UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI INGEGNERIA ELETTRICA ELETTRONICA E INFORMATICA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Mario Roberto Nuñez Pereira

CLASSIFICAZIONE NON SUPERVISIONATA DEI GRAFI MEDIANTE GEOMETRIC DEEP LEARNING

TESI DI LAUREA

Relatore:
Prof. Giuseppe Mangioni

Correlatori:
Ing. Marco Grassia

Anno Accademico 2021 – 2022

Indice

Introduzione

Negli ultimi anni si è avuto un sempre maggior interesse per lo sviluppo di sistemi informatici in grado di analizzare i dati forniti ed apprendere autonomamente delle rappresentazioni di quest'ultimi sempre più rifinite con il passare del tempo, al fine di comprendere come svolgere al meglio un determinato compito. La particolare disciplina che si occupa dello sviluppo di tali sistemi, comunemente detti "*modelli*", è nota come **Machine Learning**.

Tuttavia, a causa dello sviluppo tecnologico e delle strutture virtuali, i dati da analizzare diventano man mano più complessi e questo rende la loro analisi sempre più difficile. In particolare, l'apprendimento di rappresentazioni per dati strutturati complessi è un compito impegnativo. Nell'ultimo decennio sono stati sviluppati molti modelli di successo per alcuni tipi di dati strutturati, compresi quelli definiti su di un dominio euclideo discretizzato. Ad esempio, i dati sequenziali, come testi o video, possono essere modellati tramite ***reti neurali ricorrenti*** (RNN), capaci di catturare le informazioni sequenziali, producendo rappresentazioni efficienti, come dimostrato per i compiti di traduzione automatica e di riconoscimento vocale. Un altro esempio sono le ***reti neurali convoluzionali*** (CNN), che parametrizzano le reti neurali sulla base di priori strutturali come l'invarianza allo spostamento (*shift-invariance*), e che hanno raggiunto prestazioni senza precedenti in compiti come la classificazione delle immagini. Questi grandi successi sono però limitati a particolari tipi di dati, che hanno una struttura relazionale semplice.

In molti contesti, i dati non sono altrettanto regolari: si presentano spesso strutture relazionali complesse e l'estrazione delle informazioni da esse è fondamentale per capire come gli oggetti interagiscono tra loro. L'esempio principe di questa tipologia di dati sono i grafi; strutture in grado di rappresentare dati relazionali complessi in domini non euclidei. Essi compaiono in diversi ambiti come le reti sociali, la chimica computazionale, la biologia, i sistemi di raccomandazione ed altri. I grafi forniscono informazioni esplicite sull'accoppiamento tra le singole unità, oltre ad un quadro ben definiti per l'assegnazione di proprietà ai nodi ed ai loro collegamenti. Tuttavia, per i dati grafo-strutturati è difficile individuare reti con forti priori strutturali, poiché le strutture possono essere arbitrarie e possono variare in modo significativo tra grafi diversi. In particolare, operazioni come le classiche convoluzioni non possono essere applicate direttamente a domini di grafi irregolari.

La complessità di questi dati ha portato allo sviluppo della ricerca sul **Geometric Deep Learning** (GDL), che mira ad applicare le tecniche di deep learning a dati non euclidei. Il crescente uso dei grafi nelle applicazioni del mondo reale ha portato ad un aumento dell'interesse per lo sviluppo di

metodi d'apprendimento automatico per dati strutturati a grafo. Questi metodi, detti di **Graph Representation Learning** (GRL), mirano ad apprendere rappresentazioni vettoriali continue a bassa dimensione per dati grafo-strutturati, chiamate *incorporamenti* (*embeddings*).

Come ogni branca del Machine Learning, il GRL può essere suddiviso in due classi di problemi d'apprendimento, ***non supervisionato*** e ***supervisionato*** (o semi-supervisionato). La prima classe punta ad apprendere rappresentazioni che preservino la struttura di un grafo in ingresso, mentre la seconda apprende rappresentazioni per uno specifico compito di predizione a valle. Le rappresentazioni ottenute da queste due categorie di apprendimento sono più o meno ideali per lo svolgimento di due tipologie di compiti, ovvero induttivi (e.g. previsione degli attributi delle molecole, dove ogni molecola è un grafo) o trasduttivi (e.g. previsione delle proprietà degli utenti in una grande rete sociale).

In passato, è stato già svolto molto lavoro sull'apprendimento, supervisionato e non, di rappresentazioni a livello di nodo, ovvero incorporamenti vettoriali a bassa dimensione di singoli nodi. Un altro campo che ha recentemente attirato molta attenzione è l'apprendimento di rappresentazioni di interi grafi. Tale problema è fondamentale in molte applicazioni scientifiche e ci sono anche stati alcuni recenti progressi basati su algoritmi di passaggio di messaggi neurali che apprendono in modo supervisionato le rappresentazioni di interi grafi, con l'ottenimento di risultati all'avanguardia su una varietà di diversi compiti di previsione. Tuttavia, uno degli ostacoli più difficili per l'apprendimento supervisionato sui grafi è il fatto che spesso risulta molto costoso o addirittura impossibile raccogliere le etichette annotate dei dati in ingresso. Di conseguenza, l'elaborazione di metodi in grado di apprendere rappresentazioni non supervisionate di un intero grafo, anziché dei nodi, è un passo importante per lavorare con grafi non etichettati.

Di seguito, in questo elaborato proponiamo un approfondimento degli argomenti sopra citati ed una descrizione di vari metodi di GRL, suddivisi per tipologia d'apprendimento, per poi condurre uno studio sperimentale atto a valutare le prestazioni di un modello di rete neurale, basato sul metodo **GAT**, per il compito induttivo di classificazione non supervisionata dei grafi.

Capitolo 1 – Machine Learning

Quando si parla di **Machine Learning** (ML), anche detto **apprendimento automatico**, si parla di una particolare branca dell'informatica, sottoinsieme dell'**Intelligenza Artificiale** (AI), comprendente differenti meccanismi che permettono la creazione di sistemi intelligenti in grado di apprendere e/o migliorare le proprie capacità e prestazioni nel tempo in base ai dati che utilizzano. Il sistema, quindi, sarà in grado di imparare a svolgere determinati compiti migliorando gradualmente tramite l'esperienza [1].

L'obiettivo principale del Machine Learning è, più nello specifico, la creazione di una *rete neurale* ed/o un *modello d'apprendimento* che il sistema dovrà utilizzare per comprendere come svolgere delle specifiche operazioni o raggiungere un particolare scopo designato dallo sviluppatore. La rete neurale ed il modello, ovviamente, varieranno a seconda dello scopo scelto, del metodo d'apprendimento utilizzato e della tipologia di dati forniti al sistema. In generale, l'uso più diffuso del machine learning è quello della "*classificazione"*, ovvero l'operazione di riconoscimento delle caratteristiche dei dati analizzati per distinguerli in quelle che vengono definite come classi d'appartenenza.

Gli algoritmi, o metodi, sono i motori che alimentano tutto ciò che comprende il Machine Learning. I due tipi principali di metodi di ML attualmente utilizzati sono: **Supervised Learning** e **Unsupervised Learning** (*apprendimento supervisionato* e *non supervisionato*). La differenza tra queste due tipologie viene definita dal modo in cui ciascun algoritmo apprende i dati per fare previsioni [2].

- ***Supervised Learning***: gli algoritmi di ML supervisionato sono i più utilizzati. Tale metodo consiste nel fornire al sistema una serie di nozioni specifiche e codificate, ossia modelli ed esempi che permettono di costruire un vero e proprio **database di informazioni ed esperienze**. In questo modo, quando il sistema si trova di fronte ad un problema, non dovrà fare altro che **attingere alle esperienze inserite al suo interno**, analizzarle, e decidere quale risposta dare sulla base di esperienze già codificate. Questo tipo di apprendimento è, in qualche modo, fornito già confezionato ed il sistema deve essere solo in grado di scegliere quale sia la migliore risposta allo stimolo che le viene dato. Gli algoritmi che fanno di questa tipologia d'apprendimento hanno la capacità di effettuare delle **ipotesi induttive**, ossia ipotesi che possono essere ottenute scansionando una serie di problemi specifici per ottenere una soluzione idonea ad un problema di tipo generale [1].

- ***Unsupervised Learning***: esso prevede che le informazioni inserite all'interno del sistema non siano codificate, ovvero la macchina ha la possibilità di **attingere a determinate informazioni senza avere alcun esempio del loro utilizzo** e, quindi, senza avere conoscenza dei risultati attesi a seconda della scelta effettuata. Dovrà essere il sistema stesso, quindi, a catalogare tutte le informazioni in proprio possesso, organizzarle ed imparare il loro significato, il loro utilizzo e, soprattutto, il risultato a cui esse portano. Tale metodo offre maggiore libertà di scelta al sistema che dovrà organizzare le informazioni in maniera intelligente ed imparare quali sono i risultati migliori per le differenti situazioni che si presentano [1].

Attualmente, il Machine Learning è utilizzato ovunque. Un tipico esempio di applicazione del ML è il riconoscimento delle immagini, già utilizzabile sulla quasi totalità dei dispositivi odierni. Oltre a ciò, quando interagiamo con le banche, acquistiamo online o utilizziamo i social media, vengono usati algoritmi di ML per rendere la nostra esperienza efficiente, facile e sicura [2].

Tra le varie sottocategorie che compongono il ML, il **Geometric Deep Learning**, branca specializzata della sottocategoria del **Deep Learning**, è lo strumento da noi utilizzato per condurre lo studio trattato in questo elaborato. Al fine di una migliore comprensione dello studio condotto, degli esperimenti e dei risultati ottenuti, verrà data una descrizione adeguata dei suddetti argomenti.

1.1 – Deep Learning

Il **Deep Learning** (DL), o **apprendimento profondo**, è quel campo di ricerca del ML che si basa sulla creazione di modelli di apprendimento su più livelli, in cui gli algoritmi di reti neurali artificiali sono modellati per funzionare come l'apparato cerebrale umano, imparando da grandi quantità di dati organizzati in collezioni chiamate *dataset* [2].

Nello specifico, il funzionamento del DL prevede più livelli di *reti neurali* che sono algoritmi di calcolo statistico modellati approssimativamente sul modo di lavorare dei cervelli umani. La formazione con grandi quantità di dati è ciò che configura i neuroni nella rete neurale. Il risultato è un modello di deep learning che, una volta formato, elabora nuovi dati. I modelli di DL acquisiscono informazioni da più origini dati e analizzano tali dati in tempo reale, senza la necessità di intervento umano. Nel DL, le **unità di elaborazione grafica** (**GPU**) sono ottimizzate per i modelli di formazione poiché possono elaborare più calcoli contemporaneamente [2].

In termini semplici, il Deep Learning è un nome per le reti neurali con molti livelli o strati (*layers*). Per dare un senso ai dati osservativi, come foto o audio, le reti neurali passano i dati attraverso livelli interconnessi di nodi. Quando le informazioni passano attraverso un livello, ogni nodo del livello esegue operazioni semplici sui dati e passa in modo selettivo i risultati ad altri nodi. Ogni livello successivo si concentra su una funzionalità di livello superiore rispetto all'ultima, finché la rete non crea l'output [2].

Tra il livello di input e di output ci sono dei livelli nascosti (*hidden layers*). Mentre una rete neurale basilare potrebbe avere uno o due livelli nascosti, una rete DL potrebbe averne decine o addirittura centinaia. L'aumento del numero di livelli e nodi diversi può aumentare la precisione di una rete ma può anche comportare che un modello richiederà più parametri e risorse computazionali. Il DL classifica le informazioni attraverso i suddetti livelli di reti neurali, che hanno un set di input che riceve dati grezzi. Più livelli consentono risultati più precisi, il che significa anche essere in grado di eseguire classificazioni di dati più complesse. Inoltre, un algoritmo di DL dev'essere formato con gradi set di dati, poiché più dati riceve, più esso sarà accurato [2].

La formazione del modello di DL è ad uso intensivo di risorse. Per questo motivo, quando la rete neurale acquisisce gli input, essi vengono elaborati negli strati nascosti usando dei pesi (parametri che rappresentano la forza di connessione tra gli input) che vengono regolati durante la formazione, ed il modello quindi emette una previsione. La regolazione dei pesi avviene in base agli input di formazione per fare previsioni migliori. Nel DL molto tempo viene impiegato per la formazione delle grandi quantità di dati, ragion per cui la computazione ad elevate prestazioni tramite l'uso di GPU (se disponibili) è così importante [2].

Il principale vantaggio del DL è quello di essere in grado di rivelare *insight* e *relazioni nascoste* dai dati che in precedenza non erano visibili. Oltre a ciò, abbiamo quanto segue:

- **Analisi dei dati non strutturati**: gli algoritmi di DL possono essere formati per esaminare dati di testo analizzando post, notizie e sondaggi dei social media per fornire insight utili sul business e sui clienti.
- **Etichettatura dati**: il DL richiede dati con etichetta per la formazione, ma una volta addestrato il modello sarà in grado di etichettare nuovi dati e identificare diversi tipi di dati autonomamente.

- **Engineering delle caratteristiche**: un algoritmo di DL può risparmiare tempo perché non chiede agli utenti di estrarre manualmente le funzioni dai dati grezzi.
- **Efficienza**: quando un algoritmo di DL viene adeguatamente formato, può eseguire migliaia di attività in modo continuo e più veloce rispetto agli esseri umani.
- **Formazione**: le reti neurali utilizzate nel DL hanno la capacità di essere applicate a molti tipi di dati ed applicazioni diversi [2].

È importante sottolineare che Deep Learning e Machine Learning sono correlati tra loro, ma hanno caratteristiche differenti:

- Il Machine Learning è incentrato sulla creazione di applicazioni che possono apprendere dati per migliorarne l'accuratezza nel tempo, con algoritmi che possono essere formati per trovare modelli che consentano di prendere decisioni e previsioni migliori, ma in genere richiede l'intervento umano.
- Il Deep Learning consente ai computer di risolvere problemi più complessi, utilizzando modelli che possono anche creare nuove funzioni da soli [2].

1.2 – Geometric Deep Learning

La stragrande maggioranza del Deep Learning viene eseguita su **dati euclidei**. Ciò include i tipi di dati nel dominio unidimensionale e bidimensionale, come immagini, testo e audio. Tuttavia, tutto ciò che possiamo osservare nel nostro mondo esiste in 3D e i nostri dati dovrebbero riflettere questa caratteristica. Per fare in modo che l'apprendimento automatico arrivi a tale livello possiamo fare uso del **Geometric Deep Learning** (GDL), ovvero il campo del Deep Learning che mira a costruire reti neurali in grado di apprendere da **dati non euclidei**. Quest'ultimi possono rappresentare elementi e concetti più complessi con maggiore precisione rispetto ad una rappresentazione 1D o 2D [3].

Quando rappresentiamo le cose in un modo non euclideo, gli diamo un **bias induttivo**. Ciò si basa sull'intuizione che, forniti dei dati di tipo, formato e dimensione arbitrari, è possibile assegnare al modello la priorità ad apprendere determinati *patterns* **modificando la struttura di tali dati** [3].

La principale tipologia di dati non euclidei utilizzati nel GDL sono i **grafi**. I grafi sono un tipo di struttura dati costituita da **nodi** (*entità*) che sono collegati da **edges** (*relazioni*). Questa struttura dati astratta può essere utilizzata per modellare quasi tutto. I grafi sono una tipologia di dati molto importante nel GDL poiché ci consentono di rappresentare le singole caratteristiche, fornendo

anche informazioni su relazioni e struttura. Esistono vari tipi di grafi, ognuno con un insieme di regole, proprietà e possibili azioni [3].

Vi sono due applicazioni molto popolari nel GDL:

- ***Molecular Modeling and Learning***: per un esempio concreto di come l'apprendimento mediante grafi può migliorare i compiti di machine learning esistenti, applicati alle scienze computazionali come biologia, chimica e fisica, possiamo esaminare una delle loro principali difficoltà, cioè la rappresentazione di concetti, entità ed interazioni, poiché i risultati che si ottengono dipendono da molti fattori e relazioni esterne. Ciò è più evidente quando si parla di:
    - Reti d'interazione proteica;
    - Reti neurali;
    - Molecole;
    - Diagrammi di Feyman;
    - Mappe cosmologiche.

  I nostri attuali metodi di rappresentazione computazionale di questi concetti possono essere considerati "*in perdita*" (*lossy*), poiché perdiamo molte informazioni preziose. Tuttavia, **trattando gli atomi come nodi ed i legami come edge, possiamo salvare informazioni strutturali che possono essere utilizzate a valle nella previsione o nella classificazione**. Quindi, invece di usare una stringa che rappresenta una molecola come input per una **rete neurale ricorrente** (RNN), possiamo usare il grafo molecolare come input per il suo equivalente geometrico.
- ***3D Modeling and Learning***: per mostrare come il GDL ci consenta d'imparare da tipi di dati mai utilizzati prima è necessario porre un esempio. Consideriamo una persona che posa per una macchina fotografica (Figura 1.2.1).

Figura 1.2.1: Persone che posano per l'acquisizione di un'immagine 2D

Quest'immagine è 2D, anche se nella nostra mente siamo consapevoli che rappresenta una persona 3D. I nostri algoritmi attuali, vale a dire le **reti neurali convoluzionali** (CNN), sono in grado di predire delle "*etichette*" come la persona che posa e/o il tipo di posa data solo l'immagine 2D. **La difficoltà sorge quando le pose diventano estreme e l'angolo non è più fisso**. Spesso in un'immagine possono essere presenti vestiti o oggetti che ostruiscono la visualizzazione di un algoritmo, rendendo difficile prevedere la posa. Invece, avendo a disposizione un modello 3D della stessa persona (Figura 1.2.2), invece d'imparare da una rappresentazione 2D, che limita i dati ad un singolo angolo prospettico, sarebbe possibile eseguire una convoluzione sull'oggetto 3D stesso.

Analogamente alle CNN tradizionali, **il kernel passerebbe attraverso ogni "pixel" rappresentato come un nodo in una nuvola di punti** (ovvero, come un grafo che avvolge l'oggetto 3D). In questo modo, tutti gli angoli e le fessure del modello 3D verranno coperti e le informazioni verranno prese in considerazione. In breve, quindi, la differenza tra le CNN classiche ed il loro equivalente geometrico (detto GNN) è la previsione dell'etichetta di un oggetto data una sua immagine, rispetto alla previsione dell'etichetta di un oggetto dato un modello 3D.

Figura 1.2.2: rappresentazione 3D di una persona

Sostanzialmente, il *Geometric Deep Learning* è quella forma di machine learning che si concentra sul fornire dei metodi per l'apprendimento delle rappresentazioni di oggetti tridimensionali e grafi, il cui funzionamento pratico prevede la traduzione della struttura di quest'ultimi in una forma matematica trattabile nota come **Matrice di Adiacenza** (*Adjacency Matrix*) e la conservazione delle informazioni ad esse relative in una **Matrice delle Caratteristiche** (*Features Matrix*). Le matrici di adiacenza sono costruite in base ai collegamenti esistenti tra i nodi ed alle loro posizioni nei grafi (Figura 1.2.3).

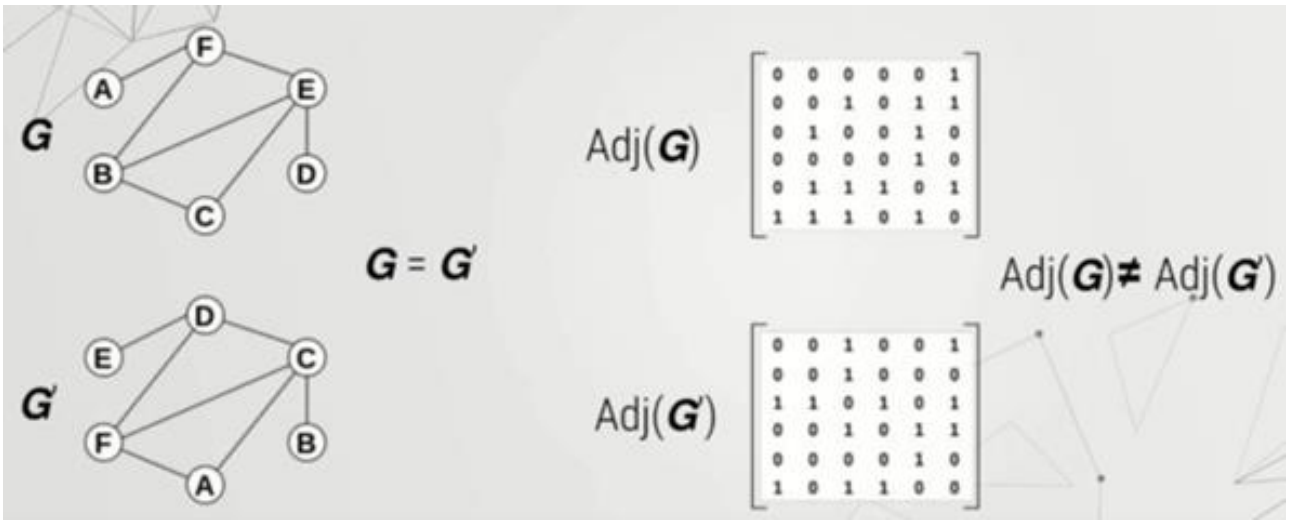

Figura 1.2.3: matrici di adiacenza diverse per grafi isomorfici

Come abbiamo precedentemente accennato, anche le normali CNN sono in grado di processare le matrici, anche se provenienti da immagini bidimensionali, ma non possono essere usate in questo caso, poiché le matrici di adiacenza possono variare a causa dell'aggiunta/rimozione di un nodo al grafo (aumentando/diminuendo la loro dimensione) o cambiando l'ordine (che porta ad una matrice completamente diversa, Figura 1.2.3). Successivamente, le matrici ottenute vengono coinvolte in una serie di operazioni matematiche complesse, variabili a seconda del metodo utilizzato, che mirano ad enfatizzare determinati aspetti dei vari nodi o degli interi grafi per creare, ad esempio, una *Graph Neural Network* che si differenzia per il compito assegnato dallo sviluppatore dello stesso.

Capitolo 2 – Metodi per la classificazione dei grafi

La ***classificazione dei grafi*** è una delle possibili applicazioni dei metodi di *Graph Representation Learning*, solitamente supervisionata, il cui compito è quello di prevedere le etichette dei grafi in ingresso. I compiti di classificazione sono intrinsecamente *induttivi*, poiché al momento del test vengono presentati nuovi grafi. A differenza dei compiti di previsione a livello di nodo (ad esempio, la classificazione dei nodi) e a livello di edge (come la previsione dei collegamenti), i compiti di classificazione dei grafi richiedono un ulteriore tipo di raggruppamento (***pooling***), al fine di aggregare le informazioni a livello di nodo in informazioni a livello di grafo. Per funzionare, tale funzione di pooling dev'essere invariante rispetto all'ordine dei nodi [4].

Il **Graph Representation Learning** (GRL) è quel ramo del ML che si occupa di studiare e sviluppare nuovi metodi per l'apprendimento automatico delle rappresentazioni dei grafi. I metodi di GRL sono stati generalmente suddivisi in tre categorie:

- **Network Embedding** (NE): concentrata sull'apprendimento di rappresentazioni non supervisionate della struttura relazionale.
- **Graph Regularized Neural Network** (GRNN): sfrutta i grafi per aumentare le perdite delle reti neurali con un obiettivo di regolarizzazione per l'apprendimento semi-supervisionato.
- **Graph Neural Network** (GNN): mira ad apprendere funzioni differenziabili su topologie discrete con struttura arbitraria.

I metodi di GRL possono essere applicati ad un'ampia gamma di applicazioni, sia ***supervisionate*** che ***non supervisionate***. Nelle *applicazioni non supervisionate*, le etichette specifiche per il compito da svolgere non vengono elaborate per l'apprendimento degli **embeddings** (*incorporamenti*), ovvero le strutture in grado di contenere in sé le informazioni presenti in un'altra struttura, piuttosto, il grafo viene utilizzato come forma di auto-supervisione. Invece, nelle applicazioni supervisionate, gli incorporamenti a livello di nodo sono direttamente ottimizzati per un compito specifico, come la classificazione di nodi o grafi, ed in questo caso, si possono applicare metodi supervisionati per l'apprendimento degli incorporamenti. Una volta ottenuti gli incorporamenti tramite il metodo selezionato, sia esso supervisionato o meno, è possibile sfruttarli per uno dato compito e noi abbiamo deciso di concentrarci su quello di classificazione. Di seguito andremo quindi ad analizzare i diversi metodi noti utilizzabili per effettuare il compito da noi scelto, raggruppati in tre tipologie: **Unsupervised Graph Embedding**, **Supervised Graph Embedding** e **Graph Kernels** [4].

2.1 – Unsupervised Graph Embedding

I metodi di seguito descritti mappano un grafo, i suoi nodi e/o i suoi bordi, su uno spazio vettoriale continuo, senza utilizzare etichette specifiche per il grafo o i suoi nodi. Alcuni di questi metodi ottimizzano l'obiettivo di imparare un incorporamento che preservi la struttura del grafo, ad esempio imparando a ricostruire una matrice di somiglianza o dissimilarità tra nodi, come la matrice di adiacenza, mentre altri applicano un obiettivo contrastivo, ad esempio contrapponendo coppie di nodi/grafi vicini a coppie di nodi/grafi distanti. Per tutti questi metodi non supervisionati, l'output del modello sull'intero grafo è $\in \mathbb{R}^{|V|\times|V|}$ di cui la funzione obiettivo incoraggia a ben prevedere l'adiacenza W o la sua trasformazione $s(W)$. Di conseguenza, questi modelli possono calcolare rappresentazioni latenti dei nodi addestrate per ricostruire la struttura del grafo. Questa rappresentazione latente può successivamente essere utilizzata per svolgere vari compiti, tra cui la previsione dei collegamenti, la classificazione dei nodi o la classificazione dei grafi [4].

2.1.1 – Metodi d'incorporazione superficiale (non supervisionati)

I **metodi d'incorporazione superficiale** (*non supervisionati*) sono metodi d'incorporamento *trasduttivo* dei grafi in cui la funzione di codifica è una semplice ricerca dell'incorporamento, che non considera le caratteristiche dei nodi anche se presenti. Questi metodi superficiali programmano un encoder come una "*look-up table*" (tabella di consultazione), parametrizzandola con una matrice $\in \mathbb{R}^{|V|\times d}$, in cui ogni riga memorizza un vettore d'incorporamento d-dimensionale per un nodo. Più concretamente, ad ogni nodo $v_i \in V$ corrisponde un vettore di incorporamento a bassa dimensione apprendibile $Z_i \in \mathbb{R}^d$ e la funzione di codifica superficiale è semplicemente:

$$Z = \mathrm{ENC}(\Theta^E) = \Theta^E \in \mathbb{R}^{|V|\times d}.$$

Gli incorporamenti dei nodi possono essere appresi in modo che la struttura dei dati nello spazio delle incorporazioni corrisponda alla struttura del grafo sottostante. Ad alto livello, questo è simile ai metodi di riduzione della dimensionalità, tranne per il fatto che i dati di input potrebbero non avere una struttura lineare. In particolare, i metodi utilizzati per la riduzione della dimensionalità non lineare spesso iniziano costruendo un grafo discreto a partire dai dati (per approssimare la varietà degli stessi) e possono essere applicati a problemi d'incorporamento dei grafi. A seguire, analizziamo due dei tipi principali di metodi d'incorporazione superficiale, ovvero quelli basati sulla distanza (*distance-based methods*) e quelli basata sul prodotto esterno (*outer product-based*) [4].

- **Distance-based methods**: questi metodi ottimizzano le incorporazioni in modo che i punti più vicini nel grafo (misurati, ad esempio, dalla loro distanza dal grafo) rimangano il più vicino possibile nello spazio d'incorporamento, utilizzando una funzione di distanza predefinita. Formalmente, la rete di decodifica calcola la distanza delle coppie con una funzione di distanza $d_2(\cdot,\cdot)$, che può portare ad incorporamenti euclidei o non euclidei:

$$\widehat{W} = \mathrm{DEC}(Z;\ \Theta^D)$$
$$\text{con } \widehat{W}_{ij} = d_2(Z_i, Z_j)$$

- **Outer Product-based methods**: questi metodi, invece, si basano su prodotti di punti a coppie per calcolare le somiglianze tra i nodi, e la rete di decodifica può essere scritta come:

$$\widehat{W} = \mathrm{DEC}(Z;\ \Theta^D) = ZZ^T.$$

  Gli incorporamenti vengono quindi appresi minimizzando la perdita (*loss*) di regolarizzazione del grafo: $\mathcal{L}_{G,REG}(W, \widehat{W};\ \Theta) = d_1(s(W), \widehat{W})$.

È importante notare che per i metodi basati sulla distanza la funzione $s(\cdot)$ misura la dissimilarità o le distanze tra i nodi (valori più alti indicano coppie di nodi meno simili), mentre nei metodi a prodotto esterno essa misura una qualche nozione di somiglianza nel grafo (valori più alti indicano coppie più simili). In generale, i metodi d'incorporazione superficiale, per lo più, sono applicabili a compiti *trasduttivi* in cui esiste un solo grafo, che rimane fisso tra l'addestramento e l'inferenza [4].

2.1.2 – Auto-encoders

I metodi d'incorporazione superficiale, descritti prima, difficilmente catturano le strutture complesse non lineari che possono presentarsi nei grafi. Gli **Auto-encoders** (*auto-codificatori*) sono stati originariamente introdotti per superare questo problema utilizzando funzioni di codifica e decodifica delle ***Deep Neural Networks*** (DNN), grazie alla loro capacità di modellare le non linearità. Per questo, i metodi di auto-codifica sono più profondi di quelli superficiali, anche se ignorano la matrice di caratteristiche del nodo X. Si tratta di reti neurali *feed-forward* in cui l'input della rete è la matrice di adiacenza W. Invece di sfruttare la struttura del grafo attraverso il termine di regolarizzazione dello stesso, gli auto-encoders incorporano direttamente la matrice di adiacenza del grafo nella funzione di codifica. Gli auto-encoders hanno generalmente una rete di codifica e decodifica composta da più strati non lineari. Per gli auto-encoders a grafo, la funzione di codifica ha la forma:

$$Z = \mathrm{ENC}(W;\ \Theta^E).$$

In altre parole, l'encoder è una funzione della sola matrice di adiacenza W. Questi modelli sono addestrati minimizzando un obiettivo di errore di ricostruzione. Questa tipologia di metodi è particolarmente indicata quando si prevedono nuovi nodi al momento del test d'inferenza. Di seguito vengono descritti alcuni esempi che fanno uso degli auto-encoders [4].

- **Structural Deep Network Embedding** (SDNE): è un metodo che apprende auto-encoders che preservano la prossimità dei nodi di primo e secondo ordine. L'encoder SDNE prende in input un vettore di nodi, ovvero una riga della matrice di adiacenza impostata esplicitamente come $s(W) = W$, e produce incorporamenti di nodi Z. Il decoder SDNE restituisce una ricostruzione $\widehat{W}$, addestrata per recuperare la matrice di adiacenza del grafo originale (Figura 2.1.1).

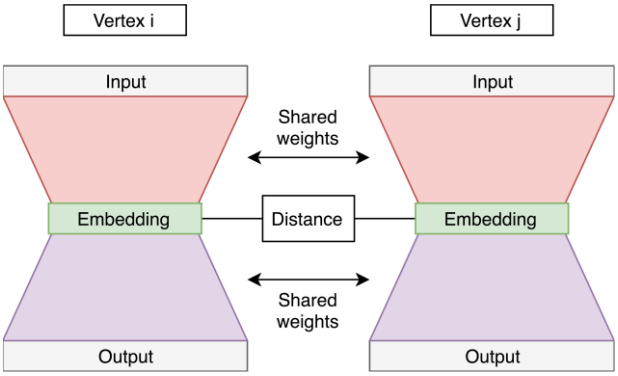

Figura 2.1.1: Illustrazione del modello SDNE. Lo strato di incorporazione (indicato con Z) è mostrato in verde

SDNE preserva la prossimità dei nodi del secondo ordine minimizzando la perdita di regolarizzazione del grafo tramite la formula:

$$\left\|\left(s(W) - \widehat{W}\right) \cdot B\right\|_{|F}^2 + \alpha_{SDNE} \sum_{ij} s(W)_{ij} \left\|Z_i - Z_j\right\|_2^2$$

dove B è la matrice indicatore per $s(W)$ con $B = 1[s(W) > 0]$. Si noti che il secondo termine è la perdita di regolarizzazione usata dai metodi d'incorporazione superficiale basati sulla distanza, mentre il primo termine è simile all'obiettivo di regolarizzazione della fattorizzazione della matrice, tranne per il fatto che $\widehat{W}$ non viene calcolato utilizzando prodotti esterni. Invece, SDNE calcola un incorporamento unico per ogni nodo del grafo utilizzando una rete di decodifica[4].

- **Deep neural Network for learning Graph Representation** (DNGR): simile a SDNE, DNGR utilizza auto-encoders per codificare e decodificare una matrice di similarità dei nodi, $s(W)$. La matrice di similarità viene calcolata utilizzando un metodo probabilistico chiamato *random surfing*, che restituisce una matrice di similarità probabilistica attraverso l'esplorazione del grafo con "passeggiate" casuali. Pertanto, DNGR cattura le dipendenze di ordine superiore nel grafo. La matrice di somiglianza $s(W)$ viene poi codificata e decodificata con una sequenza di *denoising* auto-encoders, che consentono di ridurre il disturbo in $s(W)$. DNGR è un metodo ottimizzato per minimizzare l'errore di ricostruzione [4]:

$$\mathcal{L}_{G,\mathrm{REG}}(W, \widehat{W};\ \Theta) = \left\| s(W) - \widehat{W} \right\|_F^2.$$

2.1.3 – Graph Neural Networks

Nelle **Graph Neural Networks** (GNN), sia la struttura del grafo che le caratteristiche dei nodi sono utilizzate nella funzione di codifica per apprendere le rappresentazioni strutturali dei nodi:

$$Z = \mathrm{ENC}(X, W;\ \Theta^{\mathrm{E}}).$$

Le GNN sono metodi profondi che elaborano sia l'adiacenza W sia le caratteristiche dei nodi X. Questi metodi sono induttivi e in genere superano le due tipologie precedenti per i compiti di classificazione dei nodi/grafi, soprattutto quando i nodi hanno delle caratteristiche [4].

In questa sezione vengono analizzate alcune versioni non supervisionate di GNN, mentre più avanti verranno presentate delle versioni supervisionate ().

- **Variational Graph Auto-Encoders** (VGAE): in questo metodo vengono utilizzate le convoluzioni del grafo per apprendere gli incorporamenti dei nodi $Z = \mathrm{GCN}(W, X;\ \Theta^{E})$. Qui il decoder è un prodotto esterno: $\mathrm{DEC}(Z;\ \Theta^{D}) = ZZ^T$. Il termine di regolarizzazione del grafo è l'entropia incrociata sigmoidea (*sigmoid cross entropy*) tra le adiacenze vere ed i punteggi di somiglianza degli edges previsti:

$$\mathcal{L}_{G,\mathrm{REG}}(W, \widehat{W};\ \Theta) = -\left(\sum_{ij} (1 - W_{ij}) \log\left(1 - \sigma(\widehat{W}_{ij})\right) + W_{ij} \log \sigma(\widehat{W}_{ij}) \right).$$

  Il calcolo del termine di regolarizzazione su tutte le possibili coppie di nodi è computazionalmente impegnativo nella pratica, ed il modello **Graph Auto Encoders**

(GAE) utilizza il campionamento negativo per superare questa sfida. Anche se il GAE è un modello deterministico, gli autori hanno introdotto anche VGAE, in cui utilizzano auto-encoders variazionali per codificare e decodificare la struttura del grafo. In VGAE, l'incorporazione Z è modellata come una variabile latente con un *prior* normale multivariato standard $p(Z) = \mathcal{N}(Z|0, I)$ e la rete d'inferenza ammortizzata $q_\Phi(Z|W, X)$ è anch'essa una **Graph Convolutional Network** (GCN), o **rete convuluzionale a grafo**. La VGAE viene ottimizzata minimizzando il corrispondente limite inferiore di evidenza negativa [4]:

$$\begin{aligned}\mathrm{NELBO}(W, X;\ \Theta) &= -\mathbb{E}_{q_\Phi(Z|W,X)}[\log p(W|Z)] + \mathrm{KL}(q_\Phi(Z|W, X)\|p(Z)) \\ &= \mathcal{L}_{G,\mathrm{REG}}\big(W, \widehat{W};\ \Theta\big) + \mathrm{KL}(q_\Phi(Z|W, X)\|p(Z)).\end{aligned}$$

- **Iterative generative modelling of graphs** (Graphite): la modellazione generativa iterativa dei grafi estende GAE e VGAE introducendo un decoder più complesso, che itera tra funzioni di decodifica a coppie e convoluzioni di grafi. Formalmente, il decoder a grafite ripete la seguente iterazione:

$$\widehat{W}^{(k)} = \frac{Z^{(k)}Z^{(k)^{\mathrm{T}}}}{\|Z^{(k)}\|_2^2} + \frac{11^{\mathrm{T}}}{|V|}$$
$$Z^{(k+1)} = GCN\big(\widehat{W}^{(k)}, Z^{(k)}\big)$$

  dove $Z^{(0)}$ è inizializzato utilizzando l'output della rete di codifica. Utilizzando questo processo di decodifica iterativa parametrica, Graphite apprende decoder più espressivi rispetto ad altri metodi basati sulla decodifica non parametrica a coppie. Infine, analogamente a GAE, Graphite può essere deterministico o variazionale [4].

- **Deep Graph Infomax** (DGI): è un metodo d'incorporazione a livello di grafo non supervisionato. Dati uno o più grafi *reali* (positivi), ciascuno con la sua matrice di adiacenza $W \in \mathbb{R}^{|V|\times|V|}$ e le caratteristiche dei nodi $X \in \mathbb{R}^{|V|\times d_0}$, questo metodo crea matrici di adiacenza *false* (negative) $W^- \in \mathbb{R}^{|V^-|\times|V^-|}$ e le loro caratteristiche $X^- \in \mathbb{R}^{|V^-|\times d_0}$. Il metodo addestra (i) un encoder che elabora sia campioni reali che falsi, dando rispettivamente $Z = \mathrm{ENC}(X, W;\ \Theta^E) \in \mathbb{R}^{|V|\times d}$ e $Z^- = \mathrm{ENC}(X^-, W^-;\ \Theta^E) \in \mathbb{R}^{|V|\times d}$, (ii) una funzione (*readout*) di pooling del grafo $\mathcal{R} : \mathbb{R}^{|V|\times d} \to \mathbb{R}^d$, e (iii) una funzione di discriminazione $\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ che viene addestrata per produrre

$\mathcal{D}\big(Z_i, \mathcal{R}(Z)\big) \approx 1$ e $\mathcal{D}\big(Z_i^-, \mathcal{R}(Z^-)\big) \approx 0$, rispettivamente, per i nodi corrispondenti al dato grafo reale $i \in V$ ed al grafo falso $j \in V^-$. Nella prima previsione, DGI campiona dai grafi reali (positivi). Se dovesse essere fornito un solo grafo allora potrebbero essere campionati alcuni sottografi presi da esso. Nella seconda previsione vengono campionati grafi falsi (negativi). In DGI, i campioni falsi presentano l'adiacenza reale $W^- := W$, mentre le false caratteristiche X^- sono una permutazione casuale per riga delle caratteristiche reali X, ma sono plausibili anche altre strategie di campionamento negativo. L'encoder (ENC) usato nel caso di DGI è una GCN, anche se può essere utilizzata qualsiasi GNN. La funzione *readout* $\mathcal{R}$ riassume un intero grafo (di dimensioni variabili) in un singolo vettore (di dimensioni fisse). In DGI la funzione $\mathcal{R}$ viene utilizzata come una media per riga, ma possono essere usate anche altre tecniche/funzioni di pooling dei grafi, come ad esempio quelle che tengono conto dell'adiacenza, $\mathcal{R} : \mathbb{R}^{|V|\times d} \times \mathbb{R}^{|V|\times|V|} \to \mathbb{R}^d$ [4].

2.2 – Supervised Graph Embedding

Un approccio comune per l'***embedding di reti supervisionate*** consiste nell'utilizzare un metodo di embedding di reti non supervisionato (come quelli descritti nella Sezione 2.1), per mappare prima i nodi in uno spazio vettoriale di embedding e poi utilizzare gli embedding appresi come input per un'altra rete neurale. Tuttavia, un limite importante di questo approccio a due fasi è che le incorporazioni dei nodi non supervisionate potrebbero non preservare importanti proprietà dei grafi (ad esempio, le etichette dei nodi o gli attributi), che potrebbero essere utili per un compito supervisionato a valle. Recentemente sono stati proposti diversi metodi, di seguito brevemente descritti, che combinano queste due fasi, ovvero l'*apprendimento degli incorporamenti* e la *previsione delle etichette dei nodi o dei grafi* [4].

2.2.1 – Metodi d'incorporazione superficiali (supervisionati)

Simili a quelli non supervisionati (Sezione 2.1.1), i **metodi d'incorporazione superficiali supervisionati** usano delle "*embedding look-ups*" (ricerche d'incorporamento) per mappare i nodi in incorporamenti. Tuttavia, mentre l'obiettivo dei corrispettivi metodi non supervisionati è quello di apprendere una buona rappresentazione del grafo, i metodi d'incorporazione superficiali

supervisionati mirano ad ottenere buoni risultati in qualche compito di predizione a valle, come la classificazione di nodi o grafi. I metodi superficiali hanno la particolarità di non utilizzare né le caratteristiche dei nodi X né l'adiacenza W nell'encoder, ma utilizzano l'adiacenza per garantire la coerenza. Essi sono particolarmente utili in contesti trasduttivi, se viene fornito un solo grafo, senza caratteristiche dei nodi, una frazione dei nodi è etichettata e l'obiettivo è recuperare le etichette per i nodi non etichettati [4].

- **Label Propagation** (LP): è un algoritmo molto popolare per la classificazione semi-supervisionata dei nodi basata su grafi. Con essa, le incorporazioni vengono apprese direttamente nello spazio delle etichette, cioè la funzione di decodifica supervisionata in LP è semplicemente la funzione identità:

$$\hat{y}^N = \mathrm{DEC}(Z;\ \Theta^C) = Z.$$

In particolare, LP sfrutta la struttura del grafo per smussare la distribuzione delle etichette sul grafo aggiungendo un termine di regolarizzazione alla funzione di perdita, dove l'ipotesi di fondo è che i nodi vicini debbano avere etichette simili (cioè che esista una certa coerenza di etichetta tra i nodi connessi). La regolarizzazione in LP è calcolata con gli autogeni laplaciani:

$$\mathcal{L}_{G,\mathrm{REG}}\big(W, \widehat{W};\ \Theta\big) = \sum_{ij} W_{ij} \left\| \frac{\hat{y}_i^N}{\sqrt{D_i}} - \frac{\hat{y}_j^N}{\sqrt{D_j}} \right\|_2^2,$$

dove $D_i = \sum_j W_{ij}$ è il grado del nodo v_i. La perdita supervisionata nella diffusione delle etichette è semplicemente la somma delle distanze tra le etichette previste e le etichette di verità (*ground truth labels*):

$$\mathcal{L}_{\mathrm{SUP}}^N(y^N, \hat{y}^N;\ \Theta) = \sum_{i|v_i \in V_L} \left\| y_i^N - \hat{y}_i^N \right\|_2^2.$$

Si noti che la perdita supervisionata è calcolata sui nodi etichettati, mentre il termine di regolarizzazione è calcolato su tutti i nodi del grafo. Si prevede che questi metodi funzionino bene con grafi coerenti, cioè grafi in cui la vicinanza dei nodi nel grafo è correlata positivamente con la somiglianza delle etichette [4].

2.2.2 – Metodi di regolarizzazione dei grafi

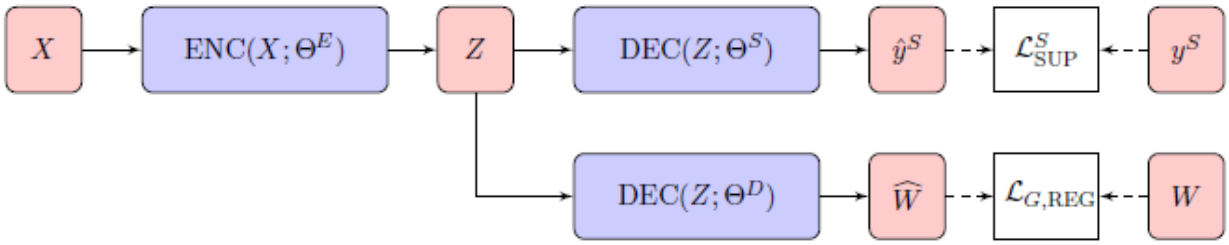

Figura 2.2.2.1: schema dei metodi supervisionati di regolarizzazione dei grafi. La struttura a grafo non viene utilizzata nelle reti di codifica e di decodifica. Agisce invece come regolarizzatore nella funzione di perdita

Anche i **metodi supervisionati di regolarizzazione dei grafi** mirano ad imparare a prevedere le proprietà dei grafi, come le etichette dei nodi. Similmente agli incorporamenti superficiali, questi metodi calcolano una perdita di regolarizzazione del grafo definita sulla struttura dello stesso e una perdita supervisionata per il compito a valle (Figura 2.2.2.1). Tuttavia, la principale differenza con i metodi superficiali risiede nella rete di codifica: anziché utilizzare le embedding look-ups, i metodi di regolarizzazione dei grafi apprendono gli incorporamenti come funzioni parametriche dettate dalle caratteristiche dei nodi, che potrebbero catturare informazioni preziose per le applicazioni finali. In altre parole, le funzioni di codifica in questi metodi possono essere scritte come:

$$Z = \mathrm{ENC}(X;\ \Theta^E).$$

I metodi di regolarizzazione dei grafi, quindi, utilizzano le caratteristiche dei nodi X ma non l'adiacenza W nell'encoder. Generalmente, questi metodi sono induttivi, infatti hanno bisogno solo di un grafo al momento dell'addestramento, ma non al momento dell'inferenza (test). Possono essere applicati quando le caratteristiche dei nodi contengono una ricca informazione [4].

2.2.3 – Graph convolution framework

In questa sezione ci si concentra sui metodi (semi)supervisionati di aggregazione degli elementi vicini, in cui l'encoder utilizza le caratteristiche di input e la struttura del grafo per calcolare gli incorporamenti:

$$Z = \mathrm{ENC}(X, W;\ \Theta^E).$$

In particolare, esaminiamo il modello di rete neurale a grafo (*Graph Neural Network model*) – che è stato il primo tentativo di utilizzare tecniche di deep learning su dati strutturati a grafo – e altri framework correlati, come le reti a passaggio di messaggi (*Message Passing Neural Networks*) [4].

- **Graph Neural Network model** (GNN): La prima formulazione dei metodi deep learning per i dati strutturati a grafo risale al modello di rete neurale a grafo (GNN) di Gori del 2005. Questa formulazione considera il problema dell'incorporazione supervisionata dei grafi come un meccanismo di diffusione dell'informazione, in cui i nodi inviano informazioni ai loro vicini finché non viene raggiunto uno stato di equilibrio stabile. Più concretamente, date le incorporazioni dei nodi inizializzate in modo casuale Z^0, viene applicata la seguente ricorsione fino alla convergenza:

  $$Z^{t+1} = \mathrm{ENC}(X, W, Z^t;\ \Theta^E),$$

  Dove i parametri Θ^E vengono utilizzati ad ogni iterazione. Dopo la convergenza $(t = T)$, le incorporazioni dei nodi Z^T vengono usate per prevedere l'output finale, come le etichette dei nodi o dei grafi:

  $$\hat{y}^S = \mathrm{DEC}(X, Z^T;\ \Theta^S).$$

  Questo processo viene ripetuto più volte fino a quando i parametri GNN, Θ^E e Θ^D, vengono appresi tramite *backpropagation*.

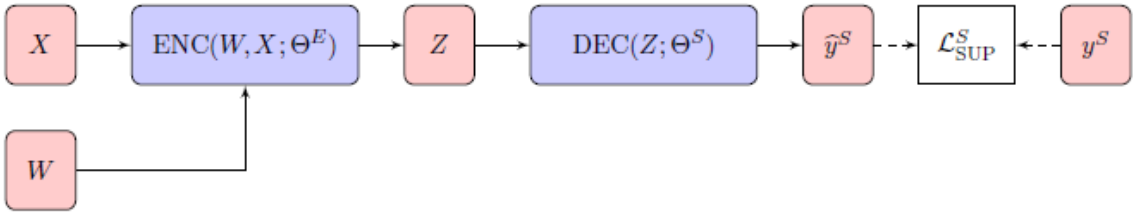

Figura 2.2.3.1: schema reti neurali a grafo (GNNs)

Sostanzialmente, piuttosto che sfruttare la struttura del grafo per agire come regolatore, le GNN sfruttano la struttura del grafo nell'encoder per propagare le informazioni tra i nodi vicini e apprendere le rappresentazioni strutturali. Le etichette vengono poi decodificate e confrontate con le etichette di verità (ad esempio, tramite la cross-entropy loss) [4].

- **Message Passing Neural Networks** (MPNN): Le MPNN provvedono a dare un framework per le GNN, che racchiude molte reti neurali a grafo recenti. A differenza del modello GNN, che funziona per un numero indefinito di iterazioni, le MPNN forniscono un'astrazione per le moderne reti neurali a grafo, che consistono in reti neurali

multistrato con un numero determinato di strati. Ad ogni strato ℓ, le funzioni di messaggio $f^{\ell}(\cdot)$ calcolano i messaggi utilizzando le rappresentazioni nascoste degli elementi vicini, che vengono poi passate alle funzioni di aggregazione $h^{\ell}(\cdot)$:

$$m_i^{\ell+1} = \sum_{j|(v_i,v_j)\in E} f^{\ell}\left(H_i^{\ell}, H_j^{\ell}\right)$$

$$H_i^{\ell+1} = h^{\ell}\left(H_i^{\ell}, m_i^{\ell+1}\right).$$

Dopo l'esecuzione di ℓ strati di passaggio di messaggi, le rappresentazioni nascoste dei nodi codificano le informazioni strutturali all'interno dei vicinati a distanza di ℓ salti [4].

2.2.4 – Spatial Graph Convolution

I **metodi** "***spaziali***" prendono spunto dalle CNN standard, dove le convoluzioni sono applicate nel dominio spaziale, come indicato dalla tipologia del grafo. Ad esempio, nella computer vision, i filtri convoluzionali sono localizzati spazialmente utilizzando patch rettangolari fisse attorno a ciascun pixel. Inoltre, poiché i pixel nelle immagini hanno un ordine naturale (in alto, a sinistra, in basso, a destra) è possibile riutilizzare i pesi dei filtri convoluzionali in ogni posizione, riducendo notevolmente il numero totale di parametri. Sebben le normali convoluzioni spaziali non possano essere applicate direttamente ai domini dei grafi, le convoluzioni spaziali per grafi utilizzano idee come il campionamento per vicinato (*neighborhood sampling*) ed i meccanismi d'attenzione (*attention mechanism*) per superare le sfide poste dalle irregolarità dei grafi [4].

2.2.4.1 – Sampling-based spatial methods

- **Inductive representation learning on large graph** (SAGE): Sebbene le GCN possano essere utilizzate in contesti induttivi, sono state originariamente introdotte per contesti trasduttivi semi-supervisionati ed i filtri appresi potrebbero dipendere fortemente dall'operatore laplaciano utilizzato per l'addestramento. Inoltre, le GCN richiedono la memorizzazione dell'intero grafo, che può essere computazionalmente costosa per grafi di grandi dimensioni. Per superare queste limitazioni è stato proposta SAGE, un framework generale per apprendere incorporamenti induttivi dei nodi riducendo la complessità computazionale delle GCN. Invece di calcolare la media dei segnali di tutti i vicini a distanza di un salto utilizzando le moltiplicazioni con la matrice laplaciana, SAGE campiona dei vicinati fissi (di dimensione q) per rimuovere la forte dipendenza da una

struttura fissa del grafo e generalizzare a nuovi grafi. Ad ogni livello di SAGE, i nodi aggregano le informazioni dei nodi campionati dal loro vicinato e la regola di propagazione può essere scritta come:

$$H_{:,i}^{\ell+1} = \sigma\left(\Theta_1^{\ell} H_{:,i}^{\ell} + \Theta_2^{\ell} \mathrm{AGG}\left(\left\{H_{:,j}^{\ell} : j | v_j \in \mathrm{Sample}(\mathcal{N}(v_i), q)\right\}\right)\right),$$

dove $\mathrm{AGG}(\cdot)$ è una funzione di aggregazione, che può essere qualsiasi operatore invariante alla permutazione, come la media (SAGE-mean) o il max-pooling (SAGE-pool) [4].

2.2.4.2 – Attention-based spatial methods

I **meccanismi d'attenzione** sono stati utilizzati con successo nei modelli linguistici e sono particolarmente utili quando si opera su lunghe sequenze di input, consentendo ai modelli d'identificare le parti rilevanti degli input stessi. I modelli per grafi basati sull'attenzione imparano a concentrarsi sugli elementi vicini importanti durante la fase di passaggio dei messaggi. Ciò offre una maggiore flessibilità nelle impostazioni induttive, rispetto ai metodi che si basano su pesi fissi, come le GCN. In generale, i metodi d'attenzione apprendono l'importanza dei vicini utilizzando funzioni parametriche i cui input sono le caratteristiche dei nodi del livello precedente [4].

- **Graph Attention Network** (GAT): è una versione di GCN basata sull'attenzione, che incorpora meccanismi di auto-attenzione durante il calcolo delle patch. Ad ogni livello, GAT osserva il vicinato di ogni nodo ed impara a scegliere selettivamente i nodi che portano alle migliori prestazioni per qualche compito a valle. L'intuizione di alto livello è simile a quella di SAGE e rende GAT adatto a problemi induttivi e trasduttivi. Tuttavia, invece di limitare la fase di convoluzione a vicinati di dimensioni fisse come in SAGE, GAT consente ad ogni nodo di partecipare alla totalità dei suoi vicini ed utilizza l'attenzione per assegnare pesi diversi a nodi diversi in un vicinato. I parametri d'attenzione sono addestrati tramite backpropagation ed il meccanismo di auto-attenzione di GAT è:

  $$g_k\left(H^{\ell}\right) = \mathrm{LeakyReLU}\left(H^{\ell} B^{\mathrm{T}} b_0 \oplus b_1^{\mathrm{T}} B H^{\ell^{\mathrm{T}}}\right),$$

  dove $\oplus$ indica la somma dei vettori riga e colonna con la trasmissione, mentre (b_0, b_1) e B sono, rispettivamente, i vettori dei pesi d'attenzione addestrabili e la matrice dei pesi. I punteggi degli edges vengono poi normalizzati con la funzione *softmax*. In pratica, gli autori delle GAT propongono di utilizzare un'attenzione a più teste (*heads*) e di combinare i segnali

propagati con una concatenazione dell'operatore di media seguito da una funzione di attivazione. GAT può essere implementato in modo efficiente calcolando i punteggi di auto-attenzione in parallelo tra gli edges e calcolando le rappresentazioni in output in parallelo tra i nodi [4].

2.3 – Graph Kernels

Un ulteriore categoria di metodi indicati per effettuare il compito di classificazione dei grafi è quella dei **Graph Kernels**, i quali fino a non molto tempo dominavano tale ambito e rimangono tutt'ora molto utilizzati. Essi permettono ai sistemi basati su kernel, come le macchine vettoriali di supporto, di lavorare con i grafi, senza dover eseguire l'estrazione di caratteristiche a lunghezza fissa e con valore reale per trasformarli in vettori di caratteristiche. In poche parole, l'idea dei Graph Kernels è quella di confrontare due grafi mediante una funzione avente come input i grafi stessi. Intuitivamente, questa ***funzione kernel*** misura la somiglianza tra i grafi per diverse nozioni di similarità. La maggior parte dei Graph Kernels calcola la somiglianza tra due grafi confrontando le loro sottostrutture, le quali possono essere sotto-grafi specifici, "*cammini*" casuali, cicli o percorsi all'interno degli stessi.

In generale, i *metodi kernel* si riferiscono ad algoritmi d'apprendimento automatico che imparano confrontando coppie di punti dati utilizzando particolari misure di somiglianza, ovvero i *kernels* per l'appunto. Per definirli, consideriamo un insieme non vuoto di punti dati $\mathcal{X}$, come $\mathbb{R}^d$ o un insieme finito di grafi, e sia k: $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ una funzione. Allora k è un kernel su $\mathcal{X}$ se esiste uno spazio di Hilbert $\mathcal{H}_k$ (uno spazio vettoriale dotato di un prodotto scalare che definisce una funzione di distanza per la quale lo spazio è uno spazio metrico completo) e una mappa delle caratteristiche ϕ: $\mathcal{X} \to \mathcal{H}_k$ tale che $k(x, y) = \langle \phi(x), \phi(y) \rangle$ per $x, y \in \mathcal{X}$, dove $\langle \cdot, \cdot \rangle$ denota il prodotto interno di $\mathcal{H}_k$. Una tale mappa delle caratteristiche esiste se e solo se k è una funzione semi-definita positiva. Un esempio banale è quello in cui $\mathcal{X} = \mathbb{R}^d$ e $\phi(\boldsymbol{x}) = \boldsymbol{x}$, nel qual caso il kernel è uguale al prodotto dei punti, $k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}$ [19].

Un concetto importante nei metodi kernel è la *matrice di Gram* $\boldsymbol{K}$, definita rispetto ad un insieme finito di punti dati $x_1, \dots, x_m \in \mathcal{X}$. La matrice di Gram di un kernel k ha elementi K_{ij}, per $i, j \in \{0, \dots, m\}$ pari al valore del kernel tra coppie di punti dati, ossia $K_{ij} = k(x_i, x_j)$. Se la matrice di Gram di k è semi-definita positiva per ogni possibile insieme di punti dati, k è un kernel. I metodi kernel hanno la proprietà desiderabile di non basarsi sulla caratterizzazione esplicita della

rappresentazione vettoriale $\phi(x)$ dei punti dati, ma di accedere ai dati solo attraverso la matrice Gram $\boldsymbol{K}$. Il vantaggio di ciò è spesso illustrato utilizzando il kernel della *funzione a base radiale gaussiana* (RBF) su $\mathbb{R}^d$, $d \in \mathbb{N}$, definito come:

$$k_{\text{RBF}}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\right),$$

dove σ è un parametro di larghezza di banda. Lo spazio di Hilbert associato al kernel RBF gaussiano ha dimensione infinita, ma il kernel può essere facilmente calcolato per qualsiasi coppia di punti $(\boldsymbol{x}, \boldsymbol{y})$. I metodi kernel sono stati sviluppati per la maggior parte dei paradigmi di apprendimento automatico, come le già citate ***macchine vettoriali di supporto*** (SVM) per la classificazione, i ***processi gaussiani*** (GP) per la regressione, la ***kernel PCA*** (Principal Component Analisys) ed il ***k-means*** per l'apprendimento non supervisionato ed il clustering, e la ***kernel density estimation*** (KDE) per la stima della densità [19].

I Graph Kernels sono una particolare tipologia di kernel convoluzionali, detti anche **R-convolution kernels**, applicati ai grafi (e non ai nodi, com'è d'uso comune). Gli R-convolution kernels confrontano le decomposizioni di due oggetti strutturati. Assumendo di aver raccolto un vettore di incorporamento dei nodi, possiamo definire il kernel convoluzionale per due grafi G e H come:

$$k_{convolution}(G, H) = \sum_{u \in G, v \in H} k(\vec{u}, \vec{v}),$$

dove k è una funzione kernel (non lineare) tra due vettori, come, ad esempio, il kernel RBF. Nei Graph Kernels, quindi, confrontiamo tutte le coppie di nodi, e poiché un grafo non è ordinato questo significa che dobbiamo confrontare due insiemi. Il kernel è chiamato convoluzionale perché ogni vettore di nodo di un grafo viene confrontato con ciascun vettore dell'altro grafo (proprio come nelle CNN in cui confrontiamo ogni filtro con tutte le parti presenti nell'immagine d'input). Inoltre, usiamo una funzione kernel (non lineare) per il confronto di due vettori al fine di catturare le dipendenze più complesse tra i nodi. La scelta della funzione kernel può assegnare diversi significati alla somiglianza (ad esempio, i giusti parametri del kernel RBF possono costringerci a considerare solo coppie di vettori in cui i vettori sono quasi identici tra loro) [12]. L'inconveniente di quanto detto è la complessità computazionale quadratica. Per due grafi, aventi n nodi, dobbiamo valutare la funzione kernel n^2 volte. Inoltre, per una raccolta di m grafi nel dataset usato per l'addestramento, abbiamo bisogno di una matrice di Gram composta da m^2 coppie della funzione kernel [12].

Esistono diverse tipologie di metodi per l'ottenimento dei Graph Kernels, basati sulle caratteristiche principali su cui ci si può concentrare per il confronto tra grafi, ovvero *isomorfismo dei grafi*, *distanza di modifica dei grafi* e *descrittori topologici* [20]. Tra questi metodi, quelli più usati sono:

- ***k*-walk kernel**: a partire da ogni nodo, si inizia un numero di camminate casuali di lunghezza esattamente k attraverso i nodi dei grafi. Il vettore del nodo di partenza è quindi la distribuzione delle etichette in tutte le camminate effettuate. Nel caso di grafi senza etichetta è comune utilizzare il grado del nodo al suo posto.
- **Shortest path kernel**: per ogni nodo, si registra il percorso più breve per raggiungere ogni altro nodo nel grafo. Quindi, in modo simile al k-walk kernel, si creano vettori che descrivono la distribuzione dell'etichetta nei percorsi raccolti.
- **Weisfeiler-Lehman kernel**: per ogni nodo del grafo, si raccolgono le etichette dai nodi nel suo intorno per poi ordinarle. Dopo di che viene creata una nuova etichetta dalla stringa risultante. Questo procedimento viene iterato un numero t arbitrario di volte, creando $t + 1$ etichette per ogni nodo. I vettori, quindi, sono costituti dagli ID di queste etichette.

Capitolo 3 – GAT per classificazione non supervisionata dei grafi

Tra i molteplici metodi brevemente descritti sopra, lo studio da noi condotto si concentra in particolare sulle GAT. Anche se accennate in precedenza (Sezione 2.2.4), a seguire viene introdotta una premessa che approfondisce il funzionamento fondamentale delle GAT, al fine di comprendere gli esperimenti condotti successivamente.

Al momento, le GAT sono uno dei tipi più popolari di GNN, generalmente usate in ambito supervisionato per la classificazione dei nodi. Questo perché, mentre nelle **Graph *Convolutional* Networks** (GCN) ogni vicino ha la stessa importanza, nelle **Graph *Attention* Networks** (GAT) viene considerata l'importanza di ciascun vicino mediante un meccanismo d'attenzione che assegna un **fattore di ponderazione ad ogni connessione** [10].

Introdotto da *Veličković et al.* nel 2017 [5], l'auto-attenzione nelle GNN si basa sull'idea che **i nodi non dovrebbero avere tutti la stessa importanza**. Si parla di *auto*-attenzione (e non solo d'attenzione) perché gli input vengono confrontati tra di loro. Questo meccanismo assegna un **fattore di ponderazione α_{ij}** (o *punteggio d'attenzione*) alla connessione tra i nodi i e j [10].

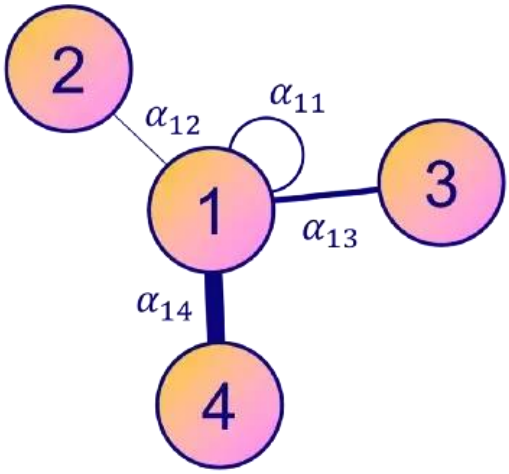

Figura 3.1: esempio di grafo in cui è applicata l'auto-attenzione

Quindi, ad esempio, supponendo una situazione come quella in Figura 3.1, potremo calcolare l'incorporamento del nodo 1 come:

$$h_1 = \alpha_{11}\mathbf{W}x_1 + \alpha_{12}\mathbf{W}x_2 + \alpha_{13}\mathbf{W}x_3 + \alpha_{14}\mathbf{W}x_4,$$

dove $\mathbf{W}$ è una matrice di peso condivisa. Il problema rimanente è quello di calcolare i punteggi d'attenzione, e per risolverlo è possibile costruire una rete neurale GAT sulla base di tre passaggi fondamentali:

- **Trasformazione lineare**: per poter calcolare l'**importanza di ogni connessione** sono necessarie delle coppie di vettori nascosti. Un modo semplice per creare queste coppie è concatenare i vettori di entrambi i nodi. Solo allora è possibile applicare una nuova ***trasformazione lineare*** con una matrice dei pesi $\mathbf{W_{att}}$:

$$a_{ij} = W_{att}^{\mathrm{T}}\left[\mathbf{W}x_i \,\|\, \mathbf{W}x_j\right].$$

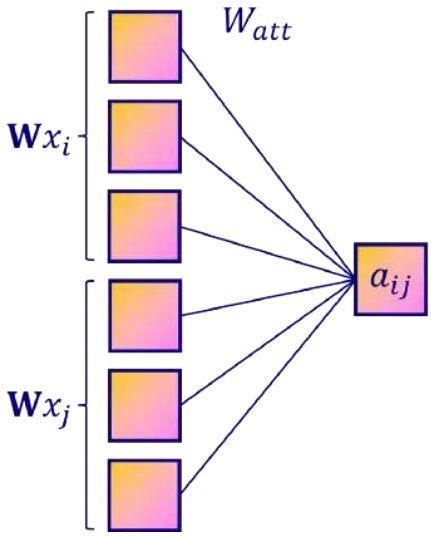

Figura 3.2: concatenazione delle coppie di vettori

- **Funzione di attivazione**: dovendo costruire una rete neurale, il secondo passo da eseguire è quello di aggiungere una funzione di attivazione. Nel caso delle GAT viene generalmente impiegata la funzione *LeakyRelu*.

$$e_{ij} = LeakyReLU\left(a_{ij}\right).$$

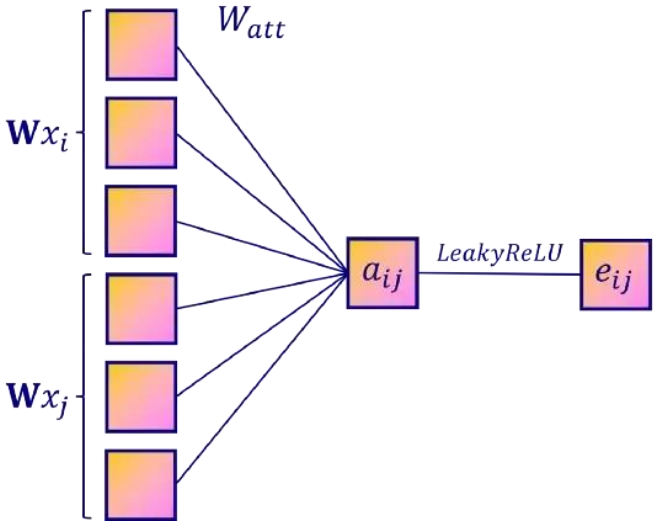

Figura 3.3: applicazione funzione di attivazione

- **Normalizzazione softmax**: per poter confrontare i vari punteggi d'attenzione tra di loro è necessario normalizzare l'output della rete. Ovvero, per poter affermare che il nodo j è più importante del nodo k per il nodo i ($\alpha_{ij} > \alpha_{ik}$), è necessario condividere la stessa scala. Un modo comune per farlo nelle reti neurali è utilizzare la ***funzione softmax***, qui applicata ad ogni nodo vicino:

$$\alpha_{ij} = softmax_i(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

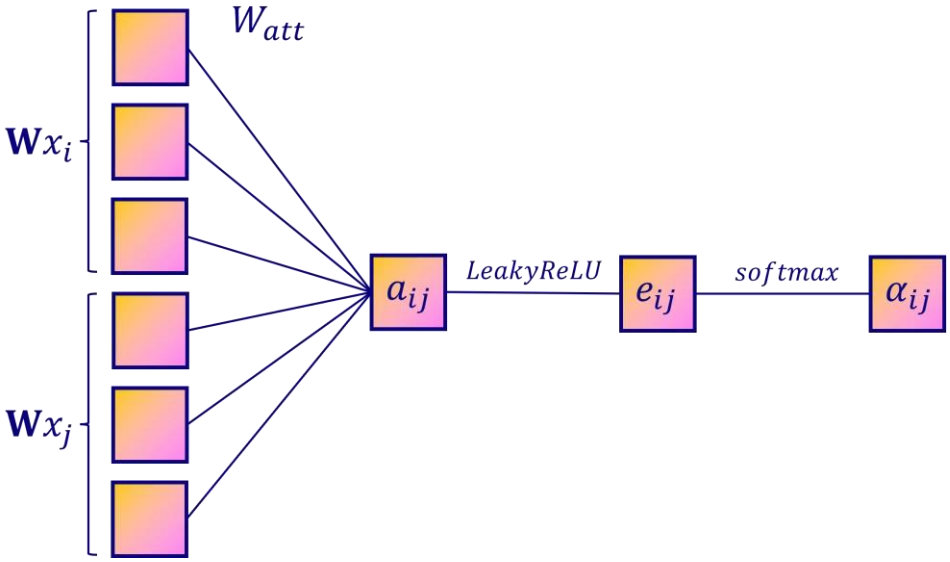

Figura 3.4: applicazione funzione di normalizzazione (softmax)

In questo modo è possibile calcolare ogni fattore α_{ij}. Tuttavia, l'auto-attenzione non è molto stabile e per questo motivo *Vaswani et al.* [6] ha introdotto la *multi-head attention*.

- **Bonus – multi-head attention**: nelle GAT, la multi-head attention consiste nel ***replicare più volte gli stessi tre passaggi*** per fare la media o concatenare i risultati.
  Invece di un singolo incorporamento per il nodo i-esimo h_i, otteniamo un vettore nascosto h_i^k per ogni *attention head*. È quindi possibile applicare uno dei due schemi seguenti:
  - **Media**: sommiamo i diversi vettori h_i^k e normalizziamo il risultato per il numero di teste di attenzione:

$$h_i = \frac{1}{n} \sum_{k=1}^{n} h_i^k$$

- **Concatenazione**: concateniamo i diversi vettori h_i^k:

$$h_i = ||_{k=1}^{n} h_i^k$$

Si usa lo **schema di concatenazione** per gli *strati nascosti* e lo **schema di media** per l'*ultimo strato* della rete.

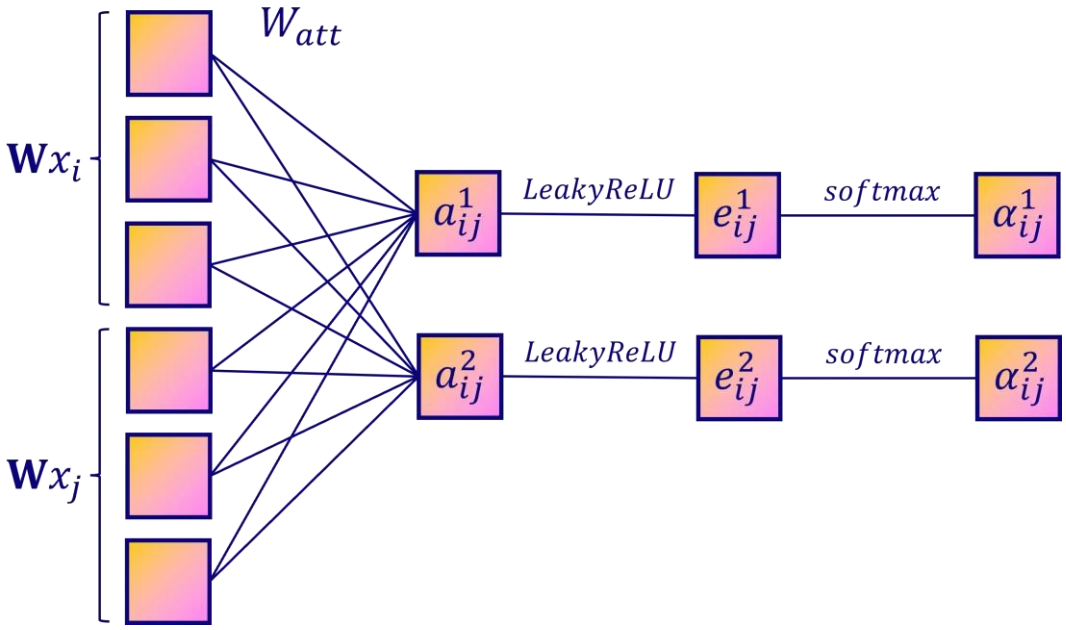

Figura 3.5: multi-head attention

Nonostante la naturale predisposizione delle GAT per la classificazione *supervisionata* dei **nodi**, il nostro studio ha lo scopo di valutare i risultati ottenuti mediante l'uso di un modello GAT in ambito *non supervisionato* per il compito di *classificazione* dei **grafi**.

3.1 – Descrizione strumenti utilizzati

3.1.1 – Modello di riferimento: InfoGraph

Per l'implementazione del nostro modello di rete neurale basato su GAT e la valutazione dei risultati da esso ottenuti si è fatto riferimento al metodo ***InfoGraph*** [8], sviluppato per l'apprendimento non supervisionato/semi-supervisionato di rappresentazioni a livello di grafo ispirato al metodo **Deep Graph Infomax** (DGI, Sezione 2.1.3), riprendendone parzialmente la struttura e confrontandone i valori d'accuratezza sui medesimi dataset testati (*MUTAG*, *PTC-MR*).

Per il problema d'apprendimento non supervisionato di rappresentazioni dei grafi, dato un insieme di grafi $\mathbb{G} = \{G_1, G_2, \dots\}$ ed un intero positivo δ (pari alla dimensione attesa dell'incorporamento), l'obiettivo è imparare una rappresentazione distribuita δ-dimensionale di ogni grafo $G_i \in \mathbb{G}$. Per fare ciò, InfoGraph si concentra sul funzionamento delle reti neurali a grafo (GNN, Sezione 2.1.3) poiché è una classe flessibile di architetture incorporanti che generano rappresentazioni a livello di nodo mediante l'aggregazione ripetuta dei nodi nelle vicinanze. Tali rappresentazioni così ottenute vengono successivamente riassunte, mediante una funzione di ***readout***, in un'unica rappresentazione a livello di grafo avente lunghezza fissa. Formalmente quindi, il k-esimo strato di una GNN è:

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, \text{AGGREGATE}^{(k)}\left(\left\{\left(h_v^{(k-1)}, h_u^{(k-1)}, e_{uv}\right) : u \in \mathcal{N}(v)\right\}\right)\right),$$

dove $h_v^{(k)}$ è il vettore delle caratteristiche del nodo v alla k-esima iterazione/strato, e_{uv} è il vettore delle caratteristiche dell'edge tra u e v, e $\mathcal{N}(v)$ è l'insieme dei vicini al nodo v. Come detto in precedenza, la funzione di *readout* può essere una semplice funzione invariante alla permutazione, come la media o una funzione di pooling a livello di grafo più sofisticata [8].

InfoGraph cerca di ottenere le rappresentazioni dei grafi massimizzando la mutua informazione tra le rappresentazioni a livello di grafo e quelle a livello di nodo corrispondenti. In questo modo le rappresentazioni dei grafi possono imparare a codificare aspetti dei dati che sono condivisi da tutte le sottostrutture. Supponiamo, quindi, di avere un set di grafi $\mathbf{G} := \{G_j \in \mathbb{G}\}_{j=1}^{N}$ con una distribuzione di probabilità empirica $\mathbb{P}$ sullo spazio degli input. Sia ϕ l'insieme dei parametri di una GNN a K strati. Dopo i primi k strati della GNN, il grafo d'ingresso sarà codificato in un insieme di rappresentazioni a livello di nodo $\left\{h_i^{(k)}\right\}_{i=1}^{N}$. Successivamente, si riassumono i vettori di caratteristiche da tutte le profondità della GNN in un unico vettore delle caratteristiche, che cattura le informazioni sulle rappresentazioni di nodo di diverse scale dimensionali centrate su ogni nodo. Per farlo si usa la concatenazione, ovvero:

$$h_\phi^i = \text{CONCAT}\left(\left\{h_i^{(k)}\right\}_{k=1}^{K}\right)$$
$$H_\phi(G) = \text{READOUT}\left(\left\{h_\phi^i\right\}_{i=1}^{N}\right),$$

dove h_ϕ^i è la rappresentazione riassuntiva a livello di nodo, centrata sul nodo i, e $H_\phi(G)$ è la rappresentazione a livello di grafo dopo l'applicazione della funzione di *readout*. Infine, si definisce

uno stimatore della mutua informazione (MI: *Mutual Information*) sulle coppie di rappresentazioni a livello di nodo/grafo, massimizzando la MI stimata sul dataset dato $\mathbf{G} := \{G_j \in \mathbb{G}\}_{j=1}^{N}$:

$$\hat{\phi}, \hat{\psi} = \arg\max_{\phi,\psi} \sum_{G \in \mathbf{G}} \frac{1}{|G|} \sum_{u \in G} I_{\phi,\psi}\left(\vec{h}_{\phi}^{u};\ H_{\phi}(G)\right).$$

$I_{\phi,\psi}$ è lo stimatore della mutua informazione modellato dal discriminatore $T_{\psi|}$ e parametrizzato da una rete neurali con parametri ψ. In InfoGraph viene utilizzato lo stimatore Jensen-Shennon, definito come:

$$I_{\phi,\psi}\left(h_{|\phi}^{i}(G);\ H_{\phi}(G)\right) := \mathbb{E}_{\mathbb{P}}[-\mathrm{sp}(-T_{\phi,\psi}(\vec{h}_{\phi}^{i}(x), H_{\phi}(x)))] - \mathbb{E}_{\mathbb{P}\times\tilde{\mathbb{P}}}[\mathrm{sp}(T_{\phi,\psi}(\vec{h}_{\phi}^{i}(x), H_{\phi}(x)))]$$

dove x è un campione di input (*campione positivo*), x' (*campione negativo*) è un input campionato da $\mathbb{P} = \tilde{\mathbb{P}}$, una distribuzione identica alla distribuzione di probabilità empirica dello spazio di input, e $\mathrm{sp}(z) = \log(1 + e^{z})$ è la funzione *softplus*. In pratica, si generano campioni negativi utilizzando tutte le possibili combinazioni di rappresentazioni a livello di nodo e rappresentazioni a livello di grafo per tutti i grafi presenti in un batch [8].

Poiché $H_{\phi}(G)$ è incoraggiato ad avere un elevata MI con le rappresentazioni a livello di nodo che contengono informazioni su tutte le scale dimensionali, ciò favorisce la codifica degli aspetti dei dati che sono condivisi tra le rappresentazioni di nodo e gli aspetti che sono condivisi tra le scale.

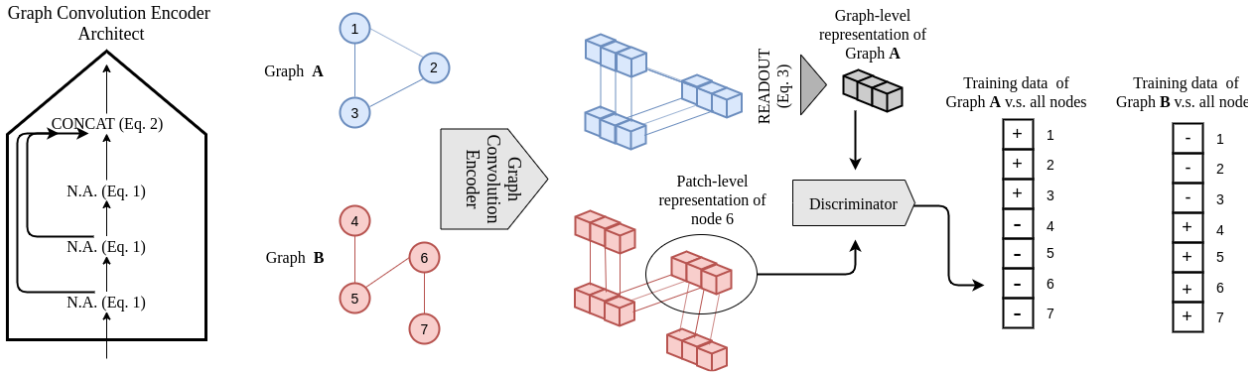

Figura 3.1.1.1: Algoritmo InfoGraph – N.A. (Neighborhood Aggregation) indica l'aggregazione dei nodi vicini. Un grafo in ingresso viene codificato in una mappa di caratteristiche mediante convoluzioni a grafo e concatenazione di salti. Il discriminatore prende in input una coppia di rappresentazioni livello nodo/grafo e decide se provengono dallo stesso grafo.

Seppur, come abbiamo detto, InfoGraph è simile a DGI, vi sono due importanti differenze dovute ai diversi problemi su cui si concentrano. In primo luogo, in DGI viene utilizzato un campionamento casuale per ottenere campioni negativi, dato che si è focalizzati sull'apprendimento degli incorporamenti dei nodi di un grafo. Tuttavia, i metodi contrastivi richiedono un gran numero di

campioni negativi per essere competitivi, quindi l'uso di una generazione orientata ai batch di campioni negativi è cruciale nel nostro caso, in quanto stiamo cercando di apprendere incorporamenti di grafi con molte istanze di grafi. In secondo luogo, come encoders convoluzionali InfoGraph fa uso dei **GIN** (*Graph Isomorphism Networks*), mentre DGI utilizza **GCN** (Sezione 2.1.3), poiché GIN fornisce un bias induttivo migliore per le applicazioni a livello di grafo [8].

3.1.2 – Modello proposto

La realizzazione del modello è stata possibile grazie all'uso della libreria **PyG** (*PyTorch Geometric*), una libreria basata su **PyTorch** per scrivere ed addestrare facilmente le GNN per un'ampia gamma di applicazioni relative ai dati strutturati, la quale fornisce una grande varietà di strumenti utili, tra cui: mini-batch loader di facile utilizzo per operare su di un numero elevato di piccoli grafi od un singolo grafo di grandi dimensioni, supporto multi-GPU, un gran numero di dataset di riferimento ed una vasta gamma di strati (o *layers*), convoluzionali e non, pronti all'uso che implementano il funzionamento delle GNN più affermate [9]. Tra quest'ultimi vi è il layer principalmente usato da noi, ovvero ***GATConv***, il quale riassume i passaggi fondamentali delle GAT precedentemente discussi (Sezione 3) e permette l'implementazione semplificata di un modello *GAT-based*.

Come accennato in precedenza, il modello GAT da noi costruito emula InfoGraph riadattando i suoi elementi chiave per l'uso delle GAT. In particolare, facciamo uso di:

- **Encoder GAT**: un encoder per l'ottenimento di coppie di rappresentazioni a livello di nodo/grafo. Per raggiungere la coppia di rappresentazioni ricercata, l'encoder esegue una serie di operazioni:
    - Viene eseguita la funzione di dropout, la quale durante l'addestramento azzera in modo casuale alcuni degli elementi del grafo in input con una certa probabilità p utilizzando dei campioni provenienti dalla distribuzione di Bernoulli. Ogni canale dei vari layers, quindi, sarà azzerato in maniera indipendente dagli altri. Questa è una tecnica affermata ed efficace per la regolarizzazione, che impedisce il co-adattamento dei neuroni artificiali della rete.
    - Vengono utilizzati (in numero variabile) dei layers GATConv, che sfruttano il meccanismo aggiuntivo della *multi-head attention* sui dati in ingresso, in modo tale da ottenere fattori di ponderazione tra i nodi più raffinati e distinguere maggiormente i nodi più importanti nell'intorno di ciascuno di essi.

- Per garantire la non linearità del sistema, i vari output di ciascun layer vengono sottoposti alla funzione di attivazione *LeakyReLU* (la quale è generalmente indicata per le GAT).
- All'ingresso di ogni *hidden layer* è predisposta una funzione di Batch Normalization, in modo tale da ottenere una distribuzione normalizzata degli input di ciascun layer nel tentativo di incrementare il più possibile le prestazioni del modello, sia in termini di velocità d'elaborazione dei dati in ingresso che di accuratezza delle previsioni.

Questi tre passaggi permettono di ottenere le singole rappresentazioni di ciascun nodo, le quali vengono poi concatenate per ottenere un unico elemento matematicamente utilizzabile. Invece, la rappresentazione a livello di grafo viene conseguita da una funzione *readout* di pooling delle rappresentazioni a livello di nodo, nello specifico la funzione di somma, la quale empiricamente risulta funzionare maggiormente.

- **Rete feed-forward**: le coppie di rappresentazioni nodi/grafo fornite dall'encoder subiscono delle ulteriori trasformazioni non lineari, da reti neurali *feed-forward* a tre strati con connessioni a salto, per la parametrizzazione delle trasformazioni precedentemente eseguite su entrambe le rappresentazioni della coppia corrente.
- **Funzione di perdita** (*loss function*): la loss function utilizzata è la stessa adoperata dallo stimatore della mutua informazione di InfoGraph $I_{\phi,\psi}$ (Sezione 3.1.1), con cui viene calcolata la differenza tra campioni positivi e campioni negativi.

3.1.3 – Dataset

A causa del recente crescente interesse per il *Graph Representation Learning*, si è avuto un incremento della richiesta di dataset di prova significativi per consentire delle valutazioni standardizzate delle reti neurali sviluppate nel settore. A tal proposito, *Christopher Morris et al.* [13] hanno introdotto **TUDataset**, una collezione di dataset di riferimento per la classificazione e la regressione dei grafi, costituita da oltre 120 dataset di varie dimensioni per una vasta gamma di applicazioni. Tutti i dataset della collezione sono facilmente accessibili da framework per l'apprendimento su grafi popolari come **PyTorch Geometric**. Poiché questa collezione comprende grafi di vari domini, forniti da diversi autori, essi differiscono per quanto riguarda il modello di grafo

utilizzato anche all'interno dello stesso dominio e le annotazioni fornite, ad esempio gli attributi discreti e continui di nodi ed edges [13].

Una categoria comune e molto utilizzata di dataset presenti in questa collezione è quella delle *piccole molecole* (**Small molecules**), con etichette di classe che rappresentano, ad esempio, la tossicità o l'attività biologica determinata in progetti di ricerca di farmaci. In questo caso un grafo rappresenta una molecola, ovvero i nodi prendono il posto degli atomi e gli edges quello dei legami chimici. Di conseguenza le etichette codificano i tipi di atomo e di legame, eventualmente con attributi chimici aggiuntivi. Qui i modelli dei grafi differiscono, ad esempio, per il fatto che gli atomi di idrogeno sono rappresentati esplicitamente dai nodi ed i legami negli anelli aromatici sono annotati di conseguenza [13]. Di questa categoria fanno parte due dei principali dataset utilizzati per la sperimentazione sia da noi che dagli autori di InfoGraph, ovvero *MUTAG* e *PTC* (MR).

Nello specifico, **MUTAG** è una raccolta di composti nitroaromatici in cui l'obiettivo è prevedere la loro mutagenicità sulla Salmonella typhimurium. I grafi d'input vengono utilizzati per rappresentare i composti chimici, dove i vertici rappresentano gli atomi e sono etichettati dal tipo di atomo (rappresentato dalla codifica *one-hot*), mentre i collegamenti tra i vertici rappresentano i legami tra gli atomi corrispondenti. Tale dataset include 188 campioni di composti con 7 caratteristiche di nodi discreti e 2 classi di appartenenza per ogni grafo [14]. Invece, **PTC** (*Predictive Toxicology Challenge*) è una collezione di 344 composti chimici rappresentati come grafi che riportano la cancerogenicità per i ratti, in cui vi sono 19 caratteristiche per ogni nodo ed anche qui 2 classi di appartenenza per ogni grafo [15]. Lo svantaggio nell'utilizzo di questi dataset, in particolare per la classificazione dei grafi, e che essi sono "sbilanciati", ovvero vi è una classe avente un numero più alto di grafi che le appartengono rispetto all'altra.

Un'altra categoria di TUDataset usata di frequente è quella rappresentante i **Social Networks**, in cui rientrano altre due sotto-collezioni di dataset adoperate durante il nostro studio e dal modello di InfoGraph per la valutazione dei risultati, cioè *REDDIT* e *IMDB*. Nei dataset **REDDIT**, realizzati dai post del social Reddit nel mese di settembre 2014, ogni grafo rappresenta un thread di discussione, dove i nodi corrispondono agli utenti, in cui due utenti sono collegati tra loro da un edge se uno ha risposto ad un commento dell'altro. L'etichetta del nodo in questo caso è la comunità, o "*subreddit*", a cui appartiene un post. Questo modello di grafo viene utilizzato per ricavare diversi insiemi di dati, in cui il compito di classificazione consiste nel distinguere i subreddit basati su discussioni da quelli basati su domande/risposte (**REDDIT-BINARY**) o nel prevedere il subreddit in cui è stato pubblicato

il thread (**REDDIT-MULTI-5K** e **REDDIT-MULTI-12K**) [13]. **IMDB**, invece, è un dataset di collaborazione cinematografica costituito dalle ego-networks di 1000 attori/attrici che hanno interpretato ruoli nei film presenti in IMDb. In ogni grafo, i nodi rappresentano attori/attrici e vi è un edge di collegamento tra di loro se compaiono nello stesso film. Questi grafi derivano dai collegamenti ottenuti per i generi d'azione e romanticismo (**IMDB-BINARY**) oppure per i generi di commedia, romanticismo e fantascienza (**IMDB-MULTI**) [16]. Al contrario di MUTAG e PTC, REDDIT ed IMDB hanno dataset perfettamente bilanciati per le varie classe di appartenenza, tuttavia sono dataset statici in cui i nodi non presentano nessuna caratteristica che possa essere utilizzata in una rete neurale.

3.2 – Confronto tra GIN e GAT

Le **Graph Isomorphism Networks** (GIN), sviluppate da *Xu et al.* [10], si concentrano sull'identità topologica per misurare la somiglianza tra grafi. Esse sono una tipologia di GNN che cercano di raggiungere il massimo potere discriminativo generalizzando il test di isomorfismo dei grafi **Weisfeiler-Lehman** (WL). Due grafi sono considerati "*isomorfi*" se esiste una mappatura tra i nodi dei grafi che preserva le adiacenze dei nodi: connessioni identiche ma una permutazione diversa dei nodi (Figura 3.2.1) [11].

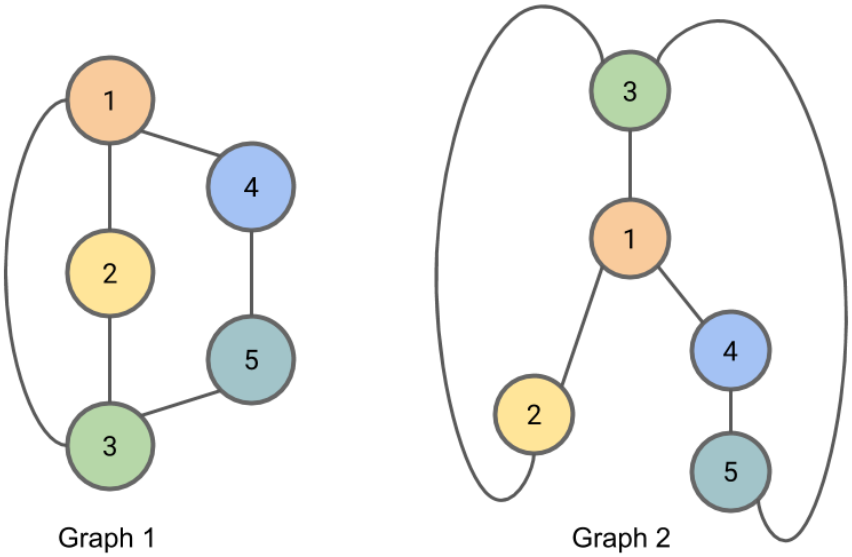

Figura 3.2.1: due grafi isomorfi

In generale, determinare se due grafi sono isomorfi quando la corrispondenza non viene fornita è un problema impegnativo, la cui difficoltà cresce di pari passo con la dimensione dei grafi;

esattamente *quanto* sia difficile questo problema rimane una questione aperta in informatica. Non è noto se esista un algoritmo in tempo polinomiale per determinare se i grafi sono isomorfi, e non è nemmeno noto se il problema sia **NP-completo** [11].

L'algoritmo del test WL produce una ***forma canonica*** per ogni grafo. Se le forme canoniche di due grafi non sono equivalenti, allora i grafi *definitivamente* non sono isomorfi. Tuttavia, è possibile che due grafi non-isomorfi condividano una forma canonica, quindi questo test da solo non può fornire una prova conclusiva che due grafi siano isomorfi. Nello specifico, l'algoritmo del test WL prevede che:

- Per ogni iterazione i dell'algoritmo si assegna una tupla $L_{i,n}$ ad ogni nodo, contenente la vecchia etichetta compressa del nodo ed un multiset di etichette compresse dei vicini del nodo. Per multiset s'intende un insieme in cui si possono avere più istanze di uno stesso elemento.
- Ad ogni iterazione si assegnerà, inoltre, a ciascun nodo n una nuova etichetta "*compressa*" $C_{i,n}$ per il set di etichette di quel nodo. Qualsiasi coppia di nodi con la stessa tupla $L_{i,n}$ oterrà la stessa etichetta compressa.

Quando si utilizza il test WL per determinare l'isomorfismo, esso può essere applicato in parallelo ai due grafi. L'algoritmo può essere terminato subito dopo l'iterazione i-esima se le dimensioni delle partizioni dei nodi partizionati dalle etichette compresse divergono tra i due grafi, poiché in questo caso i grafi non sono isomorfi. Sostanzialmente, l'idea alla base del test WL è quella di trovare per ogni nodo in ogni grafo una "*firma*" basata sull'intorno del nodo. Queste firme possono quindi essere utilizzate per verificare l'isomorfismo. Nell'algoritmo di cui sopra, le "*etichette compresse*" fungono da firme. Poiché più nodi possono avere la stessa etichetta compressa, esistono più possibili corrispondenze suggerite dall'etichettatura WL [11].

Per essere valida, almeno, quanto il test WL, una GIN deve produrre degli incorporamenti dei nodi che siano *diversi* quando si elaborano dei grafi non isomorfi. La soluzione trovata dagli autori della GIN è quella di usare due funzioni multiset iniettive apprendibili mediante un **MLP** (*Multi-Layer Perceptron*). Mentre con le GAT utilizziamo una rete neurale per apprendere i migliori fattori di ponderazione per un determinato compito, con le GIN si apprende un'approssimazione di due funzioni iniettive grazie al **Teorema di Approssimazione Universale**. Quindi GIN aggiorna le rappresentazioni dei nodi come:

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right)$$

dove ϵ determina l'**importanza del nodo target** rispetto ai suoi vicini (se $\epsilon = 0$ esso ha la stessa importanza degli altri) e può essere sia un parametro apprendibile che uno scalare fisso. Si parla di MLP per evidenziare il fatto che esiste più di un livello, poiché, secondo gli autori, uno strato non è sufficiente per l'apprendimento dei grafi in generale. Infine, per produrre un incorporamento del grafo, è necessario un raggruppamento globale degli incorporamenti dei nodi prodotti dalla GIN. Un modo semplice per ottenere l'incorporamento di un grafo consiste nell'usare la **media**, la **somma** o il **massimo** di ogni incorporamento di nodi h_v:

$$\textbf{Media}: \quad h_G = \frac{1}{N}\sum_{i=0}^{N} h_i$$

$$\textbf{Somma}: \quad h_G = \sum_{i=0}^{N} h_i$$

$$\textbf{Massimo}: \quad h_G = max_{i=0}^{N}(h_i).$$

Per poter considerare tutte le informazioni strutturali, è necessario mantenere gli incorporamenti degli strati precedenti. Inoltre, l'operatore somma risulta essere empiricamente più "*espressivo*" della media o del massimo. Ciò porta gli autori a proporre il seguente metodo di raggruppamento globale:

$$h_G = \sum_{i=0}^{N} h_i^{(0)} \| \ldots \| \sum_{i=0}^{N} h_i^{(k)}.$$

Per ogni livello, gli incorporamenti dei nodi vengono sommati ed il risultato viene concatenato. Questa soluzione combina l'espressività dell'operatore di somma con la memoria delle iterazioni precedenti della concatenazione [12].

Come vedremo anche successivamente, GIN è una delle GNN più potenti tra quelle contrastive ed ha permesso al modello InfoGraph di ottenere risultati superiori ad altri metodi molto affermati come i Graph Kernels (Sezione 2.3), di cui il test WL fa parte. In maniera speculare alle GAT, dove viene assegnato un punteggio d'importanza alle connessioni del nodo corrente con i suoi vicini, le GIN valutano l'importanza di un nodo target per i nodi nel suo intorno e considerano l'isomorfismo tra due grafi. Inoltre, rispetto alle GAT in cui vengono eseguite più volte una serie di operazioni

computazionalmente onerose, le GIN eseguono delle operazioni relativamente semplici e rapide. Ciò rende le GIN, rispetto alle GAT, sensibilmente più performanti per compiti di natura induttiva, come la classificazione dei grafi su cui noi abbiamo deciso di operare, soprattutto su dataset in cui la struttura è tanto importante quanto le caratteristiche dei singoli nodi e/o collegamenti (ad esempio, dataset rappresentanti composti chimici/molecolari). Questo viene dimostrato anche attraverso negli esperimenti da noi eseguiti (Sezione …). Come le GAT, anche le GIN possono essere facilmente implementate tramite la libreria PyG, poiché essa mette a disposizione un layer (***GINConv***) comprensivo delle operazioni svolte dalla rete, senza la necessità di implementare una da zero la struttura per le operazioni cardine sopra descritte.

3.3 – Esperimenti

3.3.1 – Configurazione esperimenti

Per il compito di classificazione non supervisionata dei grafi, abbiamo effettuato un confronto equo dei risultati adottando la stessa procedura degli autori di InfoGraph, i quali a loro volta seguono i consigli di valutazione forniti dai realizzatori della collezione di dataset TUDataset. Tale procedura standard per le GNN, consiste nell'ottimizzare il modello di rete in uso utilizzando l'algoritmo *Adam* e calcolare l'accuratezza della classificazione mediante una convalida incrociata a gruppi da 10 (*10-Fold Cross Validation Accuracy*), in cui selezioniamo il 10% di ogni *training fold* in modo uniforme e casuale come set di convalida per ottimizzare gli iperparametri come, ad esempio, numero di iterazioni, numero di layer e dimensione delle features.

I vari test sono stati effettuati provando diverse combinazioni di parametri (learning rate, batch-size, numero layers GAT, heads, ecc.) al fine di raggiungere empiricamente le prestazioni migliori possibili. Tutti i test sono stati eseguiti a parità di 100 epoche. Inoltre, mentre nel modello InfoGraph vi è uno strato per la normalizzazione del batch dopo ogni layer GIN, in determinate configurazioni non prevederemo la **Batch Normalization** (BN), per osservare il cambiamento dei risultati.

Poiché i risultati ottenuti da InfoGraph sono stati raggiunti con una versione antecedente della libreria PyTorch Geometric, sono stati condotti degli esperimenti anche con un modello GIN realizzato integrando il codice presente nella pagina GitHub degli sviluppatori [17] con il nostro. Tali esperimenti sono stati eseguiti usando la configurazione di default di InfoGraph (Tabella 1), al fine

di cercare di replicare i risultati da loro ottenuti. Di seguito vengono mostrate delle tabelle riepilogative delle configurazioni dei modelli.

Tabella 1 – InfoGraph default configuration/GIN Model configuration

InfoGraph/GIN Model				
Num. GIN Layers	Hidden Dim.	Batch size	Learning Rate	Weight Decay
5	32	128	10^{-2}	0

Tabella 2 – GAT Model configurations

GAT Model				
Parameters	Config. 1	Config. 2	Config. 3	Config. 4
Num. GAT Layers	5	2	3	3
Hidden Dim.	32	32	64	64
Heads	8	4	8	8
Dropout	0.6	0.6	0.5	0.5
Batch size	128	128	128	128
Learning Rate	10^{-2}	$5e^{-2}$	10^{-2}	10^{-2}
Weight Decay	0	$5e^{-4}$	0	0
Batch Norm.	✓	×	✓	×

3.3.2 – Risultati

Per il compito di classificazione dei grafi, gli autori di InfoGraph hanno confrontato le prestazioni ottenute dal loro modello non supervisionato con diversi altri metodi noti, sia supervisionati che non (*Graph Kernels*, *node2vec*, *sub2vec*, *graph2vec*), su diversi dataset (*MUTAG*, *PTC-MR*, *REDDIT*, *IMDB*). Con quasi tutti i dataset, InfoGraph ha raggiunto i risultati più elevati. Nella tabella seguente (Tabella 3) vengono, invece, confrontati i valori d'accuratezza ottenuti dalle varie configurazioni del nostro modello GAT (Tabella 2, Sezione 3.3.1) sui dataset su cui noi abbiamo potuto operare con quelli di InfoGraph riportati nell'articolo correlato e quelli del modello GIN integrato ([8]).

Purtroppo, con il modello da noi proposto abbiamo riscontrato dei problemi nell'utilizzo dei dataset di IMDB e REDDIT. Tali dataset si sono dimostrati incompatibili con i layers GATConv da noi utilizzati,

essendo essi dataset statici con nodi privi di caratteristiche. Tale problema di compatibilità è stato risolto tramite una ricomposizione dei batch in cui vengono partizionati i dataset durante l'esecuzione del codice. Tuttavia, pur ottenendo la corretta esecuzione del codice tramite questa soluzione, non sono stati ottenuti dei risultati validi ai fini del nostro studio, poiché il modello non si comportava correttamente mostrando ad ogni epoca gli stessi valori di accuratezza. Non siamo riusciti a determinare se questo comportamento sia dovuto ad un problema della nostra implementazione o all'inadeguatezza delle GAT con i dataset di IMDB e REDDIT. Invece, sia con il modello GIN realizzato da noi, sia con il modello originale di InfoGraph, non è stato possibile effettuare alcuna prova sui suddetti dataset a causa di un errore del codice di origine sconosciuta provocato dall'uso dei layers GINConv in entrambi i modelli, rimasto irrisolto dagli stessi autori di InfoGraph [18]. Per via di questi problemi risultati per IMDB e REDDIT non verranno riportati.

Tabella 3 – Accuracy comparison

Dataset	MUTAG	PTC-MR
(N° Graphs)	188	344
(N° Classes)	2	2
(Avg. Graph Size)	17.93	14.29
InfoGraph	89.01 ± 1.13	61.65 ± 1.43
GAT Model Config. 1	~ 73.37 ± 1.90	~ 60.37 ± 1.84
GAT Model Config. 2	~ 69.45 ± 2.42	~ 58.76 ± 2.38
GAT Model Config. 3	~ 77.25 ± 2.04	~ 61.30 ± 1.37
GAT Model Config. 4	~ 72.31 ± 1.14	~ 57.87 ± 1.55
GIN Model	~ 88.89 ± 1.24	~ 61.64 ± 1.33

- **Risultati modello GIN**: tale test ha lo scopo di mostrare i risultati ottenuti con una versione del nostro modello utilizzatrice dei layers GINConv. In tabella vengono riportati solo i risultati migliori, raggiunti con la stessa configurazione di InfoGraph (Sezione 3.3.1, Tabella 1), ma sono stati anche effettuati test cambiando i valori degli stessi parametri, modificando la funzione di non linearità o rimuovendo una parte della procedura utilizzata (come la BN). Tutti questi test hanno mostrato delle prestazioni uguali o minori a quelle del test condotto con la configurazione di default.

- **Risultati configurazione 1**: in questo test si è verificato il comportamento del nostro modello GAT utilizzando gli stessi valori di default di InfoGraph per i parametri in comune, in modo tale da vedere la differenza dei due modelli più o meno nelle stesse condizioni. Nel caso del nostro modello GAT abbiamo un'accuratezza evidentemente più bassa rispetto ad InfoGraph per entrambi i dataset, in modo sostanziale per MUTAG e sensibile per PTC-MR.
- **Risultati configurazione 2**: nel secondo esperimento abbiamo provato a far variare i parametri del nostro modello incrementando il learning rate, diminuire il numero di heads dei layer GAT, inserendo una weight decay standard e rimuovendo la BN, per vedere come variava la situazione. Come risultato abbiamo ottenuto un ulteriore calo nelle prestazioni. Ciò che ha contribuito maggiormente a questo calo è, probabilmente, la riduzione dei layers GAT e la rimozione della BN, poiché, oltre ad essere meno elaborati gli input, la distribuzione non normalizzata degli input di ciascun layer potrebbe aver influenzato il tasso d'apprendimento. L'inserimento della weight decay, invece, ha come effetto un incremento momentaneo della loss intorno all'epoca 70, per poi essere minimizzata nuovamente nelle epoche successive.
- **Risultati configurazione 3**: il terzo esperimento utilizza la configurazione con i risultati migliori. I probabili principali fattori di ciò sono l'utilizzo di un layer GAT aggiuntivo, rispetto la configurazione 2, e di una dimensione maggiore per i layers nascosti, oltre che l'uso effettivo della BN. Grazie a ciò, abbiamo ottenuto un sensibile incremento rispetto alle configurazioni precedenti, anche se non alla pari dei risultati di InfoGraph.
- **Risultati configurazione 4**: questo esperimento è stato eseguito principalmente per constatare se la rimozione della BN comporti un effettivo abbassamento delle prestazioni e che non si tratti di un caso dovuto ad altre variabili. Ciò è confermato dall'ottenimento di accuratezze più basse utilizzando gli stessi parametri della configurazione 3, la quale, invece, ha dato i valori più elevati tra i vari test.

Capitolo 4 – Conclusioni

In questo elaborato abbiamo effettuato un excursus generale sul Machine Learning, vedendo come tale disciplina si stia espandendo sempre di più con il passare del tempo per trattare in modo nuovo e più efficiente le diverse tipologie di dati. In particolare, ci siamo concentrati su una sua branca specifica, il *Geometric Deep Learning* (GDL), relativamente recente, che si occupa del trattamento di dati grafo-strutturati, categoria di dati molto potente ed espressiva ma allo stesso tempo difficile da trattare poiché consente di rappresentare strutture complesse non-euclidee (come social network, oggetti tridimensionali e composti chimici di varia natura).

Ciò che ci ha interessato maggiormente del GDL sono i diversi metodi esistenti di *Graph Representation Learning* (GRL) per l'apprendimento (supervisionato/non supervisionato) di rappresentazioni di dati strutturati a grafo difficilmente analizzabili con le classiche reti neurali come CNN o RNN. Tra questi metodi, quelli su cui ci siamo concentrati particolarmente sono le *Graph Neural Networks* (GNN), in cui figura il metodo delle *Graph Attention Networks* (GAT) da noi usato per la costruzione di un modello di GNN al fine di condurre uno studio sulle prestazioni ottenibili tramite il loro uso.

Lo studio condotto ha permesso di osservare l'impatto delle GAT sulle prestazioni, costruendo un modello di rete neurale impiegandolo per il compito di classificazione non supervisionata dei grafi e basato sulla struttura di un ulteriore modello avente risultati elevati, ovvero *InfoGraph*, in cui viene fatto uso delle potenti reti contrastive note come *Graph Isomorphism Networks* (GIN). Gli esperimenti condotti hanno mostrato che tale modello GAT non è stato capace né di superare, né di raggiungere i risultati del modello a cui è ispirato, almeno per i dataset testati. Tuttavia, seppure il nostro modello non sia stato in grado di emulare le alte prestazioni di InfoGraph, esso è stato comunque capace di ottenere delle prestazioni paragonabili, ed in alcuni casi superiori, agli altri metodi confrontati dagli autori dello stesso InfoGraph (cioè *node2vec*, *sub2vec*, *graph2vec*). Questo dimostra che le GAT siano comunque un ottimo strumento da utilizzare, sia per compiti trasduttivi che induttivi.

Riferimenti

[1]	intelligenzaartificiale.it
[2]	oracle.com
[3]	flawnsontong.medium.com
[4]	jmlr.org/papers/v23/20-852.html
[5]	arxiv.org/abs/1710.10903
[6]	arxiv.org/abs/1706.03762
[7]	arxiv.org/abs/2105.14491
[8]	openreview.net/forum?id=r1lfF2NYvH
[9]	pytorch-geometric.readthedocs.io
[10]	arxiv.org/abs/1810.00826v3
[11]	davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/
[12]	towardsdatascience.com
[13]	arxiv.org/abs/2007.08663
[14]	arxiv.org/abs/1911.08941
[15]	arxiv.org/abs/1904.01098
[16]	arxiv.org/abs/1811.03508
[17]	github.com/fanyun-sun/InfoGraph
[18]	github.com/fanyun-sun/InfoGraph/issues/6
[19]	arxiv.org/abs/1903.11835
[20]	arxiv.org/abs/2011.03854