Università degli studi Milano-Bicocca

Scuola di Economia e Statistica

Corso di Laurea Magistrale in Scienze Statistiche ed Economiche

Macchine di Boltzmann per il Topic Modeling

Relatore:

Prof. Antonio Candelieri

Correlatore:

Prof. Elisabetta Fersini

Laureando:

Federico Rausa

mat. 919795

Anno accademico 2024/2025

Acknowledgements

Non sono abbastanza frequenti le occasioni in cui puoi ringraziare qualcuno in modo ufficiale. Pertanto cercherò di sfruttare al meglio questo particolare momento. I primi che devo ringraziare sono la prof Fersini e il prof Candelieri, per avermi dato la possibilità di questa tesi. E' una tesi che presenta tre elementi piuttosto rari: 1. mi appassiona molto. 2. mi ha consentito di dedicare risorse alla comprensione di concetti che desideravo approfondire da molto tempo. 3. presenta un impatto reale su un progetto di ricerca ancora in corso (OCTIS), di notorietà internazionale (non esagero), che dubito possa non avere un impatto positivo sul mio curriculum vitae.

Vorrei ringraziare il prof Candelieri anche per la sua disponibilità e gentilezza: mi ha sempre dato ampi spazi di libertà in questo lavoro, dandomi la possibilità di perseguire i miei interessi più che compiti assegnati dall'esterno.

Non posso non ringraziare i membri della famiglia. I miei genitori, prima di tutti, per la fiducia che mi hanno sempre dato, meritata per almeno un terzo delle volte.

Mia sorella, per avermi dimostrato cosa significa lottare per la propria vita.

I nonni Regina e Giampiero, alias Gina e Babbo, per essere sempre stati un rifugio per tutti, e che pur di starmi vicino hanno cambiato regione di residenza, alla giovane età di 70 anni. Cucinano piatti deliziosi, 9 volte su 10. Ringrazio la nonna per avermi aiutato a studiare alle medie, in assoluto il periodo peggiore della mia vita.

La Zia Lella, che mi ha sempre dato un letto a Genova. Anche lei cucina altrettanto bene, 99 volte su 100. Senza di lei non avrei conosciuto metà della parte bella della città che ora conosco.

Ringrazio gli amici dell'università: non ho parole per il bene che mi hanno fatto. Chi mi conosce sa che soffro di competitività, e loro, pur essendo statistici migliori, non mi hanno fatto soffrire per niente. In particolare Gian, Ema, Andrea-Morena, Gaia, Tommaso, Beatrice, Nicole, le due Elise, Ada e anche chi ho conosciuto meno. Ringrazio Ema per avermi fatto capire il Collapsed Gibbs Sampling per la LDA. Mi ha tolto un sassolino dalla scarpa.

Ringrazio gli amici di Genova: per il poco tempo passato insieme, mi hanno dato i momenti più felici della mia vita. In particolare Giovanni, Susanna, Tisso, Madda e tutti gli altri di cui per brevità non elenco i nomi. Sono la prima ragione per cui mi trasferirei fuori dalla Lombardia.

Ringrazio gli amici di Saronno: in una terra triste e desolata, mi dimostrano ogni giorno che per essere felici basta scegliere di esserlo. In particolare Arianna, Elisa, Cristian, Nicho, Alberto, Gioele (sperando che non me lo rinfacci), Marta, e tutti gli altri, che se no qualcuno si offende.

Ringrazio in modo speciale le persone che hanno avuto un impatto colossale sulla mia vita, ma con le quali per ragioni diverse ho perso i contatti: Fabio Coppini, Alberto Ferrara, Vincenzo Forte, Sara Verzellesi, Veruska Poloni (la prof di italiano del liceo: mi ha fatto capire cosa significa lavorare sodo), Stefano Bolzonella (il prof di matematica del liceo: senza di lui mai mi sarei spinto a fare la triennale di economia, figurarsi la magistrale in statistica), Rosa Guzzetti (la prof di italiano delle medie: sola luce in un inferno). Di loro in particolare ho un ricordo speciale, e posso dire che sono parti di me. Spero di rivederli per poterli ringraziare uno ad uno.

Ultimo ma non per importanza: il lettore.

Questa tesi non è facile da digerire. Non mi trattengo dal dire che se la leggerai per intero, capendo tutto, previa prova di conoscenza, ti pago un giropizza in segno di gratitudine. Non penso infatti che tanta gente si soffermerà oltre la pagina dei ringraziamenti, per cui è giusto offrire un incentivo.

Macchine di Boltzmann per il topic modeling

Federico Rausa

Abstract

Il topic modeling ha conosciuto strade diverse negli ultimi 20 anni. Prima sviluppatosi notevolmente con le reti bayesiane, tra tutte la LDA, ha conosciuto interessanti sviluppi di performance con le Macchine di Boltzmann, in particolare con il modello Replicated Softmax (RSM), tra il 2009 e il 2013. Queste hanno però rapidamente lasciato spazio dal 2015 a strumenti generativi basati sul popolare framework delle reti neurali feedforward, in particolare dei modelli VAE e BERT, molto performanti ma carenti di interpretabilità, a causa dell'elevato numero di parametri, e computazionalmente molto costosi. Il modello RSM nella sua definizione originaria si presenta più performante della LDA in termini di perplexity, mantenendo lo stesso grado di efficienza e interpretabilità. Nell'ultimo ventennio hanno conosciuto ampia diffusione le tecniche di ottimizzazione dell'algoritmo di gradient descent (in particolare con riguardo alle reti neurali), di ottimizzazione del contrastive learning (in particolare per le Macchine Ristrette di Boltzmann) e di ottimizzazione bayesiana (universalmente note nel mondo del machine learning). In letteratura non sono tuttavia noti esperimenti in cui tali tecniche di ottimizzazione sono applicate al training del modello RSM, mentre sono note delle sue versioni migliorate, come il penalized-RSM e l'Over-RSM. In questo lavoro si intendono svolgere tali esperimenti, seguendo le tecniche di ottimizzazione bayesiana, di validazione e di comparazione dei topic models fornite dalla libreria OCTIS (Optimizing and Comparing Topic Models is Simple), riconosciuta per il suo rigore scientifico, al fine di spingere il modello RSM ai suoi limiti in efficienza, performance e interpretabilità.

Contents

1 Introduzione

Il topic modeling è una branca del Natural Language Processing (NLP) estremamente giovane. Il primo algoritmo specificamente ideato per questo task risale al 1991, e da allora la materia ha conosciuto uno sviluppo ininterrotto.

I topic models sono una classe di modelli statistici non supervisionati, applicabili in generale come strumenti di analisi dati ma comunemente considerati una branca del NLP. Presentano infatti l'esclusiva capacità di riassumere dati di testo attraverso distribuzioni di parole. Aiutano nelle attività di classificazione dei documenti e di organizzazione delle informazioni. Vengono considerati come i più sofisticati strumenti di text mining [Hotho et al., 2005], una branca del NLP, nella quale sono fondamentali per fornire strumenti di riduzione della dimensionalità di dati di testo, al fine di supportare modelli di tipo discriminativo. Esempio di applicazione tipica, in text mining, è il problema della sentiment analysis: dato un testo, si vuole addestrare un modello discriminativo, come SVM o RandomForest, al fine di assegnarvi un label (sarcastico, accademico, positivo ecc.) [Phan et al., 2008]. Per questo tipo di problema un Topic Model consente di generare dai dati di testo, direttamente intrattabili per i modelli discriminativi più semplici, poche covariate utili a migliorarne le previsioni.

Il topic modeling è un'attività interessante. Purtroppo tale task comporta quasi sempre l'utilizzo di modelli statistici quasi sempre avanzati o ad alto costo computazionale. Il modello principe del topic modeling è probabilmente la Latent Dirichlet Allocation, o LDA, che rappresenta una rete bayesiana di comprensibile interpretazione ma con una procedura di training complessa, che per essere compresa richiede una profonda conoscenza della statistica bayesiana e dei modelli probabilistici. Pertanto, molti storici, giuristi, psicologi, sociologi o linguisti che si sarebbero potuti interessare agli strumenti offerti dal topic modeling, vi hanno rinunciato o si sono fatti bastare un'utilizzo superficiale degli stessi a causa della loro complessità. Modelli precedenti alla LDA, meno complessi, presentano notevoli svantaggi.

Tra il 2009 [Hinton and Salakhutdinov, 2009] e il 2013 [Srivastava et al., 2013] sono stati proposti una classe di topic models basati sulle Restricted Bolzmann Machines, o RBM. Si tratta di reti neurali basate sulla nozione di energia e sulla distribuzione di Bolzmann, ideate dal premio Nobel per la fisica del 2024 Geoffrey Hinton. Il primo di tali modelli è il Replicated

Softmax Model [Hinton and Salakhutdinov, 2009], o RSM, sul quale poi sono state applicate diverse innovazioni.

Il Replicated Softmax è un topic model riconosciuto, di facile interpretazione e ritenuto più performante dei modelli tradizionali, per cui può rappresentare una valida alternativa per ricercatori con formazione statistica ridotta ma in grado di trarre grande vantaggio dall'utilizzo di un topic model nei propri casi studio.

Tuttavia, con l'introduzione dei Word Embeddings, in particolare di Word2Vec nel 2013, dei VAE (Variational Autoencoder) nel 2014 [Kingma and Welling, 2013] e dei BERT nel 2018 [Vaswani et al., 2017], la ricerca sui modelli basati sull'energia nell'ambito del topic modeling si è arrestata, per dare spazio ad altri NTM (Neural Topic Models) più flessibili e performanti, ma basati su ipotesi completamente differenti.

In questo lavoro, si vogliono esplorare le potenzialità del topic modeling basato sull'energia, in particolare del modello RSM. Si vogliono sperimentare i metodi di ottimizzazione comunemente noti nel mondo delle reti neurali per ottimizzare tale modello e ridurre il costo computazionale della procedura di training. Si vogliono inoltre sfruttare gli strumenti di ottimizzazione bayesiana per identificare la miglior combinazione di iperparametri del modello, incluso il numero di topics, avendo cura di non far esplodere il costo computazionale della procedura di analisi nel suo complesso.

Nel corso delle analisi ci si avvale del framework OCTIS [Terragni et al., 2021], al fine di svolgere esperimenti strutturati, comparare correttamente il RSM con topic models tradizionali, e sfruttare gli strumenti di ottimizzazione bayesiana presenti all'interno della libreria.

Parte di questo lavoro comprende un aggiornamento di OCTIS, al fine di includere al suo interno il RSM originale e alcune sue versioni successive.

1.1 obiettivi

Obiettivi del presente lavoro sono i seguenti:

- introdurre le Macchine Ristrette di Boltzmann come possibili strumenti di topic modeling, nello specifico concentrandosi sul modello RSM e sulle sue versioni potenziate
- adottare tecniche di ottimizzazione bayesiana messe a disposizione nella libreria OCTIS

per migliorare le performance degli stessi attraverso un ridotto numero di sperimentazioni

- ridurne il costo computazionale nelle fasi di training e inferenza attraverso strumenti di contrastive learning e di ottimizzazione dell'algoritmo di discesa del gradiente
- fornire un'adeguata interpretazione di tali modelli, a partire dalle più generali proprietà delle reti di Markov e della Misura di Gibbs
- confrontarli, per via teorica e sperimentale, con altri topic models riconosciuti in letteratura, come LSI, pLSI e LDA, utilizzando il framework di OCTIS per svolgere esperimenti attraverso tecniche replicabili e riconosciute in letteratura.

1.2 contenuti

I paragrafi che seguono hanno le seguenti funzioni:

- il capitolo 2 mira a fornire un quadro generale di cosa sia il topic modeling, includendo definizioni minimali dei più popolari topic models
- nel capitolo 3 si presenta la libreria di python OCTIS come strumento di preprocessing, training, ottimizzazione bayesiana e comparazione di topic models
- nel capitolo 4 si fornisce una descrizione di alcune fra le più note misure di performance per il topic modeling
- nel capitolo 5 si definisce il concetto di campo casuale di Gibbs, con numerose parentesi storiche e algoritmiche, al fine di poter comprendere a fondo le proprietà del modello RSM
- nei capitoli 6 e 7 si definiscono le RBM, famiglia di modelli di cui il modello RSM fa parte
- nei capitoli 8, 9, 10 e 11 si descrivono i topic model basati sull'energia e delle oppurtune strategie per renderli maggiormente efficienti

- nel capitolo 12 si introduce la materia dell'ottimizzazione bayesiana, con particolare focus sugli strumenti offerti da OCTIS
- nei capitolo 13 e 14 si presentano i risultati sperimentali, ottenuti attraverso codice python secondo la procedura di OCTIS
- nel capitolo 15 si conclude il lavoro, presentandone i limiti, i punti di forza e le potenzialità per sviluppi futuri
- Nell'appendice si riportano dettagli sul alcune fasi del lavoro

2 I topic model tradizionali

2.1 Terminologia di base

Obiettivo del topic modeling è, dato un documento, quello di identificarne i topic, ovvero gli argomenti, intesi come variabili latenti ad esso associati con una certa probabilità.

Ci sono alcuni termini tecnici coniati in materia.

Un topic è una variabile latente, presente in ogni documento con una certa probabilità, stimabile attraverso il contenuto del testo.

Con il termine corpus si intende una parte del dataset, ovvero la lista dei testi dei documenti. Un documento rappresenta il principale input di qualsiasi topic model. Bisogna notare che un dataset per topic models può tuttavia contenere altre variabili, come una data di pubblicazione, un autore e dei tag associati al testo, idealmente considerabili come i veri topic, e necessari quando si vuole addestrare un topic model seguendo una procedura di training supervisionato.

Con il termine vocabolario, o dizionario, si intende l'insieme di tutti i token (parole o n-grammi) presenti in un documento che sono considerati dal modello al fine di stimarne i topic. In fase di preprocessing solo alcuni dei token disponibili vengono estratti dal testo. Vengono filtrate ad esempio le stopwords, termini con alta frequenza ragionevolmente non collegate con alcun argomento specifico, come gli articoli e le preposizioni.

Anche le parole a frequenza eccessivamente bassa possono essere escluse. Quando si parla di frequenza dei token, bisogna fare una distinzione tra term frequency e document frequency. La document frequency è il numero di volte che complessivamente un token si ripete in un documento. La term frequency è la document frequency normalizzata per il massimo sul documento. Generalmente, vengono rimossi i termini con somma delle document frequency su tutti i documenti pari a 1 o inferiore a una soglia bassa specificata dall'utente, ad esempio 10.

2.2 L'assunzione BoW

Nel topic modeling è denominata assunzione BoW, Bag of Words [Barde and Bainwad, 2017], l'ipotesi secondo cui i topic trattati in un documento dipendono dal numero di volte che le parole di un dato vocabolario si ripetono nel testo del documento, e non dal loro ordina-

mento. Il vocabolario rappresenta quindi l'insieme dei token filtrati in fase di preprocessing il cui conteggio va a formare la rappresentazione BoW di ogni documento presente nel corpus. Sotto l'assunzione BoW dunque, all'interno del dataset un documento rappresenta un data point, e i token presenti nel vocabolario rappresentano le sue features.

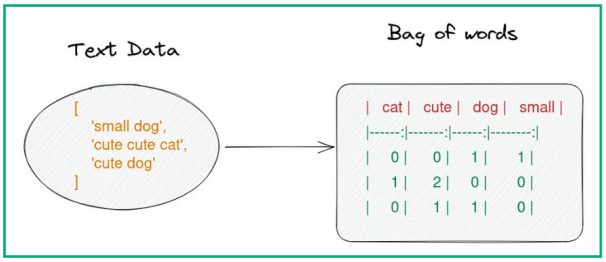


Figure 1: BoW: Esempio di matrice Documento Termine

2.3 preprocessing

Per i topic model classici, ovvero che rispettano l'assunzione BoW, il corpus può essere rappresentato come una matrice documento termine, o DTM (Document Term Matrix). Tale matrice ha dimensione DxV, dove D è il numero di documenti che contiene, V è il numero di parole presenti nel vocabolario e ogni cella comprende il numero di volte che la v-esima parola compare nel d-esimo documento. Tale valore è denominato document frequency.

La costruzione della DTM può considerarsi la conclusione del preprocessing di un corpus. Tuttavia è facile ottenere miglioramenti nelle metriche di performance dei modelli adottando delle trasformazioni della DTM, come $ln(1 + df_{wd})$, oppure adottando la matrice degli score TF-IDF [Gupta et al., 2024].

In formule, si definiscono, per ogni documento d e ogni token w in un corpus D:

$$df_{wd} = |w \in D : w \in d| \tag{1}$$

$$tf_{wd} = \frac{df_{wd}}{\max_w df_{wd}} \tag{2}$$

$$idf_{wd} = \frac{|D|}{|w \in D : w \in d|} \tag{3}$$

$$tfidf_{wd} = \frac{idf_{wd}}{|D|} \cdot tf_{wd} \cdot idf_{wd} \tag{4}$$

Dove df è la document frequency, idf è l'inverse document frequency, tf è la term frequency (document frequency normalizzata sul documento) e $tfidf$ è uno score più capace degli altri di catturare la rilevanza di un token nel corpus.

In generale, una matrice di score di dimensione DxV che collega i token ai documenti è denominata DFM (Document Feature Matrix). La DTM e la TFIDF sono le due DFM generalmente più adottate.

2.4 flusso di lavoro del topic modeling

Il processo di adozione di un topic model si basa sulle seguenti fasi:

1 - costruzione del dataset e del corpus di documenti, ovvero una lista di stringhe

2 - svolgimento del preprocessing con le medesime modalità su ogni documento

3 - divisione del corpus in train, test e validation, come nei comuni processi di machine learning

4 - costruzione della Document-Feature Matrix, se il topic model rispetta l'assunzione BoW

5 - training del modello

6 - ottimizzazione basata sulle performance ottenute con il validation set

7 - ripetizione degli step 5 e 6 fino al raggiungimento di un livello di plateau dei miglioramenti, o altro criterio di arresto

8 - valutazione finale del modello sul test

9 - costruzione di una rappresentazione dei topic

Si vede che non differisce molto dal modo in cui viene addestrato un tipico modello di machine learning. Le principali differenze sono nel preprocessing, nel tipo di modello e nell'interpretazione dei suoi output.

2.5 Cos'è un topic model

Per comprendere cosa sia effettivamente un topic model (facilmente confondibile con un qualsiasi algoritmo di clustering o di riduzione della dimensionalità) è necessario fare

riferimento agli elementi comuni dei modelli finora proposti. L'elemento comune più evidente è forse la formulazione dei suoi output. All'interno della libreria OCTIS gli output di ogni modello devono contenere 2 elementi fondamentali:

- una matrice di dimensione TxK dove T è il numero di topic e K il numero di token presenti nel dizionario
- una matrice di dimensione TxD dove D è il numero di documenti presenti in un dato corpus.

Si potrebbe erroneamente pensare che un topic model si limiti a fare clustering di documenti. Un topic model presenta infatti due caratteristiche che lo distinguono dai modelli tradizionali:

- è un modello probabilistico non supervisionato, ovvero assegna a ciascuna osservazione del dataset (i documenti) una probabilità di fare riferimento a un determinato topic (dice in che misura un topic è presente nel documento, non se il documento appartiene a un cluster o meno).
- le feature del dataset (le parole) devono essere direttamente collegabili ai topic, e ciascuno di questi deve fornire un ranking di importanza per ogni feature.

Un topic model non è quindi facilmente identificabile nella maggior parte dei metodi non supervisionati, e questo comporta un notevole grado di complessità.

2.6 Tassonomia dei modelli di topic modeling

Si introducono brevemente i modelli storicamente più importanti tra i tanti che sono stati proposti, che negli ultimi 25 anni hanno portato allo sviluppo di questa branca del Natural Language Processing. In letteratura sono presenti diverse classificazioni dei topic models. In questa tesi ci si affida in particolare alle sotto classificazioni proposte in [Abdelrazek et al., 2023] e in [Sharma et al., 2017] , a una approssimativa distinzione delle fasi di sviluppo che la materia ha conosciuto negli anni, e a classificazioni più generali dei modelli statistici sui quali diversi topic models sono costruiti.

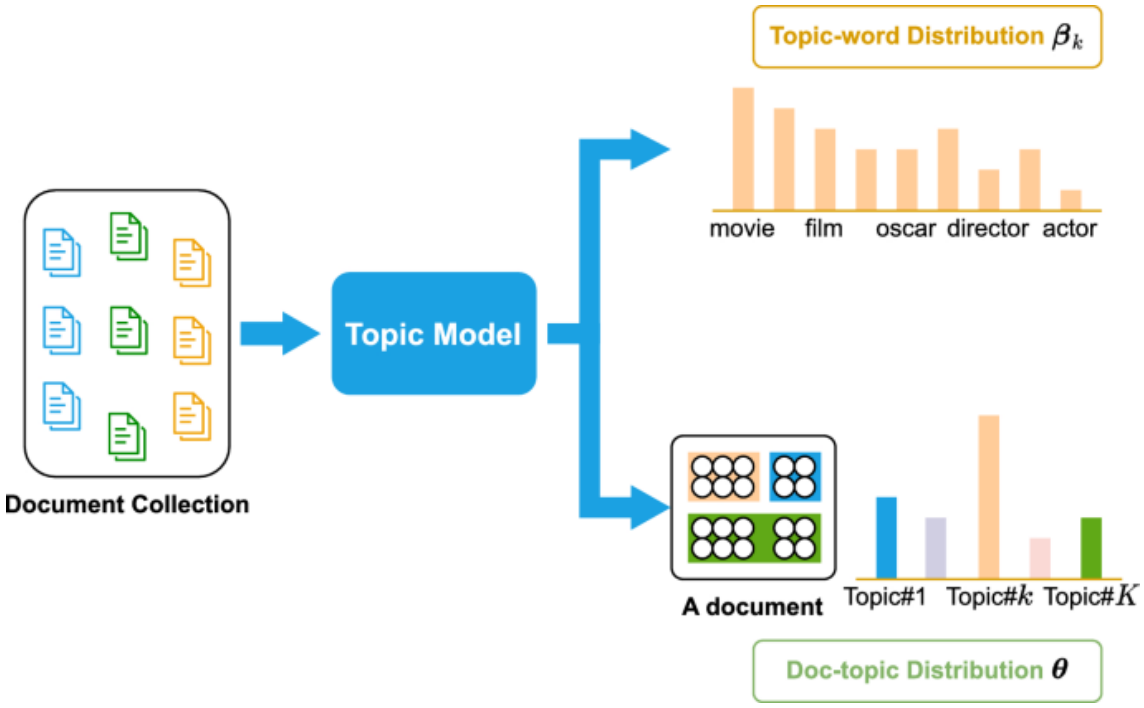


Figure 2: Schema illustrativo dell'output di un topic model, tratto da [Wu et al., 2024]

I topic models e la loro storia si potrebbero idelamente accorpare in 5 categorie, associate a 5 più o meno brevi e recenti fasi di sviluppo.

La prima tipologia di modelli si potrebbe considerare basata sui metodi di decomposizione spettrale direttamente derivanti dall'algebra lineare, come la SVD e la NMF. Una seconda classe di modelli è quella che si propone di utilizzare delle reti bayesiane per descrivere l'interazione tra i documenti e le parole attraverso i topic, e comprende il pLSI, proposto da Hofmann nel 1999, e la LDA, proposta da Blei nel 2003. Una terza classe, basate sui modelli basati sull'energia e sulle RBM, proposte da Hinton nel 2006, comprende tutta l'evoluzione del modello RSM, proposto sempre da Hinton nel 2009 e che ha conosciuto sviluppi ulteriori fino al 2013. Tale classe di modelli ha avuto vita breve, e obiettivo di questa tesi è approfondirla. Nel 2013 e nel 2014 sono stati pubblicati rispettivamente i primi paper sui Word Embeddings e sui VAE. Questi scoperte hanno rivoluzionato il mondo del NLP, dando origine ai nuovi NTM, Neural Topic Models, basati principalmente sui VAE e addestrabili con gradint-descent come le reti neurali classiche. Una quinta e ultima classe di metodi di topic modeling è figlia del paper "Attention is All You Need", pubblicato nel 2017 [Vaswani et al., 2017] e considerato l'origine dei Transformer e dei LLM. Di questa classe di

topic models sono stati invece protagonisti i BERT (Bidirectional Encoder Representations from Transformers), adattati con varie procedure a task di topic modeling. In tale classe si sfruttano i document embeddings prodotti da dei LLM preaddestrati per costruire cluster di documenti. In ciascuna di queste fasi i nuovi modelli hanno superato i precedenti in termini di metriche di validazione ma spesso richiedono un costo computazionale maggiore e rappresentano evoluzioni o combinazioni dei vecchi modelli.

2.7 Modelli basati sulla riduzione della dimensionalità

I primi modelli in grado di fornire una rappresentazione dei topic come aggregati di parole in modo convincente sono stati quelli per la riduzione della dimensionalità. La SVD (Singular Value Decomposition) [Mirzal, 2016] [Ke and Wang, 2024]e la NMF (Non Negative Matrix Factorization) [Arora et al., 2012] rappresentano i primi topic models pienamente considerabili come tali.

Un modello per la riduzione della dimensionalità ha lo scopo di mappare una matrice NxK su una matrice NxM con M << K. Sono in generale utili per ridurre il numero di feautures da passare a modelli di machine learning o per produrre rappresentazioni grafiche di dati a più di tre dimensioni.

Il topic model più conosciuto di questa specie è il modello LSI. L'algoritmo LSI, Latent Semantic Indexing, anche detto LSA, Latent Semantic Analysis, consiste semplicemente nell'applicazione della decomposizione spettrale SVD, Singular Value Decomposition, alla DFM. L'algoritmo è stato brevettato nel 1988 da un team di ricercatori della Bell Communications Research.

Il miglior topic model di questa specie risulta essere la NMF, proposta nel 2000 [Lee and Seung, 2000] che ha poi rappresentato la definizione di topic model comune alla LDA e ai modelli più recenti.

Al fine di rappresentare gli output dei topic models in grafici a due o tre assi, in materia di text mining si fa spesso ricorso a un metodo di riduzione della dimensionalità.

In particolare sono stati ampiamente adottati a questo fine la PCA (Principal Component Analysis) [Min et al., 2010] e il t-SNE (t distributed Stocastic Neighbour Embedding) . La PCA e il t-SNE si rivelano in molti casi utili per costruire cluster di termini e rappresentazioni

grafiche degli output di altri topic models, ma non sono topic models veri e propri, in quanto non forniscono l'output idealmente richiesto da un topic model. Più precisamente, non sono topic models in quanto riducono la varianza globale dei dati su un numero minore di variabili, ma senza decomporla in fattori latenti (cosa che invece avviene con SVD e NMF). In altre parole, mentre un topic model vero e proprio dovrebbe restituire due output (una matrice per collegare le parole ai topic e una matrice per collegare i topic ai documenti) questi modelli forniscono esclusivamente una rappresentazione compressa dei documenti o delle parole, senza costruire i topic.

Tra i due ha trovato più diffusione il t-SNE, pubblicato da Hinton e Van der Maaten nel 2008,[Maaten and Hinton, 2008] [Kim et al., 2016], figlio del modello SNE (Stochastic Neighbor Embedding) proposto da Hinton e Roweis già nel 2002 [Hinton and Roweis, 2002].

Il metodo t-SNE è la principale soluzione al problema del crowding presente nella PCA e nel metodo SNE originario, a causa del quale i punti nello spazio latente tendono ad accorparsi in una sfera. Questo elemento lo porta ad essere il miglior algoritmo per fornire una rappresentazione visiva di dataset anche ad alta dimensionalità, come le DFM e i dataset delle immagini come MNIST [Deng, 2012].

Il t-SNE si addestra con gradient descent attraverso l'utilizzo della divergenza di Kullback-Leibler come funzione di perdita. Questa sua formulazione ha consentito allo stesso autore Van der Maaten nel 2009 di introdurre il parametric t-SNE [Van Der Maaten, 2009], una rappresentazione del t-SNE come rete neurale. Questa formulazione consente di ridurre drasticamente il numero di parametri da ottimizzare e di fare previsioni su nuovi documenti, come un qualsiasi encoder.

2.7.1 LSI

Il Latent Semantic Indexing, o Latent Semantic Analysis, si può definire come l'applicazione della Singular Value Decomposition (SVD) alla DFM. Si è un metodo geometrico che trova numerose applicazioni, in particolare nella compressione dei dati. La SVD consente di fattorizzare una matrice $A \in \mathbb{R}^{DxW}$ in tre matrici U, Σ, V tali che:

- $$A = U\Sigma V^T \tag{5}$$

- $U \in \mathbb{R}^{DxK}, V \in \mathbb{R}^{WxK}$ sono matrici ortonormali, ovvero tali che $U^TU = I, U^T = U^{-1}, V^TV = I, V^T = V^{-1}$
- $\Sigma \in \mathbb{R}^{KxK}$ è la matrice diagonale dei valori singolari non negativi di AA^T e di A^TA
- U è la matrice degli autovettori di AA^T
- V è la matrice degli autovettori di A^TA

Se A è una matrice di dati BoW l'interpretazione delle tre matrici è alquanto intuitiva:

- La matrice Σ contiene sulla diagonale il grado di importanza di ciascun topic
- La matrice U contiene l'importanza che ciascun topic presenta per ogni documento
- La matrice V contiene i contributi dati da ogni parola a ogni topic

Normalizzare Le righe di U e le colonne di V attraverso una normalizzazione min-max consente di stimare rispettivamente le probabilità condizionate delle parole dato ciascun topic e dei topic dato ciascun documento.

2.7.2 NMF

Il modello NMF [Wang and Zhang, 2012] decompone una DFM come prodotto scalare di due matrici non negative (da cui NMF: Non Negative Matrix Factorization). Il numero di topic è il numero di colonne della prima matrice, di dimensione DxK, e di righe della seconda, di dimensione KxV, che consentono di ottenere come prodotto la matrice DTM.

La SVD è un algoritmo che per puro caso ha trovato applicazione nel topic modeling, e ha forse definito per primo cosa un topic model dovrebbe fare: restituire degli score di probabilità di ogni parola per ogni topic e di ogni topic per ogni documento. La SVD restituisce altre informazioni (come la matrice Σ) e impone dei vincoli (come l'ortogonalità delle due matrici degli score) che non si rendono necessari a tal fine. La NMF è un altro algoritmo di fattorizzazione di una matrice che si presenta naturalmente più idoneo al problema del topic modeling (e anche questa volta si è trattato di un caso: la NMF, come la SVD, è uno strumento general purpose dell'algebra lineare e della geometria).

Il suo obiettivo è quello di fattorizzare una data matrice A, contenente solo valori non negativi (da cui il nome) nel prodotto di due matrici W, H (di conseguenza anch'esse non negative) tali che $A \approx WH$.

Bisogna evidenziare la natura approssimata di questa decomposizione: Mentre nella PCA e nella SVD il risultato è deterministico, a meno di problemi numerici per matrici di grandi dimensioni, nella NMF una diversa definizione della funzione di perdita e il diverso livello di convergenza dell'algoritmo possono portare ad approssimazioni differenti.

In [Lee and Seung, 2000] si dà una dettagliata descrizione della procedura di stima. Le funzioni di perdita della stima $B = WH$ adottate sono due:

- la distanza euclidea al quadrato:

$$L(WH) = ||A - WH||^2 = \sum_{ij}(A_{ij} - B_{ij})^2 \tag{6}$$

- la divergenza di Kullback-Leibler:

$$L(WH) = D(A||WH) = \sum_{ij}\left(A_{ij}log\frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij}\right) \tag{7}$$

Nello stesso paper si propongono 4 algoritmi di update delle matrici W e H, due per ciascuna funzione di perdita, di cui una di tipo moltiplicativo e l'altra di tipo additivo (analoga al gradient descent). Lee e Seoung [Lee and Seung, 2000] dimostrano che il caso moltiplicativo e quello additivo sono equivalenti per determinati valori del learning rate usato nel caso additivo. Questo accade per entrambe le funzioni di perdita. Dimostrano anche che nel caso moltiplicativo, per entrambe le funzioni di perdita, esse sono non crescenti durante il processo di aggiornamento dei parametri. Tutti i metodi di stima si svolgono in due passi, analoghe a delle iterazioni di tipo Expectation Maximization:

- nel primo si aggiorna H fissato W
- nel secondo si aggiorna W fissato H

Nel caso moltiplicativo con funzione di perdita $||A - WH||^2$ le funzioni di update sono:

$$H_{ku} \leftarrow H_{ku} \frac{(W^T A)_{ku}}{(W^T W H)_{ku}} \tag{8}$$

$$W_{ik} \leftarrow W_{ik} \frac{(A H^T)_{ik}}{(W H H^T)_{ik}} \tag{9}$$

Nel caso additivo con funzione di perdita $||A - WH||^2$ le funzioni di update sono:

$$H_{ku} \leftarrow H_{ku} + \eta_{ku}((W^T A)_{ku} - (W^T W H)_{ku}) \tag{10}$$

$$W_{ik} \leftarrow W_{ik} + \eta_{ik}((A H^T)_{ik} - (W H H^T)_{ik}) \tag{11}$$

Dove se si vuole adottare gradient descent classico si fissano $\eta_{ku} = \eta \forall k, u$ e $\eta_{ik} = \eta \forall i, k$ con $\eta < 0.1$ costante, mentre se si vuole ottenere un risultato equivalente al caso moltiplicativo si fissano $\eta_{ku} = \frac{H_{ku}}{(W^T W H)_{ku}}$ e $\eta_{ik} = \frac{W_{ik}}{(W H H^T)_{ik}}$.

Nel caso moltiplicativo con funzione di perdita $D(A||WH)$ le funzioni di update sono:

$$H_{ku} \leftarrow H_{ku} \frac{\sum_i W_{ik} A_{iu} / (WH)_{iu}}{\sum_i W_{ik}} \tag{12}$$

$$W_{ik} \leftarrow W_{ik} \frac{\sum_u H_{ku} A_{iu} / (WH)_{iu}}{\sum_u H_{ku}} \tag{13}$$

Nel caso additivo con funzione di perdita $D(A||WH)$ le funzioni di update sono:

$$H_{ku} \leftarrow H_{ku} + \eta_{ku} \left(\sum_i W_{ik} \frac{A_{iu}}{WH_{iu}} - \sum_i W_i k \right) \tag{14}$$

$$W_{ik} \leftarrow W_{ik} + \eta_{ik} \left(\sum_u H_{ku} \frac{A_{iu}}{WH_{iu}} - \sum_u H_{ku} \right) \tag{15}$$

Anche in questo caso fissato η costante e piccolo si adotta gradient descent, mentre si ritorna alla forma moltiplicativa per $\eta_{ku} = \frac{H_{ku}}{\sum_i W_{ik}}$ e $\eta_{ik} = \frac{W_{ik}}{\sum_u H_{ku}}$.

In [Ding et al., 2008] si dimostra che quando usata con la funzione di perdita Kullback-Leibler, la NMF conduce a risultati equivalenti a quelli del pLSI.

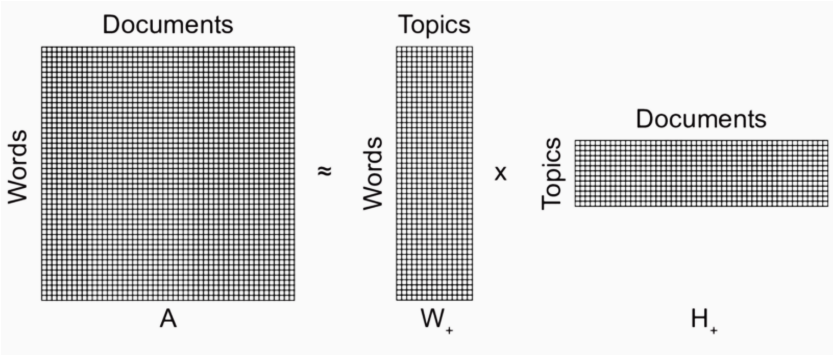


Figure 3: Rappresentazione algebrica della Matrice documento-termine sotto il topic model NMF, tratto da [Kuang et al., 2017]

2.8 Modelli Bayesiani

Il pLSI [Hofmann, 1999] è il primo modello bayesiano adottato nel mondo dei topic models. Di poco si discosta dalla pLSA (probabilistic Latent Semantic Analysis) [Hofmann, 2013].

La LDA [Blei et al., 2001] [Blei et al., 2003] rappresenta la diretta evoluzione del pLSI, e si differenzia da questo introducendo sparsità nelle matrici delle probabilità condizionate attraverso delle Prior Dirichlet. Bisogna notare che la LDA è forse il modello più conoscito e adottato in materia di topic modeling. I concetti alla base di queste reti bayesiane, come i concetti di indipendenza delle parole dai documenti condizionatamente ai topic, e quello di sparsità dei topic, hanno ispirato molti modelli successivi.

Il concetto alla base del pLSI e della LDA è in realtà ereditato dal modello NMF [Ding et al., 2008]. L'idea del pLSI è quella di considerare la seconda matrice come una probabilità condizionata $p(w|t)$ e la prima come $p(t|d)$ dove w, t, e d sono rispettivamente gli indici del token, del topic e del documento. In tale modo, la matrice risultante dalla fattorizzazione presenta $p(w|d) = \sum_t p(w|t) \cdot (t|d)$, e l'obiettivo del training è quello di massimizzare la verosimiglianza del corpus modificando le due matrici di probabilità condizionate. Questa formulazione consente alla rete bayesiana di trattare le righe delle due matrici come variabili aleatorie multinomiali.

La LDA adotta lo stesso principio, ma introduce delle prior Dirichlet sulle multinomiali per controllare il livello di sparsità dei topic attraverso i parametri della distribuzione Dirichlet, coniugata naturale della distribuzione Multinomiale. Tali ulteriori parametri sono definiti a priori dall'utente, e rappresentano quindi degli iperparametri. Recenti lavori hanno pro-

posto di selezionarli sfruttando una procedura di ottimizzazione bayesiana, soluzione messa a disposizione dalla libreria OCTIS.

2.8.1 pLSI

Il probabilistic Latent Semantic Indexing [Hofmann, 1999] è un modello basato sull'idea che ogni documento del corpus sia frutto di un processo generativo.

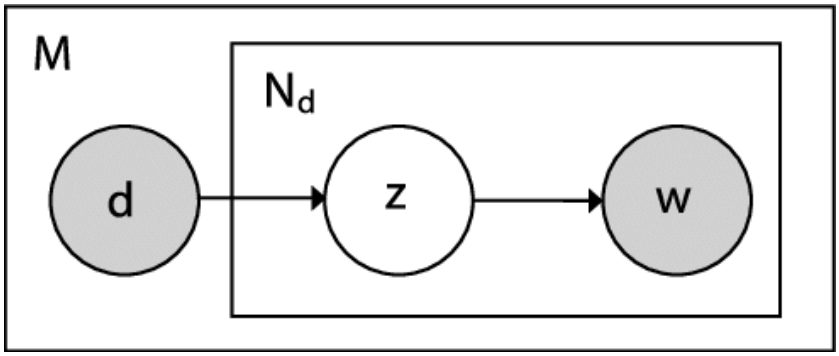


Figure 4: Schema del pLSI in plate notation
Si nota che il pLSI è una rete bayesiana, con struttura DAG, nella quale vale la proprietà locale di Markov: le parole sono indipendenti dai documenti condizionatamente ai topic. In altri termini, ogni documento è una mistura di topic, e ogni topic è una mistura di parole.

Si indicano con w, z, d, rispettivamente, gli indici delle parole nel vocabolario, dei topic e dei documenti. Ogni documento d è caratterizzato da una distribuzione multinomiale dei topic $p(z|d)$, e ogni topic z è caratterizzato da una distribuzione $p(w|z)$. Per generare la rappresentazione BoW di un testo d di N parole, per ogni token con indice i:

- prima si campiona $z_i \sim Multinom(p(z|d))$
- poi si campiona $w_i \sim Multinom(p(w|z_i))$

Il modello è identificato dalle due matrici contenenti le probabilità condizionate $p(z|d)$ e $p(w|z)$, e la verosimiglianza di ogni parola è data da $p(w|d) = \sum_z p(w|z)p(z|d)$.

L'inferenza su questi parametri segue una procedura di Expectation Maximization, ed è illustrata nell'algoritmo che segue.

Algorithm 1 EM per pLSI

Siano dati K topic in un corpus di D documenti e un vocabolario di V parole. Sia n_{wd} il numero di volte che la parola w compare nel documento d (o altra misura della document feature matrix).
- inizializzazione:
$p(z|d) = 1/K \forall d$
$p(w|z) = 1/V \forall w$
- for t in 1:T:
- Expectation step:

$$p(z|w,d) \propto p(w|z)p(z|d) \tag{16}$$

Si normalizza $p(z|w,d)$ rispetto a z.
- Maximization step:

$$p(w|z) \propto \sum_d n_{wd} p(z|w,d) \tag{17}$$

Si normalizza $p(w|z)$ rispetto a w.

$$p(z|d) \propto \sum_w n_{wd} p(z|w,d) \tag{18}$$

Si normalizza $p(z|d)$ rispetto a z.
- endfor

2.8.2 LDA

L'Allocazione di Dirichlet Latente rappresenta un'evoluzione del pLSI. Il processo generativo include l'utilizzo di due vettori di iperparametri (solitamente ridotti a due scalari ripetuti su di essi),α e β, che rappresentano le parametrizzazioni di due distribuzioni di Dirichlet, utili per controllare il livello di sparsità dei topic sui documenti e delle parole sui topic.

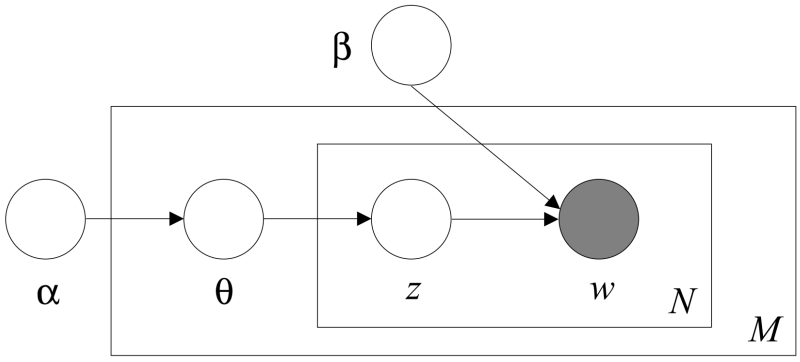


Figure 5: Schema della LDA in plate notation
Nella figura, tratta da [Blei et al., 2003], N è il numero di parole nel documento i-esimo, e M è il numero di documenti nel corpus. α e β sono i parametri costanti delle Dirichlet Prior, θ è un parametro document-level, che identifica la distribuzione di un documento sui topic, tale che $z \sim Multinom(\theta)$, z e w sono estrazioni word-level. La doppia dipendenza della parola w da z e β sottintende la presenza del parametro di una Multinomiale $\varphi_z \sim Dir(\beta)$ tale che $w \sim Multinom(\varphi_z)$. Si possono fare analoghe considerazioni a quelle fatte per il pLSI.

Il processo generativo della LDA segue queste fasi:

- per ogni documento si estrae $\theta_i \sim Dirichlet(\alpha)$ per $i = 1, ..., M$
- per ogni topic si estrae $\varphi_k \sim Dirichlet(\beta)$ per $k = 1, ..., K$
- Per ogni parola $j \in \{1, .., N_i\}$ di ogni documento $i \in \{1, ..., M\}$:
- prima si estrae un topic $z_{ij} \sim Multinom(\theta_i)$
- poi si estrae una parola $w_{ij} \sim Multinom(\varphi_{z_{ij}})$ (si noti che il parametro della multinomiale dipende dal topic z_{ij} appena campionato)

La Dirichlet è la prior coniugata della Multinomiale. Attraverso i parametri è possibile usarla per controllare la sparsità dei topic sui documenti (attraverso β) e delle parole sui

topic (attraverso α). I parametri della Dirichlet rappresentano il livello di concentrazione dei punti da essa campionati. Il supporto di una Dirichlet a k-dimensioni è un simplesso k-dimensionale. Se i suoi k parametri sono fissati uguali a una costante arbitraria, con essa è possibile controllare la sparsità dei punti sul simplesso, ovvero decidere quanto spingerli a concentrarsi nei suoi angoli (ogni angolo corrisponde a un livello di probabilità pari a 1 per un'esito della multinomiale da essa campionato, e 0 per tutti gli altri esiti in essa campionabili).

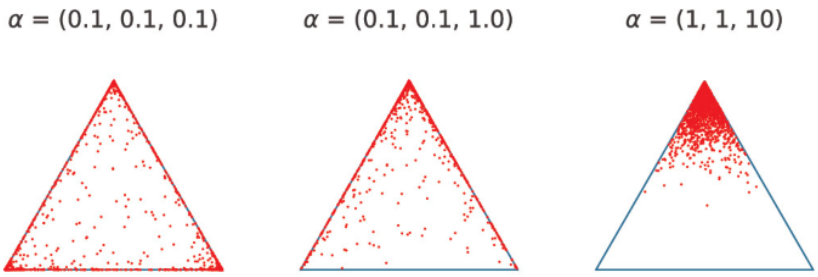


Figure 6: Esempio di Dirichlet per parametri minori di 1
Si vede che per parametri compresi tra 0 e 1 i punti campionati da una Dirichlet tendono ad allontanarsi dal centro del simplesso.

La distribuzione congiunta dei parametri della LDA racchiude la struttura di causalità descritta nel processo generativo:

$$P(\mathbf{W}, \mathbf{Z}, \theta, \varphi | \alpha, \beta) = \prod_{k=1}^{K} p(\varphi_k|\beta) \prod_{i=1}^{M} p(\theta_i|\alpha) \prod_{t=1}^{N} p(z_{it}|\theta_i) p(w_{it}|\varphi_{z_{it}}) \tag{19}$$

Dove i termini $p(\varphi_k|\beta)$ e $p(\theta_i|\alpha)$ sono le densità sotto la Dirichlet mentre $p(z_{it}|\theta_i)$ e $p(w_{it}|\varphi_{z_{it}})$ sono valori di probabilità in (0,1) stimati sotto il modello.

Obiettivo del processo di inferenza è quello di stimare le matrici di probabilità di elementi $\theta_{ik} = p(z|i)$ e $\varphi_{kw} = p(w|z)$, in modo analogo al pLSI.

Sono stati proposti due algoritmi per ottenere la stima di questi parametri:

- il Collapsed Gibbs Sampling [Griffiths and Steyvers, 2004]
- il metodo Variational Bayes (batch VB oppure, per un costo computazionale minimo, online VB) [Hoffman et al., 2010]

Entrambi i metodi sono basati su una procedura interpretabile come un processo di Expectation Maximization (EM). Le relative derivazioni vanno al di là degli obiettivi di questa tesi. Si riporta comunque di seguito lo pseudocodice del Collapsed Gibbs Sampling e del Variational Bayes.

Algorithm 2 LDA con Collapsed Gibbs Sampling

Si definisca z_{it} come l'assegnazione del topic al t-esimo token dell' i-esimo documento. Si definisca n_{kw} come il numero di volte che la parola w viene assegnata al topic k, e n_{kd} come il numero di volte che il topic k viene assegnato a una parola del documento d.
- for loop:
- step E:
Si calcola

$$(z_{it} = k|\mathbf{z}_{-i}, w_i, d_i) \propto \varphi_{kw}\theta_{kd} \tag{20}$$

e si assegna z_{it} al topic con verosimiglianza maggiore.
- step M:
Si stimano

$$\varphi_{kw} = \frac{n_{kw} + \beta}{\sum_{k'}(n_{k'w} + \beta)} \tag{21}$$

$$\theta_{kd} = \frac{n_{kd} + \alpha}{\sum_{k'}(n_{k'd} + \alpha)} \tag{22}$$

- end loop

Algorithm 3 LDA con Variational Bayes

Sono dati la Matrice documento-termine n_{dw} e gli iperparametri α, β. Si consideri l'approssimazione variazionale mean-field:

$$q(z, \theta, \varphi) = q(z|\phi)\, q(\theta|\gamma)\, q(\varphi|\lambda),$$

dove q approssima la distribuzione a posteriori del modello. Si indichi con $\Psi(\cdot)$ la funzione Digamma.

L'algoritmo seguente massimizza l'ELBO tramite coordinate ascent.

- Inizializza $\lambda_{kw} > 0 \quad \forall k, w$
- Ripeti fino a convergenza:
- **E step** (per ogni documento d):
- Inizializza $\gamma_{dk} > 0 \quad \forall k$
- Ripeti fino a convergenza locale:

$$\mathbb{E}_q[\log \theta_{dk}] = \Psi(\gamma_{dk}) - \Psi\left(\sum_{k'=1}^{K} \gamma_{dk'}\right) \tag{23}$$

$$\mathbb{E}_q[\log \varphi_{kw}] = \Psi(\lambda_{kw}) - \Psi\left(\sum_{w'=1}^{W} \lambda_{kw'}\right) \tag{24}$$

$$\phi_{dwk} \propto \exp(\mathbb{E}_q[\log \theta_{dk}] + \mathbb{E}_q[\log \varphi_{kw}]) \tag{25}$$

$$\gamma_{dk} = \alpha + \sum_{w} n_{dw}\, \phi_{dwk} \tag{26}$$

- normalizza ϕ_{dwk} rispetto a k
- end loop
- **M step**:

$$\lambda_{kw} = \beta + \sum_{d} n_{dw}\, \phi_{dwk} \tag{27}$$

- end loop

Stime puntuali dei parametri:

$$\hat{\theta}_{dk} = \frac{\gamma_{dk}}{\sum_{k'=1}^{K} \gamma_{dk'}} \tag{28}$$

$$\hat{\varphi}_{kw} = \frac{\lambda_{kw}}{\sum_{w'=1}^{W} \lambda_{kw'}} \tag{29}$$

2.9 NTM: Neural Topic Models

Le reti neurali sono una tanto giovane quanto flessibile e vasta classe di modelli statistici.

La loro applicazione ai dati di testo è ampiamente discussa, e ha portato al invenzioni come LLM, traduttori automatici, ai migliori classificatori di sentiment analsys e a molto altro. La loro applicazione nel campo del topic modeling non è quindi mancata, e ha dato origine alla generica e macroscopica categoria dei Neural Topic Models, o NTM [Wu et al., 2024]. Tuttavia le reti neurali dovrebbero a loro volta essere distinte in numerose famiglie. Ai fini degli sviluppi che hanno portato nel topic modeling, è utile evidenziarne appena 4 tipi: i modelli basati sulle RBM, i VAE, i modelli ricorrenti e i BERT.

In questo lavoro ci si concentra sulla classe delle RBM, che sebbene siano ad oggi le meno popolari sono state le prime NTM a trovare applicazione come topic models. Data la semplicità della loro struttura, presentano costo computazionale ridotto rispetto agli NTM più complessi, come i BERT. Bisogna però riconoscere che le performance migliori in questo task sono contese tra i VAE e i BERT, e che la rapida evoluzione di questi modelli possa portare a ulteriori miglioramenti del topic modeling. Tra i VAE per il topic modeling, hanno ottenuto performance dello stato dell'arte il modello ProdLDA, il modello NVLDA, entrambi NTM basati sulla LDA, e il modello AVITM, una combinazione del ProdLDA e del NVLDA. Tali modelli sono stati inclusi dagli autori stessi in OCTIS.

3 OCTIS: un framework per il topic modeling

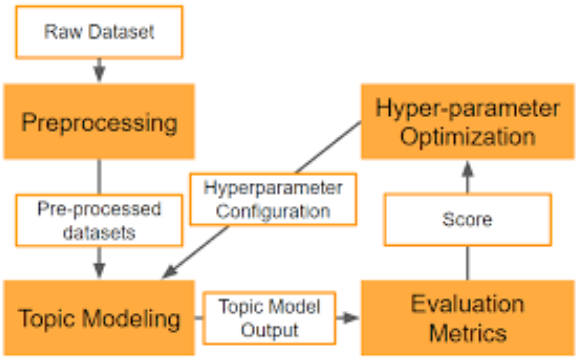


Figure 7: Flusso di lavoro generico per lo sviluppo di un topic model con Octis

OCTIS (Optimizing and Comparing Topic models Is Simple) [Terragni et al., 2021] [Terragni et al., 202

è un framework pensato per gestire l'addestramento, l'analisi, l'ottimizzazione e , in particolare, il confronto, di diversi topic models. Si tratta di una libreria Python, con codice accessibile al link: https://github.com/MIND-Lab/OCTIS. Il progetto è stato avviato, promosso e supervisionato dal 2021 dagli autori del paper omonimo.

Il framework di OCTIS presenta una serie di vantaggi, rispetto ad altre comuni librerie per il topic modeling:

- include un'ampia gamma di topic models, dal modello NMF alla LDA e ad alcuni tra gli NTM più recenti.
- consente di automatizzare l'intero flusso di analisi dati, dal preprocessing alla validazione dei modelli.
- consente di comparare diversi topic models sulla basi solide e criteri riconosciuti, in modo tale da dare ai ricercatori uno strumento di garanzia dei risultati ottenuti. I dataset scelti, le modalità di preprocessing, e le metriche di validazione sono infatti poco flessibili e basate su prassi consolidate, e consentono quindi una conduzione degli esperimenti robusta.
- offre degli strumenti di ottimizzazione bayesiana, estremamente utile per scegliere in modo efficiente la combinazione ottima degli iperparametri di ogni topic model, specialmente dove il numero di iperparametri e delle loro possibili configurazioni è alto [Terragni and Fersini, 2021].

Il repository associato al progetto è stato aggiornato dai membri di una community di utenti, ricercatori e non, al fine di migliorarne le prestazioni e includere nuovi topic models. Silvia Terragni continua a gestire tali aggiornamenti, e grazie al suo contributo è stato possibile includere i modelli RSM e Over RSM alla libreria.

Figure 8: logo del package di Octis

4 Metriche di validazione per i topic models

Il topic modeling è un task non supervisionato. Pertanto la validazione di questa classe di modelli è particolarmente difficile, in quanto tipicamente non sono presenti dati di esempio sulle reali distribuzioni dei topic per valutare la correttezza delle risposte fornite dal modello. Le metriche utilizzate si distiguono in due categorie: quelle applicabili a modelli probabilistici di qualsiasi natura (come la log verosimiglianza, la perplexity o la divergenza di Kullback-Leibler, e le metriche di classificazione per modelli supervisionati) e quelli sviluppate specificatamente per problemi di topic modeling (come le metriche di Coherence e la topic diversity). Ogni metrica può essere calcolata su un corpus specifico. Per qualsiasi metrica, è importate valutare un modello su un corpus differente da quello di training, in modo tale da evitare una sottostima delle funzioni di costo a causa di overfitting.

OCTIS dispone di 5 classi di metriche di validazione. In questa tesi vengono escluse due classi, nonché diverse metriche basate sull'adozione di word embeddings. In particolare, sono trattate alcune tra le metriche di classificazione, topic diversity e coherence. Viene trattata la perplexity, non inclusa in OCTIS, ma utilizzata da Hinton nella valutazione del modello RSM.

4.1 perplexity

La perplexity presenta molteplici definizioni, a volte estremamente differenti. Tali differenze dipendono dalla natura del modello che viene valutato con essa. Per tutti i modelli probabilistici, la perplexity si può definire come:

$$ppl(X) = \exp\left(-\frac{\sum_{i=1}^{N} \ln(p(x_i))}{N}\right) \tag{30}$$

Che equivale alla definizione della media geometrica della verosimiglianza:

$$ppl(X) = \left(\prod_{i=1}^{N} p(x_i)\right)^{-\frac{1}{N}} \tag{31}$$

Si nota facilmente che solo la prima definizione prevede un calcolo computer-feasible, mentre l'interpretazione è basata sulla seconda.

Si nota anche che la log perplexity coincide con l'entropia, ovvero con la cross-entropia tra la distribuzione del modello e se stessa:

$$\ln(ppl(X)) = -\frac{\sum_{i=1}^{N} \ln(p(x_i))}{N} = H(p;p) = H(p) + KL(p||p) \tag{32}$$

dove

$$H(p;q) = H(p) + KL(p||q) = -E_p[\log(q)] \tag{33}$$

$$KL(p||q) = \sum_x p(x) \ln\left(\frac{p(x)}{q(x)}\right) = \sum_x p(x) \ln(p(x)) - \sum_x p(x) \ln(q(x)) \tag{34}$$

è la divergenza di Kullback-Leibler, la quale va a 0 per p=q, per cui $KL(p||p) = 0$ e $H(p;p) = H(p) = \ln(ppl(X))$

$$H(p) = -E_p[\log(p)] = -\frac{\sum_{i=1}^{N} \ln(p(x_i))}{N} = \ln(ppl(X)) \tag{35}$$

Ovvero la perplexity si può interpretare come l'entropia della distribuzione del modello, anche detta Entropia di Shannon. Per topic models che presentano l'assunzione BoW, la perplexity si definisce in generale come:

$$ppl(D) = \exp\left(-\frac{\sum_{d=1}^{|D|} \sum_{i=1}^{N_d} p(\mathrm{w}_i|W_d)}{\sum_{d=1}^{|D|} N_d}\right) \tag{36}$$

Dove D è un corpus, N_d è la dimensione del d-esimo documento W_d che comprende N_d parole, w_i è la i-esima parola che compare nel documento e p è la probabilità condizionata sotto il modello valutato.

Per topic models che non rispettano l'assunzione BoW, ovvero che tengono in consid-

erazione l'ordinamento dei token nei documenti, e per tutti gli altri modelli linguistici, la perplexity per un documento spesso viene definita come:

$$ppl(W) = \exp\left(-\frac{\sum_{i=1}^{t} p(\mathrm{w}_i|\mathrm{w}_{<i})}{t}\right) \tag{37}$$

Esistono definizioni della perplexity che utilizzano 2 come base del logaritmo. Tuttavia la definizione più ricorrente in materia di topic modeling è quella basata sull'adozione del logaritmo naturale. Il valore della perplexity è invariante alla scelta della base, mentre varia il valore dell'Entropia di Shannon.

Per diversi modelli probabilistici, e in particolare per quelli appartenenti alla famiglia delle RBM, è spesso difficile calcolare la verosimiglianza esatta dei dati, a causa della presenza di funzioni di partizione difficili da calcolare. In alcuni casi è comunque possibile calcolare la perplexity esatta attraverso approssimazioni numeriche quali l'Annealed Importance Sampling, discusso per il modello RSM nella sezione 6.4. Tali approssimazioni purtroppo presentano un elevato costo computazionale, e facilmente portano a stime distorte. Pertanto, per questi modelli, ove possibile ci si limita all'adozione di un upper bound della perplexity, o non la si usa affatto.

Pur trattandosi di una metrica molto popolare per altri modelli linguistici, non è la più usata in materia di topic modeling. OCTIS non include la perplexity tra le sue metriche di validazione.

Affidandosi a questa metrica, e utilizzando l'algoritmo AIS, Hinton e Salakhutinov nel 2009 dimostrano che il modello RSM presenta risultati migliori rispetto a quelli del modello LDA [Hinton and Salakhutdinov, 2009].

4.2 Topic Diversity

La topic diversity è stata adottata la prima volta da [Dieng et al., 2020], i quali la descrivono e la utilizzano in questo modo:

> We combine coherence with a second metric, topic diversity. We define topic diversity to be the percentage of unique words in the top 25 words of all topics. Diversity close to 0 indicates redundant topics; diversity close to 1 indicates more

varied topics. We define the overall quality of a model's topics as the product of its topic diversity and topic coherence

Si supponga di aver addestrato un topic model in grado di identificare T topics, su un vocalbolario di V parole/token. Si supponga di disporre di un criterio di ordinamento dei contributi che ciascuna parola fornisce a ciascun topic attraverso il modello (ad esempio una matrice di probabilità delle parole dati i topic). Allora, selezionate per ciascun topic le top k parole più rilevanti, con k arbitrario (usualmente 10 o 25, a seconda della dimensione del vocabolario) la Topic Diversity si definisce come segue:

$$\text{TD} = \frac{|\text{unique(topk)}|}{k \cdot T} = \frac{|\text{unique(topk)}|}{|\text{topk}|} \tag{38}$$

dove topk rappresenta la lista delle T liste delle k parole più importanti per ciascun topic, mentre unique(topk) la lista delle parole che compaiono al suo interno. Il rapporto tra il numero di parole comprese nelle due liste, rappresenta la percentuale di parole uniche tra le prime topk. Per ogni topic ogni parola può infatti presentarsi una volta sola, ragione per cui i duplicati sono necessariamente doppie assegnazioni della stessa parola a topic diversi. Un topic model con buone performance dovrebbe evitare di assegnare alti score da più topic a una stessa parola, e di conseguenza dovrebbe evitare di presentare delle parole duplicate sulle prime righe della matrice che ordina per tutti i topic tutte le parole del vocabolario in base a una misura di associazione.

4.3 Coherence

Le metriche di Topic Coherence misurano la coerenza delle rappresentazioni delle parole tematiche per valutare l'interpretabilità e la significatività di un argomento. La maggior parte delle misure di coerenza si basa sull'analisi delle distribuzioni di co-occorrenza delle parole all'interno dei documenti. Un valore elevato di una metrica di topic coherence indica un elevato grado di correlazione tra le parole appartenenti al medesimo topic. Metriche di questa specie sono C_V, UMass, UCI e NPMI.

Metrica C_V

$$C_V(t, V_t) = \frac{1}{N} \sum_{i=2}^{N} \sum_{j=1}^{i-1} \log \frac{D(v_{it}, v_{jt}) + 1}{D(v_{jt})} \tag{39}$$

dove:

- $V_t = \{v_{1t}, v_{2t}, \ldots, v_{Nt}\}$ sono le top-k parole del topic t
- $D(v_{it}, v_{jt})$ è il numero di documenti che contengono entrambe le parole v_{it} e v_{jt}
- $D(v_{jt})$ è il numero di documenti che contengono la parola v_{jt}

è da notare che lo stesso autore di questa metrica ne sconsigli l'uso.

Metrica UMass

$$UMass(t, V_t) = \frac{1}{N(N-1)} \sum_{i=2}^{N} \sum_{j=1}^{i-1} \log \frac{D(v_{it}, v_{jt}) + 1}{D(v_{jt})} \tag{40}$$

La formula è molto simile a C_V, ma con una normalizzazione diversa.

Metrica UCI

$$UCI(t, V_t) = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \log \frac{P(v_{it}, v_{jt}) + \epsilon}{P(v_{it}) \cdot P(v_{jt})} \tag{41}$$

dove:

- $P(v_{it}, v_{jt})$ è la probabilità congiunta delle parole v_{it} e v_{jt}
- $P(v_{it})$ è la probabilità della parola v_{it}
- ϵ è un numero piccolo per evitare il logaritmo di zero

Metrica NPMI (Normalized Pointwise Mutual Information)

$$NPMI(t, V_t) = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \frac{\log \frac{P(v_{it}, v_{jt})}{P(v_{it}) \cdot P(v_{jt})}}{-\log P(v_{it}, v_{jt})} \tag{42}$$

La NPMI normalizza la PMI (Pointwise Mutual Information) per ottenere valori compresi tra -1 e 1, dove valori più alti indicano maggiore coerenza. Si ritiene che la NPMI sia la metrica di validazione in generale più vicina al giudizio umano sulla bontà della costruzione dei topic.

4.4 Metriche di classificazione

Le metriche di classificazione sono metriche per modelli supervisionati. Pertanto per essere valutate richiedono che i documenti appartengano a delle classi specifiche, disponibili nei dati, come ad esempio dei tag associati ai documenti. Se tali informazioni aggiuntive non sono disponibili nel dataset, le metriche di classificazione non sono utilizzabili. Un topic model è un modello non supervisionato, e in quanto tale, per costruzione, non può prevedere delle etichette date dall'esterno. Ciò che però è possibile fare è far prevedere la distribuzione del documento sui topic dal topic model, e fornire questa previsione come una rappresentazione sintetica e di dimensione fissa da fornire come input a un classificatore supervisionato che dovrebbe prevedere le etichette presenti nei dati. Migliori saranno le capacità previsive del classificatore, migliori saranno le rappresentazioni dei documenti fornite dai topic model. Modelli supervisionati impiegati a tal fine sono tipicamente le Support Vector Machines (SVM).

Le metriche di classificazione tipicamente utilizzate sono Accuracy, Recall, Precision e F1 score (media geometrica di Precision e Recall), le quali si definiscono come segue per un problema di classificazione binaria:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{43}$$

$$precision = \frac{TP}{TP + FP} \tag{44}$$

$$recall = \frac{TP}{TP + FN} \tag{45}$$

$$\text{F1} = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{46}$$

Dove, data una singola classe binaria prevedere, che può assumere valore vero-falso o 0-1, TP, TN, FP, FN sono rispettivamente i veri positivi, i veri negativi, i falsi positivi, i falsi negativi.

In un problema di classificazione multi-classe, dove la matrice di confusione ha dimensione kxk con k>2, tali definizioni sono da rivedere. L'accuracy mantiene la stessa formula, se si considerano TP, TN, FP, FN come le somme su j, con j indice della classe, $j = 1, \ldots, k$ dei valori TP_j, TN_j, FP_j, FN_j. Più semplicemente, data y_i la classe appartenente all'osservazione i-esima, con $i = 1, \ldots, N$ e $\hat{y}_i$ la classe prevista dal modello, vale:

$$accuracy = \frac{\{\#i : y_i = \hat{y}_i\}}{N} = \frac{\sum_{i=1}^{N} \mathbb{I}(y_i = \hat{y}_i)}{N} \tag{47}$$

Per precision e recall invece non è possibile estendere direttamente il caso binario a quello multi-classe. Esistono due versioni della loro estensione al caso multi-classe: macro-averaging e micro-averaging.

La definizione macro-averaging è semplicemente la media delle precision e recall per classe:

$$precision_{macro} = \frac{\sum_{j=1}^{k} precision_j}{k} \tag{48}$$

$$recall_{macro} = \frac{\sum_{j=1}^{k} recall_j}{k} \tag{49}$$

Tuttavia tale definizione non tiene conto della diversa numerosità delle classi. Pertanto esiste la versione micro-averaging:

$$precision_{micro} = \frac{\sum_{j=1}^{k} TP_j}{\sum_{j=1}^{k} TP_j + FP_j} \tag{50}$$

$$recall_{micro} = \frac{\sum_{j=1}^{k} TP_j}{\sum_{j=1}^{k} TP_j + FN_j} \tag{51}$$

La versione micro-averaging risulta molto più usata, in quanto pesa differentemente classi di diversa numerosità. La formulazione dello score F1 nel caso multi-classe coincide con quella del caso binario, scelta la modalità di calcolo di precision e recall.

In alcuni lavori si è proposto di introdurre anche metriche di regressione, seguendo lo stesso principio e adottando dei modelli di regressione. In tale caso i dati associati ai documenti dovrebbero consistere in numeri reali o interi, come ad esempio la media delle valutazioni date dagli utenti a un determinato articolo di un giornale online, o il numero di visualizzazioni che ha ricevuto. Tuttavia sono rari i dati di corpus comprensivi di queste informazioni, e non è sempre verosimile che una cattiva performance dei modelli di regressione su di essi implichi una cattiva rappresentazione dei topic.

5 Markov Networks

Esistono numerosi esempi di dati con strutture particolarmente complesse inerenti fenomeni molto comuni ma difficili da rappresentare. Esempi possono essere le immagini, i dati di testo, i grafi o particolari tipi di serie storiche. Si tratta di processi stocastici che comprendono numerose variabili aleatorie con legami di dipendenza difficili da modellare. A volte è possibile rappresentare tali processi attraverso dei modelli grafici parametrizzati, ovvero come delle "reti" di variabili aleatorie. Sono detti graphical models tutti quei modelli statistici che mirano a rappresentare attraverso una precisa struttura un numero finito di variabili aleatorie. I graphical models si dividono in due grandi famiglie: i modelli grafici diretti e i modelli grafici indiretti. Dei primi fanno parte principalemente le reti bayesiane, che consentono di utilizzare distribuzioni arbitrarie sulle variabili appartenenti al processo. I secondi sono invece quasi sempre rappresentati da modelli basati sull'energia, spesso facenti parte della famiglia dei Markov Random Fields, anche detti Markov Networks o Gibbs Random Fields, modelli vincolati all'utilizzo della distribuzione di Gibbs per descrivere il processo stocastico nel suo complesso.

Nei modelli diretti è sempre descritto un nesso di causalità tra le variabili facenti parte del processo, mentre nei modelli indiretti è possibile rilassare tale assunzione e garantire maggior flessibilità al modello. Spesso tale guadagno in flessibilità si traduce in un maggior costo computazionale per i modelli indiretti rispetto a quelli diretti, a parità di numero di parametri. Allo stesso tempo, tuttavia, la flessibilità dei modelli indiretti consente di descrivere più facilmente, e dunque con un minor numero di parametri, molti fenomeni presenti in natura, mentre i modelli bayesiani spesso risultano molto complessi al fine di rendersi utili, e tale complessità si traduce in un elevato numero di parametri. Pertanto, non è possibile in generale affermare quali fra i modelli bayesiani e i modelli basati sull'energia siano computazionalmente più efficienti, ma è possibile affermare che, fissato il numero di parametri, la procedura di training delle reti bayesiane sia in generale più leggera dal punto di vista computazionale, data la maggiore trattabilità della funzione di verosimiglianza. Inoltre, mentre le reti bayesiane risultano in generale più interpretabili rispetto ai modelli energy based, la difficoltà nel definirle è spesso superiore, in quanto dipende dalla complessità delle distribuzioni

utilizzate. La distribuzione di Boltzmann invece, per quanto implichi la presenza di molte costanti impossibili da calcolare, risulta facilmente personalizzabile, data la linearità delle funzioni per l'energia, discusse di seguito.

Gran parte degli strumenti illustrati in questa sezione provengono dal mondo della fisica, in particolare della meccanica statistica, e sono stati poi elevati a modelli statistici.

5.1 Parentesi terminologica della teoria dei grafi

Si descrivono brevemente alcuni termini tecnici utili in materia di analisi dei grafi, rappresentazioni matematiche e grafiche ripetutamente utilizzate nell'ambito delle reti neurali e dei modelli grafici probabilistici.

Si definisce grafo o modello grafico una coppia G = (V, E) dove V rappresenta l'insieme dei nodi, o vertici, del grafo, ed E l'inisieme degli archi, o collegamenti, che li uniscono.

Un grafo si dice diretto quando i suoi archi presentano una direzione, mentre è indiretto nel caso opposto.

Si dice sottografo ogni sottoinsieme di archi e nodi di un grafo originario.

Un grafo si dice ciclico quando, fissato un nodo di partenza, è possibile ritornarvi senza retrocedere all'ultimo arco percorso, mentre è detto aciclico altrimenti. Si dice ciclo di un grafo ogni un suo sottografo ciclico.

Si dice chordal un grafo che presenta in tutti i cicli con 4 o più vertici un arco che collega due vertici appartenenti al ciclo e che non appartiene al ciclo (e tale arco viene definito chord).

Si dice clique di ordine k di un grafo ogni suo sottografo che presenta tutti i nodi fra loro connessi, ovvero una clique è un grafo denso. Nulla vieta a più clique si sovrapporsi fra loro, ogni nodo è una clique di ordine 1 e ogni arco è una clique di ordine 2.

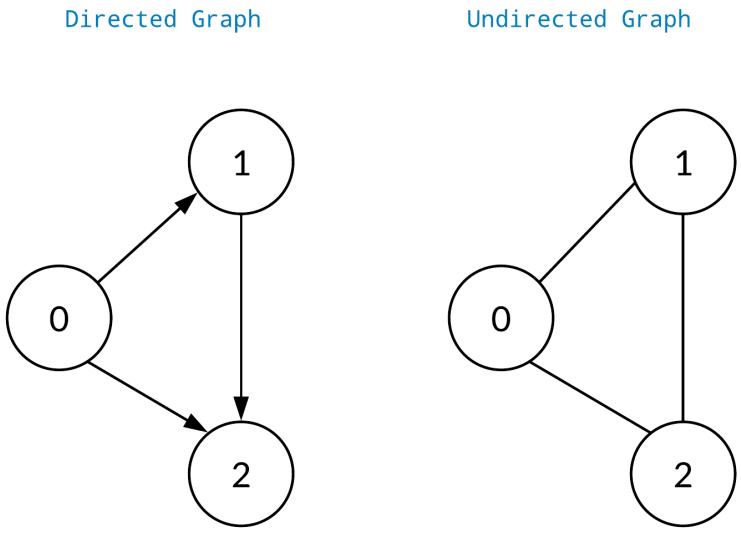


Figure 9: Differenza tra grafo diretto e indiretto
Nel grafo indiretto non esiste un rapporto di causalità, per cui un grafo indiretto è sempre ciclico.

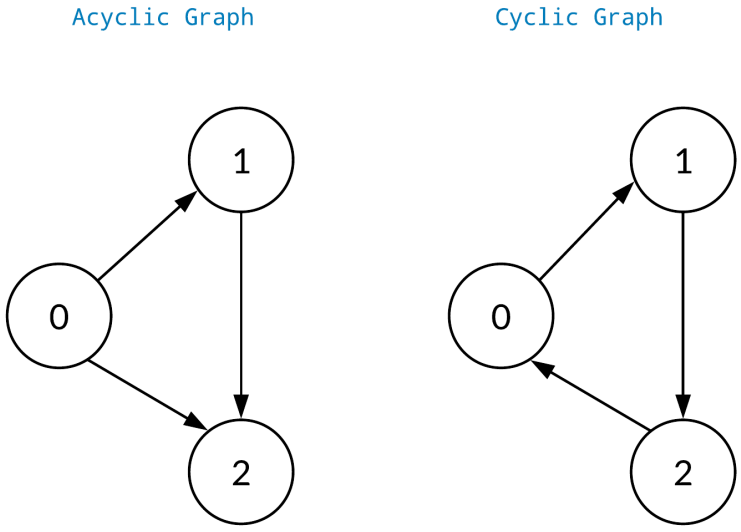


Figure 10: Differenza tra grafo diretto ciclico e aciclico (DAG)

5.2 La Misura di Gibbs

La distribuzione di Boltzmann è una generalizzazione della distribuzione di Maxwell-Boltzmann ed è genitore della Misura di Gibbs, ripetutamente utilizzata nei modelli energy based. Si tratta di tre distribuzioni ereditate dalla fisica e in particolare dalla statistica meccanica. Molti degli algoritmi intitolati a Boltzmann, come le RBM, sono in realtà basati sulla Distribuzione di Gibbs. Tali distribuzioni descrivono l'andamento di un livello di energia dato in un particolare sistema. Le regole del sistema fisico sono incorporate in una funzione, detta funzione di energia. In molti problemi di natura fisica, come il problema degli spin-glass o i problemi di previsione del movimento delle particelle di un particolare gas in una camera pressurizzata, si ricorre a una di queste distribuzioni, o direttamente a una funzione di energia. Sia dato un sistema fisico, che può essere rappresentato come un insieme di valori numerici legati da particolari relazioni funzionali. Ogni possibile combinazione di valori fissata rappresenta una precisa configurazione $s \in S$, o stato, che il sistema può assumere. la soluzione a un dato problema rappresenta una particolare configurazione del sistema cui corrisponde spesso in un punto di minimo su una particolare funzione di energia, descritta sulla base del sistema fisico in esame.

In particolare, la densità della distribuzione di Maxwell-Boltzmann è utilizzata per descrivere la velocità delle particelle di gas v:

$$f_{mb}(v) = \sqrt{\frac{2}{\pi}} \frac{v^2}{a^3} \exp\left(\frac{-v^2}{2a^2}\right) \tag{52}$$

dove a è un parametro positivo e v è la velocità della particella. Tale distribuzione descrive la probabilità che una data particella si muova a una data velocità. Tale distribuzione è imparentata con la distribuzione Chi-quadro con
La densità della distribuzione di Boltzmann si descrive invece come:

$$f_b(\varepsilon) = \frac{\exp\left(-\frac{\varepsilon}{k_b T}\right)}{Z} \propto \exp\left(-\frac{\varepsilon}{k_b T}\right) \tag{53}$$

dove, T è la temperatura, k_b è la costante di Bolzmann (omessa nella definizione della Misura di Gibbs), e Z è una somma di esponenziali su tutti i livelli di energia ε_i associati a

tutte le possibili configurazioni $s_i \in S$:

$$Z = \sum_i \exp\left(-\frac{\varepsilon_i}{k_b T}\right) \tag{54}$$

In genere, questa constante rappresenta una funzione di partizione impossibile da calcolare, a meno di un numero di possibili configurazioni del sistema molto ridotto.
L'interpretazione di questi valori in un sistema di particelle in movimento è la seguente:

$$p_i = f_b(\varepsilon_i) = \frac{N_i}{N} \tag{55}$$

ovvero la densità (che è anche una probabilità) della distribuzione di Boltzmann corrisponde alla percentuale di particelle che si trovano nello stato del sistema associato al livello di energia ε_i, dove N è il numero di particelle presenti.

L'odds di tali probabilità si traduce facilmente in una differenza tra livelli di energia:

$$\frac{p_i}{p_j} = \exp\left(\frac{\varepsilon_i - \varepsilon_j}{k_b T}\right) \tag{56}$$

La Misura di Gibbs presenta una pdf identica a quella della Distribuzione di Boltzmann, con la differenza che la costante di Boltzmann (che in quanto costante fisica non rappresenta un parametro della distribuzione, ma un numero fissato, $k_b = 1.38064910^{-23}$ J/K con J Joule e K Kelvin) viene posta pari a 1, o incorporata dentro la temperatura T, e al posto della temperatura viene inserita la temperatura inversa $\beta = \frac{1}{T}$.

$$f_g(\varepsilon_i) = \frac{\exp(-\beta\varepsilon_i)}{Z} \propto \exp(-\beta\varepsilon_i) \tag{57}$$

$$Z = \sum_i \exp(-\beta\varepsilon_i) \tag{58}$$

e si vede che continuano a valere tutte le proprietà dette prima. Si deriva naturalmente che mentre prima valeva $T > 0$ ora vale $\beta \in (0, 1]$

Ciò che si vuole fare, data una particolare funzione $E(s), E : S \to \mathbb{R}$, detta funzione di energia, è trovare $s^* = argmin_s E(s)$. Per questa ragione il problema di massimizzazione della

verosimiglianza su un modello basato sulla distribuzione di Boltzmann si traduce sempre in un problema di minimizzazione dell'energia.

La distribuzione di Boltzmann dipende dal parametro positivo della temperatura, o della temperatura inversa, che rappresenta il rumore di fondo del processo stocastico. Al crescere della temperatura, il livello dell'energia associato a una particolare configurazione del sistema fisico impatta sempre meno sulla probabilità che il sistema assuma quella configurazione. è interessante notare la stretta analogia tra la densità della distribuzione di Boltzmann e quella della Gaussiana. Si vede infatti che l'argomento dell'esponenziale nella prima è l'energia, mentre nella seconda è il quadrato della differenza dalla media, mentre il denominatore nella prima è la temperatura, mentre nella seconda è la varianza. Si nota inoltre che entrambe le distribuzioni appartengono alla famiglia esponenziale.
Si vede che per $T \to \inf$ la distribuzione di Boltzmann approssima l'uniforme. Sulla nozione di temperatura è associato l'algoritmo di ottimizzazione del Simulated Annealing, che altro non è che l'applicazione dell'algoritmo Metropolis-Hastings alla distribuzione di Boltzmann, dove la funzione di perdita da minimizzare prende il posto del valore dell'energia.
Di seguito si trattano i nomi della distribuzione di Boltzmann e della Misura di Gibbs come intercambiabili, in quanto la differenza in ambito statistico è concettualmente trascurabile, ma di fatto si fa sempre riferimento alla Misura di Gibbs.

5.3 funzioni di energia

Le funzioni di energia sono spesso associate a modelli log lineari. Vengono infatti sempre inserite all'interno di un'esponenziale, che garantisce la positività e consente di ricavarne delle probabilità. Attraverso queste funzioni, è possibile rappresentare gli archi di un grafo, assegnando dei pesi alle variabili aleatorie che si vogliono tenere in considerazione. Pe un modello grafico senza connessioni tra i suoi nodi, ovvero senza alcuna interazione tra le variabili che compongono il processo stocastico, la funzione di energia può essere modellata come un modello lineare. Per un modello grafico dove sono presenti delle connessioni tra le variabili aleatorie, è frequente l'assegnazione di pesi di interazione simmetrica tra le variabili, come proposto nel modello di Ising.

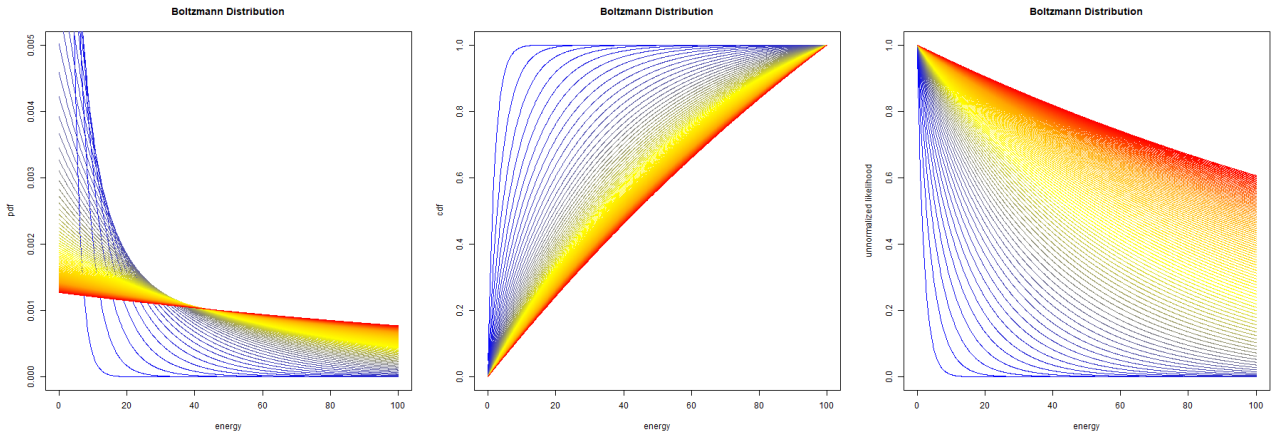


Figure 11: Distribuzione di Boltzmann su valori dati
da sinistra verso destra, rispettivamente, si vedono la pdf, la cdf e la pdf non normalizzata di una distribuzione di Boltzmann su valori dell'energia simulati. Colori caldi corrispondono a livelli di temperatura più elevata, per T = 1,..., 200. Si vede che la pdf e la cdf della Misura di Gibbs approssimano l'uniforme per valori della temperatura elevati. I valori dell'energia simulati sono tutti i numeri da 0 a 100 con passo 0.1. Il cambio del passo e del massimo condiziona la scala della pdf, mentre non incide sulla scala della cdf e della pdf non normalizzata. I grafici sono realizzati in ambiente R.

5.4 undirected graphical models: MRF

Un MRF (Markov Random Field) anche detto Rete di Markov, può essere descritto e rappresentato come un grafo G = (V, E) dove V è l'insieme dei nodi del grafo, ciascuno rappresentante una variabile aleatoria, e E è l'insieme dei degli archi rappresentanti le connessioni fra di esse. In un grafo, dati due nodi $A, B \in V$, A e B sono detti vicini se e solo se esiste un arco in E che li collega. L'assenza di un arco tra due nodi all'interno di un modello così descritto implica indipendenza condizionata tra le due variabili aleatorie corrispondenti appartenenti al processo dato il resto del grafo. In altre parole, un MRF rispetta la Proprietà Locale di Markov, in base alla quale ogni nodo è indipendente da ogni altro nodo (che non è suo vicino) dati tutti i suoi vicini.

La principale differenza tra le reti bayesiane e i MRF risiede nel fatto che mentre le prime sono modelli DAG, ovvero grafi diretti e aciclici, i secondi sono modelli grafici indiretti. Pertanto mentre le reti bayesiane possono descrivere delle gerarchie e dei rapporti di causalità tra le variabili aleatorie appartenenti al processo, dai MRF tale informazione non è ricavabile. Per la stessa ragione, a parità di numero di archi contenuti in E, un MRF presenta sempre

sia una maggiore flessibilità nella descrizione del processo in esame, sia un maggior costo computazionale.

Un MRF è sempre un GRF (Gibbs random field) se almeno una delle due seguenti condizioni è soddisfatta:

- il grafo è chordal

- la densità del grafo è strettamente positiva, per il Teorema di Hammersley e Clifford

5.5 Gibbs random field e Hammersley-Clifford Theorem

Dal 1971, con la pubblicazione del Teorema Hammersley-Clifford, è stata dimostrata l'equivalenza tra le Markov Networks e i Gibbs random fields. Per tale ragione le Reti di Markov sono spesso denominate anche MRF (Markov Random Field).

Un Gibbs random field rappresenta una rete di Markov la cui densità è espressa attraverso la distribuzione di Bolzmann, e alla cui struttura si associa una funzione di energia. Esempi di Gibbs random field sono le reti di Hopfield, le macchine di Bolzmann e le macchine di Bolzmann ristrette.

In base al Teorema di Hammersley e Clifford, data una qualsiasi rete di Markov, se tale rete presenta una verosimiglianza positiva, allora la sua densità è fattorizzabile come segue:

$$P(G) = \prod_{c \in C(G)} f_c(\mathbf{v}_c) \tag{59}$$

dove G è la rete di Markov, ovvero un grafo indiretto di variabili aleatore, $C(G)$ rappresenta l'insieme delle clique contenute in G, $\mathbf{v}_c$ è l'insieme delle variabili corrispondenti ai nodi presenti nella clique c, e f_c è detto clique potential, ovvero è una funzione delle variabili comprese nella clique.

Ciò che il Teorema di Hammersley-Clifford afferma, in particolare, è che data la densità positiva di un grafo G di variabili aleatorie, e data una specifica variabile x, allora è possibile fattorizzare la densità di G come il prodotto tra una funzione della sola x e dei suoi nodi vicini e di una funzione che esclude x, ovvero

$$Pr(U) = f(x, \mathbf{v}) = f_x(x, \partial x) f_{\setminus x}(\mathbf{v}) \tag{60}$$

Si può osservare che l'applicazione iterata della (35) all'interno di un grafo qualsiasi conduce direttamente alla rappresentazione dell'eq.(34). Si può facilmente verificare sempre attraverso la (35) che ogni MRF rispetta la proprietà di Markov locale.

Si è già fornita in precedenza la densità della distribuzione di Gibbs, spesso denominata anche misura di Gibbs:

$$P(X = x) = \frac{1}{Z(\beta)} \exp(-\beta E(x)) \tag{61}$$

Dove E è una data funzione di energia, X è uno stato, x una configurazione dello stato, β una temperatura inversa e Z una costante di normalizzazione, anche detta funzione di partizione.

Tutti i MRF possono presentare una distribuzione appartenente alla famiglia esponenziale e basata sulla misura di Gibbs, ovvero la densità di un qualsiasi MRF è rappresentabile come:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_k \mathbf{w}^{\mathrm{T}} f_k(x_k)\right) \tag{62}$$

dove

$$\mathbf{w}^{\mathrm{T}} f_k(x_k) = \sum_{i=1}^{N_k} w_{ik} f_{ik}(x_k) \tag{63}$$

dove molto spesso, come nel caso del modello di Ising, le funzioni f_{ik} sono funzioni indicatrici rispetto a una particolare configurazione della clique x_k, e w_{ik} i pesi associati.

Infine bisogna ricordare il ruolo della funzione di partizione Z:

$$Z = \sum_{x \in \chi} \exp\left(\sum_k \mathbf{w}^{\mathrm{T}} f_k(\mathbf{x}_k)\right) = \sum_{x \in \chi} \exp\left(\sum_k \sum_{i=1}^{N_k} w_{ik} f_{ik}(\mathbf{x}_k)\right) \tag{64}$$

Dove χ rappresenta l'insieme di tutte le possibili combinazioni di valori assegnabili al processo stocastico X, k è l'indice di una particolare clique, i è l'indice di una particolare configurazione della clique k, e $f_k(x_k)$ è una funzione indicatrice che vale 1 quando la clique k si trova nella configurazione i, 0 altrimenti. La rappresentazione più comune associa al peso w_{ik} il logaritmo di un clique potential, ovvero $w_{ik} = \ln(g_k(\mathbf{x}_k))$ dove qui g_k rappresenta un clique potential. In questo modo è possibile trattare un MRF come un modello log lineare.

5.6 Reti neurali feedforward e ricorsive

Negli ultimi anni sono emerse due principali tipologie di reti neurali: le reti feedforward, ancora oggi estremamente popolari, utili per problemi di apprendimento supervisionato ma anche come framework per l'addestramento di modelli generativi, e le reti ricorsive, molto meno conosciute, tra le quali sono emerse le RBM. Entrambe le tipologie di reti rappresentano dei modelli grafici nei quali ogni nodo del grafo è un neurone artificiale, ovvero una debole rappresentazione dei neuroni biologici, che interagiscono fra loro attraverso impulsi elettrici trasmessi sulle sinapsi. Allo stesso modo, nelle reti neurali i nodi della rete rappresentano i neuroni, intesi come valori numerici trasformati con una apposita funzione link (come la relu o la logit), mentre gli archi imitano le sinapsi, e consistono in semplici relazioni funzionali presenti tra alcuni neuroni della rete. Sia nelle reti ricorsive sia nelle reti feedforward la struttura della rete come grafo è definita a priori, e i suoi parametri sono ottimizzati attraverso l'algoritmo di discesa del gradiente rispetto a una data funzione di perdita nel corso di un certo numero di iterazioni di training. La differenza tra le reti feedforward e le reti ricorsive è evidente in fase di inferenza. Nelle reti feedforward il processo computazionale è organizzato in layer: ogni neurone appartiene a un layer specifico, e il layer successivo è funzione di uno o più layer precedenti. Il primo layer è l'input della rete, e l'ultimo layer è il layer di output, mentre tutti i layer intermedi sono denominati layer nascosti. Nelle reti ricorsive il processo di previsione è molto più destrutturato, e il processo di previsione cambia a seconda del tipo di rete ricorsiva in esame. Tipicamente, le reti ricorsive fanno più di una previsione per neurone, e migliorano le stime di tutti i neuroni seguendo un processo di convergenza, che facilmente comporta un costo computazionale superiore rispetto a quello delle reti feedforward. Molte reti ricorsive sono sottoclassi dei MRF, e di conseguenza seguono la distribuzione di Gibbs. Tutti i modelli che seguono sono MRF che presentano processi di inferenza e di training caratterizzati da ricorsione.

5.7 Il modello di Ising

Si tratta del primo undirected graphical model basato su funzioni di energia, e ha trovato applicazione in ambito fisico per la soluzione dello spin glass problem (problema dei vetri di spin). Il modello di Ising (o modello di Lenz-Ising), che prende il nome dai fisici Ernst Ising

e Wilhelm Lenz, è un modello matematico del ferromagnetismo in meccanica statistica. Il modello è costituito da variabili discrete che rappresentano i momenti di dipolo magnetico degli "spin" atomici che possono trovarsi in uno di due stati (+1 o -1). Gli spin sono disposti in un grafo, solitamente un reticolo (dove la struttura locale si ripete periodicamente in tutte le direzioni), consentendo a ciascuno spin di interagire con i suoi vicini. Gli spin vicini che concordano hanno un'energia inferiore a quelli che non concordano; il sistema tende all'energia più bassa, ma il calore disturba questa tendenza, creando così la possibilità di diverse fasi strutturali. è interessante notare la relazione tra la Legge di Curie, applicata ai vetri di spin, e l'approssimazione della distribuzione uniforme da parte della ditribuzione di Bolzmann per livelli elevati di temperatura. In base alla Legge di Curie, la magnetizzazione è inversamente proporzionale alla temperatura. Sopra un determinato livello di temperatura, detto Punto di Curie, qualsiasi materiale diventa ferromagnetico.

Nel modello di Ising, ogni nodo (un magnete che può assumere due direzioni) può assumere valore $S_i \in \{1, -1\}$. I nodi si posizionano su un grafo a forma di reticolo (o lattice) dove ciascun nodo presenta dai 2 ai 4 vicini, non connessi fra loro. Di conseguenza, le clique presenti nel grafo sono di ordine 2. La funzione di energia adottata nel modello di Ising è la seguente:

$$E = -\sum_{ij} J_{ij} S_i S_j - B \sum_i S_i \tag{65}$$

dove il primo termine è una somma sui nodi vicini, e il secondo termine è detto campo esterno, nel quale B è un parametro di rumore comune a tutti i nodi e J_{ij} è un peso assegnato all'interazione tra due nodi. L'interazione magnetica (prima sommatoria) tende ad allineare tutti gli atomi in una certa direzione, mentre il rumore termico (seconda sommatoria) tende a perturbare l'ordine.

Tale modello trova applicazioni solo in fisica, ai fini dello studio dei vetri spin, e la sua versione più studiata è quella che impone $B = 0$ e $J_{ij} = 1 \forall ij$.

Il modello è spesso considerato una forma particolare dello Sherrington-Kirckpatrick model, anche se le differenze sostanziali rilevano solo in ambito fisico, e la loro funzione di energia generalizzata è la medesima delle Reti di Hopfield.

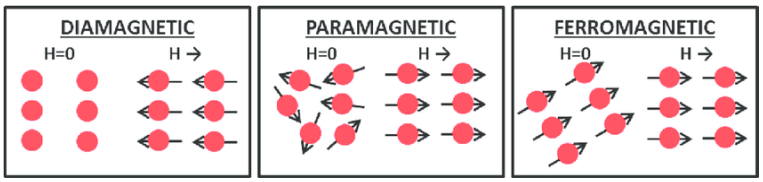


Figure 12: Differenza tra materiali paramagnetici, diamagnetici e ferromagnetici nei materiali ferromagnetici molte particelle puntano nella stessa direzione, e la loro somiglianza porta a una riduzione del livello di energia nel sistema

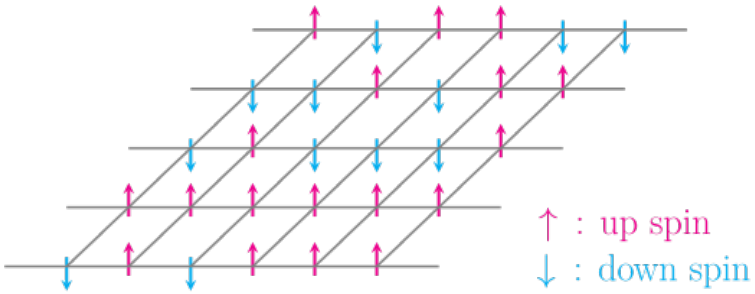


Figure 13: Rappresentazione grafica di uno spin-glass secondo il modello di Ising. Si tratta di un lattice 2D. Ogni nodo può trovarsi in soli due stati, ed è condizionato solo dagli stati dei nodi vicini.

5.8 La regola di Hebb e le Hopfield Network

La prima rete neurale ricorsiva, ovvero il primo MRF finalizzato a imitare i neuroni biologici, è la rete di Hopfield, proposta nel 1982 dal fisico americano John Hopfield [Hopfield, 2007]. Tale rete comprende tanti neuroni quante sono le variabili di input, ragion per cui è classificato come modello non supervisionato, utile per problemi di denoising o di imputazione dei dati mancanti.

Esistono altre applicazioni delle reti di Hopfield, in materia di ottimizzazione (come il famoso Traveling Salesman Problem) o in materia di pattern reconstruction (come denoiser per le immagini).

La rete è tipicamente rappresentabile come un grafo, dove ogni nodo rappresenta una variabile nota o sulla quale si vuole fare inferenza. Particolarità di questa rete è la densità delle connessioni: ogni neurone è collegato a tutti gli altri, eccetto se stesso.

La rete di Hopfield è fortemente ispirata ai principi dell'Hebbian learning, una serie di regole proposte dallo psicologo Donald Hebb nel 1949. Principe cardine delle sue idee sulla neuropsicologia è la Regola di Hebb, che può essere sintetizzata nella massima:

"neurons that fire together, wire together"

ovvero due o più neuroni che si attivano insieme sono collegati. Tale proprietà è incorporata nella funzione di energia del modello. La funzione di energia della rete di Hopfield è praticamente la stessa del modello di Ising, e quindi del modello di Sherrington e Kirckpatrick.

$$E = -\sum_{ij} J_{ij} S_i S_j - \sum_i \theta_i S_i \tag{66}$$

Attraverso la simmetria dei pesi e la natura binaria dei nodi, le reti di Hopfield, incorporano la regola di Hebb nella loro struttura.

Sia dato un problema di memorizzazione delle immagini, e siano dati n pattern di esempio, ciascuno comprendente una griglia di bit 0-1, convertiti nei valori $\{-1, 1\}$. I parametri di interazione della rete di Hopfield, in questo caso, possono essere stimati seguendo la Hebbian Rule:

$$J_{ij} = \frac{1}{n} \sum_{k=1}^{n} S_i k S_j k \tag{67}$$

Una volta stimati i pesi, sempre secondo i principi dell'Hebbian Learning, è possibile fare inferenza. Estratta una specifica variabile S_i tra quelle corrispondenti ai nodi della griglia, la si assegna come:

$$S_i = \begin{cases} 1 & \sum_j J_{ij} S_j \geq U_i \\ -1 & altrimenti \end{cases} \tag{68}$$

Dove U_i è un valore soglia, in genere 0 per ogni i.

Si può osservare che, per reti addestrate, man mano che si aggiungono iterazioni di update l'energia del modello si riduce. L'inferenza della rete non comprende quindi risultati in forma chiusa, ma segue un processo di convergenza che può essere basato su algoritmi come il Gibbs sampling e la Belief propagation.

Si vede dunque che la differenza tra le reti di Hopfield e il modello Sherrington-Kirkpatrick risiede non tanto nella struttura teorica del modello, ma nel dominio applicativo. Il modello Sherrington-Kirkpatrick viene adottato solo in ambito fisico, in particolare per lo studio

dei vetri spin e del ferromagnetismo. Le reti di Hopfield sono invece degli strumenti generalmente applicabili in problemi statistici, in particolare in denoising, pattern recognition e imputazione dei dati mancanti. Mentre nel modello si Sherrington-Kirkpatrick si tiene conto di una diversa vicinanza tra due nodi del grafo, i quali sono connessi solo ai nodi più vicini su una griglia, come i magneti, nel modello di Hopfield il grafo è denso, ovvero tutti i nodi sono potenzialmente collegati fra loro, come i neuroni. Le reti di Hopfield possono essere utilizzate anche come strumenti di ottimizzazione, quando è possibile decodificare la soluzione del problema in esame come una rete di Hopfield, e una relativa funzione di perdita da minimizzare come una funzione di energia. In [Hopfield and Tank, 1985] Hopfield mostra un esempio di risoluzione del Traveling Salesman Problem utilizzando una rete di Hopfield.

Ciò che differenzia profondamente le Hopfield Network dagli altri strumenti di analisi dati è la proprietà della memoria associativa. In base a tale proprietà, che in sostanza implica una forma sfruttamento dell'overfitting del modello, una rete di Hopfield può prendere un pattern che non è compreso nei suoi dati di training e trasformarlo, attraverso un processo di aggiornamento iterato, nel pattern di training più vicino. Pur trattandosi di un modello probabilistico, non è appropriato definire la rete di Hopfield come un modello generativo, in quanto la rete conduce esattamente al pattern più vicino tra quelli mostrati in fase di addestramento.

5.9 Macchine di Boltzmann

Come per il modello di Ising, il modello Sherrington-kirkpatrick, e le reti di Hopfield, le macchine di Boltzmann sono grafi i cui nodi rappresentano variabili aleatorie e i cui archi rappresentano pesi simmetrici, ai quali è associata la tipica funzione di energia da minimizzare. Le macchine di Boltzmann sono in realtà una versione delle reti di Hopfield, dove però alcuni nodi sono "visible units", ovvero variabili note e osservate nei dati di addestramento, mentre altri sono "hidden units", ovvero non noti ma stimabili attraverso i medesimi processi di aggiornamento iterato propri delle reti di Hopfield, in particolare la belief propagation e il Gibbs sampling. Questa modifica comporta importanti conseguenze. In primo luogo, le macchine di Boltzmann sono considerabili a tutti gli effetti modelli generativi: attraverso il processo di aggiornamento dei valori dei nodi si raggiunge un pattern diverso da quelli presenti nei dati

di training. In secondo luogo, il costo computazionale della fase di inferenza cresce al crescere del numero di hidden units. Come nelle reti di Hopfield, una macchina di Boltzmann è un grafo denso, ovvero un'unica clique. E' comunque possibile addestrare sia una rete di Hopfield sia una Macchina di Boltzmann ipotizzando l'assenza di connessioni tra alcuni neuroni. Tale crescita del costo computazionale (già considerato alto nelle reti di Hopfield) ha reso poco popolari questi modelli. La maggior parte delle macchine di Boltzmann aderisce alla Hebbian Rule ipotizzando una distribuzione Bernoulli su tutti i suoi nodi, hidden e visible. Mentre nei modelli di Ising e di Sherrington-Kirkpatrick la tipica natura dei nodi è binaria in -1,1, nelle reti di Hopfield e nelle macchine di Boltzmann il valore di un nodo può infatti essere definito sul dominio di qualsiasi variabile aleatoria. Per tali modelli i casi più comuni sono comunque le variabili Bernoulli, definite in 0,1 (che sembrano rappresentare meglio la regola di Hebb), e quindi caratterizzate da funzioni di attivazione sigmoidali, ma è comune anche l'inclusione di nodi distribuiti come normali standard. Bisogna sempre ricordare che mentre i modelli di Ising e di Sherrington-Kirkpatrick hanno un dominio di studio ristretto al ferromagnetismo, e che quindi i loro nodi rappresentano la direzione di un magnete, i nodi delle reti di Hopfield e delle macchine di Boltzmann rappresentano neuroni biologici, e sono quindi potenzialmente applicabili in qualsiasi dominio di studio.

5.10 Inferenza nelle Reti di Markov

Le Reti di Markov presentano un numero di parametri abbastanza ristretto, se si pensa ad altri modelli multivariati come le reti neurali feedforward. Ciononostante, fare inferenza con un MRF può risultare computazionalmente molto costosto, in modo particolare in problemi dove il numero di variabili aleatorie è elevato. Due sono gli algoritmi generalemente più adottati per fare inferenza in un MRF:

- il Simulated Annealing, o ricottura simulata
- la Belief Propagation, o propagazione della credibilità

Il primo è stato pensato inizialmente per fare inferenza nelle Reti di Hopfield da John Hopfield stesso, e ha poi conosciuto più ampia diffusione come un generico algoritmo di ottimizzazione globale, applicabile a una immensa varietà di problemi.

Il secondo si è rivelato più efficiente del primo per fare inferenza specificamente nelle Reti di Markov, e ha conosciuto applicazioni anche in altri modelli grafici come le reti bayesiane.

Per costruzione, ipotizzando un numero inifinito di iterazioni, entrambi gli algoritmi conducono a una soluzione che rappresenta un minimo della funzione di energia libera presente nel sistema.

Trattandosi di algoritmi fortemente esplorativi, spesso tale soluzione rappresenta il minimo globale di tale funzione di energia.

5.10.1 Simulated Annealing

Si è visto come la distribuzione di Boltzmann sia direttamente legata al livello di energia presente in un insieme di variabili (da un punto di vista fisico comunemente associato a un sistema termodinamico di particelle, da una prospettiva statistica associabile a un processo stocastico multivariato o a un modello grafico probabilistico). Sia data una funzione di perdita qualsiasi, definita positiva, funzione di tale insieme di variabili. Allora tale funzione di perdita può essere trattata come una funzione di energia, e quindi associata a un livello di verosimiglianza sotto la distribuzione di Boltzmann.

L'algoritmo di Simulated Annealing, nella sua formulazione generale, non è altro che l'algoritmo Metropolis-Hastings applicato alla distribuzione di Boltzmann, dati una funzione di perdita degli stati, un operatoratore di transizione tra gli stati e una formula di decadimento della temperatura.

Sia dato un operatore di transizione del sistema dei parametri noto. un qualsiasi operatore in grado di campionare dall'insieme di variabili attualmente dato un nuovo insieme di valori, in qualche modo correlato al primo. In altre parole, tale operatore consente di spostare il sistema dall'attuale stato in cui si trova a un nuovo stato vicino. In un modello grafico, come il modello di Ising, lo stato rappresenta l'insieme dei valori delle variabili aleatorie attualmente assegnate alla rete, e un operatore di transizione idoneo tipicamente è un operatore di Gibbs, tramite il quale è possibile campionare ogni nodo della rete a partire dai valori dei suoi nodi vicini. In problemi più semplici, come la stima di una media gaussiana, tale operatore di transizione potrebbe essere costituito da una normale, con varianza fissata e media fissata al valore della media trovata all'iterazione precedente.

Elemento determinante consiste nell'idea della temperatura descrescente. Si è visto come la temperatura rappresenti una costante della distribuzione di Boltzmann, al cui crescere tale distribuzione tende all'uniforme. Un alto livello di temperatura corrisponde quindi a un alto livello di irregolarità, e quindi a un minor tasso di rifiuto nella procedura di Metropolis Hastings.

Di seguito si riporta una formulazione dell'algoritmo di Simulated Annealing per un generico problema di ottimizzazione. Si noti che il termine $\exp\left(\frac{(e-e^*)}{T}\right)$ è il fattore di Boltzmann, ovvero il rapporto $p(e^*)/p(e)$ dove p corrisponde alla misura di Gibbs. Tale rapporto rispetta la definizione del tasso di rifiuto presente nell'algoritmo di Metropolis.

Algorithm 4 Simulated Annealing

- sia dato uno stato iniziale S_0, una funzione di energia o perdita $E(s)$, un operatore di transizione $T(\hat{s} \leftarrow s)$, una temperatura massima T_{max}, un tasso di decadimento $\alpha \in (0,1)$ e un numero di iterazioni N.
- inizializzazione:
$T = T_{max}$
$S = S_0$
$e^* = E(S)$
$S^* = S$
- for $i \in \{1, .., N\}$:
- $T = \alpha T$
- $S = T(\hat{s} \leftarrow S)$
- $e = E(S)$
- if $(e \leq e^*)$:
- $e^* = e$
- $S^* = S$
- else:
- $r \sim Unif(0,1)$
- if $\left(r \leq \exp\left(\frac{(e-e^*)}{T}\right)\right)$:
- $e^* = e$
- $S^* = S$
- endif
- endif
- endfor
- la soluzione proposta è lo stato S^*, l'energia minima raggiunta è e^*

Algorithm 5 Algoritmo di Metropolis

- sia dato uno stato iniziale S_0, una funzione di verosimiglianza $f(s) \propto p(s)$, una distribuzine proposal campionabile $g(s|s_i)$, e un numero di iterazioni N.
- inizializzazione:
$s^* \sim g(s|s_0)$
- for $i \in \{1, .., N\}$:
- $\hat{s} \sim g(s|s_i)$
- $w = \frac{f(s^*)}{f(s_i)} = \frac{p(s^*)}{p(s_i)}$
- if $w \geq 1$:
- $s^* = s_i$
- else:
- $r \sim U(0, 1)$
- if $w \geq r$:
- $s^* = s_i$
- endif
- endif
- endfor

Un'interessante applicazione dell'algoritmo fu proposta da Hopfield sul Traveling Salesman Problem (TSP). Si deve stabilire il percorso ottimale di un corriere che deve fare delle consegne in diverse città. Hopfield ha proposto questa formulazione del problema:

- lo stato è rapprsentato da un vettore ordinato di K stringhe, contenente i nomi delle K tappe in cui il corriere deve passare
- la funzione di costo è quella che a partire da questo ordinamento delle tappe calcola la lunghezza del percorso (è data a priori una matrice delle distanze fra le tappe, a partire dalla quale è facile calcolare la distanza totale percorsa)
- l'operatore di transizione è rappresentato da un semplice scambio di posizione di due tappe nel vettore ordinato

Per l'epoca in cui fu proposta, la soluzione al TSP attraverso il Simulated Annealing si rivelò la più performante tra quelle disponibili.

Si deve evidenziare in questa tesi come l'algoritmo di Simulated Annealing e quello di Annealed Importance Sampling non abbiano nulla a che fare l'uno con l'altro, malgrado il nome e l'idea di un livello di temperatura descrescente nel corso delle iterazioni. I due algoritmi hanno due funzioni completamente differenti.

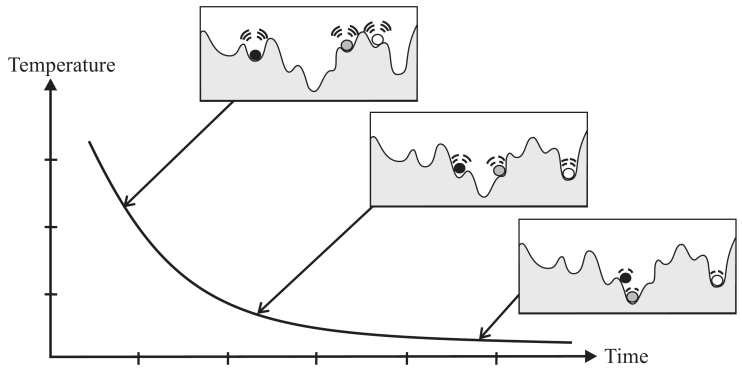


Figure 14: Simulated Annealing: illustrazione del livello di energia per diversi run

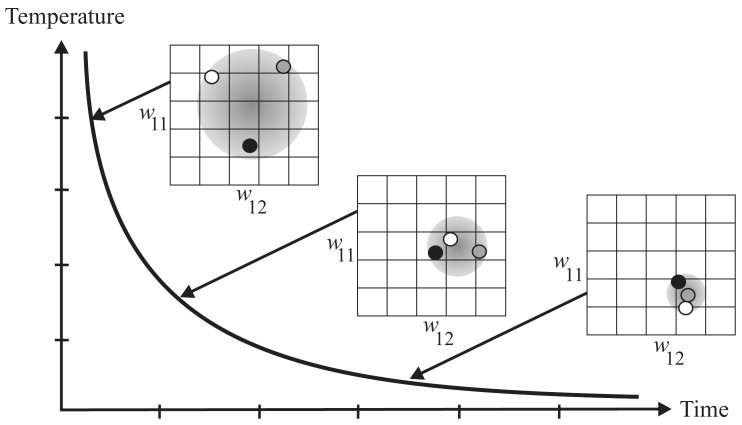


Figure 15: Simulated Annealing: illustrazione della convergenza a soluzioni di minimo globale per diverse inizializzazioni

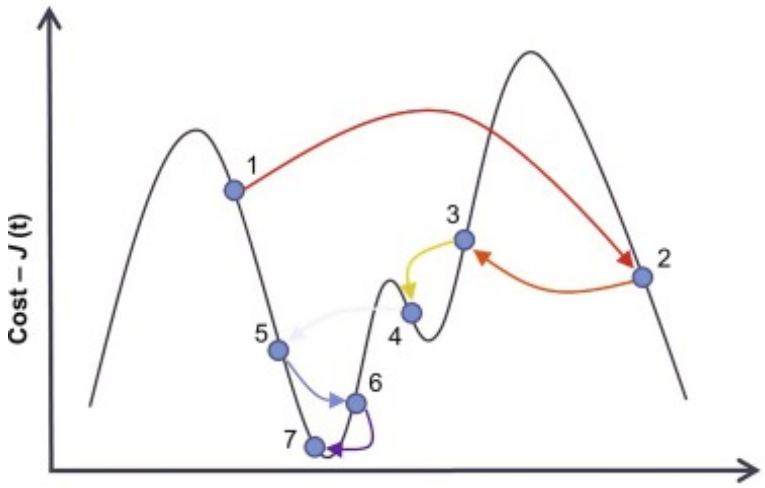


Figure 16: Simulated Annealing: esempio di salti tra stati
Al crescere del numero di iterazioni si riduce il livello di temperatura, e di conseguenza la propensione del modello ad allontanarsi dalla regione attuale.

5.10.2 Belief Propagation

Nella descrizione dell'algoritmo si fa qui riferimento a [Yedidia et al., 2003].

La Belief Propagation (BP) è un algoritmo di tipo message-passing, che può essere applicato a modelli grafici probabilistici di molte famiglie. Per modelli di variabili aleatorie binarie, come il modello di Ising e la prima rete di Hopfield, si tratta probabilmente del metodo di inferenza computazionalmente più efficiente. Nelle fasi di inferenza dei MRF di tipo linear - chain , come le HMM (Hidden Markov Models) e i CRF (Conditional Random Fields) , è possibile applicare l'Algoritmo di Viterbi, una versione computazionalmente più veloce della BP.

La BP trae vantaggio dalla riformulazione di una catena di Markov come un Factor Graph: vengono introdotti dei nuovi nodi, detti factor nodes, in ogni arco del grafo, mentre i nodi originari vengono chiamati variable nodes. In questo modo è possibile strutturare le operazioni di inferenza in due fasi:

- un fase di invio di messaggi dai variable nodes ai factor nodes
- una fase di invio di messaggi dai factor nodes ai variable nodes

Una volta che tutti i messaggi sono stati calcolati, è possibile calcolare i beliefs (le credibilità) per ogni valore che ogni variabile può assumere. Tali beliefs non sono altro che score normalizzati in modo tale da sommare a 1. Una volta che l'algoritmo raggiunge convergenza, se ciò accade, si può dimostrare che i beliefs coincidono con le effettive probabilità marginali dei valori di ciascuna variabile. A differenza del Simulated Annealing l'algoritmo BP non si limita quindi a fare inferenza sullo stato più probabile del sistema nel suo complesso, ma consente anche di stimare le distribuzioni marginali di ogni nodo. Si nota tuttavia che tale procedura, per un elevato numero di variabili, conduce facilmente a una soluzione solo per variabili binarie o categoriche.

La BP si svolge come segue:

Sia dato un factor graph $G = (V, A, E)$ dove V e A sono rispettivamente i variable nodes e i factor nodes, e E l'insieme degli archi tra un nodo $v \in V$ e un nodo $a \in A$. Siano date anche le funzioni clique rispetto alle quali la distribuzione della Rete di Markov è fattorizzabile

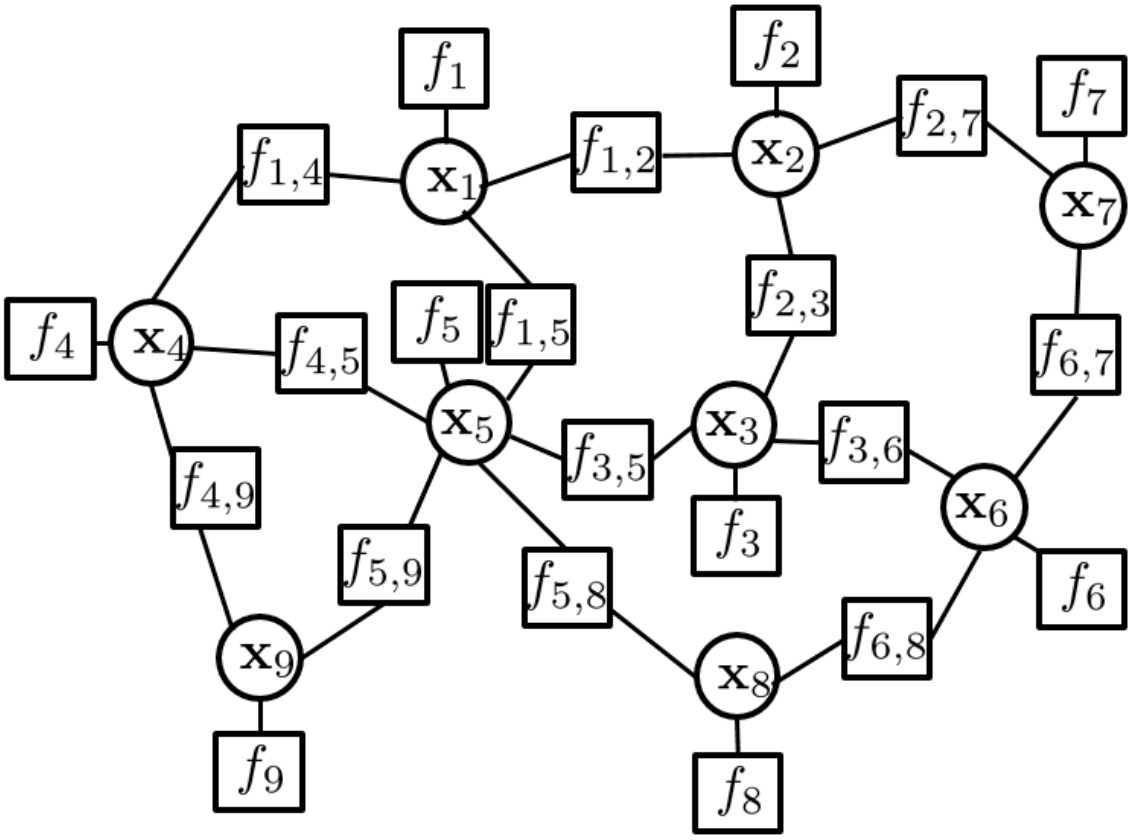


Figure 17: Esempio di Factor Graph
Si vede che nessun nodo comunica con un nodo del gruppo a cui appartiene. Questa bipartizione dei nodi consente di dividere il processo iterato di aggiornamento in due fasi. Immagine tratta da [Jian et al., 2017].

rispetto ai suoi nodi in V. Si indica di seguito con $C(G)$ l'insieme dei sottoinsiemi delle clique in $G = (V, E)$:

$$p(G) = \prod_{c \in C(G)} f_c(\mathbf{v}_c) \tag{69}$$

Nei modelli di Ising e di Hopfield, tali clique sono le funzioni $\psi_{ij}(x_i, x_j) = x_i x_j w_{ij}$ e le funzioni $\phi_i(x_i) = x_i b_i$.

Si definisca $x_v \in dom(v)$ come un valore che la variabile $v \in V$ può assumere. Nel caso binario può valere $dom(v) = -1, 1$. Si definisca $m_{av}(x_v)$ come il messaggio mandato dal factor node a al variable node v quando questa assume valore x_v, e viceversa $m_{va}(x_v)$ come il messaggio mandato da v ad a rispetto al valore x_v. Si definiscano $N(a) \subseteq V$ e $N(v) \subseteq A$ come gli insiemi dei nodi vicini ad a e a v. Si definisca $b_v(x_v)$ la probabilità marginale stimata (belief) che la variabile v assuma valore x_v, e $s_v(x_v)$ lo score corrispondente non normalizzato. Per semplicità, si assuma che il modello grafico presenti al massimo cliques di ordine 2. Allora le equazioni message passing della BP si svolgono nelle seguenti due fasi:

- $$m_{va}(x_v) = \prod_{a^* \in N(v) \setminus a} m_{a^* v}(x_v) \tag{70}$$

- $$m_{av}(x_v) = \phi_v(x_v) \sum_{v \in N(a) \setminus v} \sum_{x_{v^*} \in dom(v^*)} \psi(x_v, x_{v^*}) \prod_{v^* \in N(a) \setminus v} m_{v^* a}(x_{v^*}) \tag{71}$$

Se un nodo non presenta factor nodes vicini, oltre all'unico a cui è connesso, il termine $m_{a^* v}(x_v)$ è sostituito da una uniforme in (0,1). Una volta raggiunta convergenza di $m_{va}(x_v)$ e di $m_{av}(x_v)$ $\forall a, v, x_v$ le beliefs si calcolano come:

$$s_v(x_v) = \prod_{a \in N(v)} m_{av}(x_v) \tag{72}$$

e vengono infine normalizzate:

$$b_v(x_v) = \frac{s_v(x_v)}{\sum_{x \in dom(v)} s_v(x)} \tag{73}$$

Figure 18: Illustrazione grafica della belief propagation
Nell'immagine, le x sono i variable nodes, le f sono i factor nodes. I nodi x e i nodi f sono aggiornati in fasi alternate. Immagini tratte da [Sudderth, 2014].

La belief propagation ha ispirato la formulazione delle Macchine di Boltzmann Ristrette come factor graph. Nel prossimo capitolo si vede come una formulazione di questo tipo consente di rendere il processo di inferenza istantaneo, e il processo di training computazionalmente molto più accessibile.

Algorithm 6 Belief Propagation

- sia definito un grafo di variabili aleatorie $G = (V, E)$ con $x_v \in dom(v) \forall v \in V$. Siano definiti $a \in A$ tali per cui G diventa fattorizabile tra gli insiemi V e A.
- inizializzazione:
$m_{va}(x_v) = 1 \forall a \in A, v \in V, x_v \in dom(v)$
- loop: riperti fino a convergenza di $m_{va}(x_v) \forall v, a, x_v$:
- $m_{va}(x_v) = \prod_{a^* \in N(v) \setminus a} m_{a^* v}(x_v)$
- $m_{av}(x_v) = \phi_v(x_v) \sum_{v \in N(a) \setminus v} \sum_{x_{v^*} \in dom(v^*)} \psi(x_v, x_{v^*}) \prod_{v^* \in N(a) \setminus v} m_{v^* a}(x_{v^*})$
- endloop
- $s_v(x_v) = \prod_{a \in N(v)} m_{av}(x_v)$
- $b_v(x_v) = \frac{s_v(x_v)}{\sum_{x \in dom(v)} s_v(x)}$
- assegna ad ogni nodo v il valore x_v : $x_v = \text{argmax}_{x \in dom(v)} b_v(x)$

6 RBM: Macchine di Boltzmann Ristrette

Si è visto come le macchine di Boltzmann presentino una capacità di rappresentare fenomeni complessi di molto superiore a quella delle reti di Hopfield. Si è visto anche che il costo computazionale di questa capacità si presenta particolarmente oneroso e insostenibile per un numero di hidden units elevato. Nel 2006, G. Hinton propone una particolare classe di Macchine di Boltzmann, denominate RBM. Il primo nome dato a questa sotto famiglia modelli fu Harmonium, in quanto ispirati a uno schema grafico proposto da [Smolensky, 1986] che era stato chiamato così, ma successive pubblicazioni già nel 2000 [Teh and Hinton, 2000] la ribattezzarono con il nome (più rappresentativo) Resticted Boltzmann Machine (RBM). Nelle RBM i nodi sono divisi in due categorie, dette hidden units e visible units, come nelle comuni Macchine di Boltzmann. La differenza deriva dal fatto che nelle RBM le hidden units e le visible units sono raccolte in due layer distinti, analoghi a quelli delle reti feedforward. In questo modo ogni hidden unit è collegata solo alle visible units e a tutte le visible units, e che ogni visible unit è collegata solo alle hidden units e a tutte le hidden units (da cui la restrizione nel nome). In altre parole, ogni visible unit è indipendente da tutte le altre visible units, condizionatamente al layer delle hidden units, e viceversa ogni hidden unit è indipendente dalle altre hidden units date le visible units. Pertanto una RBM è un grafo bipartito, ovvero nel quale i nodi sono raggruppati in due classi distinte, nelle quali ogni nodo è connesso solo a nodi appartenenti all'altra classe. Inoltre è un factor graph, ovvero un grafo che consente di rappresentare la fattorizzazione di una funzione. Quando una rete di Markov è un factor graph la funzione che consente di fattorizzare è la sua densità di probabilità. Attraverso questa struttura "ristretta" le RBM riescono a sfruttare la proprietà di Markov locale per velocizzare metodi di stima basati su message passing equations, come la belief propagation. Si vede infatti che dato il layer visible è possibile stimare direttamente il layer hidden, e viceversa. Sempre nel 2006, Hinton propone un metodo di training specifico per le RBM, chiamato contrastive divergence, molto più economico in termini computazionali rispetto alla belief propagation, che rappresenta il fondamento dei metodi di contrastive learning. Le RBM addestrate con contrastive learning sono la base da cui verranno sviluppate anche reti neurali profonde come le DBN (Deep Belief Networks) e le DBM (Deep Boltzmann

Machines).

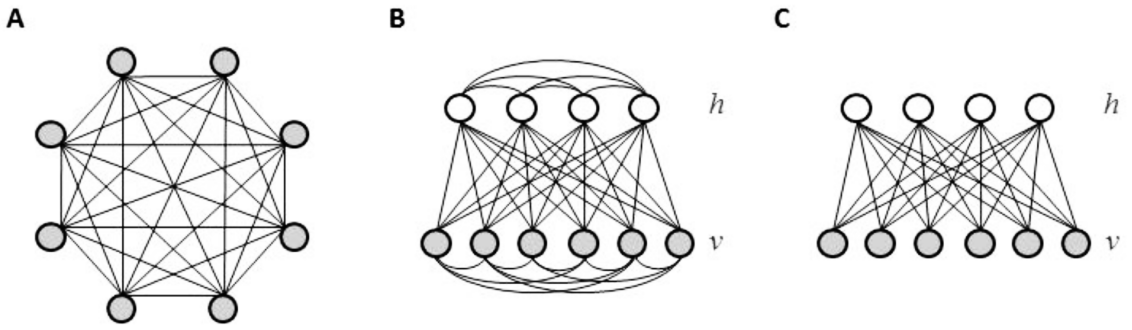


Figure 19: Reti di Hopfield, Macchine di Boltzmann, e Macchine di Boltzmann Ristrette a confronto

6.1 Energia e inferenza nelle Bernoulli-RBM

La versione più semplice di una RBM è la Bernoulli-RBM, nella quale ogni nodo visibile e latente presenta una funzione di attivazione sigmoidale, e comprende quindi una variabile binaria in $\{0, 1\}$.

Sia data una Bernoulli-RBM con V visible units e H hidden units. Allora l'energia negativa della RBM, data una matrice di pesi di interazione W, un vettore di offsets A e un vettore di offsets B sarà data da

$$-E(\mathbf{v}, \mathbf{h}) = \sum_{i=1}^{V} \sum_{j=1}^{H} w_{ij} v_i h_j + \sum_{i=1}^{V} a_i v_i + \sum_{j=1}^{H} b_j h_j = \mathbf{v}^{\mathrm{T}} \mathbf{W} \mathbf{h} + \mathbf{v}^{\mathrm{T}} \mathbf{A} + \mathbf{h}^{\mathrm{T}} \mathbf{B} \tag{74}$$

Dove $w_{ij} = \mathbf{W}[i, j]$, $a_i = \mathbf{A}[i]$, $b_j = \mathbf{B}[j]$, $v_i = \mathbf{v}[i]$, $h_j = \mathbf{h}[j]$, $\mathbf{W} \in \mathbb{R}^{V \mathrm{x} H}$, $\mathbf{A} \in \mathbb{R}^{V}$, $\mathbf{B} \in \mathbb{R}^{H}$, $\mathbf{v} \in \{0, 1\}^{V}$ e $\mathbf{h} \in \{0, 1\}^{H}$.

Bisogna da notare che quando si eseguono approssimazioni mean-field dei layer, ovvero quando si adottano i valori attesi dei nodi al posto di eventuali valori dati o campionati, per variabili bernoulliane vale $\mathbf{v} \in [0, 1]^{V}$ e $\mathbf{h} \in [0, 1]^{H}$.

La densità di probabilità della RBM (in questo caso definita sul dominio discreto, e quindi la sua funzione di probabilità) presenta la seguente forma:

$$\mathrm{P}(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h})) \tag{75}$$

Per cui, marginalizzando rispetto al layer latente, un layer visibile presenta la seguente probabilità:

$$\mathrm{P}(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h} \in \mathbb{H}} \exp(-E(\mathbf{v}, \mathbf{h})) \tag{76}$$

dove $\mathbb{H}$ è l'insieme di tutte le possibili configurazioni dell'hidden layer.

Purtroppo non è possibile calcolare la funzione di partizione Z. Si vede infatti che:

$$Z = \sum_{\mathbf{v} \in \mathbb{V}} \sum_{\mathbf{h} \in \mathbb{H}} \exp(-E(\mathbf{v}, \mathbf{h})) \tag{77}$$

dove ancora $\mathbb{V} := \{0, 1\}^V$.

Per cui ci si accontenta dell'energia negativa come funzione di log verosimiglianza, in quanto è evidente come

$$\exp(-E(\mathbf{v}, \mathbf{h})) \propto \mathrm{P}(\mathbf{v}, \mathbf{h}) \tag{78}$$

Esiste tuttavia un algoritmo, chiamato Annealed Importance Sampling (AIS) che consente di ricavare la funzione di partizione Z. Tale stima tuttavia presenta un elevato costo computazionale, e varia al variare dei parametri di interazione $\mathbf{W}$ e di offset $\mathbf{A}, \mathbf{B}$ che identificano il modello. Pertanto la stima delle RBM si basa sul metodo della contrastive divergence, che adotta una funzione di perdita differente sia dalla Massima Verosimiglianza esatta sia dalla Divergenza di Kullback-Leibler. Il contrastive learning è illustrato con maggiore dettaglio nel prossimo capitolo. La conoscenza della constante di normalizzazione Z non è quindi necessariamente richiesta per fare inferenza sui parametri o per campionare i dati. Si rende invece necessaria, in materia di topic modeling, nel calcolo della perplexity del modello RSM. In questo capitolo verrà quindi fornita una descrizione generale dell'algoritmo AIS, mentre nel capitolo dedicato al modello RSM verranno illustrate le ulteriori problematiche connesse alla stima di Z in tale modello, e delle efficienti soluzioni per la stima della perplexity. Si può comunque notare che il calcolo dell'upper bound della perplexity non richiede la conoscenza della funzione di partizione e non mostra problemi di precisione numerica e di costo computazionale, per cui può essere utilizzato come proxy della perplexity di RSM.

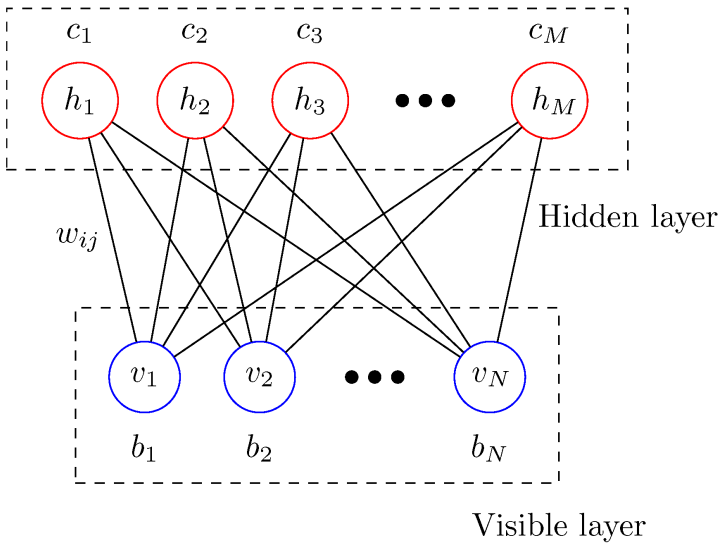


Figure 20: Parametri di una generica RBM
Si noti che pesi di interazione tra i duel layer sono simmetrici. Immagine tratta da [Oh et al., 2020].

Il grande vantaggio delle RBM rispetto alle reti di Markov può essere apprezzato nel calcolo delle probabilità condizionate dei due layer. Data la proprietà di Markov locale, infatti, è possibile definire

$$\mathrm{P}(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_{i=1}^{V} w_{ij} v_i\right) \tag{79}$$

e specularmente

$$\mathrm{P}(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_{j=1}^{H} w_{ij} h_j\right) \tag{80}$$

Dove $\sigma(x) = \frac{\exp(x)}{1+\exp(x)}$ è la funzione di attivazione sigmoide. Si vede che è una specificazione della funzione di attivazione softmax, in quanto

$$\mathrm{P}(v_i = 1|\mathbf{h}) = \frac{\exp(-E(v_i = 1))}{\exp(-E(v_i = 1)) + \exp(-E(v_i = 0))} \tag{81}$$

dove, per la proprietà di Markov locale:

$$-E(v_i|\mathbf{h}) = a_i v_i + \sum_{j=1}^{H} w_{ij} h_j v_i \tag{82}$$

che si vede essere 0 per $v_i = 0$, da cui l'attivazione sigmoidale. Si vede quindi che la densità condizionata del layer visibile è perfettamente fattorizzabile nei suoi nodi, ovvero:

$$p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{V} \exp(-E(v_i|\mathbf{h})) = \exp(-E(\mathbf{v}, \mathbf{h})) \tag{83}$$

Si vede che, date le distribuzioni condizionate, è possibile campionare ciascun layer dato l'altro. Il metodo di campionamento tipico per le variabili Bernoulli è lo step accettazione-rifiuto: dato $\mathrm{p} = \mathrm{P}(v_i = 1|\mathbf{h})$, e campionato $u \sim U(0, 1)$, se $p > u$ campiono $v_i = 1$, altrimenti $v_i = 0$. Analoga operazione è applicabile sulle hidden units bernoulliane. Questo meccanismo consente anche di campionare un layer dato se stesso. Se infatti è possibile campionare $\mathbf{h}_1$ a partire da $\mathbf{v}_0$, è anche possibile campionare $\mathbf{v}_2$ a partire da $\mathbf{h}_1$. Tale step di campionamento, anche detto operatore di transizione, è essenziale sia ai fini del training con la contrastive divergence sia ai fini della stima della funzione di partizione attraverso AIS.

Esistono altre versioni delle RBM in grado di trattare variabili gaussiane su un solo layer o su entrambi, e RBM in grado di gestire altri tipi di variabili aleatorie, come Beta e Poisson. Il modello RSM, introdotto nel capitolo successivo, rappresenta una RBM in grado di trattare variabili multinomiali sul layer visibile e delle variabili bernoulli sul layer latente.

6.2 Energia libera e densità marginale del visible layer

Si è detto che , marginalizzando rispetto al layer latente, un layer visibile presenta la seguente probabilità:

$$\mathrm{P}(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h} \in \mathbb{H}} \exp(-E(\mathbf{v}, \mathbf{h})) \tag{84}$$

Utile proprietà delle Bernoulli-RBM, e in generale di tutte le RBM per cui vale $\mathbb{H} := \{0, 1\}^H$, è la possibilità di riscrivere il numeratore in forma chiusa. Si può infatti dimostrare che

$$p(\mathbf{v}) = \mathrm{P}(\mathbf{v})Z = \exp\left(\sum_{i=1}^{V} a_i v_i\right) \prod_{j=1}^{H} \left(1 + \exp\left(b_j + \sum_{i=1}^{V} v_i w_{ij}\right)\right) \tag{85}$$

che è la densità marginale del layer visibile. Si definisce free energy, o energia libera, di

Proof 6.1: Free energy come densità marginale del visible layer

$$p(\mathbf{v}) = \sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\, \mathbf{h})) \tag{87}$$

$$p(\mathbf{v}) = \sum_{\mathbf{h}\in\mathbb{H}} \exp\left(\sum_{i=1}^{V} v_i a_i + \sum_{j=1}^{H} h_j b_j + \sum_{j=1}^{H}\sum_{i=1}^{V} v_i h_j w_{ij}\right) \tag{88}$$

$$p(\mathbf{v}) = \sum_{\mathbf{h}\in\mathbb{H}} \exp\left(\sum_{i=1}^{V} v_i a_i\right) \exp\left(\sum_{j=1}^{H} h_j b_j\right) \exp\left(\sum_{j=1}^{H}\sum_{i=1}^{V} v_i h_j w_{ij}\right) \tag{89}$$

$$p(\mathbf{v}) = \exp\left(\sum_{i=1}^{V} v_i a_i\right) \sum_{\mathbf{h}\in\mathbb{H}} \exp\left(\sum_{j=1}^{H} h_j b_j\right) \exp\left(\sum_{j=1}^{H}\sum_{i=1}^{V} v_i h_j w_{ij}\right) \tag{90}$$

$$p(\mathbf{v}) = \exp\left(\sum_{i=1}^{V} v_i a_i\right) \prod_{j=1}^{H} \sum_{h_j\in\{0,1\}} \exp\left(h_j b_j\right) \exp\left(h_j \sum_{i=1}^{V} v_i w_{ij}\right) \tag{91}$$

$$p(\mathbf{v}) = \exp\left(\sum_{i=1}^{V} v_i a_i\right) \prod_{j=1}^{H} \left(1 + \exp\left(b_j + \sum_{i=1}^{V} v_i w_{ij}\right)\right) \tag{92}$$

da cui la tesi.

una qualsiasi RBM, il logaritmo negativo della verosimiglianza marginale dei layer visibili, e si scrive quindi:

$$-F(\mathbf{v}) = -\ln(p(\mathbf{v})) = \sum_{i=1}^{V} v_i a_i + \sum_{j=1}^{H} \ln\left(1 + \exp\left(b_j + \sum_{i=1}^{V} v_i w_{ij}\right)\right) \tag{86}$$

In altre parole, l'energia libera è il livello di energia complessivamente associato a una particolare configurazione delle visible units, ed è quindi definibile come la densità di probabilità associata alla distribuzione di Boltzmann dopo aver marginalizzato rispetto alle hidden units.

Di seguito la relativa dimostrazione:

6.3 Interpretazione delle RBM come PoE

Nel 1999 G.E.Hinton propone i PoE (Product of Experts) come una nuova classe di modelli. Si tratta di una strategia che consente di modellare una distribuzione complessa come il prodotto di diverse distribuzioni molto più semplici. Tale combinazione avviene attraverso il prodotto delle loro densità, adeguatamente normalizzato. Un PoE si definisce

quindi come:

$$P(x) = \frac{1}{Z} \prod_{j=1}^{M} f_j(x) \tag{93}$$

dove M è il numero di distribuzioni semplici, f_j le rispettive densità, x una variabile aleatoria o un processo stocastico comune e Z una costante di normalizzazione:

$$Z = \int \prod_{j=1}^{M} f_j(x) dx \tag{94}$$

I PoE sono stati negli anni sostituiti dai MoE (Mixture of Experts) nei quali diverse densità non vengono combinate attraverso un prodotto ma attraverso una somma pesata con dei pesi dipendenti dagli input. In altre parole, la densità di un MoE si definisce come:

$$f(x) = \sum_{j=1}^{M} w_j(x) f_j(x) \tag{95}$$

Dove i pesi w_j vengono in genere ricavati da una speciale gating function. Per questa ragione, in ambito informatico, i PoE vengono spesso associati a un'operazione booleana "and" mentre i MoE sono associati a un'operatore "or". In altre parole, mentre in una mistura o in una somma, un esperto può incidere grandemente sulla probabilità della distribuzione combinata, in un prodotto è necessario che tutti gli esperti siano concordi nel fornire alta probabilità a un determinato evento.

Sempre Hinton troverà proprio nelle RBM una forma di PoE. La distribuzione condizionata $P(v_i|\mathbf{h})$ può infatti essere vista come il prodotto delle densità di tanti esperti quante sono le hidden units. Una RBM può quindi essere vista come una distribuzione congiunta di tanti esperti latenti sui dati visibili. Se si vuole si può anche ribaltare il concetto: ogni unità visibile è una distribuzione su tutte le hidden units.

Va notato infine come l'algoritmo Contrastive Divergence fu proposto da Hinton nel 2000 come procedura di training sui dati per un qualsiasi PoE, e solo nel 2006 fu pubblicata la sua applicazione alle RBM.

6.4 AIS: Annealed Importance Sampling

L'AIS è una procedura di approssimazione numerica della funzione di ripartizione di una distribuzione di cui si conosce solo la funzione di densità di probabilità. Siano date due distribuzioni, definite sul medesimo supporto, A e B. Si ipotizzi che A sia una distribuzione proposta ben nota, e che B sia una speciale distribuzione posterior. Siano p_A e p_B le relative funzioni di densità di probabilità, definite sul medesimo supporto, e Z_A e Z_B le rispettive funzioni di partizione, tali che

$$P_A(x) = \frac{p_A(x)}{Z_A} \tag{96}$$

$$P_B(x) = \frac{p_B(x)}{Z_B} \tag{97}$$

Si assuma inoltre che p_A, p_B siano funzioni note, che Z_A sia dato e che si voglia stimare Z_B. In altre parole, data la funzione di partizione della proposal B si vuole conoscere la funzione di partizione della posterior A.

Si faccia un'ultima assunzione: che esista la possibilità di campionare una osservazione x_i' data un'osservazione x_i, ovvero che sia definito un'operatore di transizione $T(x' \leftarrow x)$.

Nell'ipotesi (inverosimile) in cui $p_A \sim p_B$ potrebbe essere una buona approssimazione del rapporto Z_A/Z_B la seguente:

$$\frac{Z_B}{Z_A} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{p_B(x_i)}{p_A(x_i)} = \hat{r} \tag{98}$$

Se tale ipotesi fosse soddisfatta, si potrebbe facilmente ricavare $\hat{Z_B} = \hat{r} Z_A$. Purtroppo è molto probabile che la proposal a disposizione (una uniforme o una gaussiana multivariata) risulti molto distante dalla posterior di cui si voglia stimare Z_B.

In questo scenario, l'algoritmo AIS funziona come segue.

L'applicazione della procedura AIS a una Bernoulli RBM richiede quindi di definire la distribuzione proposal, la distribuzione posterior e l'operatore di transizione. L'osservazione x si traduce nel layer visibile $\mathbf{v}$. La proposal tipicamente adottata in questo caso è l'uniforme, con probabilità $P_B(x) = \frac{1}{2^H}$ dove H è il numero di hidden units. La posterior, quando l'hidden

Algorithm 7 Annealed Importance Sampling

- si specifica un numero di iterazioni S e un numero di iterazioni N
- si inizializzano una serie di temperature inverse β_s per $s = 0...S$
tali che $0 = \beta_0 < ... < \beta_S = 1$
- si definiscono una serie di distribuzioni intermedie
di densità $p_s(x) = p_A(x)^{\beta_s} p_B(x)^{1-\beta_s}$.

▷ Si vede dunque che $p_0 = p_A$ e che $p_S = p_B$

- for i in $\{1...N\}$:
- si campiona x_{i0} da p_0
- for s in $\{1...S-1\}$:
- si campiona $x_{is+1} \sim T(x \leftarrow x_{is})$
- endfor
- si calcola $w_i = \prod_{s=1}^{S} \frac{p_s(x_{is})}{p_{s-1}(x_{is})}$
- endfor
- si stima $\hat{r} = \frac{1}{N} \sum_{i=1}^{N} w_i$
- si stima $\hat{Z_S} = \hat{r} Z_0$

layer presenta attivazione sigmoide, è calcolabile in forma chiusa come descritto nella sezione precedente. L'operatore di transizione è invece tipicamente definito come un campionamento in due fasi del visible layer: dato $\mathbf{v}_0$, l'operatore $T(\mathbf{v}' \leftarrow \mathbf{v}_0)$ consiste nel campionare prima $\mathbf{h}_0 \sim P(\mathbf{h}|\mathbf{v}_0)$ e poi $\mathbf{v}' \sim P(\mathbf{v}|\mathbf{h}_0)$.

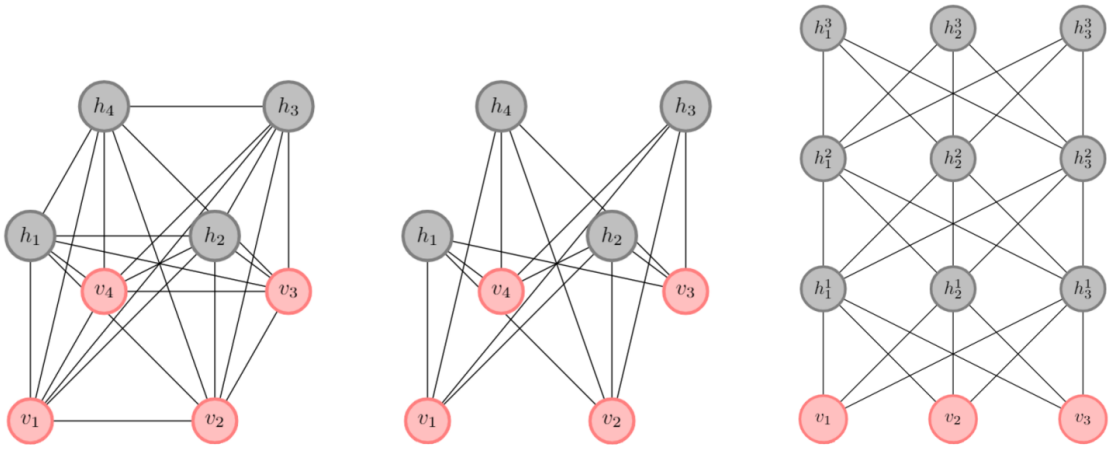
6.5 Deep Boltzmann Machines e Deeb Belief Networks

Le DBN, nella loro formulazione più semplice, consistono in diverse RBM impilate una sopra l'altra: la prima rete ricava dai nodi visibili una serie di variabili latenti, mentre la seconda prende quelle variabili latenti e le inserisce nel suo layer di visible units, ricava poi un ulteriore strato di variabili latenti e li passa a una terza, e così via. In realtà, trattandosi di grafi indiretti che possono essere ciclici, l'inferenza sui primi layer richiederebbe la conoscenza a priori dei layer successivi, ragion per cui la posterior di ogni hidden layer risulterebbe intrattabile. Nelle DBM non sarebbe quindi possibile fare inferenza seguendo una procedura greedy, ossia layer by layer, e procedure di inferenza più rigorose richiederebbero l'utilizzo di metodi Monte Carlo, in particolare Gibbs sampling, o di Belief Propagation, come nelle classiche macchine di Boltzmann. Tali procedure comporterebbero tuttavia un costo computazionale insostenibile per la maggior parte delle applicazioni ad alta dimensionalità.

Sono considerate DBM tuttavia ogni forma di reti neurali profonde non supervisionate

addestrate con metodi di contrastive learning (applicabili solo addestrando una RBM per ogni layer, ovvero seguendo la procedura greedy nella fase di inferenza), MCMC o belief propagation. Un esempio di DBM è il modello Over-RSM, una rete con due hidden layers dove i parametri degli archi tra il primo e il secondo layer e degli archi tra il terzo e il secondo layer sono condivisi. Il parametric t-SNE è una rete non supervisionata, ma non è una DBM in quanto il suo training segue una procedura di backpropagation, ragion per cui si configura come una rete feedforward e non come una Macchina di Boltzmann.

Le DBN (Deep Belief Networks) sono invece un ibrido tra reti neurali feedforward e DBM: sono modelli nelle quali i primi layer sono inizializzati attraverso metodi di contrastive learning non supervisionato, seguendo una procedura greedy layer by layer, e in una seconda i layer successivi sono addestrati attraverso metodi supervisionati e backpropagation. In altre parole, le DBN sono reti neurali profonde il cui training si divide in una fase di pre-training basato sui metodi di contrastive learning e una fase di fine-tuning basato su backpropagation. Nel caso delle DBN non supervisionate, la fase di pre-training coinvolge solo i primi hidden layer mentre la fase di fine tuning coinvolge solo l'ultimo o gli ultimi layer. Nel caso delle DBN supervsionate, la fase di fine tuning può comprendere anche il layer coinvolti dalla fase di pre-training. La presenza di pesi simmetrici e la struttura partizionata in layer rendono le DBM e le DBN non supervisionate la prima forma di modelli generativi autoencoder.



(a) General Boltzmann machine (b) Restricted Boltzmann machine (c) Deep Boltzmann machine

Figure 21: Diverse tipologie di Macchine di Boltzmann
Immagine tratta da [Srivastava and Sundararaghavan, 2023].

7 Training di RBM: metodi di contrastive learning

In (G.E. Hinton., 2000) viene introdotto l'algoritmo del contrastive divergence per addestrare modelli PoE (Product of Experts) dai dati. Questo algoritmo si mostra molto più efficace della Belief Propagation per le RBM. La funzione di perdita della procedura di training copre quattro funzioni di perdita simultaneamente:

- la log verosimiglianza negativa dei dati
- l'energia libera dei dati (che corrisponde alla log verosimiglianza negativa)
- la differenza tra la free energy reale dei dati e la free energy sotto il modello stimato
- la divergenza di Kullback-Leibler tra la reale distribuzione dei dati e la distribuzione sotto il modello stimato

Si può vedere come queste misure coincidono in una RBM.

7.1 equivalenza tra la divergenza di KL e la log verosimiglianza

Per qualsiasi modello probabilistico, si può dimostrare l'equivalenza tra la ricerca dei parametri attraverso la minimizzazione della KL tra la distribuzione vera e la distribuzione sotto il modello, e massimizzazione della log verosimiglianza dei dati sotto il modello.

In altre parole, si vuole dimostrare che:

$$argmin_\theta \ \mathrm{KL}(q(x)||P(x;\theta)) = argmax_\theta \ln(P(x;\theta)) \tag{99}$$

dove q è la distribuzione reale dei dati, P è la distribuzione sotto il modello e $\theta \in \Theta$ dove Θ è lo spazio parametrico nel quale ogni ogni possibile modello P è esattamente identificato dalla configurazione dei suoi parametri θ.

7.2 gradiente ideale per una RBM

I parametri di una RBM ottimizzata con contrastive learning vengono ottimizzati con l'algoritmo di discesa del gradiente, come nel caso delle reti feedforward. Tale metodo di training richiede di poter calcolare il gradiente della funzione di perdita rispetto al parametro che si vuole ottimizzare. Si è visto nel paragrafo precedente che la minimizzazione della funzione di perdita della RBM, la divergenza KL, corrisponde alla minimizzazione della log verosimiglianza negativa dei dati, ovvero alla free energy sotto il modello. Si è anche visto

Proof 7.1: equivalenza tra la divergenza di KL e la log verosimiglianza di una RBM

$$\mathrm{KL}(q||P) = \sum_x q(x) \ln\left(\frac{q(x)}{P(x;\theta)}\right) \tag{100}$$

da cui

$$\mathrm{KL}(q||P) = \sum_x q(x) \ln(q(x)) - \sum_x q(x) \ln(P(x;\theta)) \tag{101}$$

Si vede che il primo termine non dipende da θ, pertanto

$$argmin_\theta \ \mathrm{KL}(q||P) = argmin_\theta - \sum_x q(x) \ln(P(x;\theta)) = argmax_\theta \sum_x q(x) \ln(P(x;\theta)) \tag{102}$$

Se abbiamo a disposizione un campione di N osservazioni $o_1 ... o_N$ possiamo approssimare q attraverso la distribuzione empirica dei dati, ovvero:

$$q(x) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}(x = o_n) \tag{103}$$

Dove $\mathbb{I}$ è la funzione indicatrice, ovvero

$$\mathbb{I}(x = o_n) = \begin{cases} 1 & x = o_n \\ 0 & altrimenti \end{cases} \tag{104}$$

Allora vale

$$\sum_x q(x) \ln(P(x;\theta)) = \sum_x \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}(x = o_n) \ln(P(x;\theta)) = \frac{1}{N} \sum_{n=1}^{N} \ln(P(o_n;\theta)) \tag{105}$$

Che corrisponde alla log verosimiglianza media dei dati sotto il modello, da cui la tesi.

$$argmin_\theta \ \mathrm{KL}(q||P) = argmax_\theta \frac{1}{N} \sum_{n=1}^{N} \ln(P(o_n;\theta)) \tag{106}$$

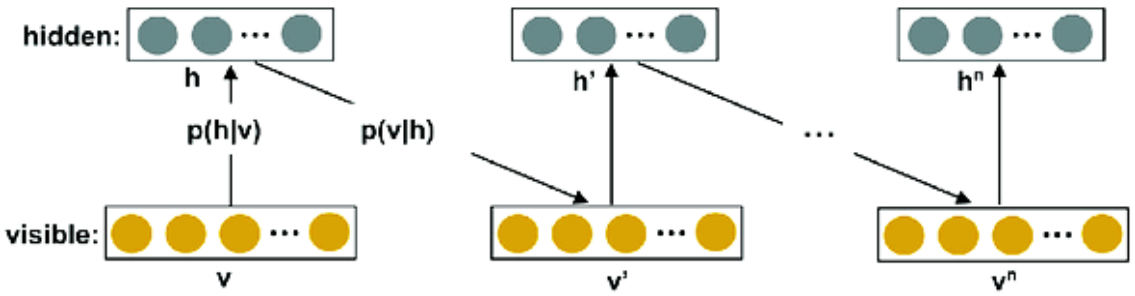


Figure 22: Illustrazione grafica del processo di contrastive divergence per K iterazioni, le unità hidden vengono campionate da quelle visible e viceversa (si hanno quindi K Gibbs Sampling leapfrog step). Al crescere di K cresce la distanza tra i dati originari e quelli campionati, e quindi l'errore di ricostruzione. Si nota quindi un'analogia con i modelli Autoencoder probabilistici, nei quali però non vale il principio di simmetria dei pesi.

che i parametri che identificano univocamente una RBM sono i pesi di interazione W tra le visible units e le hidden units, e i pesi di offset A, B rispettivamente per le visible units e le hidden units. Si può dimostrare che il gradiente esatto della log verosimiglianza, e quindi di massima verosimiglianza, per il peso di interazione w_{ij} si definisce come

$$\frac{\partial \ln(P(\mathbf{v}))}{\partial w_{ij}} = \mathbb{E}_{data}[v_i h_j] - \mathbb{E}_{model}[v_i h_j] = \mathbb{E}_{q(\mathbf{v})P(\mathbf{h}|\mathbf{v})}[v_i h_j] - \mathbb{E}_{P(\mathbf{v},\mathbf{h})}[v_i h_j] \tag{107}$$

Mentre per gli offsets il gradiente esatto risulta:

$$\frac{\partial \ln(P(\mathbf{v}))}{\partial a_i} = \mathbb{E}_{data}[v_i] - \mathbb{E}_{model}[v_i] \tag{108}$$

$$\frac{\partial \ln(P(\mathbf{v}))}{\partial b_j} = \mathbb{E}_{data}[h_j] - \mathbb{E}_{model}[h_j] \tag{109}$$

Dove $\mathbb{E}_{data}, \mathbb{E}_{model}$ rappresentano i valori attesi rispettivamente sotto la reale distribuzione dei dati e sotto il modello. I due valori attesi sono anche denominati fase positiva e fase negativa, o wake phase e sleep phase. I nomi provengono da un algoritmo di ottimizzazione simile, detto algoritmo dormi-veglia (wake-sleep algorithm) e si ricollegano a una interpretazione del gradiente. La fase positiva, o di veglia, è quella in cui il modello è "sveglio" e quindi vede la realtà. Il suo peso dovrebbe aumentare, per avvicinarsi a quello più vicino ai dati. La

fase negativa, o di sonno, è invece quella in cui il modello generativo "sogna", e campiona i dati da se stesso. In questa fase il suo parametro dovrebbe discostarsi dai dati generati. In sintesi, si vede che il segno e la magnitudo del gradiente dipendono da quanto i dati reali si discostano da quelli campionati, e si avvicina a 0 quando sono simili.

Il primo termine, in una RBM, è semplice da ricavare:

$$\mathbb{E}_{data}[v_i, h_j] = v_{in}\mathbb{E}_{model}[h_{jn}|\mathbf{v}_n] \tag{110}$$

Dove v_{in} è il valore dell'i-esima variabile aleatoria osservata per l'istanza n-esima, e h_{jn} è il valore atteso (o un valore campionato) della j-esima hidden unit dato il visible layer.

Il secondo termine invece è intrattabile.

Si può infatti dimostrare che

$$\mathbb{E}_{model}[v_i h_j] = \frac{\partial \ln Z}{\partial w_{ij}} \tag{111}$$

dove Z non è ricavabile in forma chiusa. Attraverso gli algoritmi di contrastive divergence (k-step, mean field e persistent) è però possibile ottenere delle approssimazioni inesatte del valore atteso del visible layer nella fase negativa (attraverso il quale si ricava poi il valore atteso condizionato dell'hidden layer). Si tratta quindi di gradienti di funzioni obiettivo diverse da quelli di log verosimiglianza (e quindi anche da quelli di free energy e divergenza KL) che empiricamente conducono a risultati di training simili con un costo computazionale ridotto.

Di seguito si dimostra la derivazione del gradiente di massima verosimiglianza per un singolo peso di interazione.

Analoghe sono le dimostrazioni per gli offset dei due layer.

Proof 7.2: gradiente esatto della free energy

$$\ln(P(\mathbf{v})) = \ln\left(\sum_{\mathbf{h}\in\mathbb{H}} P(\mathbf{v},\ \mathbf{h})\right) = \ln\left(\frac{1}{Z}\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right) \tag{112}$$

$$\ln\left(\frac{1}{Z}\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right) = \ln\left(\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right) - \ln(Z) \tag{113}$$

Si impongono $L_1 = \ln\left(\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right)$ e $L_2 = \ln(Z)$, e si studiano separatamente la fase positiva $\frac{\partial L_1}{w_{ij}}$ e la fase negativa $\frac{\partial L_2}{w_{ij}}$.
Di seguito la fase positiva (wake phase):

$$\frac{\partial L_1}{\partial w_{ij}} = \frac{1}{\left(\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right)}\left(\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))(-1)\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}\right) \tag{114}$$

$$\frac{\partial L_1}{\partial w_{ij}} = \frac{1}{\left(\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right)}\left(\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))v_i h_j\right) \tag{115}$$

$$\frac{\partial L_1}{\partial w_{ij}} = \sum_{\mathbf{h}\in\mathbb{H}} \frac{\exp(-E(\mathbf{v},\mathbf{h})v_i h_j)}{\left(\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right)} = \sum_{\mathbf{h}\in\mathbb{H}} P(\mathbf{h}|\mathbf{v})v_i h_j \tag{116}$$

$$\sum_{\mathbf{h}\in\mathbb{H}} P(\mathbf{h}|\mathbf{v})v_i h_j = v_i \sum_{\mathbf{h}\in\mathbb{H}} P(\mathbf{h}|\mathbf{v})h_j = v_i \mathbb{E}_{data}[h_j|\mathbf{v}] = \mathbb{E}_{data}[v_i h_j] \tag{117}$$

Di seguito la fase negativa (sleep phase):

$$\frac{\partial L_2}{w_{ij}} = \frac{\partial}{w_{ij}}\ln(Z) = \frac{\partial}{w_{ij}}\ln\left(\sum_{\mathbf{v}\in\mathbb{V}}\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right) \tag{118}$$

$$\frac{\partial L_2}{w_{ij}} = \frac{1}{\left(\sum_{\mathbf{v}\in\mathbb{V}}\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right)}\left(\sum_{\mathbf{v}\in\mathbb{V}}\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))(-1)\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial w_{ij}}\right) \tag{119}$$

$$\frac{\partial L_2}{w_{ij}} = \frac{1}{\left(\sum_{\mathbf{v}\in\mathbb{V}}\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right)}\left(\sum_{\mathbf{v}\in\mathbb{V}}\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))v_i h_j\right) \tag{120}$$

$$\frac{\partial L_2}{w_{ij}} = \sum_{\mathbf{v}\in\mathbb{V}}\sum_{\mathbf{h}\in\mathbb{H}} \frac{\exp(-E(\mathbf{v},\mathbf{h}))v_i h_j}{\left(\sum_{\mathbf{v}\in\mathbb{V}}\sum_{\mathbf{h}\in\mathbb{H}} \exp(-E(\mathbf{v},\mathbf{h}))\right)} \tag{121}$$

$$\frac{\partial L_2}{w_{ij}} = \sum_{\mathbf{v}\in\mathbb{V}}\sum_{\mathbf{h}\in\mathbb{H}} P(\mathbf{v},\mathbf{h})v_i h_j = \mathbb{E}_{model}[v_i h_j] \tag{122}$$

Si è quindi dimostrata la tesi, ovvero:

$$\ln(P(\mathbf{v})) = \frac{\partial L_1}{w_{ij}} - \frac{\partial L_2}{w_{ij}} = \mathbb{E}_{data}[v_i h_j] - \mathbb{E}_{model}[v_i h_j] \tag{123}$$

7.3 KCD: k-step contrastive Divergence

Se fosse possibile ottenere dei campioni dal modello, attraverso una stima Monte Carlo, sarebbe possibile approssimarlo. Una approssimazione possibile potrebbe basarsi sull'operatore di transizione in precedenza definito $T(\mathbf{v}' \leftarrow \mathbf{v})$, che, attraverso una procedura ripetuta di Gibbs Sampling, potrebbe condurci a un'approssimazione del valore atteso del visible layer sotto il modello.

Si ricorda che la transizione $T(\mathbf{v}' \leftarrow \mathbf{v})$ consiste in una operazione in due step:

- nella prima si campiona $\mathbf{h}_0 \sim P(\mathbf{h}|\mathbf{v}_0)$

- nella seconda si campiona $\mathbf{v}_1 \sim P(\mathbf{v}|\mathbf{h}_0)$

L'applicazione iterata di questo operatore rappresenta un processo di campionamento di Gibbs che conduce eventualmente a convergenza.

Il valore atteso di ogni hidden unit si potrebbe allora ricavare sempre attraverso la distribuzione condizionata, dato il visible layer, come nella fase positiva. Il problema è che tale approssimazione richiederebbe un numero molto elevato di iterazioni di Gibbs Sampling, e per ottenere il gradiente esatto della massima verosimiglianza tale numero dovrebbe essere infinito. La buona notizia è che risultati empirici mostrano che gradienti inesatti e approssimati, anche a una sola iterazione di Gibbs Sampling, conducono a buoni risultati, spesso a un numero minore di iterazioni.

Il k-step contrastive divergence fissa un numero k di iterazioni di Gibbs Sampling per passare dal visible layer osservato al visible layer campionato. Per k infinito tale procedura porterebbe a una stima esatta del gradiente di massima log verosimiglianza.

Al crescere di k cresce evidentemente il tempo di calcolo per ottenere il gradiente di una singola osservazione a una singola iterazione di training, ragion per cui la sua scelta richiede di tenere in considerazione un importante trade-off tra performance e costo computazio+nale.

Non è quindi un caso che la scelta più comune sia 1-step CD, e che in letteratura non sia presente quasi nessun esempio di k-step CD con k $>$ 10. Non è però da scartare l'ipotesi di utilizzo di un k elevato. Al crescere di k infatti cresce l'errore di ricostruzione dei dati, e di conseguenza il gradiente risulta molto più severo. Tuttavia, in [Hinton and Salakhutdinov, 2009], Hinton adotta un metodo di crescita graduale del parametro k: per le prime iterazioni utilizza k=1, e verso la fine lo aumenta fino a k=5.

Nel seguente algoritmo è presentata la più semplice versione di training di una generica RBM con gradient descent, con un dataset e un learning rate. Per semplicità sono omesse le procedure di ottimizzazione del training di reti neurali, come SGD, divisione in mini batch, momentum, penalizzazioni, e altri metodi di ottimizzazione discussi nel capitolo 12.

Algorithm 8 Discesa del gradiente con k-step CD

- sono date N osservazioni del layer visibile, $\mathbf{x}_1 ... \mathbf{x}_N$, è definita $\mathcal{L}$ funzione di log verosimiglianza, sono fissati un learning rate α, un numero di iterazioni T e K step di contrastive Gibbs Sampling.
- inizializzazione dei parametri: $W_0[i,j], A_0[i], B_0[j] \sim N(0,1) \forall i,j$
- for $t \in \{1,..,T\}$:
- for $n \in \{1,..,N\}$:
- $\mathbf{v}_0 = \mathbf{x}_n$
- $\mathbf{h}_0 = \mathbb{E}_p[\mathbf{h}|\mathbf{v}_0]$
- for $k \in \{1,..,K\}$:
- $\mathbf{v}_k \sim T(\mathbf{v}' \leftarrow \mathbf{v}_{k-1})$
- endfor
- $\mathbf{h}_K = \mathbb{E}_p[\mathbf{h}|\mathbf{v}_K]$
- $\frac{\hat{\partial \mathcal{L}}}{\partial W} = \mathbf{v}_0 \mathbf{h}_0^{\mathrm{T}} - \mathbf{v}_K \mathbf{h}_K^{\mathrm{T}}$
- $\frac{\hat{\partial \mathcal{L}}}{\partial A} = \mathbf{v}_0 - \mathbf{v}_K$
- $\frac{\hat{\partial \mathcal{L}}}{\partial B} = \mathbf{h}_0 - \mathbf{h}_K$
- $W_t = W_{t-1} - \alpha \frac{\hat{\partial \mathcal{L}}}{\partial W}$
- $A_t = A_{t-1} - \alpha \frac{\hat{\partial \mathcal{L}}}{\partial A}$
- $B_t = B_{t-1} - \alpha \frac{\hat{\partial \mathcal{L}}}{\partial B}$
- endfor
- endfor

7.4 MFCD: Mean Field Contrastive Divergence

La MFCD fu introdotta da Welling e Hinton nel 2002 [Welling and Hinton, 2002]. Si tratta di una semplice modifica del 1-step CD, nel quale al posto di ricampionare il visible layer dopo aver stimato gli hidden (nella prima fase dell'operatore di trainsizione $T(\mathbf{v}' \leftarrow \mathbf{v})$) si prende direttamente il suo valore atteso. In questo modo si ottiene un gradiente ancor meno severo del gradiente con k=1 (in quanto l'errore di ricostruzione è tanto più grande quanto maggiore è k).

Un approssimazione di questo tipo è in letteratura denominata mean-field. Questa leggera modifica consente di risparmiare il costo computazionale di un passo di campionamento (da

ripetersi per ogni osservazione per ogni iterazione di training), rendendo il MFCD molto più veloce ed efficiente del 1-step CD.

In genere, è consigliato l'uso del MFCD come metodo di pre-training, e quello del classico k-step CD come metodo di fine-tuning.

Di seguito l'algoritmo, anche qui nella versione più semplice di gradient descent.

Algorithm 9 Discesa del gradiente con MFCD

- sono date N osservazioni del layer visibile, $\mathbf{x}_1 ... \mathbf{x}_N$, è definita $\mathcal{L}$ funzione di log verosimiglianza, sono fissati un learning rate α e un numero di iterazioni T.
- inizializzazione dei parametri: $W_0[i,j], A_0[i], B_0[j] \sim N(0,1) \forall i,j$
- for $t \in \{1,..,T\}$:
- for $n \in \{1,..,N\}$:
- $\mathbf{v}_0 = \mathbf{x}_n$
- $\mathbf{h}_0 = \mathbb{E}_p[\mathbf{h}|\mathbf{v}_0]$
- $\mathbf{v}_{mf} = \mathbb{E}_p[\mathbf{v}|\mathbf{h}_0]$
- $\mathbf{h}_{mf} = \mathbb{E}_p[\mathbf{h}|\mathbf{v}_{mf}]$
- $\frac{\hat{\partial \mathcal{L}}}{\partial W} = \mathbf{v}_0 \mathbf{h}_0^{\mathrm{T}} - \mathbf{v}_{mf} \mathbf{h}_{mf}^{\mathrm{T}}$
- $\frac{\hat{\partial \mathcal{L}}}{\partial A} = \mathbf{v}_0 - \mathbf{v}_{mf}$
- $\frac{\hat{\partial \mathcal{L}}}{\partial B} = \mathbf{h}_0 - \mathbf{h}_{mf}$
- $W_t = W_{t-1} - \alpha \frac{\hat{\partial \mathcal{L}}}{\partial W}$
- $A_t = A_{t-1} - \alpha \frac{\hat{\partial \mathcal{L}}}{\partial A}$
- $B_t = B_{t-1} - \alpha \frac{\hat{\partial \mathcal{L}}}{\partial B}$
- endfor
- endfor

7.5 PCD: Persistent Contrastive Divergence

La PCD fu introdotta da T.Tieleman nel 2008 [Tieleman, 2008]. Il suo costo computazionale è uguale a quello della 1-step CD. Le sue performance possono tuttavia eguagliare quelle del gradiente esatto. Il metodo consiste nell'inizializzare delle visible units "immaginarie" anche dette "fantasy particles" per ciascuna osservazione, e utilizzarle come punto di partenza dell'operatore di transizione per fare Gibbs Sampling nella fase negativa. Ad ogni iterazione di training, non vengono passati solo i dati osservati, ma anche le unità visibili immaginarie, che vengono aggiornate alla fine dell'iterazione con l'ultimo elemento del processo di campionamento. Il processo MCMC è quindi "persistente" nel corso delle iterazioni, da cui il nome PCD. Inizialmente, la PCD performa peggio rispetto al 1-step CD, ma al crescere

del numero di iterazioni la log verosimiglianza tende ad essere migliore. Almeno dal punto di vista teorico, l'algoritmo di training più efficiente in termini computazionali potrebbe essere basato su una fase iniziale di pre-training, nella quale si adotta MFCD, e su una fase finale di fine-tuning, nella quale si adotta PCD. Per un elevato numero di iterazioni potrebbe comunque essere conveniente l'uso della sola PCD.

Algorithm 10 Discesa del gradiente con PCD

- sono date N osservazioni del layer visibile, $\mathbf{x}_1 ... \mathbf{x}_N$, è definita $\mathcal{L}$ funzione di log verosimiglianza, sono fissati un learning rate α e un numero di iterazioni T.
- inizializzazione dei parametri: $W_0[i,j], A_0[i], B_0[j] \sim N(0,1) \forall i,j$
- inizializzazione delle fantasy particles: $\mathbf{v}^*_{0n} \sim D$ con D con supporto sui dati
- for $t \in \{1,..,T\}$:
- for $n \in \{1,..,N\}$:
- $\mathbf{v}_0 = \mathbf{x}_n$
- $\mathbf{h}_0 = \mathbb{E}_p[\mathbf{h}|\mathbf{v}_0]$
- $\mathbf{v}^*_{tn} \sim T(\mathbf{v}' \leftarrow \mathbf{v}^*_{t-1,n})$
- $\mathbf{h}_{tn} = \mathbb{E}_p[\mathbf{h}|\mathbf{v}^*_{tn}]$
- $\frac{\hat{\partial \mathcal{L}}}{\partial W} = \mathbf{v}_0 \mathbf{h}_0^{\mathrm{T}} - \mathbf{v}^*_{tn} \mathbf{h}^{\mathrm{T}}_{tn}$
- $\frac{\hat{\partial \mathcal{L}}}{\partial A} = \mathbf{v}_0 - \mathbf{v}^*_{tn}$
- $\frac{\hat{\partial \mathcal{L}}}{\partial B} = \mathbf{h}_0 - \mathbf{h}_{tn}$
- $W_t = W_{t-1} - \alpha \frac{\hat{\partial \mathcal{L}}}{\partial W}$
- $A_t = A_{t-1} - \alpha \frac{\hat{\partial \mathcal{L}}}{\partial A}$
- $B_t = B_{t-1} - \alpha \frac{\hat{\partial \mathcal{L}}}{\partial B}$
- endfor
- endfor

8 RSM: Replicated Softmax Model

Il modello Replicated Softmax è stato proposto da Hinton e Salakhutdinov nel 2009 [Hinton and Salakhutdinov, 2009]. Si tratta sia della prima RBM in grado di gestire conteggi di parole, attraverso un visible layer composto da una singola variabile multinomiale. In precedenza si era tentato di adattare RBM con visible layer di variabili Poisson per modellare i conteggi di parole all'interno dei documenti, ma la variabilità delle lunghezze dei testi ne ostacolava il training e il raggiungimento di performance accettabili. Il modello è anche il primo modello a superare in termini di performance, in particolare in termini di perplexity, la LDA di Blei, che dal 2003 ha mantenuto indiscussa il ruolo di topic model per eccellenza. Il vantaggio del modello RSM sui modelli precedenti risiede nella sua capacità di cambiare la distribuzione di probabilità sul vocabolario per testi differenti. Mentre modelli grafici diretti, come le reti bayesiane, mantenevano fissate le probabilità condizionate delle parole dati i topic, nel modello RSM, un modello grafico indiretto, per ogni documento esiste una probabilità condizionata di ogni parola dato un topic differente. Il modello RSM, come la maggior parte dei topic model, rispetta l'assunzione BoW, ovvero non tiene conto dell'ordinamento delle parole nel testo.

8.1 definizioni

Di seguito si introduce la notazione utilizzata nel paper originario del modello RSM, differente da quella finora adottata nella descrizione delle RBM.

Siano dati un dizionario di K parole (o token) e un corpus di N documenti, di indici $n = 1...N$, ciascuno di lunghezza $D_n \in \mathbb{N}$. Si assuma inoltre che ogni documento possa appartenere a F possibili topic, codificati come variabili Bernoulli. Un documento potrebbe allora essere rappresentato come una matrice one-hot encoded $R \in \mathbb{R}^{D_n \text{x} K}$. Tuttavia, per l'assunzione BoW, per cui il modello non considera l'ordinamento dei token, è possibile ridurla a un vettore $\mathbf{v}_n \in \mathbb{R}^K$ di K conteggi v_{kn}, pari alle somme delle colonne di R. Si vede che deve valere $D_n = \sum_{k=1}^{K} v_{kn}$. Per ogni documento n, con rappresentazione BoW $\mathbf{v}_n \in \mathbb{N}^K$, il modello RSM consente di ricavare la rappresentazione semantica dei topic $\mathbf{h} \in \{0,1\}^F$. In altre parole, ogni topic h_j è una variabile binaria, che può rappresentare un cluster a cui un documento può appartenere con una certa probabilità. Bisogna notare che la funzione

di attivazione dell'hidden layer è sigmoidale, per cui le probabilità di appartenenza di un documento ai topic non sommano a 1, ma sono fra loro indipendenti condizionatamente alla rappresentazione BoW del documento (per la proprietà locale di Markov nelle RBM). Più semplicemente, uno stesso documento può appartenere a più topic simultaneamente.

In questo contesto, la funzione di energia del modello RSM è quella che segue:

$$-E(\mathbf{v}, \mathbf{h}) = \sum_{j=1}^{F} \sum_{k=1}^{K} v_k h_j w_{kj} + \sum_{k=1}^{K} v_k a_k + D \sum_{j=1}^{F} h_j b_j \tag{124}$$

Bisogna notare l'importante differenza dalle tipiche funzioni di energia: la lunghezza del documento D viene moltiplicata per gli offset dei topic, in modo tale da evitare che per documenti molto lunghi la magnitudo dei topic non diventi relativamente trascurabile rispetto a quella dei primi due termini.

Si può notare come l'espressione sopra sia equivalente alla funzione di energia classica delle Bernoulli-RBM, applicata alla matrice R come intero layer visibile, nel quale si ha una condivisione dei parametri tra parole in diversa posizione $i = 1...D$. In questa espressione vale $v_{ik} \in \{0, 1\}$:

$$-E(\mathbf{v}, \mathbf{h}) = \sum_{i=1}^{D} \sum_{j=1}^{F} \sum_{k=1}^{K} v_{ik} h_j w_{kj} + \sum_{i=1}^{D} \sum_{k=1}^{K} v_{ik} a_k + \sum_{i=1}^{D} \sum_{j=1}^{F} h_j b_j \tag{125}$$

Si noti la condivisione dei parametri ai token posizionati in punti diversi del testo, da cui l'aggettivo "replicated". Per comodità si continuerà di seguito a trattare un documento come un vettore di conteggi $\mathbf{v}$ anziché come la matrice one-hot R, e si trascureranno quindi gli indici di posizione i.

Probabilità marginale del visible layer (e quindi del documento) e funzione di ripartizione mantengono le stesse espressioni tipiche delle RBM:

$$P(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h} \in \mathbb{H}} \exp(-E(\mathbf{v}, \mathbf{h})) \tag{126}$$

$$Z = \sum_{\mathbf{v} \in \mathbb{V}} \sum_{\mathbf{h} \in \mathbb{H}} \exp(-E(\mathbf{v}, \mathbf{h})) \tag{127}$$

Anche la probabilità condizionata dei topic dati i documenti mantiene la stessa espressione

di quella degli hidden layer binari presenti nelle classiche Bernoulli-RBM:

$$P(h = 1|\mathbf{v}) = \sigma\left(b_j + \sum_{k=1}^{K} w_{kj} v_k\right) \tag{128}$$

dove $\sigma(x) = \frac{1}{1+\exp(-x)}$ è l'attivazione sigmoide.

Diversa è invece l'inferenza sul layer visibile. Essendo esso costituito in realtà da una sola distribuzione multinomiale, con parametro di dimensione D noto, ciò che deve essere stimato è il vettore di probabilità delle parole nel vocabolario:

$$P(v_{ik} = 1|\mathbf{h}) = \frac{\exp\left(a_k + \sum_{j=1}^{F} w_{kj} h_j\right)}{\sum_{k'=1}^{K} \exp\left(a_{k'} + \sum_{j=1}^{F} w_{k'j} h_j\right)} \tag{129}$$

In altre parole, il layer visible presenta una funzione di attivazione softmax, che garantisce la normalizzazione e la somma unitaria delle probabilità condizionate delle parole dati i topic.

8.2 training

Si vede che il modello RSM è univocamente identificato dai parametri di interazione W e di offset A, B, esattamente come le generiche RBM. Il suo training è effettuato con gradient descent e contrastive divergence.

Preso un batch di N documenti, i gradienti medi dei parametri sono i seguenti:

$$\frac{1}{N}\sum_{n=1}^{N} \frac{\partial \ln(P(\mathbf{v}_n))}{\partial w_{kj}} = \mathbb{E}_{data}[v_k h_j] - \mathbb{E}_{model}[v_k h_j] \tag{130}$$

$$\frac{1}{N}\sum_{n=1}^{N} \frac{\partial \ln(P(\mathbf{v}_n))}{\partial b_j} = \mathbb{E}_{data}[h_j] - \mathbb{E}_{model}[h_j] \tag{131}$$

$$\frac{1}{N}\sum_{n=1}^{N} \frac{\partial \ln(P(\mathbf{v}_n))}{\partial a_k} = \mathbb{E}_{data}[v_k] - \mathbb{E}_{model}[v_k] \tag{132}$$

Dove le fasi positive e negative sono ricavate come descritto nel capitolo dedicato alla contrastive divergence.

In particolare, l'operatore di transizione utilizzato dalla catena del Gibbs sampler è basato sulla distribuzione multinomiale, ovvero $T(\mathbf{v}'_n \leftarrow \mathbf{v}_n)$ segue queste due fasi:

- $\mathbf{h}_{0n} \sim Bern(P(\mathbf{h}|\mathbf{v}_{0n}))$

- $\mathbf{v}_{1n} \sim Multinom(P(\mathbf{v}|\mathbf{h}_{0n}), D_n)$

Quando si vuole ottenere il valore atteso del visible layer, e non un campionamento, come nel caso della MFCD, è possibile prendere il prodotto $D_n P(\mathbf{v}|\mathbf{h}_{0n})$.

Nel paper originario del 2009 è stato adottato solo il k-step CD, con k gradualmente incrementato nel corso delle iterazioni di training, fino a un massimo di 5. Il tempo di training nei dataset utilizzati da Hinton e Salakhutdinov andava dalle ore ai giorni (tenuto conto del numero elevato di iterazioni di training: oltre 1000 per ogni dataset).

Una vulnerabilità del modello RSM è quella delle stopwords: quando non rimosse, tendono ad accumulare pesi di interazione elevati con tutti i topic, finendo per danneggiare la capacità del modello di differenziare adeguatamente i topic. Rimedi possibili sono l'applicazione della trasformazione $\ln(1 + x)$ ai valori presenti nella Document-Feature Matrix, e l'introduzione di metriche di penalizzazione sui pesi di interazione.

Nelle sperimentazioni del capitolo 13 si adottano tutti i metodi di ottimizzazione disponibili, inclusi:

- il MFCD e il PCD per la contrastive divergence

- le penalizzazioni L1 e L2 per introdurre sparsità nei pesi di interazione tra hidden e visible unit (poche parole dovrebbero collegarsi a ciascun topic, e la maggior parte dei pesi dovrebbe avvicinarsi a 0)

- tecniche di ottimizzazione introdotte negli anni successivi per le reti feedforward, come Momentum, Adam e RmsProp.

- tecninche di ottimizzazione bayesiana, per ottenere una configurazione ottimale degli iperparametri.

Tali modifiche avranno sperabilmente un impatto positivo e rilevante sia sulle metriche di performance sia sul costo computazionale.

8.3 interpretazione

Il modello Replicated Softmax non consente di ricavare una matrice di probabilità condizionate delle parole dati i topic, come nella LDA. Ciò comporta una perdita in termini di interpretazione, ma è conseguenza della maggiore flessibilità del modello indiretto, che

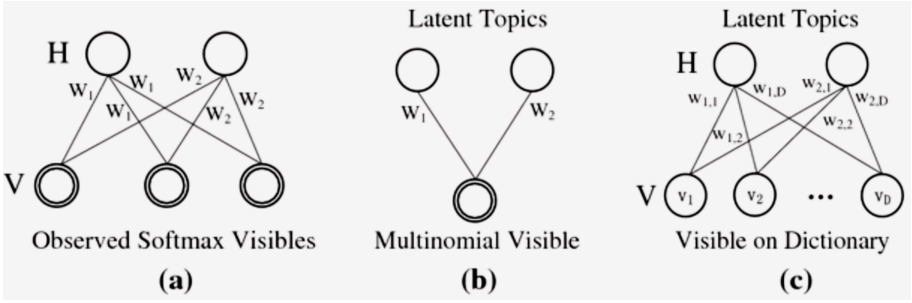


Figure 23: illustrazione grafica del modello RSM
Immagine tratta da [Vogt et al., 2014]

consente di cambiare la relazione di ogni topic con ogni parola in documenti differenti.

La forma del layer multinomiale e la simmetria dei pesi di interazione garantiscono allo stesso tempo flessibilità ed economicità al modello: il costo computazionale dei passi di inferenza e campionamento non cresce al crescere del numero di parole nel testo.

Volendo stabilire un ordine di importanza delle parole per ogni topic, è possibile prendere come proxy delle probabilità congiunte i pesi di interazione. Pesi di interazione w_{kj} più alti corrispondono a una maggiore connessione tra il token k e il topic j. Ordinare i pesi w_{kj} rispetto all'indice k, consente di ricavare la lista ordinata dei token più importanti per il topic j (e viceversa per ogni token).

Quando sono introdotte delle penalizzazioni sui parametri di interazione tra topic e token del modello RSM, la matrice dei pesi W tende a diventare sparsa, e questo può aiutare a differenziare maggiormente i topic. La sparsità dei topic è auspicabile in un topic model. Nella LDA la distribuzione di Dirichlet viene sfruttata proprio per controllarla, e i miglioramenti rispetto al modello pLSI sono ampiamente riconosciuti in letteratura. Un modello RSM che introduce penalizzazioni L1 o L2 sulla matrice W è denominato penalized Replicated Softmax.

Valgono infine tutte le interpretazioni generalmente valide per una qualsiasi RBM: i token sono fra loro condizionatamente indipendenti dati i topic, e i topic insieme forniscono un modello PoE, ossia una distribuzione congiunta, su ciascun token.

9 oRSM: Over Replicated Softmax: una DBM per il topic modeling

Nel 2013, Hinton, Salakhutdinov e Srivastava introductono il modello Over-RSM [Srivastava et al., 2013] Si tratta di una diretta modifica del Replicated Softmax, mirata a sfruttare i vantaggi della struttura delle DBM. L'Over-RSM è certamente più performante del modello RSM, ma sorprende il fatto che ne mantenga lo stesso costo computazionale.

9.1 definizioni

La struttura del Modello Over-RSM si basa su tre layer:

- nel primo si ha il vettore di visible units, associato a una variabile multinomiale, con attivazione softmax e dimensione data dalla lunghezza del testo

- nel secondo si hanno un vettore di hidden units sigmoide, rappresentante i topic

- nel terzo si ha un vettore di hidden units, stavolta softmax, e distribuito come una multinomiale con una dimensione fissata dall'utente, e che quindi rappresenta un iperparametro deciso dall'utente

Importante considerazione è la condivisione dei parametri di interazione simmetrica tra il primo layer e il secondo layer e tra il terzo layer e il secondo layer. Anche i parametri di offset A sono condivisi tra il primo e il terzo layer. In altre parole, dati W, A, B, e dati $\mathbf{v} \in \mathbb{N}^K$ il primo layer (visible multinomiale), $\mathbf{h} \in \{0,1\}^F$ il secondo layer (hidden Bernoulli) e $\mathbf{H} \in \mathbb{N}^K$ il terzo layer (hidden multinomiale) si ha che B è l'offset di $\mathbf{h}$, A è sia l'offset di $\mathbf{v}$ sia l'offset di $\mathbf{H}$, e infine W è sia la matrice di interazione simmetrica tra $\mathbf{v}$ e $\mathbf{h}$ sia tra $\mathbf{h}$ e $\mathbf{H}$.

La funzione di energia può definirsi come segue:

$$-E(\mathbf{v}, \mathbf{h}, \mathbf{H}) = \sum_{k=1}^{K} \sum_{j=1}^{F} w_{jk} h_j (v_k + g_k) + \sum_{k=1}^{K} a_k (v_k + g_k) + (M + D) \sum_{j=1}^{F} b_j h_j \qquad (133)$$

Dove $v_k = \mathbf{v}[k] \in \mathbb{N}, h_j = \mathbf{h}[j] \in \{0,1\}, g_k = \mathbf{H}[k] \in \mathbb{N}$, D è il numero di parole del testo e $M \in \mathbb{N}$ è un parametro fissato, indicante la dimensione della multinomiale sul terzo layer. In questo caso $(M + D)$ ricopre il ruolo che D occupava nel RSM, finalizzato a evitare che gli offset dei topic diventino relativamente trascurabili nella funzione di energia.

La probabilità condizionata del secondo layer (dei topic) si ricava come:

$$P(h_j = 1|\mathbf{h}, \mathbf{H}) = \sigma\left(b_j + \sum_{k=1}^{K} w_{jk}(v_k + g_k)\right) \tag{134}$$

Si può osservare come i layer softmax del primo e del terzo layer coincidono, e si ricavano come:

$$P(H_{ik} = 1|\mathbf{h}) = P(v_{ik} = 1|\mathbf{h}) = \frac{\exp\left(a_k + \sum_{j=1}^{F} w_{kj}h_j\right)}{\sum_{k'=1}^{K} \exp\left(a_{k'} + \sum_{j=1}^{F} w_{k'j}h_j\right)} \tag{135}$$

Si può quindi notare che tale DBM non stima i topic sull'ultimo layer ma sul secondo, come nella RSM classica. Il terzo layer presenta parametri identici a quelli del primo, fatta eccezione per la dimensione della multinomiale, e di fatto può essere visto come una sua duplicazione.

Si vede infatti che la probabilità condizionata del primo layer dato il secondo è sempre uguale alla probabilità condizionata del terzo layer dato il secondo.

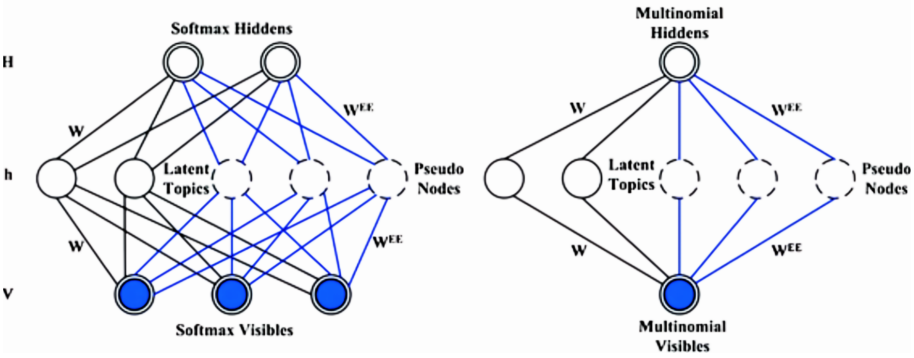


Figure 24: illustrazione grafica del modello Over-RSM
Immagine tratta da [Xu et al., 2017]

9.2 mean field training

In [Srivastava et al., 2013] sono proposti due alternativi metodi di training per il oRSM: Il primo come metodo di training principale, che prevede particolari approssimazioni mean-field iterate, il secondo come metodo di pre-training, di costo computazionale ridotto ma di efficacia non superiore a quella del classico RSM.

Di seguito si presenta il metodo di training principale. Se si volesse applicare un metodo di contrastive learning, sarebbe necessario poter campionare il visible layer da se stesso attraverso un Gibbs Sampler identificato dall'operatore di transizione $T(\mathbf{v}' \leftarrow \mathbf{v})$. Si è visto come tale processo di campionamento di Gibbs consista nel campionare prima le hidden units date le visible e poi le visible units date le hidden. In una DBM tuttavia i layer hidden sono due e sono fra loro interdipendenti, ragion per cui non è possibile trattarle come semplici RBM.

Il gradienti della free energy negativa (o di massima log verosimiglianza) sono i seguenti:

$$\frac{\partial \ln(P(\mathbf{v}))}{\partial w_{kj}} = \mathbb{E}_{data}[(v_k + g_k)h_j] - \mathbb{E}_{model}[(v_k + g_k)h_j] \tag{136}$$

$$\frac{\partial \ln(P(\mathbf{v}))}{\partial a_k} = \mathbb{E}_{data}[v_k + g_k] - \mathbb{E}_{model}[v_k + g_k] \tag{137}$$

$$\frac{\partial \ln(P(\mathbf{v}))}{\partial b_j} = \mathbb{E}_{data}[h_j] - \mathbb{E}_{model}[h_j] \tag{138}$$

Si possono notare le molte analogie con il modello RSM.

Si è visto infatti che per calcolare il valore atteso condizionato dei topic $\mathbf{h}$ (il secondo layer) è necessario disporre sia di $\mathbf{v}$ sia di $\mathbf{H}$.

Non è però possibile ottenere il valore atteso del terzo layer $\mathbf{H}$ senza disporre prima del secondo.

Si può però adottare un'approssimazione di tipo mean field e MCMC per ricavare i valori attesi dei due hidden layer, sfruttando delle equazioni di passaggio di messaggi. Si indicano con μ_j e μ_k, rispettivamente, le probabilità condizionate $P(h_j = 1|\mathbf{v}, \mathbf{H})$ e $P(g_{ij} = 1|\mathbf{h})$.

Si inizializza $\mu_k \sim U(0, 1)$, e si avvia la seguente catena di message-passing equations, che viene ripetuta fino a convergenza di $\mu_j \forall j$:

$$\mu_j \leftarrow \sigma\left(b_j + \sum_{k=1}^{K} w_{jk}(v_k + M\mu_k)\right) \tag{139}$$

$$\mu_k \leftarrow \frac{\exp\left(a_k + \sum_{j=1}^{F} w_{kj}\mu_j\right)}{\sum_{k'=1}^{K} \exp\left(a_{k'} + \sum_{j=1}^{F} w_{k'j}\mu_j\right)} \tag{140}$$

Risultati empirici mostrano come tali equazioni spesso convergano dopo sole due iterazioni. Una volta disponibili, si hanno a disposizione dei valori approssimati dei valori attesi $\hat{\mathbf{h}}$ e $\hat{\mathbf{H}}$, che è possibile utilizzare per campionare il visible layer sotto la distribuzione del modello. Per semplicità, si potrebbe sintetizzare questo processo nell'operatore $\mu(\hat{\mathbf{h}}, \hat{\mathbf{H}} \leftarrow \mathbf{v})$. Volendo definire un operatore di transizione tra due visible layer del tipo $T(\mathbf{v}' \leftarrow \mathbf{v})$, necessario per fare Gibbs Sampling, esso potrebbe comporsi di queste fasi:

Algorithm 11 Definizione dell'operatore di transizione di Gibbs di oRSM

- $\hat{\mathbf{h}}_0, \hat{\mathbf{H}}_0 = \mu(\hat{\mathbf{h}}, \hat{\mathbf{H}} \leftarrow \mathbf{v}_0)$
- $\mathbf{h}_0[j] \sim Bern(\hat{\mathbf{h}}_0[j])\forall j$
- $\mathbf{H}_0 \sim Multinom(\hat{\mathbf{H}}_0, M)$
- $\mathbf{v}_1 \sim Multinom(P(\mathbf{v}|\mathbf{h}_0), D)$
- $\hat{\mathbf{h}}_1, \hat{\mathbf{H}}_1 = \mu(\hat{\mathbf{h}}, \hat{\mathbf{H}} \leftarrow \mathbf{v}_1)$
- $\mathbf{h}_1[j] \sim Bern(\hat{\mathbf{h}}_1[j])\forall j$
- $\mathbf{H}_1 \sim Multinom(\hat{\mathbf{H}}_1, M)$

Bisogna quindi notare che ogni step di Gibbs sampling richiesto da un algoritmo di contrastive divergence richiede di ottenere delle nuove stime di $\hat{\mathbf{h}}, \hat{\mathbf{H}}$. Se si adotta 1-step CD, MFCD o PCD (nei quali la catena di Gibbs si ferma dopo la prima iterazione) il processo di stima sui due Hidden layer va ripetuto due volte. Per k-step CD con k>1, si avranno quindi k+1 iterazioni di approsimazioni Mean-Field, ovvero di adozioni dell'operatore $\mu(\hat{\mathbf{h}}, \hat{\mathbf{H}} \leftarrow \mathbf{v})$.

Si vede che questa procedura di training è computazionalmente più onerosa di quella richiesta dal RSM, e gli autori hanno proposto quindi una procedura di pre-training più economica.

La procedura fin qui descritta è quindi maggiormente consigliabile in fase di fine-tuning.

9.3 un algoritmo di pre-training efficiente

Sempre in [Srivastava et al., 2013] viene proposta una funzione di probabilità condizionata del primo hidden layer del modello oRSM che non richiede l'utilizzo del processo di convergenza $\mu(\hat{\mathbf{h}}, \hat{\mathbf{H}} \leftarrow \mathbf{v})$ e che è calcolabile i forma chiusa in un solo passaggio. Tale probabilità condizionata si basa sull'ipotesi (verosimile ma non scontata, data la condizione

di parameter sharing) che la multinomiale sul visible layer $\mathbf{v}$ e quella sul hidden layer $\mathbf{H}$ presentino parametri di probabilità softmax identiche. Se si fa tale assunzione, è possibile affermare che:

$$P(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \left(1 + \frac{M}{D}\right)\sum_{k=1}^{K} w_{kj}v_k\right) \tag{141}$$

Con questa espressione, è possibile ottenere un operatore di transizione di Gibbs $T(\mathbf{v}' \leftarrow \mathbf{v})$ molto più economico del precedente, e sostenere un costo computazionale equivalente a quello di una RSM anche in fase di training. L'algoritmo di $T(\mathbf{v}' \leftarrow \mathbf{v})$ risulta allora lo stesso del Replicated Softmax.

Come si è già detto, una volta che si è definito tale operatore di transizione, è possibile adottare qualsiasi procedura contrastive learning per addestrare il modello sui dati.

Tale espressione della probabilità condizionata può anche essere adottata in fase di inferenza, a training concluso, per stimare la distribuzione dei topic in un documento a costo computazionale ridotto.

Bisogna notare che l'utilizzo di questa procedura non consente di accedere ai vantaggi della profondità di una DBM, ed è proposto per tanto dagli autori come un modo più efficiente per fare pre-training di oRSM.

L'addestramento del oRSM con questa procedura è equivalente all'addestramento di un RSM con D+M visible unit.

9.4 interpretazione: il vantaggio di una prior

Si vede che importante caratteristica del oRSM è quella di essere una DBM mantenendo lo stesso numero di parametri del RSM, che è a sua volta una RBM.

Tttavia il principale vantaggio del oRSM sul RSM come topic model è la sua capacità di gestire documenti con pochi token. Per $D_n < M$, infatti, si vede come il primo layer esercita un impatto minore sul layer dei topic rispetto al terzo layer.

L'ultimo hidden layer può quindi essere visto come una prior del documento, che risulta tanto più influente quanto maggiore è la carenza di dati in una particolare osservazione.

L'iperparametro M si può quindi interpretare come la forza della prior, ovvero il numero di parole che un documento invisibile, rappresentato dal terzo layer, contiene. Quando il

documento visibile presenta lunghezza $D_n > M$, la prior perde rilevanza, e il topic layer viene condizionato maggiormente dalle visible units.

Va infine notato come il ruolo della prior viene annullato quando si adotta la probabilità condizionata "economica" descritta nel paragrafo precedente. Tale probabilità condizionata implica infatti l'assunzione che il layer softmax del visible layer e del terzo layer siano uguali, rendendo di fatto inutile la presenza del terzo layer.

Quando un oRSM viene addestrato utilizzando solo la procedura di pre-training e in fase di inferenza adotta la probabilità condizionata semplificata, esso diventa assolutamente equivalente al modello RSM.

Tutte le strategie di miglioramento del training del modello RSM sono applicabili anche al modello oRSM.

L'oRSM si presenta quindi come una evoluzione del RSM molto più flessibile e personalizzabile, i cui vantaggi sono ottenuti con un incremento del costo computazionale controllabile.

10 Perplexity per il modello RSM

Nella sezione 4.1 si è descritta la perplexity come misura generale di validazione per un language model. In questo capitolo si vuole evidenziare come tale metrica sia strettamente legata alla log verosimiglianza del modello RSM, e presentare due algoritmi di stima:

- una stima puntuale della perplexity
- una stima dell'upper-bound della perplexity

Si vedrà come il secondo metodo risulti strettamente più semplice e affidabile del primo, ragion per cui lo si adotterà come proxy nella validazione del modello sui dataset di OCTIS, nel capitolo 13.

10.1 relazione con la verosimiglianza

Si ricorda la definizione della perplexity, precedentemente citata, per un qualsiasi modello di topic modeling che rispetti l'assunzione BoW:

$$ppl(D) = \exp\left(-\frac{\sum_{d=1}^{|D|}\sum_{i=1}^{N_d} p(\mathrm{w}_i|W_d)}{\sum_{d=1}^{|D|} N_d}\right) \tag{142}$$

Dove D è un corpus, N_d è la dimensione del d-esimo documento W_d che comprende N_d parole, w_i è la i-esima parola che compare nel documento e p è la probabilità condizionata sotto il modello valutato.

Il problema di stima della perplexity, per ogni parola presente nel testo, si traduce quindi nel problema di stima di $p(\mathrm{w}_i|W_d)$. Se un modello rispetta l'assunzione di indipendenza delle parole condizionatamente ai topic, di indipendenza fra i topic condizionatamente al documento, e di esclusività di un topic per ogni parola, come la LDA, tale probabilità potrebbe essere riscritta come

Sfortunatamente la terza condizione non è soddisfatta nel modello RSM: non è infatti possibile fattorizzare le probabilità condizionali dei topic rispetto al documento, in quanto ogni token non è associato esclusivamente a un topic, ma all'intero hidden layer. In altre parole, non è possibile marginalizzare la probabilità di ogni parola rispetto a un singolo topic, ma per ottenere l'effettiva probabilità marginale di una parola si dovrebbe sommare rispetto

a tutti i possibili vettori dei topic. Il numero di vettori possibili sarebbe 2^F dove F è il numero di topic. In altre parole, per il modello RSM varrebbe

$$p(\mathrm{w}_i|W_d) = \sum_{\mathbf{t}} p(\mathrm{w}_i t|W_d) = \sum_{\mathbf{t}} p(\mathrm{w}_i|\mathbf{t})p(\mathbf{t}|W_d) \tag{143}$$

dove $\mathbf{t}$ rappresenta un vettore di topic, $\mathbf{t} \in \{0,1\}^F$.

Solo questa parte del calcolo risulterebbe infattibile. Sarebbe infatti equivalente calcolare la probabilità del dcumento complessivo condizionatamente a tutti i topic possibili, divisa per la funzione di ripartizione:

$$p(W_d) = \frac{\sum_{\mathbf{t}} p(W_d|\mathbf{t})}{\sum_{\mathbf{v}} \sum_{\mathbf{t}} p(\mathbf{v}|\mathbf{t})} = \frac{\sum_{\mathbf{t}} p(W_d|\mathbf{t})}{Z} \tag{144}$$

Si noti che il numeratore è calcolabile marginalizzando rispetto ai topic: è infatti equivalente riscrivere

$$p(W_d) = \frac{1}{Z} \sum_{\mathbf{t}} p(W_d|\mathbf{t}) = \frac{1}{Z} \left(\sum_{k=1}^{K} \mathrm{a}_k v_k \right) \prod_{j=1}^{F} \left(1 + \exp\left(b_j + \sum_{k=1}^{K} \mathrm{w}_{kj} v_k \right) \right) \tag{145}$$

Dove il numeratore è stato riscritto in base alla formula della log verosimiglianza come free-energy. Si veda a tal proposito la dimostrazione 6.1.

Assumendo di conoscere la funzione di ripartizione Z (invariante al documento), si potrebbe allora calcolare la perplexity puntuale come:

$$ppl(D) = \exp\left(\frac{1}{|D|} \sum_{i=1}^{N_d} \frac{1}{N_d} \ln(p(W_d)) \right) \tag{146}$$

dove il termine $\frac{1}{N_d} \ln(p(W_d))$ è inteso come la log verosimiglianza media per parola nel testo.

10.2 AIS per il modello RSM

Si è visto che, dato Z, è possibile stimare la probabiltà marginale di un documento, e di conseguenza avere una stima puntuale della perplexity. Può di conseguenza essere applicato l'algoritmo AIS, già descritto nella sezione 6.4. Essendo le hidden units del modello RSM

caratterizzate da un layer sigmoidale, i principi dell 'AIS restano gli stessi. Come si è visto, gli elementi richiesti per l'utilizzo di tale algoritmo sono i seguenti:

- una funzione di verosimiglianza con funzione di ripartizione non nota
- una distribuzione proposal con funzione di ripartizione nota
- un operatore di transizione, o leapfrog step, che consente di campionare attraverso il modello un'osservazione vicina a quella data

La funzione di verosimiglianza data in questo caso corrisponde alla free energy del documento $p(\mathbf{v})$, presentata nella sezione 6.2.

L'operatore di transizione è il classico passo leapfrog per cui si campiona prima l'hidden layer dei topic condizionatamente al vettore BoW del documento, e poi lo stesso vettore BoW del documento dal layer softmax condizionato al layer hidden. Tale vettore sarà campionato da una multinomiale con dimensione pari al numero di parole nel documento. La funzione proposal sarà una uniforme $p_b(W_d) = \frac{1}{2^F}$ dove quindi $Z_b = 2^F$ con F numero di topic.

La precisione della stima della perplexity migliora al crescere del numero di passi S. In [Hinton and Salakhutdinov, 2009] Hinton e Salakhutdinov utilizzano $S = 10000$, e mediano le stime su 100 iterazioni di AIS per ogni documento. Avendo infatti ogni documento una diversa lunghezza, la stima di Z non è comune a tutti i documenti, ma specifica per ogni documento di lunghezza N_d. Il prezzo compuazionale di tale stima è quindi rilevante, e può risultare insostenibile per un elevato numero di topic e di documenti. Stime leggermente distorte della funzione di partizione possono facilmente condurre a stime della perplexity approssimativamente nulle o superiori alle stime dell'upper-bound, e di conseguenza prive di affidabilità. Di conseguenza, nella validazione del modello RSM si fa abitualmente ricorso all'upper bound della perplexity.

10.3 Approssimazione della perplexity per il modello RSM

Una semplice approssimazione di tipo mean field della perplexity può essere ricavato a partire dal seguente lower bound della verosimiglianza:

$$p(W_d) \approx p(W_d|\mathbb{E}(\mathbf{h}|\mathbf{v}_d)) = p^*(W_d) \tag{147}$$

Dove $\mathbb{E}(\mathbf{h}|\mathbf{v}_d)$ corrisponde all'attivazione sigmoidale dei topic condizionatamente al documento di cui si vuole calcolare la probabilità marginale.

La perplexity di un corpus di documenti D si valuta allora attraverso la funzione già vista:

$$ppl(D) \approx \exp\left(\frac{1}{|D|}\sum_{i=1}^{N_d}\frac{1}{N_d}\ln(p^*(W_d))\right) = ppl_{upbo}(D) \tag{148}$$

Negli esperimenti sui dataset di OCTIS si adotterà tale stima, per ragioni di tempo e di affidabilità.

11 Ottimizzazione di Gradient Descent

L'adozione dell'algoritmo di discesa del gradiente nella procedura di training accomuna le RBM alle reti neurali feedforward. Tra queste ultime, dal 2010 fino ad oggi, hanno ottenuto vasta popolarità degli algoritmi di ottimizzazione del gradiente, utili principalmente a tre scopi:

- riduzione del rischio, tipico dell'ottimizzazione con gradient descent, di concludere il training in punti di minimo locale, perdendo così le soluzioni globalmente ottime del problema di ricerca

- ottimizzazione del rapporto tra exploration ed exploitation

- velocizzazione del processo di training, con conseguente riduzione del costo computazionale

Tra questi algoritmi, hanno in particolare raggiunto vasta diffusione sgd, minibatch-gd, momentum, adagrad, rmsprop e adam. Non c'è ragione di dubitare che la loro adozione nel processo di training delle RBM posso condurre a miglioramenti di performance e costo computazionale.

In secondo luogo, trattandosi di modelli parametrici con elevata dimensionalità, le RBM possono trarre vantaggio da metodi di penalizzazione di tipo L1 o L2. Quando le penalizzazioni sono applicate ai parametri di interazione del modello RSM, esso viene anche denominato penalized-RSM.

11.1 Stocastic Gradient Descent e Mini batch Gradient Descent

I più semplici metodi di ottimizzazione del gradient descent sono probabilmente il metodo sgd e il minibatch-gd. La procedura di stocastic gradient descent si differenzia da quella di full gradient descent per l'ordine con cui le N osservazioni presenti nei dati di training vengono utilizzate, in ogni epoca (iterazione di addestramento), per calcolare il gradiente dei parametri e aggiornarli. Nel full-gd, ovvero nel gradient descent nella sua forma più elementare, il loro ordinamento è fissato. Invece, nel sgd, all'inizio di ogni epoca tutti i dati di training vengono mischiati in maniera casuale.

Il minibatch-gd rappresenta invece una forma di aggregazione di più osservazioni in una sola: anziché aggiornare i parametri del modello per ogni singola osservazione, si aggregano molte osservazioni in un gruppo (detto batch) di numerosità fissa (la dimensione del batch,

che è un iperparametro). Per ogni batch vengono calcolati tutti i gradienti delle osservazioni al loro interno e solo la loro media viene utilizzata per aggiornare effettivamente i pesi del modello.

Sia sgd sia minibatch-gd conducono a soluzioni più robuste rispetto a minimi locali in un minor numero di iterazioni. Quando vengono utilizzate insieme si parla di stocastic minibatch gradient descent.

11.2 Learning rate

Il learning rate è un fattore, tipicamente compreso tra 0.1 e 0, per cui il gradiente di ogni parametro viene moltiplicato prima di essere sommato (se si è alla ricerca di un massimo) o sottratto (se si è alla ricerca di un minimo) al parametro corrispondente per ogni iterazione di training. Il learning rate è presente in tutte le procedure di gradient descent

$$w_t = w_{t-1} + \alpha \frac{\partial L}{\partial w_{t-1}} \tag{149}$$

11.3 momentum

Il momentum è un fattore (iperparametro) compreso tra 0 e 1 che definisce una media pesata tra il gradiente ad una data iterazione e il gradiente all'iterazione precedente. Si può dire che attraverso il momentum il gradiente diventa una media mobile che evolve nel corso delle iterazioni di training. L'utilizzo del momentum riduce il livello di rumore e stabilizza la direzione del gradiente.

In formule, dato m il momentum, α il learning rate, w il parametro da ottimizzare:

$$G_t = mG_{t-1} + (1-m)\frac{\partial L}{\partial w_{t-1}} \tag{150}$$

$$w_t = w_{t-1} + \alpha G_t \tag{151}$$

In questo modo si riducono le oscillazioni e la varianza del gradiente, favorendo una più rapida convergenza.

11.4 AdaGrad

AdaGrad divide il learning rate per la radice della somma dei quadrati dei gradienti. In questo modo, solo alcuni pesi vengono effettivamente aggiornati in misura rilevante, e si ottiene un learning rate differente per ogni iterazione.

$$S_t = \sum_i \left(\frac{\partial L}{\partial w_{it-1}} \right)^2 \tag{152}$$

$$w_{it} = w_{it-1} + \alpha \frac{1}{\sqrt{S_t + \varepsilon}} \frac{\partial L}{\partial w_{it-1}} \tag{153}$$

Dove ε è una costante molto piccola, necessaria pre prevenire eventuali divisioni di 0. Nell'implementazione in OCTIS è fissata a 1e-8.

AdaGrad tende a convergere molto rapidamente a una soluzione, ragion per cui è preferibile nei casi in cui si fissa un numero molto basso di epoche di training. Soffre infatti del rischio che il learning rate si annulli, e che l'aggiornamento si blocchi.

11.5 RMSProp

RMSProp rappresenta un'evoluzione di AdaGrad, includendo un fattore di decadimento simile al momentum, e utilzza la media dei quadrati dei gradienti al posto della somma. Attraverso il termine di decadimento sulla media dei quadrati dei gradienti, RMSProp previene il problema di AdaGrad per cui il learning rate adattivo tende ad annullarsi.

$$M_t = \frac{1}{|W|} \sum_{i=1}^{|W|} \left(\frac{\partial L}{\partial w_{it-1}} \right)^2 \tag{154}$$

$$G_t = \gamma M_{t-1} + (1 - \gamma) M_t \tag{155}$$

$$w_{it} = w_{it-1} + \alpha \frac{1}{\sqrt{G_t + \varepsilon}} \frac{\partial L}{\partial w_{it-1}} \tag{156}$$

11.6 Adam

Adam [Kingma, 2014] sta per Adaptive Moment, e mira a combinare i vantaggi del momentum (la riduzione delle oscillazioni) con quelli del RMSProp (la presenza di un learning rate adattivo). Presenta due fattori di momentum, uno per il gradiente e uno per il suo quadrato. Nella loro stima presenta anche una fase di correzione del bias.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left(\frac{\partial L}{\partial w_{it-1}} \right) \tag{157}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\partial L}{\partial w_{it-1}} \right)^2 \tag{158}$$

$$M = \frac{m_t}{1 - \beta_1^t} \tag{159}$$

$$V = \frac{v_t}{1 - \beta_2^t} \tag{160}$$

$$w_{it} = w_{it-1} + \alpha \frac{1}{\sqrt{V + \varepsilon}} M \tag{161}$$

11.7 penalizzazioni L1 e L2

La penalizzazione dei modelli parametrici è vastamente discussa. Le sue prime applicazioni riguardano i modelli di regressione lineare penalizzati, detti Ridge (per la penalizzazione L2) e Lasso (per la penalizzazione L1), introdotti da Hastie e Tibshirani. In materia di reti neurali feedforward, Yoshua Bengio è il primo propositore, ma le sue teorie sono estendibili anche alle reti di Markov e alle RBM. Le penalizzazioni consistono in modifiche della funzione di perdita, dalle quali consegue una modifica della funzione del gradiente. La penalizzazione L1 consiste nell'incrementare una data funzione di perdita L per la somma dei moduli dei parametri del modello, moltiplicati per un fattore di importanza:

$$pL = L + \gamma \sum_{w \in W} |w| \tag{162}$$

dove W è l'insieme dei parametri di interazione del modello con funzione di perdita L. Bisogna notare che non è mai consigliato penalizzare i parametri legati a una singola variabile aleatoria, ovvero i bias e le intercette. Di conseguenza, gli unici parametri interessati a questa

modifica nel modello RSM e in qualsiasi altra RBM dovrebbero essere quelli che legano una unità visibile con una unità latente. Nel caso della penalizzazione L2, al posto dei moduli si hanno dei quadrati:

$$pL = L + \gamma \sum_{w \in W} w^2 \tag{163}$$

In entrambi i casi il parametro γ è un iperparametro positivo, che si assume 0 quando si omette qualsiasi penalizzazione. Attraverso i moduli e i quadrati si garantisce la positività della somma, e di conseguenza la funzione peggiorativa della modifica. Bisogna notare che nel caso in cui L non sia una funzione di perdita, ovvero una funzione di costo da minimizzare, ma una funzione da massimizzare, come ad esempio una funzione di verosimiglianza dei dati, il termine di penalizzazione andrebbe sottratto alla funzione di perdita originaria. Il modello Replicated Softmax penalizzato, detto penalized-RSM, è considerato in generale più performante del modello RSM classico. La ragione è la stessa che rende la LDA notevolmente superiore rispetto al pLSI: la sparsità dei dati testuali. Si può dire che le prior Dirichlet stanno alla LDA come le penalizzazioni dei pesi di interazione stanno al Replicated Softmax. La penalizzazione rende infatti sparsa la matrice dei pesi di interazione, che rappresentano la forza con cui ciascun token si lega a ciascun topic. Aumentare il fattore di penalizzazione γ oltre un certo limite può condurre a un annullamento di tutti i parametri, ma se opportunamente fissato può rendere la matrice dei pesi sparsa (ovvero prevalentemente composta da valori approssimativamente nulli) e fornire quindi un maggior grado di interpretabilità al modello.

12 Ottimizzazione Bayesiana in OCTIS

L'ottimizzazione bayesiana è un'area centrale dell'autoML (Automated Machine Learning). Si tratta di uno strumento per ottimizzare gli iperparametri di un modello di apprendimento automatico, rispetto a una determinata misura di costo.

Quasi ogni modello statistico, fatta eccezione per alcuni modelli molto semplici o di tipo non parametrico, richiede all'utente di specificare degli iperparametri prima del suo adattamento ai dati. Tali iperparametri possono a volte essere stabiliti in base al buonsenso, alla conoscenza del dominio dei dati da parte dell'utente o alla conoscenza generale del modello utilizzato. Spesso tuttavia, la loro scelta dipende in buona parte dall'esito che hanno avuto su una funzione di perdita scelta. In passato era prassi comune provare diverse combinazioni di iperparametri, con procedure di grid search, per selezionare quella che conduceva ai migliori risultati. L'avvento di procedure di training che richiedono un elevato costo computazionale, a causa della complessità del modello, con un elevato numero di parametri e iperparametri, e della complessità dei dati, con un elevato numero di osservazioni e di features, ha reso impraticabili tali procedure di ricerca.

Il nome, "ottimizzazione bayesiana", deriva dall'approccio generale della statistica bayesiana: data una conoscenza a priori dei risultati di diversi esperimenti riguardo all'impatto che differenti combinazioni degli iperparametri del modello hanno avuto sulla funzione di perdita, si vuole scegliere la nuova combinazione di iperparametri tale da:

1. minimizzare la funzione di perdita considerata
2. migliorare la capacità di uno stimatore di prevedere il valore della funzione di perdita a partire da una arbitraria combinazione di iperparametri

Un algoritmo di ottimizzazione bayesiana, nella formulazione classica, comprende le seguenti componenti:

- una misura di perdita $L = L(h)$ con $h \in H$, $L : H \to \mathbb{R}$
- un modello surrogato $F(h)$ in grado di restituire uno stimatore $\hat{L}(h)$ e una stima della varianza associata $\hat{Var}(\hat{L}(h))$. Si tratta quindi di un modello di Machine Learning di

tipo supervisionato (di regressione), e comprendente una stima della varianza delle sue stime (e.g. Random Forest e Processi Gaussiani). La valutazione del modello surrogato in un punto non dovrebbe comportare un elevato costo computazionale.

- una funzione di acquisizione $G : H \to \mathbb{R}$ con $G(F(h)) = g \in \mathbb{R}$. Tale funzione fornisce uno score di "convenienza" nella scelta degli iperparametri h: non si tratta di una mera applicazione della regola per cui si sceglie $h = \text{argmin}_h \hat{L}(h)$, ma di una funzione che tiene conto sia del valore atteso di $\hat{L}(h)$ sia della sua varianza, in modo tale da fornire un equilibrio tra l'ottimizzazione degli iperparametri e l'esplorazione del modello, ovvero la raccolta di nuove osservazioni $L(h)$ in regioni meno note di H. Le funzioni di aquisizione non comportano un elevato costo computazionale.

- un algoritmo iterativo di ottimizzazione, applicabile su G. la sua funzione è quella di massimizzare G su tutto lo spazio H, e quindi di specificare esattamente la combinazione ottimale degli iperparametri da fornire al modello. Essendo le funzioni di acquisizione (e quindi i modelli surrogati) funzioni semplici da valutare, l'algoritmo di ottimizzazione dovrebbe portare rapidamente a una soluzione. Dalla scelta del modello surrogato in genere deriva la scelta di tale algoritmo (e.g. per un modello surrogato Random Forest, la scelta di un algoritmo genetico è quasi obbligata).

Se si riconduce il problema di ottimizzazione generico a un problema di tuning degli iperparametri di un modello di machine learning, la funzione di perdita si traduce nelle seguenti componenti:

- un dataset comprendente un set di osservazioni, o istanze, x.

- un algoritmo di machine learning con parametri w, addestrabile con un set di iperparametri $h \in H$, in grado di restituire un output $y = M(x; w)$.

- una misura di perdita $L = L(x; M(x; w))$ che può essere vista, indirettamente, come una funzione degli iperparametri $L = L(h) + u$ con u errore casuale non dipendente da h. La misura di costo da ottimizzare può essere arbitrariamente scelta, e può essere anche diversa rispetto a quella minimizzata dal modello nel corso dell'addestramento.

In altre parole, l'ottimizzazione bayesiana mira non solo ad automatizzare, ma anche a rendere più efficiente, esaustiva e precisa la ricerca degli iperparametri da parte di chi adotta un qualsiasi modello di machine learning.

Di seguito si riporta una descrizione più dettagliata delle componenti di un processo di ottimizzazione bayesiana.

12.1 Modelli Surrogati

12.1.1 Random Forest

Il Random Forest (RF) [Breiman, 2001] [Hastie et al., 2008] è un modello di ensemble learning degli alberi di decisione. Come questi, può essere utilizzato sia per prevedere variabili continue (regressioni) sia per prevedere variabili categoriali (e quindi per problemi di classificazione). Essendo praticamente tutte le funzioni di perdita funzioni continue, in materia di ottimizzazione bayesiana si parla sempre di RF per la regressione.

Nei problemi di regressione, il modello RF addestra diversi alberi di regressione, e la sua stima finale è la media delle loro previsioni.

Un albero di regressione è un modello predittivo basato su un criterio "divide et impera" applicato ricorsivamente a sottoinsiemi di osservazioni.

In fase di previsione, ogni osservazione viene passata dai nodi genitori ai nodi figli, a partire dal nodo root e a finire nel nodo foglia. La media delle osservazioni facenti parte di questo gruppo fornisce la previsione del modello.

Il modello Random Forest applica una media delle previsioni di numerosi alberi di regressione, introducendo due elementi di differenziazione:

- un campionamento casuale delle osservazioni s (bootstrap)
- un campionamento casuale dei regressori m

Se $m = p$ dove p è il numero dei regressori, ovvero se tutti i regressori sono inclusi nel modello, il modello Random Forest è equivalente a un'applicazione del bagging a un numero arbitrario di alberi (strategia di ensemble learning nella quale si applica solo il primo campionamento).

Avendo non una sola stima, bensì un vettore di stime, il modello RF consente di prevedere anche la varianza di ogni singola previsione. Questa caratteristica lo rende utile per problemi di ottimizzazione bayesiana, dove la funzione di acquisizione richiede non solo la stima puntuale della funzione di perdita ma anche la sua varianza, dati gli iperparametri.

All'interno di Octis, il modello RF utilizza 100 alberi di regressione, ciascuno dei quali utilizza tutti i regressori disponibili, con un criterio di pruning tale per cui ogni albero continua a costruire nuovi nodi finché non raggiunge un numero minimo di osservazioni presenti in ogni split pari a 3. Il modello utilizzato da Octis è la funzione RandomForestRegressor della libreria python scikit-learn.

Algorithm 12 pseudocodice di un albero di regressione

L'albero di regressione segue queste fasi di training:
- inizialmente, tutte le osservazioni sono assegnate al nodo root
- per ogni iterazione di training:
- per ogni nodo / gruppo dell'albero:
- per ogni variabile / feature X_j:
- per ogni valore x_{ji}:
- divido il gruppo corrente in due gruppi, sopra e sotto il valore soglia scelto
- calcolo la media della variabile obiettivo nei due gruppi, $\hat{y_{left}}$ e $\hat{y_{right}}$.
- calcolo la funzione di perdita $L(y, \hat{y})$
- endfor
- endfor
- scelgo il valore soglia $x_j i$ tale da minimizzare $L(y, \hat{y})$
- endfor

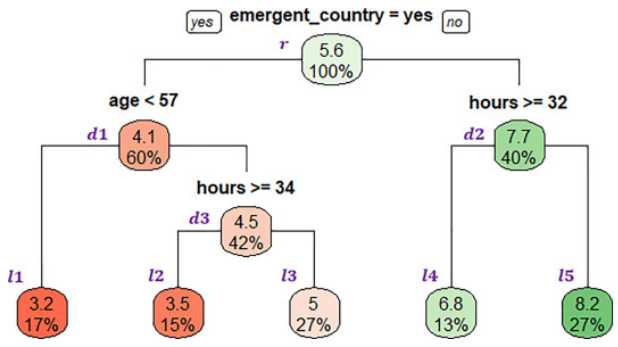


Figure 25: Esempio di albero di regressione in R
in ogni nodo è presente la media della variabile obiettivo, con la variabile di input e il corrispondente valore soglia utilizzati per eseguire lo split.

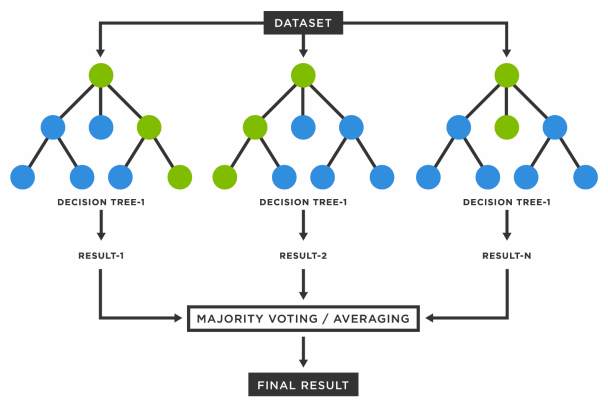


Figure 26: Illustrazione grafica del Random Forest

12.1.2 Extra Trees

La regressione Extra Trees è un altro modello supervisionato disponibile in Octis per stimare valore atteso e varianza della funzione di perdita condizionatamente ai valori degli iperparametri del modello. Si tratta di un algoritmo che introduce diverse modifiche al modello RF e agli alberi di regressione che utilizza. Le sue performance in termini di previsione non si discostano molto da quelle del modello RF, tuttavia presenta un costo computazionale minore. La differenza principale dell'Extra Trees (Extremely Randomized Trees) rispetto al Random Forest (RF) risiede nel suo approccio di divisione dei nodi. Mentre nel RF si cerca il miglior split possibile tra un sottoinsieme casuale di feature, l'Extra Trees introduce un ulteriore livello di randomizzazione. Per ogni feature candidata, esso seleziona in modo casuale una soglia di split tra il suo minimo e massimo valore osservato nel nodo, anziché cercare la soglia ottimale (come il valore che massimizza la riduzione della varianza). Questa soglia casuale t_k per la feature k-esima è tipicamente campionata in modo uniforme nell'intervallo $(min(x_k), max(x_k))$ tra i punti dati nel nodo. Il migliore tra questi split casuali (valutato secondo il criterio di riduzione dell'errore) viene poi scelto per partizionare il nodo. Pertanto nell'Extra Trees, a differenza del RF, la funzione di costo non viene minimizzata tramite una ricerca esaustiva, ma semplicemente valutata per un insieme predefinito di coppie (feature, soglia casuale).

Le randomizzazioni introdotte sono quindi due:

- la selezione casuale di un sottoinsieme di feature per ogni nodo (come nel RF)

- la selezione casuale della soglia di split per ciascuna feature

Di conseguenza si riduce drasticamente la varianza del modello, poiché decrementa la correlazione tra gli alberi dell'ensemble senza aumentare eccessivamente il bias di ciascun albero individuale.

La procedura di split è molto più veloce rispetto a RF. Poiché non è necessaria una ricerca esaustiva per la soglia ottimale, il costo computazionale per nodo scende da $O(mn\log(n))$ a $O(m)$ (dove m è il numero di feature candidate e n il numero di campioni). Questo rende l'Extra Trees particolarmente vantaggioso su dataset di grandi dimensioni o con feature ad alta cardinalità.

Infine l'elevata randomizzazione agisce come un meccanismo di regolarizzazione naturale. Gli alberi individuali, essendo meno adattati ai dettagli rumorosi del training set, sono più "deboli" ma anche meno correlati.

In conclusione, la regressione Extra Trees può essere vista come una versione più "rumorosa" e computazionalmente efficiente del Random Forest. A livello pratico, in materia di ottimizzazione bayesiana di Topic Models, il ridotto numero di osservazioni e di feature rende più conveniente l'adozione di un modello più preciso anche se a maggior costo computazionale, come il RF. Si vede nelle sezioni successive che il modello tuttavia più usato per la scelta degli iperparametri risulta il Gaussian Process (GP) definito nel prossimo capitolo.

12.1.3 Processi Gaussiani

Un processo gaussiano è un modello di regressione non parametrico basato sull'ipotesi che i dati seguano un processo stocastico. In geostatistica prende anche il nome di Kriging, con alcune specificazioni. Il modello sfrutta la definizione della Normale Multivariata per modellare la distribuzione congiunta delle osservazioni di training e delle nuove osservazioni per le quali la variabile obiettivo deve essere prevista.

Sia dato un problema di regressione definito come segue

$$y_i = f(x_i) + e_i$$

La distribuzione congiunta delle osservazioni di training e dei nuovi dati è quella che

segue:

$$\begin{bmatrix} Y \\ Y^* \end{bmatrix} = \begin{bmatrix} f(X) \\ f(X^*) \end{bmatrix} + \begin{bmatrix} e \\ e^* \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k(X,X) + \sigma^2 I & k(X,X^*) \\ k(X^*,X) & k(X^*,X^*) + \sigma^2 I \end{bmatrix} \right)$$

Dove i 4 termini della matrice di covarianza sono 4 sottomatrici. In particolare, la funzione k è una funzione kernel fornita dall'utente. Applicata a una matrice di osservazioni, restituisce una matrice di distanze tra le stesse. In questa definizione, X è la matrice del disegno, per la quale la variabile risposta Y è osservata, mentre X^* è la matrice delle nuove osservazioni, o il vettore corrispondente a una nuova singola osservazione, per la quale si vuole stimare Y^*.

In un processo gaussiano, la posterior della variabile da prevedere è quella che segue:

$$P(Y^*|Y, X, X^*) \sim N(\mu, \Sigma) \tag{164}$$

e dalla stessa è possibile stimare i momenti di tale variabile. In particolare il valore atteso si definisce come:

$$\mu^* = k(X^*, X)(k(X,X) + \sigma^2 I)^{-1} Y \tag{165}$$

mentre la matrice di covarianza è quella che segue:

$$\sigma^* = k(X^*, X^*) + \sigma^2 I - k(X^*, X)(k(X,X) + \sigma^2 I)^{-1} k(X, X^*) \tag{166}$$

La funzione kernel k è tipicamente definita dall'utente, ma l'opzione più comune è quella della Radial Basis Function (RBF kernel):

$$k(X_1, X_2) = RBF(X_1, X_2) = \exp\left(-\frac{|X_1 - X_2|^2}{2\sigma^2} \right) \tag{167}$$

Dall'adozione di tale kernel consegue il fatto che sulle previsioni del modello pesino maggiormente le osservazioni vicine all'osservazione di cui si deve prevedere la variabile risposta, nello spazio delle variabili di input.

Se si vuole evitare che il processo gaussiano interpoli esattamente i dati, ma conduca a

una soluzione più robusta, tipicamente si utilizza il seguent kernel (in scikit-learn chiamato white noise kernel):

$$k(X_1, X_2) = RBF(X_1, X_2) + \sigma^2 \tag{168}$$

Implementazioni pratiche ddi un processo gaussiano devono inoltre tenere conto del problema di inversione della matrice

$$A = (k(X, X) + \sigma^2 I)^{-1} \tag{169}$$

richiesta dalle funzioni per la stima della varianza e della media di ciascuna istanza. Si noti qui che $RBF(X, X)$ è la matrice quadrata delle distanze fra le N osservazioni della matrice del disegno X. Per questa ragione si ricorre alla Decomposizione di Cholesky della matrice stessa per ricavare l'inversa più rapidamente. La decomposizione di Cholesky semplifica l'inversione di una matrice semidefinita positiva, trasformandola in una matrice triangolare superiore per la quale il calcolo dell'inversa è computazionalmente più semplice e numericamente più stabile.

La decomposizione di Cholesky consente di ricavare L matrice triangolare tale che

$$A = LL^T \tag{170}$$

da cui consegue che

$$A^{-1} = (LL^T)^{-1} = (L^{-1})^T L^{-1} \tag{171}$$

dove essendo L triangolare è possibile ricavare L^{-1} in modo semplice.

Se si definisce

$$P = (L^{-1})^T \tag{172}$$

allora la matrice di covarianza di tutte le osservazioni si definisce come

$$A^{-1} = P^T P \tag{173}$$

Il vettore dei pesi di ciascuna osservazione si definisce come

$$\alpha = A^{-1}Y = (k(X,X) + \sigma^2 I)^{-1}Y \tag{174}$$

da cui è quindi possibile ricavare le previsioni del processo gaussiano:

$$\mu^* = \sum_{i=1}^{N} k(X^*, X_i)\alpha_i = k(X^*, X)A^{-1}Y \tag{175}$$

$$\sigma^* = k(X^*, X^*) + \sigma^2 I - k(X^*, X)A^{-1}k(X, X^*) \tag{176}$$

A titolo esemplificativo, di seguito si presenta uno pseudocodice:

Algorithm 13 pseudocodice di un Processo Gaussiano

- FASE DI TRAINING
- Sia data una collezione $(X,Y) = (X_i, Y_i)_{i=1}^N$ con $X_i \in \mathbb{R}^p$, $Y_i \in \mathbb{R}$, una funzione kernel $k : \mathbb{R}^{2p} \to \mathbb{R}_+$, e un iperparametro $\lambda \geq 0$.
- $A = (K(X,X) + \sigma^2 I)^{-1}$

dove $K(X,X)_{ij} = k(X_i, X_j) \forall i,j = 1...N$

- $L = chol(A)$ ovvero si ricava $L : A = LL^T$
- $P = (L^{-1})^T$
- $A^{-1} = P^T P$
- $\alpha = A^{-1}Y$
- FASE DI PREVISIONE
- Sia data una nuova osservazione X^* di cui si vuole prevedere Y^*
- $\mathbb{E}(Y^*|X*, X, Y) = \mu^* = \sum_{i=1}^N k(X^*, X_i)\alpha_i$
- $Var(Y^*|X*, X, Y) = \sigma^* = k(X^*, X^*) + \sigma^2 I - K(X^*, X)A^{-1}K(X, X^*)$

dove $K(X^*, X)_i = k(X^*, X_i)$

12.2 Funzioni di acquisizione

In Bayesian Optimization and Data Science [Archetti and Candelieri, 2019b] Candelieri e Archetti forniscono una lista delle funzioni di acquisizione tradizionali, mettendole a confronto con altre più nuove emerse in anni recenti. Le prime comprendono:

- PI (Probability of Improvement)
- EI (Expected Improvement)
- UCB (Upper Confidence Bound)

- LCB (Lower Confidence Bound)

Mentre tra le funzioni di acquisizione più recenti ricordano:

- Scaled Expected Improvement (2018)
- GP-Hedge (2010)
- Thompson Sampling (1933), applicabile anche come modello surrogato
- una famiglia di funzioni di acquisizone basate sul concetto di entropia (2009-20019)
- Knowledge gradient
- Look-Ahead, anch'essa una famiglia di funzioni pensate per ottimizzare non solo la prossima iterazione di BO ma tutte quelle che seguono (2009-2016)
- Sequentially Bayesian K-optimal (2018)

All'interno di Octis sono disponibili solo le funzioni di acquisizione più tradizionali, fatta eccezione per UCB.

Di seguito si definiscono solamente le proprietà delle funzioni di acquisizione disponibili in Octis.

12.2.1 Lower Confidence Bound

Sia dato ϵ un parametro arbitrario, e le funzioni $\mu(x)$ e $\sigma(x)$ i valori restituiti dal modello surogato per una combinazione degli iperparametri x del modello definita sullo spazio di ricerca. Allora la funzione di acquisizione dell'estremo inferiore di confidenza si definisce come:

$$\mathrm{LCB}(x) = \mu(x) - \epsilon\sigma(x) \tag{177}$$

L'Upper Confidence Bound presenta una definizione speculare:

$$\mathrm{UCB}(x) = \mu(x) + \epsilon\sigma(x) \tag{178}$$

Si nota che la funzione LCB presenta per costruzione valori inferiori. Per problemi di minimizzazione si adotta LCB, mentre per problemi di massimizzazione si utilizza UCB. La combinazione selezionata sarà quindi il minimo osservato di LCB, mentre sarà il massimo per la funzione UCB. Si noti che al crecere di ϵ cresce il livello di exploration mentre si riduce quello di exploitation (che è massima per $\epsilon = 0$). In OCTIS si adotta sempre la LCB (ogni problema di massimizzazione è convertibile in un problema di minimizzazione cambiando di segno la funzione obiettivo).

Tra le funzioni di acquisizione tradizionali, la LCB è quella che lascia più spazio alla componente di exploration del processo di ricerca.

12.2.2 Probability of Improvement

Si definisca $g(x) = -L(x)$ la funzione obiettivo da massimizzare, e $g(x^*)$ il valore più alto di essa finora osservato. La probabilità di miglioramento si definisce come:

$$\mathrm{PI}(x) = P(g(x) \leq g(x^*) + \epsilon) = \Phi\left(\frac{g(x^*) - \mu(x) - \epsilon}{\sigma(x)}\right) \tag{179}$$

Anche qui il parametro $\epsilon \geq 0$ serve a bilanciare il trade-off tra eploration ed exploitation. Quest'ultima è massima per $\epsilon = 0$. Va notato che questa funzione è, per $\epsilon = 0$ sbilanciata verso l'exploitation.

12.2.3 Expected Improvement

Mantenendo le definizione di $g(x)$ e di $g(x^*)$, il miglioramento atteso si definisce come segue:

$$\mathrm{EI}(x) = \begin{cases} (g(x^*) - \mu(x))\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases} \tag{180}$$

dove le funzioni Φ e ϕ sono rispettivamente la cdf e la pdf della Normale Standard,e Z è il valore standardizzato del valore best seen della funzione obiettivo:

$$Z = \begin{cases} \frac{g(x^*)-\mu(x)-\epsilon}{\sigma(x)} & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases} \tag{181}$$

Anche questa funzione bilancia automaticamente i volumi di exploration ed exploitation attraverso ϵ.

12.3 Ottimizzazione della funzione di acquisizione

Una volta addestrato il modello surrogato in modo da poter calcolare la funzione di acquisizione, si pone il problema della ricerca del massimo della stessa [Archetti and Candelieri, 2019a]. In questo modo l'algoritmo di ottimizzazione bayesiana può selezionare un punto nello spazio degli iperparametri da passare al modello per una nuova fase di training. Le tecniche generalmente adottate a tal proposito sono le seguenti:

- semplice campionamento casuale dei punti
- ottimizzazione basata sul gradiente, come l'algoritmo BFGS (prediletta per Processi Gaussiani)
- ottimizzazione basata su algoritmi genetici (prediletta per modelli tree-based)
- campionamento casuale basato sulla posterior del modello surrogato (e.g. Thompson Sampling, possibile solo per modelli surrogati probabilistici, come i Processi Gaussiani)

In Octis, l'opzione adottata è la prima. Si hanno infatti poche osservazioni per il modello surrogato (se si avesse la possibilità materiale di addestrare il modello centinaia di volte probabilmente non si farebbe ricorso a un algoritmo di ottimizzazione bayesiana). Inoltre gli iperparametri di un modello di topic modeling sono in genere meno di 10. Non risulta quindi costoso calcolare la funzione di acquisizione per un largo insieme di punti campionati uniformemente dallo spazio degli iperparametri. Questa scelta tuttavia rende più randomica l'estrazione dei punti, favorendo l'aspetto esplorativo del modello surrogato. I

Table 1: octis data descriptions

Name in OCTIS	num Docs	num Words	num Labels	Language
20NewsGroup	16309	1612	20	English
BBC_News	2225	2949	5	English
DBLP	54595	1513	4	English
M10	8355	1696	10	English
DBPedia_IT	4251	2047	5	Italian
Europarl_IT	3613	2000	NA	Italian

13 RSM : esperimenti sui dataset di OCTIS

In questo capitolo si vuole vedere come il modello RSM può essere applicato ai dataset già pubblicati con OCTIS. Si tratta di dataset il cui preprocessing è stato già eseguito, per cui rappresentano un affidabile punto di partenza per confrontare diversi topic models. I dataset disponibili in lingua inglese sono i seguenti:

20NewsGroups: Comprende i discorsi di diversi gruppi di discussione appartenenti a 20 tipi diversi, che ne riflettono i topic

BBC-News: Comprende articoli di giornale raccolti dalla testata BBC

DBLP: (Digital Bibliography and Library Project) Comprende i riferimenti bibliografici di diverse pubblicazioni scientifiche di tipo informatico.

M10: Una seconda raccolta di riviste in materia di computer science, etichettate come inerenti a 10 possibili tematiche.

Di seguito una sintesi dei dataset già sottoposti a preprocessing, in lingua inglese e italiana:

Si vede che il dataset che presenta insieme il minor numero di osservazioni e di parole, ovvero che richiede il minor costo computazionale per qualsiasi topic model, è 20NewsGroups, ragion per cui lo si adotterà nella maggior parte degli esperimenti. Gli esperimenti di seguito presentati utilizzeranno i dataset 20NewsGroup, M10 e BBC News. I dataset in lingua italiana saranno esclusi in quanto meno diffusi in letteratura. Il dataset DBLP verrà in molti casi escluso a causa della mancanza del tempo necessario ad analizzarlo.

13.1 esperimento 1: diverse tecniche di contrastive learning per il modello RSM

Come primo esperimento si mettono a confronto i diversi metodi di contrastive learning citati. In questo esperimento, e in tutti quelli che seguono, si impone un numero massimo di iterazioni pari a 1000. Va evidenziato come questo limite sia molto restrittivo se messo a confronto con il numero di iterazioni adottate in [Hinton and Salakhutdinov, 2009], dove il numero di iterazioni va da 10000 per dataset di grandi dimensioni a 100000 per dataset di dimensioni medio-piccole, come i tre dataset qui considerati.

Ioltre, per semplificare il processo sperimentale, il numero di topic assegnato in ciascun dataset si ritiene noto a priori (20 per 20NewsGroups, 10 per M10, 5 per BBC). Per il calcolo del gradiente si usa un semplice stocastic gradient descent. Gli altri iperparamentri sono fissati come segue:

- epoche: 1000
- learning rate: 0.0001
- batch size: 20
- decay/fattore di penalizzazione: 0 (modello non penalizzato)

Per ciascun dataset, si mettono allora a confronto le 4 alternative tecniche di constrastive divergence:

- persistent CD
- mean field CD
- 1-step CD
- 2-step CD

Per la K-step Contrastive Divergence si ripete l'esperimento per K=1 e K=2. Non è utile tentare con un valore di K superiore, a causa del ridotto numero di epoche.

Per ogni dataset, vengono monitorate, ogni 20 iterazioni/epoche:

- l'accuracy
- l'approssimazione della perplexity
- la topic diversity
- l'indice di coherence NPMI
- Il tempo medio in secondi per iterazione

Si deve fare in particolare attenzione a come evolve la perplexity, in quanto si lega direttamente al livello di enegia e alla log verosimiglianza del modello. Una perplexity stazionaria implica spesso un gradiente nullo, e quindi un arresto o una paralisi (anche solo temporanea) del processo di training.

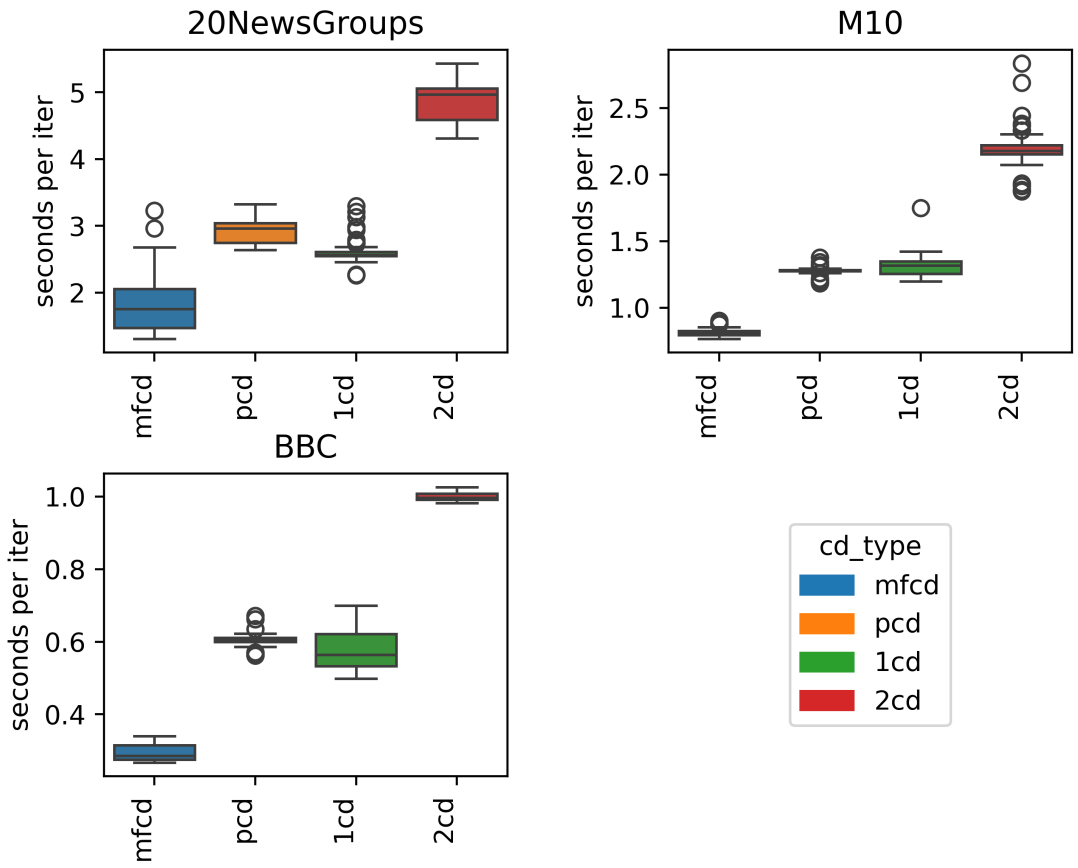


Figure 27: CD types: Tempo medio in secondi per iterazione
Per ciascun dataset, si mettono a confronto tre tecniche di contrastive divergence. Si vede come in tutti e tre i casi la persistet CD risulta equivalente al 1-step CD, come il tempo cresca linearmente in K, e come la mean field CD risulti nettamente più veloce degli altri metodi. Si noti che, se il tempo richiesto per iterazione fosse 1.5 secondi con MFCD, 2.5 per PCD e 1-step CD, e 4.5 secondi con 2-step CD, si pù stimare su 1000 epoche un tempo di training pari a 1500 secondi per il primo (circa 25 minuti), 2500 secondi per il secondo e il terzo (circa 42 minuti) e 4500 secondi per il quartp (circa 75 minuti). Ci si potrebbe chiedere come mai, essendo il dataset 20NewsGroup con un numero di documenti inferiore e un vocabolario ridotto rispetto agli altri due, il suo costo computazionale risulti di tanto superiore. La ragione sta nel numero di topic scelto: si vede che al crescere del numero di topic, cresce il tempo richiesto per iterazione.

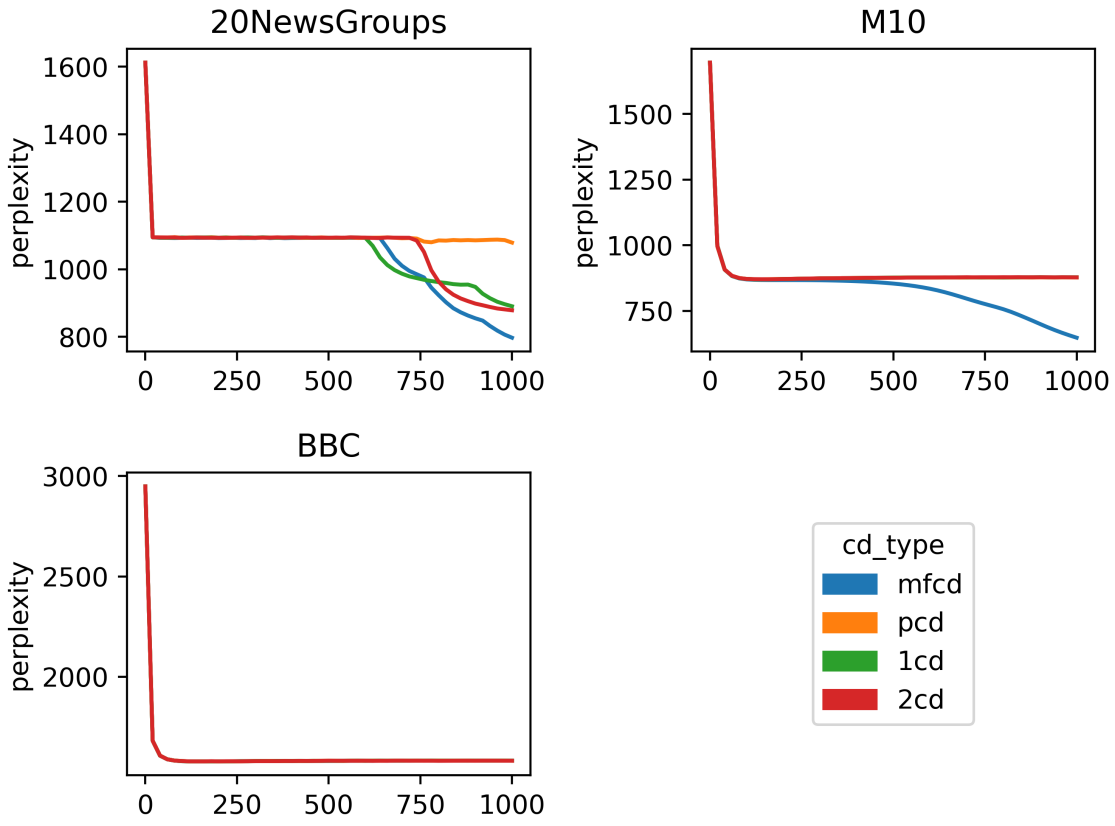


Figure 28: CD types: Perplexity
Si vede che la perplexity decade prima in due dataset su tre con MFCD. Non ci sono particolari differenze sulle altre metriche.

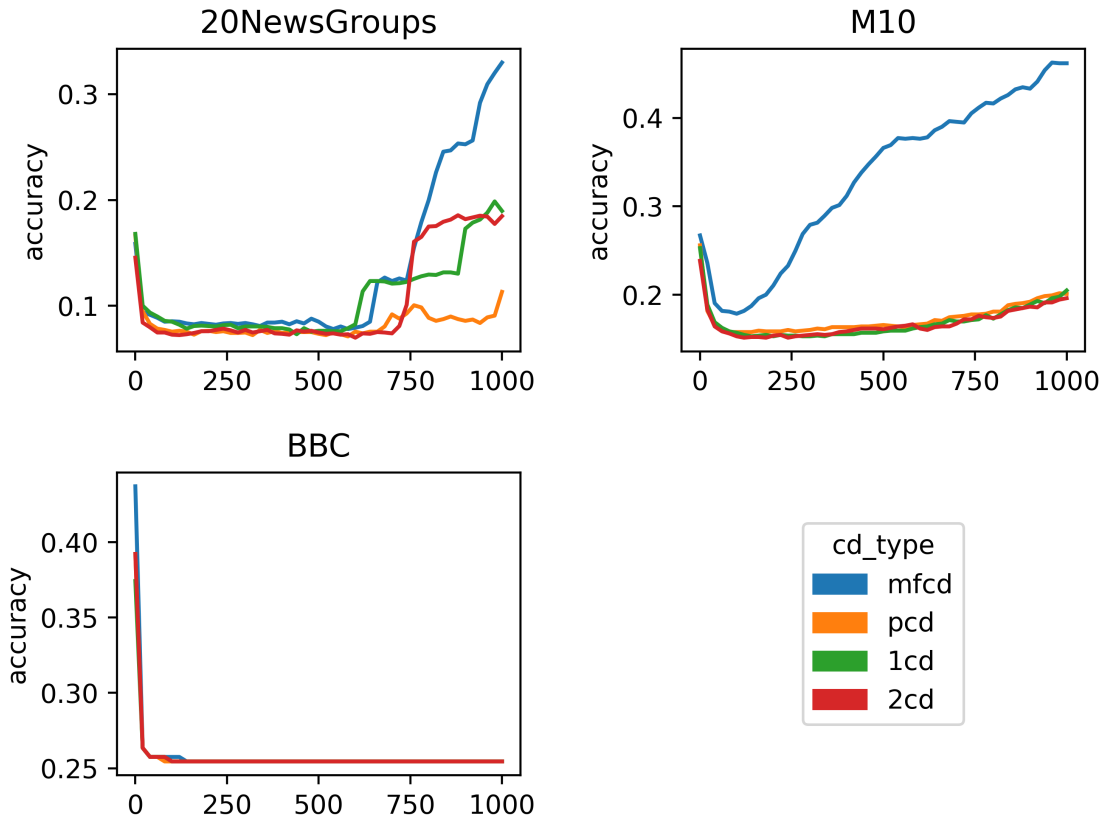


Figure 29: CD types: accuracy
Si vede che la MFCD conduce velocemente a un miglioramento.

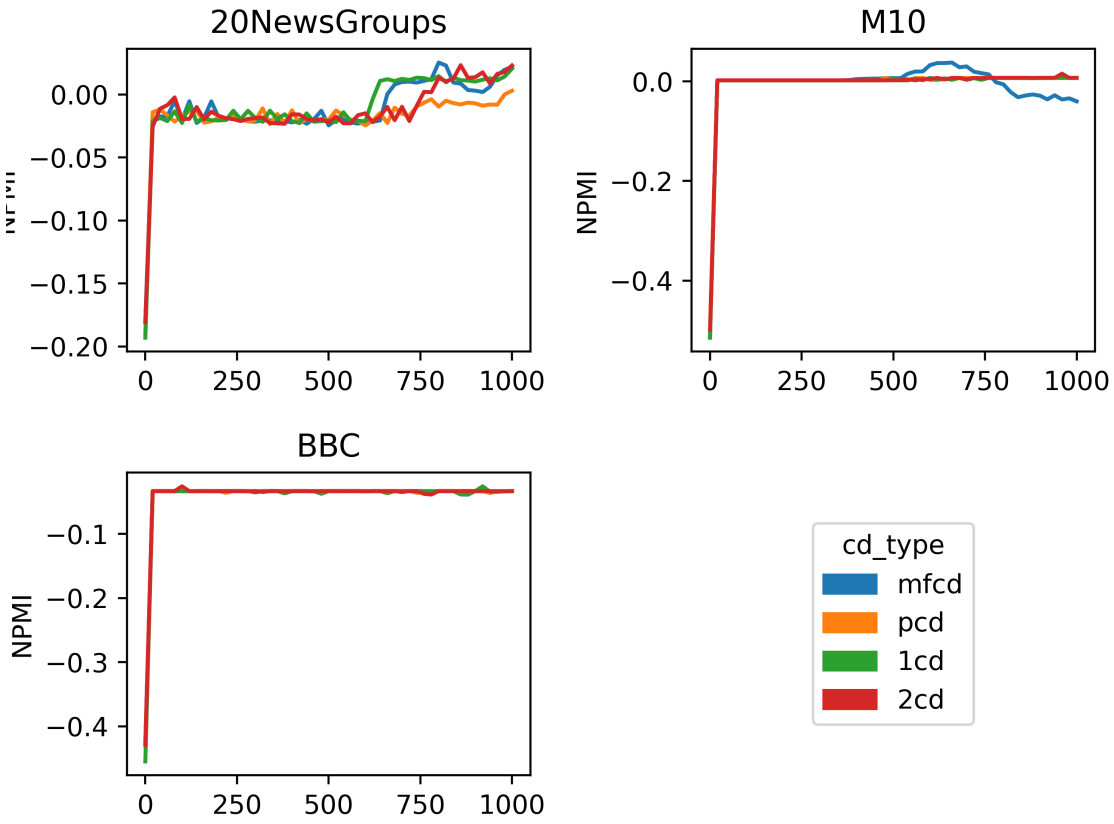


Figure 30: CD types: NPMI corehence
In tutti i dataset le performance in termini di coherence da parte del modello RS sembrano piuttosto povere.

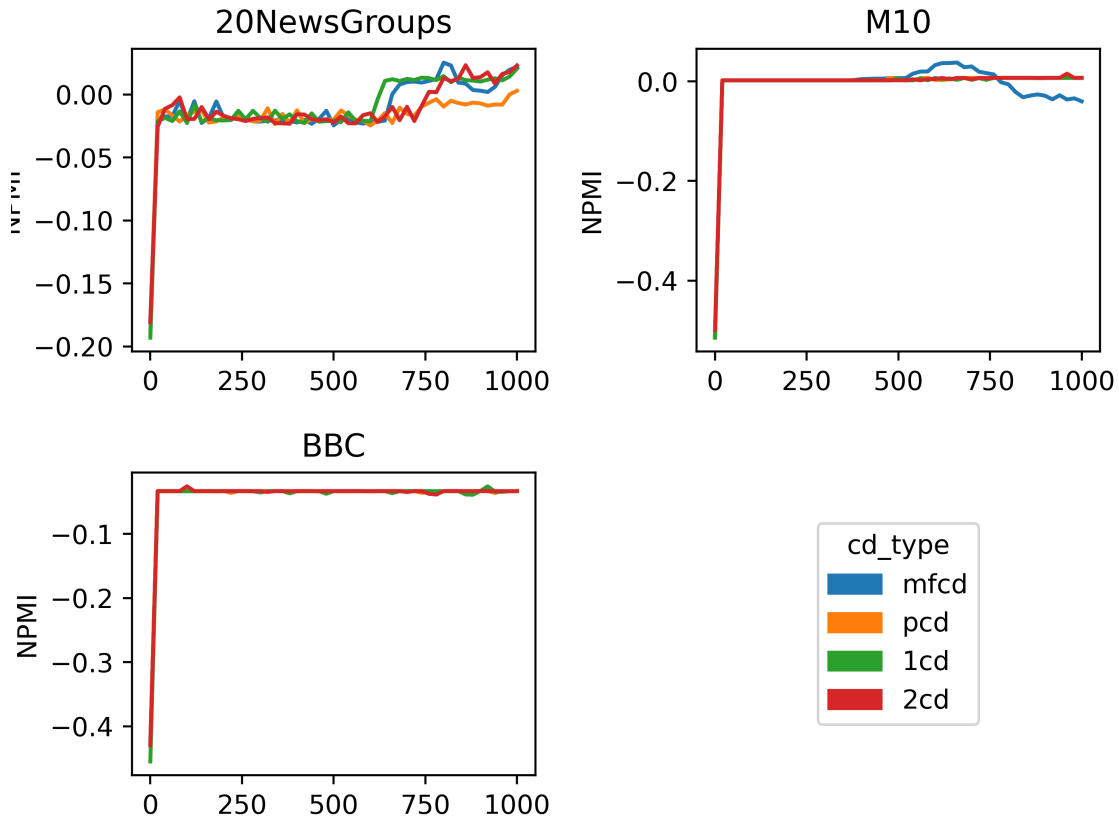


Figure 31: CD types: NPMI corehence
In tutti i dataset le performance in termini di coherence da parte del modello RS sembrano piuttosto povere.

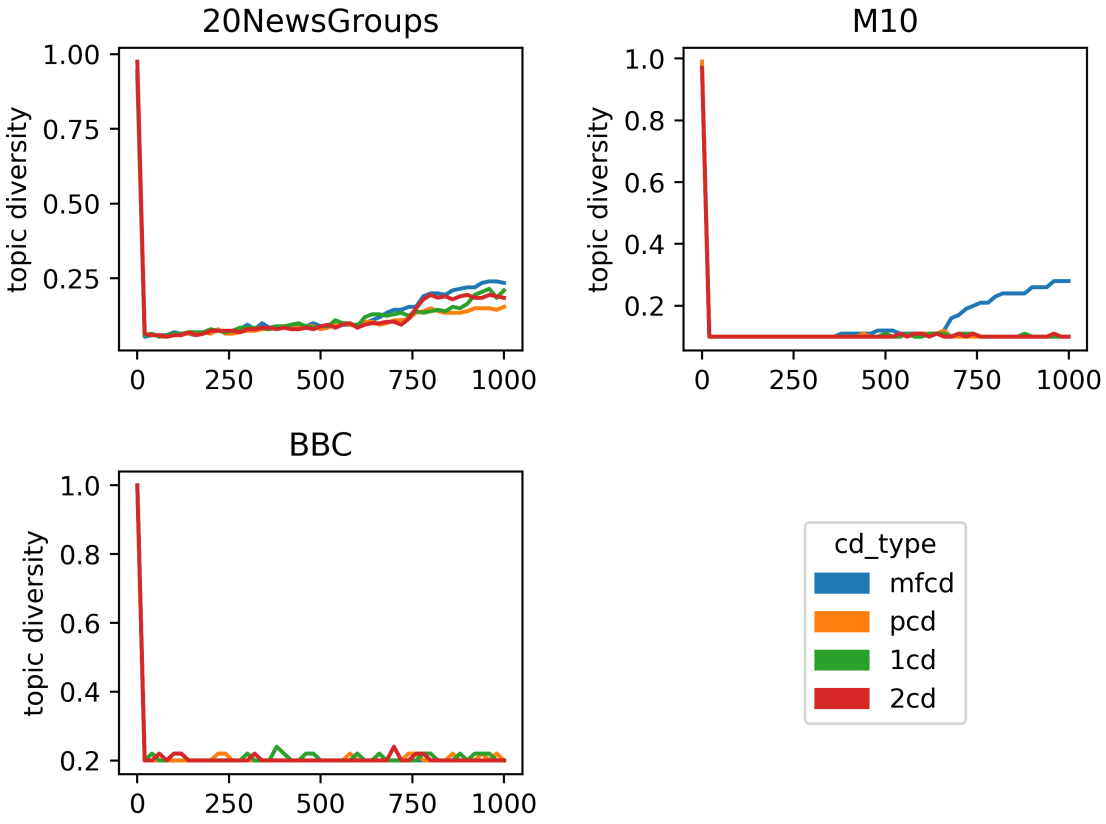


Figure 32: CD types: topic diversity
Come per l'accuracy e la perplexity, il metodo MFCD mostra un evidente superiorità rispetto agli altri metodi. Tuttavia, il modello non risulta particolarmente performante sotto questa metrica.

In questo esperimento si è vista una globale superiorità del metodo MFCD rispetto agli altri 3 proposti. La Mean Field Contrastive Divergence risulta una procedura di training veloce, computazionalmente più economica rispetto alla tradizionale 1-step CD e molto più performante sotto le 1000 iterazioni, a livello di perplexity, accuracy e topic diversity. E' verosimile che per un numero più alto di iterazioni tale distanza svanisca, e che anzi, tecniche come la 2-step CD risultino più performanti della MFCD. Tuttavia, essendo obiettivo di questa tesi massimizzare l'efficienza di questo modello mantenendo il numero di epoche inferiore a 1000, si decide di optare per questa tecnica di contrastive divergence in tutti gli esperimenti successivi.

13.2 esperimento 2: confronto con penalized RSM

Di seguito si mettono a confronto il modello RSM con MFCD e stocastic gradient descent con la sua versione penalizzata, denominata pRS o Penalized Replicated Softmax.. Gli iperparametri rimangono gli stessi adottati nell'esperimento 1, con la differenza che nei modelli penalizzati si adotta un parametro di penalizzazione maggiore di 0. Si provano in particolare i modelli con penalizzazione pari a 0.1 e a 0.001, usando penalizzazioni di tipo L1 e L2.

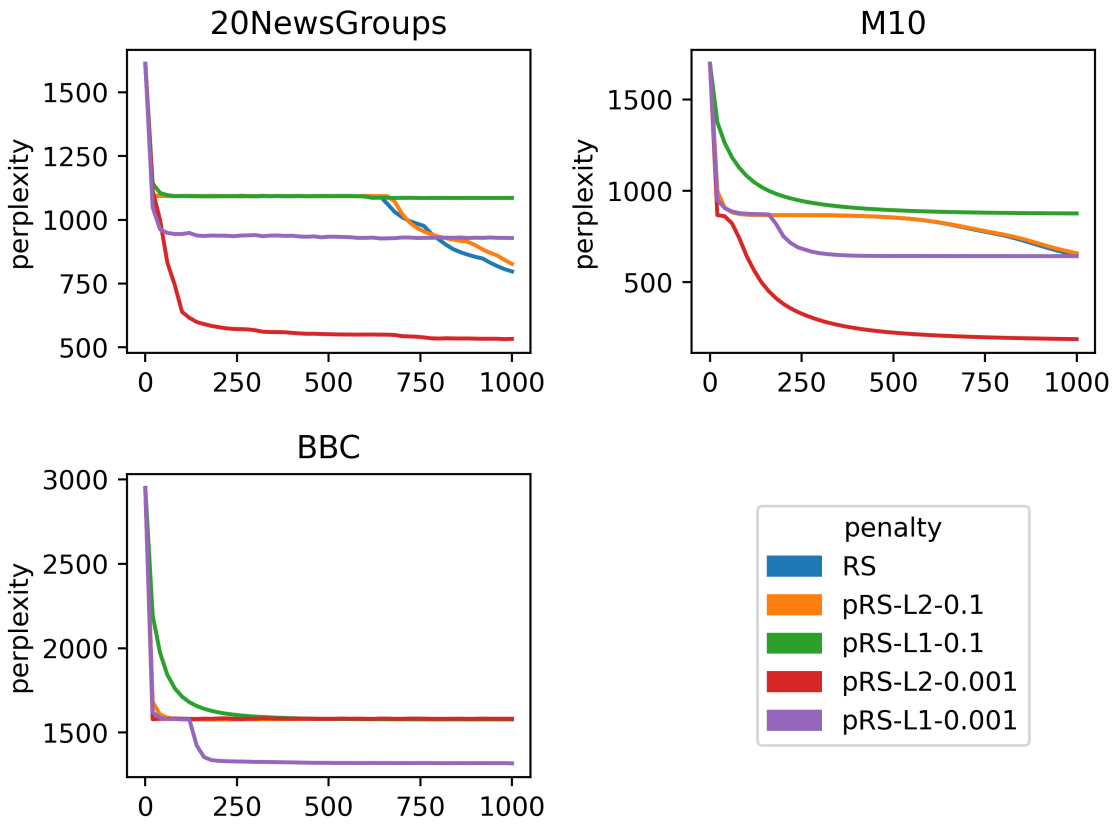


Figure 33: pRSM: perplexity

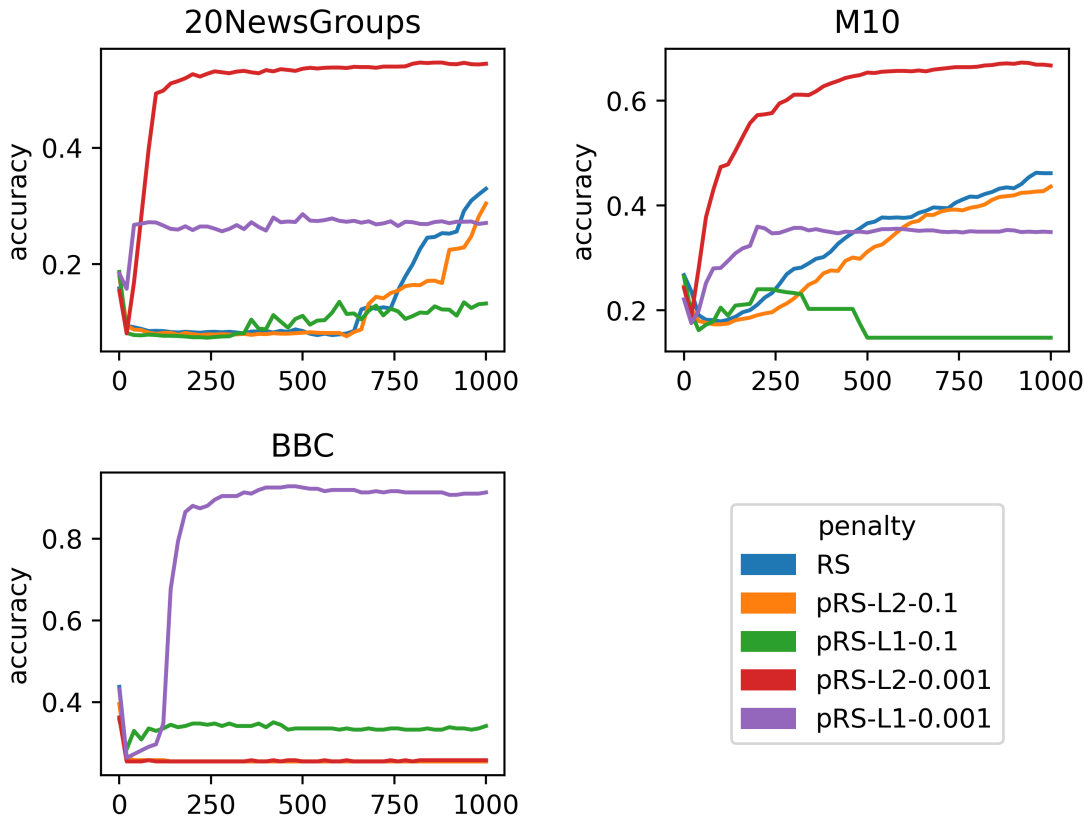


Figure 34: pRSM: accuracy
Accuracy e perplexity risultano massime con decay pari a 0.001, usando una penalizzazione L2 per i primi due dataset e una penalizzazione L1 per i documenti in BBC.

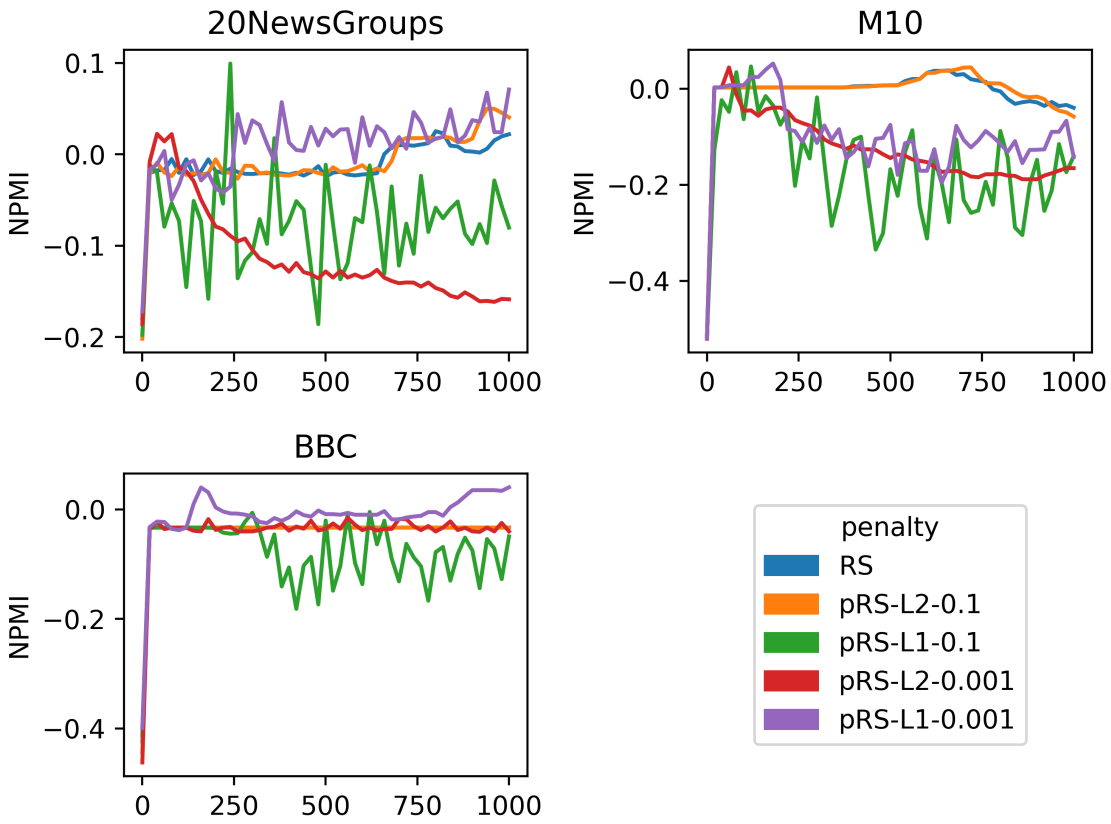


Figure 35: pRSM: NPMI coherence

Come nell'esperimento precedente, questa metrica sembra comportarsi in modo opposto rispetto alle altre: ciò che minimizza la perplexity e massimizza l'accuracy, peggiora la coherence. Si noti l'instabilità della coherence su tutti i dataset quando si adotta la penalizzazione di tipo L1.

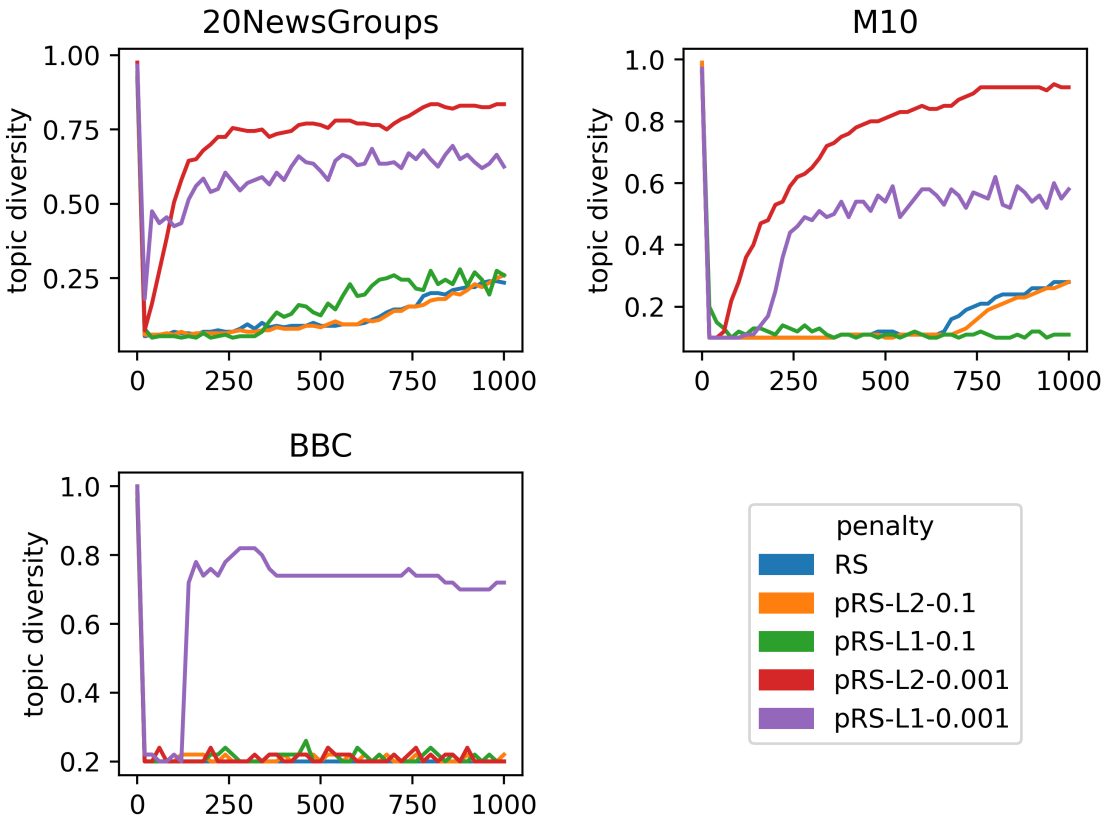


Figure 36: pRSM: topic diversity
La topic diversity sembra riflettere i risultati mostrati dall'accuracy e dalla perplexity.

Si vede per la maggior parte dei risultati che una penalizzazione troppo alta (con decay ≥ 0.1) aggrava le performance, mentre una penalizzazione molto leggera, e del tipo corretto, può portare a miglioramenti molto significativi. In questo caso, è evidente come la penalizzazione L2 fosse la più adatta ai dataset 20NewsGroups e M10, mentre per il dataset BBC la penalizzazione di tipo L1 risulta essere l'unica in grado di apportare miglioramenti significativi. L'introduzione di un fattore di penalizzazione sulla matrice dei pesi di interazione porta la stessa ad avvicinarsi a una forma sparsa, e ciò ha un effetto positivo sulle metriche prese in considerazione, incluse quelle riferite all'interpretabilità del modello. Una adeguata scelta del tipo e del valore della penalizzazione può condurre con meno di 200 iterazioni di MFCD a un livello di perplexity superiore a quello che si potrebbe raggiungere con più di 1000 iterazioni di 1-step CD in assenza di penalizzazione.

13.3 esperimento 3: comparazione di diverse tecniche di gradient descent per il modello RSM

Di seguito si monitorano le performance di gradient descent con diversi ottimizzatori per il training. Ogni metodo di ottimizzazione punta a migliorare la formazione del gradiente. I metodi qui sperimentati sono i seguenti:

- sgd (nessun ottimizzatore)
- momentum (che incorpora sgd nel caso in cui momentum=0)
- adagrad
- rmsprop
- adam

Diversi ottimizzatori comportano l'inclusione di nuovi iperparametri, ai quali devono essere assegnati dei valori di default:

- il parametro del momentum è posto uguale a 0.9
- il parametro di rmsprop è posto uguale a 0.9
- i due iperparametri di adam sono posti pari a 0.9 e a 0.99

- Adagrad non presenta alcun nuovo iperparametro

In tutti i casi si adotta la MFCD. Le metriche di performance monitorate e i valori assegnati agli altri iperparametri restano i medesimi degli esperimenti precedenti.

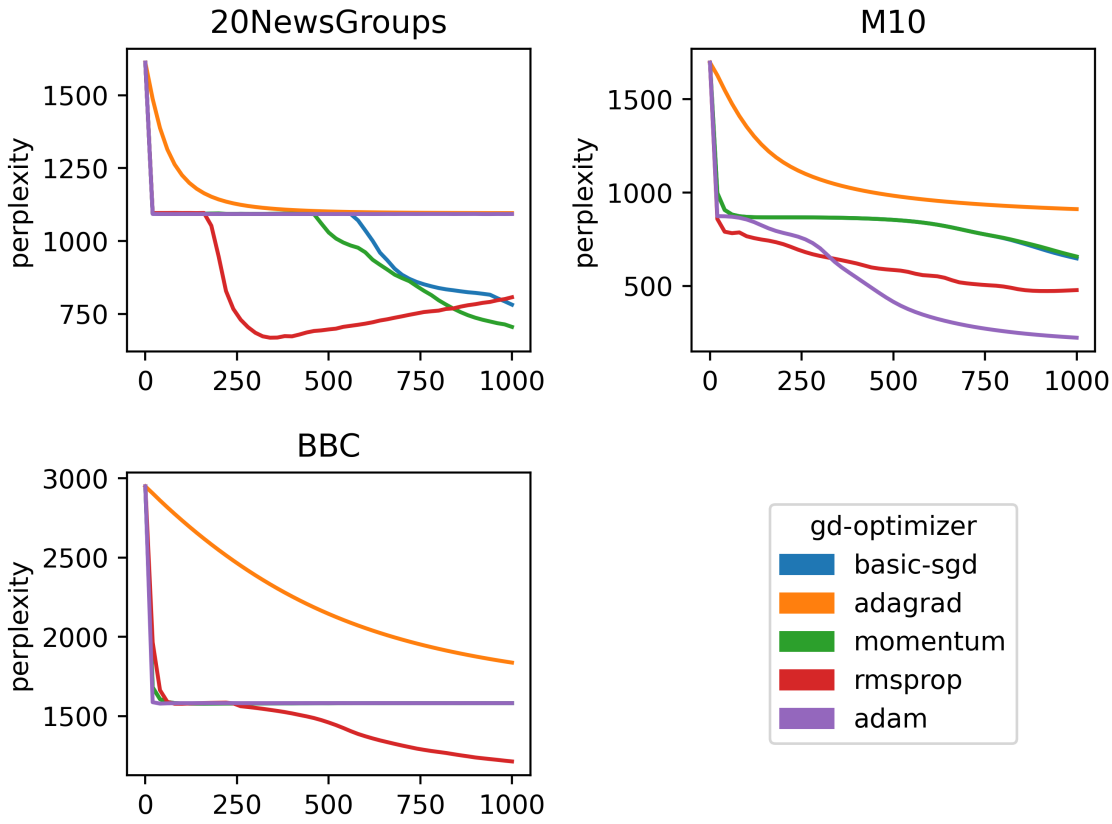


Figure 37: optimizers: perplexity
Si vede che rmsprop porta più rapidamente a una discesa della perplexity in tutti i dataset, sebbene non risulti il migliore nel lungo termine.

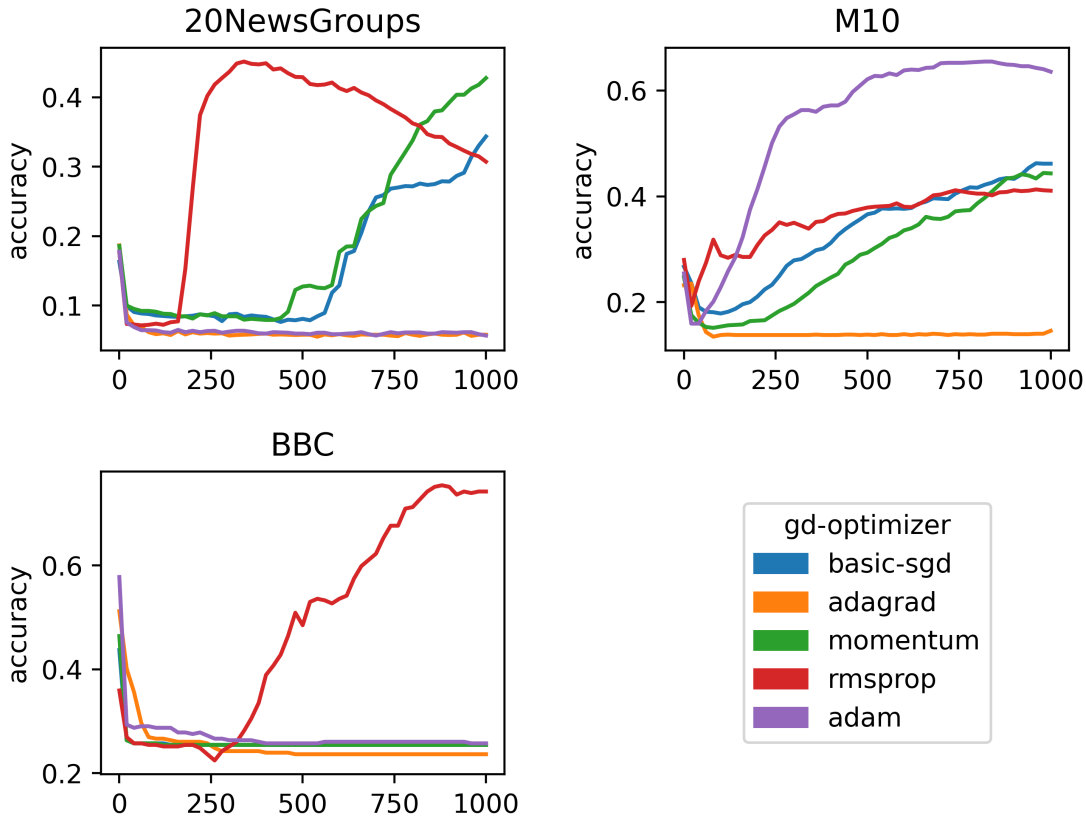


Figure 38: optimizers: accuracy
Valgono i medesimi commenti fatti per l'immagine precedente: rmsprop risulta più performante nel breve termine, ma è superato da adam sul dataset M10 e da momentum sul dataset 20NewsGroups.

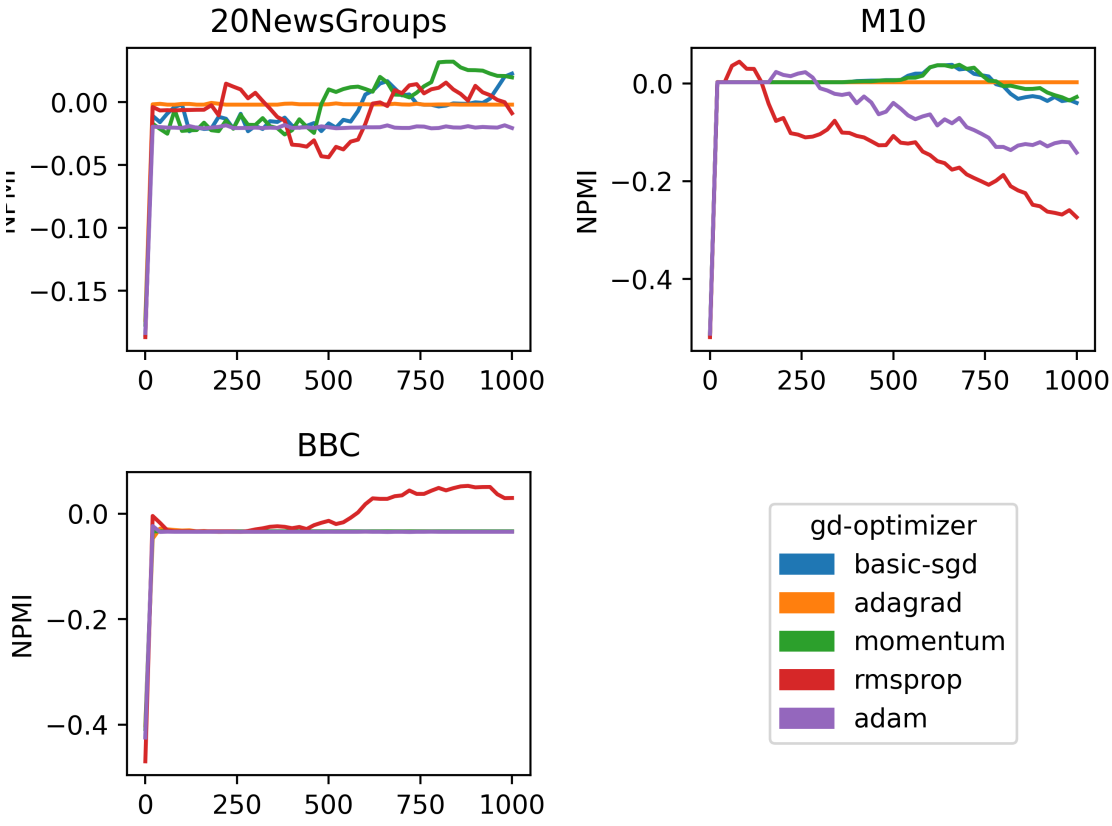


Figure 39: optimizers: NPMI coherence
Si noti che a differenza di ciò che accade negli esperimenti precedenti, la coherece risulta complessivamente migliorata, sebbene non raggiunga mai valori di molto superiori a 0.

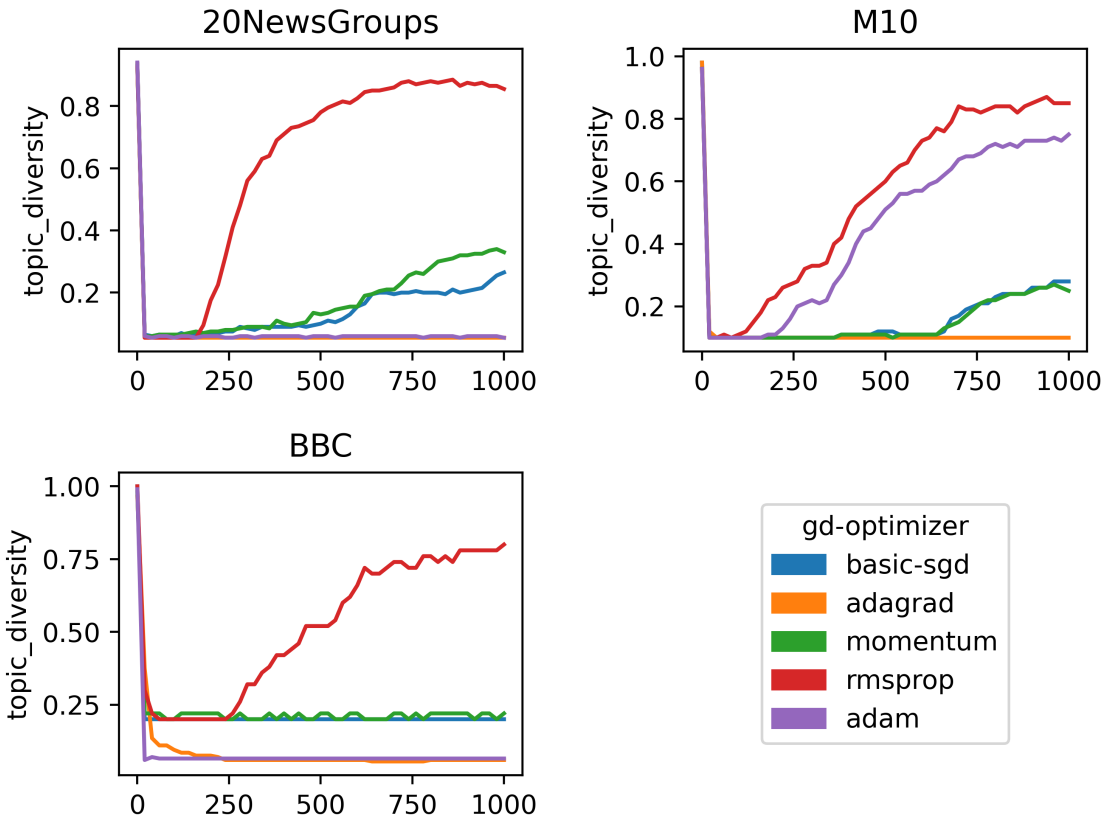


Figure 40: optimizers: topic diversity
Su questa metrica il metodo rmsprop risulta universalmente più performante.

L'adozione di ottimizzatori di gradient descent, come momentum, rmsprop e adam, può portare a miglioramenti significativi del modello RSM, se adottati con un adeguato tuning degli iperparametri. L'ottimizzatore adagrad si rivela peggiore del gradient descent non ottimizzato. Degli ottimizzatori proposti, quello che sembra introdurre maggiori miglioramenti sulla maggior parte dei dataset e delle metriche risulta essere rmsprop. Anche adam presenta buone performance nel caso del dataset M10, ma richiede di impostare due iperparametri, e presenta il costo computazionale maggiore. Per questa ragione, nel caso in cui si volesse applicare una procedura di ottimizzazione bayesiana sul modello RSM, un ottimizzatore di tipo rmsprop potrebbe risultare la scelta più adatta, dato il minor dimensione dello spazio di ricerca e il minor costo della sua esplorazione.

13.4 esperimento 4: Ottimizzazione bayesiana del Replicated Softmax

In questa sezione si applicano gli strumenti di ottimizzazione bayesiana al modello RS. Si adotta un Processo Gaussiano come modello surrogato, e la Lower Confidence Bound come funzione di acquisizione. La funzione obiettivo che si è scelto di adottare è la Topic Diversity, ma sarebbe interessante ripetere l'esperimento adottando l'accuracy o la coherence come funzioni obiettivo. Non si è scelta l'accuracy per una ragione pratica: in applicazioni reali del topic modeling, che è una procedura non supervisionata, è inverosimile che per ogni documento siano presenti dei label già assegnati, necessari per calcolare questa metrica. Non si è scelta la coherence alla luce degli esperimenti precendenti: sembra cha la metrica NPMI non sia in alcun modo correlata, o che addirittura sia negativamente correlata, con la log verosimiglianza del modello. Metriche di coherence diverse dalla NPMI conducono a risultati analoghi.

In tutti i casi si vuole adottare MFCD e ottimizzazione rmsprop. Di conseguenza si deve ottimizzare il rispettivo iperparametro. Si vuole includere nello spazio di ricerca anche un fattore di penalizzazione di tipo L2. Per tutti i dataset l'obiettivo di ottimizzazione bayesiana è quello di ottimizzare i parametri riportati nella tabella che segue, definiti sul medesimo spazio di ricerca (si veda il valore di default assegnato finora negli esperimenti precedenti).

Per i dataset 20NewsGroups e M10 sono state adottate 20 epoche di training per ogni

Table 2: spazio di ricerca scelto per l'ottimizzazione bayesiana del Replicated Softmax

iperparametro	abbreviazione	tipo	minimo	massimo	default value
batch size	btsz	Integer	1	300	20
learning rate	lr	Real	0.00001	0.005	0.0001
L2 penalty	decay	Real	0	0.2	0
rmsprop decay	rms decay	Real	0	0.99	0.9

Table 3: esiti del processo di ottimizzazione bayesiana del RSM

	20NewsGroups	M10	BBC
best topic diversity	0.35	0.35	0.94
best iteration	13	13	3
train epochs	20	20	120
num topics	20	10	5
learning rate	0.004773	0.00407	0.00497
batch size	74	46	298
L2 penalty	0.00378	0.0013547	0.03
rmsprop decay	0.796	0.723	0.0163

Dei valori sopra, si fa presente che train epochs e num topics sono fissati a priori, mentre learning rate, batch size, L2 penalty e rmsprop decay sono gli iperparametri ottimizzati.

prova. Per il dataset BBC ne sono state adottate 120. In tutti i dataset, anche in questo caso, il numero di topic si ritiene noto a priori (20 per 20NewsGroups, 10 per M10, 5 per BBC). Per tutti e tre i dataset si riporta nella tabella che segue l'esito del processo di ricerca.

Come negli esperimenti precedenti, si mettono a confronto le metriche NPMI, topic diversity, accuracy e perplexity per il modello RS non ottimizzato (con sgd e con rmsprop) con il modello RS addestrato per 1000 epoche con gli iperparametri ottenuti.

L'adozione di strumenti di ottimizzazione bayesiana sulla topic diversity, anche per poche epoche di training, porta a una minimizzazione del livello di perplexity. Su tutti e tre i dataset proposti, il processo di ottimizzazione bayesiana ha condotto a una soluzione che prevede un fattore di penalizzazione non nullo.

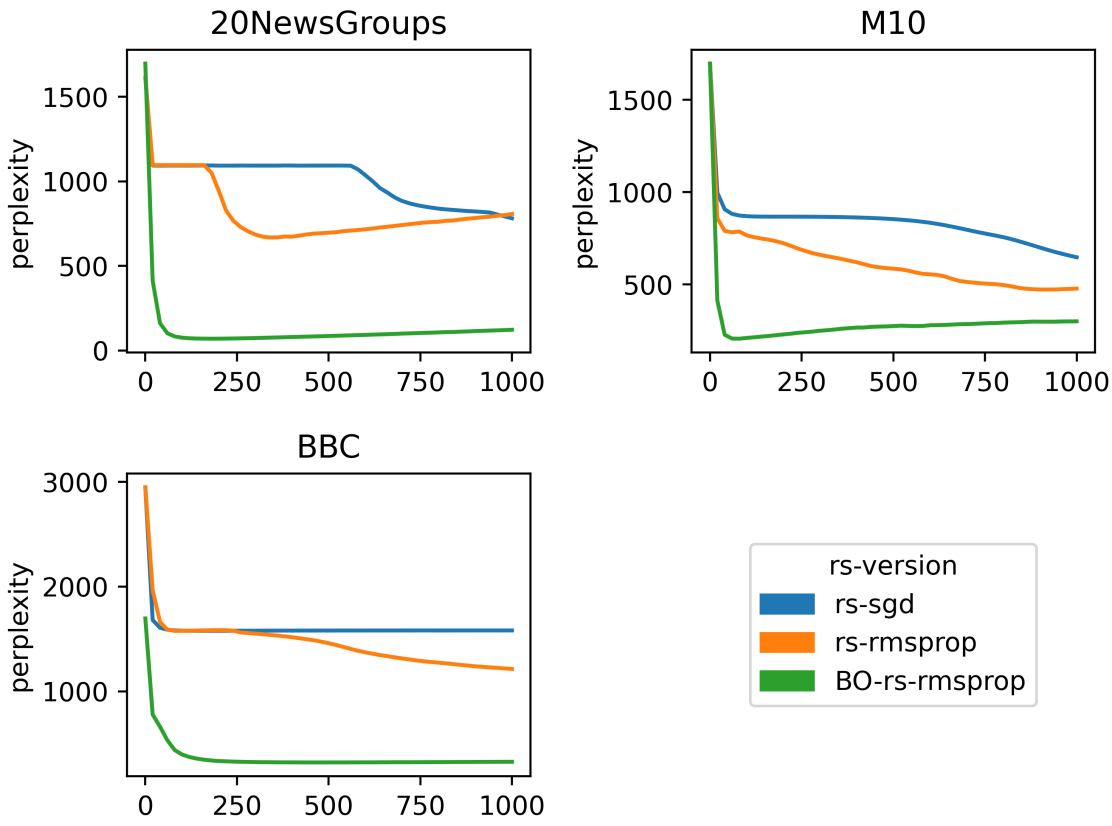


Figure 41: bayesian optimization : perplexity
In tutti i dataset, la scelta della topic diversity come metrica da ottimizzare fornisce una combinazione di iperparametri che consente di raggiungere i livelli minori di perplexity. Si noti che dopo le prime 200 epoche di training la scelta di questi iperparametri non conduce a nessun miglioramento. Questo è verosimile, in quanto i modelli addestrati in fase di ottimizzazione bayesiana avevano meno di 200 epoche di training.

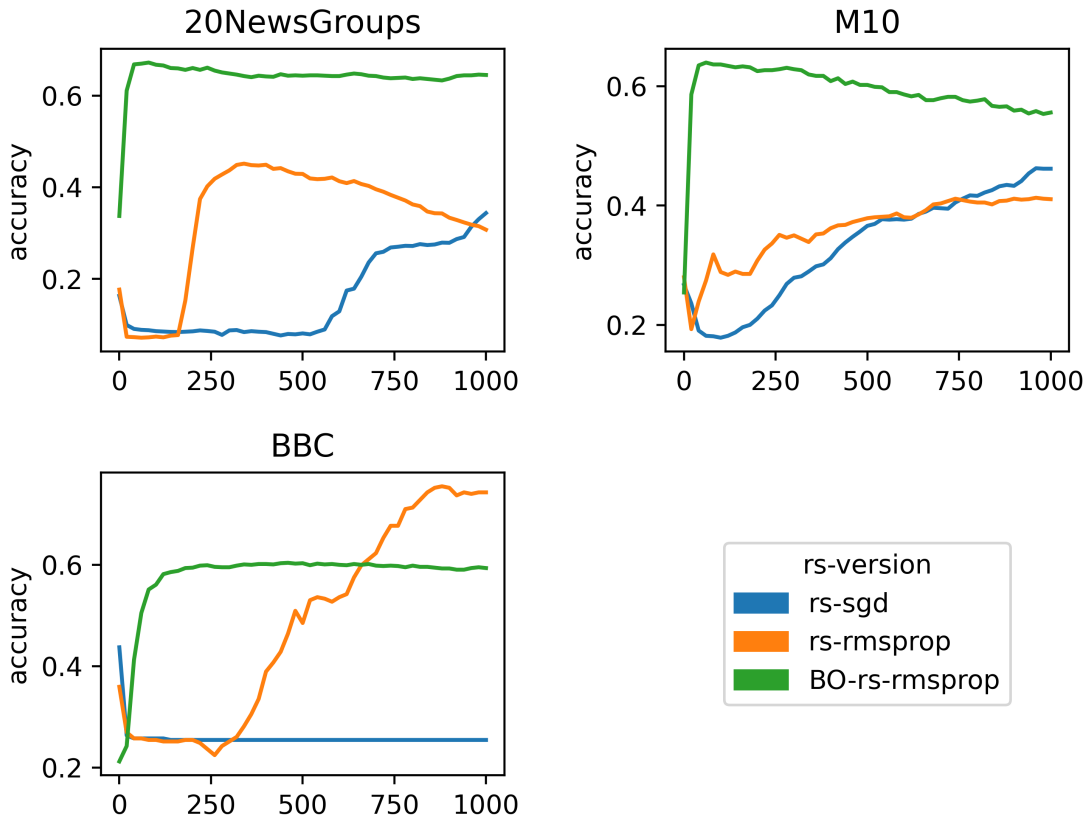


Figure 42: bayesian optimization : accuracy
Valgono considerazioni analoghe a quelle fatte per la perplexity.

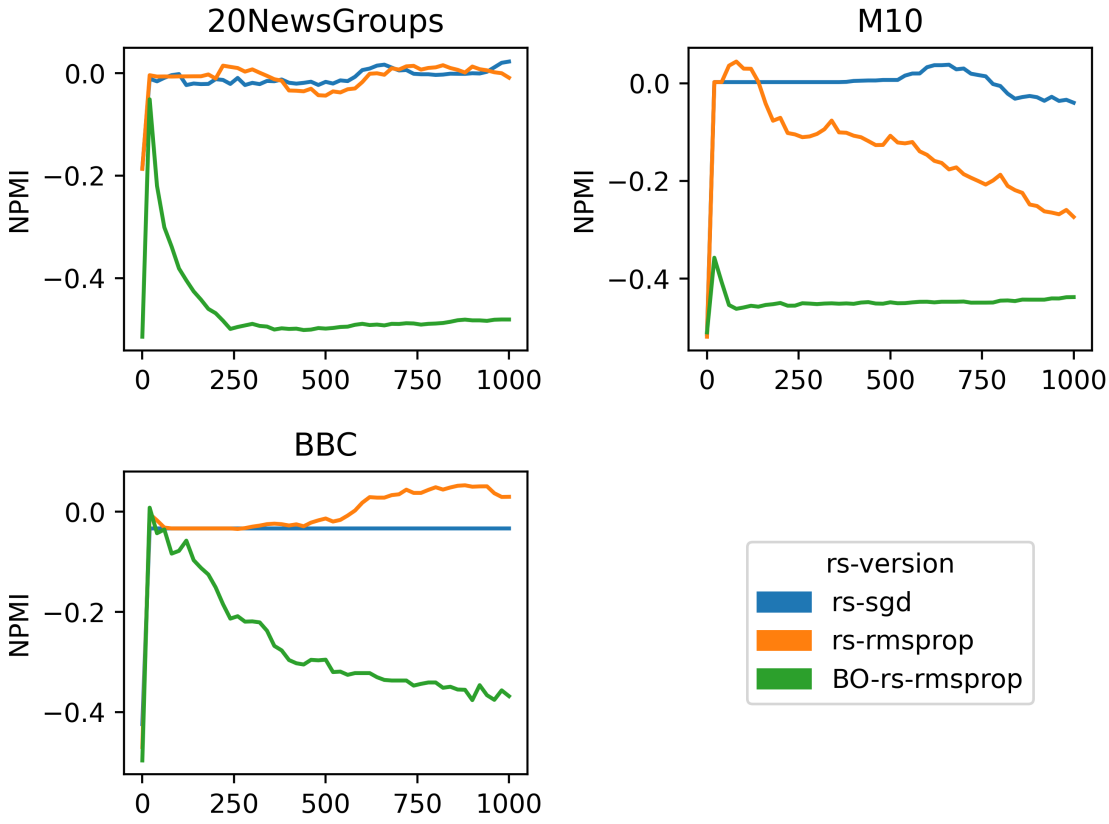


Figure 43: bayesian optimization : NPMI coreherence
Dopo un iniziale miglioramento dell'inidice NPMI, il modello ottimizzato presenta una bassa coherence. Questo va a supporto della tesi per cui potrebbe esistere un trade-off tra perplexity e coherence.

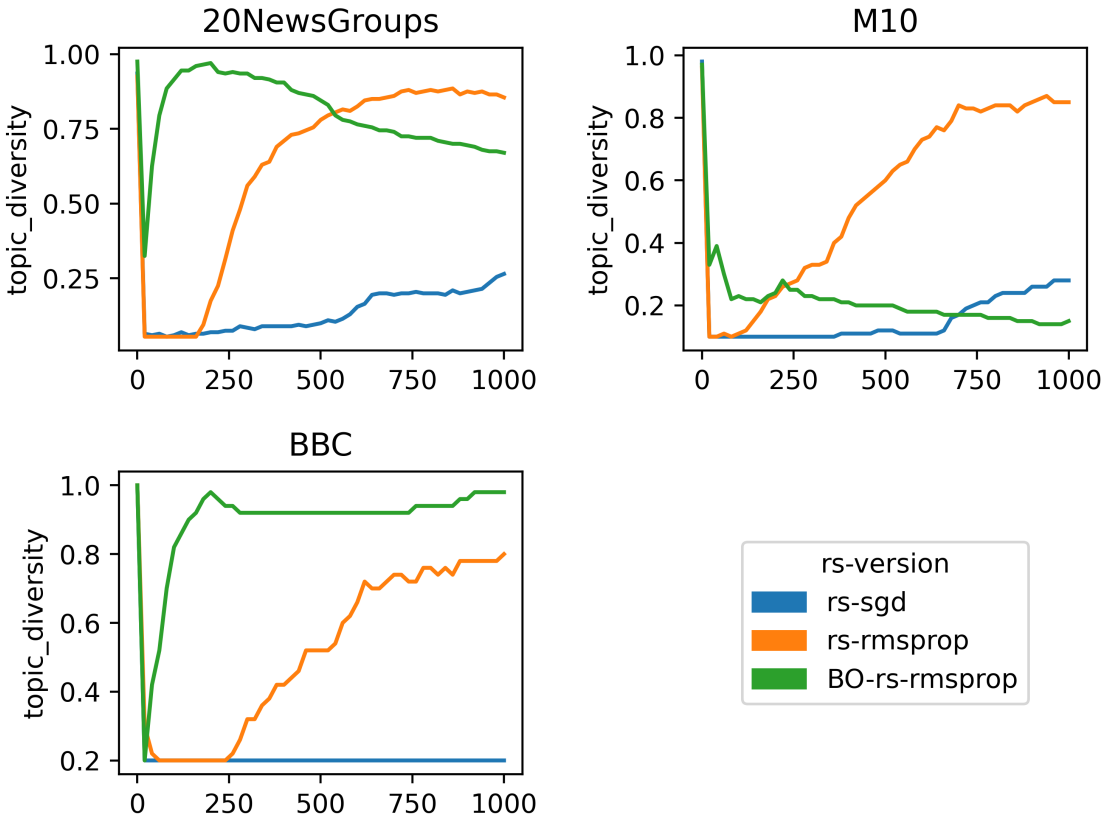


Figure 44: bayesian optimization : topic diversity
Si noti che la topic diversity era la metrica obiettivo dell'ottimizzazione bayesiana. Come ci si può aspettare, le performance risultano massimizzate solo nell'iterazione di training corrispondente al numero di epoche assegnate nel processo di ottimizzazione bayesiana.

13.5 Relazione tra intercette del visible layer e frequenza delle parole

Si possono interpretare i bias di ciascuna hidden e visible unit come delle proxy delle loro frequenze marginali: più è alta l'intercetta, più è frequente trovare la parola in un documento, senza considerare i topic di appartenenza.

Questa interpretazione trova conferma se si osserva la relazione approssimativamente lineare tra il valore assegnato dal modello al bias di ciascuna parola e il logaritmo della frequenza di quest'ultima nella matrice documento termine.

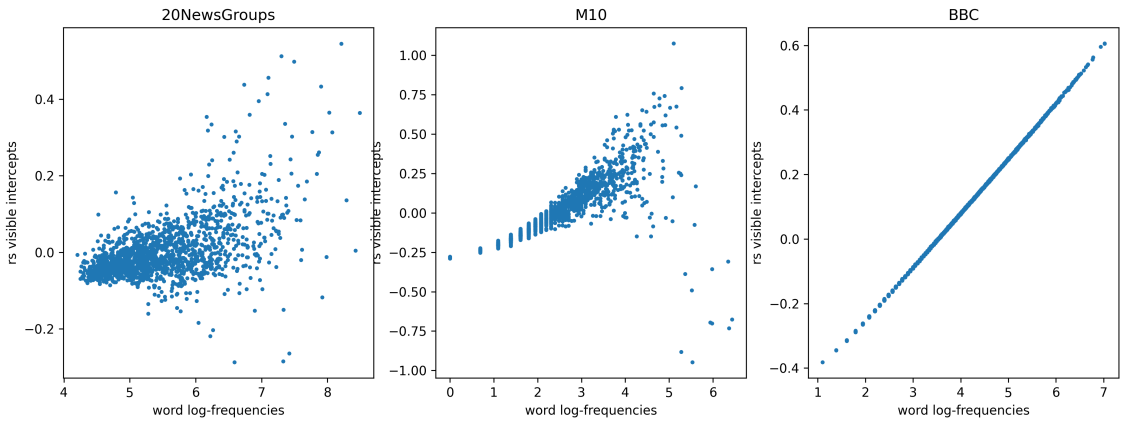


Figure 45: Intercette di RS con MFCD-SGD

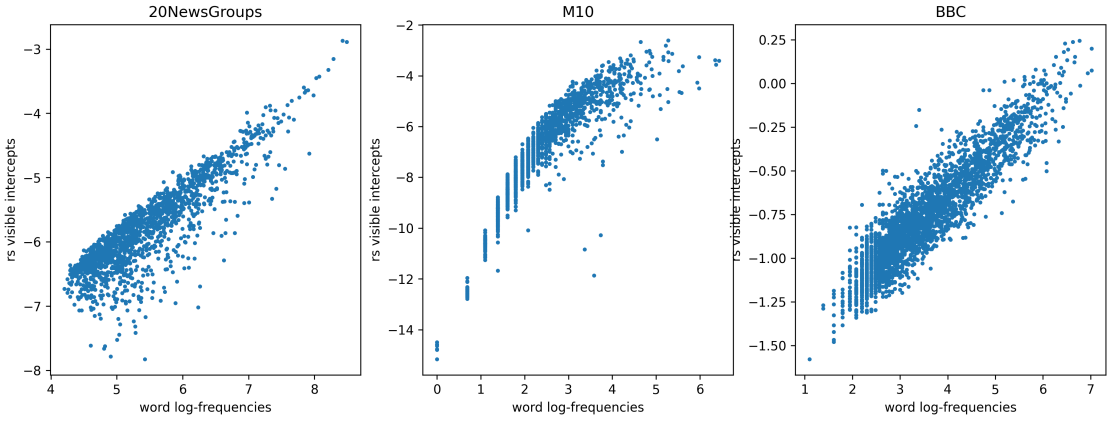


Figure 46: Intercette di RS con MFCD-RMSPROP

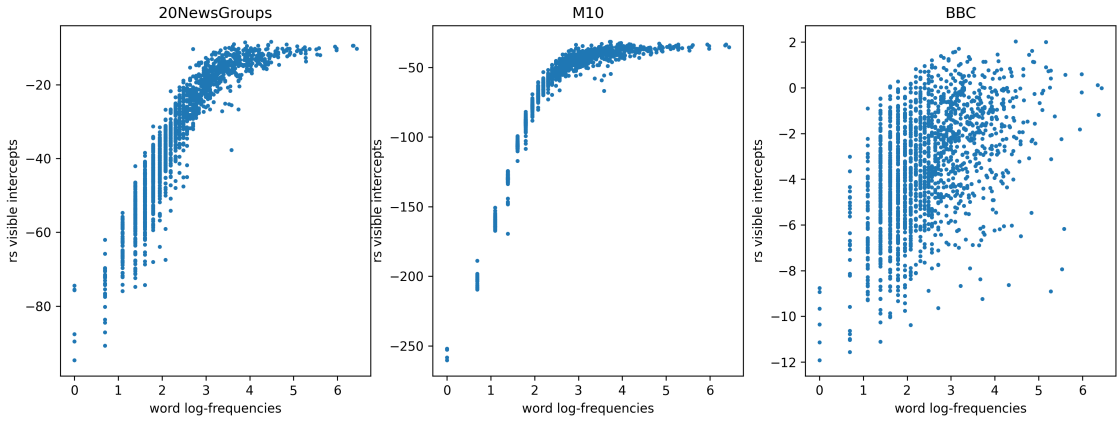


Figure 47: Intercette di RS con ottimizzazione bayesiana

13.6 Distribuzione dei pesi di interazione

In questa sezione si vedono gli istogrammi dei valori presenti nella matrice di interazione, con pesi simmetrici, tra visible e hidden layer. Si mettono a confronto in particolare i pesi ottenuti nei modelli con le migliori performance di ciascun esperimento:

- RS con MFCD, senza ottimizzatore e senza penalizzazione
- RS con penalizzazione L2 di valore 0.001
- RS con ottimizzatore rmsprop e nessuna penalizzazione
- RS con iperparametri scelti attraverso ottimizzazione bayesiana

Come negli esperimenti finora eseguiti, per ragioni di sintesi l'analisi dei tre dataset si svolge in parallelo.

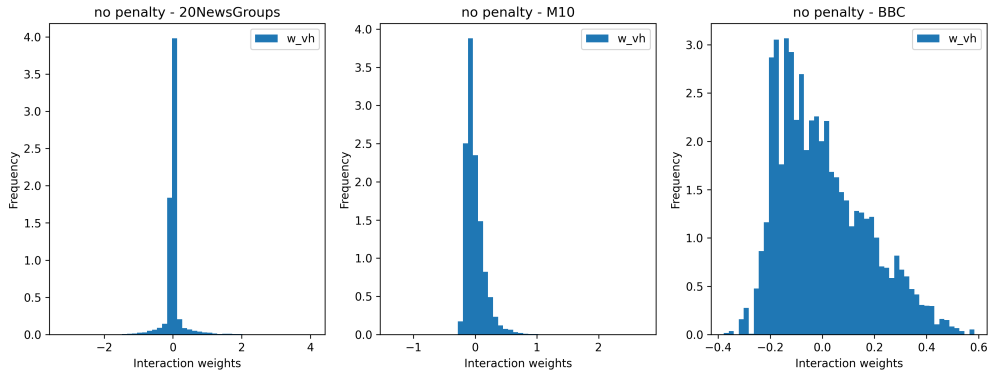


Figure 48: Pesi di interazione non penalizzati, con sgd
Si vedono i pesi di interazione del modello RS addestrato con MFCD e SGD, senza alcuna penalizzazione. La maggior parte dei pesi si concentra intorno a 0. Ciò è dovuto principalmente al fatto che vengono inizializzati da una normale a media nulla con varianza molto piccola. Pochi pesi sono davvero rilevanti sotto questi modelli.

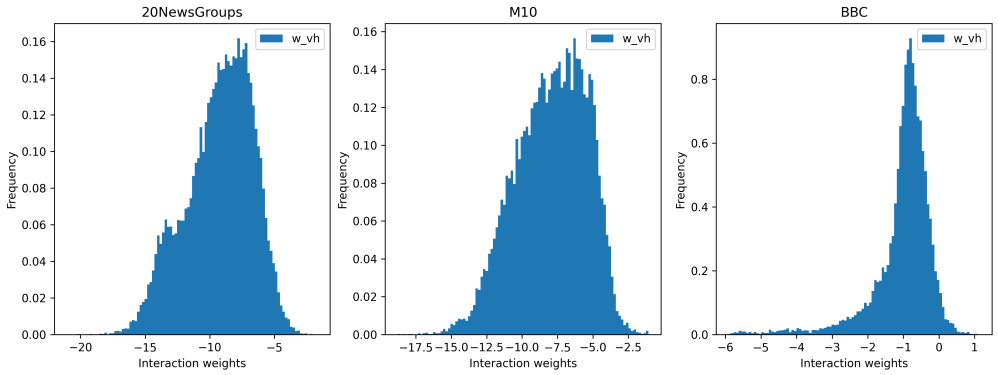


Figure 49: Pesi di interazione non penalizzati, con rmsprop
I pesi si distribuiscono intorno a una media, ma non sono centrati in 0. Il modello con ottimizzatore rmsprop presenta performance migliori in perplexity e accuracy, ma non si presenta come molto interpretabile.

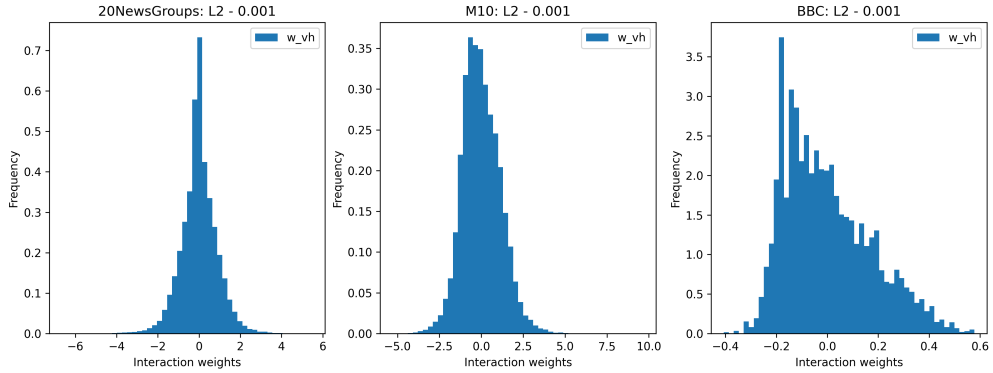


Figure 50: Pesi di interazione con penalizzazione L2 e fattore 0.001
La penalizzazione L2 porta a uno spostamento dei pesi intorno a 0. In questo modo, è possibile associare a ciascun peso una misura di corrispondenza positiva, negativa o nulla tra ogni visible unit e ogni hidden unit. La distribuzione sembra seguire una campana più liscia che nel caso di rmsprop non penalizzato.

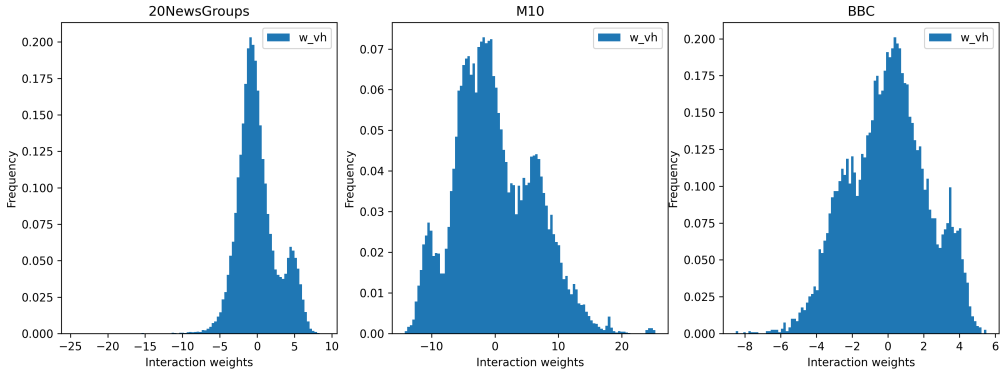


Figure 51: Pesi di interazione con penalizzazione L2 fissata con ottimizzazione bayesiana
Si fa presente che in tutti gli esperimenti di ottimizzazione bayesiana il parametro di penalizzazione L2 non è mai scelto nullo. In questo caso i pesi sembrano seguire una gaussiana multimodale. Nei tre dataset, in particolare, sembra che vi sia sempre un picco sulla parte positiva, a indicare il fatto che molte parole devono avere un peso positivo alto associato a un topic.

13.7 esperimento 5: Confronto con LDA per numero di topic

Nel presente esperimento, si vuole confrontare la LDA (ritenuta migliore rispetto ad altri tradizionali modelli come pLSI e NMF) con il modello RS. In particolare, si mettono a confronto la versione del modello RS con rmsprop e mfcd senza penalizzazione (con learning rate 0.0001 e batch size 20) e la stessa versione con livello di penalizzazione L2, learning rate, batch size e decay di rmsprop fissati con un processo di ottimizzazione bayesiana (con 15 modelli RS di prova da 20 epoche di training ciascuno, con topic diversity come metrica obiettivo). Il confronto avviene sul numero di topic dei modelli, da 4 a 20 con passo 4 (in altre parole si valuta ciascun modello per un numero di topic pari a 4,8,12,16 e 20, per risparmio di costo computazionale). I parametri usati per la LDA sono i seguenti: alpha pari a 1/(numero di topic), numero di documenti per iterazione (chunksize) 2000, 1000 iterazioni, 1 passo di update sul corpus per iterazione, e decay 0.005 (per la descrizione di questi iperparametri si rimanda a [Hoffman et al., 2010]).

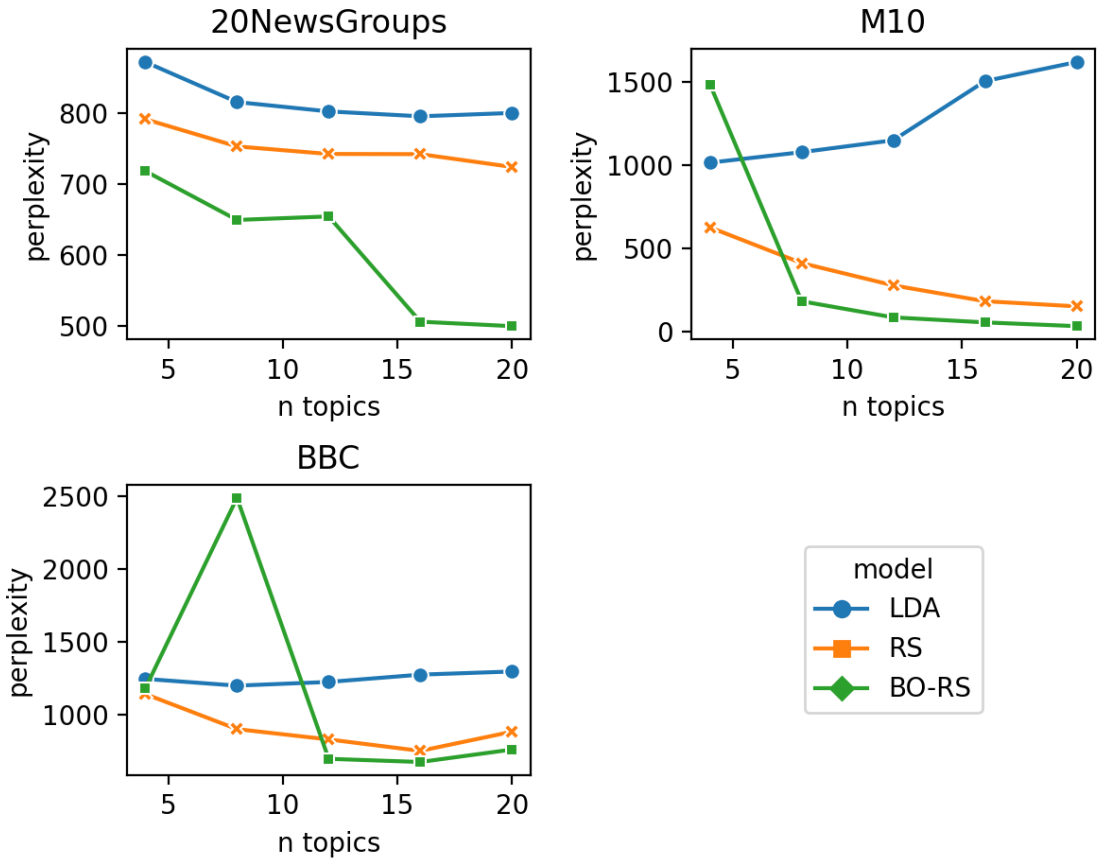


Figure 52: numero di topic contro perplexity per LDA e RS
Si vede che il modello RS con MFCD e rmsprop presenta sempre una perplexity migliore, e che l'ottimizzazione bayesiana dello stesso (BO-RS) comporta benefici rilevanti su questa metrica.

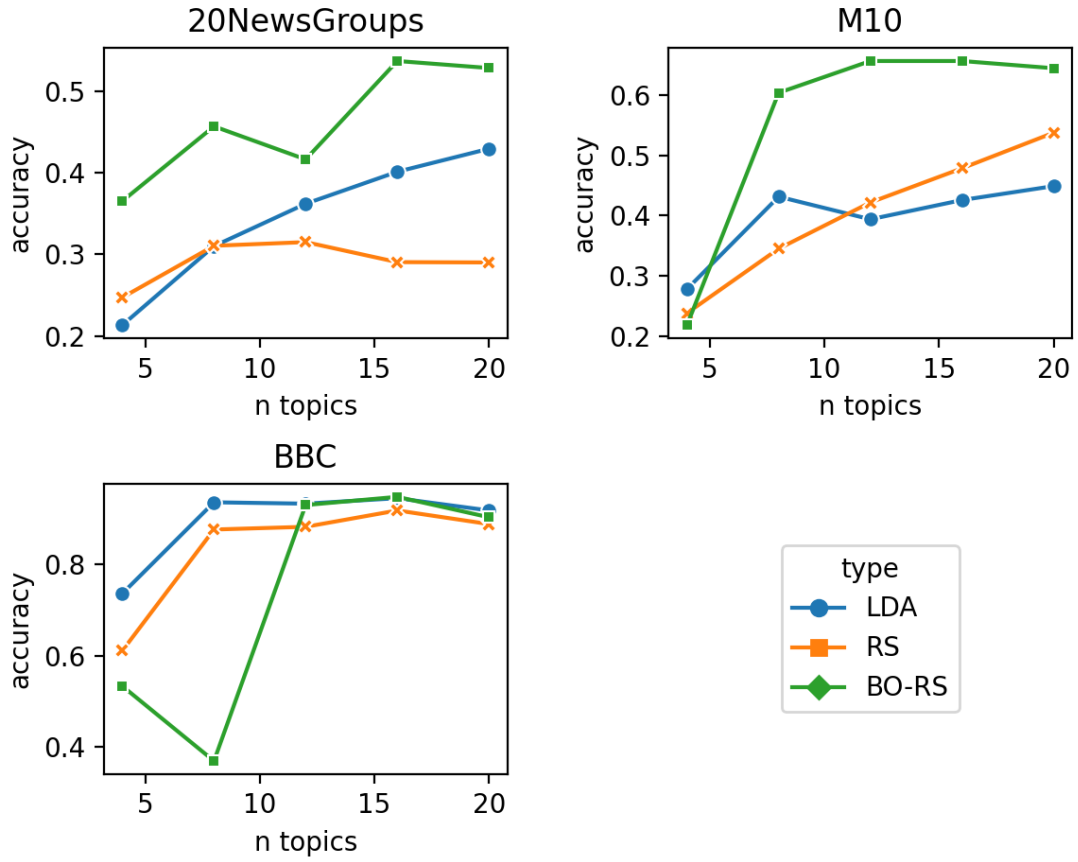


Figure 53: numero di topic contro perplexity per LDA e RS
I risultati in termini di accuracy riflettono quelli sulla perplexity. Si noti che variazioni repentine dell'accuracy, per alcuni dataset, si presentano proprio vicino al reale numero di topic.

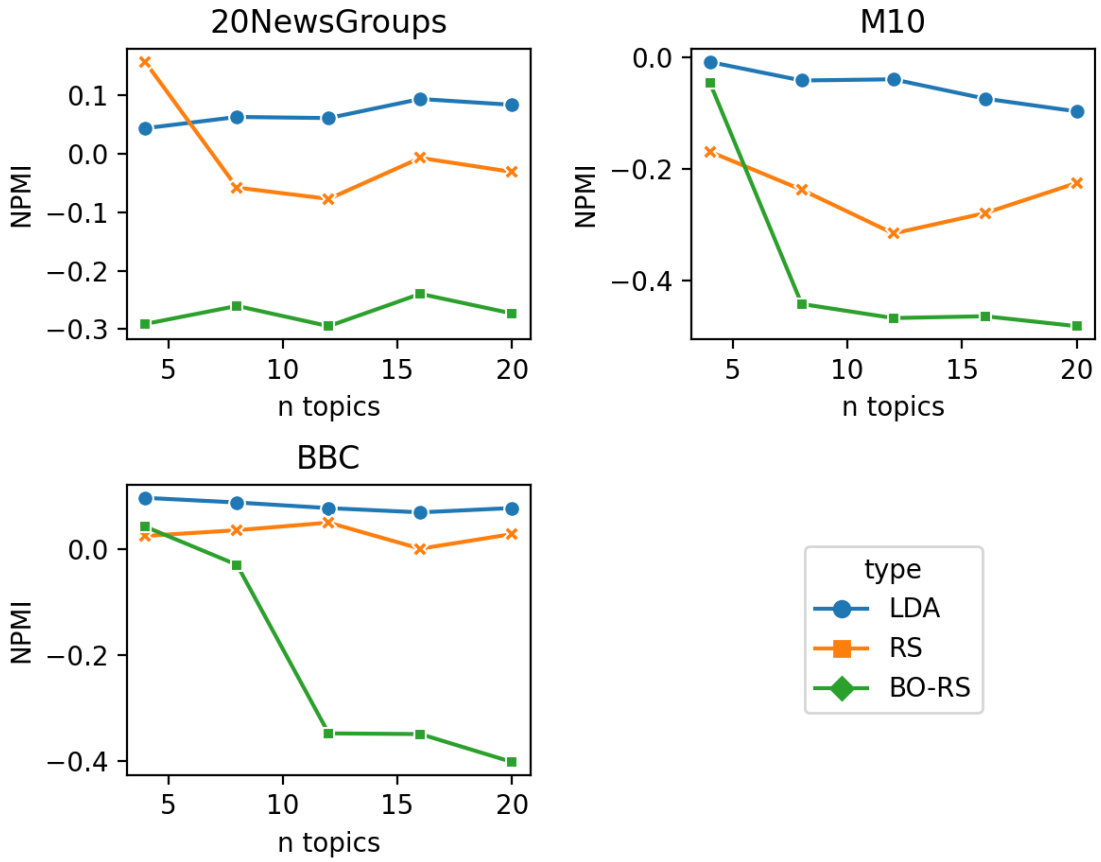


Figure 54: numero di topic contro NPMI coherence per LDA e RS
Si vede la netta superiorità della LDA sul modello RS in termini di NPMI. L'uso di un ottimizzatore rmsprop consente di mantenere un livello di coherence non di molto inferiore.

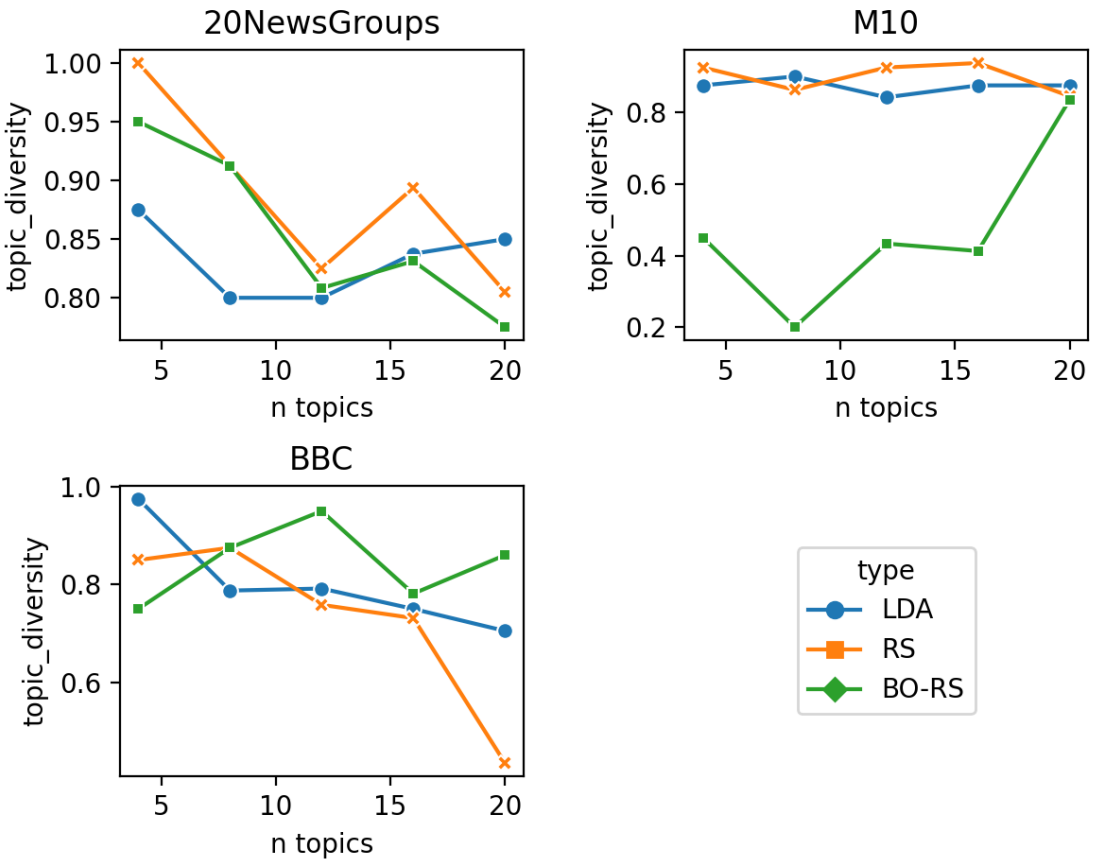


Figure 55: numero di topic contro topic diversity per LDA e RS
Su questa metrica l'ottimizzazione bayesiana conduce a risultati peggiori rispetto al modello RS non ottimizzato. Ciò è dovuto al fatto che il numero di epoche per le prove di ottimizzazione bayesiana era molto basso (20). La LDA performa in modo stabile sul numero di topic.

13.8 Interpretazione dei topic

In questa sezione si mostrano le tabelle con le prime 5 parole più importanti per ciascun topic. Queste tabelle derivano direttamente dalla matrice dei pesi di interazione, e sono ottenute ordinando per ogni topic le parole ad esso connesse con un peso più alto. La tabella viene poi troncata dopo la k-esima posizione. La tabella medesima, troncata dopo le 10 righe, è stata finora adottata per il calcolo dell'indice NPMI e della topic diversity. Le tavole mostrate di seguito comprendono solo le prime 5 parole più importanti per ciascun topic per i dataset 20NewsGroups e M10, e le prime 10 per BBC.

Ai fini di una maggiore chiarezza visiva, le colonne dei topic sono state anche ordinate in modo tale da porre in posizioni vicine topic a più alta somiglianza. L'ordinamento è stato prodotto dal dendrogramma di un modello di clustering agglomerativo con Complete Linkage e distanza del coseno applicata alla matrice dei pesi di interazione. Questo clustering tratta i topic come osservazioni e i pesi che li collegano alle visible units come features. Va detto che tale modello cambia solamente l'ordinamento delle righe della tabella descrittiva dei topic, senza modificarne il contenuto. Per completezza, si riportano sia le tabelle dei topic sia i dendrogrammi utilizzati per ordinarne le righe. Tali grafici possono rappresentare un supporto aggiuntivo nell'interpretazione dei topic.

Per semplicità, si mostrano in questa sezione solo le tavole dei modelli che per ciascun dataser si sono rivelati più interessanti ai fini dell'analisi. Si rimanda all'appendice A per una visione più esaustiva dei risultati di ciascun topic model. I dendrogrammi utilizzati per ordinare le stesse tavole sono riportati nell'appendice B.

In molti casi, come si può vedere dalle tavole in appendice A, non si ha una sufficiente differenziazione dei topic, da cui si deriva un valore basso di topic diversity. Spesso la coherence viene valutata più alta per topic model non addestrati, e risulta invece bassa per modelli che sembrano descrivere in modo abbastanza chiaro i topic. Si può osservare come il modello RS di base, con MFCD e senza alcun ottimizzatore di discesa del gradiente, porta a una descrizione uguale su tutti i topic, da cui un indice di topic diversity molto scarso come osservato nell'esperimento 1. Un miglioramento significativo dell'interpretabilità si può riconoscere nei topic costruiti dallo stesso modello con penalizzazione 0.001 di tipo L2 per i dataset 20NewsGroups e M10, e di tipo L1 per BBC. Si può constatare inoltre come

Table 4: 20NewsGroups: pRS con penalizzazione L2 di livello 0.001

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_{18}$	drive	people	support	find	question
$topic_3$	support	ground	information	bike	leave
$topic_{14}$	support	people	include	bike	question
$topic_1$	playoff	printer	russian	ranger	hockey
$topic_9$	score	car	app	traffic	universe
$topic_5$	wire	remote	mailing	bullet	survey
$topic_7$	video	blue	sale	bus	board
$topic_{15}$	truck	font	atheist	film	atheism
$topic_{13}$	atheism	turkish	criminal	encryption	crime
$topic_{16}$	stat	king	morality	verse	pray
$topic_6$	israeli	planet	backup	meg	temperature
$topic_{12}$	orbit	widget	motif	science	shuttle
$topic_{10}$	greek	genocide	sin	shell	soul
$topic_{11}$	homosexuality	gay	homosexual	player	season
$topic_8$	baseball	annual	disease	college	medicine
$topic_{17}$	chip	clipper	movie	mhz	tape
$topic_2$	encryption	modem	floppy	scsi	byte
$topic_{19}$	bike	circuit	voltage	motorcycle	brake
$topic_4$	brake	auto	dealer	tire	plane
$topic_{20}$	connector	voltage	battery	oil	vga

Molti topic sembrano ben definiti. Ad esempio il topic 20 sembra riguardare il lavoro di un elettricista-tuttofare, il 4 sarà inerente le auto, il 19 le moto e i veicoli elettrici, il topic 11 riguarderà il genere, il 10 le religioni, e ragionamenti analoghi possono farsi per gli altri topic.

l'ottimizzatore rmsprop migliora l'indice di topic diversity in tutti i dataset (e ciò avviene coerentemente con i risultati descritti nell'esperimento 3). Infine, si vede come i topic model ottimizzati con BO presentano una cattiva costruzione dei topic. Questo è coerente con i risultati dell'esperimento 4, nel quale si vede come la topic diversity, sebbene massimizzata nelle prime 100 epoche di training, presenti performance decrescenti nelle epoche successive. I topic presentati in tabella sono infatti costruiti a partire dalla matrice dei pesi del modello ottenuto dopo 1000 iterazioni di training. I modelli che fanno uso di ottimizzazione bayesiana sono quindi ottimali se addestrati per un ridotto numero di epoche, di poco superiore al numero di epoche di training assegnate in fase di collezione di data points per il processo di ottimizzazione bayesiana.

Table 5: M10: pRS con penalizzazione L2 di livello 0.001

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_6$	decision	policy	make	power	information
$topic_2$	algorithm	time	dna	sequence	efficient
$topic_5$	system	neural	network	computation	genetic
$topic_3$	material	fuel	technique	regulation	site
$topic_4$	simulation	theory	rate	high	water
$topic_7$	datum	gene	time	real	expression
$topic_8$	model	climate	stock	change	access
$topic_{10}$	mechanic	protein	local	magnetic	structure
$topic_1$	evolutionary	human	selection	stochastic	rule
$topic_9$	base	detection	recognition	presentation	feature

Su questo dataset, quella che segue è l'unica descrizione con un alto grado di differenziazione dei topic, tra quelle prodotte negli esperimenti 1-4. In grado di costruire risultati con topic diversity accettabile su questo dataset sono presenti solo i modello non penalizzati con adam (che presenta perplexity minima) e rmsprop. In questo dataset, rmsprop primeggia in topic diversity, adam in perplexity, mentre pRS-L2 presenta un grado di coherence maggiore, senza mostrare cattive performance sulle altre metriche.

Table 6: BBC : pRS con penalizzazione L1 di livello 0.001

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_1$	award	tax	economy	labour	election
$topic_5$	firm	company	market	analyst	share
$topic_3$	game	player	award	win	club
$topic_2$	game	club	match	team	side
$topic_4$	club	government	win	claim	country

Dalle metriche dell'esperimento 2 questo modello sembra presentare le migliori performance sul dataset BBC sotto ogni metrica. Le sue performance su questo dataset (accuracy intorno a 0.8, perplexity inferiore a 1500, topic diversity intorno a 0.8, coherence positiva) sono molto simili a quelle del modello RS non penalizzato con ottimizzazione RMSprop.

14 Over-RSM : esperimenti sui dataset di OCTIS

In questo capitolo si confrontano gli esiti ottenuti con il modello RSM con quelli del modello Over-RSM. In questo caso ci si limita a valutare le potenzialità del modello Over RSM con diversi metodi di contrastive divergence e con diversi metodi di ottimizzazione di gradient descent.

14.1 esperimento 6: comparazione di diverse tecniche di contrastive learning per il modello Over-RSM

Si ripete l'esperimento 1, utilizzando questa volta il modello oRSM. I valori degli iperparametri restano gli stessi assegnati di default negli esperimenti precedenti. Il numero di topic si assume ancora noto a priori. Infine, l'iperparametro M, ovvero la forza della prior sul contenuto del documento, viene fissato pari a 50, come fatto in [Srivastava et al., 2013]. In altre parole, si assume che ogni documento deve avere almeno 50 parole. Nei documenti con più di 50 parole, il visible layer avrà più forza del prior hidden layer. Nei documenti con poche parole, il prior layer prevarrà nella descrizione del documento. Il numero di epoche di pretraining è fissato a 500, mentre il numero totale di epoche di training è fissato a 1000. Bisogna notare che è possibile differenziare i metodi di contrastive divergence durante le epoche di pretraining e dopo la loro conclusione. Per semplicità, in questo esperimento si adotterà lo stesso metodo per la prima e per la seconda fase di training.

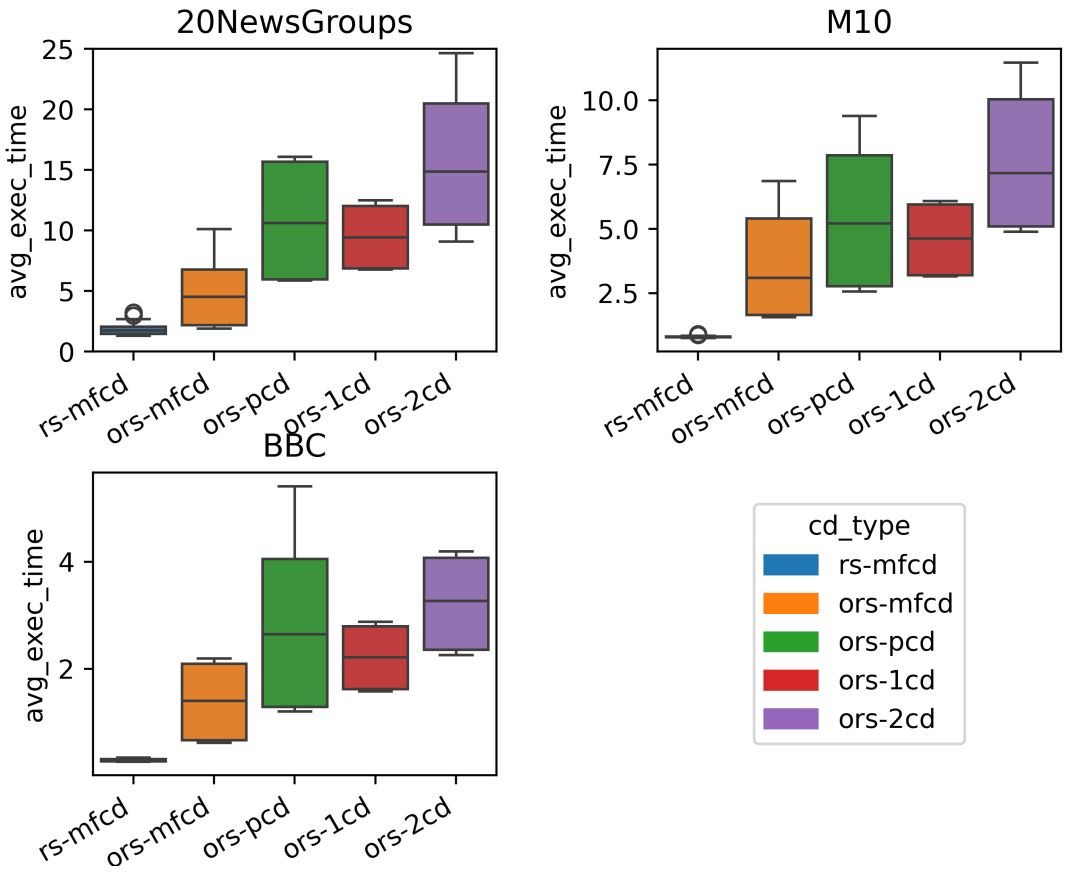


Figure 56: cd per oRSM : secondi per iterazione

Si possono fare analoghe considerazioni a quelle fatte sul costo computazionale dei diversi metodi di CD nell'esperimento 1. Il metodo MFCD risulta significativamente più veloce, anche per il modello oRSM. Tuttavia le distanze tra MFCD e k-step CD sembrano essersi ridotte rispetto a quello che accadeva nell'esperimento 1. Questo è dovuto al fatto che il modello oRSM è strutturalmente più lento del modello RSM in fase di inferenza e campionamento. L'alto range dei boxplot è spiegato dalla diversa durata di un'iterazione di pre-training e di un'iterazione di training vero e proprio (nella quale si svolge un processo di convergenza per ciascun batch). Si noti che la MFCD applicata al RS classico presenta un costo computazionale significativamente inferiore, e con un range più ristretto.

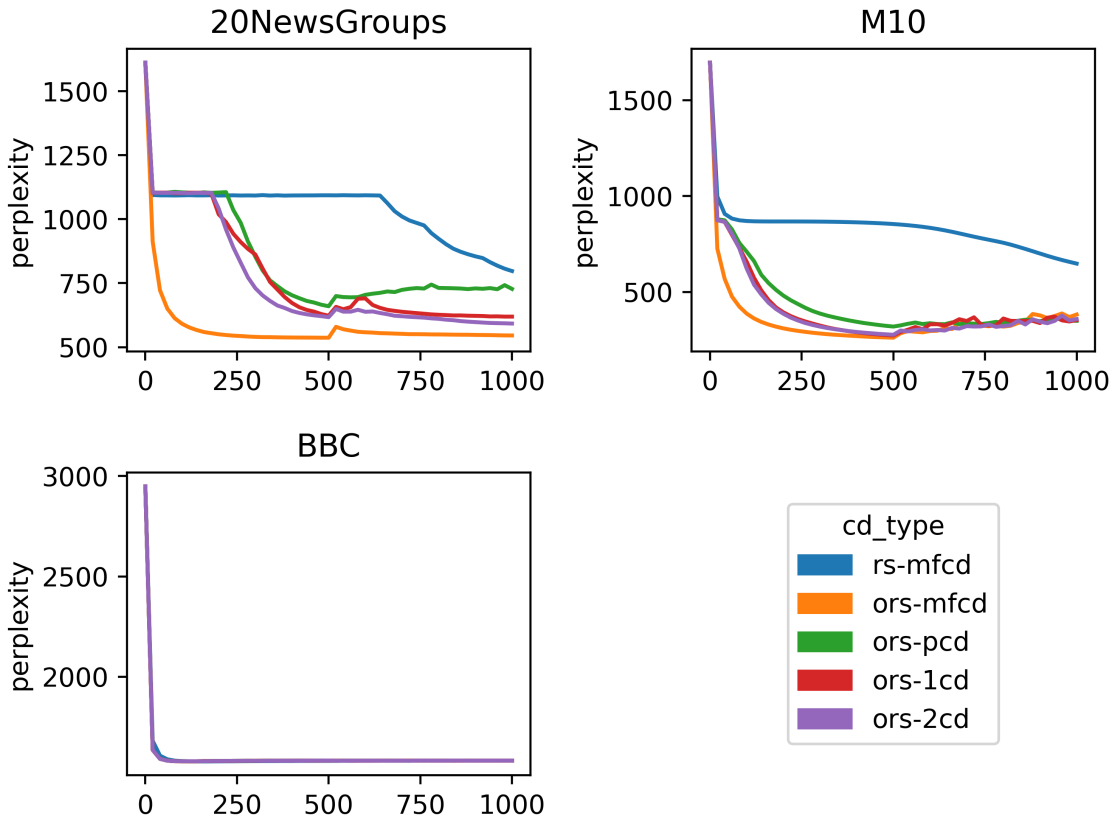


Figure 57: cd per oRSM : perplexity
Nei dataset 20NewsGroups e M10 il metodo MFCD accelera il training, come accadeva nell'esperimento 1, mentre non differisce dagli altri metodi per il dataset BBC. Si nota che all'iterazione 500 si presenta un'anomalia, che tende a stabilizzarsi nelle iterazioni successive. Questo accade perché da quell'iterazione si ha un passaggio dalla prima fase alla seconda fase di training. Nella seconda fase di training non si registrano miglioramenti, da cui si può dedurre che il modello oRSM richieda più di 1000 iterazioni per sfruttare le sue potenzialità sotto questa metrica.

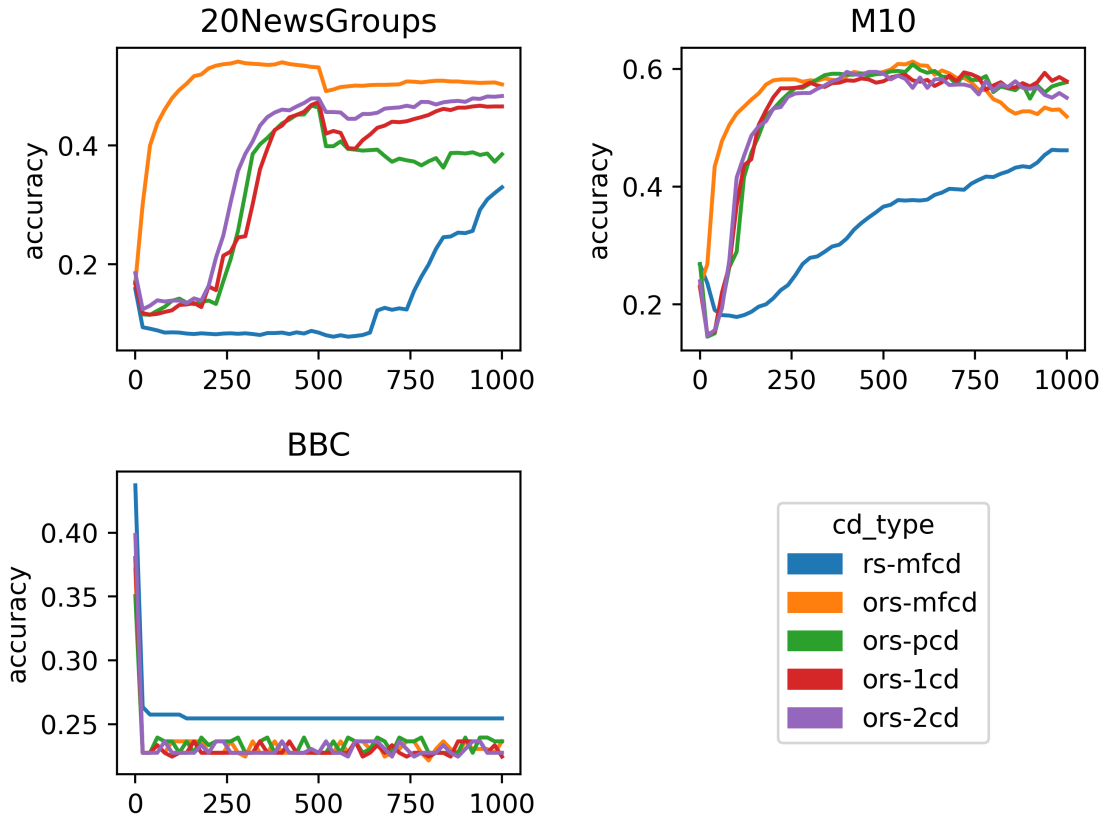


Figure 58: cd per oRSM : accuracy
Valgono considerazioni analoghe a quelle fatte per la perplexity. Si noti che sul dataset BBC presenta performance migliori il modello RSM rispetto a qualsiasi tipo di CD del modello oRSM, mentre vale il contrario per i dataset 20NewsGroups e M10.

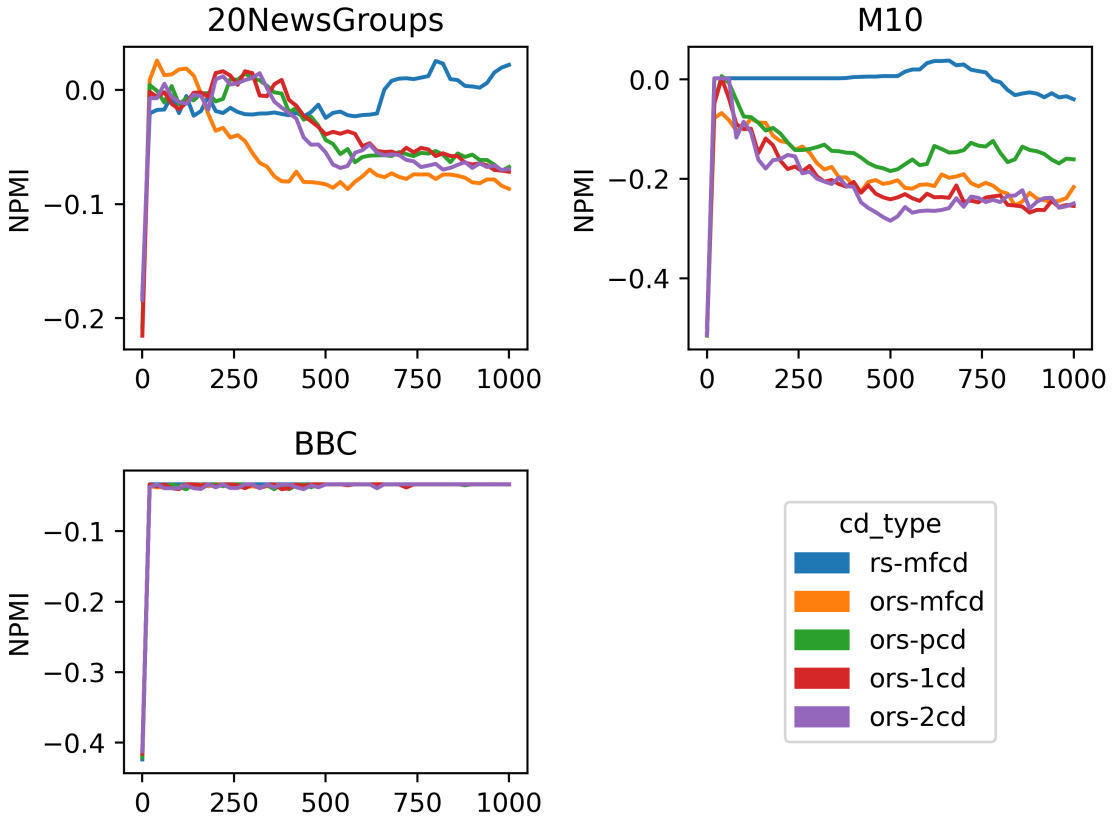


Figure 59: cd per oRSM : coherence NPMI
Su questa metrica il modello oRSM performa sempre peggio della versione del modello RSM con MFCD.

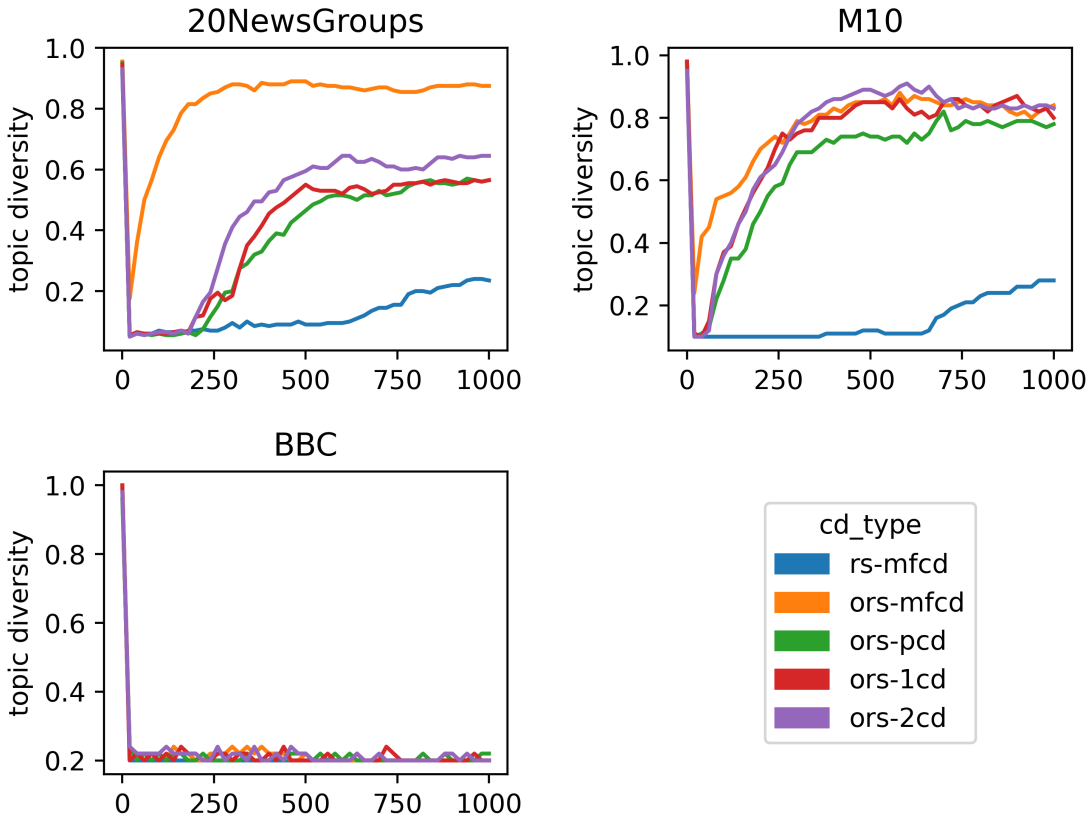


Figure 60: cd per oRSM : topic diversity
Come per la perplexity, sui primi due dataset il metodo MFCD accelera il miglioramento del modello.

14.2 esperimento 7: comparazione di diverse tecniche di gradient descent per il modello Over-RSM

Si ripete qui l'esperimento 2, rispetto al modello oRSM. Si vuole quindi analizzare l'impatto che la scelta di un diverso ottimizzatore di gradient descent ha sul training e sul pretraining del modello oRSM. Anche qui, per tutti i dataset di OCTIS, si fissano 1000 epoche di training complessive, con le prime 500 eseguite secondo il metodo di pretraining (imponendo l'uguaglianza tra l'unità softmax del visible layer e quella del terzo layer) e si pone M=50. Analoghamente a come fatto nell'esperimento 3, si adotta sempre la MFCD. I risultati sono messi a confronto con quelli del modello RSM con MFCD e ottimizzazione rmsprop.

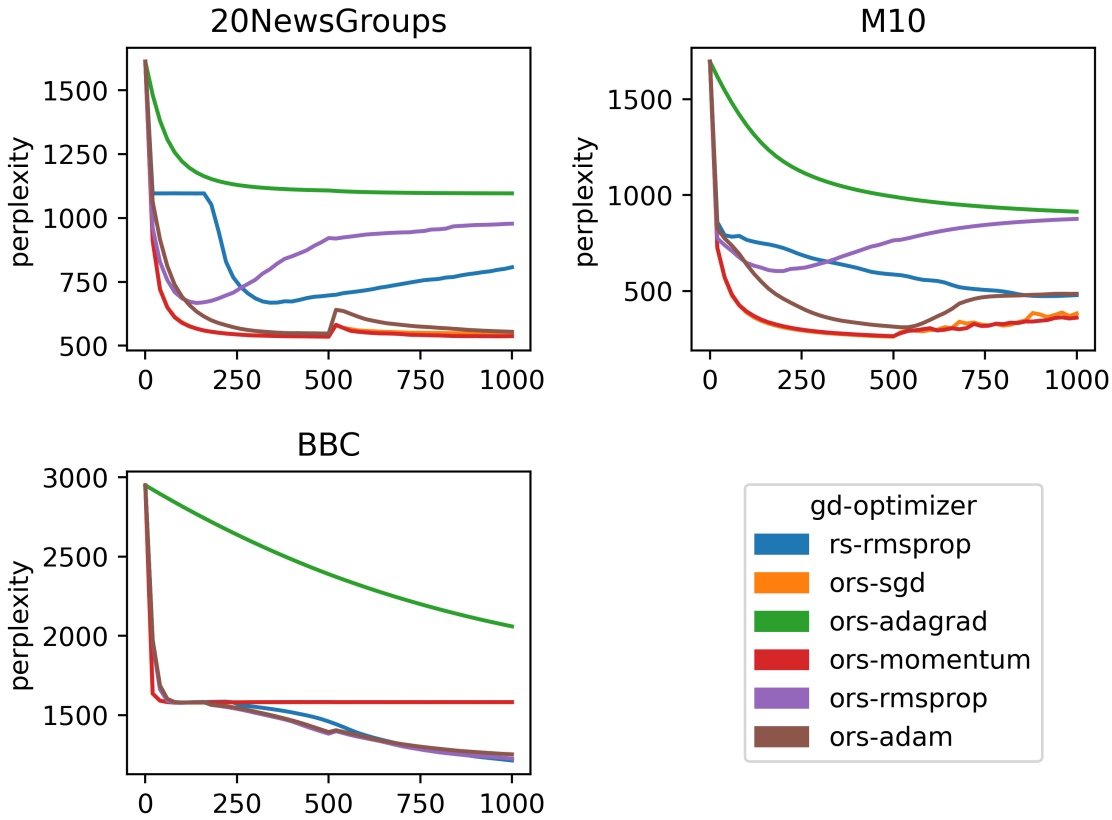


Figure 61: optimizers per oRSM : perplexity
I metodi adam e momentum sembrano performare meglio in quasi tutti i dataset. Il metodo momentum non conduce a risultati significativamente diversi da quelli del gradient descent senza ottimizzatore. Il metodo adagrad, come nel caso del modello RSM classico, non apporta alcun beneficio.

Il modello Over-RS si presenta più robusto rispetto al modello RS rispetto a peggioramenti

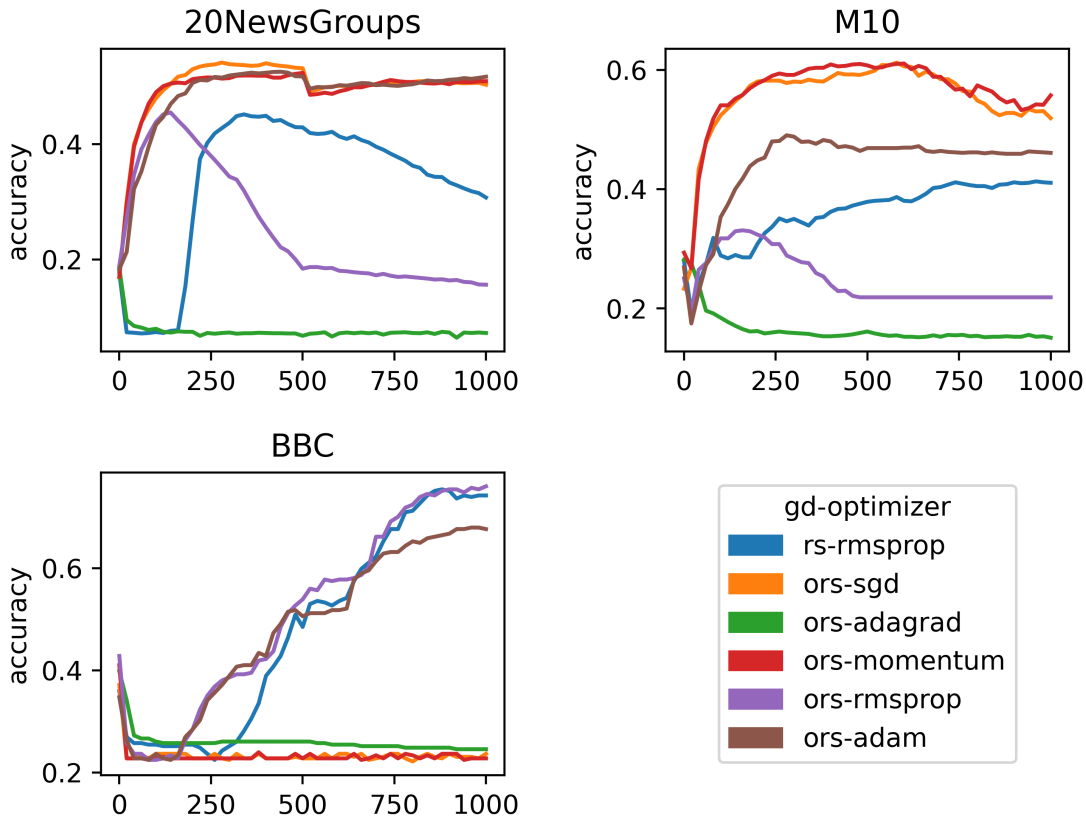


Figure 62: optimizers per oRSM : accuracy
Il metodo rmsprop sembra risultare molto meno efficace nel modello Over-RSM rispetto al caso in cui viene applicato al modello RS classico. Risultano equivalenti nel caso del dataset BBC.

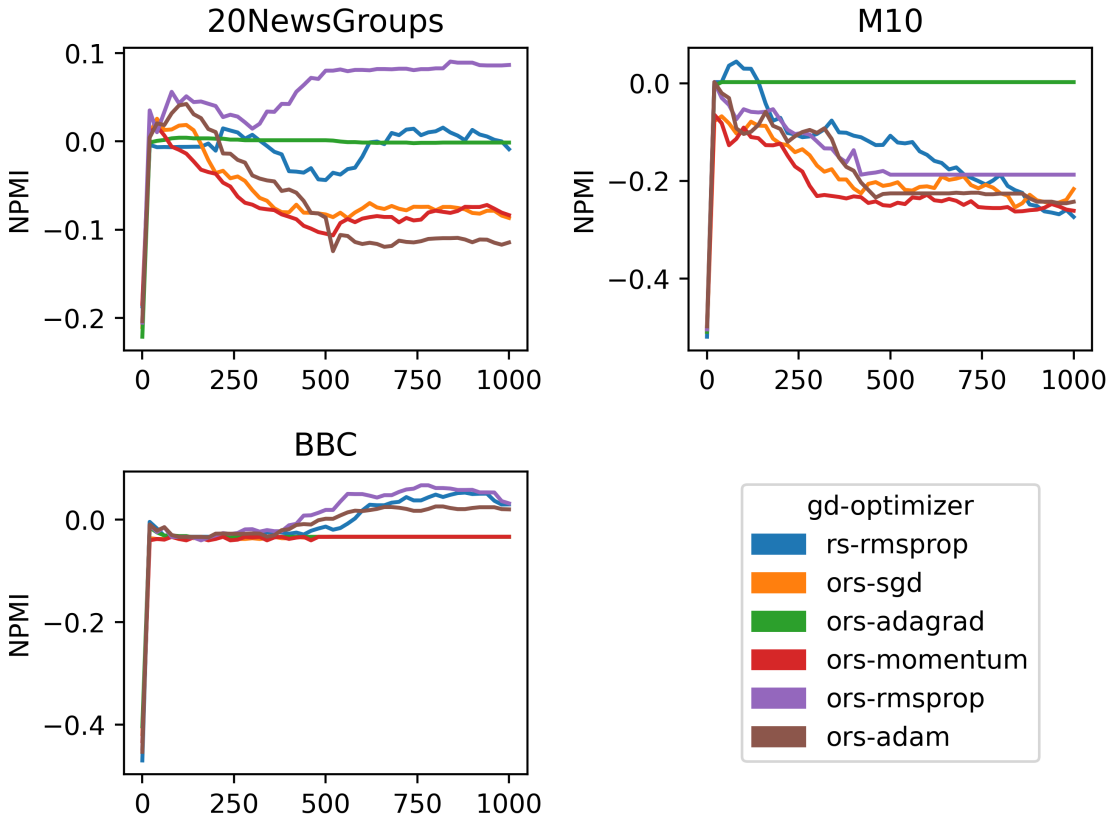


Figure 63: optimizers per oRSM : NPMI coherence

Si vede che il metodo rmsprop applicato al modello oRSM porta a livelli di coherence più alti per i dataset 20NewsGroups e BBC. In generale, la coherence del modello oRSM risulta migliorata se messa a confronto con quella del modello RSM, come visto nell'esperimento 3.

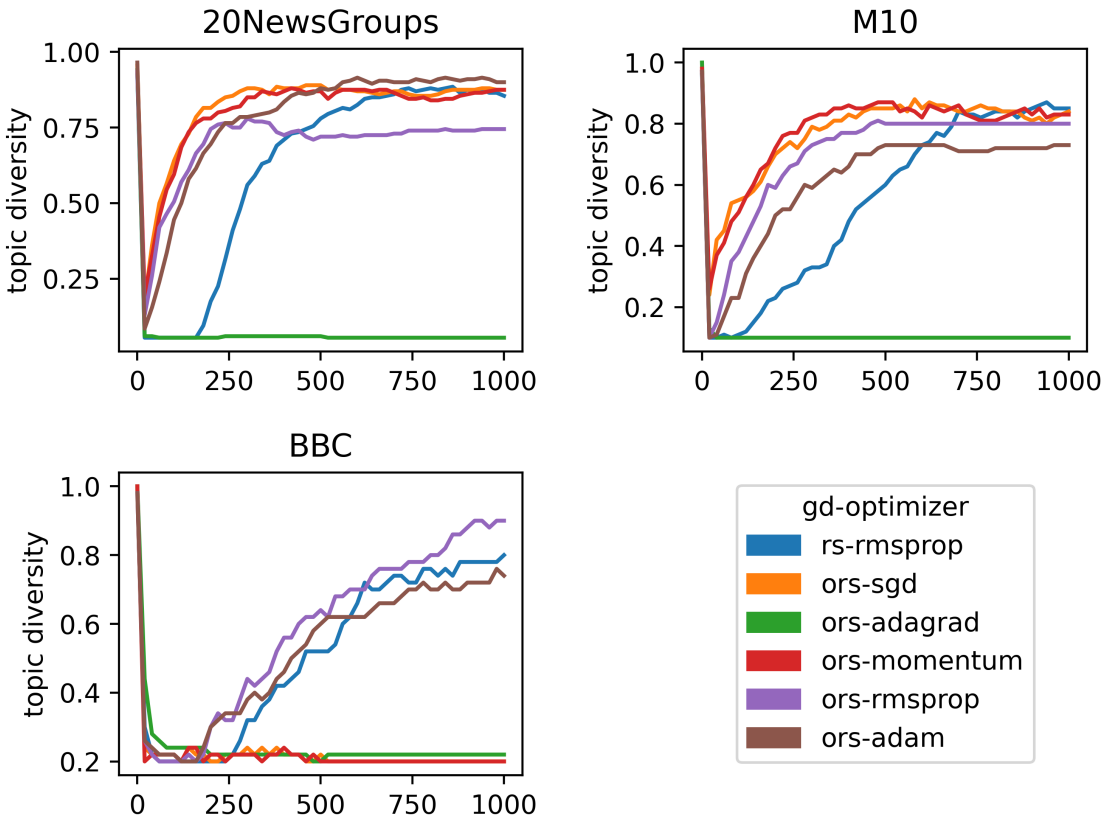


Figure 64: optimizers per oRSM : topic diversity
Il modello oRSM sembra consentire di raggiungere livelli di topic diversity più alti rispetto al modello RSM, persino quando questo presenta iperparametri fissati con ottimizzazione bayesiana su un numero ridotto di epoche (si veda il grafico della topic diversity dell'esperimento 4).

Table 7: 20NewsGroups: Over-RS con ottimizzazione rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_1$	port	driver	serial	printer	meg
$topic_7$	format	file	image	server	version
$topic_{20}$	faith	sin	scripture	passage	biblical
$topic_{16}$	game	baseball	playoff	score	team
$topic_{18}$	ide	scsi	boot	controller	pitcher
$topic_5$	armenian	village	massacre	turkish	soldier
$topic_{17}$	gun	insurance	ticket	bike	ride
$topic_3$	monitor	card	slot	connector	video
$topic_9$	bike	car	engine	ride	tire
$topic_4$	encrypt	escrow	chip	clipper	key
$topic_6$	enforcement	administration	gun	crime	criminal
$topic_{15}$	address	mailing	internet	list	clipper
$topic_{14}$	genocide	turkish	soviet	muslim	armenian
$topic_2$	source	widget	terminal	printer	mouse
$topic_{10}$	driver	sale	controller	drive	scsi
$topic_{11}$	advance	driver	serial	controller	turkish
$topic_{13}$	connector	ground	pin	wheel	front
$topic_{19}$	vga	sale	offer	monitor	condition
$topic_8$	orbit	satellite	convert	image	earth
$topic_{12}$	shipping	condition	sale	offer	book

delle metriche di performance nel corso delle iterazioni. Non risulta più conveniente in termini computazionali, di perplexity e di accuracy, ma presenta potenzialità maggiori in termini di coherence e topic diversity. Anche senza un'attento tuning dei suoi iperparametri, è possibile ottenere livelli di perplexity molto bassi in un numero ridotto di iterazioni. In generale, la MFCD non sembra una scorciatoia sufficiente per ottenere benefici dalla seconda fase di training di questo modello.

14.3 Interpretazione dei topic

Analogamente a come fatto per gli esperimenti 1-4, si riportano di seguito le tavole dei topic migliori, per dataset, prodotte dal modello Over-RS in questo capitolo. Se le descrizioni migliori (in base al giudizio soggettivo di chi scrive) del modello RS erano tutte prodotte dal pRS, in questo capitolo i migliori topic vengono formati da Over-RS con ottimizzazione rmsprop.

Table 8: M10: Over-RS con ottimizzazione rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_7$	artificial	approach	face	hybrid	control
$topic_{10}$	markov	recognition	regulatory	hide	logic
$topic_9$	networked	volatility	interest	symposium	rate
$topic_4$	natural	gas	pricing	composite	option
$topic_8$	proceeding	sense	remote	estimate	forecast
$topic_3$	formal	making	group	fault	decision
$topic_1$	symposium	presentation	poster	reinforcement	grammatical
$topic_2$	symposium	poster	bayesian	presentation	neural
$topic_5$	hole	black	fold	protein	material
$topic_6$	return	monetary	risk	work	market

Table 9: BBC: Over-RS con ottimizzazione rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_3$	game	player	play	film	season
$topic_4$	election	tax	labour	economy	taxis
$topic_5$	firm	company	government	analyst	market
$topic_1$	include	give	work	back	add
$topic_2$	film	win	man	game	work

15 conclusioni

L'obiettivo di questa tesi era quello di fornire una panoramica del modello Replicated Softmax, e di ottimizzarne l'implementazione al fine di renderlo uno strumento pratico e adeguato per comuni task di topic modeling.

I risultati sperimentali ottenuti nei capitoli 13 e 14 mettono in evidenza alcune importanti informazioni su come sia possibile ottimizzare questo modello.

- Il metodo MFCD fornisce una drastica riduzione del tempo di training
- Un fattore di penalizzazione può migliorare rapidamente il modello
- L'adozione di ottimizzatori di gradient descent può anch'essa portare a migliroamente significativi
- La configurazione degli iperparametri attraverso l'ottimizzazione bayesiana velocizza notevolmente la scelta degli iperparametri, cruciale nel topic modeling
- Il modello Over-RS non presenta per poche iterazioni vantaggi significativi sul RS classico, ma può rappresentare un'alternativa robusta

Numerosi sono gli esperimenti che si sarebbero potuti condurre, e che non sono stati presi in esame. In particolare:

- Non si è affrontato il tema della scelta del numero di topic. Si può pensare al processo di ottimizzazione bayesiana come a uno strumento per identificarlo, a fronte di un numero di run del modello e di un numero di epoche di training adeguatamente alti.
- Come per il numero di topic, anche il parametro M presente nel modello Over-RSM potrebbe trarre vantaggio dall'adozione di una strategia di ottimizzazione bayesiana basata su un numero sufficientemente alto di epoche di training.
- Sarebbe interessante riproporre il processo di ottimizzazione bayesiana sul modello RSM applicando un ottimizzatore diverso da rmsprop, come adam e momentum, e adottando una funzione obiettivo differente dalla topic diversity, come l'indice NPMI o l'accuracy.

- Non sono state svolte comparazioni delle performance del modello RSM con quelle di altri topic model più avanzati basati sui VAE, come ProdLDA, AEVB e AVITM. Pur essendo probabile che questi modelli risultino sotto ogni metrica qui considerata superiori al modello RSM ottimizzato, è possibile che con le strategie qui presentate questo si riveli un'opzione computazionalmente più efficiente.

- Non sembra inverosimile che il parametro K della K-step contrastive divergence possa portare benefici quando portato a valori alti, come 4 o 5, in casi in cui il numero di iterazioni disponibili possa essere effettivamente alto. In alcuni lavori sulle RBM si è proposto di accrescere gradualmente K nel corso delle iterazioni.

Complessivamente, questo lavoro mette in evidenza le potenziali applicazioni di strumenti come la MFCD, la penalizzazione dei pesi, l'ottimizzazione del gradiente e gli strumenti di ottimizzazione bayesiana nel mondo delle RBM e del topic modeling.

Si è evidenziato come l'utilizzo combinato di diversi metodi di ottimizzazione possa portare il modello RS a mostrare in pochi minuti livelli di performance che con metodi di training più semplici sarebbe stato in grado di raggiungere solamente in ore o giorni, o che non avrebbe raggiunto affatto.

Il modello RS rappresenta quindi un'alternativa adeguata ed elegante rispetto ai topic models tradizionali.

Table 10: M10 - RS semplice con mfcd

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_2$	gas	market	stock	paper	effect
$topic_7$	gas	market	stock	paper	effect
$topic_9$	gas	market	model	decision	natural
$topic_{10}$	model	gas	natural	market	base
$topic_3$	neural	network	gene	expression	datum
$topic_4$	model	network	neural	gene	expression
$topic_1$	system	control	model	base	decision
$topic_6$	system	control	base	model	decision
$topic_5$	network	system	neural	control	recurrent
$topic_8$	network	system	neural	control	gene

Table 11: M10 - RS con penalizzazione L2 0.001

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_6$	decision	policy	make	power	information
$topic_2$	algorithm	time	dna	sequence	efficient
$topic_5$	system	neural	network	computation	genetic
$topic_3$	material	fuel	technique	regulation	site
$topic_4$	simulation	theory	rate	high	water
$topic_7$	datum	gene	time	real	expression
$topic_8$	model	climate	stock	change	access
$topic_{10}$	mechanic	protein	local	magnetic	structure
$topic_1$	evolutionary	human	selection	stochastic	rule
$topic_9$	base	detection	recognition	presentation	feature

A Appendice tavole dei topic

Table 12: M10 - RS con penalizzazione L1 0.001

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_7$	high	index	dna	soil	natural
$topic_6$	base	model	protein	structure	system
$topic_5$	index	policy	soil	rate	high
$topic_8$	high	index	policy	soil	rate
$topic_{10}$	high	policy	index	soil	dna
$topic_3$	time	real	space	code	finance
$topic_2$	decision	make	making	agent	support
$topic_4$	gene	expression	datum	cluster	microarray
$topic_1$	system	neural	network	design	recurrent
$topic_9$	model	network	neural	recurrent	learn

Table 13: M10 - RS con rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_2$	water	symbolic	symposium	method	combine
$topic_7$	symposium	presentation	poster	paper	oral
$topic_9$	reinforcement	pricing	hole	black	poster
$topic_8$	gauge	hole	black	grammatical	fault
$topic_5$	bind	transcription	proceeding	forecast	sense
$topic_6$	making	decision	make	agent	sequential
$topic_1$	web	formal	sequential	real	tree
$topic_4$	expression	biclustering	infer	inference	regulatory
$topic_3$	conformal	observable	mechanic	carlo	laminate
$topic_{10}$	symposium	networked	presentation	feedback	control

Table 14: M10 - bayesian optimized RS con rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_6$	book	leak	lake	preserve	renormalization
$topic_2$	book	lake	leak	story	japanese
$topic_7$	leak	lake	book	story	japanese
$topic_{10}$	book	lake	leak	japanese	acknowledgment
$topic_5$	lake	book	leak	story	japanese
$topic_1$	book	lake	leak	story	japanese
$topic_8$	lake	book	leak	story	renormalization
$topic_3$	lake	book	leak	story	japanese
$topic_4$	leak	lake	book	story	japanese
$topic_9$	book	leak	lake	story	japanese

Table 15: M10 - over-RS

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_4$	social	make	decision	quantum	diesel
$topic_{10}$	market	stock	diesel	performance	water
$topic_6$	experience	search	capital	black	online
$topic_9$	gas	carbon	energy	microstructure	option
$topic_2$	dna	water	expression	protein	real
$topic_1$	bayesian	capital	case	inference	intelligence
$topic_5$	inference	financial	market	option	volatility
$topic_7$	method	algorithm	application	discrete	equation
$topic_3$	learn	learning	recognition	language	agent
$topic_8$	network	neural	control	analysis	system

Table 16: M10 - over-RS con rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_7$	artificial	approach	face	hybrid	control
$topic_{10}$	markov	recognition	regulatory	hide	logic
$topic_9$	networked	volatility	interest	symposium	rate
$topic_4$	natural	gas	pricing	composite	option
$topic_8$	proceeding	sense	remote	estimate	forecast
$topic_3$	formal	making	group	fault	decision
$topic_1$	symposium	presentation	poster	reinforcement	grammatical
$topic_2$	symposium	poster	bayesian	presentation	neural
$topic_5$	hole	black	fold	protein	material
$topic_6$	return	monetary	risk	work	market

Table 17: BBC News - RS semplice con mfcd

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_1$	good	game	win	play	government
$topic_3$	good	game	win	play	government
$topic_4$	game	good	win	play	government
$topic_2$	good	game	win	government	play
$topic_5$	good	game	win	play	government

Table 18: BBC News - RS con penalizzazione L2 0.001

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_2$	win	good	game	government	show
$topic_1$	win	good	game	government	show
$topic_5$	win	good	game	government	show
$topic_3$	win	good	game	government	show
$topic_4$	win	good	game	government	show

Table 19: BBC News - RS con penalizzazione L1 0.001

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_1$	award	tax	economy	labour	election
$topic_5$	firm	company	market	analyst	share
$topic_3$	game	player	award	win	club
$topic_2$	game	club	match	team	side
$topic_4$	club	government	win	claim	country

Table 20: BBC News - RS con rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_4$	play	win	award	beat	player
$topic_5$	market	economy	analyst	growth	firm
$topic_2$	award	election	law	party	labour
$topic_1$	game	player	play	test	win
$topic_3$	government	work	world	give	game

Le sue performance su questo dataset (accuracy intorno a 0.8, perplexity inferiore a 1500, topic diversity intorno a 0.8, coherence positiva) sono molto simili a quelle del modello RS penalizzato con L1 al livello 0.001.

Table 21: BBC News - bayesian optimized RS con rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_2$	receptive	master	manufacturing	classical	graph
$topic_4$	neural	robotic	robot	modularity	reinforcement
$topic_1$	plan	crop	inverse	remote	educational
$topic_3$	supply	petroleum	protein	export	crude
$topic_5$	engine	diesel	monthly	blackout	outage

Table 22: BBC News - over-RS

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_5$	game	good	win	play	government
$topic_3$	game	good	win	play	government
$topic_2$	game	good	win	play	government
$topic_1$	game	good	win	play	government
$topic_4$	game	good	win	play	government

Si vede che la topic diversity è 0.

Table 23: BBC News - over-RS con rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_3$	game	player	play	film	season
$topic_4$	election	tax	labour	economy	taxis
$topic_5$	firm	company	government	analyst	market
$topic_1$	include	give	work	back	add
$topic_2$	film	win	man	game	work

Table 24: 20NewsGroups - RS semplice con mfcd

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_{10}$	armenian	game	team	turkish	window
$topic_{19}$	encryption	law	government	key	clipper
$topic_{20}$	key	file	window	server	chip
$topic_{16}$	image	file	space	program	launch
$topic_1$	car	time	good	make	bike
$topic_4$	car	time	good	make	bike
$topic_3$	car	time	high	power	make
$topic_6$	car	time	high	power	problem
$topic_{18}$	car	time	high	power	bike
$topic_2$	car	good	make	year	bike
$topic_{13}$	car	good	make	bike	year
$topic_{15}$	car	good	make	year	bike
$topic_{17}$	car	good	time	bike	make
$topic_8$	car	good	make	time	bike
$topic_{14}$	car	good	time	make	year
$topic_5$	car	good	make	year	bike
$topic_9$	car	good	time	make	bike
$topic_{11}$	car	good	make	year	bike
$topic_7$	car	good	make	time	year
$topic_{12}$	car	good	make	year	bike

Table 25: 20NewsGroups - RS con penalizzazione L2 0.001

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_{18}$	drive	people	support	find	question
$topic_3$	support	ground	information	bike	leave
$topic_{14}$	support	people	include	bike	question
$topic_1$	playoff	printer	russian	ranger	hockey
$topic_9$	score	car	app	traffic	universe
$topic_5$	wire	remote	mailing	bullet	survey
$topic_7$	video	blue	sale	bus	board
$topic_{15}$	truck	font	atheist	film	atheism
$topic_{13}$	atheism	turkish	criminal	encryption	crime
$topic_{16}$	stat	king	morality	verse	pray
$topic_6$	israeli	planet	backup	meg	temperature
$topic_{12}$	orbit	widget	motif	science	shuttle
$topic_{10}$	greek	genocide	sin	shell	soul
$topic_{11}$	homosexuality	gay	homosexual	player	season
$topic_8$	baseball	annual	disease	college	medicine
$topic_{17}$	chip	clipper	movie	mhz	tape
$topic_2$	encryption	modem	floppy	scsi	byte
$topic_{19}$	bike	circuit	voltage	motorcycle	brake
$topic_4$	brake	auto	dealer	tire	plane
$topic_{20}$	connector	voltage	battery	oil	vga

Table 26: 20NewsGroups - RS con penalizzazione L1 0.001

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_{11}$	planet	atheist	moon	surface	solar
$topic_{19}$	image	color	bit	format	graphic
$topic_1$	problem	window	work	font	part
$topic_9$	window	manager	screen	work	application
$topic_{18}$	display	program	widget	application	font
$topic_4$	planet	atheist	moon	surface	solar
$topic_{13}$	system	screen	work	problem	time
$topic_2$	chip	clipper	encryption	phone	bit
$topic_{16}$	drive	scsi	disk	hard	ide
$topic_{10}$	people	thing	government	gay	kill
$topic_{12}$	make	offer	word	engine	trade
$topic_5$	planet	atheist	moon	surface	earth
$topic_7$	year	child	law	life	kill
$topic_3$	planet	atheist	moon	earth	surface
$topic_{15}$	launch	space	satellite	mission	surface
$topic_{14}$	gun	firearm	rate	control	defense
$topic_8$	time	homosexual	rom	original	direct
$topic_{20}$	bus	access	define	line	color
$topic_6$	good	offer	excellent	objective	car
$topic_{17}$	game	team	player	play	season

Si noti come i topic 9 e 18 presentano parole simili e siano per questo affiancati. Dalle prime 5 parole, si vede che alcuni topic sembrano duplicati (come i topic 11, 3, 5 e 4, che riportano descrizioni identiche, ma non risultano affiancati). Molti topic sembrano tuttavia molto ben definiti: 11, 3, 5 e 4 parlano di spazio e pianeti, 19 parla di immagini, 18 parla di software, 13, 2 e 16 parlano di hardware (e sono affiancati), 15 parla di viaggi nello spazio, 14 parla di armi, 10 parla di politica, e 17 parla di giochi. Non saprei denominare con la stessa sicurezza i significati degli altri topic, se ve ne sono. L'esercizio di denominazione dei topic a partire dalle top-k parole che richiamano è di natura puramente soggettiva.

Table 27: 20NewsGroups - RS con rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_{12}$	cop	batf	execute	mouse	fine
$topic_{17}$	ride	bike	motorcycle	principle	broadcast
$topic_8$	cap	home	devil	lose	pen
$topic_{14}$	surrender	trade	stat	cell	harm
$topic_{11}$	clipper	scheme	chip	encryption	algorithm
$topic_5$	convert	voltage	define	pin	condition
$topic_{19}$	surrender	ide	controller	scsi	cult
$topic_{20}$	turkish	armenian	village	genocide	massacre
$topic_3$	bike	ride	tire	car	brake
$topic_{13}$	pitcher	pitch	score	playoff	game
$topic_2$	orbit	launch	shuttle	mission	satellite
$topic_7$	health	hospital	insurance	patient	doctor
$topic_{16}$	printer	font	tax	laser	print
$topic_{18}$	escrow	chip	clipper	phone	encryption
$topic_4$	image	shipping	format	color	manual
$topic_9$	terminal	remote	host	port	element
$topic_6$	motherboard	ide	scsi	connector	mhz
$topic_{15}$	font	widget	window	default	motif
$topic_1$	internet	newsgroup	faq	mailing	pub
$topic_{10}$	homosexuality	scripture	belief	atheist	faith

Table 28: 20NewsGroups - bayesian optimized RS con rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_{16}$	update	competitive	electricity	varying	receiver
$topic_2$	stationary	unitary	organizing	module	receiver
$topic_5$	ozone	perception	exploit	gaussian	extended
$topic_{20}$	analytical	vary	neighbor	prototype	creative
$topic_3$	check	segmentation	screen	varying	simulator
$topic_9$	sustainability	enzyme	concurrent	dissertation	direction
$topic_4$	minimum	wide	ownership	track	instruction
$topic_6$	indirect	perception	evaluate	traditional	mutual
$topic_{10}$	automation	flexibility	mutual	weather	creative
$topic_{11}$	competition	competitive	direction	gradient	analytical
$topic_{12}$	school	forecasting	outage	silicon	organize
$topic_{17}$	subject	construct	mutual	creative	ownership
$topic_1$	organizing	solar	perception	receiver	video
$topic_{15}$	prior	track	company	salinity	analytical
$topic_7$	governor	dominance	outage	deposition	window
$topic_{18}$	econometric	formalism	symbol	gaussian	tolerant
$topic_{14}$	competition	subject	partial	indirect	salinity
$topic_8$	inclusion	journal	direction	psychology	alpha
$topic_{13}$	journal	receiver	symbol	coding	intrusion
$topic_{19}$	econometric	category	variation	fourier	southern

Table 29: 20NewsGroups - over-RS

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_{14}$	page	water	monitor	condition	green
$topic_{18}$	sun	sin	image	widget	verse
$topic_{2}$	israeli	jewish	pitcher	guide	surrender
$topic_{17}$	engine	mission	student	launch	moon
$topic_{8}$	conference	research	terrorist	border	define
$topic_{19}$	film	camera	plane	mhz	tank
$topic_{13}$	game	season	fan	baseball	team
$topic_{3}$	motif	president	manager	package	industry
$topic_{11}$	magazine	police	insurance	car	cop
$topic_{7}$	launch	bike	orbit	ride	battery
$topic_{16}$	manual	window	auto	output	car
$topic_{15}$	software	computer	mail	email	interested
$topic_{4}$	monitor	encryption	hardware	instal	key
$topic_{5}$	patient	doctor	treatment	medical	pain
$topic_{10}$	chip	bus	bank	scsi	power
$topic_{12}$	surrender	encryption	homosexual	chemical	sex
$topic_{6}$	tax	batf	sin	firearm	argument
$topic_{1}$	kill	burn	fire	weapon	blood
$topic_{9}$	ball	cpu	bite	thread	quality
$topic_{20}$	bike	atheist	scsi	stuff	shit

Table 30: 20NewsGroups - over-RS con rmsprop

	$word_1$	$word_2$	$word_3$	$word_4$	$word_5$
$topic_1$	port	driver	serial	printer	meg
$topic_7$	format	file	image	server	version
$topic_{20}$	faith	sin	scripture	passage	biblical
$topic_{16}$	game	baseball	playoff	score	team
$topic_{18}$	ide	scsi	boot	controller	pitcher
$topic_5$	armenian	village	massacre	turkish	soldier
$topic_{17}$	gun	insurance	ticket	bike	ride
$topic_3$	monitor	card	slot	connector	video
$topic_9$	bike	car	engine	ride	tire
$topic_4$	encrypt	escrow	chip	clipper	key
$topic_6$	enforcement	administration	gun	crime	criminal
$topic_{15}$	address	mailing	internet	list	clipper
$topic_{14}$	genocide	turkish	soviet	muslim	armenian
$topic_2$	source	widget	terminal	printer	mouse
$topic_{10}$	driver	sale	controller	drive	scsi
$topic_{11}$	advance	driver	serial	controller	turkish
$topic_{13}$	connector	ground	pin	wheel	front
$topic_{19}$	vga	sale	offer	monitor	condition
$topic_8$	orbit	satellite	convert	image	earth
$topic_{12}$	shipping	condition	sale	offer	book

B Appendice figure

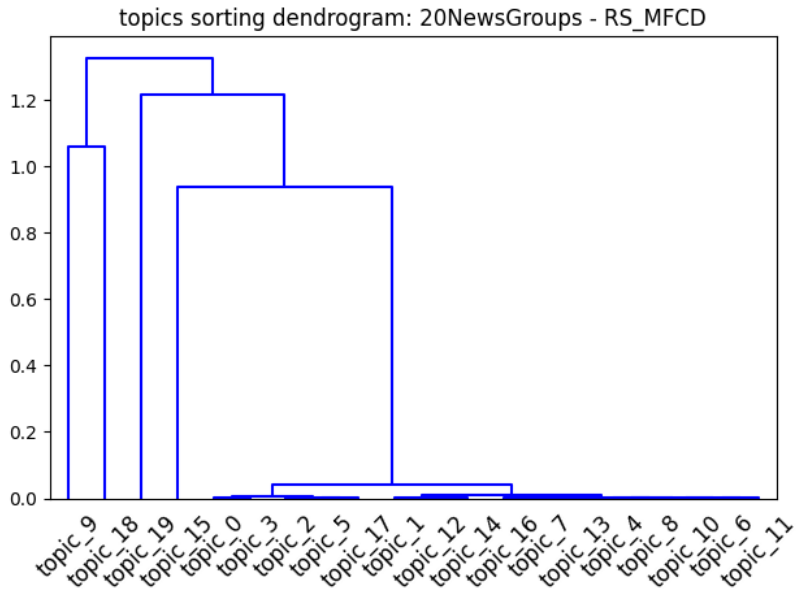


Figure 65: dendrogramma applicato ai topic di RS con MFCD su 20NewsGroups
Il dendrogramma è basato su Complete linkage e distanza del coseno.

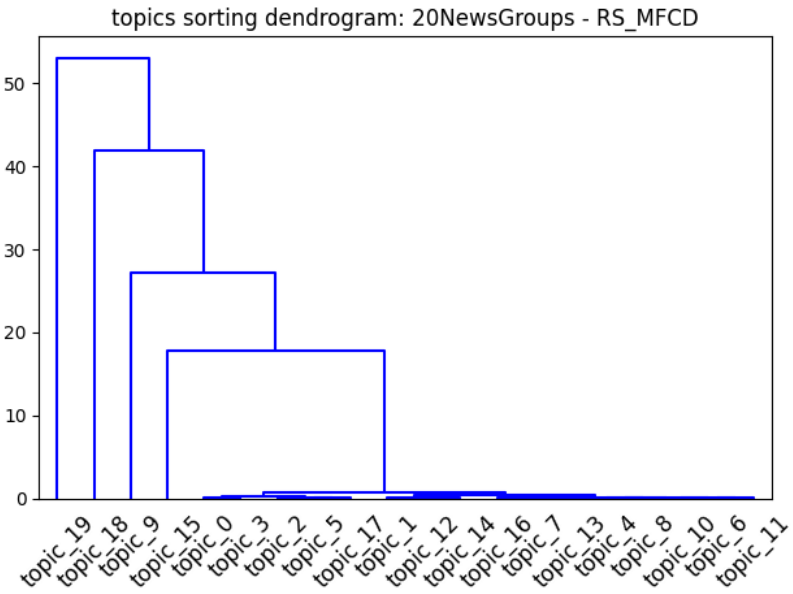


Figure 66: dendrogramma applicato ai topic di RS con MFCD su 20NewsGroups con Ward linkage

Il dendrogramma è basato su Ward linkage e distanza euclidea.

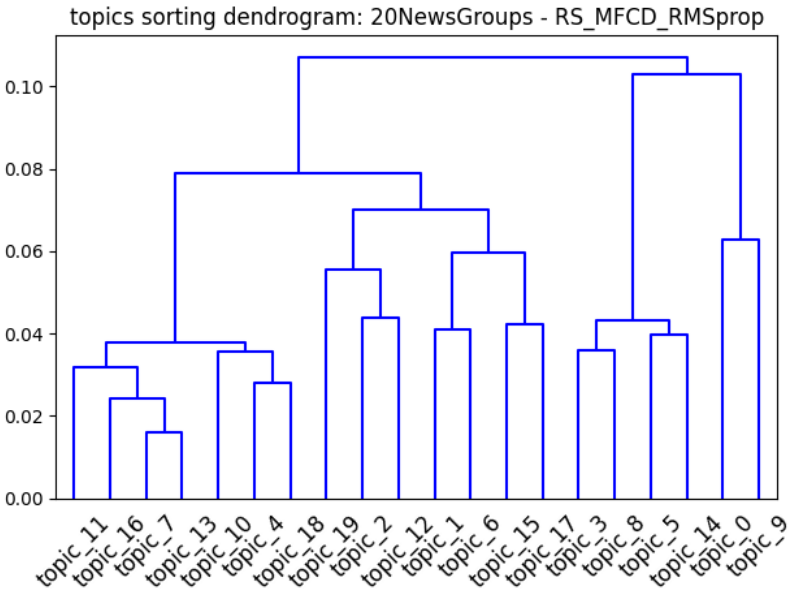


Figure 67: dendrogramma applicato ai topic di RS con MFCD e rmsprop su 20NewsGroups

Il dendrogramma è basato su Complete linkage e distanza del coseno.

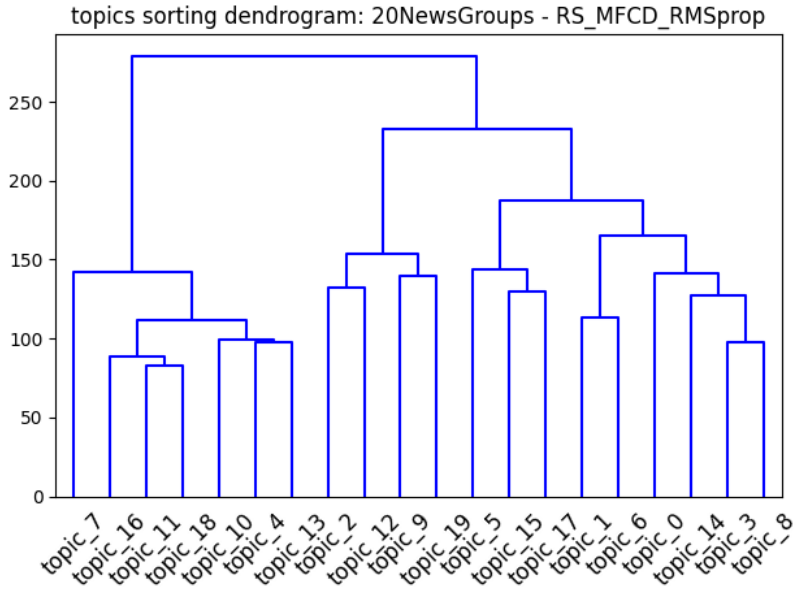


Figure 68: dendrogramma applicato ai topic di RS con MFCD e rmsprop su 20NewsGroups con Ward linkage

Il dendrogramma è basato su Ward linkage e distanza euclidea.

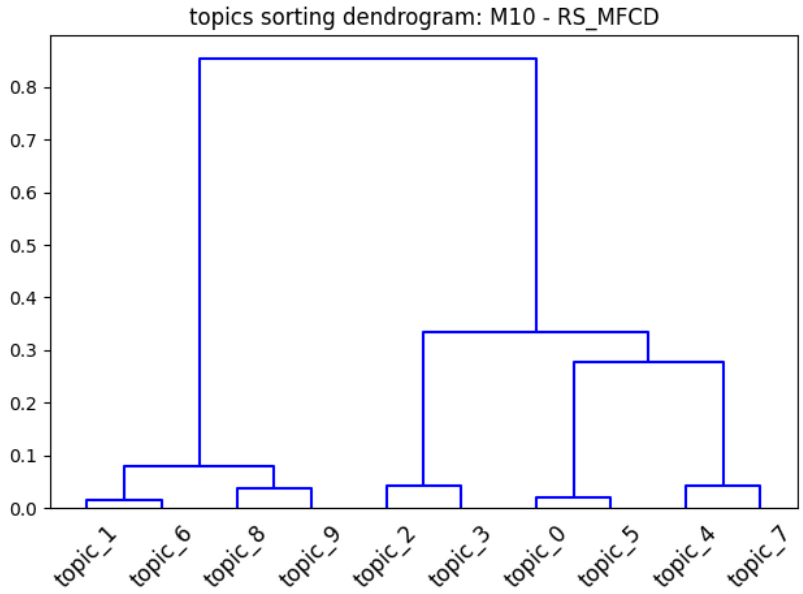


Figure 69: dendrogramma applicato ai topic di RS con MFCD su 20NewsGroups

Il dendrogramma è basato su Complete linkage e distanza del coseno.

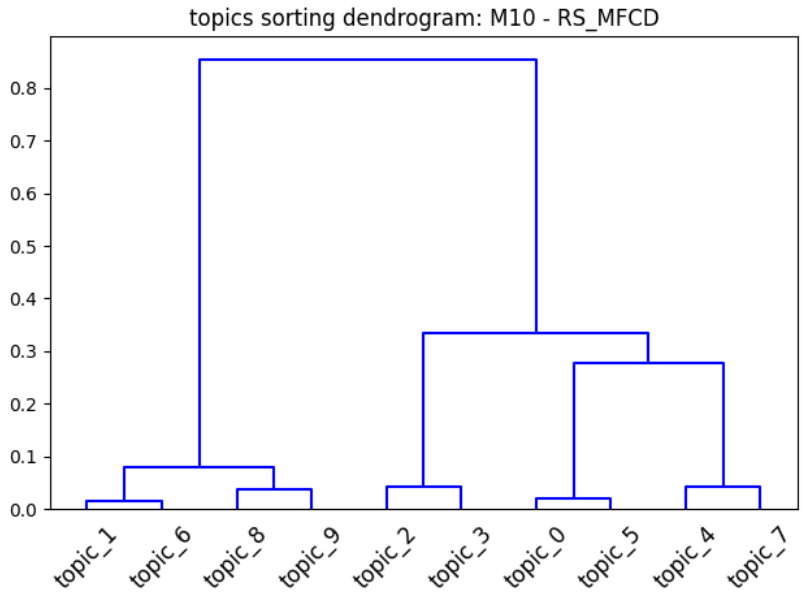


Figure 70: dendrogramma applicato ai topic di RS con MFCD su M10 con Ward
Il dendrogramma è basato su Ward linkage e distanza euclidea.

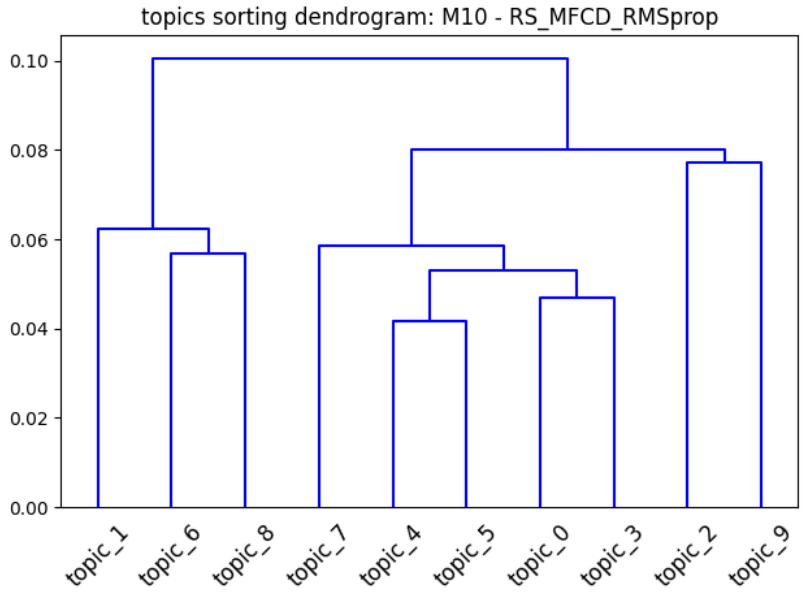


Figure 71: dendrogramma applicato ai topic di RS con MFCD e rmsprop su M10
Il dendrogramma è basato su Complete linkage e distanza del coseno.

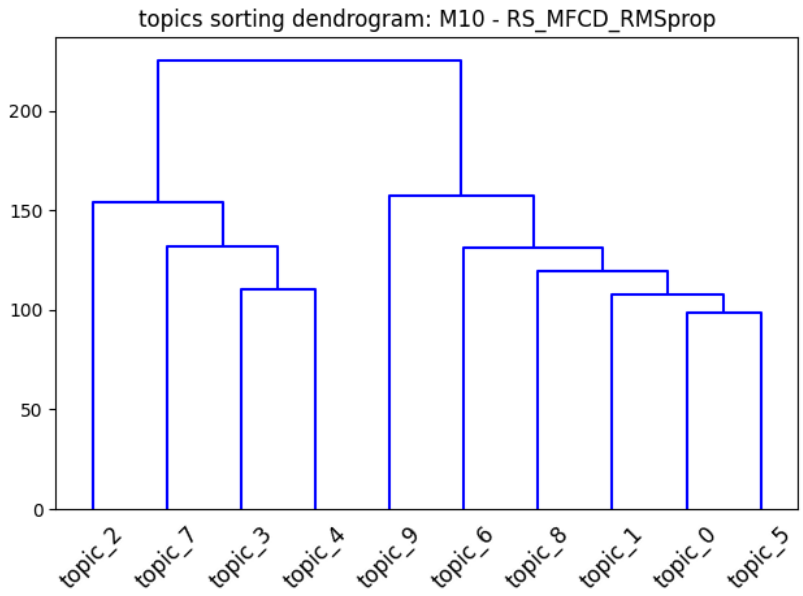


Figure 72: dendrogramma applicato ai topic di RS con MFCD e rmsprop su M10 con Ward linkage

Il dendrogramma è basato su Ward linkage e distanza euclidea.

C Appendice codici

Per una completa definizione dei modelli RSM e Over-RSM, si fa riferimento agli script RSM.py e oRSM.py presenti nella cartella models del repository di OCTIS: https://github.com/MIND-Lab/OCTIS

C.1 Utilizzo e validazione del RS in Octis

```
################## caricamento dataset di Octis

from octis.dataset.dataset import Dataset

dataset_M10 = Dataset()
dataset_M10.fetch_dataset("M10")

dataset_BBC = Dataset()
dataset_BBC.fetch_dataset("BBC_News")

dataset_20ng = Dataset()
```

```
dataset_20ng.fetch_dataset("20NewsGroup")
```

```
############ caricamento di un corpus arbitrario in Octis
#### il file corpus.txt deve contenere una riga per ogni documento

import os
import warnings
warnings.filterwarnings("ignore")
from octis.preprocessing.preprocessing import Preprocessing

### data path
root_dir =  os.getcwd()
root_dir = root_dir.replace('\\', '/')
data_path = root_dir + '/preprocessed_datasets'
raw_txt_path = data_path + '/20NewsGroup/corpus.txt'
raw_labels_path = data_path + '/20NewsGroup/labels.txt'



######## preprocessing mode
p = Preprocessing(
vocabulary=None, max_features=None, remove_punctuation=True,
lemmatize=False,  stopword_list='english', split=True,
min_chars=2, min_words_docs=1,
max_df=1.0, min_df=0.0, language='english',
remove_stopwords_spacy = True,
verbose=True)

######## final dataset
#each row of the txt file is seen as a single document
dataset = p.preprocess_dataset(
documents_path = raw_txt_path,
labels_path = raw_labels_path
)
```

```
# train RSM Model
from octis.models.RSM import RSM
```

```
model_rsm = RSM(num_topics=3, epochs=500,
lr=0.001, btsz=100, momentum=0.9, decay=0.001,
train_optimizer='momentum', cd_type='mfcd')

# Get model output
output_rsm = model_rsm.train_model(dataset)
```

```
# train over-RSM Model
from octis.models.oRSM import oRSM

model_orsm = oRSM(num_topics=5, epochs=1000, pretrain_epochs=500,
lr=0.001, btsz=100, momentum=0.1, decay=0.01,
train_optimizer='rmsprop', cd_type='mfcd')

# Get model output
output_orsm = model_orsm.train_model(dataset)
```

```
# train LDA
from octis.models.LDA import LDA

model_lda = LDA(num_topics=20, alpha=0.1,
passes = 2, iterations=1000, chunksize=500)

# Get model output
output_lda = model_lda.train_model(dataset)
```

```
####### output of a topic model

# The output of each octis model
# is a dictionary of at least two of this keys
print(list(output_rsm.keys()))

#['topic-word-matrix', 'topics',
#'topic-document-matrix', 'test-topic-document-matrix']

'''
```

```
topic-word-matrix: matrix of topic-word scores
(in RS is the interaction weight w_vh)


topic-document-matrix:
matrix of topic-word scores
(is a matrix of conditional probabilities of topics given
the docs)

test-topic-document-matrix:
same as topic-document-matrix but for test corpus

topics:
is the table of the topk words
(in Octis the integer topk can be 10 or less)
'''
```

```
  # train RSM Model with 2-step cd, penalty L1, momentum
  from octis.models.RSM import RSM

  model_rsm = RSM(num_topics=10, epochs=1000,
  lr=0.0001, btsz=100,

  momentum=0.9,

  decay=0.001, penalty_L1 = True,
  train_optimizer='momentum', cd_type='kcd', K=2)

  # Get model output
  output_rsm = model_rsm.train_model(dataset)
```

```
  # train RSM Model with persistent cd, penalty L2, no optimizer
  # and using the log(1+x) transformation of the dtm
  from octis.models.RSM import RSM

  model_rsm = RSM(
  num_topics=10, epochs=1000,
```

```
  lr=0.0001, btsz=100,

  decay=0.001, penalty_L1 = False,

  logdtm = True,
  train_optimizer='sgd', cd_type='pcd')

  # Get model output
  output_rsm = model_rsm.train_model(dataset)
```

```
  # get output dictionary of pre-trained RSM
  output_rsm = model_rsm.get_model_output(topk=10)
```

```
######### Octis validation metrics

from octis.evaluation_metrics.diversity_metrics import TopicDiversity
from octis.evaluation_metrics.coherence_metrics import Coherence
from octis.evaluation_metrics.classification_metrics import
AccuracyScore, F1Score, RecallScore

cnpmi = Coherence(texts = dataset.get_corpus(), topk=10, measure='c_npmi')
topic_diversity = TopicDiversity(topk=10)
acc = AccuracyScore(dataset)
f1 = F1Score(dataset)
rec = RecallScore(dataset)



print("RSM results:")
accuracy = acc.score(output_rsm)
print('Accuracy: '+str(accuracy))

F1s = f1.score(output_rsm)
print('F1 score: '+str(F1s))

recall = rec.score(output_rsm)
print('Recall: '+str(recall))
```

```
topic_diversity_score = topic_diversity.score(output_rsm)
print("Topic diversity: "+str(topic_diversity_score))

npmi_score = cnpmi.score(output_rsm)
print("Coherence: "+str(npmi_score))



print("LDA results:")
accuracy = acc.score(output_lda)
print('Accuracy: '+str(accuracy))

F1s = f1.score(output_lda)
print('F1 score: '+str(F1s))

recall = rec.score(output_lda)
print('Recall: '+str(recall))

topic_diversity_score = topic_diversity.score(output_lda)
print("Topic diversity: "+str(topic_diversity_score))

npmi_score = cnpmi.score(output_lda)
print("Coherence: "+str(npmi_score))
```

```
########## perplexity upper bound for RS model

## ppupbo of rsm

ppl =
model_rsm.trained_model.ppl_upbo(model_rsm.hyperparameters["dtm"])
print('RSM training set perplexity: ', ppl)

val_ppl = model_rsm.trained_model.ppl_upbo(model_rsm.hyperparameters["val_dtm"
                                        ])
print('RSM test set perplexity: ', val_ppl)
```

```
########## perplexity upper bound for LDA model
```

```
# get validation texts from octis Dataset (handles when no partition exists)
parts = dataset_20ng.get_partitioned_corpus(use_validation=True)
train_texts, val_texts, test_texts = parts

# build gensim bow for validation and compute log perplexity
dictionary = model_lda.trained_model.id2word
bow_val = [dictionary.doc2bow(doc) for doc in val_texts]

# pass bow_val and total_docs=len(bow_val)
lp_val = model_lda.trained_model.log_perplexity(bow_val, total_docs=len(
                                        bow_val))
print("validation log perplexity (upper bound):", -lp_val)
print("validation perplexity (upper bound):", np.exp(-lp_val))
```

C.2 Funzioni per descrivere i topic

```
### function to describe a topic model topics in latex

def describe_topics(tm_output, topk=5, coltop=True):
topics = tm_output['topics']
topics = topics[0:topk , :]
df = pd.DataFrame(topics)
df.index = [f'topic_{{{i+1}}}' for i in range(len(topics))]
df.columns = [f'word_{{{i+1}}}' for i in range(topk)]
if coltop:
print(df.transpose().to_latex())
else:
print(df.to_latex())
```

```
####### ricostruzione delle descrizioni dei topic
###del modello RSM salvato come file .pkl

topk=10
rsm = load_pkl(f'esperimenti_finali/{expname}.pkl')
```

```
w_vh, w_v, w_h = rsm.trained_model.W
topics = rsm.trained_model.topic_words(topk)

out = {'topics' : topics, 'topic-word-matrix': w_vh.T}



#### obtain latex table of topic model

print(describe_topics(tm_output=out, topk=topk))
```

```
from scipy.cluster.hierarchy import dendrogram, linkage, leaves_list
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

def describe_topics_sorted(tm_output, id2word, topk=5, coltop=True):

'''
returns the sorted latex table of per topic topk ranked words
'''



##numpy 2d array of words, of dim ntopic x topk
topics = tm_output['topics']

###numpy 2d array of numbers, of dim nword x ntopic
top_doc = tm_output['topic-word-matrix']

## get id of words in topics
all_words = [id2word[i] for i in range(len(id2word.keys()))]
main_words = np.unique(np.array([w for top in topics for w in top]))
ids = [np.where(np.array(all_words) == w)[0][0] for w in main_words]

## clustering of topics
```

```
data = top_doc.T
Z = linkage(data, method='complete', metric='cosine')
## another good linkage is method='ward' ,
## but that requires metric='euclidean'

# 2. Get the sorted indices (the leaf order of the dendrogram)
sorted_indices = leaves_list(Z)

## show topics sorted
df = pd.DataFrame(topics)
df.index = ["$\\text{topic}" + f"_{{{i+1}}}$" for i in range(len(topics))]
df.columns = ["$\\text{word}" + f"_{{{i+1}}}$" for i in range(topk)]
df = df.iloc[sorted_indices]
if coltop:
return df.transpose().to_latex()
else:
return df.to_latex()




def dendro_topics(tm_output, title = '', color='blue', filename=None):

'''
makes the plot of the dendrogram used to sort the topics table
'''

###numpy 2d array of numbers, of dim nword x ntopic
top_doc = tm_output['topic-word-matrix]

# clustering of topics
data = top_doc.T
Z = linkage(data, method='complete', metric='cosine')

# 1. Create the topic labels: topic_0, topic_1, etc.
topic_labels = [f'topic_{i}' for i in range(len(data))]
```

```
# 2. Plot dendrogram
# color_threshold=0 and above_threshold_color='black' removes color
                                        differentiation
dendrogram(
Z,
labels=topic_labels,
color_threshold=0,
above_threshold_color=color
)

# 3. Rotate x-axis labels by 45 degrees
plt.xticks(rotation=45)

# 4. Set a title
plt.title(title)

plt.tight_layout() # Ensures labels aren't cut off
plt.show()

if filename is not None:
plt.savefig(f'{filename}.png')
```

C.3 Esperimenti sul modello RS

```
########## functions for monitoring RS training over epochs:

from octis.evaluation_metrics.diversity_metrics import TopicDiversity
from octis.evaluation_metrics.coherence_metrics import Coherence
from octis.evaluation_metrics.classification_metrics import AccuracyScore,
                                        F1Score, RecallScore
from tqdm import tqdm
import time
import numpy as np
import pandas as pd
import json
```

```
import pickle


## useful generic functions to handle json and pkl


def save_json(data, filename):
'''example usage: save_json(mydata, 'data.json')'''
with open(filename, 'w') as f:
json.dump(data, f, indent=4)


def load_json(filename):
with open(filename, 'r') as f:
data = json.load(f)
return data


def save_pkl(data, filename):
'''example usage: save_pkl(mydata, 'data.pkl')'''
with open(filename, 'wb') as f:
pickle.dump(data, f)

def load_pkl(filename):
with open(filename, 'rb') as f:
data = pickle.load(f)
return data




def monit_and_train_rsm(expname, rsm, dataset, eval_each=10):

'''
expname: string, the name of the experiment to save
```

```
rsm: initialized not trained empty model with the hyperparameters to use
between the hyperparameters there is the number of epochs, that is extracted
                                        to
eval_each: length of the epochs interval for the metrics measurements
'''


cnpmi = Coherence(texts = dataset.get_corpus(), topk=10, measure='c_npmi')
topic_diversity = TopicDiversity(topk=10)
acc = AccuracyScore(dataset)

train_epochs = rsm.hyperparameters['epochs']
C = int(train_epochs/eval_each)
RSM_results = {key: np.zeros(C+1).tolist() for key in ['accuracy', '
                                        topic_diversity', 'coherence',
  'train_ppl', 'val_ppl', 'avg_exec_time', 'epoch_id']}


rsm.initialize_model_structure(rsm.hyperparameters, dataset)

rsm_struct_hyper =
{key: rsm.hyperparameters[key] for key in ['softstart', 'epochs','num_topics',
                                        'dtm', 'val_dtm',
  'monitor_ppl', 'monitor_time', 'logdtm']}

rsm.trained_model.set_structure_from_dtm(**rsm_struct_hyper)

rsm_train_hyper =
{key: rsm.hyperparameters[key] for key in ['epochs', 'btsz',
  'lr', 'momentum', 'K', 'decay', 'penalty_L1', 'penalty_local',
  'train_optimizer', 'cd_type',
  'rms_decay', 'adam_decay1', 'adam_decay2',
  'increase_speed']}

rsm.trained_model.set_train_hyper(**rsm_train_hyper)
```

```
output_rsm = rsm.get_model_output()
RSM_results['accuracy'][0] = acc.score(output_rsm)
RSM_results['topic_diversity'][0] = topic_diversity.score(output_rsm)
RSM_results['coherence'][0] = cnpmi.score(output_rsm)
RSM_results['train_ppl'][0] = rsm.trained_model.ppl_upbo(rsm.hyperparameters['
                                        dtm'])
RSM_results['val_ppl'][0] = rsm.trained_model.ppl_upbo(rsm.hyperparameters['
                                        val_dtm'])

for c in tqdm(range(C)):
start = time.time()
for t in range(eval_each):
rsm.trained_model.train_epoch()
end = time.time()

output_rsm = rsm.get_model_output()
d = c+1
RSM_results['accuracy'][d] = acc.score(output_rsm)
RSM_results['topic_diversity'][d] = topic_diversity.score(output_rsm)
RSM_results['coherence'][d] = cnpmi.score(output_rsm)
RSM_results['train_ppl'][d] = rsm.trained_model.ppl_upbo(rsm.hyperparameters['
                                        dtm'])
RSM_results['val_ppl'][d] = rsm.trained_model.ppl_upbo(rsm.hyperparameters['
                                        val_dtm'])
RSM_results['avg_exec_time'][d] = (end - start)/ eval_each
RSM_results['epoch_id'][d] = d*eval_each

save_my_exp(expname, RSM_results, rsm)

return RSM_results, rsm



def save_my_exp(expname, RSM_results, model_rsm):

'''
```

```
expname: name of the experiment
RSM_results: output[0] of the function monit_and_train_rsm
model_rsm: output[1] of the function monit_and_train_rsm

before starting to use this function, You have to initialize
the proper json file path

for example:

## how to destroy your data - initialize your empty json
## save_json({}, 'esperimenti_finali/lastexps.json')


output:

this function saves the model performance in the json
and the model object in a pickle file named as 'expname'.pkl

'''

myexps = load_json('esperimenti_finali/lastexps.json')
myexps[expname] = RSM_results
save_json(myexps, 'esperimenti_finali/lastexps.json')
filename = f'esperimenti_finali/{expname}.pkl'
save_pkl(model_rsm, filename)
```

```
################## Codice per l'esperimento 1:
#Contrastive Divergence tecniques

### compare different cd types

## mfcd
```

```
#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='sgd')
RSM_results0, rsm0 = monit_and_train_rsm('rsm_exp0_20ng_mfcd_sgd_1000', rsm,
                                        dataset_20ng, eval_each=20)

#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='sgd')
RSM_results1, rsm1 = monit_and_train_rsm('rsm_exp0_M10_mfcd_sgd_1000', rsm,
                                        dataset_M10, eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='sgd')
RSM_results2, rsm2 = monit_and_train_rsm('rsm_exp0_BBC_mfcd_sgd_1000', rsm,
                                        dataset_BBC, eval_each=20)


## 1cd

#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                        train_optimizer='sgd', K=1)
RSM_results3, rsm3 = monit_and_train_rsm('rsm_exp0_20ng_1cd_sgd_1000', rsm,
                                        dataset_20ng, eval_each=20)

#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                        train_optimizer='sgd', K=1)
RSM_results4, rsm4 = monit_and_train_rsm('rsm_exp0_M10_1cd_sgd_1000', rsm,
                                        dataset_M10, eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                        train_optimizer='sgd', K=1)
```

```
RSM_results5, rsm5 = monit_and_train_rsm('rsm_exp0_BBC_1cd_sgd_1000', rsm,
                                         dataset_BBC, eval_each=20)


## pcd

#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='pcd',
                                         train_optimizer='sgd', K=1)
RSM_results6, rsm6 = monit_and_train_rsm('rsm_exp0_20ng_pcd_sgd_1000', rsm,
                                         dataset_20ng, eval_each=20)

#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='pcd',
                                         train_optimizer='sgd', K=1)
RSM_results7, rsm7 = monit_and_train_rsm('rsm_exp0_M10_pcd_sgd_1000', rsm,
                                         dataset_M10, eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='pcd',
                                         train_optimizer='sgd', K=1)
RSM_results8, rsm8 = monit_and_train_rsm('rsm_exp0_BBC_pcd_sgd_1000', rsm,
                                         dataset_BBC, eval_each=20)


## 2cd

#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                         train_optimizer='sgd', K=2)
RSM_results9, rsm9 = monit_and_train_rsm('rsm_exp0_20ng_2cd_sgd_1000', rsm,
                                         dataset_20ng, eval_each=20)

#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                         train_optimizer='sgd', K=2)
```

```
RSM_results10, rsm10 = monit_and_train_rsm('rsm_exp0_M10_2cd_sgd_1000', rsm,
                                           dataset_M10, eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                           train_optimizer='sgd', K=2)
RSM_results11, rsm11 = monit_and_train_rsm('rsm_exp0_BBC_2cd_sgd_1000', rsm,
                                           dataset_BBC, eval_each=20)
```

```
################## Codice per l'esperimento 2: penalized RS

## mfcd sgd penalization L2-0.1

#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.001, cd_type='mfcd',
                                           train_optimizer='sgd', decay = 0.001)
RSM_results0, rsm0 = monit_and_train_rsm('
                                           rsm_exp51_20ng_mfcd_sgd_pen001_L2_1000'
                                           , rsm, dataset_20ng, eval_each=20)

#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.001, cd_type='mfcd',
                                           train_optimizer='sgd', decay = 0.001)
RSM_results1, rsm1 = monit_and_train_rsm('
                                           rsm_exp51_M10_mfcd_sgd_pen001_L2_1000',
                                            rsm, dataset_M10, eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.001, cd_type='mfcd',
                                           train_optimizer='sgd', decay = 0.001)
RSM_results2, rsm2 = monit_and_train_rsm('
                                           rsm_exp51_BBC_mfcd_sgd_pen001_L2_1000',
                                            rsm, dataset_BBC, eval_each=20)
```

```
## mfcd sgd penalization L1-0.1

#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.001, cd_type='mfcd',
 train_optimizer='sgd', decay = 0.001, penalty_L1=True)
RSM_results0, rsm0 =
 monit_and_train_rsm('rsm_exp51_20ng_mfcd_sgd_pen001_L1_1000', rsm,
                                        dataset_20ng, eval_each=20)

#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.001,
 cd_type='mfcd', train_optimizer='sgd', decay = 0.001, penalty_L1=True)
RSM_results1, rsm1 =
 monit_and_train_rsm('rsm_exp51_M10_mfcd_sgd_pen001_L1_1000', rsm, dataset_M10
                                        , eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.001,
cd_type='mfcd', train_optimizer='sgd', decay = 0.001, penalty_L1=True)
RSM_results2, rsm2 =
 monit_and_train_rsm('rsm_exp51_BBC_mfcd_sgd_pen001_L1_1000', rsm, dataset_BBC
                                        , eval_each=20)
```

```
################## Codice per l'esperimento 3: RS con MFCD e
## ottimizzazione di gradient descent, senza penalizzazione

### compare different optimizers

## adagrad

#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                       train_optimizer='adagrad')
RSM_results012, rsm12 = monit_and_train_rsm('rsm_exp1_20ng_mfcd_adagrad_1000',
                                        rsm, dataset_20ng, eval_each=20)
```

```
#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='adagrad')
RSM_results13, rsm13 = monit_and_train_rsm('rsm_exp1_M10_mfcd_adagrad_1000',
                                        rsm, dataset_M10, eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='adagrad')
RSM_results14, rsm14 = monit_and_train_rsm('rsm_exp1_BBC_mfcd_adagrad_1000',
                                        rsm, dataset_BBC, eval_each=20)


## momentum

#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='momentum')
RSM_results15, rsm15 = monit_and_train_rsm('rsm_exp1_20ng_mfcd_momentum_1000',
                                         rsm, dataset_20ng, eval_each=20)

#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='momentum')
RSM_results16, rsm16 = monit_and_train_rsm('rsm_exp1_M10_mfcd_momentum_1000',
                                        rsm, dataset_M10, eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='momentum')
RSM_results17, rsm17 = monit_and_train_rsm('rsm_exp1_BBC_mfcd_momentum_1000',
                                        rsm, dataset_BBC, eval_each=20)


## rmsprop
```

```
#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='rmsprop')
RSM_results18, rsm18 = monit_and_train_rsm('rsm_exp1_20ng_mfcd_rmsprop_1000',
                                        rsm, dataset_20ng, eval_each=20)

#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='rmsprop')
RSM_results19, rsm19 = monit_and_train_rsm('rsm_exp1_M10_mfcd_rmsprop_1000',
                                        rsm, dataset_M10, eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='rmsprop')
RSM_results20, rsm20 = monit_and_train_rsm('rsm_exp1_BBC_mfcd_rmsprop_1000',
                                        rsm, dataset_BBC, eval_each=20)



## adam

#20ng
rsm = RSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='adam')
RSM_results21, rsm21 = monit_and_train_rsm('rsm_exp1_20ng_mfcd_adam_1000', rsm
                                        , dataset_20ng, eval_each=20)

#m10
rsm = RSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='adam')
RSM_results22, rsm22 = monit_and_train_rsm('rsm_exp1_M10_mfcd_adam_1000', rsm,
                                         dataset_M10, eval_each=20)

#bbc
rsm = RSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
```

```
                                        train_optimizer='adam')
RSM_results23, rsm23 = monit_and_train_rsm('rsm_exp1_BBC_mfcd_adam_1000', rsm,
                                         dataset_BBC, eval_each=20)
```

```
################## funzioni per l'esperimento 4: ottimizzazione bayesiana del
                                        modello RS



from octis.optimization.optimizer import Optimizer
from skopt.space.space import Real, Integer
from octis.evaluation_metrics.diversity_metrics import TopicDiversity
from octis.evaluation_metrics.coherence_metrics import Coherence
from octis.evaluation_metrics.classification_metrics import AccuracyScore,
                                        F1Score, RecallScore



def bo_rsm(expname, rsm, dataset, search_space, eval_metric,
epochs_bo=20, epochs_train=100, number_of_call=15, model_runs=1, eval_each=10,
surrogate_model='GP', acq_func='LCB'):

mydict = load_json('esperimenti_finali/lastboexps.json')


rsm.hyperparameters['epochs'] = epochs_bo
optimizer=Optimizer()
optResult=optimizer.optimize(rsm, dataset,
eval_metric, search_space, save_path="esperimenti_finali/bo_exps", # path to
                                        store the results
number_of_call=number_of_call, # number of optimization iterations
model_runs=model_runs, surrogate_model=surrogate_model, acq_func=acq_func)

optResult.save_to_csv(f"esperimenti_finali\{expname}.csv")

optres_dict = optResult.x_iters.copy()
```

```
optres_dict['y_objective'] = optResult.func_vals

idbest = np.argmax(optResult.func_vals)
best_params = {k : v[idbest] for k, v in optResult.x_iters.items()}
best_y = optResult.func_vals[idbest]

mydict[expname] = {'best_y': best_y,  'idbest':int(idbest), 'best_params' :
                                        best_params, 'iterations': optres_dict}
save_json(mydict, 'esperimenti_finali/lastboexps.json')

rsm.hyperparameters['epochs'] = epochs_train
hypers = rsm.hyperparameters.copy()
for k, v in best_params.items():
hypers[k] = v

del hypers['val_dtm']
del hypers['dtm']

print(f'best metric: {best_y}')
print('Train the model with the following hyperparams')
print(best_params)

newrsm = RSM(**hypers)
RSM_results, rsm = monit_and_train_rsm(expname, newrsm, dataset, eval_each=
                                        eval_each)

return RSM_results, rsm, mydict[expname].copy()
```

```
##### codice per l'esperimento 4:
####ottimizzazione bayesiana del modello RS


## 20NewsGroups
rsm = RSM(num_topics=20, btsz=20, lr=0.0001, cd_type='mfcd', train_optimizer='
```

```
                                        rmsprop')
search_space = {"lr": Real(low=0.00001, high=0.005), "btsz": Integer(low=1,
                                        high=300), 'decay': Real(low=0.0, high=
                                        0.2),
  'rms_decay' : Real(low=0.0, high=0.99)}
eval_topdiv10 = TopicDiversity(topk=10)
RSM_results0, rsm0, resdict0 = bo_rsm('bo_rsm_mfcd_rmsprop_20ng_1000', rsm,
                                        dataset_M10, search_space,
eval_metric=eval_topdiv10, epochs_bo=20, epochs_train=1000,
number_of_call=15, model_runs=1, eval_each=20,
surrogate_model='GP', acq_func='LCB')

## M10
rsm = RSM(num_topics=10, btsz=20, lr=0.0001, cd_type='mfcd', train_optimizer='
                                        rmsprop')
search_space = {"lr": Real(low=0.00001, high=0.005), "btsz": Integer(low=1,
                                        high=300), 'decay': Real(low=0.0, high=
                                        0.2),
  'rms_decay' : Real(low=0.0, high=0.99)}
eval_topdiv10 = TopicDiversity(topk=10)
RSM_results, rsm, resdict = bo_rsm('bo_rsm_mfcd_rmsprop_M10_1000', rsm,
                                        dataset_M10, search_space,
eval_metric=eval_topdiv10, epochs_bo=20, epochs_train=1000,
number_of_call=15, model_runs=1, eval_each=20,
surrogate_model='GP', acq_func='LCB')


## BBC News
rsm = RSM(num_topics=5, btsz=20, lr=0.0001, cd_type='mfcd', train_optimizer='
                                        rmsprop')
search_space = {"lr": Real(low=0.00001, high=0.005), "btsz": Integer(low=1,
                                        high=300), 'decay': Real(low=0.0, high=
                                        0.2),
  'rms_decay' : Real(low=0.0, high=0.99)}
eval_topdiv10 = TopicDiversity(topk=10)
RSM_results0, rsm0, resdict0 = bo_rsm('bo_rsm_mfcd_rmsprop_BBC_1000', rsm,
```

```
                                        dataset_M10, search_space,
eval_metric=eval_topdiv10, epochs_bo=120, epochs_train=1000,
number_of_call=15, model_runs=1, eval_each=20,
surrogate_model='GP', acq_func='LCB')
```

C.4 codice per gli esperimenti sul modello over-RS

```
############ funzioni per gli esperimenti sul modello Over-RS



# from FRutils import save_json, load_json, save_pkl, load_pkl
# from octis.evaluation_metrics.diversity_metrics import TopicDiversity
# from octis.evaluation_metrics.coherence_metrics import Coherence
# from octis.evaluation_metrics.classification_metrics import AccuracyScore,
                                        F1Score, RecallScore
from tqdm import tqdm
from octis.models.oRSM import oRSM
import time
# import numpy as np
# import pandas as pd



def monit_and_train_ors(expname, ors, dataset, eval_each=10):

cnpmi = Coherence(texts = dataset.get_corpus(), topk=10, measure='c_npmi')
topic_diversity = TopicDiversity(topk=10)

train_epochs = ors.hyperparameters['epochs']
C = int(train_epochs/eval_each)
ORS_results = {key: np.zeros(C+1).tolist() for key in ['accuracy', '
                                        topic_diversity', 'coherence',
  'train_ppl', 'val_ppl', 'avg_exec_time', 'epoch_id']}
```

```
ors.initialize_model_structure(ors.hyperparameters, dataset)

ors_struct_hyper = {key: ors.hyperparameters[key] for key in ['softstart', '
                                        epochs','num_topics', 'dtm', 'val_dtm',
  'monitor_ppl', 'monitor_time', 'logdtm']}
ors.trained_model.set_structure_from_dtm(**ors_struct_hyper)
ors_train_hyper = {key: ors.hyperparameters[key] for key in ['epochs', 'btsz',
  'lr', 'momentum', 'K', 'decay', 'penalty_L1', 'penalty_local',
  'train_optimizer', 'cd_type',
  'rms_decay', 'adam_decay1', 'adam_decay2',
  'increase_speed', 'M', 'pretrain_epochs', 'epsilon']}
ors.trained_model.set_train_hyper(**ors_train_hyper)

output_ors = ors.get_model_output()
try:
acc = AccuracyScore(dataset)
ORS_results['accuracy'][0] = acc.score(output_ors)
except:
ORS_results['accuracy'][0] = None
ORS_results['topic_diversity'][0] = topic_diversity.score(output_ors)
ORS_results['coherence'][0] = cnpmi.score(output_ors)
ORS_results['train_ppl'][0] = ors.trained_model.ppl_upbo(ors.hyperparameters['
                                        dtm'])
ORS_results['val_ppl'][0] = ors.trained_model.ppl_upbo(ors.hyperparameters['
                                        val_dtm'])

for c in tqdm(range(C)):
start = time.time()
for t in range(eval_each):
ors.trained_model.train_epoch()
end = time.time()

output_ors = ors.get_model_output()
d = c+1
try:
acc = AccuracyScore(dataset)
```

```
ORS_results['accuracy'][d] = acc.score(output_ors)
except:
ORS_results['accuracy'][d] = None
ORS_results['topic_diversity'][d] = topic_diversity.score(output_ors)
ORS_results['coherence'][d] = cnpmi.score(output_ors)
ORS_results['train_ppl'][d] = ors.trained_model.ppl_upbo(ors.hyperparameters['
                                        dtm'])
ORS_results['val_ppl'][d] = ors.trained_model.ppl_upbo(ors.hyperparameters['
                                        val_dtm'])
ORS_results['avg_exec_time'][d] = (end - start)/ eval_each
ORS_results['epoch_id'][d] = d*eval_each

save_my_exp0(expname, ORS_results, ors)

return ORS_results, ors


def save_my_exp0(expname, ORS_results, model_ors):
myexps = load_json('esperimenti_finali/lastexps0.json')
myexps[expname] = ORS_results
save_json(myexps, 'esperimenti_finali/lastexps0.json')
filename = f'esperimenti_finali/{expname}.pkl'
save_pkl(model_ors, filename)
```

```
########### funzioni per salvare esperimenti di BO per oRSM

from octis.optimization.optimizer import Optimizer
from skopt.space.space import Real, Integer
from octis.evaluation_metrics.diversity_metrics import TopicDiversity
from octis.evaluation_metrics.coherence_metrics import Coherence
from octis.evaluation_metrics.classification_metrics import AccuracyScore,
                                        F1Score, RecallScore
```

```
def bo_ors(expname, ors, dataset, search_space, eval_metric,
epochs_bo=20, epochs_train=100, number_of_call=15, model_runs=1, eval_each=10,
surrogate_model='GP', acq_func='LCB'):

mydict = load_json('esperimenti_finali/lastboexps0.json')



ors.hyperparameters['epochs'] = epochs_bo
optimizer=Optimizer()
optResult=optimizer.optimize(ors, dataset,
eval_metric, search_space, save_path="esperimenti_finali/bo_exps0", # path to
                                        store the results
number_of_call=number_of_call, # number of optimization iterations
model_runs=model_runs, surrogate_model=surrogate_model, acq_func=acq_func)

optResult.save_to_csv(f"esperimenti_finali\{expname}.csv")

optres_dict = optResult.x_iters.copy()
optres_dict['y_objective'] = optResult.func_vals

idbest = np.argmax(optResult.func_vals)
best_params = {k : v[idbest] for k, v in optResult.x_iters.items()}
best_y = optResult.func_vals[idbest]

mydict[expname] = {'best_y': best_y,  'idbest':int(idbest), 'best_params' :
                                         best_params, 'iterations': optres_dict}
save_json(mydict, 'esperimenti_finali/lastboexps0.json')

ors.hyperparameters['epochs'] = epochs_train
hypers = ors.hyperparameters.copy()
for k, v in best_params.items():
hypers[k] = v

del hypers['val_dtm']
del hypers['dtm']
```

```
print(f'best metric: {best_y}')
print('Train the model with the following hyperparams')
print(best_params)

newors = oRSM(**hypers)
ORS_results, ors = monit_and_train_ors(expname, newors, dataset, eval_each=
                                       eval_each)

return ORS_results, ors, mydict[expname].copy()
```

```
#### codice per l'esperimento 5: Contrastive Divergence per Over-RSM

### compare different cd types

## mfcd

#20ng
ors = oRSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                       train_optimizer='sgd', M=50,
                                       pretrain_epochs=500)
ORS_results0, ors0 = monit_and_train_ors('ors_exp0_20ng_mfcd_sgd_1000', ors,
                                       dataset_20ng, eval_each=20)

#m10
ors = oRSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                       train_optimizer='sgd', M=50,
                                       pretrain_epochs=500)
ORS_results1, ors1 = monit_and_train_ors('ors_exp0_M10_mfcd_sgd_1000', ors,
                                       dataset_M10, eval_each=20)

#bbc
ors = oRSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                       train_optimizer='sgd', M=50,
                                       pretrain_epochs=500)
```

```
ORS_results2, ors2 = monit_and_train_ors('ors_exp0_BBC_mfcd_sgd_1000', ors,
                                         dataset_BBC, eval_each=20)


## 1cd

#20ng
ors = oRSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                         train_optimizer='sgd', K=1, M=50,
                                         pretrain_epochs=500)
ORS_results3, ors3 = monit_and_train_ors('ors_exp0_20ng_1cd_sgd_1000', ors,
                                         dataset_20ng, eval_each=20)

#m10
ors = oRSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                         train_optimizer='sgd', K=1, M=50,
                                         pretrain_epochs=500)
ORS_results4, ors4 = monit_and_train_ors('ors_exp0_M10_1cd_sgd_1000', ors,
                                         dataset_M10, eval_each=20)

#bbc
ors = oRSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                         train_optimizer='sgd', K=1, M=50,
                                         pretrain_epochs=500)
ORS_results5, ors5 = monit_and_train_ors('ors_exp0_BBC_1cd_sgd_1000', ors,
                                         dataset_BBC, eval_each=20)


## pcd

#20ng
ors = oRSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='pcd',
                                         train_optimizer='sgd', K=1, M=50,
                                         pretrain_epochs=500)
ORS_results6, ors6 = monit_and_train_ors('ors_exp0_20ng_pcd_sgd_1000', ors,
                                         dataset_20ng, eval_each=20)
```

```
#m10
ors = oRSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='pcd',
                                        train_optimizer='sgd', K=1, M=50,
                                        pretrain_epochs=500)
ORS_results7, ors7 = monit_and_train_ors('ors_exp0_M10_pcd_sgd_1000', ors,
                                        dataset_M10, eval_each=20)

#bbc
ors = oRSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='pcd',
                                        train_optimizer='sgd', K=1, M=50,
                                        pretrain_epochs=500)
ORS_results8, ors8 = monit_and_train_ors('ors_exp0_BBC_pcd_sgd_1000', ors,
                                        dataset_BBC, eval_each=20)


## 2cd

#20ng
ors = oRSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                        train_optimizer='sgd', K=2, M=50,
                                        pretrain_epochs=500)
ORS_results9, ors9 = monit_and_train_ors('ors_exp0_20ng_2cd_sgd_1000', ors,
                                        dataset_20ng, eval_each=20)

#m10
ors = oRSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                        train_optimizer='sgd', K=2, M=50,
                                        pretrain_epochs=500)
ORS_results10, ors10 = monit_and_train_ors('ors_exp0_M10_2cd_sgd_1000', ors,
                                        dataset_M10, eval_each=20)

#bbc
ors = oRSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='kcd',
                                        train_optimizer='sgd', K=2, M=50,
                                        pretrain_epochs=500)
```

```
ORS_results11, ors11 = monit_and_train_ors('ors_exp0_BBC_2cd_sgd_1000', ors,
                                           dataset_BBC, eval_each=20)
```

```
############## codice per l'esperimento 6:
###### ottimizzatori di gradient descent per
###### Over-RS con MFCD non penalizzato

### compare different optimizers

## adagrad

#20ng
ors = oRSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                           train_optimizer='adagrad', M=50,
                                           pretrain_epochs=500)
ORS_results012, ors12 = monit_and_train_ors('ors_exp11_20ng_mfcd_adagrad_1000'
                                           , ors, dataset_20ng, eval_each=20)

#m10
ors = oRSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                           train_optimizer='adagrad', M=50,
                                           pretrain_epochs=500)
ORS_results13, ors13 = monit_and_train_ors('ors_exp11_M10_mfcd_adagrad_1000',
                                           ors, dataset_M10, eval_each=20)

#bbc
ors = oRSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                           train_optimizer='adagrad')
ORS_results14, ors14 = monit_and_train_ors('ors_exp11_BBC_mfcd_adagrad_1000',
                                           ors, dataset_BBC, eval_each=20)


## momentum

#20ng
ors = oRSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
```

```
                                        train_optimizer='momentum', M=50,
                                        pretrain_epochs=500)
ORS_results15, ors15 = monit_and_train_ors('ors_exp11_20ng_mfcd_momentum_1000'
                                        , ors, dataset_20ng, eval_each=20)

#m10
ors = oRSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='momentum', M=50,
                                        pretrain_epochs=500)
ORS_results16, ors16 = monit_and_train_ors('ors_exp11_M10_mfcd_momentum_1000',
                                         ors, dataset_M10, eval_each=20)

#bbc
ors = oRSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='momentum', M=50,
                                        pretrain_epochs=500)
ORS_results17, ors17 = monit_and_train_ors('ors_exp11_BBC_mfcd_momentum_1000',
                                         ors, dataset_BBC, eval_each=20)


## rmsprop

#20ng
ors = oRSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='rmsprop', M=50,
                                        pretrain_epochs=500)
ORS_results18, ors18 = monit_and_train_ors('ors_exp11_20ng_mfcd_rmsprop_1000',
                                         ors, dataset_20ng, eval_each=20)

#m10
ors = oRSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='rmsprop', M=50,
                                        pretrain_epochs=500)
ORS_results19, ors19 = monit_and_train_ors('ors_exp11_M10_mfcd_rmsprop_1000',
                                        ors, dataset_M10, eval_each=20)
```

```
#bbc
ors = oRSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='rmsprop', M=50,
                                        pretrain_epochs=500)
ORS_results20, ors20 = monit_and_train_ors('ors_exp11_BBC_mfcd_rmsprop_1000',
                                        ors, dataset_BBC, eval_each=20)



## adam

#20ng
ors = oRSM(num_topics=20, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='adam', M=50,
                                        pretrain_epochs=500)
ORS_results21, ors21 = monit_and_train_ors('ors_exp11_20ng_mfcd_adam_1000',
                                        ors, dataset_20ng, eval_each=20)

#m10
ors = oRSM(num_topics=10, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='adam', M=50,
                                        pretrain_epochs=500)
ORS_results22, ors22 = monit_and_train_ors('ors_exp11_M10_mfcd_adam_1000', ors
                                        , dataset_M10, eval_each=20)

#bbc
ors = oRSM(num_topics=5, epochs=1000, btsz=20, lr=0.0001, cd_type='mfcd',
                                        train_optimizer='adam', M=50,
                                        pretrain_epochs=500)
ORS_results23, ors23 = monit_and_train_ors('ors_exp11_BBC_mfcd_adam_1000', ors
                                        , dataset_BBC, eval_each=20)
```

```
### esempio di codice per
### fare ottimizzazione bayesiana del modello oRSM

## 20ng
```

```
ors = oRSM(num_topics=20, btsz=20, lr=0.0001, cd_type='mfcd', train_optimizer=
                                        'sgd',
M=50, pretrain_epochs=500)
search_space = {"lr": Real(low=0.00001, high=0.005), "btsz": Integer(low=1,
                                        high=50), 'decay': Real(low=0.0, high=0
                                        .2)}
eval_topdiv10 = TopicDiversity(topk=10)
ORS_results, ors, resdict = bo_ors('bo_ors_exp6_mfcd_sgd_20ng', ors,
                                        dataset_20ng, search_space,
eval_metric=eval_topdiv10,
epochs_bo=20, epochs_train=1000,
number_of_call=15, model_runs=1, eval_each=20,
surrogate_model='GP', acq_func='LCB')



## M10

ors = oRSM(num_topics=10, btsz=20, lr=0.0001, cd_type='mfcd', train_optimizer=
                                        'sgd',
M=50, pretrain_epochs=500)
search_space = {"lr": Real(low=0.00001, high=0.005), "btsz": Integer(low=1,
                                        high=50), 'decay': Real(low=0.0, high=0
                                        .2)}
eval_topdiv10 = TopicDiversity(topk=10)
ORS_results, ors, resdict = bo_ors('bo_ors_exp6_mfcd_sgd_M10', ors,
                                        dataset_M10, search_space,
eval_metric=eval_topdiv10,
epochs_bo=20, epochs_train=1000,
number_of_call=15, model_runs=1, eval_each=20,
surrogate_model='GP', acq_func='LCB')


## BBC
```

```
ors = oRSM(num_topics=5, btsz=20, lr=0.0001, cd_type='mfcd', train_optimizer='
                                        sgd',
M=50, pretrain_epochs=500)
search_space = {"lr": Real(low=0.00001, high=0.005), "btsz": Integer(low=1,
                                        high=50), 'decay': Real(low=0.0, high=0
                                        .2)}
eval_topdiv10 = TopicDiversity(topk=10)
ORS_results, ors, resdict = bo_ors('bo_ors_exp6_mfcd_sgd_bbc', ors,
                                        dataset_BBC, search_space,
eval_metric=eval_topdiv10,
epochs_bo=100, epochs_train=1000,
number_of_call=15, model_runs=1, eval_each=20,
surrogate_model='GP', acq_func='LCB')
```

List of Figures

List of Tables

List of Algorithms

List of Proofs

References

[Abdelrazek et al., 2023] Abdelrazek, A., Eid, Y., Gawish, E., Medhat, W., and Hassan, A. (2023). Topic modeling algorithms and applications: A survey. *Information Systems*, 112:102131.

[Archetti and Candelieri, 2019a] Archetti, F. and Candelieri, A. (2019a). The acquisition function. In *Bayesian Optimization and Data Science*, pages 57–72. Springer.

[Archetti and Candelieri, 2019b] Archetti, F. and Candelieri, A. (2019b). *Bayesian optimization and data science*, volume 849. Springer.

[Arora et al., 2012] Arora, S., Ge, R., and Moitra, A. (2012). Learning topic models–going beyond svd. In *2012 IEEE 53rd annual symposium on foundations of computer science*, pages 1–10. IEEE.

[Barde and Bainwad, 2017] Barde, B. V. and Bainwad, A. M. (2017). An overview of topic modeling methods and tools. In *2017 international conference on intelligent computing and control systems (ICICCS)*, pages 745–750. IEEE.

[Blei et al., 2001] Blei, D., Ng, A., and Jordan, M. (2001). Latent dirichlet allocation. *Advances in neural information processing systems*, 14.

[Blei et al., 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

[Deng, 2012] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142.

[Dieng et al., 2020] Dieng, A. B., Ruiz, F. J., and Blei, D. M. (2020). Topic modeling in embedding spaces. *Transactions of the Association for Computational Linguistics*, 8:439–453.

[Ding et al., 2008] Ding, C., Li, T., and Peng, W. (2008). On the equivalence between non-negative matrix factorization and probabilistic latent semantic indexing. *Computational Statistics & Data Analysis*, 52(8):3913–3927.

[Griffiths and Steyvers, 2004] Griffiths, T. L. and Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National academy of Sciences*, 101(suppl_1):5228–5235.

[Gupta et al., 2024] Gupta, T., Yadav, A. K., and Kumar, M. (2024). Topic modeling with latent dirichlet allocation (lda) using tf-idf and bag of words. In *International Conference on Data Analytics & Management*, pages 1–12. Springer.

[Hastie et al., 2008] Hastie, T., Tibshirani, R., and Friedman, J. (2008). Random forests. In *The elements of statistical learning: Data mining, inference, and prediction*, pages 587–604. Springer.

[Hinton and Roweis, 2002] Hinton, G. E. and Roweis, S. (2002). Stochastic neighbor embedding. *Advances in neural information processing systems*, 15.

[Hinton and Salakhutdinov, 2009] Hinton, G. E. and Salakhutdinov, R. R. (2009). Replicated softmax: an undirected topic model. *Advances in neural information processing systems*, 22.

[Hoffman et al., 2010] Hoffman, M., Bach, F., and Blei, D. (2010). Online learning for latent dirichlet allocation. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

[Hofmann, 1999] Hofmann, T. (1999). Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57.

[Hofmann, 2013] Hofmann, T. (2013). Probabilistic latent semantic analysis. *arXiv preprint arXiv:1301.6705*.

[Hopfield, 2007] Hopfield, J. J. (2007). Hopfield network. *Scholarpedia*, 2(5):1977.

[Hopfield and Tank, 1985] Hopfield, J. J. and Tank, D. W. (1985). "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152.

[Hotho et al., 2005] Hotho, A., Nürnberger, A., and Paaß, G. (2005). A brief survey of text mining. *Journal for Language Technology and Computational Linguistics*, 20(1):19–62.

[Jian et al., 2017] Jian, D., Kar, S., and Moura, J. (2017). Distributed convergence verification for gaussian belief propagation.

[Ke and Wang, 2024] Ke, Z. T. and Wang, M. (2024). Using svd for topic modeling. *Journal of the American Statistical Association*, 119(545):434–449.

[Kim et al., 2016] Kim, M., Kang, K., Park, D., Choo, J., and Elmqvist, N. (2016). Topiclens: Efficient multi-level visual topic exploration of large-scale document collections. *IEEE transactions on visualization and computer graphics*, 23(1):151–160.

[Kingma, 2014] Kingma, D. P. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kuang et al., 2017] Kuang, D., Brantingham, P., and Bertozzi, A. (2017). Crime topic modeling. *Crime Science*, 6.

[Lee and Seung, 2000] Lee, D. and Seung, H. S. (2000). Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[Min et al., 2010] Min, K., Zhang, Z., Wright, J., and Ma, Y. (2010). Decomposing background topics from keywords by principal component pursuit. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 269–278.

[Mirzal, 2016] Mirzal, A. (2016). Clustering and latent semantic indexing aspects of the singular value decomposition. *International Journal of Information and Decision Sciences*, 8(1):53–72.

[Oh et al., 2020] Oh, S., Baggag, A., and Nha, H. (2020). Entropy, free energy, and work of restricted boltzmann machines. *Entropy*, 22(5).

[Phan et al., 2008] Phan, X.-H., Nguyen, L.-M., and Horiguchi, S. (2008). Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceedings of the 17th international conference on World Wide Web*, pages 91–100.

[Sharma et al., 2017] Sharma, D., Kumar, B., and Chand, S. (2017). A survey on journey of topic modeling techniques from svd to deep learning. *International Journal of Modern Education and Computer Science*, 9(7):50.

[Smolensky, 1986] Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Technical report.

[Srivastava et al., 2013] Srivastava, N., Salakhutdinov, R. R., and Hinton, G. E. (2013). Modeling documents with deep boltzmann machines. *arXiv preprint arXiv:1309.6865*.

[Srivastava and Sundararaghavan, 2023] Srivastava, S. and Sundararaghavan, V. (2023). Generative and discriminative training of boltzmann machine through quantum annealing. *Scientific Reports*, 13(1):7735.

[Sudderth, 2014] Sudderth, E. (2014). Belief propagation. `https://cs.brown.edu/courses/cs242/lectures/lec02a_beliefProp.pdf`. Lecture 2A, CSCI 2420: Probabilistic Graphical Models, Brown University Computer Science.

[Teh and Hinton, 2000] Teh, Y. W. and Hinton, G. E. (2000). Rate-coded restricted boltzmann machines for face recognition. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

[Terragni and Fersini, 2021] Terragni, S. and Fersini, E. (2021). An empirical analysis of topic models: uncovering the relationships between hyperparameters, document length

and performance measures. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 1408–1416.

[Terragni et al., 2021] Terragni, S., Fersini, E., Galuzzi, B. G., Tropeano, P., and Candelieri, A. (2021). Octis: Comparing and optimizing topic models is simple! In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: system demonstrations*, pages 263–270.

[Tieleman, 2008] Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071.

[Van Der Maaten, 2009] Van Der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. In *Artificial intelligence and statistics*, pages 384–391. PMLR.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Vogt et al., 2014] Vogt, D., Magnaguagno, F., Clever, H., and Billard, A. (2014). Visual trajectory analysis via replicated softmax-based models. *Signal, Image and Video Processing*, 8(Suppl 1):183–190.

[Wang and Zhang, 2012] Wang, Y.-X. and Zhang, Y.-J. (2012). Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on knowledge and data engineering*, 25(6):1336–1353.

[Welling and Hinton, 2002] Welling, M. and Hinton, G. E. (2002). A new learning algorithm for mean field boltzmann machines. In *International conference on artificial neural networks*, pages 351–357. Springer.

[Wu et al., 2024] Wu, X., Nguyen, T., and Luu, A. T. (2024). A survey on neural topic models: methods, applications, and challenges. *Artificial Intelligence Review*, 57(2):18.

[Xu et al., 2017] Xu, B., Lin, H., Wang, L., Lin, Y., Xu, K., Wei, X., Huang, Dong", e. J., Nie, J., Ruan, T., Liu, Y., and Qian, T. (2017). Tripartite-replicated softmax model for document representations. In *Information Retrieval*, pages 109–121, Cham. Springer International Publishing.

[Yedidia et al., 2003] Yedidia, J. S., Freeman, W. T., Weiss, Y., et al. (2003). Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8(236-239):0018–9448.