**Graph Neural Networks**

Prima di parlare delle GNN dovremmo parlare dell’input per questi modelli: i grafi.

**Cos’è un grafo?**

La parte più importante di una GNN è il grafo.

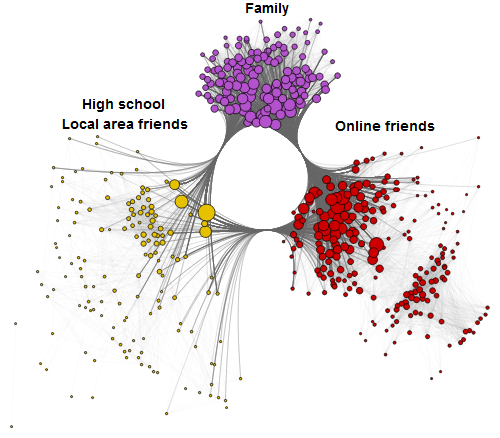
In informatica, un grafo è una struttura dati composta da due componenti: **nodi** (vertici) e **archi.**

Un grafo **G** può essere definito come **G = (V, E)**, dove **V** è l’insieme dei nodi ed **E** sono gli archi tra di loro.

Un grafo può rappresentare cose i social media o le molecole.

Nei social network i nodi del grafo sono gli utenti e gli archi sono le connessioni tra loro.

Un grafo di un social network potrebbe essere simile a questo:



Un grafo è spesso rappresentato da **A**, una matrice di adiacenza.

Se un grafo ha **n** nodi, **A** ha una dimensione di **(n x n).**

Di solito i nodi hanno un insieme di caratteristiche (per esempio, il profilo utente). Se i nodi hanno **f** caratteristiche, la matrice **X** delle caratteristiche dei nodi ha una dimensione di **(n x f).**

**Le Graph Neural Networks operano su dati a grafo**

Prima di tutto, chiariamo perché abbiamo bisogno delle Graph Neural Networks. Le GNN sono in grado di gestire i dati strutturati a grafo.

Spesso lavoriamo con dati tabulari, dati di immagini o dati di testo, tuttavia anche i dati strutturati a grafo sono molto comuni in numerose aree.

Ad esempio, le molecole sono solitamente modellate come un grafo costituito da atomi (nodi) e legami (archi). Un altro esempio interessante sono i grafi della conoscenza di grandi dimensioni, che vengono solitamente utilizzati per i sistemi di raccomandazione da società di e-commerce come Amazon o Netflix. Oltre a questi esempi, ci sono molte aree in cui è possibile trovare dati a grafo, come reti IT, social network, rilevamento di frodi, sport e molti altri.

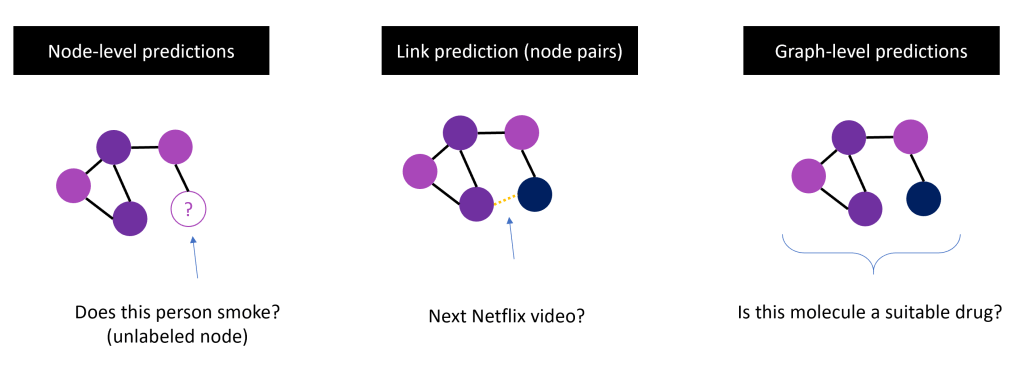
Ma come possiamo sfruttare questi grafi per svolgere compiti di Machine Learning?

Proprio come le classiche attività di machine learning, possiamo eseguire diversi tipi di previsioni in base ai dati del grafo.

Prima di tutto, possiamo fare previsioni a livello di nodo. Ciò significa che si hanno nodi senza label/tag e si cerca di prevedere determinate proprietà in base agli altri nodi e le loro connessioni. Ad esempio, prevedere se una persona fuma o meno (classificazione dei nodi), in base alle informazioni degli amici più stretti in un social network.

Un altro task comune è prevedere i collegamenti nei grafi. Ciò significa che si prevede se il nodo X e il nodo Y avranno, in futuro, una connessione. Ad esempio, si desidera prevedere quali film saranno più probabilmente guardati da una persona su Netflix.

Infine, si può utilizzare l’intero grafo come input (previsione a livello di grafo) ed eseguire previsioni basate sull’intera rappresentazione. Questo di solito viene fatto in medicina, quando si predice la prestazione/idoneità di una nuova molecola come farmaco.

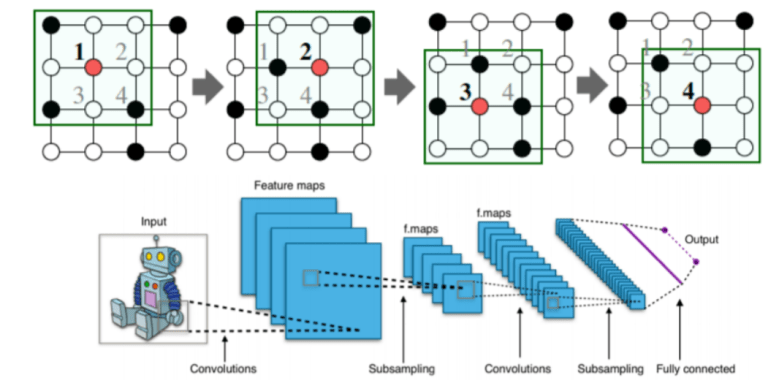


**Come gestire i dati a grafo**

Come sappiamo le reti neurali richiedono un vettore di input di dimensione fissa, poiché hanno un numero fisso di neuroni nel primo livello.

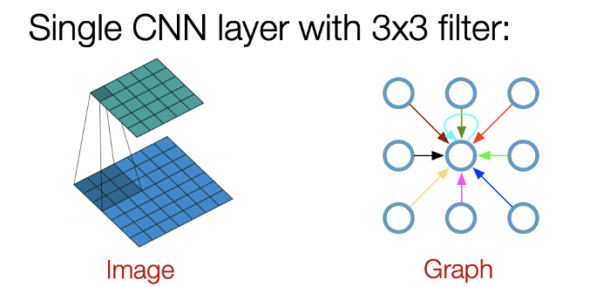
Come possiamo rappresentare i dati di un grafo, con nodi ed archi, come input per una rete neurale?

**Idea: Convolutional Networks**



Negli strati convolutivi si fanno slittare i filtri con determinati pesi sulle immagini, così facendo si esegue l’estrazione automatica delle caratteristiche delle immagini.

**Dalle Immagini ai Grafi**



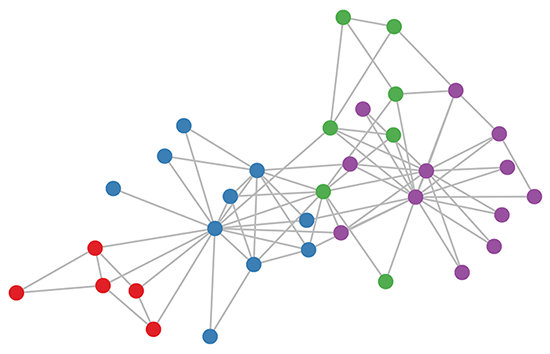
Possiamo considerare i nodi di un grafo regolare a griglia come se fossero i pixel di un’immagine.

L’intuizione che ci permette di raggiungere il nostro obiettivo è che la convoluzione prende una piccola parte rettangolare dell’immagine, applica una funzione ad essa e produce una nuova parte (un nuovo pixel).

Quello che succede è che il nodo centrale della piccola parte rettangolare aggrega le informazioni dai suoi vicini, oltre che da sé stesso, per produrre un nuovo valore.

**Real-World Graphs**

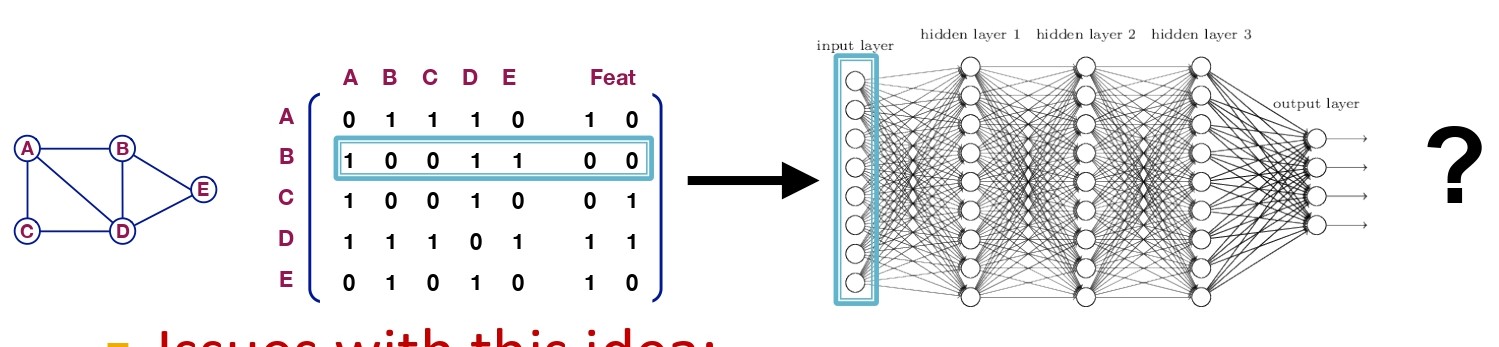
Tuttavia, le reti del mondo reale come i social networks, biological network, medical networks, information networks ecc. non hanno una struttura regolare, ma appaiono come la seguente immagine.

****

Con queste reti è molto difficile eseguire le **CNNs** a causa della dimensione arbitraria del grafo e della topologia complessa.

**Un approccio Naive**

* Unire la matrice di adiacenza con la matrice delle features.
* Dare i vettori delle due matrici unite in input alla rete neurale profonda.

****

Con questo approccio si verificano alcuni problemi:

* parametri, ci sono troppi parametri “una costante moltiplicata per n”.
* Non è applicabile a grafi di dimensione diverse.
* Non è invariante all’ordinamento dei nodi. Se si cambiano i label dei nodi del grafo cambia anche la matrice di adiacenza e quindi anche i vettori che si danno in input alla rete neurale.

Quindi, ricapitolando, per rappresentare un grafo come un vettore di dimensione fissa andiamo incontro ad alcune difficoltà come:

**Difficoltà 1: dimensione e forma dei grafi**

I dati dei grafi sono così complessi che hanno creato molte sfide per gli algoritmi di machine learning esistenti.

Il motivo è che gli strumenti convenzionali di Machine Learning e Deep Learning sono specializzati in tipi di dati semplici.

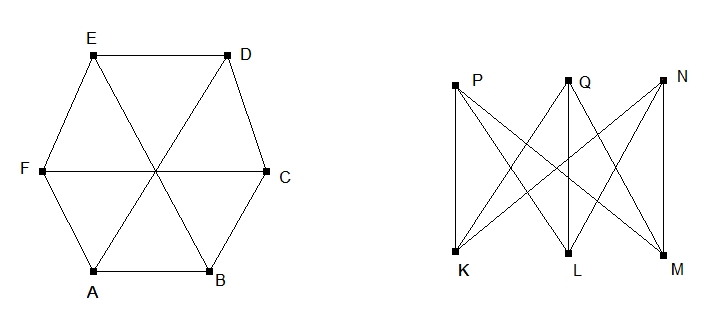
Ma i grafi sono più complessi, senza forma fissa, con una dimensione variabile di nodi non ordinati, dove i nodi possono avere diverse quantità di vicini.

Non possiamo quindi ottenere un vettore di dimensione fissa semplicemente utilizzando i nodi e gli archi disponibili.

Pertanto, abbiamo bisogno di un algoritmo che comprima le informazioni in un vettore di dimensione fissa (embedding), indipendentemente dal numero di nodi e di archi nel grafo.

**Difficoltà 2: isomorfismo**

L’isomorfismo nei grafi significa che esistono strutture dei grafi che appaiono in modo diverso nello spazio, ma queste strutture diverse potrebbero rappresentare lo stesso grafo.



Questi due grafi hanno esattamente gli stessi nodi ed archi. Anche se hanno un aspetto diverso e l’ordine dei nodi potrebbe essere diverso, rappresentano comunque lo stesso oggetto. Se modifichiamo l’ordine, ciò influenzerà la matrice di adiacenza e alla fine l’input per la rete neurale cambierà.

Cerchiamo dunque una rappresentazione che sia identica per entrambi i grafi illustrati.

Abbiamo come requisito che il nostro algoritmo debba essere invariante alla permutazione.

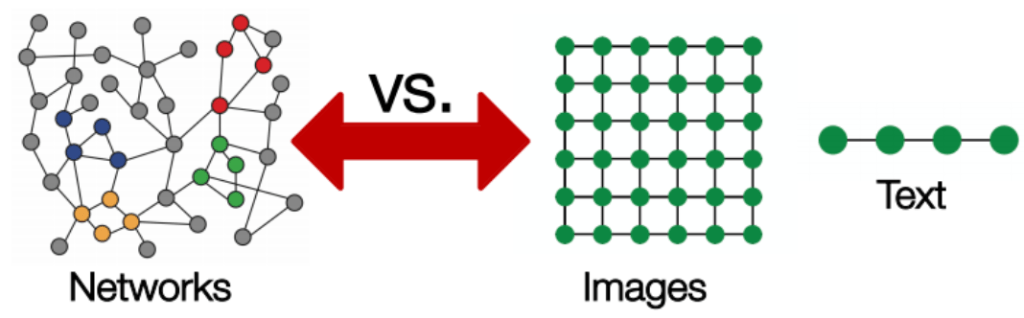
**Difficoltà 3: spazio non euclideo**

Il dominio dei grafi non è euclideo.

Per le immagini con la stessa struttura e dimensione possiamo pensarle come grafi a griglia di dimensioni fisse. Il testo ed il parlato sono sequenze, quindi possiamo considerarli come grafi lineari.

Per le immagini possiamo semplicemente definire filtri a dimensione fissa, che operano su una struttura a griglia, come avviene con le reti neurali convoluzionali (CNN). Nel caso dei grafi, non abbiamo una griglia fissa e quindi dobbiamo operare su strutture alquanto dinamiche.

Non è possibile mappare un grafo in un sistema di coordinate come è possibile farlo con altri tipologie di dati.



**Dobbiamo ancora trovare una risposta alla domanda precedente.**

**Come possiamo rappresentare un grafo come input per una rete neurale?**

Fortunatamente, le GNN sono in grado di gestire tutti i problemi sopra menzionati. Le GNN, come vedremo in dettaglio in seguito, hanno livelli di passaggio di messaggi, che apprendono le informazioni sulla struttura del grafo e le sue caratteristiche. Inoltre le GNN funzionano proprio come qualsiasi altra rete neurale feed forward.

Nelle GNN si ottengono gli incorporamenti a livello di nodo, che sono fondamentalmente vettori generati artificialmente, che contengono tutta la conoscenza del grafo: caratteristiche del nodo, caratteristiche dei vicini, le informazioni di connessione e le caratteristiche dell’arco.

I livelli delle GNN, come vedremo, sono a dimensione fissa pertanto possono essere utilizzati come input diretto per una rete neurale. Inoltre, la propagazione delle informazioni può essere eseguita su qualsiasi struttura del grafo per incorporare le informazioni sui nodi. Pertanto, le GNN risolvono le nostre difficoltà precedenti.

A seguito vediamo come funzionano le GNN. Iniziamo con introdurre il concetto di Embedding (incorporamento).

**Node Embeddings**

Nella teoria dei grafi, implementiamo il concetto di Node Embedding.

Significa mappare i nodi in uno spazio di Embedding (incorporamento)

d-dimensionale, in modo che i nodi simili nel grafo siano mappati l’uno vicino all’altro.

Il nostro obiettivo è mappare i nodi in modo che la somiglianza nello spazio di embedding approssimi la somiglianza dei nodi nella rete.



**Le due componenti chiave**

**Encoder:** Mappare un nodo in un vettore di bassa dimensione.

**Similarity Function:** Definisce come le relazioni nella rete di input si associano alle relazioni nello spazio di embedding.

Definiamo **u** e **v** due nodi di un grafo.

sono due vettori delle features (caratteristiche) dei nodi.

Ora definiremo la funzione encoder , che convertono le caratteristiche dei nodi in vettori di bassa dimensione.

La similarity function tra i due nodi **u** e **v** del grafo deve essere all’incirca uguale al prodotto scalare dei vettori prodotti dalla funzione di Encoder sui nodi **u** e **v**.



**Local Network Neighborhoods**

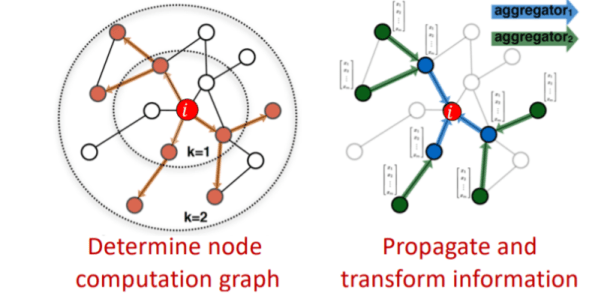
Descrive le strategie di aggregazione e definisce i grafi computazionali.

Il vicinato di un nodo definisce un grafo computazionale.

Dato un nodo del grafo vogliamo fare una qualche previsione per quel nodo, per farlo dobbiamo prima definire un grafo computazionale per quel nodo.

Ogni nodo ha un grafo computazionale/di calcolo univoco, poiché ha vicini diversi.

Come si mostra nella seguente figura, vediamo come il nodo **i**, in rosso, è connesso ai suoi vicini e ai vicini dei suoi vicini.

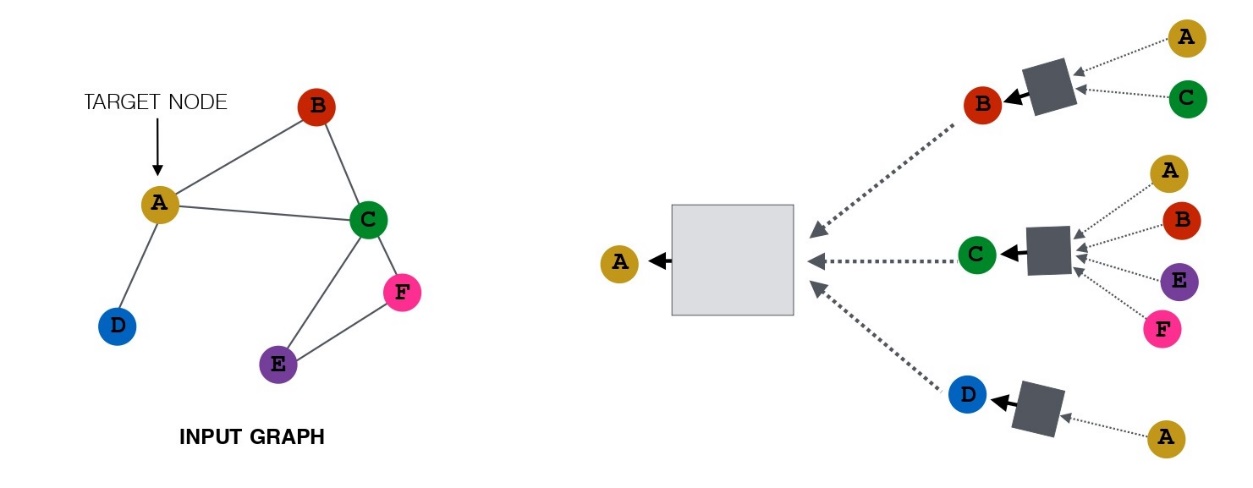


Una volta definito un grafo computazionale per il nodo **i** lo utilizziamo per la propagazione e l’aggregazione dell’informazione dai suoi vicini e dai vicini dei vicini del nodo **i,** per fare una previsione su tale nodo.

Quindi si propaga l’informazione attraverso il grafo per calcolare le caratteristiche del nodo.

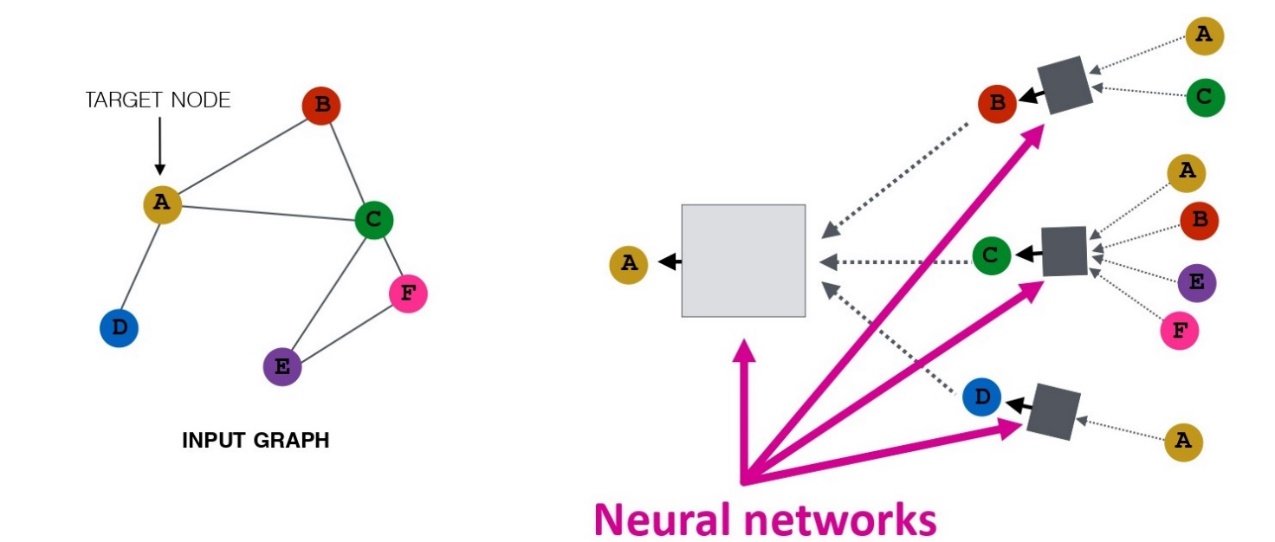
In questo modo, acquisiamo la struttura della rete attorno al nodo **i** e allo stesso tempo prendiamo in prestito le informazioni dai nodi vicini per arricchire la descrizione o rappresentazione del nodo **i.**

L’idea chiave per fare ciò è generare degli incorporamenti (embeddings) per i nodi, basati sui vicinati della rete locale.



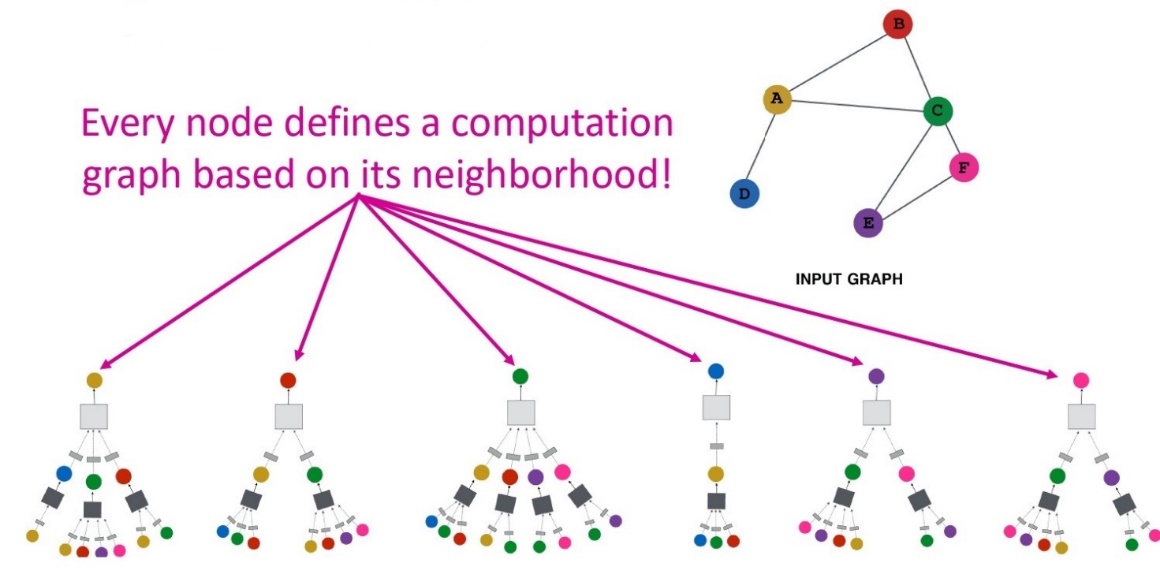
Ad esempio, vogliamo creare un incorporamento per il nodo A del grafo sopra, per farlo dobbiamo aggregare l’informazione dai suoi nodi vicini (B, C, e D) che a loro volta aggregano l’informazione dai loro vicini generando i loro rispettivi incorporamenti.

I nodi per aggregare le informazioni dai loro vicini utilizzano le reti neurali.



Notiamo che il vicinato di ciascun nodo (vicini ed i vicini dei vicini) definisce un grafo computazionale.

Quindi per ciascun grafo computazionale di ogni nodo possiamo eseguire l’aggregazione dell’informazione ottenendo così diverse architetture di reti neurali.



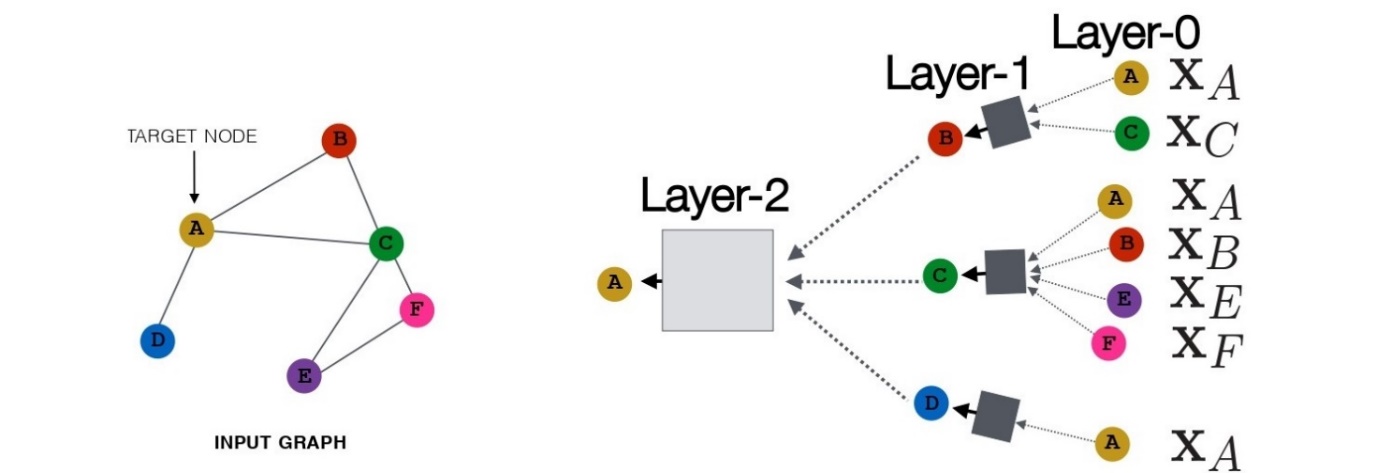
Con il grafo usato per l’esempio precedente, otteniamo per ciascun nodo un grafo computazionale con due livelli di profondità, ma ci sono grafi molto più gradi con molti più nodi dai quali si genererebbero grafi computazionali con più di due livelli di profondità.

Il modello per l’aggregazione può essere utilizzato anche con grafi computazionali con profondità arbitraria.

I nodi hanno un incorporamento in ogni livello.

Al livello-0 l’incorporamento di un nodo u è la sua caratteristica di input, .

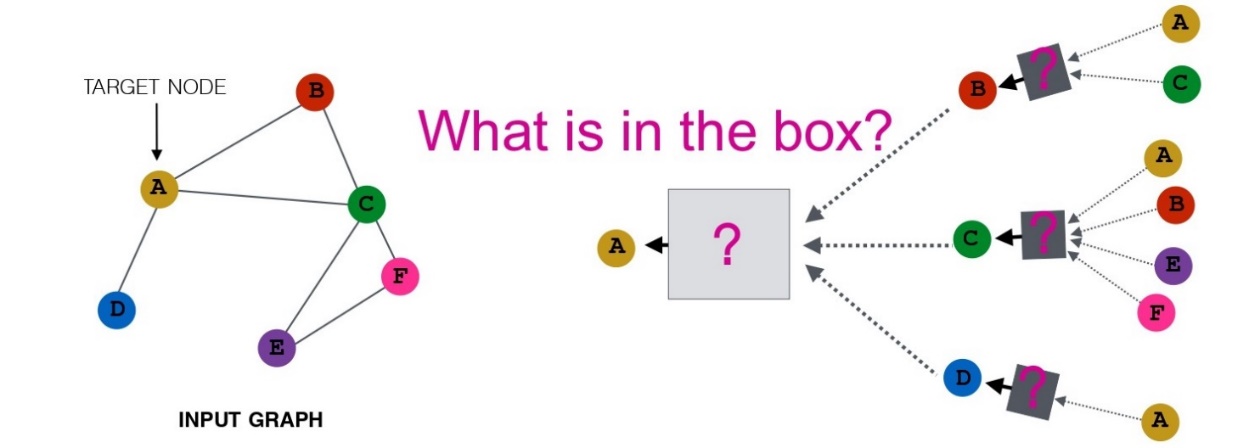
Al livello-k l’incorporamento di un nodo ottiene le informazioni dai nodi che sono entro i k passi (hops) di distanza.



**Neighborhood Aggregation**

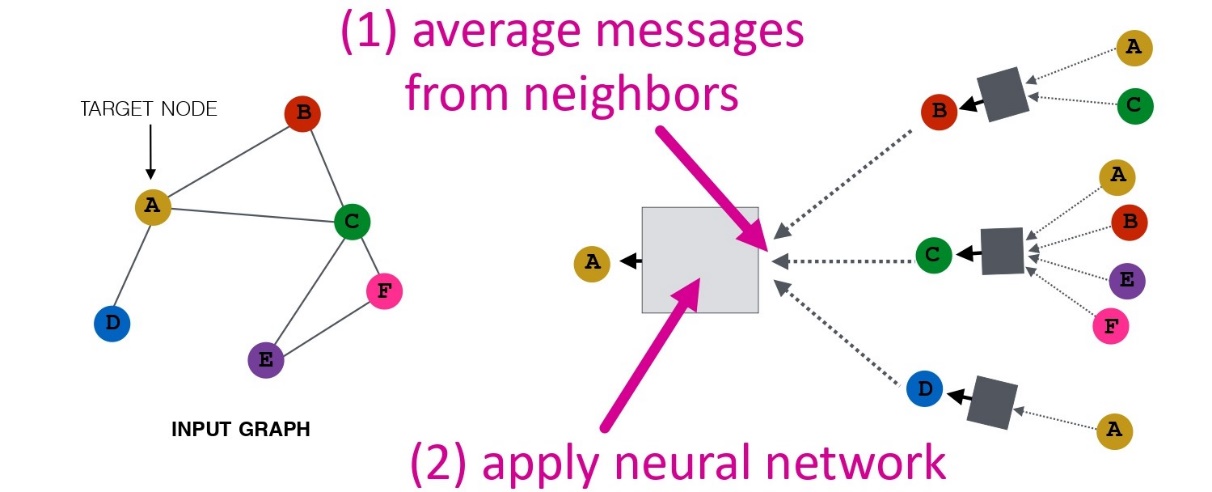
Con questo meccanismo di propagazione e aggregazione dell’informazione per generare incorporamenti dei nodi, la distinzione chiave sta nel modo in cui viene fatta l’aggregazione dell’informazione attraverso i livelli del grafo computazionale.

In altre parole, la distinzione sta nel “cosa mettiamo all’interno delle scatole grigie per aggregare le informazioni dai nodi vicini”.



Il primo approccio base è quello di utilizzare la **media** come aggregatore delle informazioni ottenute dai nodi vicini.

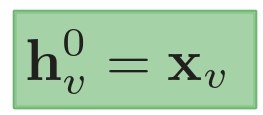
Dopodiché prendiamo queste informazioni aggregate con la media e le diamo in input alle reti neurali presenti nei box.



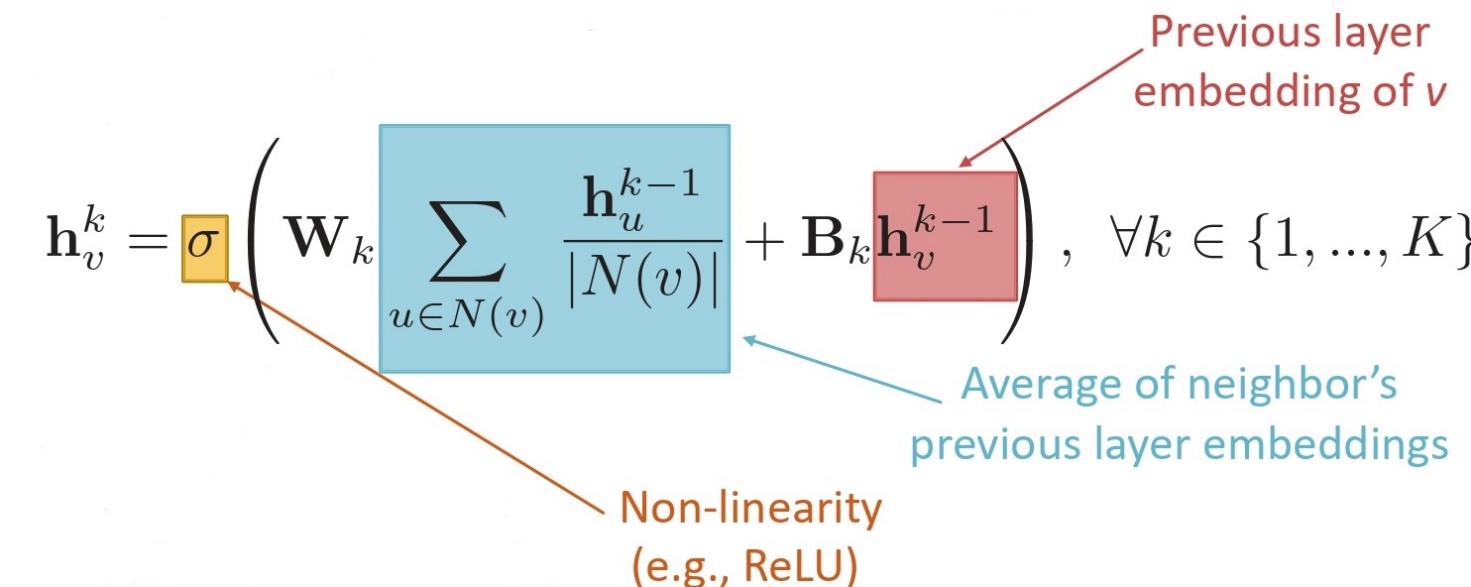
Le reti neurali richiedono che le aggregazioni siano invarianti nell’ordine, ciò significa che si deve ottenere lo stesso risultato indipendentemente dall’ordine delle componenti (input features), quindi per l’aggregazione usiamo funzioni come la somma, la media, il massimo, perché sono funzioni invarianti alla permutazione. Questa proprietà consente di eseguire le aggregazioni.

Formalmente quello che facciamo aggregando con la media è:

1. **Inizializzare gli input features dei nodi:** al livello-0 gli incorporamenti sono uguali agli input features dei nodi.



1. **Ad ogni livello eseguire la funzione di aggregazione per ciascun nodo:**

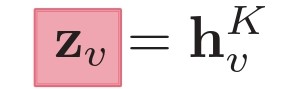


La parte in blu è fondamentalmente la media degli incorporamenti di tutti i vicini del nodo **v**.

La parte in rosso è l’incorporamento del livello precedente del nodo **v** moltiplicato per un Bias .

La parte in giallo è la funzione di attivazione che viene eseguita su entrambe le parti precedenti.

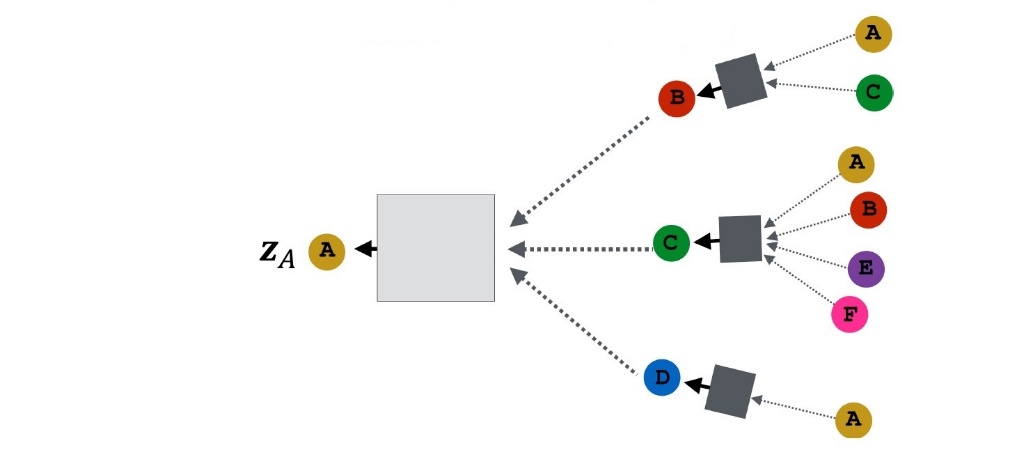
1. **L’ultima equazione allo strato finale:**

****

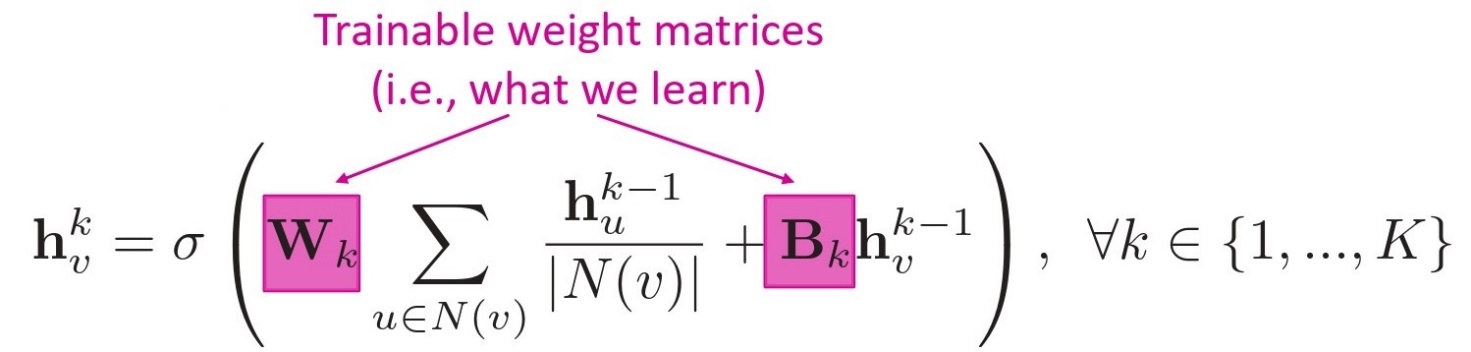
È l’incorporamento finale dopo k strati di aggregazione del vicinato.

**Addestramento del modello**

Come facciamo ad addestrare questo modello per generare gli incorporamenti finali?



Dobbiamo definire una funzione di perdita (loss function) sugli incorporamenti.



Possiamo dare in input questi incorporamenti ad una qualsiasi funzione di perdita ed eseguire la discesa del gradiente per addestrare i parametri dei pesi.

Se, per esempio, la matrice dei pesi W fosse di tutti zeri, allora il nuovo incorporamento del nodo non impara nulla dai nodi vicini e prende solo le caratteristiche del nodo stesso.

Se invece i pesi della matrice W fossero molto elevati si considererebbero solamente le informazioni dai nodi vicini e si trascurerebbero le caratteristiche del nodo stesso.

Quindi è molto importante trovare i pesi ottimali per ottenere sia le caratteristiche dai vicini che dal nodo stesso.

Questo è importante se si vuole, per esempio, fare una previsione riguardo ad una persona (nodo) in un social network e si conoscono chi sono gli amici della persona e le loro caratteristiche, possiamo imparare come combinare queste proprietà con quelle della persona stessa per fare una previsione, aggiustando i parametri.

**Unsupervised Training**

Nell’addestramento non supervisionato utilizziamo solo la struttura del grafo: nodi simili hanno incorporamenti simili.

La funzione perdita senza supervisione può essere una perdita basata sui:

* Cammini casuali
* Vicinanza dei nodi nel grafo
* Qualsiasi altra tecnica per misurare la somiglianza tra i nodi

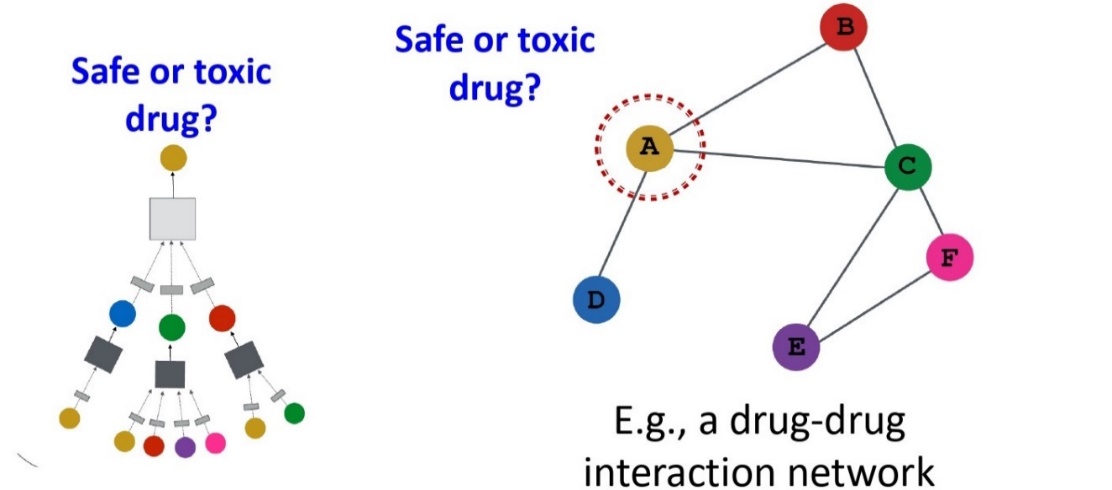
**Supervised Training**

Addestra direttamente il modello per un’attività supervisionata come la classificazione dei nodi.

Per esempio, se consideriamo la rete di interazioni tra farmaci dove i nodi sono i farmaci e gli archi sono le loro interazioni, ed i nodi sono classificati come farmaco sicuro o farmaco tossico.

Supponiamo ora di dover fare una previsione per il nodo A.

Il nodo A è un farmaco sicuro o tossico?



Fondamentalmente quello che si fa è calcolare l’incorporamento finale del nodo A (encoder output) e su questo calcolare una funzione di perdita ed eseguire la propagazione all’indietro della perdita per aggiustare i parametri dei pesi.

**Funzione di perdita**

**Cross Entropy Loss**

Per capire meglio come funziona questa funzione di perdita, consideriamo un classificatore che predice se un dato animale è cane, gatto o cavallo con una probabilità associata a ciascuno.

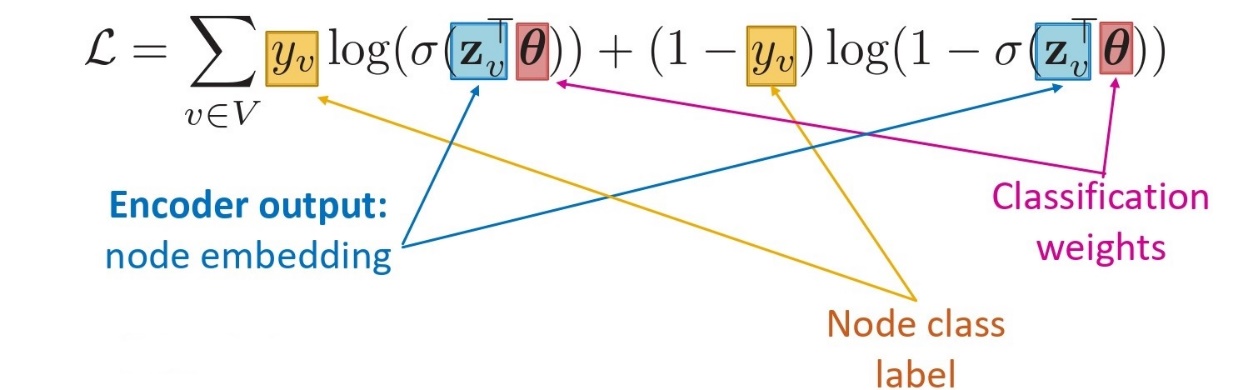
Supponiamo che l’immagine originale sia del cane e che il modello preveda i valori 0.7, 0.2, 0.1 come probabilità per le tre classi (con distribuzione di probabilità chiamata Q) in cui la vera probabilità assomiglia [1, 0, 0] (con distribuzione di probabilità P). Ciò che vogliamo è che le nostre probabilità previste [0.7, 0.2, 0.1] siano vicine a quella originale distribuzione di probabilità [1, 0, 0].

Quindi l’entropia incrociata assicura di ridurre al minimo la differenza tra le due probabilità, definite Q e P. Essa può essere formalmente dichiarata come:

Dove H è la funzione di entropia incrociata, P può essere la distribuzione attuale e Q è l’approssimazione della distribuzione target.

Quindi la funzione di perdita cerca di ridurre il gap tra il valore previsto e il valore effettivo.

Vediamo un esempio di funzione di perdita:



Questa funzione di perdita è la perdita dell’entropia incrociata (cross entropy loss).

Nella parte sinistra della somma abbiamo in azzurro l’incorporamento finale del nodo V di interesse (Encoder output) e si fa la combinazione lineare con un vettore di pesi di classificazione, il risultato viene dato in input ad una funzione di regressione logistica che dà in output una probabilità di appartenenza alla classe y = 1 (safe), nella parte destra della somma si fa il complemento che dà in output la probabilità di appartenenza alla classe y=0 (toxic).

Su questi due output si applica il logaritmo ottenendo la probabilità in scala logaritmica anziché nell’intervallo unitario standard [0, 1].

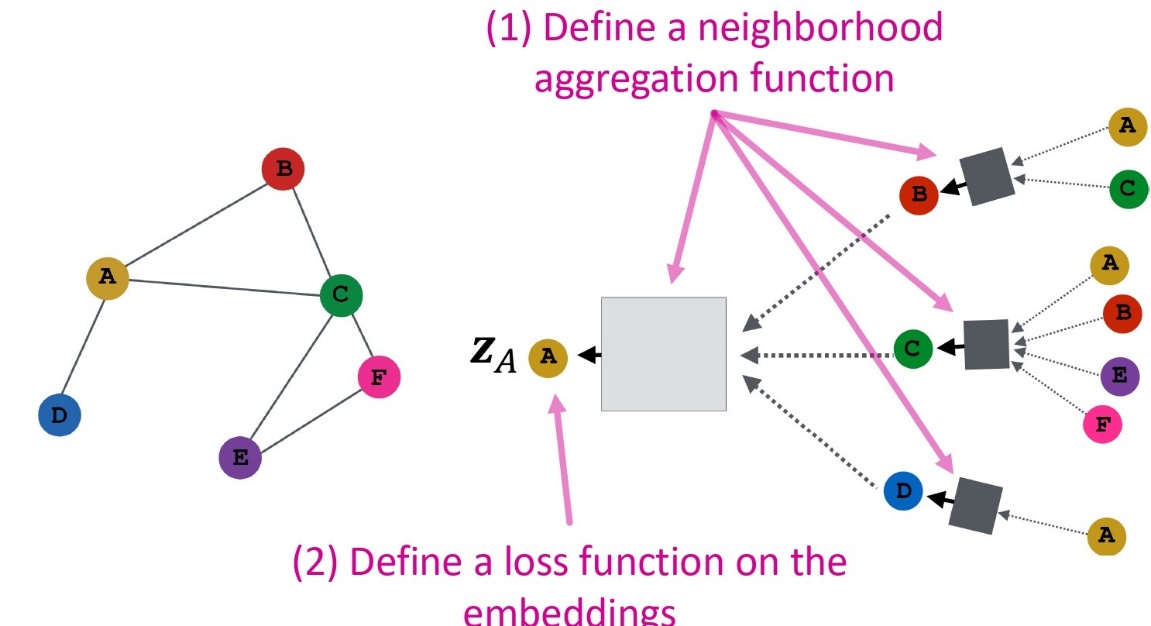
Dopodiché, si moltiplica per il label di appartenenza effettivo del nodo V nella parte sinistra della somma, e per il complemento del label nella parte destra.

I valori dei label dei nodi (), nel caso della classificazione binaria sono 0 o 1, il che implica che una delle due parti della somma venga azzerata in base al label del nodo.

**Progettazione del modello: panoramica**

Ricapitolando, dato un grafo per prima cosa dobbiamo calcolare gli incorporamenti dei nodi del grafo, per farlo dobbiamo definire i grafi computazionali per ciascun nodo.

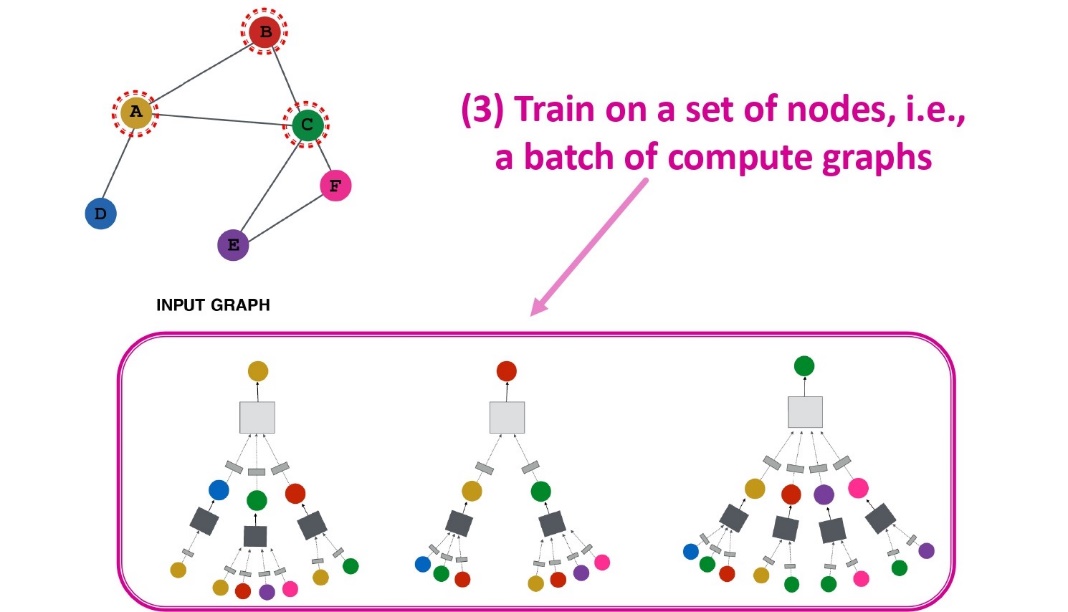
Vediamo come farlo per un dato nodo:

* Definiamo il grafo computazionale basato sul vicinato del nodo.
* Definiamo una funzione di aggregazione delle informazioni propagate dai nodi vicini.
* Definiamo una funzione di perdita sull’incorporamento finale del nodo (Encoder output) per aggiustare i parametri e minimizzare la perdita.

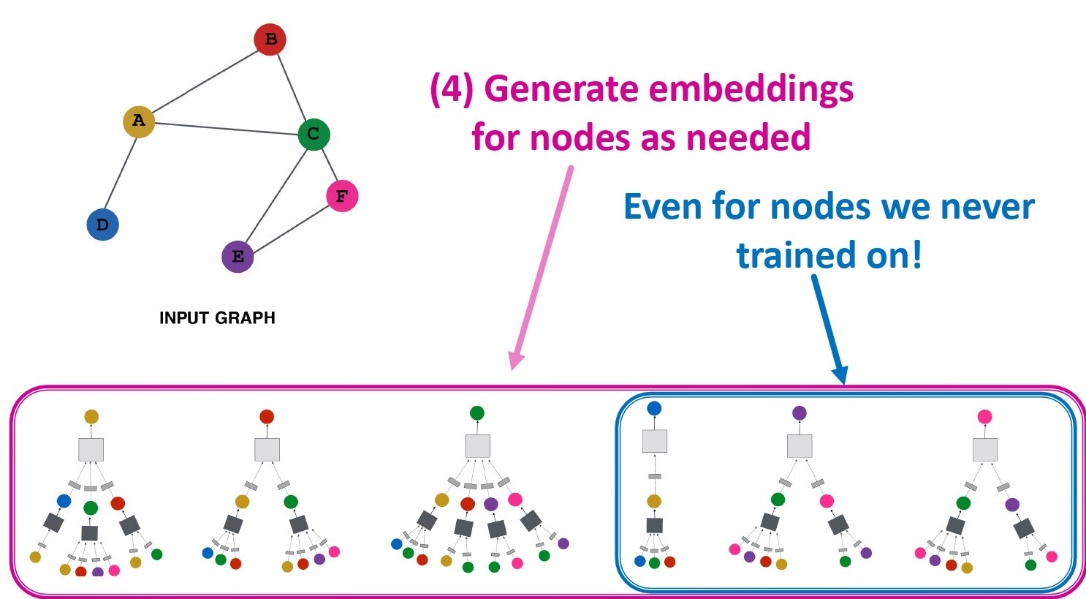
Un aspetto interessante è che possiamo addestrare il modello su un sottografo, cioè su un sottoinsieme di nodi del grafo.

Per esempio addestriamo il modello solo su tre nodi del grafo dei quali conosciamo la loro classe di appartenenza.

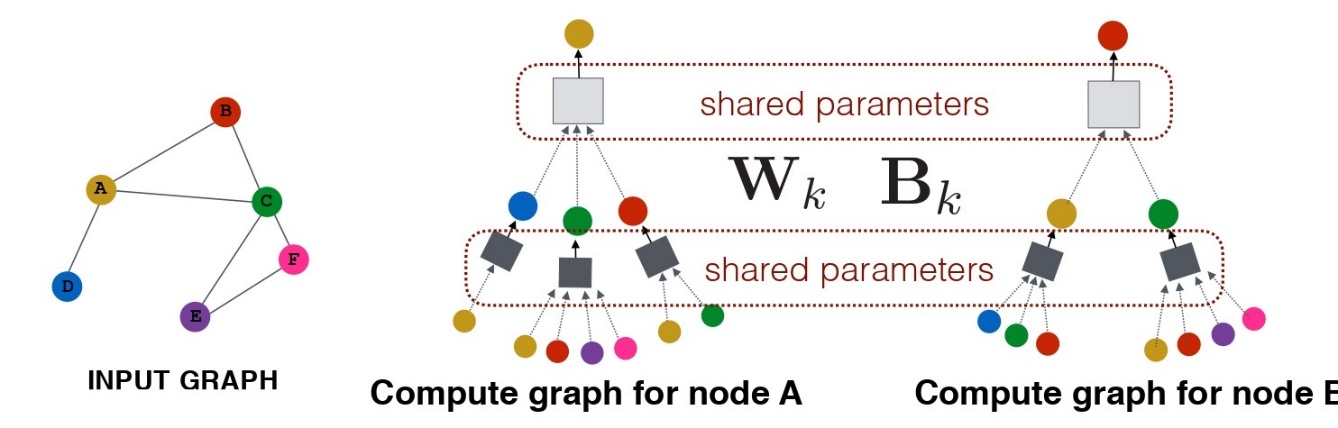
Quindi creiamo solamente tre grafi computazionali dei rispettivi tre nodi, e addestriamo il modello solo su questi.



Una volta addestrato il modello, cioè una volta che abbiamo ottenuto i valori dei pesi affinché la perdita sia minima, possiamo applicare il modello e fare previsioni sugli altri nodi del grafo, sui quali non abbiamo addestrato il modello, e dei quali non conosciamo la loro classe di appartenenza.



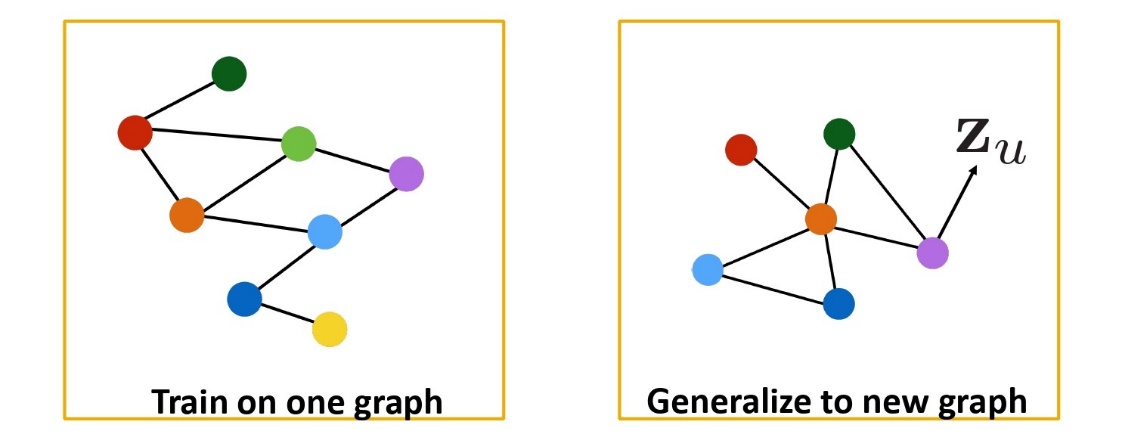
La ragione per la quale possiamo fare ciò è perché gli stessi parametri di aggregazione sono condivisi per tutti i nodi.



Anche se abbiamo diversi grafi computazionali, i parametri che governano le aggregazioni sono condivisi attraverso le diverse architetture.

Il numero dei parametri del modello è sottolineare a |V|, cioè non superano mai il numero dei nodi del grafo che è costante, e possiamo generalizzarli anche per gli altri nodi non conosciuti del grafo.

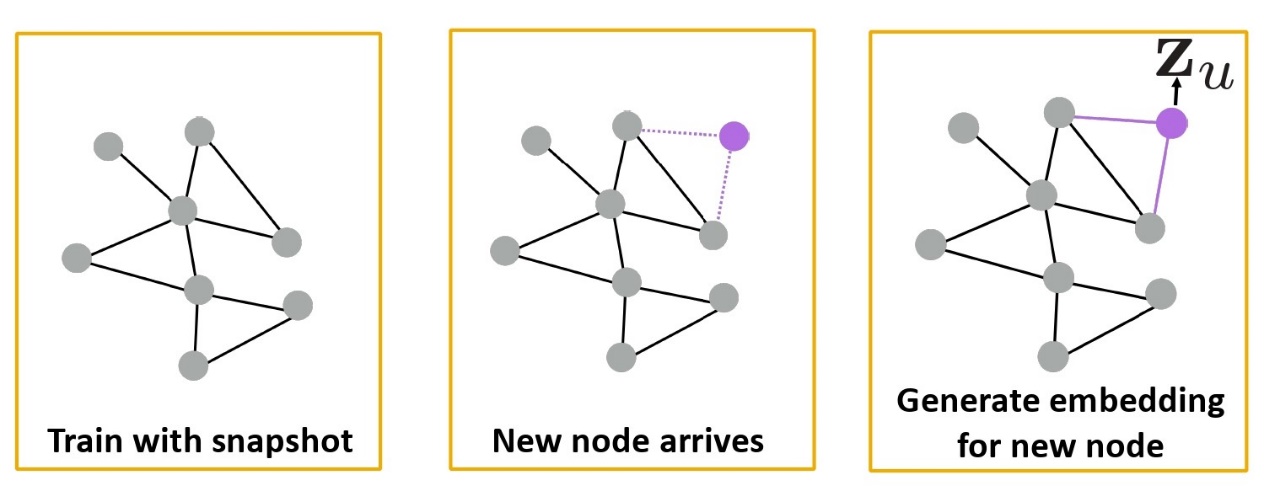
Possiamo addestrare i parametri W e B su un determinato grafo e quando arriva un nuovo grafo possiamo applicarvi gli stessi parametri.



Ad esempio, possiamo addestrare i parametri su un grafo di interazioni fra proteine di un organismo A e generare, con gli stessi parametri, incorporamenti sui dati raccolti di un organismo B.

Possiamo fare lo stesso anche nel caso arrivino nuovi nodi a far parte del grafo.

Per esempio in un social network abbiamo una istantanea del grafo e su questa vi addestriamo i parametri, successivamente arriva un nuovo nodo nella rete ed utilizzando gli stessi parametri possiamo generare l’incorporamento per il nuovo nodo.



Questo metodo di trasferire i parametri alle parti sconosciute del grafo è molto utile nel caso in cui abbiamo un grafo molto grande che è troppo grande per fare su esso l’addestramento, quindi quello che possiamo fare è fare l’addestramento solo su una parte del grafo e poi applicare il modello sulle altre parti del grafo.

**GraphSAGE and Graph Convolutional Network**

**GraphSAGE Idea**

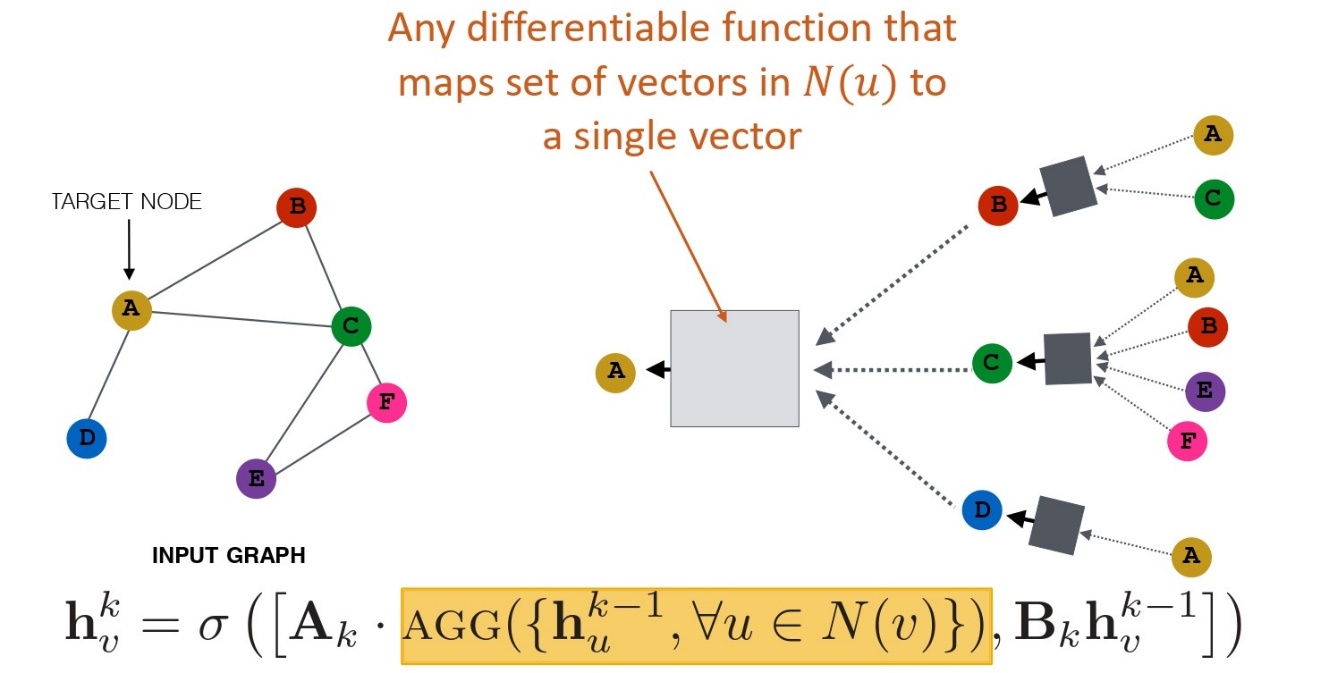
Finora abbiamo utilizzato le reti neurali per aggregare e trasformare le informazioni, calcolando la media degli incorporamenti dei nodi vicini e sommando l’incorporamento del nodo stesso.

Ora proviamo a generalizzare questo meccanismo.

Prendiamo le caratteristiche del nodo stesso dal precedente livello ed utilizziamo la matrice B per trasformarle.

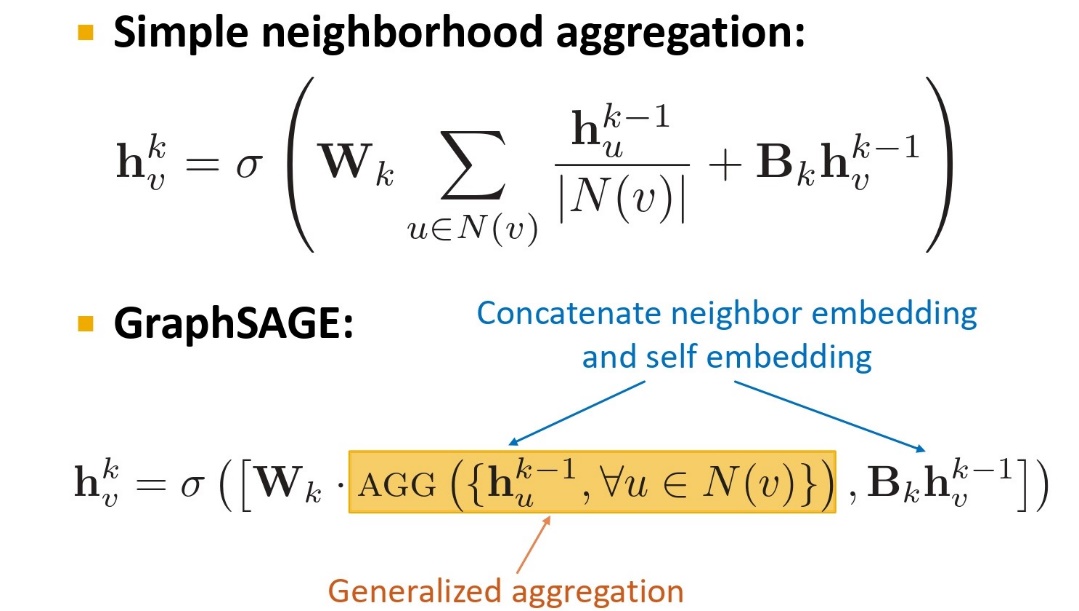
Successivamente prendiamo gli incorporamenti dei nodi vicini dal precedente livello e, per aggregare questi incorporamenti utilizziamo una funzione di aggregazione generale e trasformiamo questa aggregazione con una matrice dei pesi W.

Dopodiché, concateniamo l’aggregazione degli incorporamenti dei vicini e l’incorporamento del nodo stesso e li diamo in input ad una funzione di attivazione **.**

****

In questo modo generalizziamo utilizzando una funzione generale per l’aggregazione e, anziché avere una somma dell’incorporamento del nodo stesso, lo teniamo separato dall’aggregazione concatenandoli prima di darli entrambi in input a .

Ricapitolando:

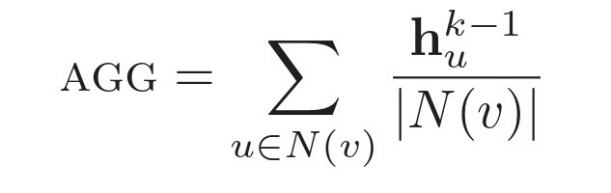
****

Generalizzando possiamo analizzare le proprietà delle diverse funzioni di aggregazione.

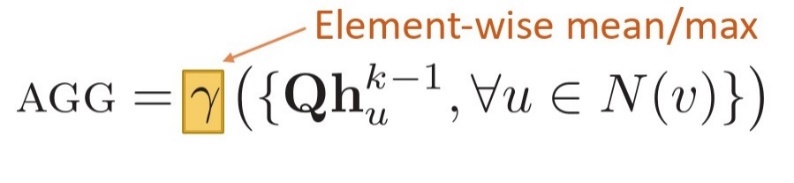
**Funzioni di aggregazione: Varianti**

Come possibili funzioni di aggregazione nella forma generalizzata possiamo usare:

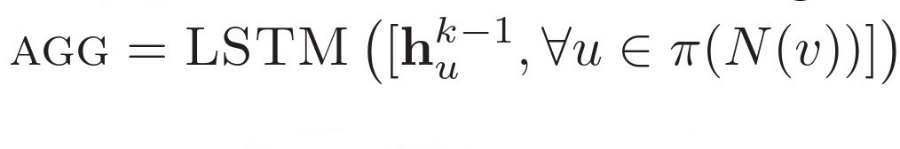
* **La Media:** come abbiamo detto finora, prende le informazioni dai nodi vicini, le somma e le normalizza per il numero dei nodi vicini.



* **Pool:** per aggregare gli incorporamenti dei nodi vicini si applica una strategia di pooling, dove prediamo i messaggi dai nodi vicini e li trasformiamo, successivamente a questi vettori dei nodi vicini trasformati con dei pesi, vi applichiamo un’operazione deterministica (tipicamente il massimo o la media).



* **LSTM:** possiamo utilizzare una rete neurale come la LSTM per aggregare i messaggi dai nodi vicini. Le reti neurali LSTM non sono invarianti all’ordine delle componenti, quindi quando facciamo l’addestramento lo facciamo su molti ordinamenti casuali.



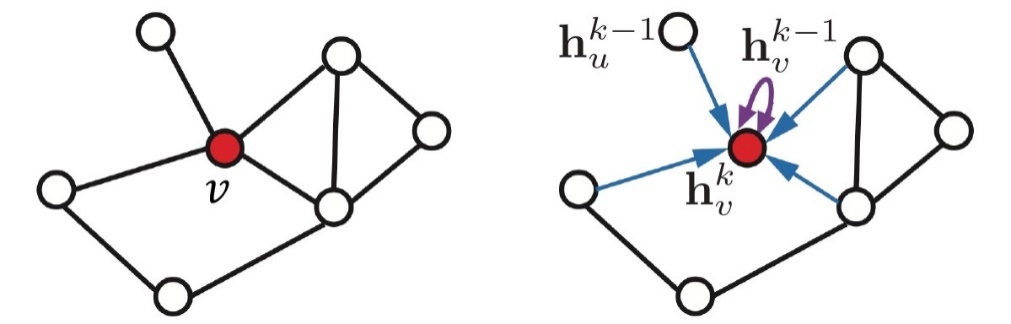
**Ricapitolando: GCN, GraphSAGE**

Idea chiave: si generano gli incorporamenti dei nodi basati sui vicinati locali.

I nodi aggregano i “messaggi” dai loro vicini utilizzando reti neurali.

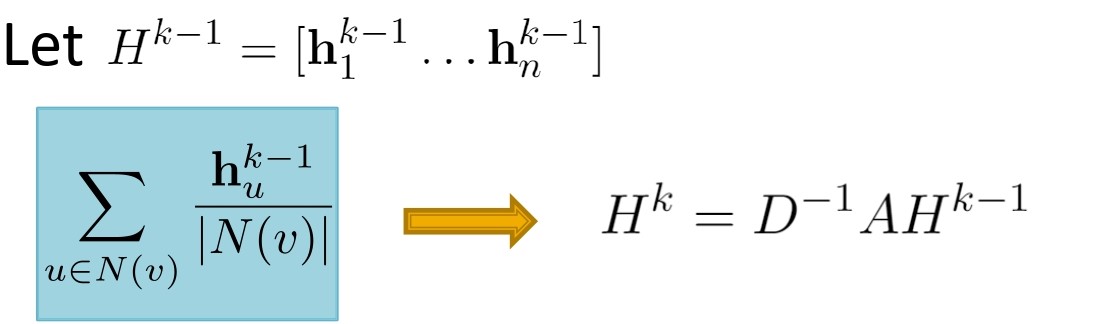
**Graph Convolutional Network:** si fa la media delle informazioni del vicinato e si impilano i livelli della rete.

**GraphSage:** si generalizza l’aggregazione del vicinato.



**Implementazione**

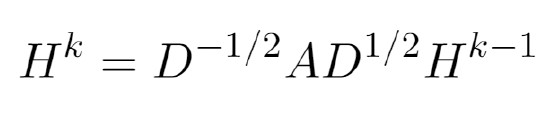
È possibile eseguire molte aggregazioni in modo efficiente mediante operazioni con matrici.



L’aggregazione dell’informazione dai nodi vicini fatta con la media, può essere fatta anche facendo la moltiplicazione riga per colonna tra matrici, dove A è la matrice di adiacenza, D è la matrice (diagonale) dei gradi dei nodi e è la matrice degli incorporamenti del livello precedente.

Così facendo otteniamo l’incorporamento .

Un altro esempio è come si fa l’aggregazione nelle **Graph Convolutional Network (GCN)** con la moltiplicazione tra matrici:



**Graph Attention Networks (GAT)**

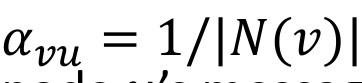
**Aggregazione semplice del vicinato**

****

Come abbiamo ripetuto più volte, l’aggregazione semplice consiste nell’aggregare le informazioni dai vicini e trasformarle con dei pesi per poi combinarle con la trasformazione dell’incorporamento del nodo stesso.

Come un certo nodo può passare più informazioni rispetto un altro nodo?

* **Operatore convoluzionale del grafo:**
* Aggrega i messaggi dai nodi vicini, N(v)
* Desideriamo un peso (che ci indichi quanto importante è un dato arco che collega il nodo un certo vicino:



è il peso dell’arco che collega U a V e che regola l’importanza del messaggio dal nodo U al nodo V.

è definito basandosi sulla proprietà strutturale del grafo.

In questo caso, basandosi sull’equazione precedente, tutti i nodi vicini al nodo V sono di ugual importanza.

Ma non vogliamo che tutti i nodi abbiamo la stessa importanza.

Quindi, possiamo fare di meglio che assegnare una uguale importanza ad ogni nodo vicino?

**Obiettivo:** specificare importanze arbitrarie per i vicini di ciascun nodo del grafo.

**Idea:** calcolare l’incorporamento per ciascun nodo del grafo seguendo una strategia di attenzione:

* I nodi partecipano nel messaggio del loro vicinato
* Specificare pesi diversi per i diversi nodi in un vicinato

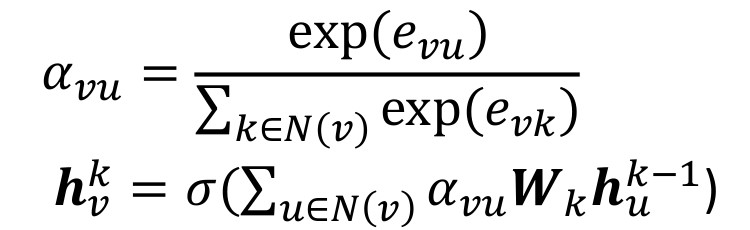
**Meccanismo di Attenzione**

Sia calcolato come sottoprodotto di un meccanismo di attenzione :

* Nel meccanismo di attenzione , calcoliamo i coefficienti di attenzione tra le coppie di nodi u, v basandoci sui loro messaggi:



* indica l’importanza del messaggio dal nodo u al nodo v
* Normalizziamo i coefficienti usando la funzione softmax per essere confrontabile tra i diversi vicinati di diversa dimensione:



Ora l’aggregazione la si fa moltiplicando anche per , che è la normalizzazione dei coefficienti di attenzione del vicinato del nodo v.

Il meccanismo di attenzione è una funzione calcolata sui messaggi tra due nodi vicini per ricavare il loro coefficiente di attenzione .

Ma che forma ha questo meccanismo di attenzione ?

**Meccanismo di attenzione**

L’approccio è agnostico per la scelta della funzione :

* Usiamo una rete neurale costituita da un singolo livello
* La funzione può avere dei parametri, i quali dovranno essere stimati

I parametri di sono addestrati congiuntamente alle matrici dei pesi W e B.

**Multi-head attention:**

L’idea è che in un dato livello, del grafo computazionale, si ripete R volte una funzione di attenzione, e ciascuna ripetizione è fatta su parametri diversi, in altre parole otteniamo R coefficienti di attenzione diversi per ciascun arco.

Gli output poi verranno aggregati.

Quali sono i benefici che porta calcolare questi coefficienti di attenzione per ciascun arco?

**Proprietà del meccanismo di attenzione**

Il beneficio principale del meccanismo di attenzione è che ci consente di specificare diversi valori di importanza per vicini diversi.

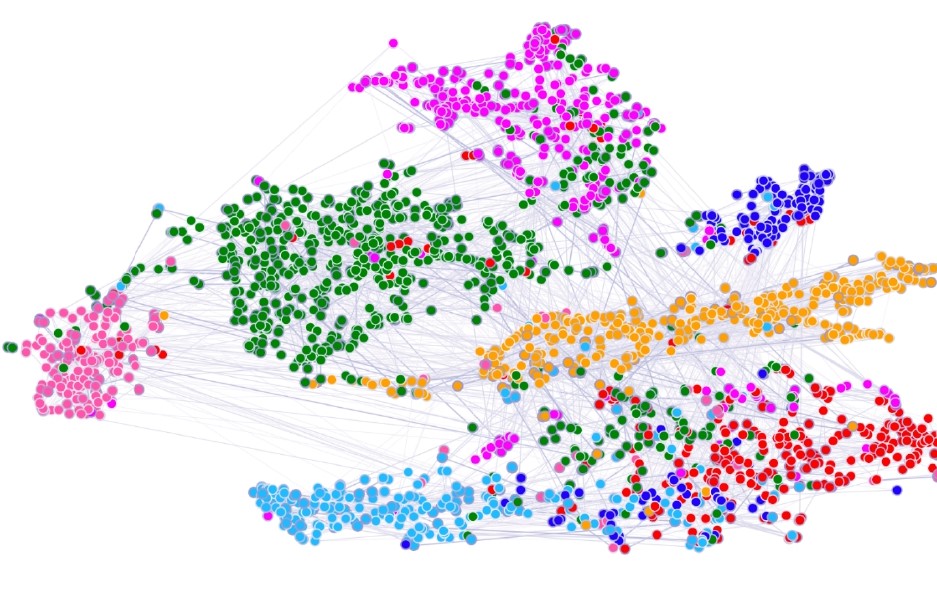
Inoltre è un meccanismo localizzato solamente nei vicinati della rete locale, ed è un meccanismo condiviso solamente dagli archi quindi non dipende dalla struttura globale del grafo, e ciò consente di applicare questo meccanismo anche su nuovi nodi e grafi che non abbiamo mai visto.

**Esempio: Cora Citation Net**

In questo esempio abbiamo un grafo di citazioni in cui ogni nodo è un documento e gli archi rappresentano le citazioni tra diversi documenti che appartengono a diverse aree scientifiche.

L’obiettivo è predire qual è l’argomento/area scientifica di un dato documento (nodo).

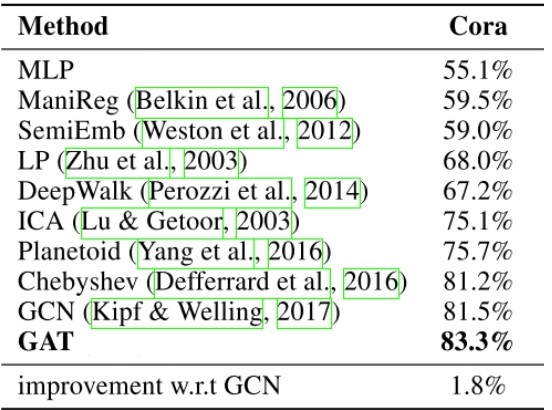
Gli argomenti dei documenti sono indicati dai diversi colori dei nodi.



Vediamo come l’accuratezza varia in base ai metodi utilizzati per la previsione.

Per esempio vediamo che utilizzando solo il testo del documento per fare la previsione della classe di appartenenza di un documento (MLP) otteniamo il 55% di accuratezza.

Se invece utilizziamo come metodi per la previsione, metodi basati sui grafi come ad esempio il Graph Convolutional Network (GCN) otteniamo l’81% di accuratezza, ed utilizzando il meccanismo di GAT otteniamo un incremento fino ad arrivare all’83% di accuratezza.

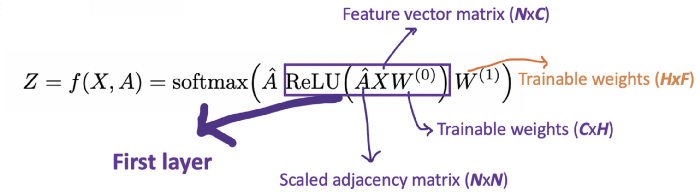


**Diverse Varianti di GNN**

In questa relazione abbiamo visto 3 diverse varianti di GNN:

1. **Graph Convolutional Networks (GCN)**, possiamo riassumere il loro funzionamento come segue utilizzando il prodotto tra matrici:

* Moltiplichiamo la matrice di adiacenza in cui abbiamo inserito anche i self-loop con la matrice delle features X:
* Facciamo la media moltiplicando la matrice di adiacenza con il reciproco della matrice dei gradi , in questo modo si ridimensionano solamente le righe di , e poi moltiplichiamo per la matrice delle features X: .
* Per ridimensionare anche le colonne di moltiplichiamo una seconda volta per e otteniamo:
* Poiché abbiamo moltiplicato 2 volte per la stessa matrice dei gradi , per bilanciare anziché usare utilizziamo :
* Chiamiamo , e supponiamo di avere 2 livelli nel grafo per raggiungere tutti i nodi, quello che otteniamo come incorporamento finale usando la funzione ReLu per il primolivello e la funzione Softmax per il secondo livello è:

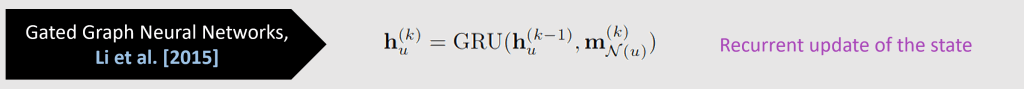


Dove N indica il numero di nodi, C è la dimensione della matrice delle features, H è il numero di nodi nel hidden layer e F è la dimensione del vettore risultante.

1. **GraphSAGE**, si generalizzano le funzioni di aggregazione per valutarne le proprietà.
2. **Graph Attention Network (GAT)**, si calcola un coefficiente di importanza a ciascun arco.

Ma oltre a queste tre varianti ce ne sono altre, per esempio un’altra implementazione comune di GNN consiste nell’usare un **Multi-Layer Perceptron (MLP)** come funzione di aggregazione. Attraverso i parametri aggiustabili nel modello, possiamo imparare la migliore aggregazione dei vicini.

Un’altra variante sono le **Gated Graph Neural Networks (GGNN)**, che utilizzano come funzione di attivazione, detta anche **Updater** (è la funzione che prende in input il risultato dell’aggregazione), un’unita Gated Recurrent che è una versione semplificata di LSTM.



Ciò consente di apprendere meglio le informazioni sequenziali sugli stati.

Esistono molte altre varianti di GNN che utilizzano diverse funzioni di aggregazione e di aggiornamento, ma che per ragioni di tempo e di spazio non entriamo in dettaglio su ciascuna di queste.

**Esempio di Applicazione**

**Pinterest**

Pinterest è un social network basato sulla condivisione di immagini, video e fotografie. Permette agli utenti di creare bacheche in cui catalogare le immagini in base a temi predefiniti oppure da loro scelti.

La stessa immagine riguardante ad una cucina per esempio, può essere salvata all’interno di molte collezioni diverse di utenti diversi, oppure le immagini nelle collezioni di utenti diversi possono avere qualcosa in comune.

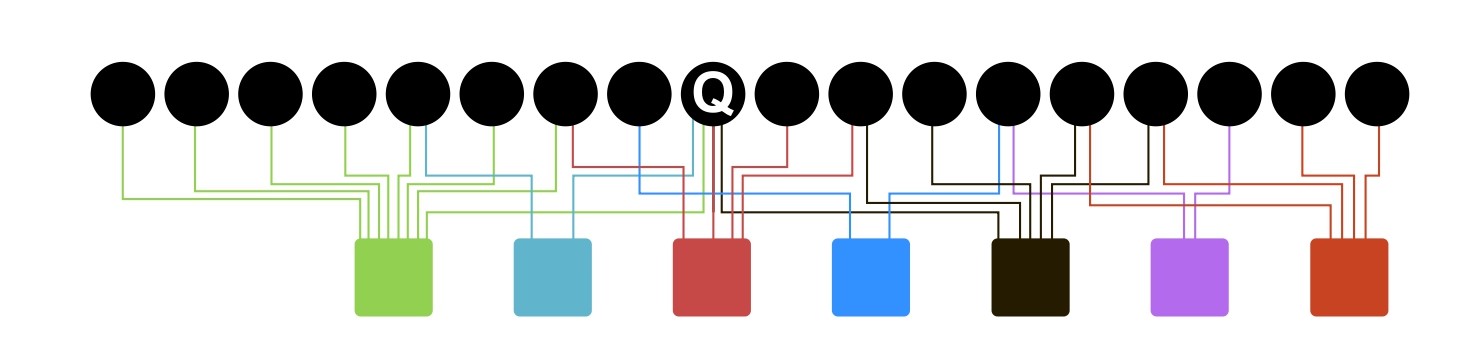
Un **pin** è un’immagine, testo o link che collega ad una immagine.

Una **board** invece è una collezione di idee, cioè di pins che hanno qualcosa in comune.



**Pinterest Graph**

Otteniamo un grafo bipartito in cui le due tipologie di nodi sono i pin e le board che sono collegati tra di loro, dove ciascun arco indica che c’è qualcosa in comune tra i pin e le board.



Questo grafo è costituito da 2 miliardi di pin, 1 miliardo di board e da 20 miliardi di archi.

Il grafo è dinamico quindi è necessario applicare il modello a nuovi nodi senza riqualificarlo.

Utilizziamo questo grafo per migliorare le raccomandazioni e comprendere il contenuto del grafo.

**PinSAGE**

Problema: se prendiamo un’immagine con oggetti ambigui, la computer vision potrebbe fare degli errori.

Per esempio potrebbe fare confusione con la ringhiera di un cancello e la ringhiera di un letto.



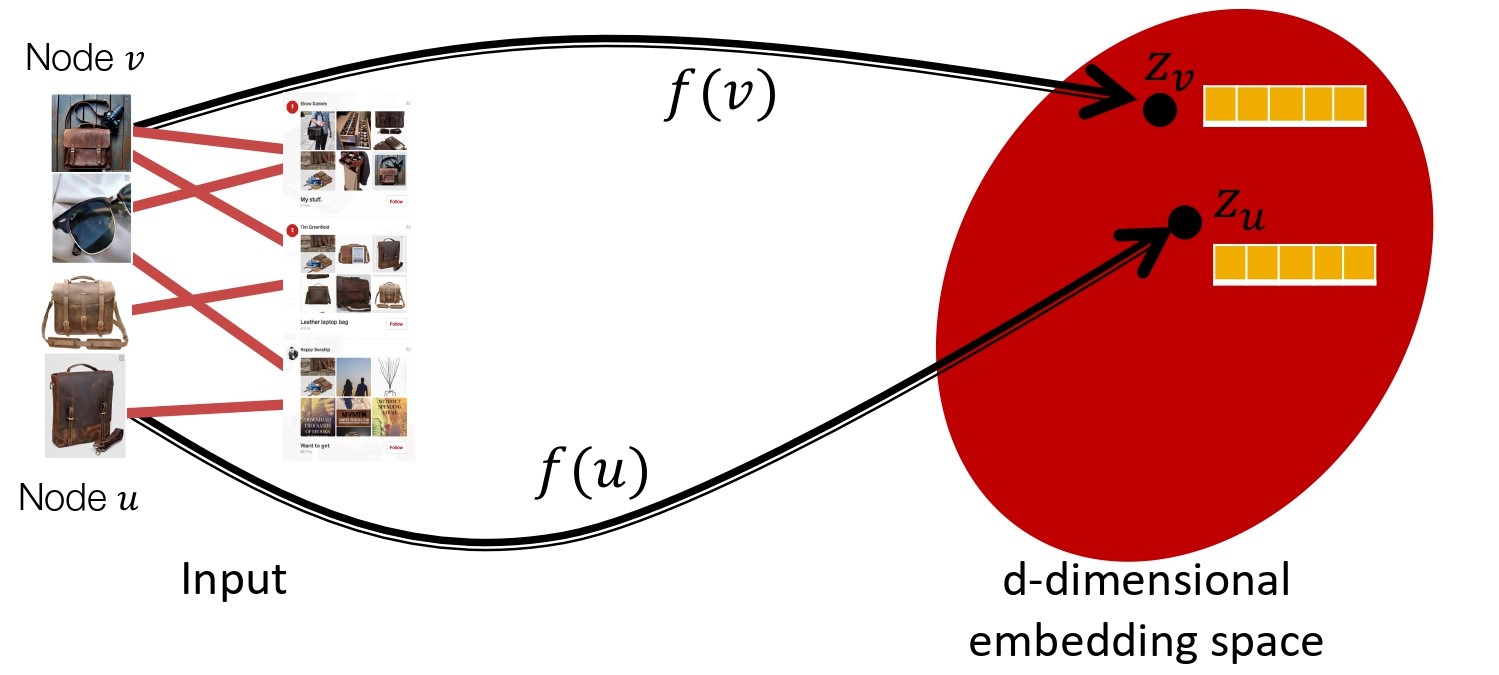
Possiamo utilizzare il grafo bipartito di pinterest per prendere in prestito le informazioni dai nodi vicini nel grafo (immagini/pin), e generare incorporamenti per i nodi e risolvere ciò che è simile e ciò che è diverso.

Nell’esempio sopra abbiamo detto che la computer vision potrebbe confondersi con le due ringhiere, ma in un grafo i cancelli ed i letti sono raramente vicini, il letto apparirà nel grafo assieme ad altri letti, così come la recinzione del giardino apparirà assieme ad altre recinzioni.

Quindi possiamo utilizzare queste informazioni per distinguere i due oggetti.

Gli incorporamenti dei pin possono essere essenziali per varie attività come la raccomandazione di pin, classificazione, raggruppamento, e per offrire servizi come la ricerca, shopping e annunci.

**Incorporamenti dei nodi**

****

Nella figura qui sopra, abbiamo un grafo bipartito in cui ci sono le immagini (nodi) che appartengono alle collezioni.

Il nostro obiettivo è mappare i nodi in uno spazio di incorporamento d-dimensionale tramite una funzione di Encoder, tale che i nodi che sono correlati sono mappati vicini.

Vediamo un task per capire meglio.

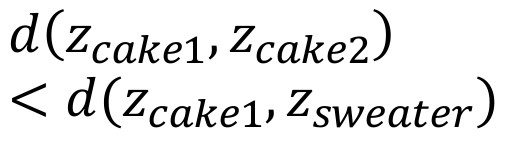
**Task**

In questo task dobbiamo raccomandare, consigliare agli utenti i pin correlati.

Per esempio se un utente sta guardando una torta, perché è alla ricerca di una torta in particolare, la successiva immagine che potremmo raccomandargli sarebbe quella di un’altra torta, e non l’immagine di un maglione come vediamo in figura.

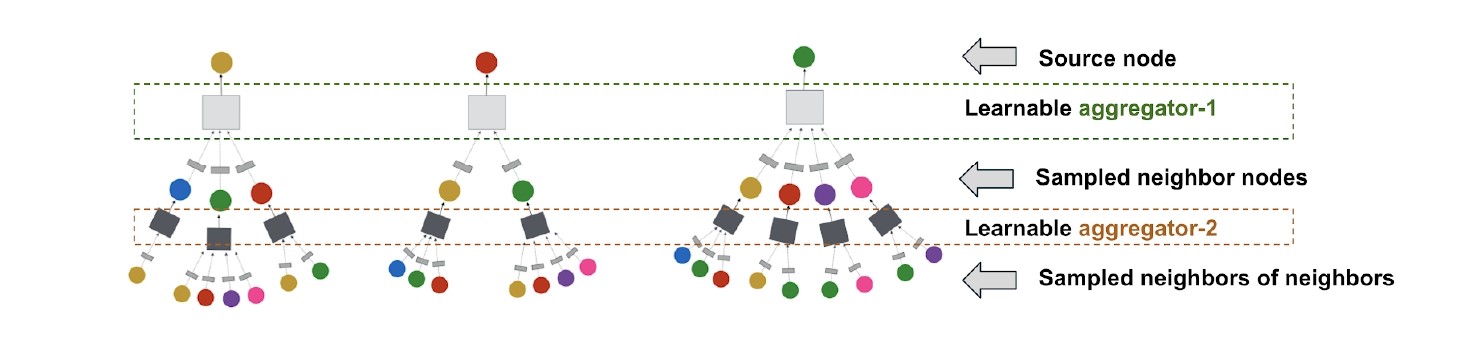


Per farlo dobbiamo calcolare gli incorporamenti dei nodi, e gli incorporamenti della torta 1 e della torta 2 dovranno essere più vicini tra loro rispetto gli incorporamenti della torta 1 e del maglione.



Per calcolare gli incorporamenti delle immagini (nodi) srotoliamo il grafo bipartito in diversi livelli per creare un grafo computazionale e propagare ed aggregare le informazioni dai nodi vicini come abbiamo già visto.

Così facendo creiamo un incorporamento per una data immagine.



Non è necessario l’intero grafo durante la fase di addestramento.

Creiamo i grafi computazionali con alcuni nodi della rete.

**PinSAGE: Esperimenti**

Osserviamo l’accuratezza del modello per fare raccomandazioni in base alla tipologia di creazione degli incorporamenti dei nodi.

Per la valutazione dei metodi usiamo la Mean Reciporcal Rank (MRR) che è un indice statistico che produce una lista di possibili risposte ad una interrogazione (query), ordinate per probabilità di correttezza.

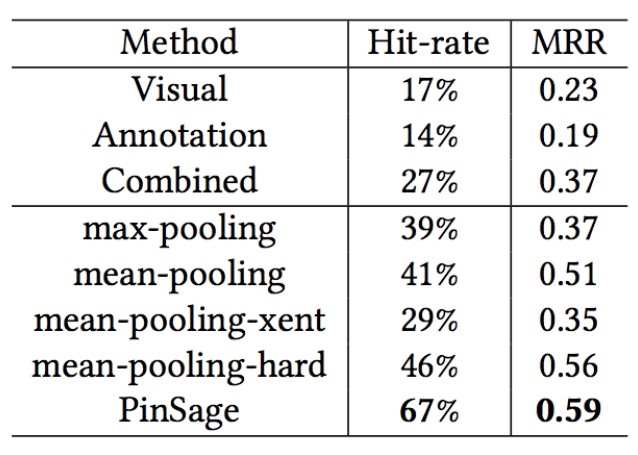
**Visuale:** Utilizziamo solamente l’informazione visiva fornita da una rete neurale convoluzionale (CNN) ed usiamo i nodi più vicini nello spazio di embedding per le raccomandazioni.

**Annotation:** Creiamo gli incorporamenti dei nodi vicini basandoci solo sui tag tramite il metodo Word2vec.

**Combinato:** Concateniamo gli incorporamenti ottenuti con i due metodi precedenti ed usiamo gli stessi dati e la stessa funzione di perdita che usiamo in PinSAGE.

Quello che osserviamo è che con il metodo visuale otteniamo un MRR di 0.23, con il metodo annotation otteniamo un calo con un MRR di 0.19, mentre combinandoli otteniamo un incremento fino ad un MRR di 0.37.

Se invece usiamo la GNN PinSAGE otteniamo un miglioramento del 60% del MRR sugli altri metodi, cioè un MRR di 0.59.



Vediamo un paio di esempi:

Data un’immagine come query, quali sono gli incorporamenti più vicini nello spazio di embedding ad essa?

Sopra vediamo le raccomandazioni ottenute con il metodo Viasuale e sotto vediamo le raccomandazioni ottenute con PinSAGE.



Notiamo subito che con PinSage le raccomandazioni ottenute sono tutte immagini di bambole di porcellana molto simili alla query.

Invece notiamo che con il metodo Visual otteniamo alcuni errori, cioè immagini che non sono bambole di porcellana.

Un altro esempio con una query più complessa notiamo come con PinSAGE otteniamo ancora raccomandazioni simili alla query, mentre con il metodo Visual otteniamo molti più errori rispetto alla query precedente.



Il punto è che utilizzando informazioni basate sui grafi possiamo imparare come arricchire l’informazione di un dato nodo usando le informazioni provenienti dai nodi vicini, e ciò migliora gli incorporamenti che possiamo creare.

**Conclusione**

Negli ultimi anni, le GNN sono diventate strumenti potenti e pratici per qualsiasi problema che può essere modellato mediante grafi. Inoltre con le GNN abbiamo un modello di ML che può essere applicato sui dati strutturati in rete, che sono sempre più ricorrenti. Questo progresso è dovuto ai progressi in potenza espressiva, flessibilità del modello e algoritmi di addestramento.