

# Apprendimento sui grafi

# Il Learning come Problema di Ottimizzazione

- Nel contesto del machine learning e in particolare del deep learning, l'**apprendimento** può essere formulato come un **problema di ottimizzazione**: l'obiettivo è trovare i **parametri (pesi)** di un modello che **minimizzano una funzione di perdita**, cioè una misura dell'errore tra le predizioni del modello e i valori reali.
- **Formulazione generale**

Data una funzione  $f(x; \theta)$  che rappresenta il modello, dove:

- $x$ : input (es. immagine, testo, dati numerici),
- $\theta$ : parametri del modello (pesi),
- $y$ : etichetta reale o valore target,

il nostro obiettivo è:

$$\theta^* = \arg \min_{\theta} \mathcal{L}(f(x; \theta), y)$$

Dove  $\mathcal{L}$  è la **funzione di perdita** (loss function), come ad esempio MSE (Mean Squared Error) per regressione o *cross-entropy* per classificazione.

# Funzioni di attivazione

## ■ Sigmoide

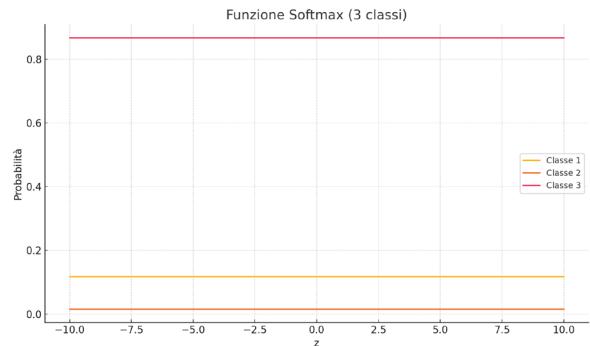
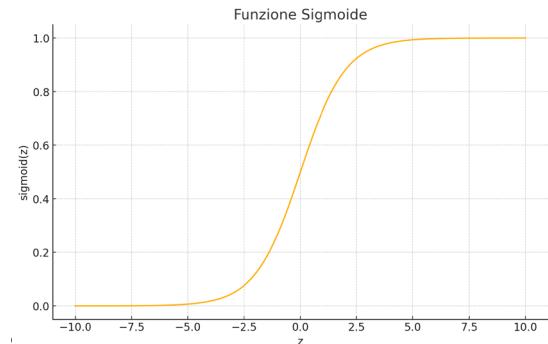
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Output: tra 0 e 1.
- Usata per classificazione binaria (logistic regression).

## ■ Softmax

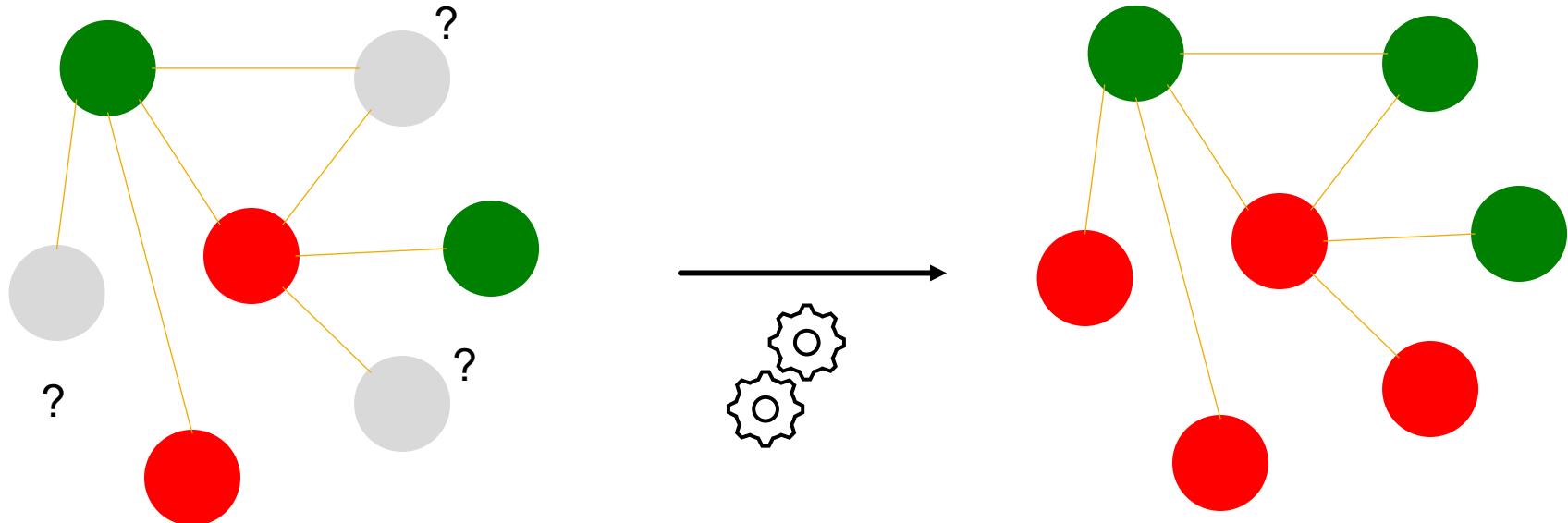
$$softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- Output: distribuzione di probabilità su  $K$  classi.
- Usata per classificazione multclasse.



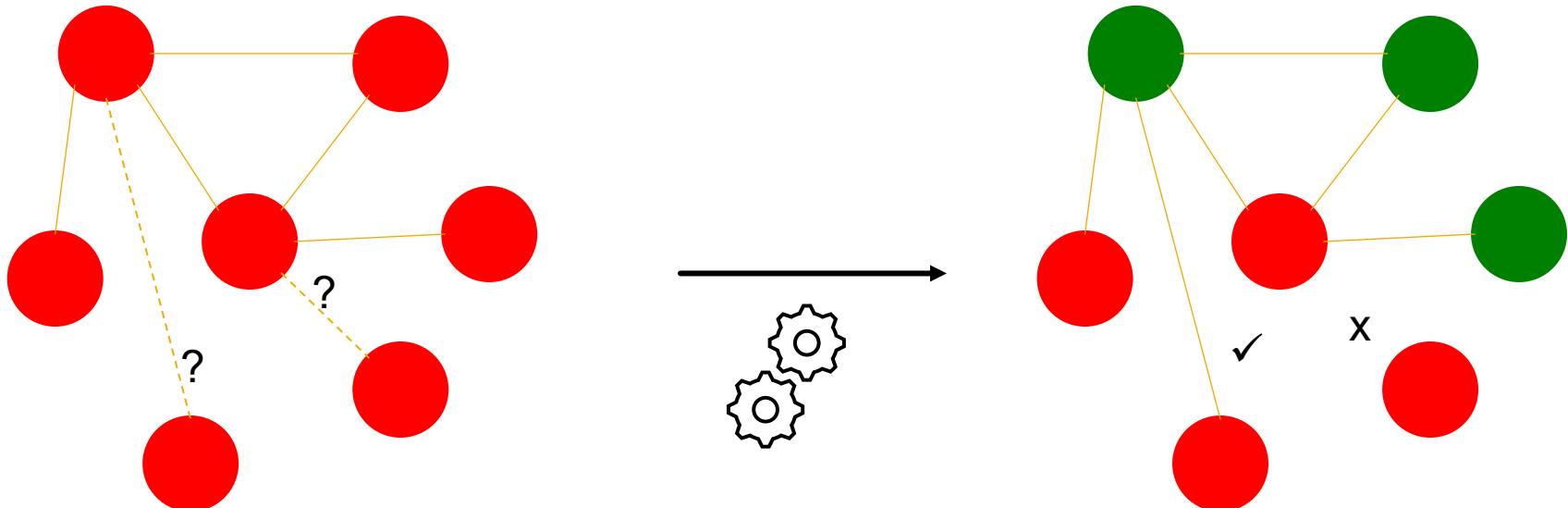
# Machine learning sulle Reti

## ■ Classificazione Nodi



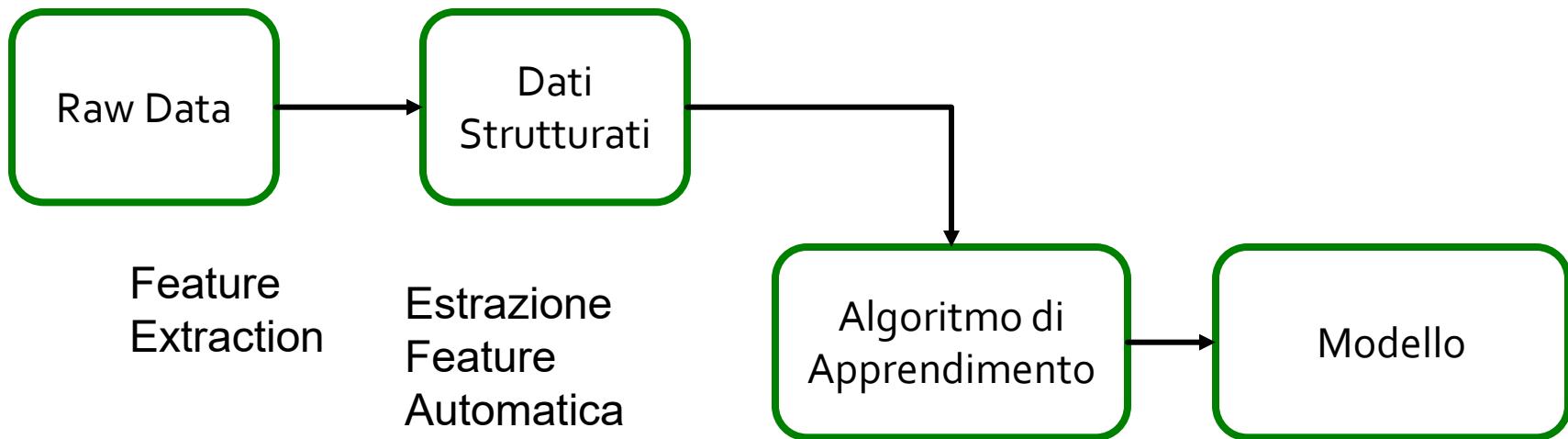
# Machine learning sulle Reti

## ■ Predizione Link



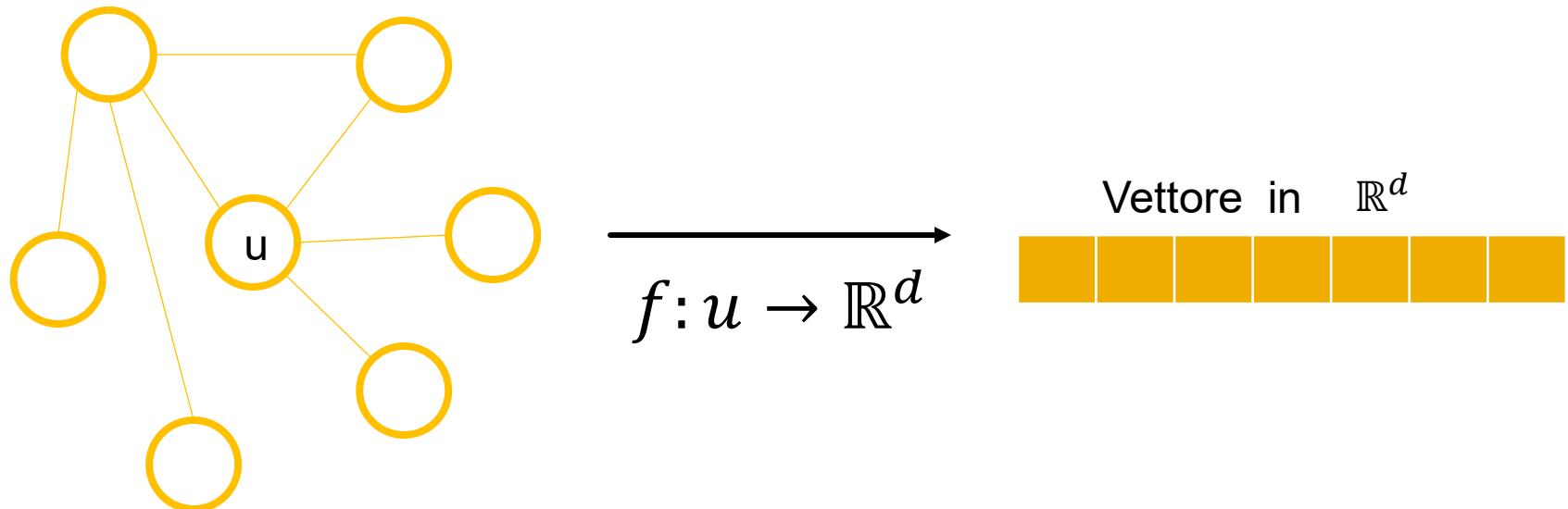
# Ciclo Machine Learning

- Il ciclo di vita del Machine Learning (supervisionato) richiede ogni volta l'ingegnerizzazione delle feature



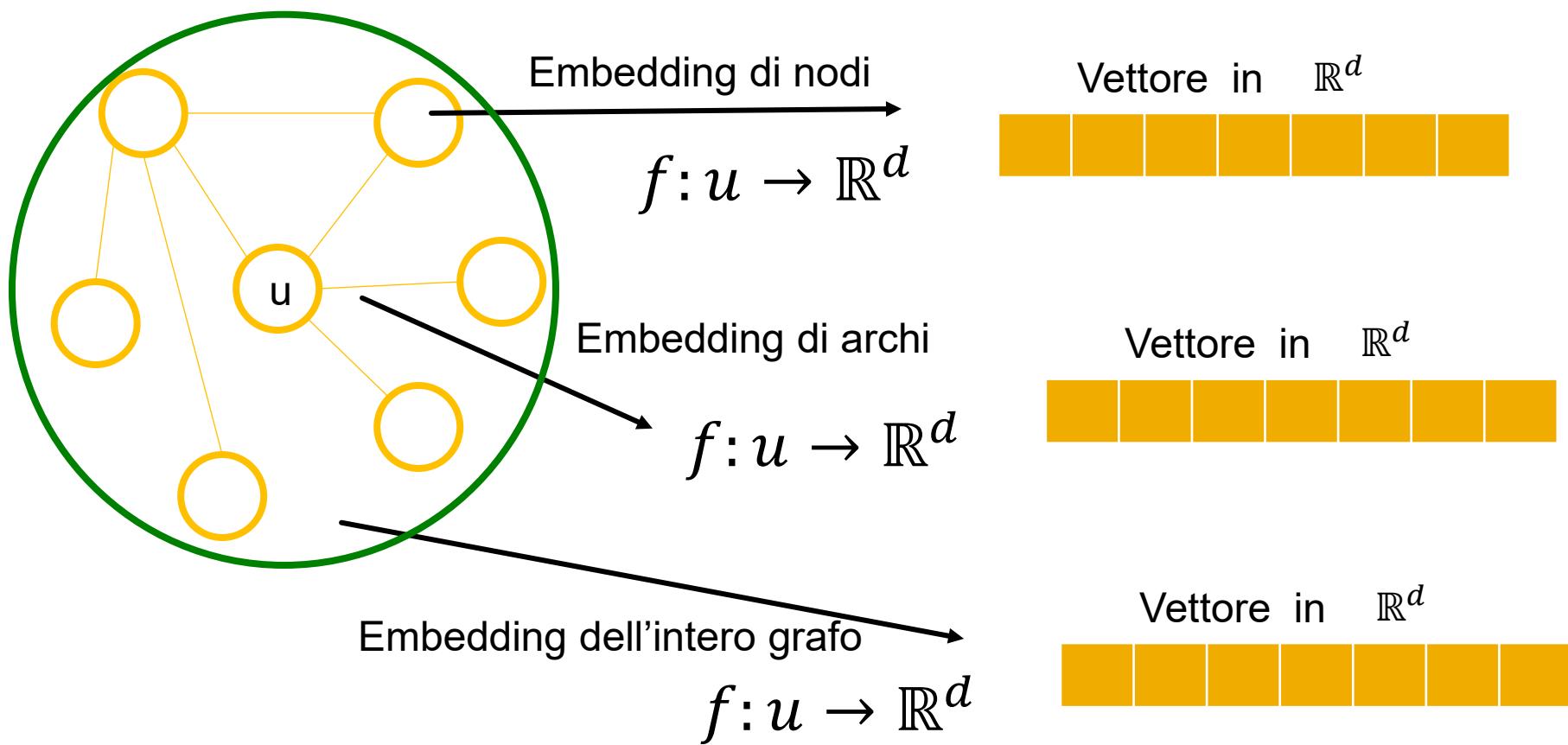
# Apprendimento features nei grafi

- Obiettivo: Apprendere le feature per effettuare il machine learning sui grafi



# Apprendimento features nei grafi

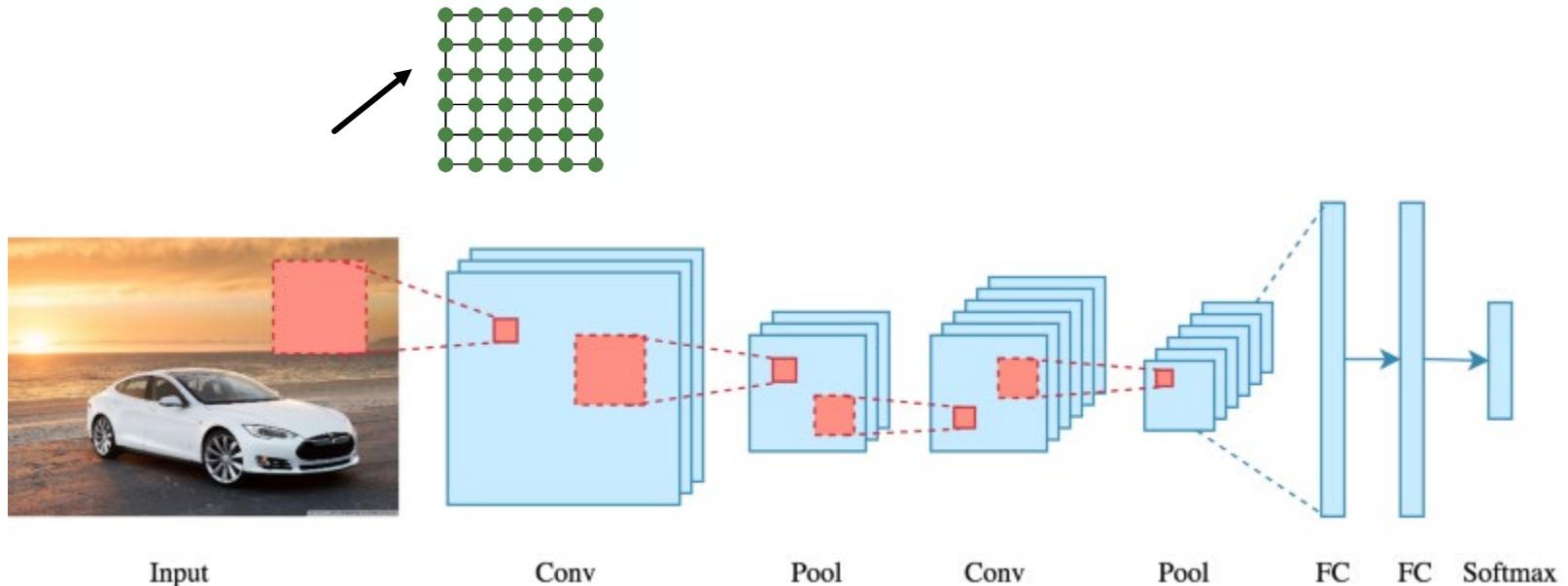
- Obiettivo: Apprendere le feature per effettuare il machine learning sui grafi



- Mappare ogni nodo in uno spazio a basse dimensioni
  - La similarità nello spazio di embedding conserva la similarità nella rete
  - Codificare le informazioni della rete e generare una rappresentazione per i nodi
  - Es:
    - Grafo → Matrice di adiacenza → Dimensioni Latenti → Applicazioni

# Perché è un task difficile

- Esempio di deep learning
  - CNN per immagini a dimensione fissa (griglie)

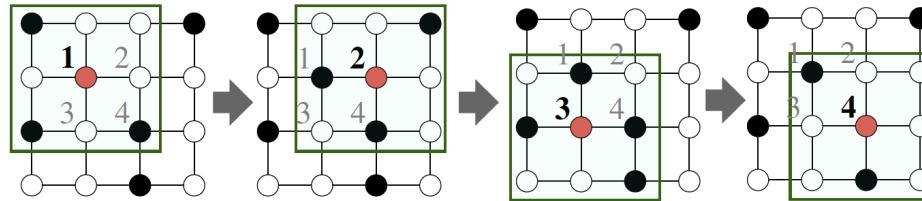


- RNN o word2vec per testi o sequenze



# Perché è un task difficile

- Le network sono complesse
  - Struttura topologica complessa



- Non c'è un ordine o punti di riferimento
- Dinamiche
- Multimodali
- Ecc.

- Supponiamo di avere un grafo G
  - $V$  insieme dei vertici
  - $A$  matrice di adiacenza (binaria)
  - Non ci sono ulteriori features

# Centralità del Nodo

- **Centralità degli autovettori**

$$Ax = \lambda x$$

Primo autovalore  $\lambda_{max}$  e primo autovettore  $x_{max}$  usato come centralità

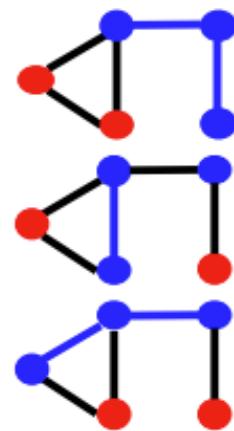
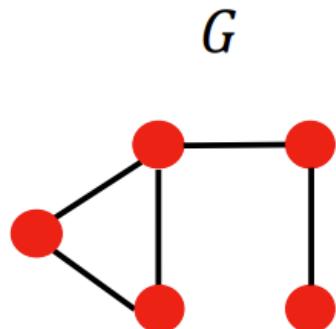
- **Betweenness centrality**

$$c_v = \sum_{s \neq t \neq v} \frac{\text{\#shortest path tra } s \text{ e } t \text{ che passano da } v}{\text{\#shortest path tra } s \text{ e } t}$$

- **Closeness centrality**

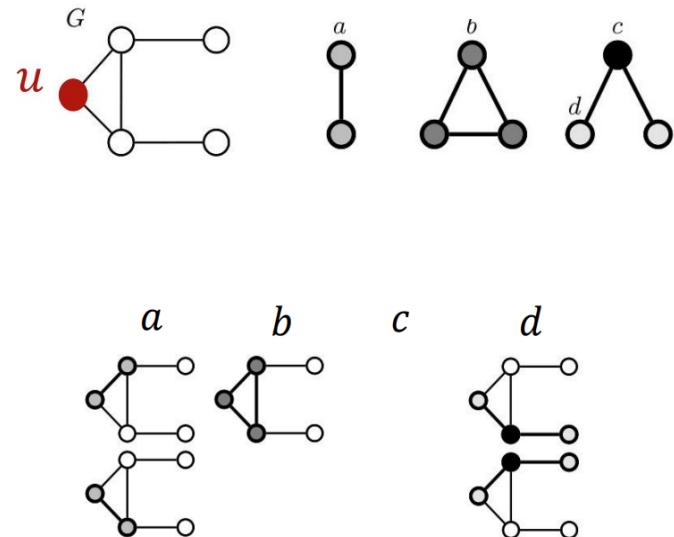
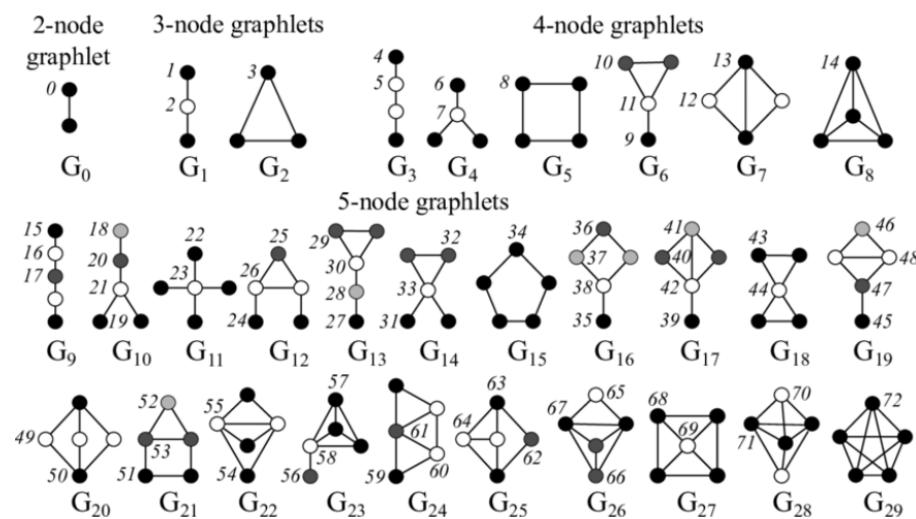
$$c_v = \frac{1}{\sum_{u \neq v} \text{lunghezza shortest path tra } u \text{ e } v}$$

- Subgraph isomorphism (indotto)
- Problema NP-Hard



# Graphlet

- Graphlet: piccoli sottografi che descrivono la topologia del vicinato di un nodo
- Se consideriamo le graphlet di dimensione 2-5 otteniamo un vettore a 73 dimensioni.
- Lo chiamiamo Graphlet Degree Vector (GDV)



$$\text{GDV}(u) = [2, 1, 0, 2]$$

# Graph Kernel

- Un graph kernel è una funzione che quantifica la similarità tra due grafi mappandoli in uno spazio ad alta dimensione, consentendo l'applicazione di metodi di apprendimento automatico basati su kernel

# Metodi kernel

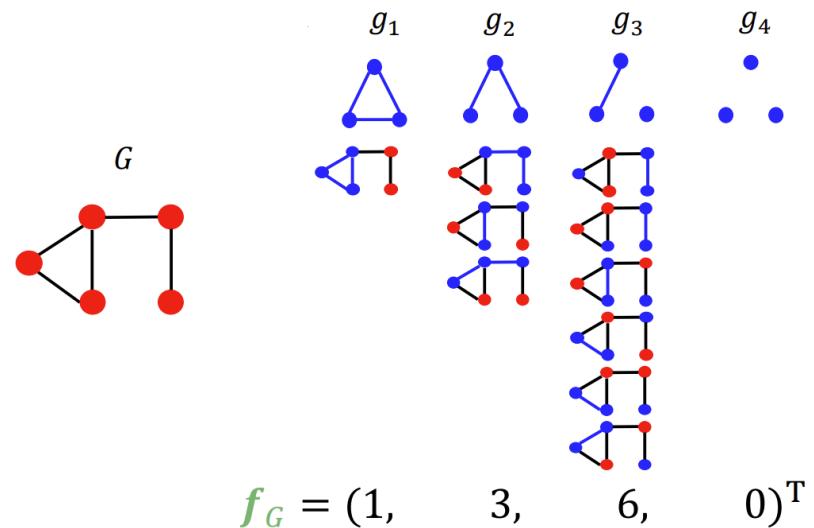
- Usati nei modelli di ML tradizionali
- Idea: disegnare un kernel invece di estrarre un vettore di features
- Kernel:  $K(G, G') \in R$  misura la similarità tra i dati
- Matrice Kernel:  $\mathbf{K}(K(G, G'))_{G,G}$ , sempre semidefinita positiva (autovalori positivi).
- Esiste una rappresentazione delle features  $\phi()$  in modo tale che:  $K(G, G') = \phi(G)^T \phi(G')$

# Due Kernel

- **Graphlet Kernel**
- **Weisfeiler Lehman Kernel**
- Altri kernel
  - Random walk
  - Shortest path graph kernel
  - ecc

# Graphlet kernel

- Dato un grafo  $G$  e una la dimensione delle graphlet  $k$  definiamo  $G_k(g_1, g_2, \dots g_{n_k})$  la lista di graphlet e  $f_G \in R^{n_k}$  vettore contenente il conteggio delle graphlet di dimensione  $k$  presenti in  $G$



# Graphlet kernel

- Il graphlet kernel e' calcolato come il prodotto scalare dei due vettori

$$K(G, G') = f_G^T f_{G'}$$

Normalizzare la funzione  $f_G$

$$h_G = \frac{f_G}{\sum_i f_G(i)}$$

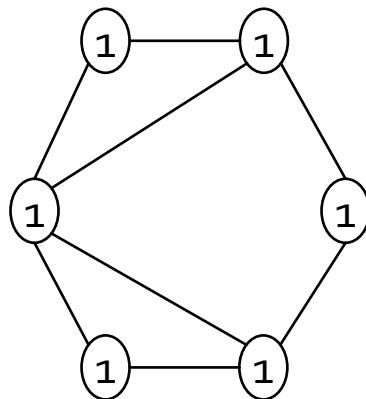
$$K(G, G') = h_G^T h_{G'}$$

Computazionalmente pesante, dato un grafo con  $n$  nodi l'enumerazione richiede  $n^k$

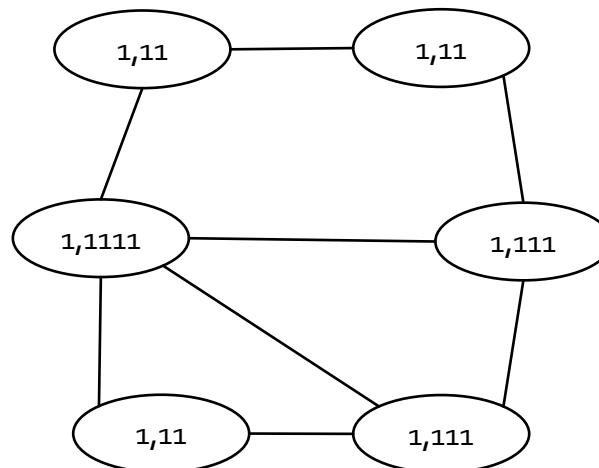
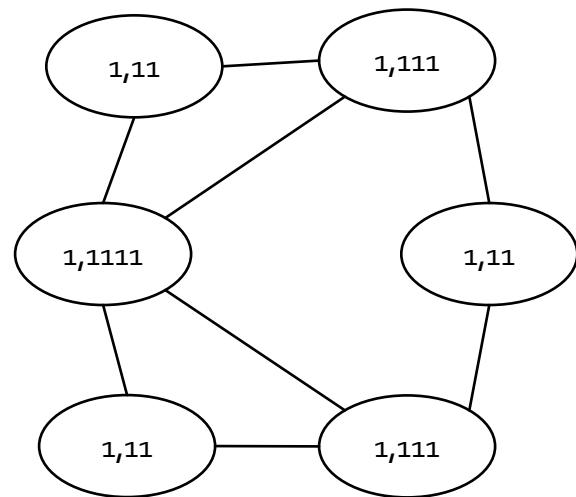
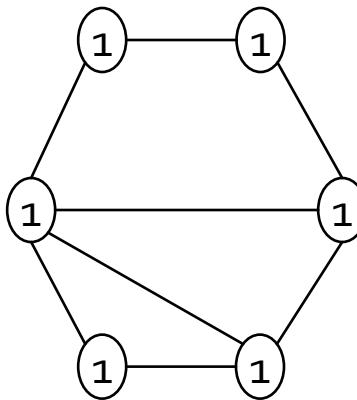
# Weisfeiler-Lehamn Kernel

- Costruire un descrittore di graph feature  $\phi$  efficiente
- Usare iterativamente il vicinato per arricchire il vocabolario di un nodo.
- Algoritmo di color refinement
- Dato un grafo  $G$  con un insieme di nodi  $V$ 
  - Assegnare un colore iniziare ad ogni nodo  $c^{(0)}(v)$ .
  - Iterativamente raffinare i colori
    - $c^{(k+1)}(v) = \text{HASH} \left( \left\{ c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)} \right\} \right)$
    - Hash mappa differenti valori in differenti colori
    - Dopo K step otteniamo  $c^{(K)}(v)$  che sintetizza la struttura del vicinato a K-hop

$G$

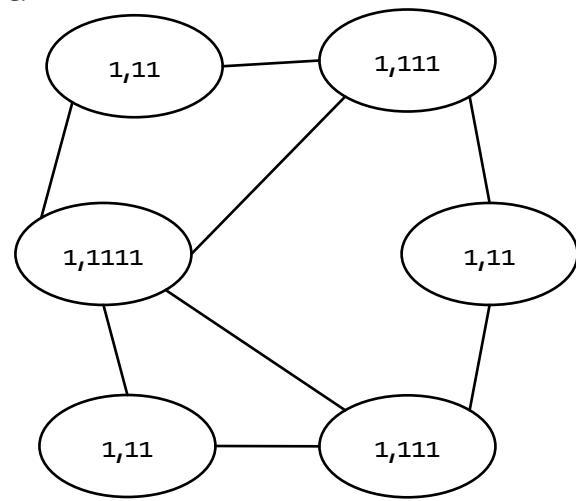


$G'$

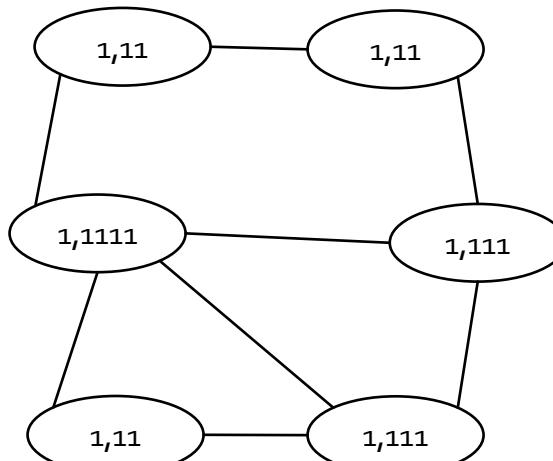


$1,11 \rightarrow 3$   
 $1,111 \rightarrow 4$   
 $1,1111 \rightarrow 5$

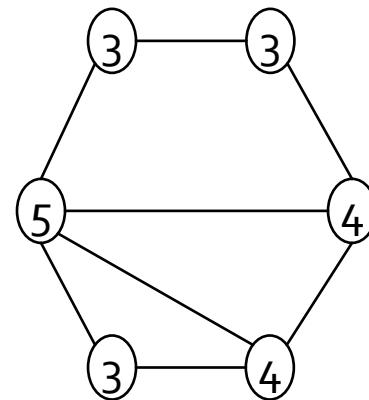
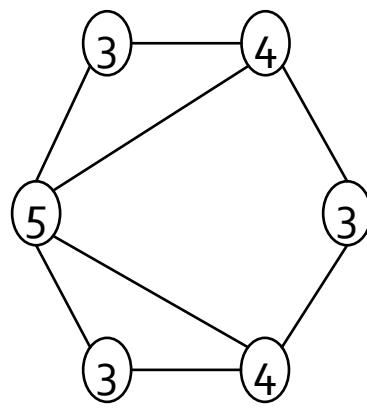
$G$

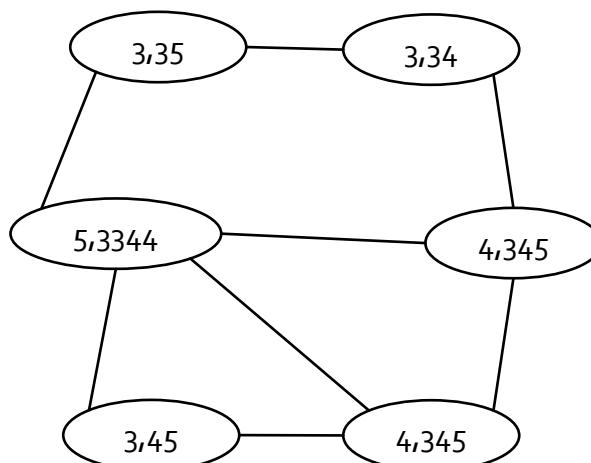
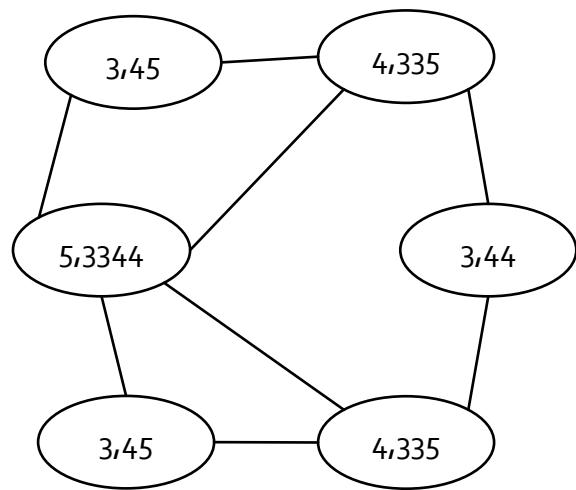
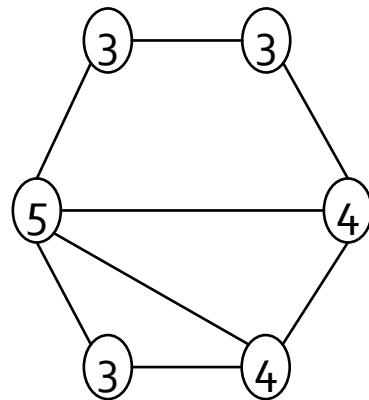
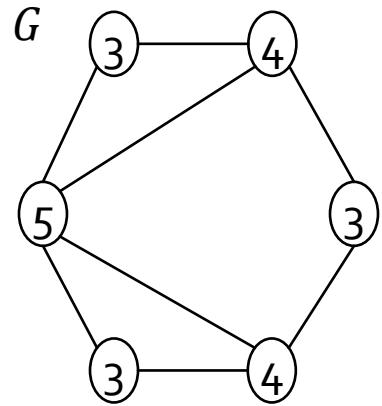


$G'$

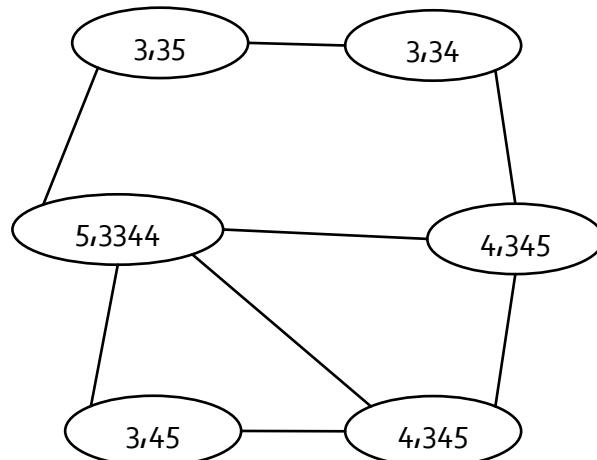
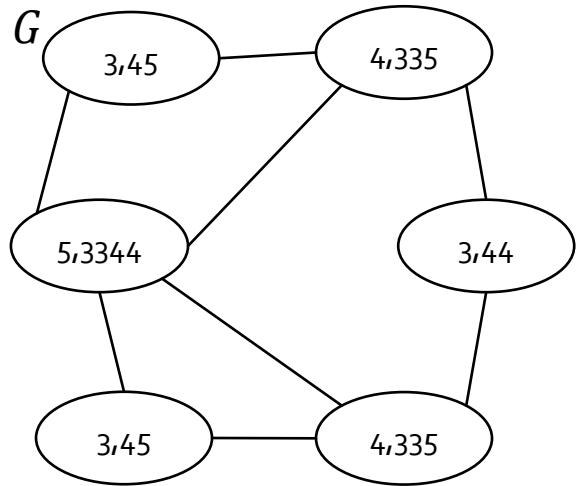


$$\begin{aligned}1,11 &\rightarrow 3 \\1,111 &\rightarrow 4 \\1,1111 &\rightarrow 5\end{aligned}$$



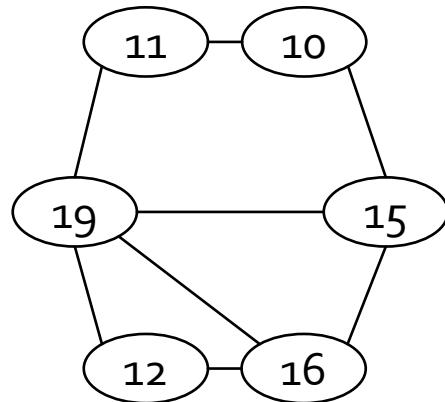
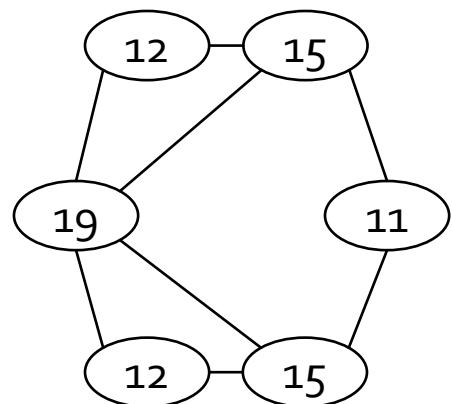


$3,34 \rightarrow 10$   
 $3,44 \rightarrow 11$   
 $3,45 \rightarrow 12$   
 $4,335 \rightarrow 15$   
 $4,345 \rightarrow 16$   
 $5,3344 \rightarrow 19$



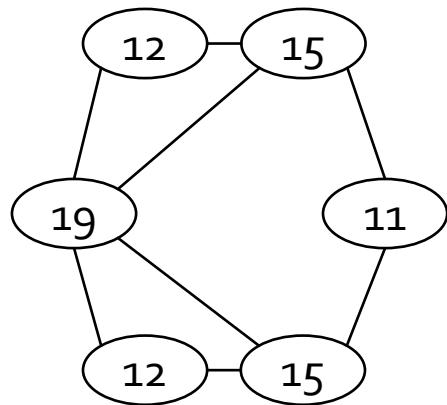
$G'$

- $3,34 \rightarrow 10$
- $3,35 \rightarrow 11$
- $3,44 \rightarrow 11$
- $3,45 \rightarrow 12$
- $4,335 \rightarrow 15$
- $4,345 \rightarrow 16$
- $5,3344 \rightarrow 19$

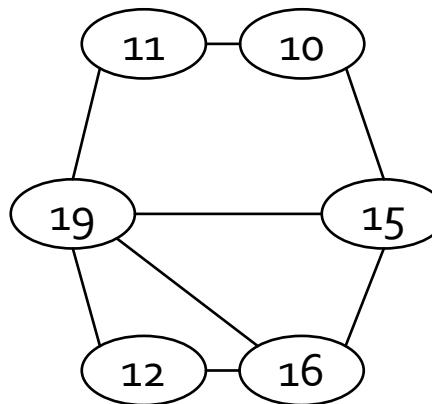


$1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19$   
 $G$

$G$



$G'$



1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19

$$\phi(G) = (6,0,3,2,1,0,0,0,0,0,1,2,0,0,2,0,0,0,1)$$

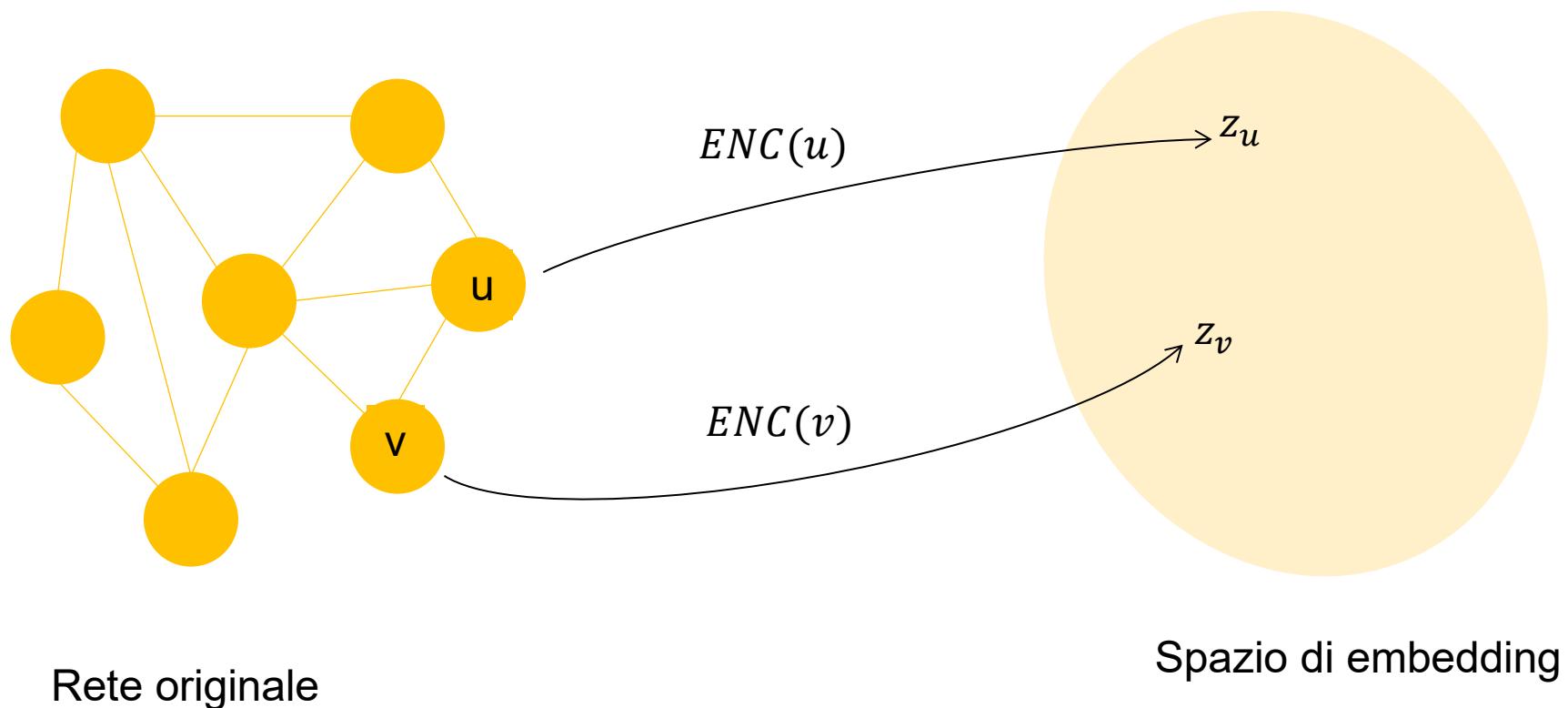
$$\phi(G') = (6,0,3,2,1,0,0,0,0,0,1,1,1,0,0,1,1,0,0,1)$$

$$K(G, G') = \phi(G)\phi(G') = 56$$

# Embedding di nodi

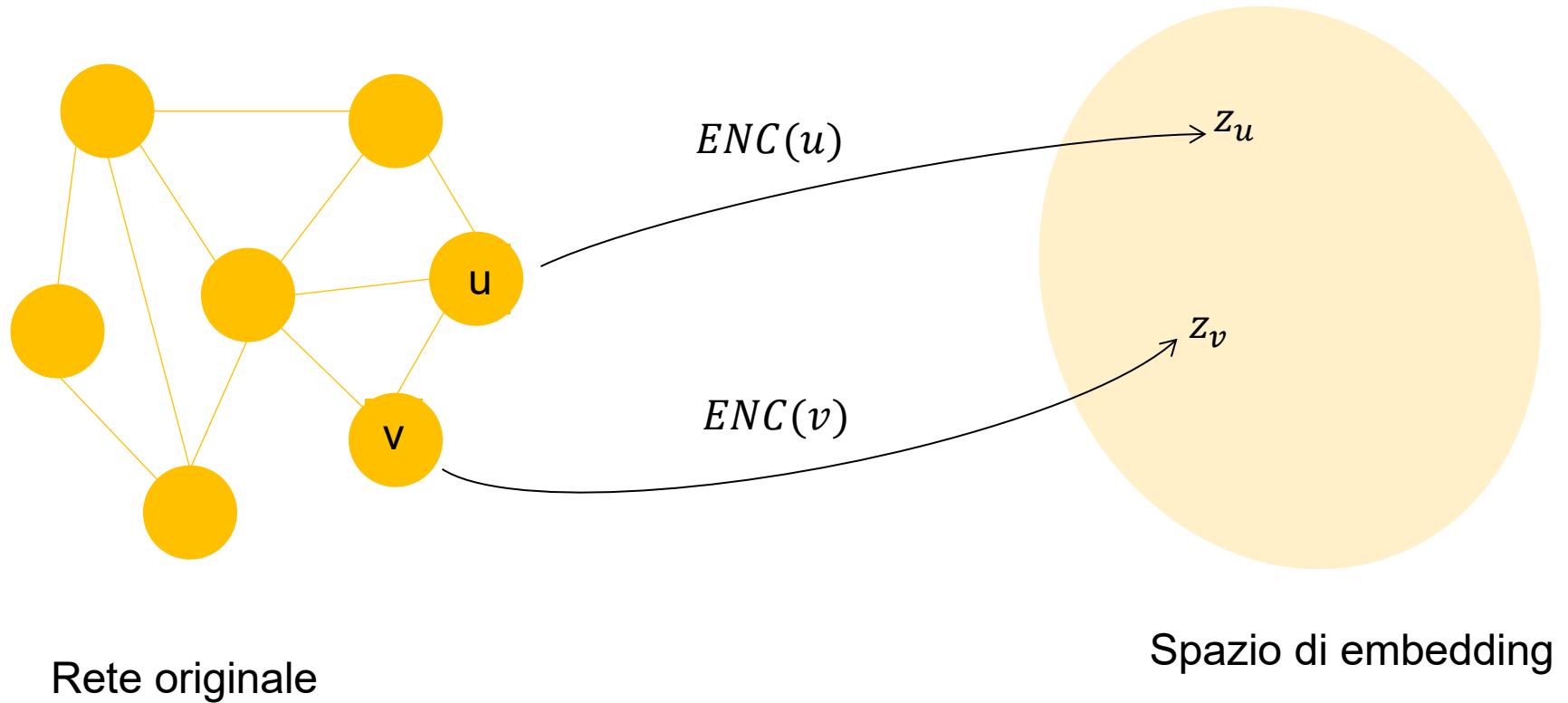
# Embedding dei nodi

- Obiettivo è codificare i nodi in modo tale che la similarità nello spazio di embedding approssimi quella nella rete originale



# Embedding dei nodi

- Obiettivo:  $\text{sim}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$



# Task

- Definire un encoder (mapping dai nodi all'embedding)
- Definire una funzione di similarità tra i nodi
- Ottimizzare i parametri in modo tale che l'encoder

$$\text{sim}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$$

# Componenti

- **Encoder:** mappa ogni nodo in uno spazio a basse dimensioni

$$ENC(v) = \mathbf{z}_v$$

- Similarità: specifica le relazioni tra lo spazio originale e quello dell'embedding

$$\text{sim}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$$

# Codifica «Shallow»

- L'encoder è un embedding-lookup

$Z =$


Una colonna per nodo

# Come si definisce la similarità tra i nodi?

- Due nodi dovrebbero avere un embedding simile se:
  - Sono connessi?
  - Condividono vicini?
  - Hanno ruoli strutturali simil?
  - Ecc

Approccio basato sulla random  
walk

# Notazione

- Vettore  $\mathbf{z}_u$ 
  - Embedding del nodo u
- Probabilità  $P(v|\mathbf{z}_u)$ :
  - Probabilità predetta di visitare il nodo v in una random walk che parte dal nodo u

Funzioni non linerari usate per predire le probabilità:

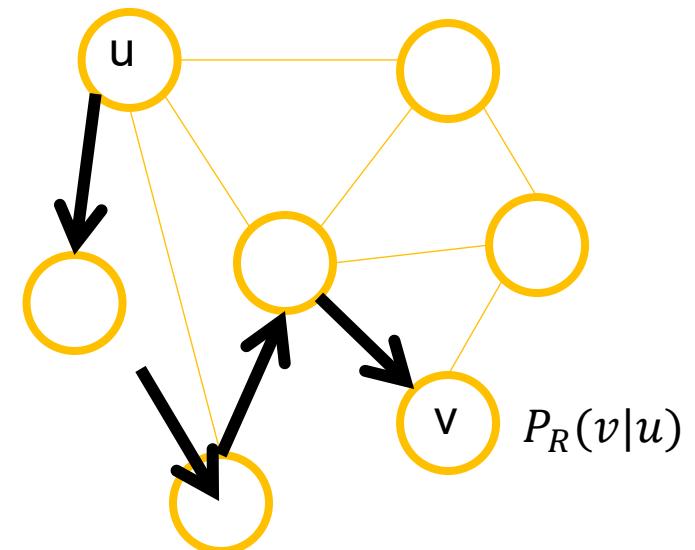
- **Softmax**
  - Convertire un vettore di k numeri reali (predizioni) in k probabilità la cui somma è pari a 1:  $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1..k} e^{z_j}}$
- **Sigmoide**
  - Funzione per trasformare valori reali in un intervallo (0,1):
$$S(x) = \frac{1}{1+e^{-x}}$$

# Random walk Embedding

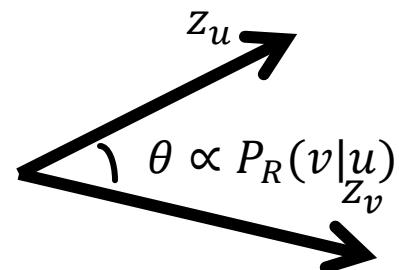
- Dato un grafo e un nodo di partenza selezioniamo un suo vicino in modo random, ci spostiamo sul vicino e ripetiamo il processo
  - La sequenza di nodi è una random walk

$z_u^T z_v \approx$  Probabilità che u e v co-occorrano in una random walk sulla rete

- Stimare la probabilità di visitare un nodo  $v$  in una random walk partendo da un nodo  $u$  usando una strategia R



- Ottimizzare l'embedding per codificare queste statistiche delle random walk  
(nel nostro caso la similarità del coseno  $\cos(\theta)$ )



# Perche' la random walk

- **Espressività:** definizione stocastica flessibile della similarità tra i nodi, che incorpora informazioni sia locali che di ordine più elevato
- **Idea:** se una random walk che parte dal nodo  $u$  visita  $v$  con alta probabilità  $u$  e  $v$  sono simili.
- **Efficiente:** non necessita di considerare tutte le coppie quando si fa il training

# Feature learning non supervisionato

- **Intuizione:** trovare un embedding dei nodi in uno spazio d-dimensionale che preserva la similarità
- Idea: Apprendere l'embedding tale che i nodi nelle vicinanze siano vicini nell'embedding
- Dato un nodo  $u$ , come definiamo i nodi vicini?
  - $N_R(u)$  vicinato di  $u$  ottenuto con una strategia R

# Feature learning non supervisionato

- Dato  $G=(V,E)$
- Trovare un mapping  $f: u \rightarrow \mathbb{R}^d, f(u) = z_u$
- Log-likelihood:

$$\max_f \sum_{u \in V} \log P(N_R(u)|z_u)$$

- Eseguiamo una random walk breve con lunghezza fissa usando una strategia R
- Per ogni nodo  $u$  collezioniamo  $N_R(u)$ , il multiset di nodi visitati con le random walk da  $u$
- Ottimizziamo l'embedding:

$$\max_f \sum_{u \in V} \log P(N_R(u) | z_u)$$

Maximum likelihood

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|z_u))$$

Parametriziamo  $P(v|z_u)$  usando la funzione softmax:

$$P(v|z_u) = \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)}$$

# Ottimizzazione della random Walk

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log \left( \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)} \right)$$

Computazionalmente dispendioso richiede  
tempo  $O(V^2)$

Soluzione approssimata: effettuare il **Negative Sampling**

# Negative sampling

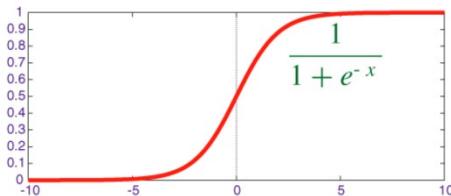
- Tecnica utilizzata principalmente nel contesto dell'apprendimento automatico su grafi e nel natural language processing (NLP),
- **Obiettivo:** rendere più efficiente l'addestramento di modelli che coinvolgono una funzione loss basata su softmax

- Sia  $D=\{(u,v)\}$  l'insieme di coppie di nodi (o parole) positive, cioè quelle che rappresentano relazioni effettive (archi esistenti in un grafo o contesti osservati nel linguaggio).
- Addestrare un modello per distinguere queste coppie da quelle non osservate (negative).
- Il negative sampling: campionare per ogni coppia positiva  $(u,v) \in D$  un insieme di  $k$  coppie negative  $\{(u,v_1^-), \dots, (u,v_k^-)\}$ , dove ciascun  $v_i^-$  è scelto tra i nodi che non sono connessi a  $u$ .
- L'assunzione è che questi siano verosimilmente negativi (cioè non collegati).

$$\log \left( \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)} \right)$$

$$\approx \log(\sigma(z_u^T z_v)) - \sum_{i=1}^k \log(\sigma(z_u^T z_{n_i})), n_i \sim P_V$$

Funzione sigmoide



Invece di normalizzare rispetto a tutti i nodi  
normalizziamo rispetto a k nodi random  $n_i$

# Negative sampling

- Campioniamo  $k$  nodi in base al degree
  - Piu' grande è  $k$  piu' robusta è sa stima
  - Piu' grande è  $k$  piu' aumenta il bias sugli eventi negativi
- $K$  tra 5 e 20 è una buona scelta

# Gradiente discendente

- Dopo aver ottenuto la funzione obiettivo dobbiamo minimizzarla

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|z_u))$$

- Usiamo il gradiente discendente
  - Inizializziamo  $z_i$  a un qualche valore random per ogni  $i$
  - Iteriamo fino ad ottenere la convergenza
    - Per ogni  $i$  calcoliamo la derivata  $\frac{\partial \mathcal{L}}{\partial z_i}$
    - Per ogni  $i$  effettuiamo uno step nella direzione della derivata:  $z_i \leftarrow z_i - \eta \frac{\partial \mathcal{L}}{\partial z_i}$

# Gradiente discendente stocastico

- Invece di valutare il gradiente su tutti i dati, lo valutiamo su ogni singolo dato di training
  - Inizializziamo  $z_i$  a un qualche valore random per ogni  $i$

$$\mathcal{L}^{(u)} = \sum_{v \in N_R(u)} -\log(P(v|z_u))$$

- Iteriamo fino ad ottenere la convergenza:
  - Per ogni  $i$  e per ogni  $j$  calcoliamo la derivata  $\frac{\partial \mathcal{L}^{(i)}}{\partial z_j}$
  - Per ogni  $i$  effettuiamo uno step nella direzione della derivata:  $z_j \leftarrow z_j - \eta \frac{\partial \mathcal{L}^{(i)}}{\partial z_j}$

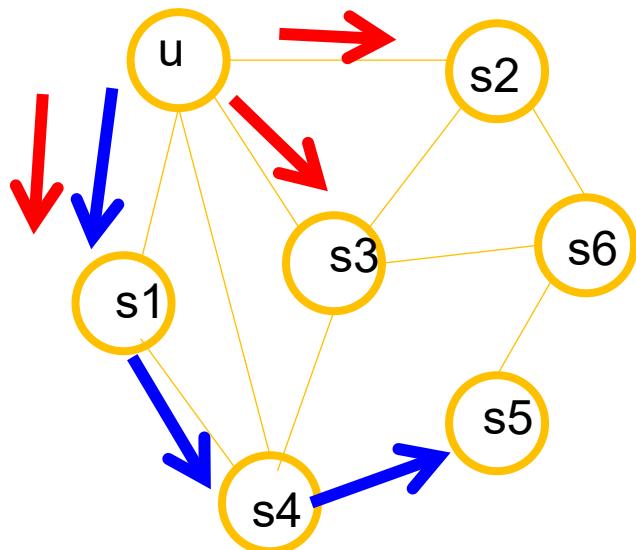
- La nozione di similarità usata fino ad ora è restrittiva..

# node2vec

- Embedding dei nodi simili nella rete vicini nello spazio di features
- Nozione flessibile di vicinato  $N_R(u)$  consente un embedding migliore
- Random walk R del secondo ordine per generare il vicinato  $N_R(u)$  di un nodo u

# node2vec

- Idea: usare una biased random walk flessibile che consente di misurare un tradeoff tra vicinato locale e globale
- Due strategie speculari per definire  $N_R(u)$



*Random walk di lunghezza 3*

$$N_{BFS}(u) = \{S_1, S_2, S_3\}$$

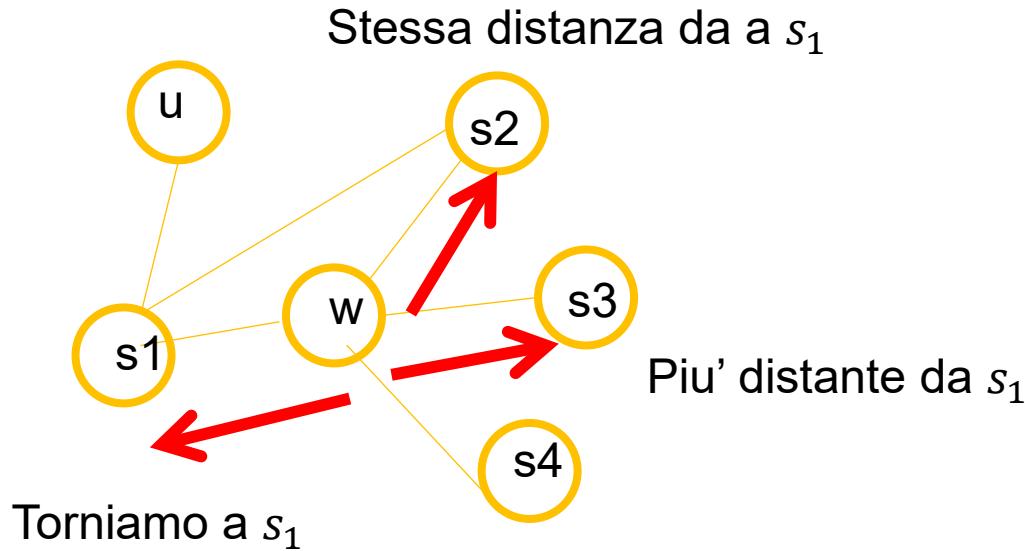
$$N_{DFS}(u) = \{S_1, S_4, S_5\}$$

# Interpolare BFS e DFS

- Due parametri
  - **Parametro di ritorno p**
    - Ritorna ad un nodo già visitato
  - **Parametro in-out q**
    - Muoviti in profondità verso l'esterno (DFS) vs verso l'interno (BFS)
    - q è il rapporto tra DFS e BFS

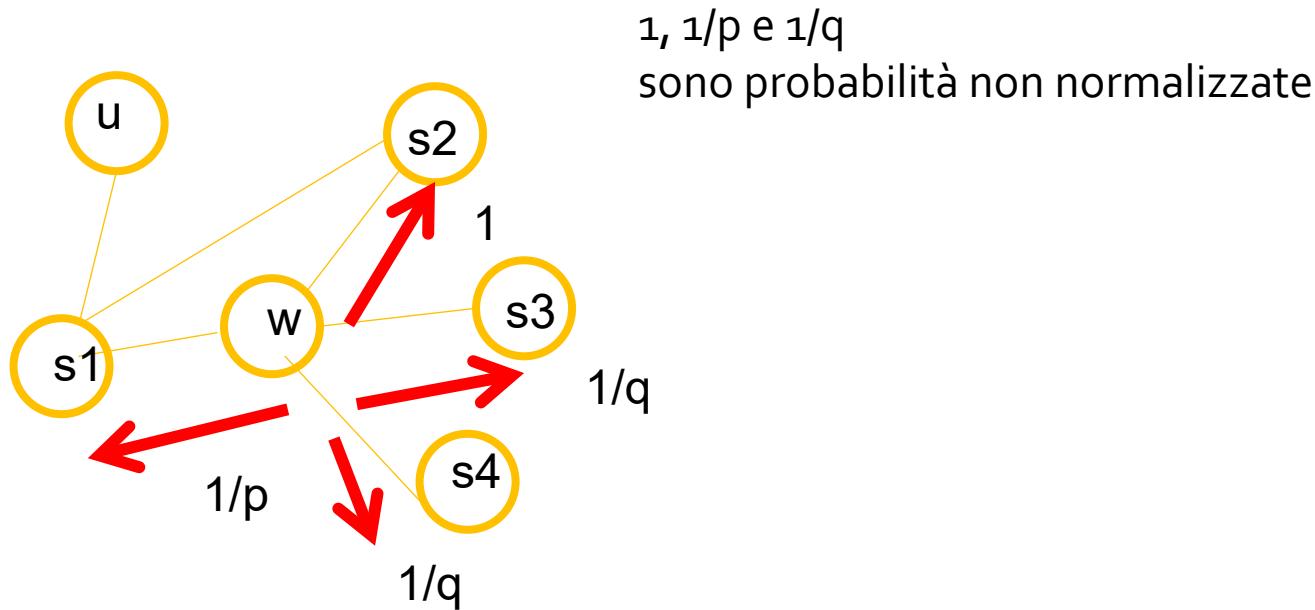
# Biased random walk

- La random walk attraversa l'arco  $(s_1, w)$  i vicini di  $w$  possono essere solamente:



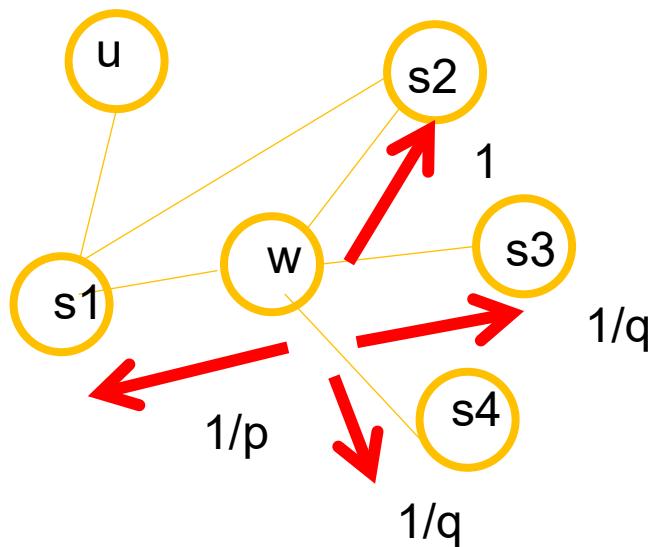
- Idea: ricordiamoci da dove siamo venuti!

- Dove andiamo adesso?
- $p, q$  modellano le probabilità di transizione (p parametro di ritorno, q parametro di allontanamento)



- Dove andiamo adesso?
- $p, q$  modellano le probabilità di transizione (p parametro di ritorno, q parametro di allontanamento)

$1, 1/p$  e  $1/q$   
sono probabilità non normalizzate

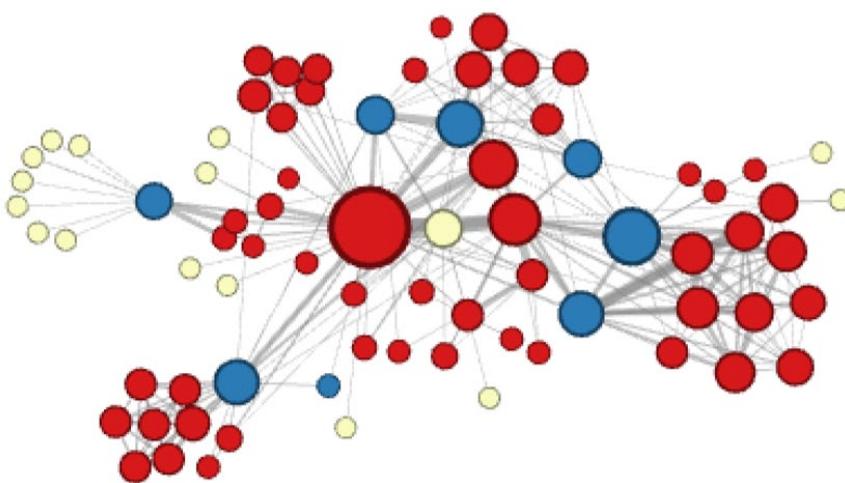
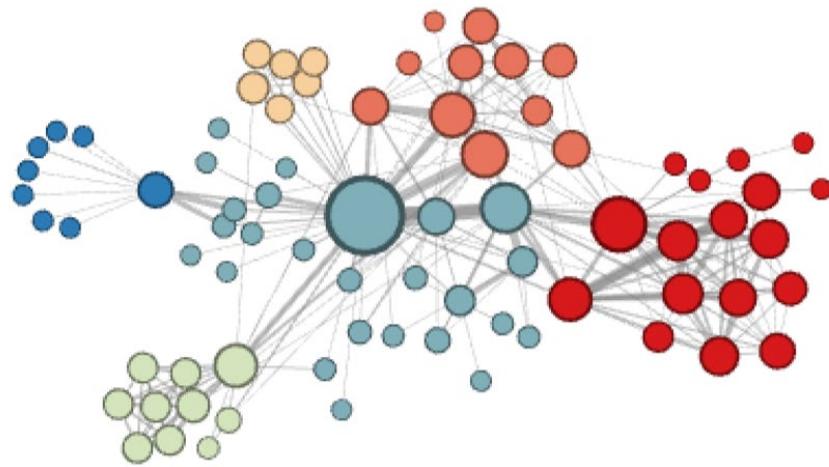


Target t	prob	$\text{dist}(s_1, t)$
$s_1$	$1/p$	0
$s_2$	1	1
$s_3$	$1/q$	2
$s_4$	$1/q$	2

BFS-like walk: valore basso per p  
DFS-like walk: valore basso per q

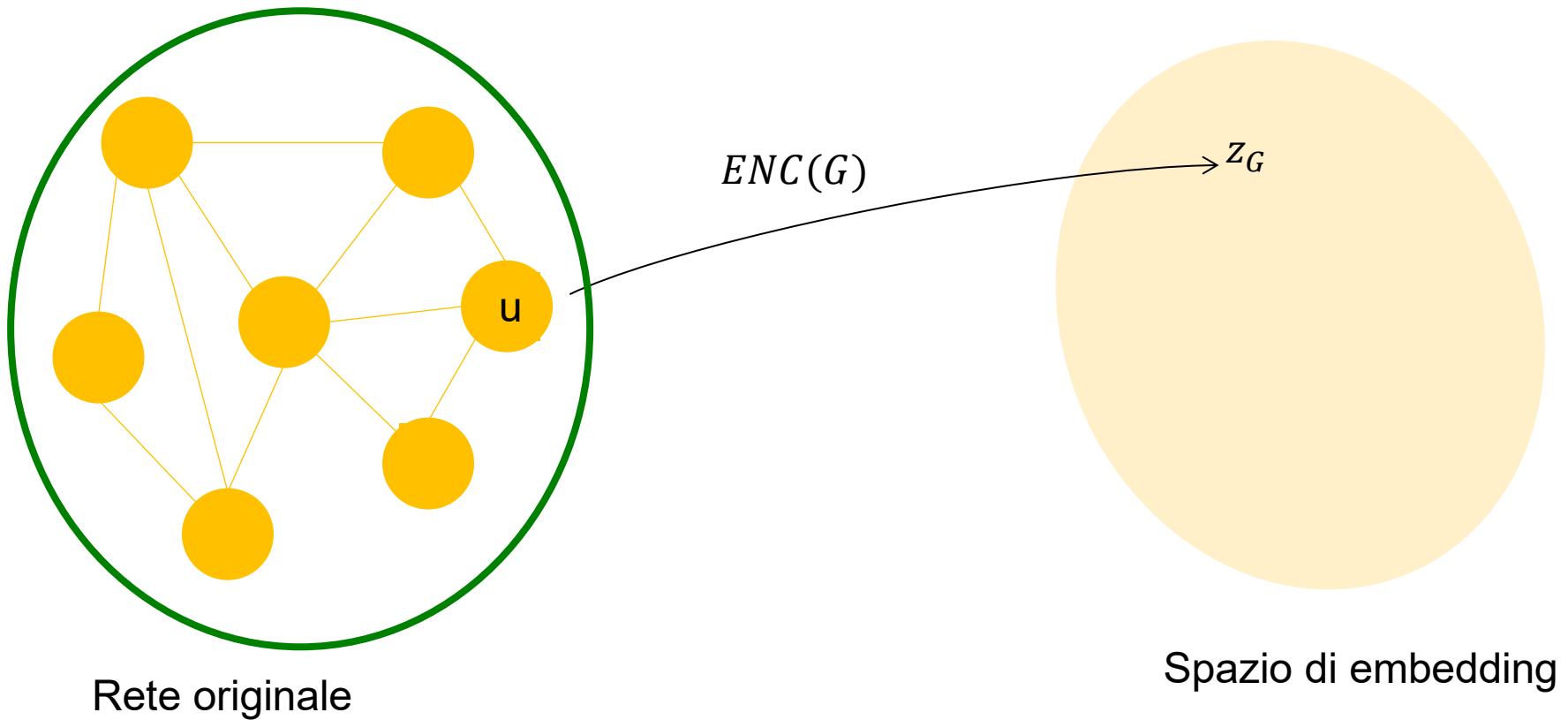
# Algoritmo node2vec

- Calcolare le probabilità della random walk
- Simulare  $r$  random walk di lunghezza  $l$  che partono dal nodo  $u$
- Ottimizzare la funzione obiettivo di node2vec con il gradiente discendente stocastico
- Complessità lineare



# Embedding dell'intero grafo

- Obiettivo è codificare il grafo nello spazio di embedding



# Approccio 1

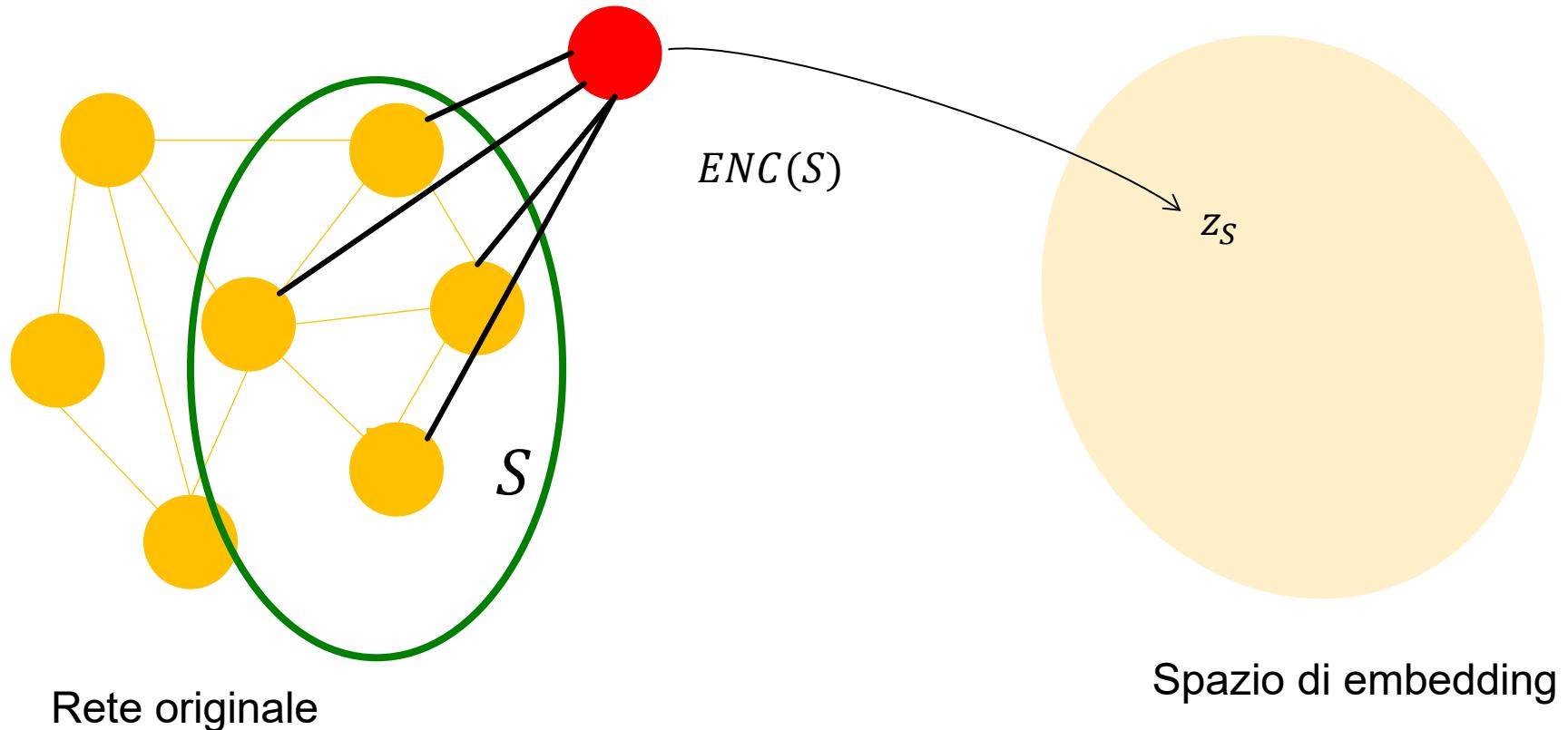
- Eseguire un algoritmo per l'embedding sui nodi del grafo
- Prendere la somma dell'embedding (o la media) dei nodi del grafo  $G$  o del sottografo

$$Z_G = \sum_{v \in G} z_v$$

Usato da Duvenaud et al 2016 per classificare le molecole in base alla loro struttura

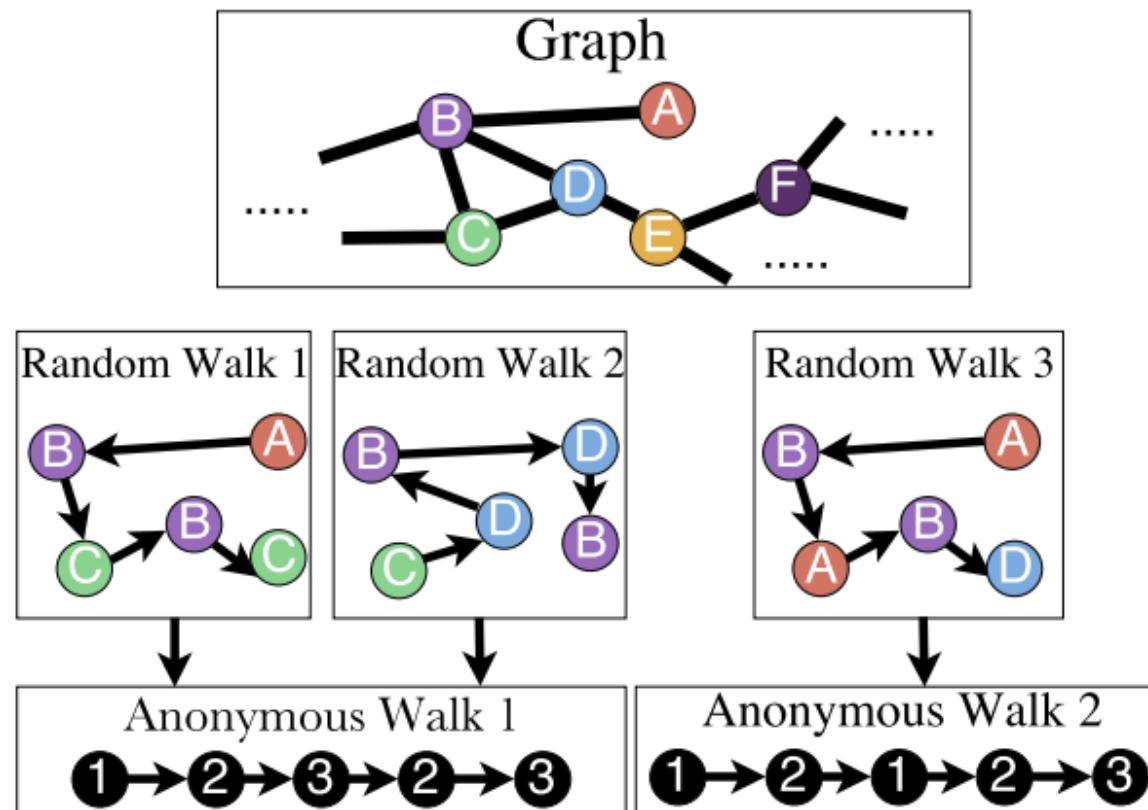
# Approccio 2 (Li et al .2016)

- Introdurre un nodo virtuale per rappresentare il grafo o sottografo della rete e eseguire un metodo standard per l'embedding dei nodi



# Approccio 3: Anonymous walk embedding

- Gli stati nelle anonymous walk corrispondono all'indice del primo nodo visitato.



# Numero delle anonymous walk

- Il numero di anonymous walk cresce esponenzialmente.
- $k=3$  abbiamo 5 anonymous walk
- $a_1 = 111, a_2 = 112, a_3 = 121, a_4 = 122, a_5 = 123$

# Idea 1

- Contare tutte le possibili anonymous walk  $w_i$  di lunghezza  $l$  nel grafo e memorizzare il conteggio.
- Rappresentare il grafo come la distribuzione di probabilità su queste walk
- Es.
  - $l=3$
  - Possiamo rappresentare il grafo come un vettore a 5 dimensioni
  - $Z_G[i] = \text{probabilità dell'anonymous walk } a_i \text{ in G}$

# Idea 2

- Campionare le anonymous walk, il conteggio completo puo' essere pesante
- Sampling consente di approssimare la vera distribuzione
- Quante random walk  $m$  dobbiamo usare?
  - Vogliamo che l'errore sia entro  $\epsilon$  con probabilità  $\delta$

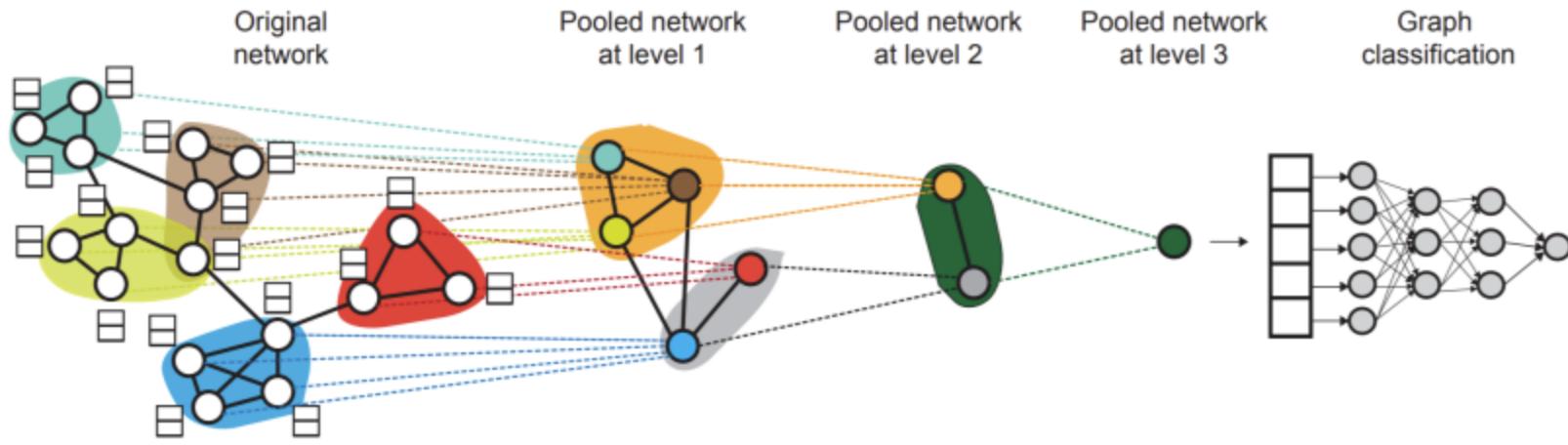
$$m = \left\lceil \frac{2}{\epsilon^2} (\log(2^\eta - 2) - \log(\delta)) \right\rceil$$

Con  $\eta$  il numero totale di anonymous walk di lunghezza  $l$

Esempio:

Ci sono  $\eta = 877$  anonymous walk di lunghezza  $l = 7$ . Se settiamo  $\epsilon = 0.1$  and  $\delta = 0.01$  dobbiamo generare  $m = 53200$  random walk

# Embedding gerarchico



# Come usare l'embedding

- Come usare l'embedding dei nodi  $z_i$ 
  - Clustering di  $z_i$
  - Predizione dell'etichetta di  $i$  sulla base del valore di  $z_i$
  - Predizione di un arco  $(i,j)$  sulla base  $z_i, z_j$ 
    - Vari modi, media, prodotto, differenza tra gli embedding

# Embedding e fattorizzazione di matrici

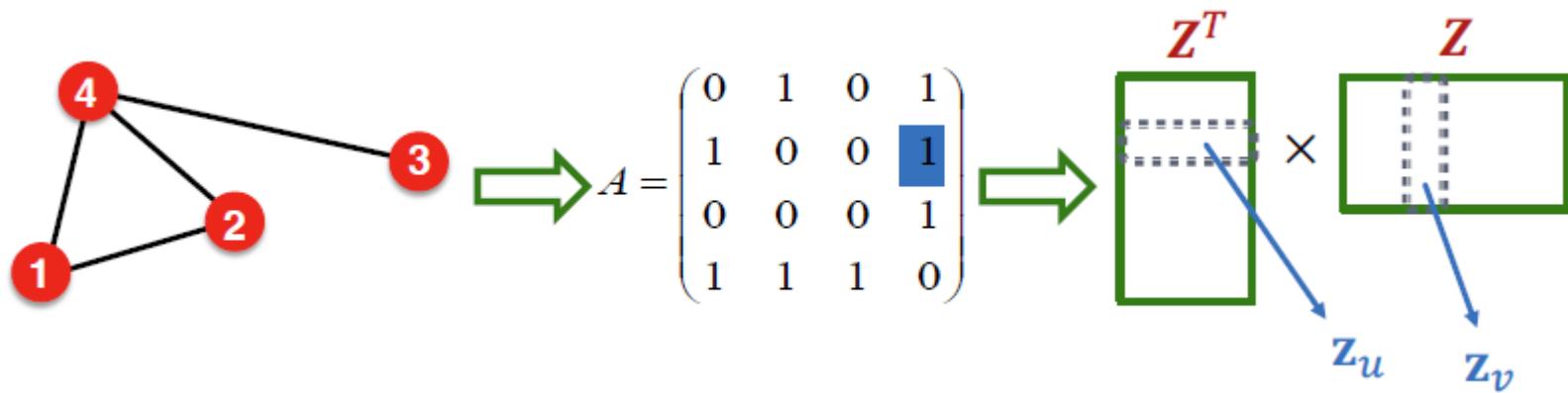
- Gli encoder come embedding lookup

$Z =$


Una colonna per nodo

Obiettivo: massimizzare  $Z^T Z$  per quelle coppie di nodi che sono simili

- La similarità più semplice: i nodi  $u$  e  $v$  sono simili se sono connessi da un arco
- Significa  $Z_v^T Z_u = A_{u,v}$
- Ne segue  $Z^T Z = A$



- L'embedding in uno spazio a d dimensioni (numero di righe di  $Z$ ) è molto più piccolo di n
- La fattorizzazione esatta  $A = Z^T Z$  in generale non è possibile
- Possiamo apprendere  $Z$  in modo approssimato
- Obiettivo

$$\min_Z \|A - Z^T Z\|_2$$

Ottimizzare  $Z$  in modo da minimizzare la norma di Frobenius

Noi abbiamo utilizzato la Softmax invece della L2 ma l'obiettivo è lo stesso

- DeepWalk e node2vec calcolano una similarità sui nodi più complessa basata su random walk
- Es. Deep Walk è equivalente alla fattorizzazione della seguente matrice

$$\log \left( vol(G) \left( \frac{1}{T} \sum_{r=1..T} (D^{-1} A)^r ) D^{-1} \right) \right) - \log b$$

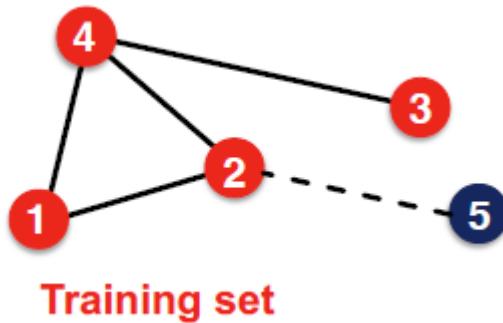
**Table 1: The matrices that are implicitly approximated and factorized by DeepWalk, LINE, PTE, and node2vec.**

Algorithm	Matrix
DeepWalk	$\log \left( \text{vol}(G) \left( \frac{1}{T} \sum_{r=1}^T (\mathbf{D}^{-1} \mathbf{A})^r \right) \mathbf{D}^{-1} \right) - \log b$
LINE	$\log \left( \text{vol}(G) \mathbf{D}^{-1} \mathbf{A} \mathbf{D}^{-1} \right) - \log b$
PTE	$\log \begin{pmatrix} \alpha \text{vol}(G_{\text{ww}})(\mathbf{D}_{\text{row}}^{\text{ww}})^{-1} \mathbf{A}_{\text{ww}} (\mathbf{D}_{\text{col}}^{\text{ww}})^{-1} \\ \beta \text{vol}(G_{\text{dw}})(\mathbf{D}_{\text{row}}^{\text{dw}})^{-1} \mathbf{A}_{\text{dw}} (\mathbf{D}_{\text{col}}^{\text{dw}})^{-1} \\ \gamma \text{vol}(G_{\text{lw}})(\mathbf{D}_{\text{row}}^{\text{lw}})^{-1} \mathbf{A}_{\text{lw}} (\mathbf{D}_{\text{col}}^{\text{lw}})^{-1} \end{pmatrix} - \log b$
node2vec	$\log \left( \frac{\frac{1}{2T} \sum_{r=1}^T \left( \sum_u \mathbf{X}_{w,u} \underline{\mathbf{P}}_{c,w,u}^r + \sum_u \mathbf{X}_{c,u} \underline{\mathbf{P}}_{w,c,u}^r \right)}{(\sum_u \mathbf{X}_{w,u})(\sum_u \mathbf{X}_{c,u})} \right) - \log b$

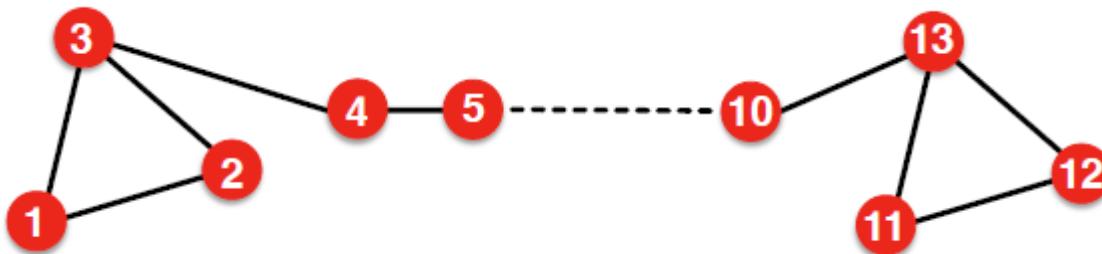
- Ottenuta l'immersione  $z_G$ 
  - Possiamo usarlo per fare delle predizioni
    - Opzione 1
      - Similarità:  $z_{G_1}^T z_{G_2}$
      - Usare una Rete neurale che Prende in input  $z_G$  per classificare G

# Limitazioni

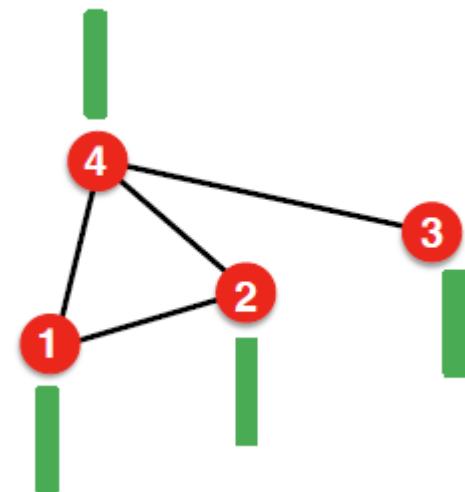
- Metodi ***trasduttivi***: non si possono ottenere embedding per nodi diversi da quelli presenti nel training set



- Non catturano la similarità strutturale



- Non si possono usare le feature dei nodi

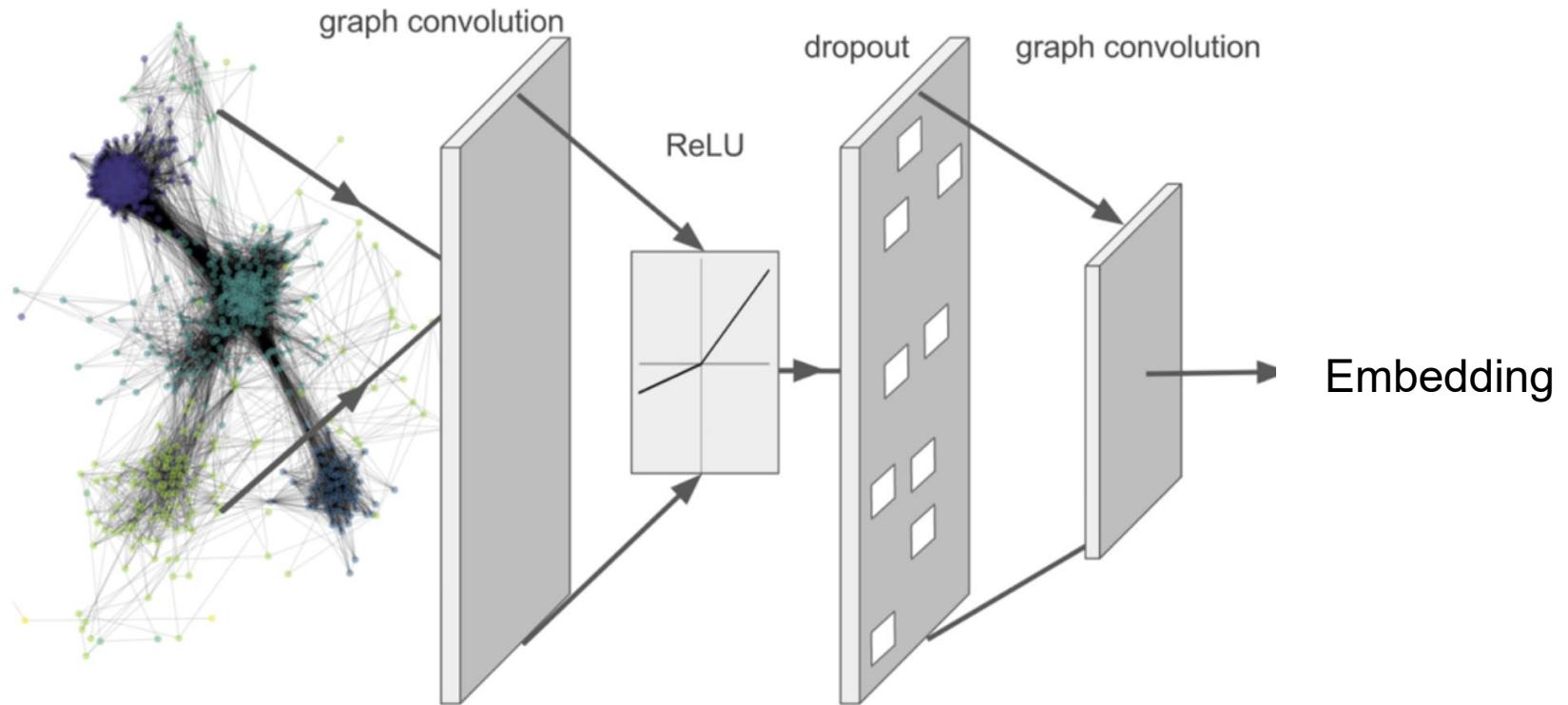


# Introduzione al Deep Learning

# Deep Graph encoder

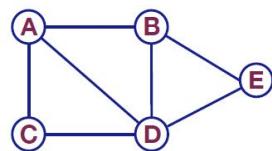
- Discutiamo metodi deep basati su graph neural networks

$Enc(v)$  = multiple layer di trasformazioni non lineari della struttura del grafo

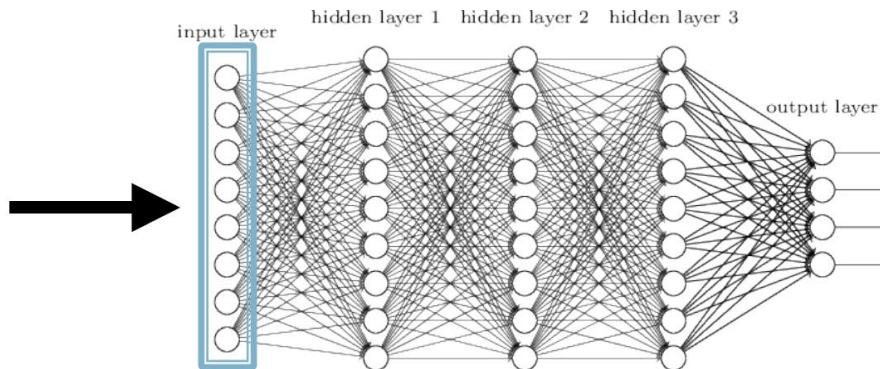


# Approccio Naive

- Uniamo matrice di adiacenza e feature
- La diamo in input ad una deep neural net



	A	B	C	D	E	Feat
A	0	1	1	1	0	1 0
B	1	0	0	1	1	0 0
C	1	0	0	1	0	0 1
D	1	1	1	0	1	1 1
E	0	1	0	1	0	1 0



- $O(N)$  parametri
- Non applicabile a grafi differenti
- Sensibile all'ordinamento

# Deep Learning concetti di base

- Apprendimento supervisionato: dato in input  $x$  l'obiettivo è predire  $y$ .
- $x$  può essere:
  - Vettore di numeri reali
  - Sequenza (es. testo)
  - Matrici (immagini)
  - Grafi (potenzialmente con feature nei nodi e negli archi)

Formuliamo il task come un problema di ottimizzazione

- Formuliamo il task come un problema di ottimizzazione

$$\min_{\Theta} \mathcal{L}(y, f(x))$$

Funzione obiettivo

- $\Theta$ : set di parametri da ottimizzare
  - Possono essere, scalari, vettori, matrici
  - $\Theta = \{Z\}$
- $\mathcal{L}$ : loss function. Es.

$$\mathcal{L}(y, f(x)) = \|y - f(x)\|_2$$

Altri esempi di loss function

L1 loss, huber loss, cross entropy, max margin

- Cross Entropy (CE)
  - Es. Etichetta  $y$  è categoriale (one-hot encoding)
  - $y : \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}$
  - $f(x) = \text{Softmax}(g(x))$
  - Es.  $f(x) = \begin{array}{|c|c|c|c|c|} \hline 0.1 & 0.3 & 0.4 & 0.1 & 0.1 \\ \hline \end{array}$
- $CE(y, f(x)) = - \sum_{i=1 \dots C} (y_i \log(f(x))_i)$
- Total loss sui dati di training:

$$\mathcal{L} = \sum_{(x,y) \in T} CE(y, f(x))$$

- $T$  Training set contenente tutte le coppie  $(x, y)$

# Gradiente

- Come ottimizzare la funzione obiettivo?
- Vettore del gradiente: direzione e velocità dell'incremento più alto

$$\nabla_{\Theta} \mathcal{L} = \left( \frac{\partial \mathcal{L}}{\partial \Theta_1}, \frac{\partial \mathcal{L}}{\partial \Theta_2}, \dots \right)$$

Con  $\Theta_i$  : componenti di  $\Theta$

- Derivata direzionale di una funzione a più variabili lungo un dato vettore rappresenta la velocità istantanea di cambio della funzione lungo il vettore
- Il gradiente è la derivata direzionale nella direzione del più alto incremento

- Algoritmo iterativo: ripetutamente aggiorna i pesi nella direzione opposta del gradiente fino a quando non converge
- $\Theta \leftarrow \Theta - \eta \nabla_{\Theta} \mathcal{L}$
- Training: ottimizza  $\Theta$  iterativamente
  - Iterazione: uno step nella direzione del gradiente discendente
  - Learning Rate (LR)  $\eta$ 
    - Parametro che controlla lo step del gradiente
    - Può variare durante il training (LR Scheduling)
    - Stop ideale 0 *Gradiente*

# Gradiente discendente stocastico: Minibatch

- Size del batch: numero di punti nel minibatch
- Iterazioni: uno step di SGD sul minibatch
- Epoca: passo completo sul dataset (# iterazioni è uguale al rapporto tra dataset size e batch size)

# Neural Network

- Obiettivo:  $\min_{\Theta} \mathcal{L}(y, f(x))$
- Nel deep learning la funzione  $f$  può essere molto complessa
- Iniziamo un caso semplice (funzione lineare):  
$$f(x) = W \cdot x, \quad \Theta = \{W\}$$
- Se  $f$  restituisce uno scalare allora  $W$  è un vettore

$$\nabla_W f = \left( \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial w_3}, \dots \right)$$

Se  $f$  restituisce un vettore allora  $W$  è una matrice dei pesi

$$\nabla_W f = W^T \text{ (matrice dello Jacobiano di } f)$$

# Back-propagation

- Funzioni più complesse:

$$f(x) = W_2(W_1x), \quad \Theta = \{W_1, W_2\}$$

Derivazione (funzioni composte)

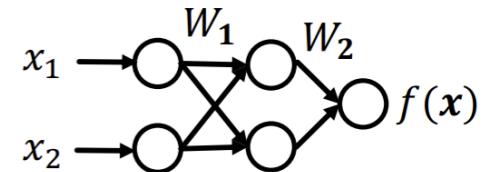
$$\begin{aligned} f(x) &= W_2(W_1x), \\ h(x) &= W_1x \\ g(z) &= W_2z \end{aligned}$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$
$$\nabla_x f = \frac{\partial f}{\partial (W_1x)} \cdot \frac{\partial (W_1x)}{\partial x}$$

Back-propagation: usiamo la regola della derivazione delle funzioni composte per propagare il gradiente nei passi intermedi fino ad ottenere il gradiente di  $\mathcal{L}$  ovvero  $\Theta$

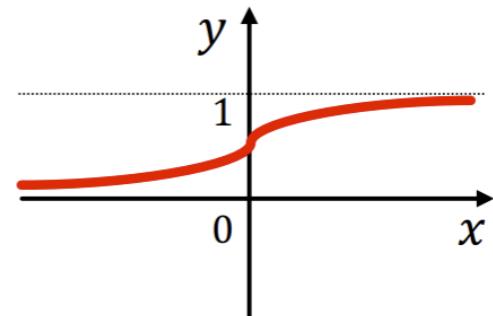
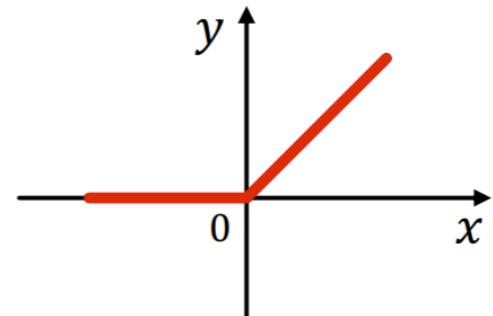
# Back-propagation Esempio

- 2 Layer
- $f(x) = g(h(x)) = W_2(W_1x)$
- $\mathcal{L} = \sum_{(x,y) \in B} \|(y - f(x))\|_2$  somma della L2 loss nel minibatch B
- Layer nascosto: rappresentazione intermedia dell'input  $x$
- $h(x) = W_1x$
- $f(x) = W_2h(x)$



# Non linearità

- $f(x) = W_2(W_1x)$ ,  $W_1 W_2$  è un'altra matrice
- $f(x)$  è ancora lineare non importa quante sono le matrici che componiamo
- Introduciamo la non linearità
  - Rectified linear unit (ReLU)
    - $ReLU(x) = \max(x, 0)$
  - Sigmoide
  - $\sigma(x) = \frac{1}{1+e^{-x}}$



# Percettrone Multi-layer (MLP)

- Ogni layer dell'MLP combina trasformazioni lineari e non lineari

$$x^{(l+1)} = \sigma(W_l x^{(l)} + b^l)$$

- Dove  $W_l$  è la matrice dei pesi che trasforma la rappresentazione del layer nascosto  $l$  a quello successivo  $l+1$
- $b^l$  bias del layer  $l$  è aggiunto alla trasfromazione lineare  $x$
- $\sigma$  funzione non lineare

- Funzione Obiettivo:

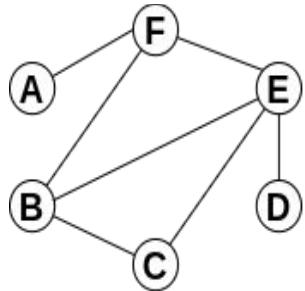
$$\min_{\Theta} \mathcal{L}(y, f(x))$$

- $f$  può essere un layer lineare o un MLP o altri tipi di reti neurali (GNN)
- Campionamento minibatch sull'input  $x$
- Propagazione forward: calcoliamo  $\mathcal{L}$  dato  $x$
- Propagazione backward: otteniamo il gradiente  $\nabla_{\Theta} \mathcal{L}$  usando la regola della derivazione delle funzioni composte
- Usiamo il gradiente discendente stocastico per ottimizzare  $\Theta$  nelle diverse iterazioni

- Supponiamo di avere un grafo G
  - V insieme dei vertici
  - A matrice di adiacenza
  - $X \in \mathbb{R}^{m \times |V|}$  matrice delle features dei nodi
  - Features
    - Social net: Profilo utente
    - Reti Biologiche: Profilo espressione genica, annotazioni funzionali
    - ...

# Torniamo ai grafi

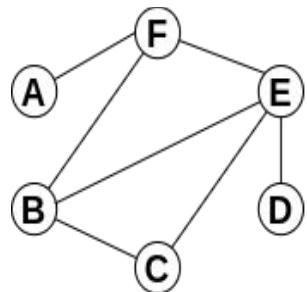
Grafo



Matrice di adiacenza

	A	B	C	D	E	F
A	0	0	0	0	0	1
B		0	1	0	1	1
C			0	0	1	0
D				0	1	0
E					0	1
F						0

Grafo

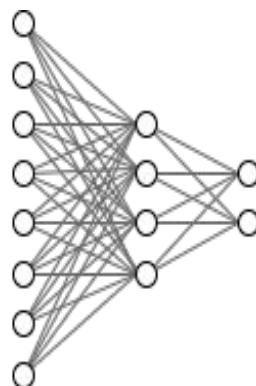
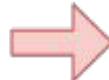


Matrice di adiacenza

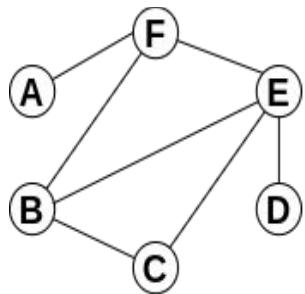
	A	B	C	D	E	F
A	0	0	0	0	0	1
B		0	1	0	1	1
C			0	0	1	0
D				0	1	0
E					0	1
F						0

Rete neurale

0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	1	0
0	0	0	0	1	0
0	1	1	1	0	1
1	1	1	0	1	0



Grafo

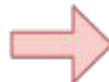


Matrice di adiacenza

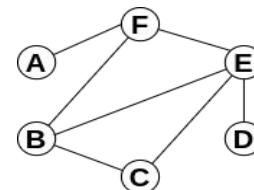
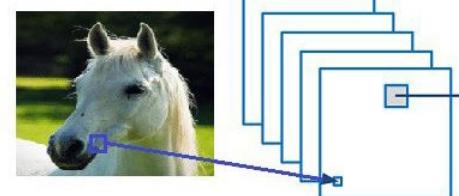
	A	B	C	D	E	F
A	0	0	0	0	0	1
B		0	1	0	1	1
C			0	0	1	0
D				0	1	0
E					0	1
F						0

Rete neurale

0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	1	0
0	0	0	0	1	0
0	1	1	1	0	1
1	1	1	0	1	0

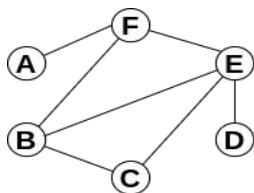
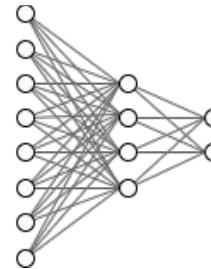


Convolution Neural network



0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	1	0
0	0	0	0	1	0
0	1	1	1	0	1
1	1	1	0	1	0

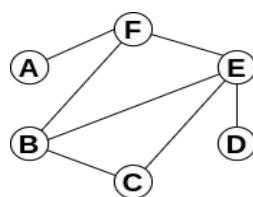
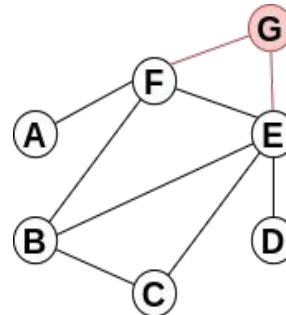
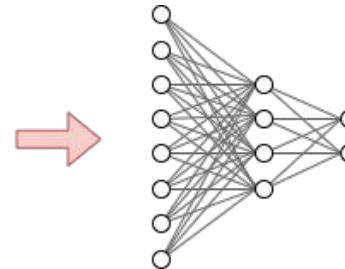
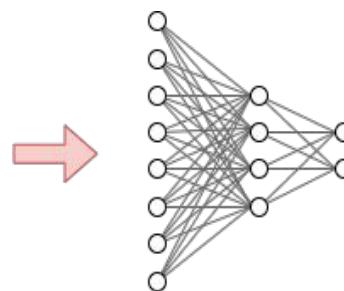
1	0	1
1	1	0
0	1	0


$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$


## ■ Problemi

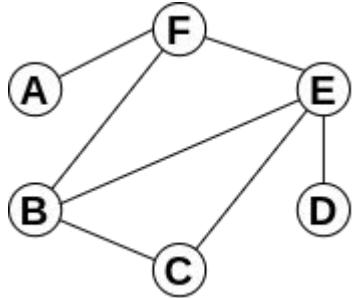
- Differenti dimensioni
- Non invariante all'ordinamento dei nodi

# Dimensioni diverse


$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$


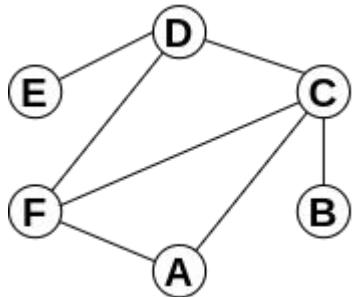
# Ordinamento dei nodi

**$G$**

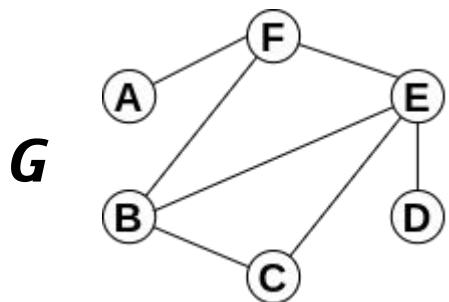


$$G = G'$$

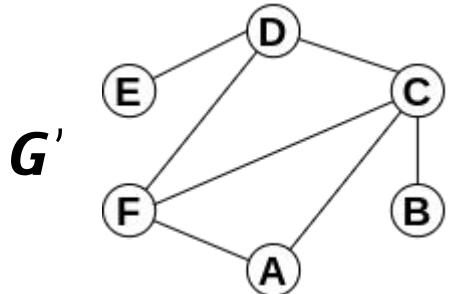
**$G'$**



# Non invariante all'ordinamento dei nodi



$$G = G'$$



$$\text{Adj}(G)$$

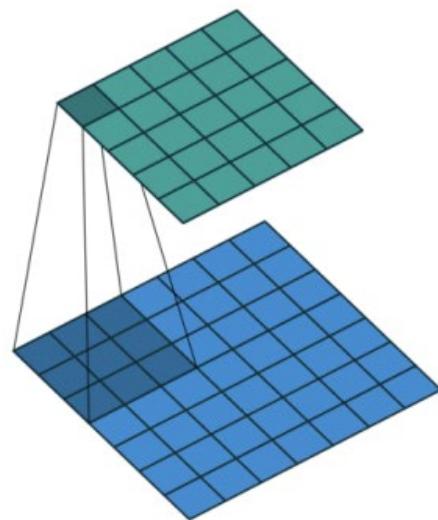
0	0	0	0	0	0	1
0	0	1	0	1	1	1
0	1	0	0	1	0	0
0	0	0	0	1	0	0
0	1	1	1	0	1	1
1	1	1	0	1	0	0

$$\text{Adj}(G) \neq \text{Adj}(G')$$

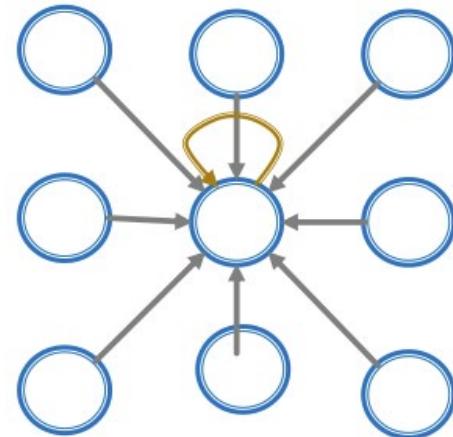
$$\text{Adj}(G')$$

0	0	1	0	0	1
0	0	1	0	0	0
1	1	0	1	0	1
0	0	1	0	1	1
0	0	0	0	1	0
1	0	1	1	0	0

- Convolutional neural network (CNN) con filtro 3x3:



Immagine

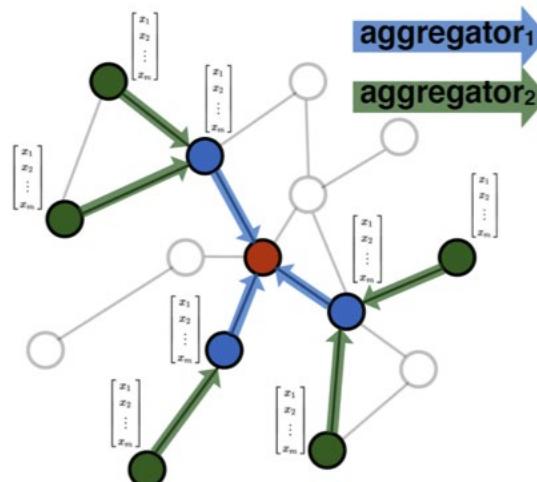
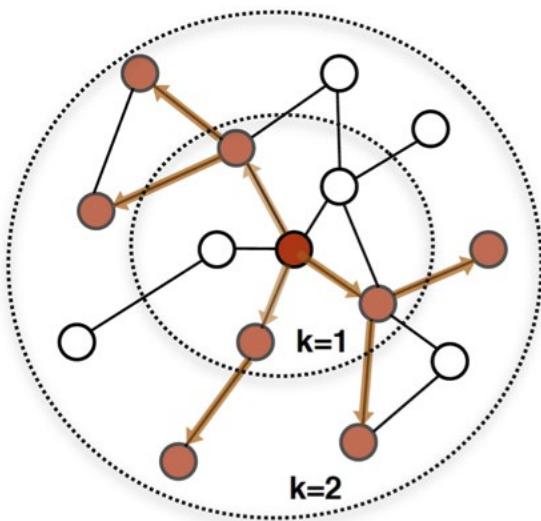


Grafo

- Idea: Trasforma l'informazione che c'è nei vicini e combinala
  - Trasforma i “messaggi”  $h_i$  dai vicini:  $W_i h_i$
  - sommali:  $\sum_i W_i h_i$

# GCN: Graph Convolutional Networks (Kipf e Welling ICLR 17)

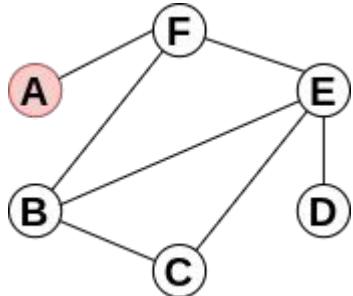
- Il vicinato dei nodi definisce il calcolo
  - Definiamo grafo computazionale del nodo
  - Propaghiamo e trasformiamo l'informazione



# Grafo di computazione

Il vicinato definisce il suo grafo di computazione

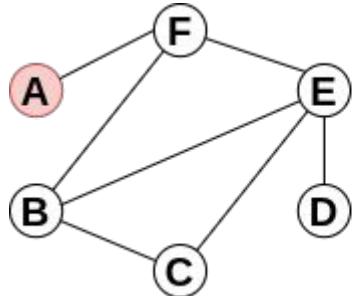
Grafo di input



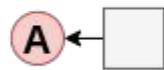
# Grafo di computazione

Il vicinato definisce il suo grafo di computazione

Grafo di input



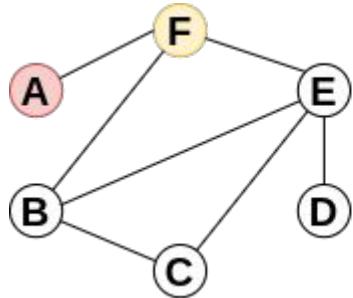
Grafo di computazione



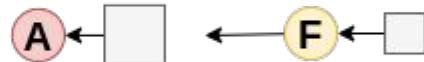
# Grafo di computazione

Il vicinato definisce il suo grafo di computazione

Grafo di input



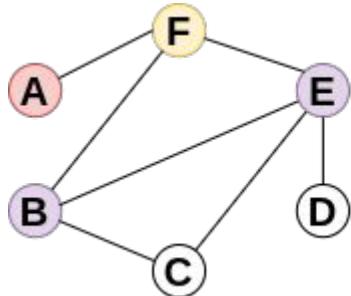
Grafo di computazione



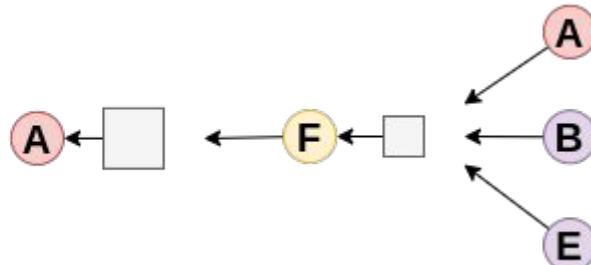
# Grafo di computazione

Il vicinato definisce il suo grafo di computazione

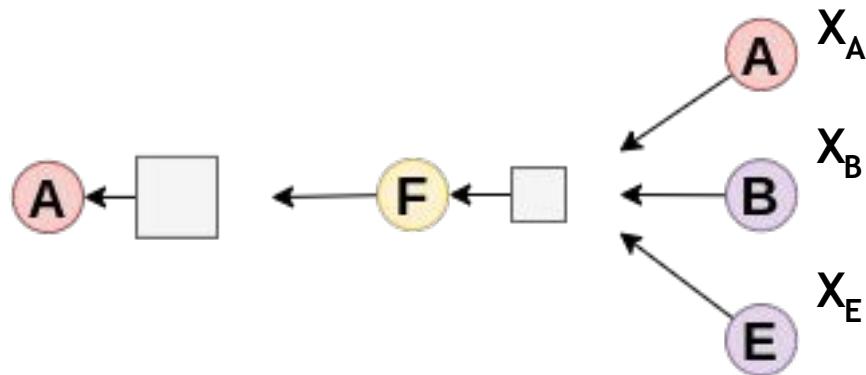
Grafo di input



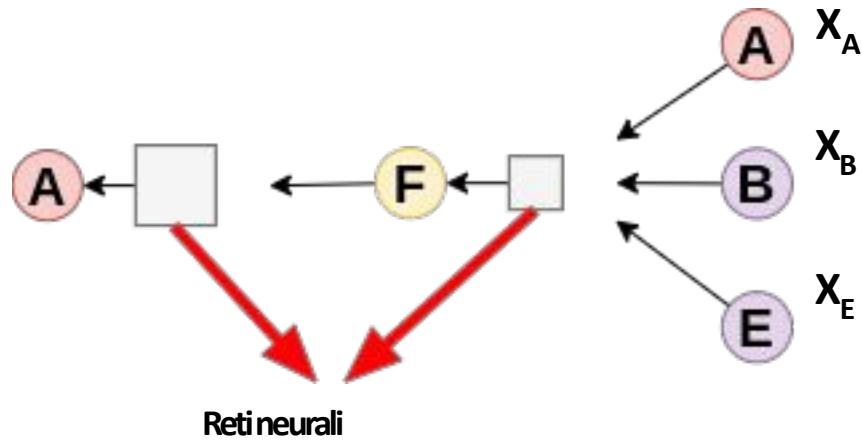
Grafo di computazione



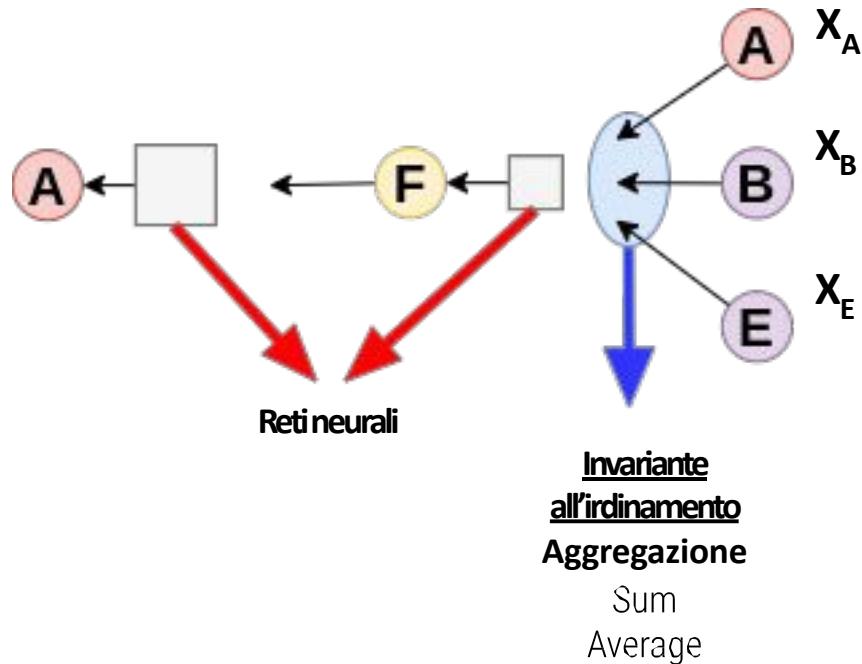
# Grafo di computazione



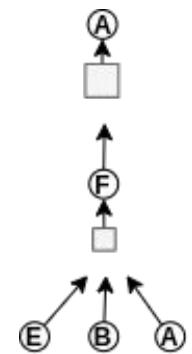
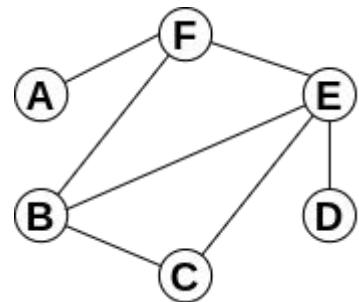
# Grafo di computazione



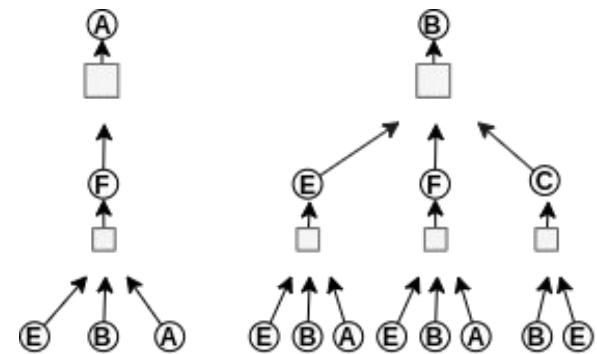
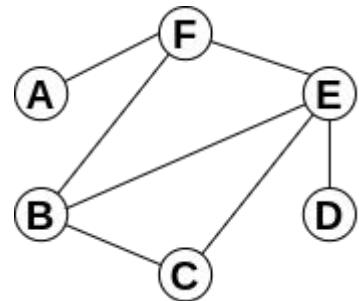
# Grafo di computazione



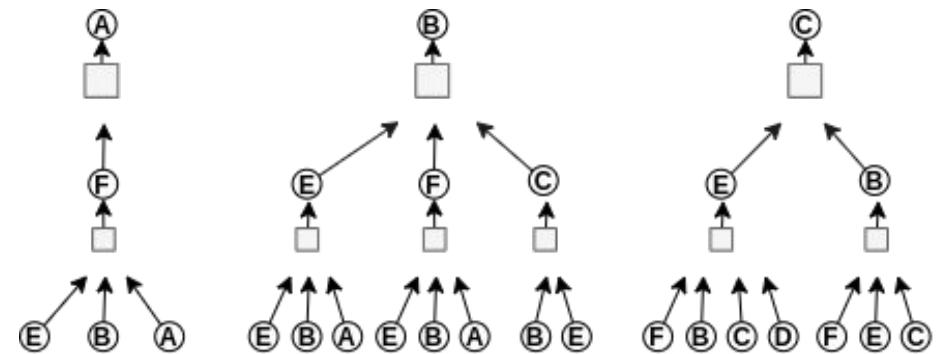
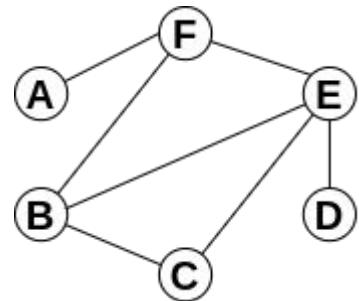
# Ogni nodo ha il suo grafo di computazione



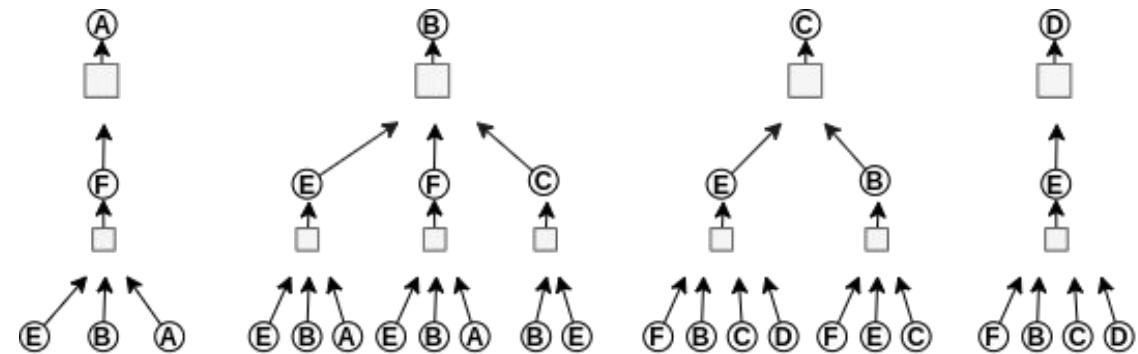
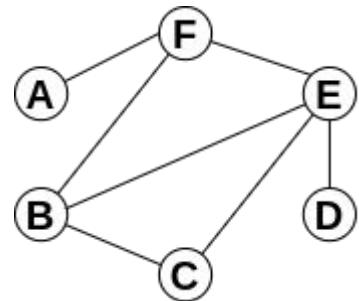
# Ogni nodo ha il suo grafo di computazione



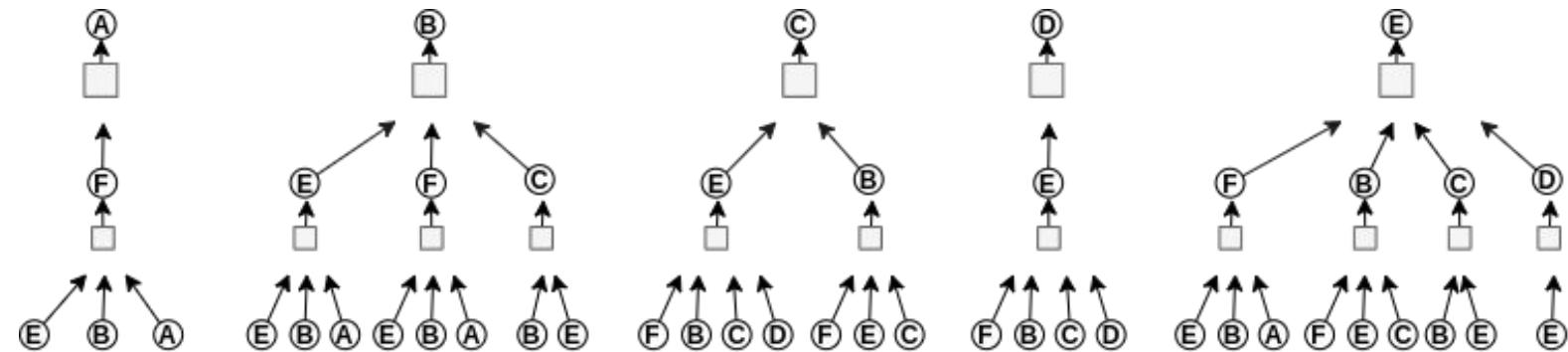
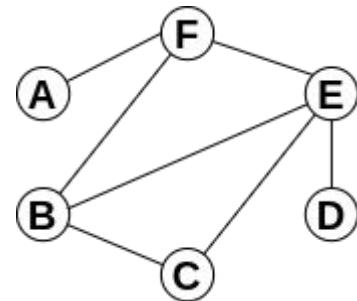
# Ogni nodo ha il suo grafo di computazione



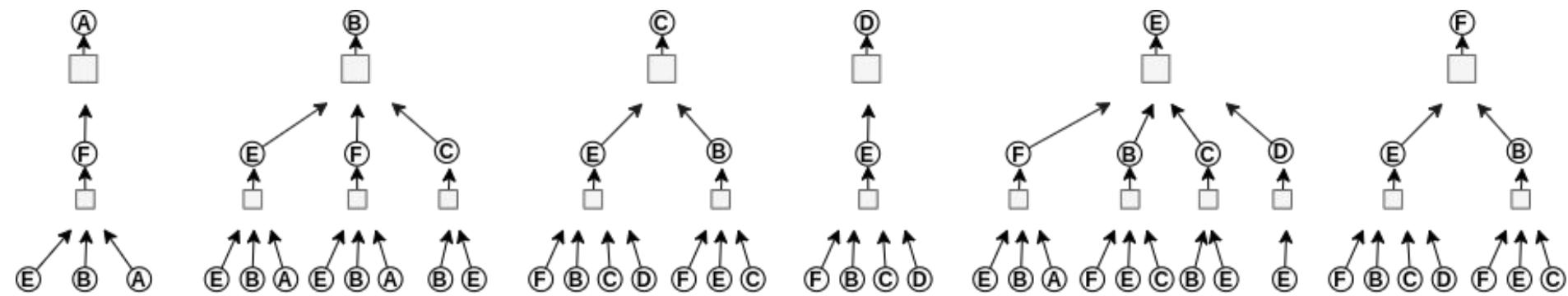
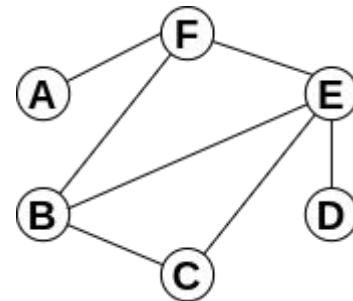
# Ogni nodo ha il suo grafo di computazione



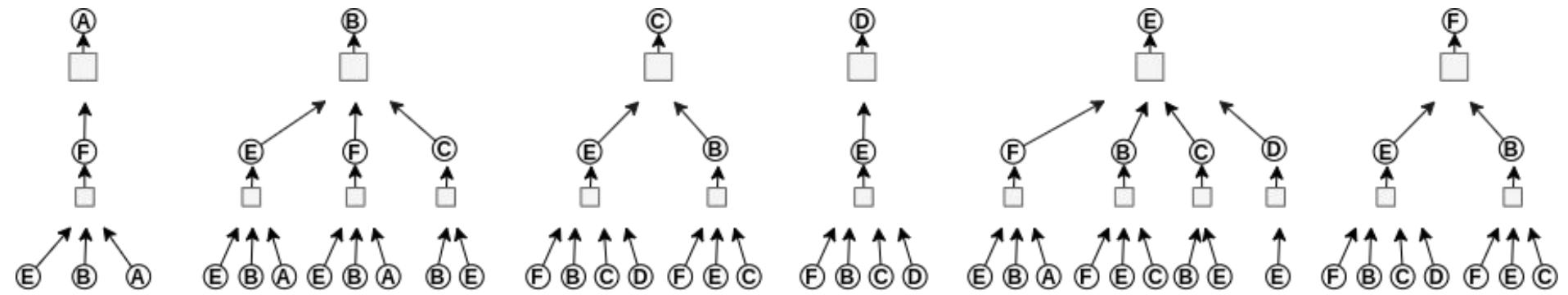
# Ogni nodo ha il suo grafo di computazione



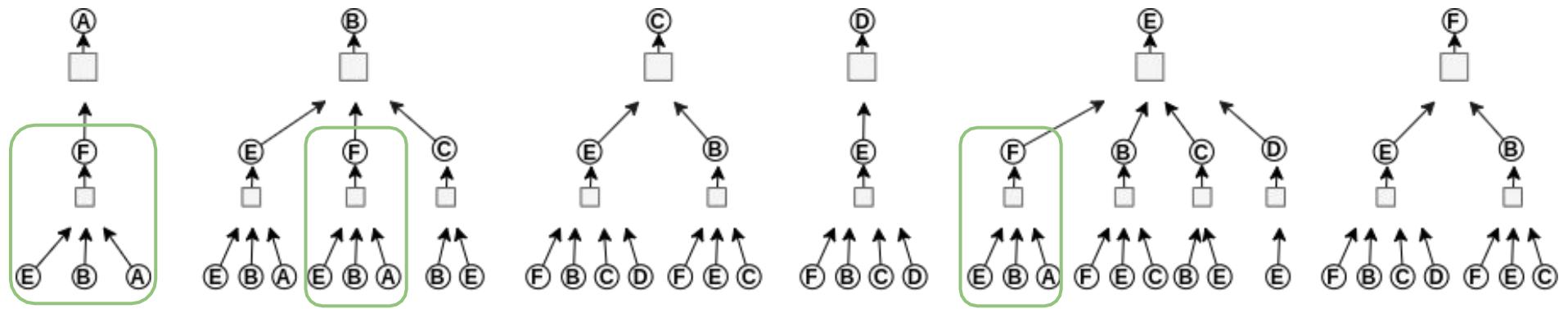
# Ogni nodo ha il suo grafo di computazione



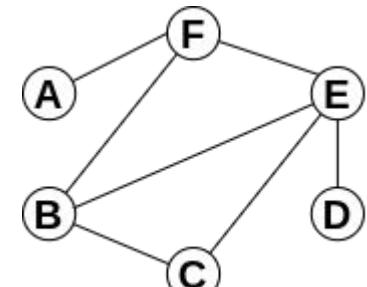
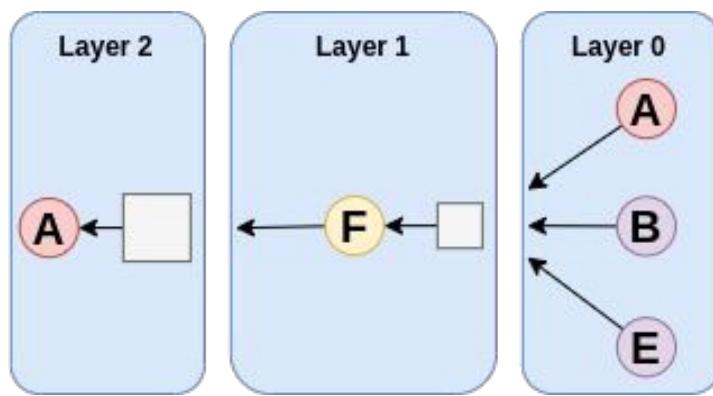
# Ci sono ridondanze?



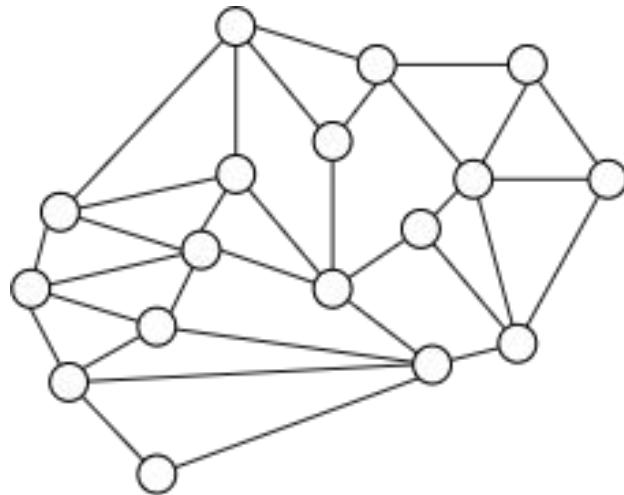
# Ci sono ridondanze?



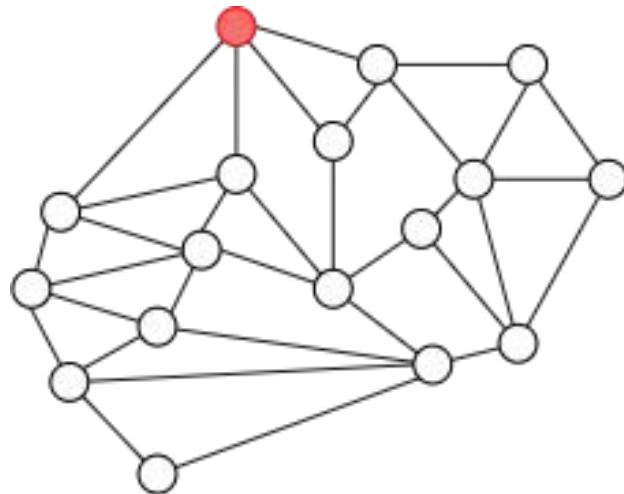
# Layer



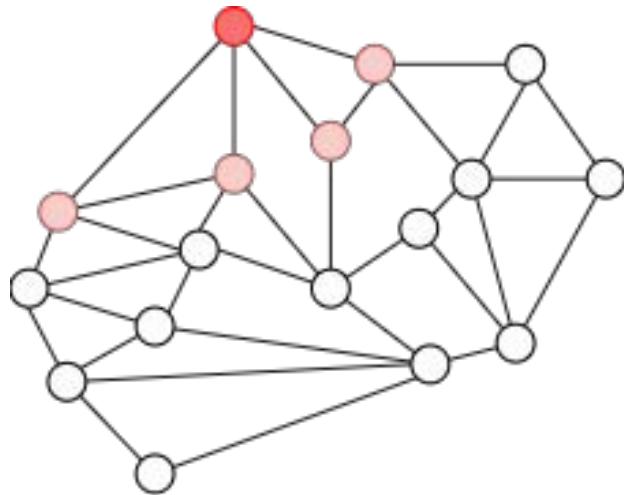
# Quanto andare distanti dal nodo?



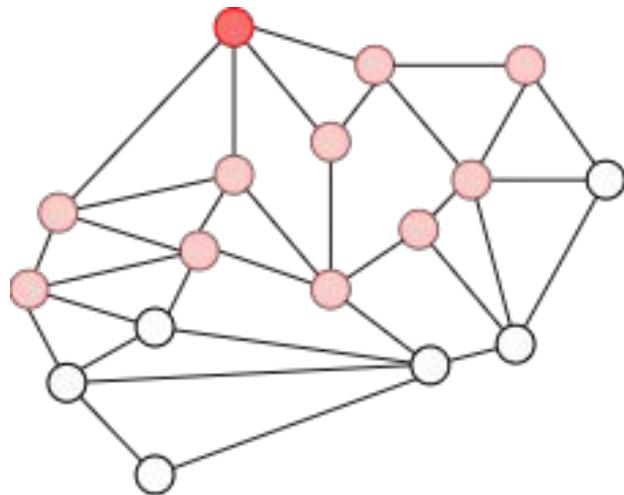
# Quanto andare distanti dal nodo?



# Quanto andare distanti dal nodo?

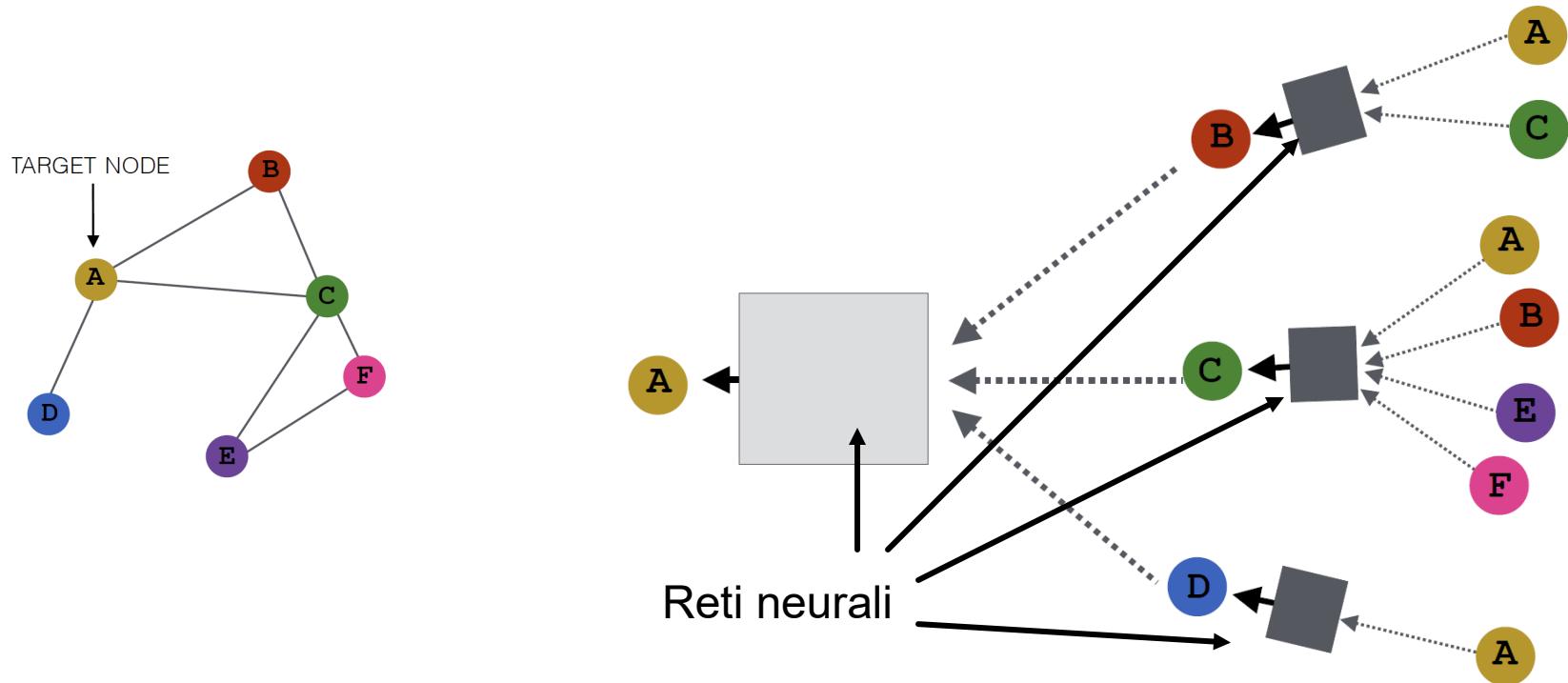


# Quanto andare distanti dal nodo?

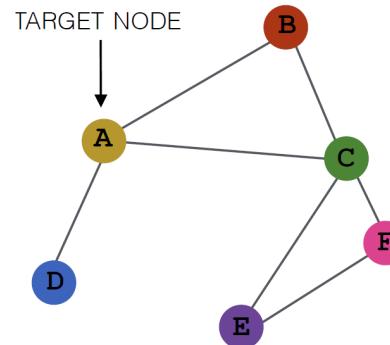


Genera l'embedding dei nodi in base al vicinato locale

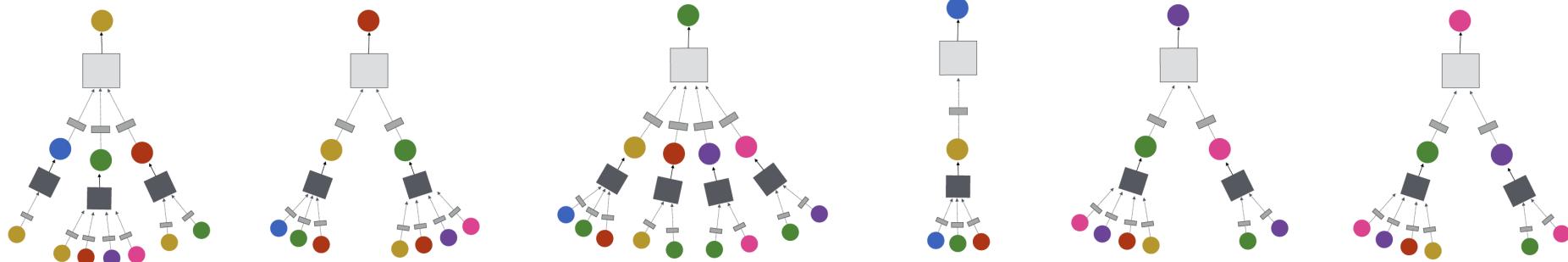
I nodi aggregano l'informazione dei vicini usando una rete neurale



- Intuizione: il vicinato definisce il grafo computazionale

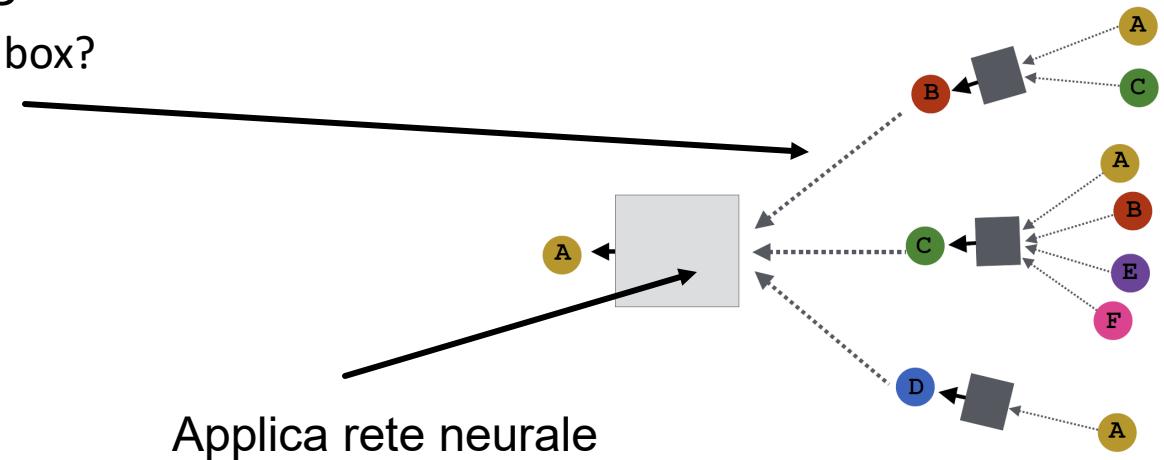


Ogni nodo definisce un grafo di computazione basato sui suoi vicini



# Deep Model: diversi layer

- I modelli possono avere un numero di layer arbitrario
  - Nodi hanno un embedding ad ogni layer
  - Layer-0 è l'embedding del nodo u e delle sue feature  $x_u$
  - Layer-k prendi informazioni dai layer che sono a k hop.
- Possiamo aggregare in modo differente
  - Come definiamo il box?
    - Media da ogni vicino
    - NN



# Encoder

- Media dei messaggi in connessione ad una rete neurale

Layer 0, l'embedding è uguale alle feature dei nodi

$$h_v^0 = x_v$$

$$h_v^k = \sigma \left( W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right) \forall k$$

$$\in \{1, \dots K\}$$

Funzione non lineare

$$z_v = h_v^K$$

# Encoder

$$h_v^0 = x_v$$
$$h_v^k = \sigma \left( W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right) \forall k$$

**Parametri appresi**

$$\in \{1, \dots K\}$$
$$z_v = h_v^K$$

- $z_v$  embedding in funzione del grafo
- Setting supervisionato: minimizzare una funzione loss

$$\min_{\Theta} \mathcal{L}(y, f(z_v))$$

$y$ : etichetta del nodo

$\mathcal{L}$  potrebbe essere L2 se  $y$  numero reale o cross entropy se  $y$  categoriale

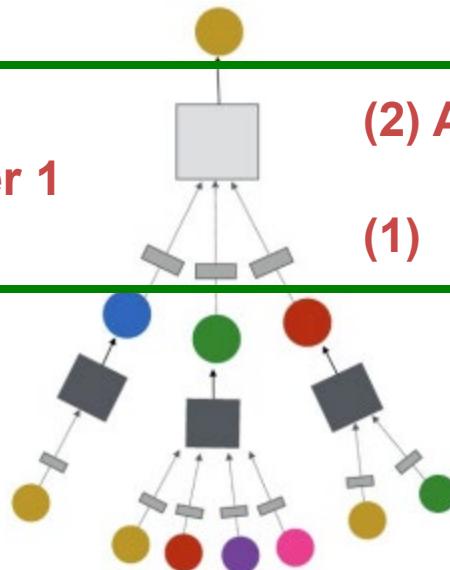
# Training

- Addestriamo il modello per un task supervisionato
    - Es. Rete interazione drug-drug
    - Tossicità di una drug
- $\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(z_v^T \theta)) + (1 - y_v) \log(1 - \sigma(z_v^T \theta))$
- Etichetta della classe
- Valore di embedding
- Gli stessi parametri appresi per una parte dei nodi possono essere usati per il resto della rete

**GNN layer 1**

**(2) Aggregazione**

**(1) messaggi**



## (1) Calcolo del messaggio

Funzione messaggio

$$m_u^{(l)} = MSG^{(l)} \left( h_u^{(l-1)} \right)$$

ogni nodo crea un messaggio che verrà inviato agli altri nodi

Es.

$$m_u^{(l)} = W^{(l)} h_u^{(l-1)}$$

## (2) Aggregazione

Ogni nodo aggrega i messaggi dei vicini

$$h_v^{(l)} = AGG^{(l)} \left( \{m_u^{(l)}, u \in N(v)\} \right)$$

Ese.

$$h_v^{(l)} = \sum \left( \{m_u^{(l)}, u \in N(v)\} \right)$$

- L'informazione del nodo v stesso potrebbe essere persa

$$h_v^{(l)} = CONCAT \left( AGG^{(l)} \left( \{m_u^{(l)}, u \in N(v)\} \right), m_v^{(l)} \right)$$

- Graph Convolutional Networks (GCN)

$$h_v^{(l)} = \sigma \left( \sum_{u \in N(v)} W^{(l)} \frac{h_u^{(l-1)}}{|N(v)|} \right)$$

- GraphSage

$$h_v^{(l)} = \sigma \left( W^{(l)} \text{CONCAT} \left( h_v^{(l-1)}, \text{AGG} \left( \{h_u^{(l-1)}, \forall u \in N(v)\} \right) \right) \right)$$

- Graph Attention Networks

$$h_v^{(l)} = \sigma \left( \sum_{u \in N(v)} \alpha_{vu} W^{(l)} h_u^{(l-1)} \right)$$

# Pytorch Geometric



<https://pytorch-geometric.readthedocs.io/en/latest/index.html>