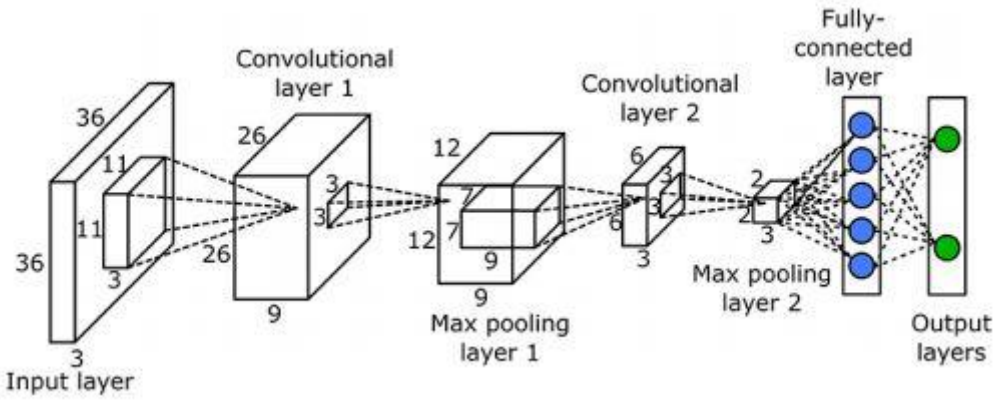

Figura 13: Nell'immagine sono evidenziati il Fully-Connected Layer e l'Output Layer.

3. Boltzmann Machine

3.1 Introduzione generale

Una macchina di Boltzmann rappresenta una rete neurale del tipo "*neuron-like*" generalmente composta da un numero indefinito di strati deputati alla cosiddetta "*feature-detection*". Come si vedrà in seguito, differentemente dalle comuni reti neurali la logica di attivazione dei nodi segue dinamiche stocastiche.

L'utilizzo del suddetto sistema si colloca principalmente nell'ambito delle metodologie "*unsupervised*" di Machine Learning ed in particolare dei modelli generativi che verranno illustrati a breve.

Al fine di delineare dettagliatamente le caratteristiche di una generica Boltzmann Machine è d'obbligo far propri alcuni concetti inerenti modelli di reti neurali.

A questo proposito, a livello tecnico una BM si configura come un sistema "*Autoencoder*", vale a dire una tipologia di rete neurale atta a funzionare attraverso due processi:

- **Encode**: creazione di una rappresentazione compressa di una data immagine in ingresso al sistema;
- **Decode**: ricostruzione dell'immagine finale estrapolata dalla rete in modo quanto più fedele possibile al dato originale presentato in input. Dunque, come evidenzia l'architettura classica di un "*Autoencoder*", questa tipologia di rete mira ad una riduzione della dimensionalità

dell'informazione da estrarre nello spazio latente, per poi ricondurre le "*features*" estrapolate all'immagine in output nello spazio osservato.

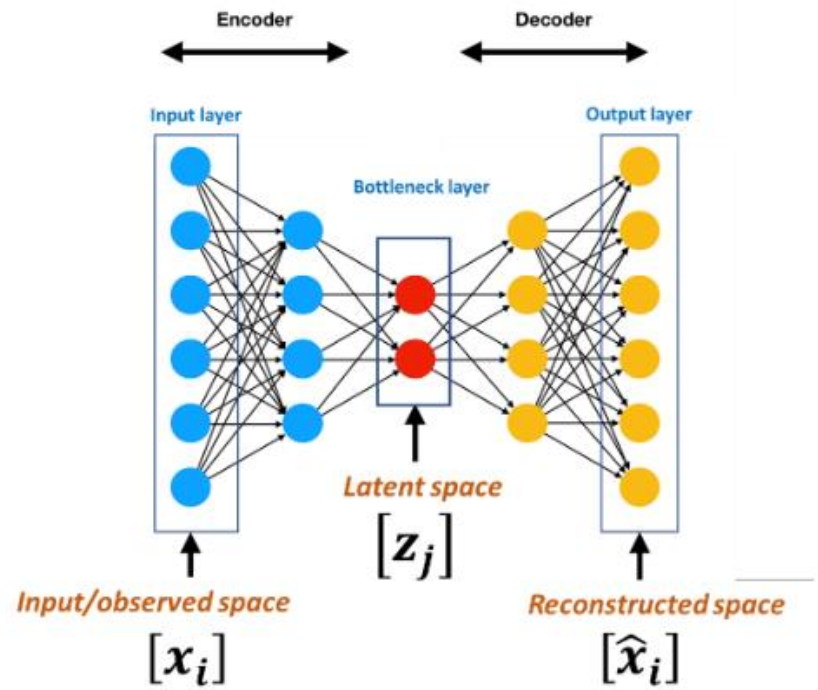

Figura 14: Architettura di una rete "Autoencoder".

Una volta che l'"*Autoencoder*" è addestrato, la parte adibita alla "*feature extraction*" viene resa superflua mentre la parte "*Decode*" inizia a generare i dati nello spazio osservato agendo nello spazio latente attraverso la creazione di campioni randomici, i quali vengono poi "mappati" nello spazio osservato. Tale funzionamento è proprio di un modello generativo, che costituisce la base tecnica di una qualunque macchina di Boltzmann. La differenza, tuttavia, consta nell'architettura, nella rappresentazione dello spazio latente e nel processo di addestramento.

Proseguendo, un ulteriore importante concetto è dato dalla "*Density Estimation*" intesa come la distribuzione di probabilità con la quale una variabile si attesta ad un determinato valore. Si parla di "*Density Estimation*" in quanto è impossibile nella pratica essere a conoscenza di tutti i valori di una variabile randomica ragion per cui non si dispone della effettiva funzione densità, ma di una stima della stessa scaturita da un'osservazione limitata nel tempo. La conoscenza della stima della funzione densità è fondamentale per un modello generativo.

I modelli generativi sono caratterizzati da due tipologie di funzioni densità:

- **Explicit Density Estimation (*EDE*):** in questo caso funzioni densità predefinite sono utilizzate per approssimare il legame fra le osservazioni e le probabilità annesse. I dati osservati sono

adattati per la funzione predefinita manipolando un set fissato di parametri della funzione. Ad esempio, il "*fitting*" delle informazioni può venire adottato attraverso l'uso di una distribuzione normale ottenuta con media e deviazione standard. Si parla così talvolta di "funzione densità parametrica";

- **Implicit Density Estimation (*IDE*):** in tale situazione, non vengono adoperate funzioni predefinite, ma si ricorre ad un algoritmo per approssimare la distribuzione di probabilità dei dati. Anche se i metodi *IDE* usano parametri per effettuare approssimazioni, questi non possono essere manipolati allo stesso modo di quanto accade in *EDE*.

A titolo di completezza, si riporta la tassonomia di differenti modelli generativi basati sulla tipologia di stime di funzioni densità utilizzate.

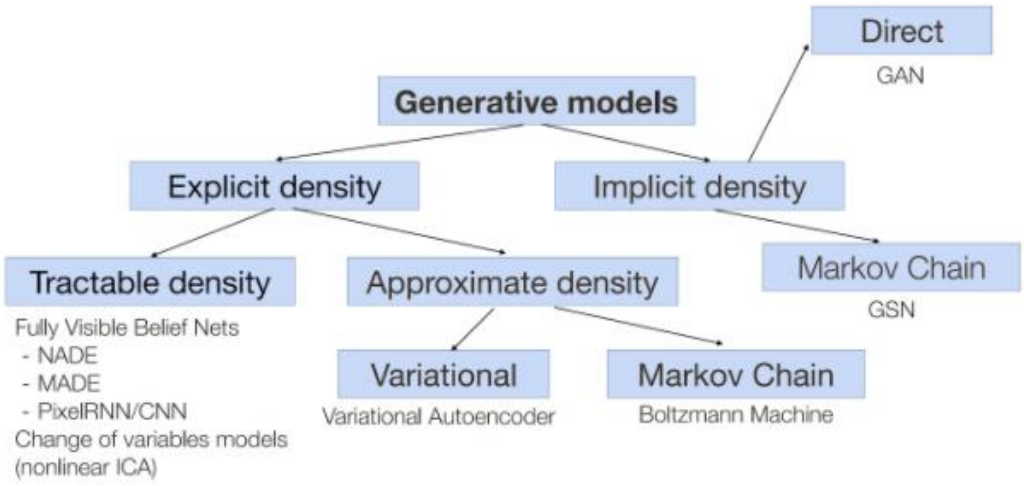

Figura 15: Rappresentazione di modelli generativi: una BM è basata su una funzione densità esplicita.

Altra peculiarità di una BM consiste nel soddisfacimento delle proprietà della cosiddetta "*Markov Chain*", ossia di un modello teorico probabilistico in grado di stimare una sequenza di eventi, in cui lo stato futuro di un sistema dipende solamente dallo stato presente, non da quelli passati.

Infine, concorre alla definizione di una BM il concetto di modello grafico. Un generico modello grafico consta di nodi ed eventualmente di direzioni associate al collegamento degli stessi. Si parla, dunque, di "*Directed Graphical Model*" ed "*Undirected Graphical Model*" significando che le due tipologie differiscono per la tipologia di legame fra i vari nodi (unidirezionale e bidirezionale). Una Boltzmann Machine (BM) è un modello probabilistico generativo strutturato come un grafo bidirezionale che soddisfa la proprietà di Markov. *BMs* apprendono la densità di probabilità da dati in input per generare nuovi campioni a partire

dalla stessa distribuzione. A BM è dotata di uno strato di input anche detto "*visible layer*" e di uno o più strati intermedi. Non è presente uno strato di output.

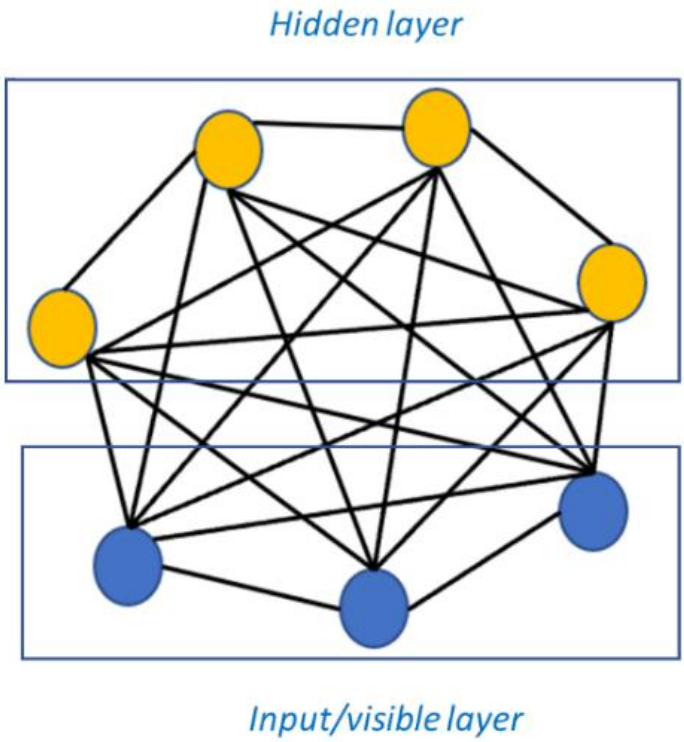

Figura 16: Tipica architettura di una Boltzmann Machine.

I neuroni della rete, sulla base dei dati usati per effettuare il processo di addestramento della rete, stabiliscono di essere attivi o meno tramite un processo stocastico. Una fondamentale differenza fra una BM ed altre popolari architetture di reti neurali consta nella interconnessione dei vari neuroni: in una BM il collegamento coinvolge non solo strati differenti ma anche neuroni appartenenti ad uno stesso strato. Questa considerazione comporta la generale impraticabilità del modello stesso a causa delle ingenti capacità computazionali richieste per il "*training*". D'altra parte, si suole fare affidamento ad una versione "*restricted*" della rete che prevede l'eliminazione delle connessioni interne ad uno stesso strato e ciò facilita il processo di addestramento del modello.

3.2 Processo di addestramento: algoritmo "*Contrastive Divergence*"

Le "*Restricted Boltzmann Machines*" sono solitamente addestrate usando l'algoritmo "*Contrastive Divergence*". In questa sezione, si procede a discutere a proposito delle peculiarità di detta procedura nonché del settaggio di importanti meta-parametri (*learning rate*, momento, valori iniziali dei pesi, numero di neuroni facenti parte dello strato

intermedio, dimensione di ciascun *mini-batch*) capaci di influire sull'efficacia dell'addestramento della rete.

Come si è accennato, una *RBM* è definita da una serie di binari, stocastici *pixels* (le cosiddette "*visible units*" poiché i loro stati sono osservati) e binari "*feature detectors*" corrispondenti alle unità "nascoste" in quanto non sono noti i relativi stati.

L'energia della configurazione (**v**, **h**) con **v** "*visible units*" ed **h** "*hidden units*" è data da:

$$E(\mathbf{v}, \mathbf{h}) = - -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

dove v_i, h_j sono gli stati binari della generica unità visibile *i* e della unità nascosta *j*, a_i, b_j sono i loro *biases* e w_{ij} è il peso relativo a ciascuna delle loro connessioni.

La rete assegna una probabilità ad ogni coppia di nodi (visibili e nascosti) mediante la seguente formula:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(v,h)}$$

in cui la quantità *Z* è ottenuta sommando tutte le possibili coppie di nodi visibili e nascosti come segue:

$$Z = \sum_{v,h} e^{-E(v,h)}.$$

Si sa, inoltre, che la probabilità generica di un nodo visibile è data dalla somma di tutti i possibili vettori di nodi nascosti:

$$p(\mathbf{v}) = \frac{1}{Z} \sum_h e^{-E(v,h)}.$$

La probabilità che la rete assegna ad una certa immagine di *training* può, dunque, essere innalzata attraverso una opportuna regolazione dei pesi e dei *biases* per ridurre l'energia delle altre immagini, specialmente di quelle caratterizzate da bassi valori di energia e che pertanto forniscono un notevole contributo alla funzione *Z*. Riportiamo ora la derivata del vettore *v* di nodi visibili rispetto ai pesi delle connessioni:

$$\frac{\partial \log p(\boldsymbol{v})}{\partial_{w_{ij}}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$$

dove le quantità entro i *brackets* sono i valori veri all'interno della distribuzione caratterizzante le tipologie di nodi.

Questa formula consente, infine, di aggiornare i pesi relativi alle connessioni dando luogo ad una semplice regola di *training* il cui soddisfacimento ad ogni "epoca" del processo di addestramento mira ad ottenere la convergenza, ossia la tendenza all'azzeramento fra i valori degli stati binari forniti in ingresso alla rete e quelli scaturiti a valle del "*training*" medesimo:

$$\Delta_{w_{ij}} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model})$$

dove ϵ rappresenta il *learning rate*.

Dal momento che non ci sono connessioni fra unità nascoste in una *RBM* è molto semplice ottenere un campione "*unbiased*" di $\langle v_i h_j \rangle_{data}$. Assegnata una immagine di training randomicamente, **v**, lo stato binario, h_j, di ciascuna unità nascosta, *j*, è settata al valore unitario con una probabilità:

$$p\left(h_j = 1 | \boldsymbol{v}\right) = \sigma(b_j + \sum_i v_i w_{ij})$$

dove $\sigma(x)$ rappresenta una funzione sigmoide.

Per le stesse ragioni di cui sopra, dato un vettore di "*hidden units*" ottenere un campione "*unbiased*" di uno stato di un nodo visibile risulta banale:

$$p\left(v_i = 1 | \boldsymbol{h}\right) = \sigma(a_i + \sum_j h_j w_{ij})$$

D'altra parte, estrarre tramite il modello una immagine del tipo $\langle v_i h_j \rangle_{model}$ risulta, in generale, oneroso poiché ciò significherebbe aggiornare tutti i nodi nascosti in parallelo sfruttando l'equazione relativa alla probabilità unitaria di ogni nodo nascosto unitamente ad un successivo aggiornamento delle unità visibili tramite equazione precedente. Questi processi di aggiornamento vengono condotti utilizzando il cosiddetto campionatore di Gibbs, per cui si può definire una differenziazione degli algoritmi "*Contrastive divergence*" proprio sulla base degli *step* necessari alla terminazione della procedura. Normalmente si assumerà il caso CD_1, ovvero una unica fascia di campioni di Gibbs. Questa considerazione ha prodotto l'ideazione di una procedura di addestramento più agevole ad opera di *Hinton* (2002):

- si settano gli stati delle unità visibili all'interno di un vettore di "*training*";
- si computano gli stati delle unità nascoste in parallelo attraverso la corrispondente equazione;
- una volta determinati gli stati binari delle "*hidden units*", viene attuato un processo di "ricostruzione" che consiste nel porre pari ad uno ciascun v_i con probabilità data dall'ultima equazione riportata.

Con tale nuova metodologia, la determinazione dei pesi delle connessioni è data da:

$$\Delta_{w_{ij}} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon})$$

Per quel che concerne la determinazione dei valori dei "*biases*" si adotta una versione semplificata di tale procedura basata sull'uso degli stati delle unità anziché delle coppie (**v**, **h**).

3.2.1 Aggiornamento degli stati nascosti

Supponendo di adottare CD_1 la probabilità di attivare una unità nascosta, *j*, è calcolata modellando la stessa come una funzione sigmoide σ:

$$p\left(h_j = 1\right) = \sigma(b_j + \sum_i v_i \, w_{ij})$$

e tale accensione avviene nella pratica se tale probabilità è superiore rispetto ad un numero randomico uniformemente distribuito fra 0 e 1.

Si raccomanda sempre di rendere binari gli stati piuttosto che utilizzare le loro probabilità. Infatti, qualora queste ultime vengano usate, ciascuna unità nascosta può comunicare un valore reale alle unità visibili durante la ricostruzione. Questo avvenimento potrebbe seriamente compromettere il collo di bottiglia nella trasmissione dell'informazione legato al fatto che una unità nascosta può in media trasmettere al più un bit.

È tuttavia importante sottolineare come per l'ultimo aggiornamento delle unità nascoste sia assolutamente superfluo l'utilizzo degli stati binari stocastici in luogo delle probabilità poiché nulla dipende da quale stato viene scelto, evitando, inoltre, campionamento di rumore inutile.

3.2.2 Aggiornamento degli stati visibili

Come in precedenza, la corretta modalità di aggiornamento degli stati visibili quando si adotta una "*reconstruction*" consiste nel portarli stocasticamente a 0 oppure 1 con probabilità:

$$p\left(v_i = 1\right) = \sigma(a_i + \sum_j h_j \, w_{ij})$$

Come si è detto nel paragrafo 3.2.1, comunemente si raccomanda l'uso delle probabilità in sostituzione dei valori binari campionati per scongiurare eventuale campionamento di rumore accellerando così la fase di addestramento.

3.2.3 Scelta della dimensione del mini-batch ed effetti sul "learning rate"

In generale, un processo di addestramento di una rete neurale comporta l'aggiornamento progressivo dei pesi così da ottimizzare la "*feature extraction*". Normalmente, anziché adottare algoritmi di "*gradient descent*" sulla totalità del Dataset di "*training*", si usa

suddividere lo stesso in più porzioni di una certa dimensione, detta "*batch-size*". Questo espediente implica una velocizzazione delle procedure di addestramento su GPU (*Graphical Processing Units*).

Per evitare di dover cambiare il "*learning rate*" quando la dimensione del mini-batch viene modificata, si può considerare di dividere il gradiente totale calcolato su un singolo mini-batch per la dimensione dello stesso, cosicché quando si parla di "*learning rates*" si assumerà che questi scaturiscano dalla media fra il prodotto del valore del gradiente riferito ad un particolare mini-batch e il mini-batch medesimo, non del gradiente totale per il mini-batch.

Inoltre, si sconsiglia di adottare dimensioni troppo elevate dei mini-batches in vista dell'utilizzo degli algoritmi "*stochastic gradient descent*", dal momento che l'aumento di un fattore N delle stesse permette una stima del gradiente maggiormente affidabile ma non determina un incremento corrispondente del "*learning rate*", dunque l'effetto sortito si esprime in un aggiornamento percentualmente inferiore dei pesi della rete.

3.2.4 Divisione del "*training set*" in mini-batches

Per Datasets che contengono un certo numero di classi equiprobabili, la dimensione ideale del mini-batch è spesso uguale al numero di classi ed ogni mini-batch dovrebbe contenere un esempio di ciascuna classe per ridurre l'errore di campionamento nel momento in cui si procede alla stima del gradiente per l'intero "*training set*" da un singolo mini-batch.

Per altri Datasets, si può pensare di randomizzare dapprima l'ordine degli esempi per l'addestramento e poi utilizzare mini-batches di dimensione pari a 10 circa.

3.2.5 Monitoraggio del processo di addestramento

Quando ha inizio il training di una "*neural network*", indipendentemente dal fatto che sia una RBM, un noto parametro è dato dall'errore quadratico fra i dati in ingresso e quelli "ricostruiti" dalla rete. Solitamente, nelle prime fasi del processo tende a decadere rapidamente per poi adagiarsi a valori via via più stabili successivamente. A causa della presenza di rumore nella stima del gradiente, "*reconstruction error*" su singoli mini-batches fluttuerà gentilmente dopo una forte iniziale discesa.

Per quanto conveniente e semplice da calcolare, "*reconstruction error*" è una indicazione parziale del progresso della procedura di addestramento, in quanto non rappresenta la

funzione che CD_n approssima specialmente allorquando $n \gg 1$ e sistematicamente questo concetto confonde due quantità:

- la differenza fra la distribuzione di probabilità del "*training set*" e la distribuzione all'equilibrio della RBM;
- il tasso di mescolamento dei campioni di Gibbs appartenenti alla catena di Markov.

Infatti, se il "*mixing rate*" è molto basso "*reconstruction error*" sarà esiguo anche quando le distribuzioni dei dati e modelli sono molto diverse. Dunque, per conseguenza, decrescite del "*reconstruction error*" non implicano necessariamente un miglioramento dei modelli inferenziali e lo stesso si può dire a proposito degli incrementi associati al "*reconstruction error*", i quali potrebbero derivare dalla presenza di un ingente "*mixing rate*". In ogni caso, incrementi considerevoli costituiscono un indizio non favorevole all'ottenimento di un modello ottimizzato a meno che questi non divengano temporanei per causa di netti cambiamenti nel "*learning rate*", "*density-sparsity*" ed altri meta-parametri.

3.2.6 Monitoraggio dell'Overfitting

Nel momento in cui si attua il processo di apprendimento di un modello generativo, occorre tenere conto della probabilità che il modello inferenziale assegna ad un "*datapoint*". Quando si verifica che questa probabilità comincia a decrescere per un Dataset di validazione, è necessario arrestare il "*training*". Purtroppo, per l'addestramento di grandi RBMs è molto difficoltoso calcolare questa probabilità poiché viene richiesta la funzione Z (*partition function*). Tuttavia, un ulteriore modalità di monitoraggio consiste nella comparazione delle energie legate al Dataset di training e a quello di validazione del modello. L'energia di un determinato vettore di dati dipende linearmente dal numero di neuroni dello strato intermedio della rete:

$$e^{-F(\boldsymbol{v})} = \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}, \text{ con}$$

$$F(\boldsymbol{v}) = -\sum_i v_i a_i - \sum_j p_j x_j + \sum_j (p_j log p_j + \big(1-p_j\big)\log\big(1-p_j\big)),$$

con $x_j = b_j + \sum_i v_i w_{ij}$ il numero totale delle unità nascoste j e $p_j = \sigma\big(x_j\big)$ è la probabilità che $h_j = 1$ assegnato il vettore $\boldsymbol{v}$.

Quando il modello non è affetto da *overfitting*, l'energia media della parte "*training data*" e quella della parte "*validation data*" dovrebbero equivalersi. Se il modello dovesse subire *overfitting*, l'energia media del Dataset di validazione aumenterà rispetto a quella del Dataset di *training* e la differenza risultante è rappresentativa della quantità di *overfitting* della rete.

3.2.7 Learning rate

Questo parametro diviene fondamentale per assicurare un corretto esito dell'algoritmo "*Contrastive Divergenc*e" giacché un valore molto elevato in genere provoca una drastica riduzione del "*reconstruction error*" e di conseguenza i pesi relativi alle connessioni degli strati visibile e nascosto esploderebbero.
Qualora il "*learning rate*" fosse ridotto durante l'apprendimento, "*reconstruction error*" decadrebbe in modo significativo. Ciò non necessariamente rappresenta una positività. Il fenomeno si deve, in parte, ad una riduzione della soglia di rumore nella procedura stocastica di "*updating*" dei pesi e spesso è seguito da una riduzione della capacità di apprendimento nel lungo periodo. Verso la fine dell'apprendimento, in ogni caso, è pagante diminuire il *learning rate*.

3.2.8 Come impostare il learning rate per i pesi e i "biases"

Una regola a carattere generale ideata nell'ambito del settaggio del "*learning rate*" consiste nel fare riferimento sia all'istogramma dei pesi aggiornati che a quello dei pesi. Gli aggiornamenti progressivi dovrebbero ammontare a circa 10^{-3} volte i pesi stessi (e comunque entro un ordine di grandezza). Gli stessi sarebbero, in ogni caso, da particolarizzare in considerazione della tipologia di neuroni adottata dalla rete: unità con elevato *fan-in* richiederebbero aggiornamenti lievi poiché in tal caso piccoli scostamenti dei pesi nella stessa direzione potrebbero alterare il segno del gradiente. Al contrario, è possibile applicare ingenti variazioni ai "*biases*" con maggiore facilità.

3.2.9 Valori iniziali dei pesi e dei "biases"

Tipicamente i pesi sono inizializzati a valori randomici scelti mediante l'uso di una distribuzione Gaussiana a media nulla con una deviazione standard pari a circa 0.01. Adottare valori randomici più elevati potrebbe accellerare l'apprendimento iniziale, ma d'altra parte rischierebbe di produrre un modello leggermente meno ottimizzato.
È importante assicurare, in questo senso, che i valori iniziali dei pesi non consentano alle unità visibili di condurre le probabilità dei neuroni "nascosti" molto prossime allo zero oppure uno poiché ciò rallenterebbe notevolmente la fase di apprendimento. Se le statistiche

utilizzate per il "*learning*" fossero stocastiche, i valori iniziali dei pesi sarebbero da impostare a zero indistintamente dal momento che il campionamento di rumore renderebbe le unità nascoste differenti le une dalle altre benché caratterizzate da connessioni neuronali identiche.

Usualmente, è conveniente inizializzare il "*bias*" di una unità visibile i a $\log[\frac{p_i}{1-p_i}]$ ove p_i è quella porzione di vettore di "*training*" caratterizzata dalle unità i attivate. Se questa operazione non sarà compiuta, la fase iniziale dell'apprendimento sfrutterà le unità neuronali dello strato intermedio per rendere le unità i accese con una probabilità pari approssimativamente pari a p_i.

3.2.10 Weight-decay

Nelle RBMs il decadimento dei pesi consiste nell'aggiungere un ulteriore termine al normale calcolo del gradiente. Questo termine è rappresentato dalla derivata di una funzione, detta "*penalty function*". Per garantirne una adeguata efficacia, è opportuno che venga moltiplicata per il "*learning rate*" onde evitare che modifiche di quest'ultimo parametro producano cambiamenti nella funzione ottimizzata piuttosto che alterare la procedura di ottimizzazione.

La tecnica del decadimento dei pesi viene introdotta per quattro sostanziali ragioni:

- riduzione di "*overfitting*" rispetto al "*training Dataset*": il termine "penalità" viene applicato su ogni mini-batch, l'approccio bayesiano dividerebbe il cosiddetto "*weight-cost*" (un coefficiente della funzione penalità) per la dimensione del training set, "*weight-decay*" esprimerebbe l'effetto di una distribuzione gaussiana la cui varianza non dipende dal training set; nella pratica, anziché effettuare la divisione, si usano maggiori "*weight-costs*" per training set più piccoli;
- capacità di rendere più circoscritto il campo ricettivo dei neuroni dello strato intermedio facendo divenire irrilevanti i pesi inutilizzati;
- esigenza di scrollare le unità nascoste associate a pesi molto grandi e che permangono nettamente in uno stato di accensione o disattivazione (queste unità potranno essere nuovamente efficaci attraverso il concetto di "*sparsity target*" come vedremo in seguito);
- miglioramento del "*mixing rate*" dei campioni di Gibbs. Infatti, con pesi piccoli la catena di Markov mixa più velocemente, nel caso estremo (valori azzerati dei pesi) tale catena di Markov raggiunge in un unico step la distribuzione stazionaria.

La tecnica "*weight-decay*" può venire adottata o mediante la funzione "penalità" (con annesso coefficiente "*weight-cost*") oppure attraverso l'utilizzo della derivata della somma dei valori assoluti dei pesi (L1). Con la funzione L1 si sortisce l'effetto di azzerare molti pesi e contemporaneamente di rendere forte la crescita di pochi altri. Questa peculiarità agevola l'interpretazione dei pesi e comporta un deciso restringimento del campo ricettivo.

3.2.11 Sparse hidden activities

Le unità neuronali nascoste che sono raramente attivate sono maggiormente semplici da interpretare rispetto a quelle attive per circa metà del tempo.
È possibile prestabilire una certa occasionalità nelle attività delle unità dello strato intermedio specificando un cosiddetto "*sparsity target*" quale la probabilità desiderata di attivazione $p \ll 1$. Questo valore non è da confondere con la effettiva probabilità di attivazione q. q viene stimata attraverso una media esponenzialmente decadente della probabilità media che una unità è attiva in ogni mini-batch:

$$q_{new} = \lambda q_{old} + (1 - \lambda) q_{current}$$

dove $q_{current}$ è la probabilità media di attivazione dell'unità nascosta sul corrente mini-batch.
Per effettuare una misura della "*sparse penalty*" si suole usare una "*loss function*" del tipo "*cross-entropy*" per cui si ha:

$$Sparsity\ penalty \propto -p\ log\ q - (1 - p)\log(1 - q).$$

La derivata di questa funzione rispetto all'input totale vale $q - p$. Tale derivata scalata di un parametro chiamato "*sparsity-cost*" viene sfruttata per sia il "*bias*" che per i pesi dell'unità neuronale "nascosta". È importante applicare la stessa derivata ad entrambi in quanto se la derivata è solo applicata al "*bias*" questo tipicamente continuerà a divenire sempre più negativo assicurando la permanente attivazione dell'unità, ma i pesi continueranno ad assumere valori positivi crescenti per renderla più utile ed interpretabile.

3.2.12 Numero di unità nascoste

In generale quando si procede all'addestramento di modelli generativi con una enorme dimensionalità di dati, il numero di bits specificante un "*data vector*" determina i vincoli imposti dal caso di allenamento della rete sui parametri del modello. Questi possono essere

di molti ordini di grandezza superiori rispetto al numero dei bits richiesti per specificare una etichetta. Per informare in termini generici, è ragionevole ipotizzare un *fitting* di circa un milione di parametri per un training set di dimensione diecimila per il quale ciascuna immagine è rappresentata con un migliaio di pixels. In tal caso, ci si orienta nell'ordine di un migliaio di "*hidden units*" globalmente connesse. Qualora tali unità fossero localmente connesse o usassero "*weight-sharing*", ne servirebbero in quantità ancora di molto maggiori.

4. Quantum Deep Learning

Negli ultimi anni, come è stato ampiamente sviscerato, il *Deep Learning* ha avuto un profondo impatto sul Machine Learning ed Intelligenza Artificiale.
Al tempo stesso, ed è la motivazione per cui si presenta questa sezione, gli algoritmi quantistici hanno dimostrato maggiori efficienza e *performances* in relazione ad una determinata tipologia di problemi classicamente quasi del tutto intrattabili.
Scopo di questo capitolo è sottolineare come il *Quantum Computing* non solo riduca il tempo per il training di una "*Restricted Boltzmann Machine*", ma fornisca anche un *framework* più completo e ricco per il Deep Learning in grado di condurre a significativi miglioramenti nell' ottimizzazione delle funzioni oggetto.

4.1 Introduzione

Il *Deep Learning* è una recente tecnica del *Machine Learning* basata sul presupposto che, al fine di affrontare e risolvere sofisticati problemi di *Artificial Intelligence* come il riconoscimento visuale, possa essere necessario adottare un modello di apprendimento composto da una serie di "*layers*" di astrazione del vettore di dati in input alla "*deep neural network*" cosicché le operazioni di "feature extraction" vengano ripartite fra gli strati della rete secondo criteri stabiliti dai progettisti.
Le Boltzmann Machines costituiscono un sottoinsieme delle reti neurali "*deep*" che formalmente sono formate da nodi ricorrenti con relative connessioni non dotate di alcuna direzionalità e così rappresentano un modello generativo per i dati. Da un punto di vista fisico, tali reti modellano il Dataset di addestramento come un *modello di Ising* all'equilibrio termico. I nodi della rete sono atti ad effettuare una codifica delle *features* (caratteristiche) delle immagini per le quali si richiede il riconoscimento, le connessioni corrispondenti, dall' altro lato, sono adibiti alla rappresentazione delle dipendenze statistiche delle "*features*". Il set di nodi che codificano i dati osservati sono indicati con il nome "*visible units*" $\boldsymbol{v}$, mentre i nodi facenti parte dello spazio latente o nascosto sono noti come "*hidden units*" $\boldsymbol{h}$. Si supporrà nelle successive discussioni che entrambe le tipologie di nodi assumano esclusivamente valori binari.
Una BM fornisce la probabilità di una assegnata configurazione di nodi visibili e nascosti attraverso la *distribuzione di Gibbs*:

$$P\ (v,h) = \frac{e^{-E(v,h)}}{Z},$$

dove Z è un fattore normalizzato noto come "*partition function*" e l'energia $E\ (v,h)$ di una data configurazione (v,h) di unità visibili e nascoste è data da:

$$E\ (v,h) = -\sum_i v_i\, b_i\ - \sum_j h_j\, d_j - \sum_{i,j} w_{ij}^{v,h}\, v_i h_j - \sum_{i,j} w_{ij}^{v}\, v_i v_j - \sum_{i,j} w_{ij}^{h}\, h_i h_j,$$

in cui b, d sono vettori chiamati "*biases*" in grado di attribuire una carenza energetica alle unità binarie di valore unitario mentre $w_{ij}^{v,h}$, w_{ij}^{v}, w_{ij}^{h} sono i pesi con cui regolare le carenze energetiche se entrambe le unità connesse ("*visible*, *hidden*") acquisiscono valore unitario. Si indicherà, poi, con n_v ed n_h i numeri rispettivamente dei nodi visibili e nascosti.

Assegnato un vettore di dati in input alla rete e costitutivi del training set disponibile, il termine "*learning*" equivale ad attuare un processo tramite il quale si possano modificare le intensità delle interazioni nel grafo della rete in maniera tale da massimizzare la probabilità che la Boltzmann Machine riproduca le osservazioni fornite. Come conseguenza, il processo di addestramento utilizza algoritmi di discesa del gradiente per individuare pesi, "*biases*" che ottimizzano la cosiddetta "*maximum-likelihood objective function*":

$$O_{ML}\ = \frac{1}{N_{train}} \sum_{v\ \epsilon\ xtrain} \log\ (\sum_{h=1}^{n_h} P(v,h)) - \frac{\lambda}{2} w^T w,$$

dove N_{train} è la dimensione del training set, x_{train} è il set di vettori di dati per l'addestramento e λ è una costante per mitigare l'Overfitting. La derivata della funzione O_{ML} rispetto ai pesi vale:

$$\frac{\partial O_{ML}}{\partial w_{i,j}} = \left\langle v_i h_j \right\rangle_{data} - \left\langle v_i h_j \right\rangle_{model} - \lambda w_{i,j},$$

in cui le "*brackets*" sono indicativi dei valori attesi sui dati e il modello per la BM.

Il calcolo dei gradienti direttamente dalla equazione esplicitante $P\ (v,h)$ e dalla derivata della funzione "*objective maximum likelihood*" è esponenzialmente difficoltoso dati n_v ed n_h, dunque gli approcci classici al problema ricorrono all' algoritmo "*Contrastive Divergence*" e sue varianti. Purtroppo, questa tipologia di algoritmi non conduce ad una soluzione esatta del problema, bensì a soluzioni sub-ottime e pertanto è da escludere un impiego per il training di una "*full Boltzmann Machine*". Si introducono nei paragrafi successivi due algoritmi di calcolo quantistici in grado di velocizzare la fase di addestramento e fornire così un miglior *framework* per *deep learning*.

4.2 GEQS Algoritmo

L'algoritmo GEQS (*Gradient Estimation via Quantum Sampling*) e, come vedremo l'algoritmo GEQAE (*Quantum Amplitude Estimation*), si basano sulla preparazione di un pacchetto di stati dotati di una distribuzione di probabilità quale quella di Gibbs, a valle della quale vengono prodotti campioni per il calcolo dei valori attesi di cui si rimanda alla formula all' espressione della derivata della O_{ML}.

Normalmente, per la fase "*state preparation*" vengono talvolta considerati altre tipologie di algoritmi, le quali, però, non sempre evidenziano una certa superiorità quantistica in questo settore in quanto fanno uso spesso di distribuzioni di probabilità uniformi nel generare i campioni in uscita.

D'altra parte, invece, gli algoritmi oggetto di discussione ricorrono all'utilizzo di distribuzioni di probabilità non uniformi, e tale scelta viene giustificata dal fatto che aprioristicamente è già noto che dai valori dei pesi, "*biases*" alcune configurazioni sono meno probabili di altre. Gli algoritmi ottengono questa distribuzione usando la cosiddetta approssimazione "*mean-field*" (*MF*).

L'approssimazione *MF*, $Q(v,h)$, è definita come una distribuzione prodotto in grado di minimizzare la cosiddetta "*Kullback-Leibler divergence*". Il fatto che sia una distribuzione prodotto significa che può essere efficientemente calcolata e può essere adoperata per stimare in modo verosimile la "*partition function*":

$$Z_Q = \sum_{v,h} Q(v,h)\log(\frac{e^{-E(v,h)}}{Q(v,h)}).$$

In questo caso $Z_Q \ll Z$ e l'uguaglianza viene raggiunta se e solo se $KL(Q\|P) = 0$.

Si assuma che una costante k sia conosciuta in modo che si verifichi che:

$$P(v,h) \leq \frac{e^{-E(v,h)}}{Z_Q} \leq kQ(v,h)$$

e si definisca la seguente probabilità "normalizzata" della configurazione come

$$P(v,h) := \frac{e^{-E(v,h)}}{kZ_Q Q(v,h)}.$$

Si osservi che:

$$Q(v,h)P(v,h) \propto P(v,h),$$

che significa che se lo stato:

$$\sum_{v,h} \sqrt{Q(v,h)}\ |v\rangle\ |h\rangle$$

viene preparato e ciascuna delle ampiezze è moltiplicata per $\sqrt{P(v,h)}$ poi il risultato sarà proporzionale allo stato desiderato.

Il processo soprastante può essere reso operativo aggiungendo un registro quantico per calcolare $P(v,h)$ ed usando la quantum superposizione per preparare lo stato:

$$\sum_{v,h} \sqrt{Q(v,h)}\ |v\rangle\,|h\rangle\,|P(v,h)\rangle(\sqrt{1-P(v,h)})\,|0\rangle + \sqrt{P(v,h)})\,|1\rangle).$$

Lo stato di *Gibbs* target viene ottenuto se il qubit posto maggiormente a destra risulta essere 1. Preparare lo stato precedente è abbastanza conveniente poiché i valori $e^{-E(v,h)}$ e $Q(v,h)$ possono essere calcolati in un tempo polinomiale rispetto alle unità visibili e nascoste della rete. La probabilità di successo di preparare lo stato in questa maniera vale:

$$P_{success} = \frac{Z}{kZ_Q} \geq \frac{1}{k}.$$

In termini pratici, l'algoritmo utilizza la "*quantum amplitude amplification*" per incrementare quadraticamente le probabilità di successo qualora il valore della stessa fosse esiguo.

La complessità dell'algoritmo è determinata dal numero di operazione quantiche necessarie nel calcolo del gradiente. Poiché la valutazione dell'energia richiede un numero di operazioni che scala linearmente con il numero totale di connessioni nel modello il costo combinato relativo alla stima del gradiente risulta:

$$O^{\sim}(N_{train}E(\sqrt{k} + \max_{x \in xtrain} \sqrt{k_x})),$$

dove k_x è il valore di k corrispondente al caso in cui le unità visibili siano x. Il costo della stima $Q(v,h)$ e Z_{MF} vale $O^{\sim}(E)$ e dunque non contribuisce asintoticamente al costo. Al contrario, il numero di operazioni richieste per stimare classicamente il gradiente usando l'ottimizzazione "*layer-by-layer*" scala di un fattore

$$O^{\sim}(N_{train}lE),$$

dove l è il numero dei *layers* nella dRBM ed *E* il numero delle connessioni nella BM. Assumendo che k è una costante, l'approccio quantico fornisce un vantaggio asintotico per il training delle reti *deep*.

Il numero dei qubits richiesti è minimale rispetto agli esistenti algoritmi di *Quantum Machine Learning*. Ciò avviene poiché il Dataset di training non necessita di essere memorizzato in un *Quantum Database*, che richiederebbe $O^{\sim}(N_{train})$ qubits logici. Piuttosto, qualora $P(v,h)$ sia calcolato con $\log(1/\varepsilon)$ bits e vi si possa fare accesso come un Database Oracle, solamente un numero pari a $O(n_v + n_h + \log(1/\varepsilon))$ qubits sarebbero sufficienti per l'algoritmo GEQS. Per giunta, l'esatto valore di k non deve essere tassativamente noto. Se, infatti, dovesse

essere scelto un valore di k non soddisfacente per tutte le configurazioni, l'algoritmo sarebbe capace di approssimare il gradiente a patto di collocare i valori di $P(v, h)$ fra 0 e 1. Dunque, l'algoritmo può sempre essere reso efficiente al prezzo di introdurre errori nella risultante distribuzione di probabilità, mantenendo k fissato come viene incrementata la dimensione della BM.

4.3 GEQAE Algoritmo

Nuove modalità di training, come quelle legate all'algoritmo GEQAE (*Gradient Estimation via Quantum Amplitude Estimation*), sono sfruttabili qualora il Dataset di training sia immagazzinato in un DB quantistico Oracle, la qual cosa permette l'accesso ai dati in superposizione piuttosto che in maniera sequenziale. L'idea alla base di tale algoritmo è utilizzare il fondamentale principio quantistico della "*data superposition*" attraverso la stima dell'ampiezza degli stati, il che comporta una sostanziale riduzione della varianza nella stima del gradiente rispetto all'algoritmo GEQS. Come conseguenza, GEQAE produce un netto miglioramento della stima del gradiente per *training set* vasti.

Il *Quantum Oracle Database*, in questo contesto, può essere immaginato come una subroutine quantistica in grado di generare i dati per l'apprendimento della macchina (alla stregua di una *pre-trained quantum Boltzmann Machine*). Giacché il Dataset di training deve essere memorizzato in un *Quantum Computer*, GEQAE richiede un numero superiore di qubits rispetto all'algoritmo GEQS; tuttavia, questa negatività viene compensata dal fatto che la "*quantum superposition*" consente l'addestramento sull'intero bacino di dati del Dataset di training in unico passo rispetto ad una modalità di *learning* basata su accessi sequenziali. Ciò garantisce una stima accurata del gradiente con accesso al Dataset di training al più in numero di volte pari a $O(\sqrt{N_{train}})$.

Sia U_O un *quantum Oracle* che, per ogni indice i, commette la seguente operazione:

$$U_O \left|i\right\rangle \left|y\right\rangle := \left|v\right\rangle \left|v \oplus x_i\right\rangle,$$

dove x_i è il vettore di training. Tale U_O può essere usato per preparare le unità visibili nello stato dell' $i - esimo$ vettore di training. Una successiva *query* al DB Oracle è sufficiente per preparare una superposizione uniforme su tutti i vettori di dati adibiti all'addestramento, la qual cosa, ripetendo poi la procedura di "*state preparation*" comune anche all'algoritmo GEQS, viene poi convertita nello stato:

$$\frac{1}{\sqrt{N_{train}}}\sum_{i,h}\sqrt{Q(X_i,h)}\ |i\rangle\ |x_i\rangle\ |h\rangle(\sqrt{1-P(x_i,h)}\ |0\rangle+\sqrt{P(x_i,h)}\ |1\rangle.$$

GEQAE calcola i valori attesi come $\langle v_i h_j\rangle$ sui dati e il modello stimando:

- $P(1)$, la probabilità che il qubit maggiormente a destra sia unitaria;
- $P(11)$, la probabilità di avere $v_i = h_j = 1$ e $P(1) = 1$.

Segue che:

$$\langle v_i h_j\rangle = \frac{P(11)}{P(1)}.$$

Queste due probabilità possono essere stimate campionando, ma un metodo più efficiente consiste nell'uso della stima dell'ampiezza (un algoritmo quantistico che utilizza la stima della fase dell'algoritmo di Groover) per convogliare queste probabilità in una stringa di qubits. Se si stabilisce di voler scalare l'errore di campionamento ad un fattore $1/\sqrt{N_{train}}$, la complessità della *query* del GEQAE risulta:

$$O^{\sim}(\sqrt{N_{train}}E(k+\max_x k_x)).$$

Ciascun calcolo di energia richiede $O^{\sim}(E)$ operazioni aritmetiche, pertanto la formula precedente permette di constatare che il numero di operazioni *non-query* scala come:

$$O^{\sim}(\sqrt{N_{train}}E^2(k+\max_x k_x)).$$

Se la probabilità di successo è nota entro un fattore costante, la tecnica dell'amplificazione dell'ampiezza può venir adoperata per incrementarla. L'originale probabilità di successo viene poi calcolata dalla probabilità amplificata. Ciò riduce la complessità della *query* dell'algoritmo GEQAE alla quantità:

$$O^{\sim}(\sqrt{N_{train}}E(\sqrt{k}+\max_x \sqrt{k_x})).$$

GEQAE è perciò preferibile al GEQS se $\sqrt{N_{train}} \gg E$.

	Operations	Qubits	Exact
ML	$\tilde{O}(N_{\text{train}}2^{n_v+n_h})$	0	Y
CD–k	$\tilde{O}(N_{\text{train}}\ell E k)$	0	N
GEQS	$\tilde{O}(N_{\text{train}}E(\sqrt{\kappa}+\max_x\sqrt{\kappa_x}))$	$O(n_h+n_v+\log(1/\mathcal{E}))$	Y
GEQAE	$\tilde{O}(\sqrt{N_{\text{train}}}E^2(\sqrt{\kappa}+\max_x\sqrt{\kappa_x}))$	$O(n_h+n_v+\log(1/\mathcal{E}))$	Y
GEQAE (QRAM)	$\tilde{O}(\sqrt{N_{\text{train}}}E^2(\sqrt{\kappa}+\max_x\sqrt{\kappa_x}))$	$O(N_{\text{train}}+n_h+n_v+\log(1/\mathcal{E}))$	Y

Figura 17:Panoramica generale dei parametri impiegati per ciascun approccio considerato; un algoritmo si dice esatto quando il campionamento rappresenta l'unica possibile causa di errore. GEQS e GEQAE non sono esatti se non è rispettata la condizione (5).

4.4 Parallelizzabilità degli algoritmi di training

L' algoritmo "*Contrastive Divergence*" è parallelizzabile, vale a dire che quasi la totalità delle sue componenti possono essere distribuite su diversi nodi di una stessa GPU in parallelo. Tuttavia, il processo di addestramento di ciascun "*layer*" che si avvale di $k\ step$ di campioni non è altrettanto facilmente parallelizzabile.

D'altra parte, la natura degli algoritmi GEQS e GEQAE consente di incrementare il parallelismo nel contesto del *training* di una dRBM. Per motivare tale affermazione, si noti che l'energia vale la somma delle energie di ciascuno strato, che viene valutata come $\log(M) = O(\max(n_v, n_h)\log(\max(n_v, n_h)))$ con aggiunta del fattore $O(\log(l))$ relativo all'impiego del numero l di *layers*. La profondità in termini di passaggi dell'algoritmo GEQS vale:

$$O(\log([k + \max_x k_x]MlN_{train})).$$

Poiché ciascuna delle derivate di output con l'algoritmo GEQAE può essere determinata indipendentemente, la profondità del GEQAE vale:

$$O(\sqrt{N_{train}[k + \max_x k_x]}\log(Ml)).$$

Tale profondità può essere ridotta, pagando in termini di aumento delle dimensioni circuitali; in alternativa, dividendo il training set in mini-batches e mediando le derivate risultanti.

Training attraverso l'uso dell'algoritmo $k\ step$ "*Contrastive Divergence*" richiede una profondità:

$$O(kl^2\log(MN_{train}))\,,$$

dove il fattore $O(l^2)$ è legato alla natura feed-forward dell'algoritmo CD, la quale non caratterizza invece gli algoritmi quantistici illustrati in precedenza.

4.5 Risultati numerici conseguiti

Riguardo al comportamento degli algoritmi presentati, è lecito esaurire alcune questioni:

- Quali sono i tipici valori di k?
- Come differiscono i modelli trainati con CD_1 da quelli trainati con GEQS e GEQAE?
- I modelli "*full Boltzmann Machines*" sono migliori rispetto a quelli dRBM?

Per poter affrontare questi quesiti, $P(v, h)$ e O_{ML} necessitano di essere calcolate classicamente, richiedendo uno sforzo temporale che cresce esponenzialmente rispetto alla

quantità $\max\{n_v, n_h\}$. Per ragioni riconducibili alle limitazioni di calcolo, si è stabilito di addestrare una dRBM composta da $l \in \{2,3\}$ strati, $n_h \in \{2, \ldots, 8\}$ unità nascoste, ed $n_v \in \{4, \ldots, 12\}$ unità visibili. Nell'analisi è stato fatto uso di Dataset di training sintetici i cui elementi, come si illustra di seguito, assumono determinati valori sulla base di quattro condizioni preordinate:

$$[x_1]_j = 1 \; se \; j \leq {}^{n_v}/_2 \; altrimenti \; 0$$

$$[x_2]_j = j \; mod \; 2,$$

e le altre due condizioni valide per le negazioni dei bit. Al fine di ampliare la dimensionalità del training set, è stato aggiunto del rumore Bernoulliano $N \in [0, 0.5]$ a ciascuno dei bits nella stringa di bit ottenuta per l'apprendimento della dRBM. Nella fattispecie, presi ciascuno dei quattro pattern di cui alle condizioni precedenti, i bits sono stati condotti a probabilità N.

Sono stati impiegati 10000 esempi per l'apprendimento in ognuno degli esperimenti numerici; ogni vettore contiene 4,...,12 "*binary features*". Lo scopo di questo processo si ricorda essere l'inferimento di un modello generativo.

La figura riportata in basso evidenzia come raddoppiando il numero di unità visibili della rete non si consegua un sostanziale incremento di k per questo Dataset di training (con $N = 0$), benché le dimensioni dello spazio di Hilbert cambino di un fattore 2^6. Questo risultato dimostra che in primo luogo k dipende dalla qualità dell'approssimazione "*Mean-field*" piuttosto che dai parametri n_v ed n_h. Inoltre, $k \approx 1000$ tipicamente risulta in una adeguata approssimazione dello stato di Gibbs ideale.

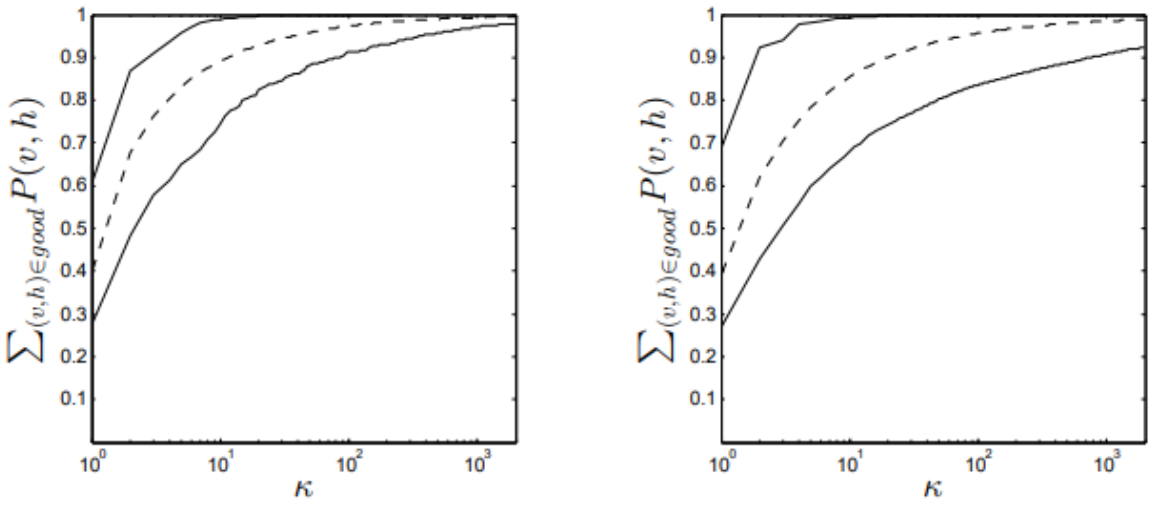

Figura 18: RBM addestrata con 8 unità nascoste, 6 unità visibili (sinistra) e 12 unità visibili (destra). Le linee tratteggiate costituiscono i valori medi, le linee continue sono ottenute con una confidenza del 95%.

Inoltre, è stato esaminato lo *scaling* di k per una RBM non addestrata:

$$k_{est} = \sum_{v,h} {P^2(v,h)}/{Q(v,h)}.$$

È stato appurato, come si mostra in basso, che per piccole RBMs randomiche (non trainate), $k-1 \in O(\sigma^2(w_{ij})E)$ per $\sigma^2(w_{ij})E \ll 1$.

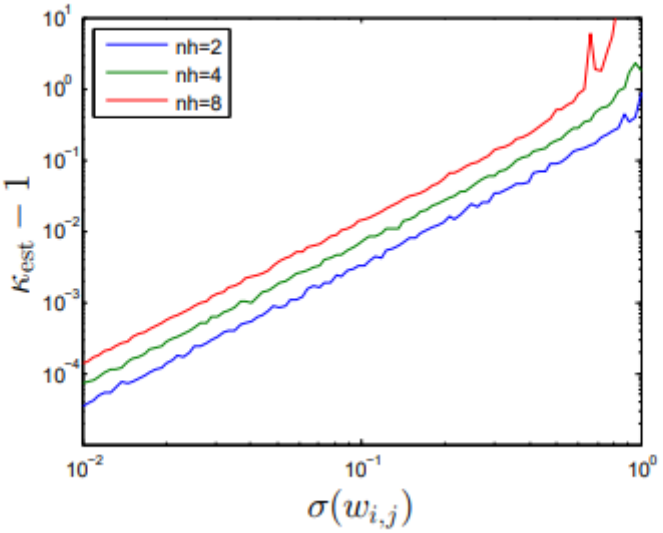

Figura 19: Andamento di k in funzione della deviazione standard dei pesi in una sintetica RBM con 4 unità visibili e pesi Gaussiani con una media pari a 0 e varianza indicata in ascissa.

Quest'ultimo dato sperimentale conduce ad un importante questione: la determinazione dei pesi di una Boltzmann Machine. Come emerge dai grafici della figura 21, RBMs di grandi dimensioni addestrate usando l'algoritmo "*Contrastive Divergence*" hanno pesi che tendono a cadere rapidamente nel momento in cui viene incrementato n_h. Per $N = 0$, lo *scaling* empirico vale $\sigma^2 \epsilon\, O(E^{-1})$ che suggerisce che $k-1$ non divergerà all'aumentare di n_h. Anche se $N = 0.2$ riduce σ^2 considerevolmente, anche lo *scaling* è diminuito.

Questi risultati in unione a quelli presentati nella figura 20 suggeriscono che k diviene maggiormente gestibile in termini di valori ottimizzati per reti strutturalmente più grandi e complesse.

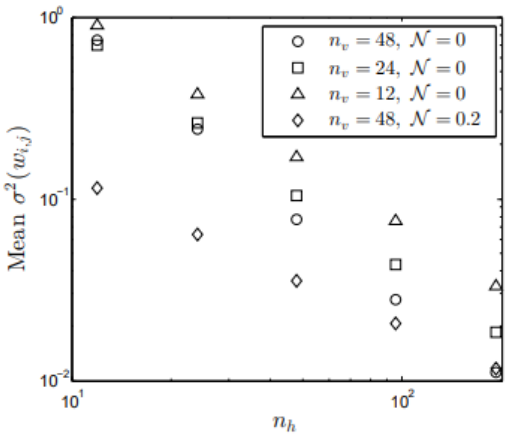

Figura 20: Deviazione standard dei pesi per grandi RBMs trainate sulla base della formula 22 usando CD con λ = 0.01.

Sono stati, inoltre, constatati i benefici degli algoritmi GEQS e GEQAE attraverso la comparazione dei valori medi della quantità O_{ML} calcolati con l'approccio "*Contrastive Divergence*" e con l'approccio quantistico per dRBM. La differenza fra i parametri ottimali è significativa per RBM di dimensioni ridotte, mentre nel caso di reti "*deep*" può essere dell'ordine del 10%. I dati riportati nella tabella seguente sottolineano come le tecniche di training basate su *Machine Learning* conducano a notevoli miglioramenti nella qualità dei modelli risultanti. Si osserva anche che l'algoritmo "*Contrastive Divergence*" è in grado di sovraperformare gli algoritmi "*Gradient Descent*" basati su ML solamente in specifiche situazioni "*constrained*". Ciò avviene poiché la natura stocastica del "Contrastive Divergence" lo rende meno sensibile ai minimi locali.

La capacità di *modeling* di una "*full BM*" è in grado in maniera netta di sovraperformare rispetto ad una dRBM in termini della qualità della "*objective function*". Infatti, è stato visto che una "*full BM*" dotata di 6 unità visibili e 4 unità nascoste può raggiungere un valore $O_{ML} \approx -1.84$. dRBM con un numero comparabile di connessioni fra neuroni, come verrà mostrato nella tabella sottostante, ha un valore $O_{ML} \approx -2.3$, ovvero circa il 25% in meno. Poiché gli algoritmi quantistici studiati sono perfettamente utilizzabili sia per addestrare "*full BMs*" che dRBMs, è stato così fornito un framework che non solo arricchisce i metodi classici ma che in tale contesto potenzialmente conduce ad un miglioramento degli aspetti modellistici.

n_v	n_{h1}	n_{h2}	CD	ML	% Improvement
6	2	2	−2.7623	−2.7125	1.80
6	4	4	−2.4585	−2.3541	4.25
6	6	6	−2.4180	−2.1968	9.15
8	2	2	−2.8503	−3.5125	−23.23
8	4	4	−2.8503	−2.6505	7.01
8	6	4	−2.7656	−2.4204	12.5
10	2	2	−3.8267	−4.0625	−6.16
10	4	4	−3.3329	−2.9537	11.38
10	6	4	−2.9997	−2.5978	13.40

Figura 21:Valori medi della funzione OML individuati con l'algoritmo "Contrastive Divergence" e l'algoritmo gradient descent per una dRBM dotata di 3 layers ed N=0.

4.6 Conclusioni

Un fondamentale risultato è legato al fatto che il problema dell'apprendimento tramite esempi di una Boltzmann Machine è riconducibile ad un problema di "*quantum state preparation*". Tale processo non richiede l'uso della "*Contrastive Divergence*" e nemmeno approssimazioni circa la topologia dei grafi rappresentativi del sistema. Il framework fornito dall'utilizzo dell'approccio quantistico al problema ha consentito nella fase di "*state preparation*" la ridefinizione dell'approssimazione "*mean-field*" a generare stati simili od equivalenti del tutto ai desiderati "*Gibbs states*". Il metodo fondato su tale fase consente ad una BM di essere addestrata usando un numero di operazioni che non dipende esplicitamente dal numero di "*layers*" in una dRBM. Garantisce, inoltre, una riduzione quadratica nel numero di volte con cui occorra fare accesso ai dati di training ed inoltre permette anche l'apprendimento di una BM di tipo "*full*".

I citati algoritmi, per concludere, possono essere parallelizzabili su una moltitudine di processori quantistici e l'insieme dei vantaggi percepiti, in un prossimo futuro, permetterà la diffusione del *Machine Learning* da una prospettiva quantistica.

Bibliografia

A Practical Guide to Training Restricted Boltzmann Machines, Toronto, August 2010, Geoffrey Hinton.

Quantum Deep Learning, USA, May 2015, Nathan Wiebe, Ashish Kapoor and Krysta M. Svore.

Quantum Boltzmann Machine, University of Waterloo, Canada, July 2017, Mohammad H. Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy and Roger Melko.