Graph Neural Networks

Tema 2 - Shallow Encoders

UNIVERSIDAD POLITÉCNICA DE MADRID

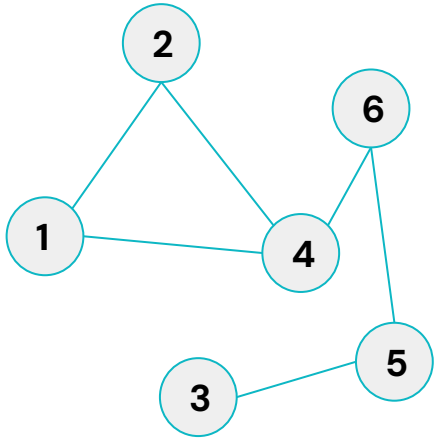
Máster
Deep Learning

Índice


Aprendizaje representacional para grafos
1
Embeddings para nodos en un grafo: **Shallow embeddings**
2
Representación por caminos aleatorios: **RandomWalk**
3
Un *mejor* camino aleatorio: **Node2Vec**
4
Embeddings para aristas
5
Embeddings para grafos
6

1. Aprendizaje representacional para grafos

Aprendizaje tradicional



¿Qué características tiene el *nodo 4*?

Tiene **grado** 3.

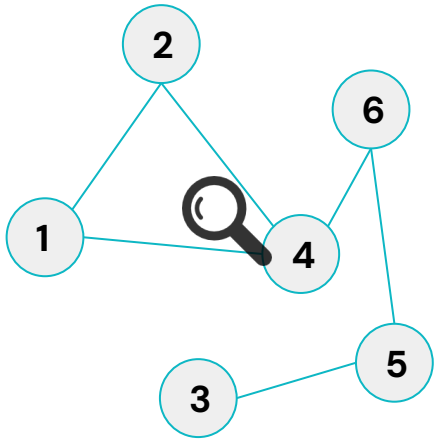
Tiene **betweenness** 0.6.

Tiene **closeness** 0.625.

Tiene **neighborhood** 1.

...

Aprendizaje tradicional



Feature engineering manual

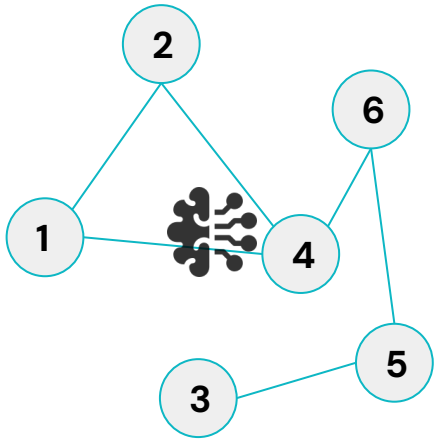
1. Extraer atributos especializados para una tarea concreta.

Grado, centralidad, vecindario, etc.

2. Entrenar un modelo en base a esas características.

SVM, RandomForest, XGBoost, etc.

Aprendizaje representacional



que respete que nodos similares tengan **representaciones cercanas en el espacio latente**

Feature engineering automático

1. Extraer atributos ~~especializados~~ **genéricos** para ~~una tarea concreta~~ **muchas tareas**.

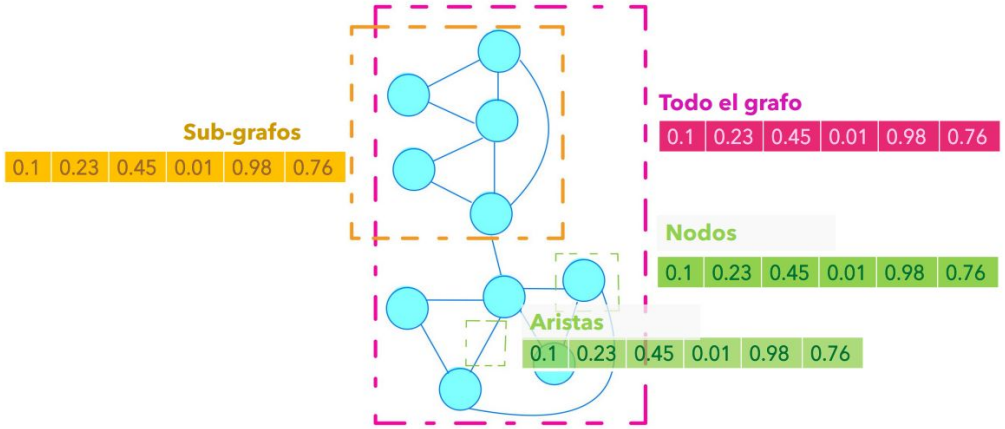
Embedding abstracto y genérico.

2. Entrenar un modelo en base a esas características.

SVM, RandomForest, XGBoost, etc.

2. Shallow embeddings

Shallow embeddings



Obtendremos **representaciones vectoriales** de nodos, aristas, sub-grafos, etc, de forma que estarán proyectadas en un **espacio latente de cierta dimensión**.

Para **nodos similares**, sus representaciones deberán tener también un **alto coeficiente de similitud**.

Shallow embeddings

Diseño manual

Las características (grado, centralidad, etc) deben ser seleccionadas y calculadas manualmente, lo que introduce cierto sesgo

Especificidad al dominio

Algunas métricas podrán ser útiles en ciertos dominios pero no generalizar bien a otros.

Pérdida de información estructural

Aunque útiles, no siempre capturan relaciones complejas en los grafos

Captura de patrones complejos

Flexibilidad en la integración y la captura de patrones tanto locales como globales de alta dimensión

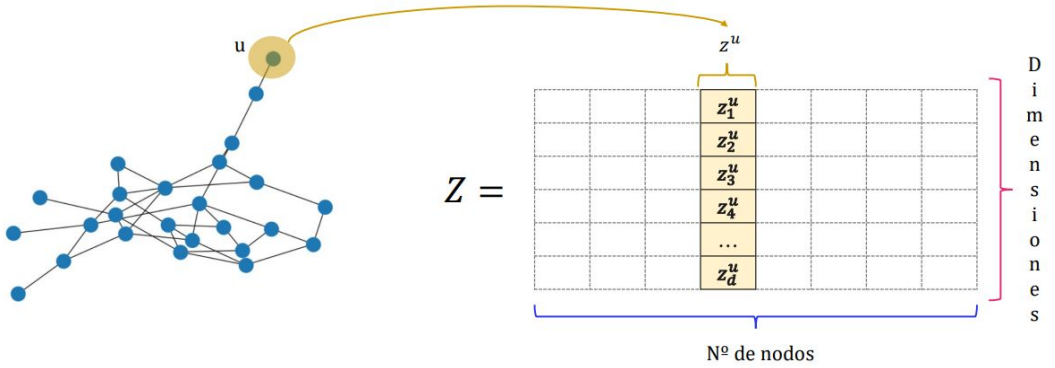
Shallow embeddings

¿Por qué usar shallow embeddings frente a vectores formados por características predefinidas como el grado o la centralidad?

Shallow embeddings

Este ***embedding*** deberá recoger la **información posicional** de un nodo y su **contexto** en el grafo

Dejaremos que un algoritmo de **machine learning** extraiga patrones de esta representación

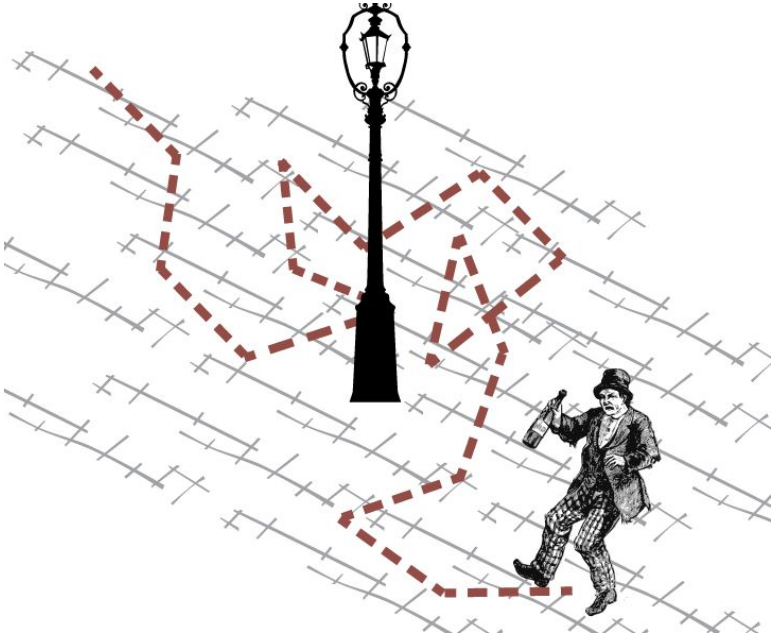


3. Representación por caminos aleatorios

Caminos aleatorios

Un camino aleatorio que atraviesa un grafo nos da una **perspectiva no sesgada** del contexto de los nodos.

Nodos **con patrones de conexión similares** tienden a generar caminos con **distribuciones parecidas**.



Caminos aleatorios

Definiremos la similitud por caminos aleatorios entre dos nodos u y v como la probabilidad de que v aparezca en un camino aleatorio que empieza en u

Flexibilidad en la captura de patrones

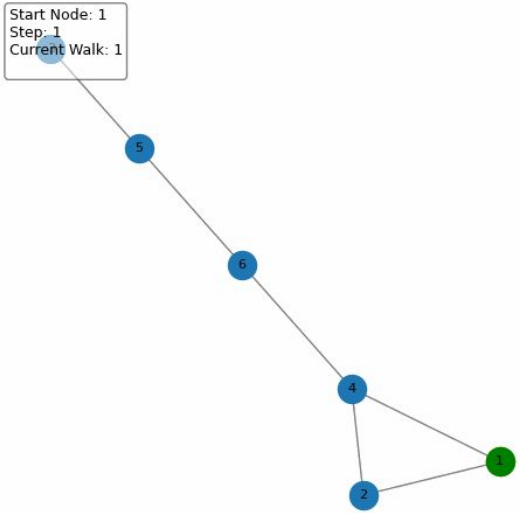
Dependiendo de la longitud que fijemos permitiremos obtener tanto información local como global

Cómputo *ridículamente* eficiente

Su potencia está en su simpleza. Su complejidad es lineal y el algoritmo se reduce a generar números aleatorios.

Random Walk

1. Fijamos una **longitud máxima** para el camino
2. Partimos de **cierto nodo**.
3. Obtenemos sus **vecinos directos**.
4. Seleccionamos uno de manera **aleatoria**.
5. **Avanzamos** al nodo seleccionado.
6. Si ya hemos llegado a la longitud fijada **terminamos el algoritmo**, si no, **volvemos al paso 2**.



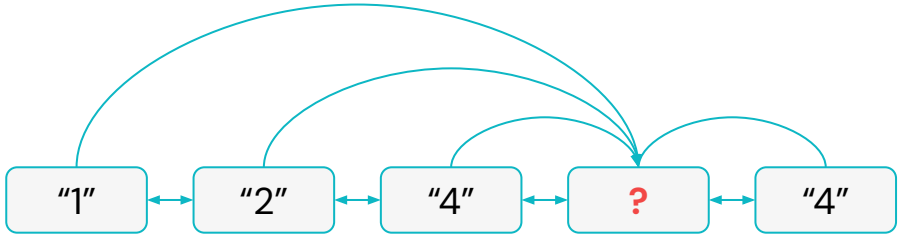
Todos los *saltos* son equiprobables!

DeepWalk = RandomWalk + Word2Vec

Para cada nodo podemos computar **múltiples Random Walks** de manera que todas esas secuencias **representen en su conjunto el contexto de ese nodo en el grafo**.

Podemos **entender el camino como una secuencia** (o incluso una *frase*) y aplicar **Word2Vec** para aprender representaciones de cada nodo basadas en su contexto.

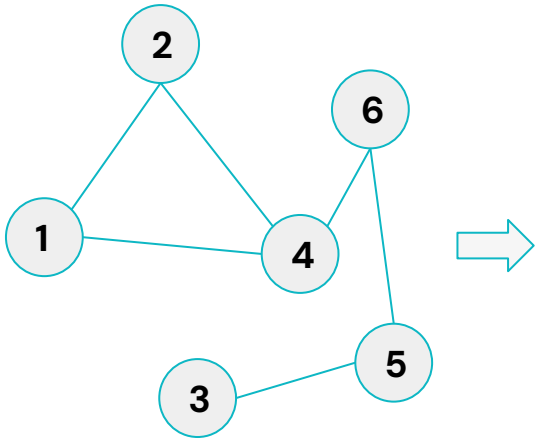
[1, 2, 4, 6, 4] → "1 2 4 6 4"



El objetivo de entrenamiento de Word2Vec es reconstruir un elemento de la frase (palabra/nodo) dado el resto.

DeepWalk

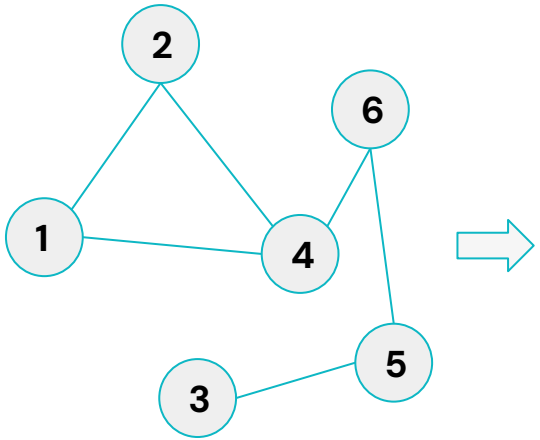
Con estas características hemos obtenido una ***equivalencia tabular*** y podemos usar cualquier algoritmo de machine learning para resolver una tarea de ...



node	feat_0	feat_1	...	feat_n
1	0.213	-1.345		-1.324
2	1.231	-0.234		-0.324
3	-0.324	2.345		1.678
4	-0.121	1.234		2.453
5	1.234	-0.234		0.643
6	0.534	1.344		0.546

DeepWalk

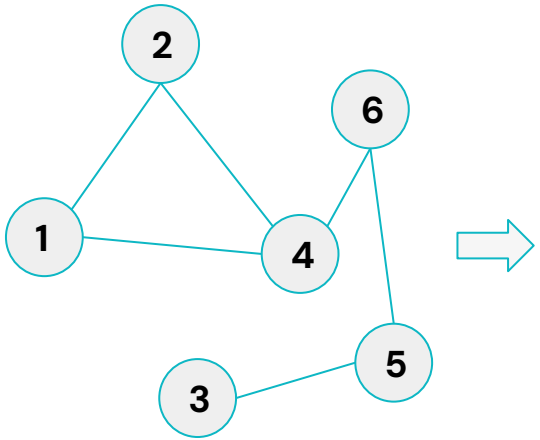
Con estas características hemos obtenido una ***equivalencia tabular*** y podemos usar cualquier algoritmo de machine learning para resolver una tarea de **clasificación**



node	feat_0	feat_1	...	feat_n	label
1	0.213	-1.345		-1.324	0
2	1.231	-0.234		-0.324	0
3	-0.324	2.345		1.678	1
4	-0.121	1.234		2.453	1
5	1.234	-0.234		0.643	0
6	0.534	1.344		0.546	1

DeepWalk

Con estas características hemos obtenido una ***equivalencia tabular*** y podemos usar cualquier algoritmo de machine learning para resolver una tarea de **regresión**



node	feat_0	feat_1	...	feat_n	label
1	0.213	-1.345		-1.324	0.4
2	1.231	-0.234		-0.324	0.9
3	-0.324	2.345		1.678	0.1
4	-0.121	1.234		2.453	1.0
5	1.234	-0.234		0.643	0.5
6	0.534	1.344		0.546	0.3

4. Random Walk sesgado: Node2Vec

Limitaciones de Random Walk

No evita que se pueda volver al nodo anterior lo que puede suponer **quedarse atrapado** en zonas densas del grafo o a la realización de una **exploración redundante**.

Selecciona nodos de manera equiprobable lo que supone que con mayor facilidad **se estanque en áreas cercanas y no pueda explorar patrones globales del grafo**.

Node2Vec

Node2Vec es una **generalización** de Random Walk que introduce dos hiperparámetros:

p: Return parameter (BFS-like)

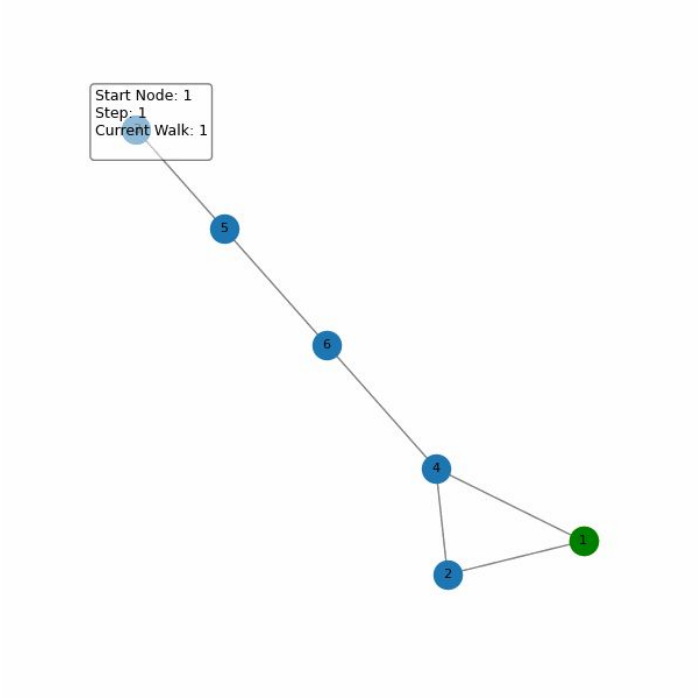
Controla la probabilidad de **volver al nodo anterior**, valores altos(>1) **reducen la probabilidad de regresar**.

q: In-out parameter (DFS-like)

Controla la probabilidad de **visitar nodos no visitados recientemente**, valores altos(>1) **favorecen la exploración**.

Node2Vec

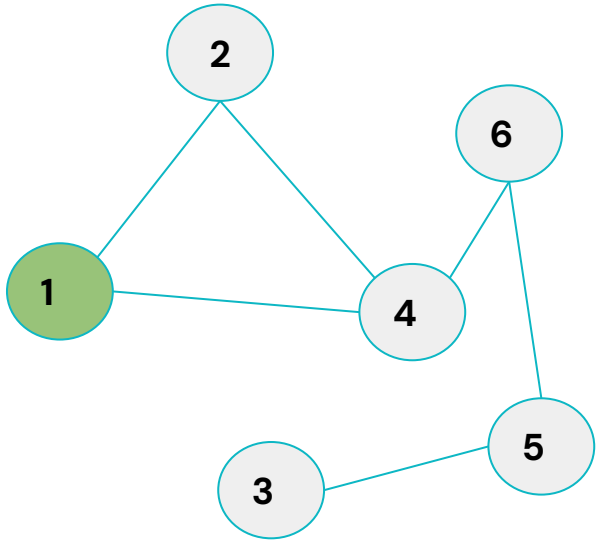
1. Fijamos una **longitud máxima** para el camino, **p** y **q**.
2. Partimos de **cierto nodo**.
3. Obtenemos sus **vecinos directos**.
   a. Si el vecino es el nodo previo, la probabilidad es **1/p.**
   b. Si es vecino directo del nodo previo, la probabilidad es **1**.
   c. Si no es ninguna de ambas (vecino lejano), la probabilidad es de **1/q.**
4. **Avanzamos** al nodo seleccionado.
5. Si ya hemos llegado a la longitud fijada **terminamos el algoritmo**, si no, **volvemos al paso 2**.



Este node2vec está realizado con p=5, q=2

Node2Vec

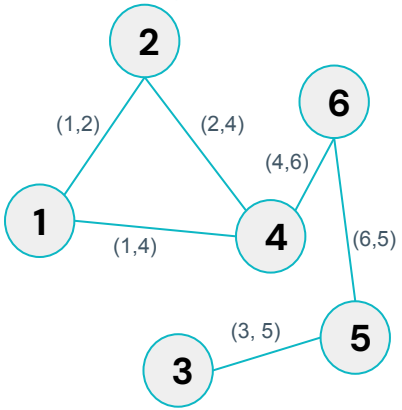
1. Fijamos una **longitud máxima** para el camino, **p** y **q**.
2. Partimos de **cierto nodo**.
3. Obtenemos sus **vecinos directos**.
   a. Si el vecino es el nodo previo, la probabilidad es **1/p.**
   b. Si es vecino directo del nodo previo, la probabilidad es **1**.
   c. Si no es ninguna de ambas (vecino lejano), la probabilidad es de **1/q.**
4. **Avanzamos** al nodo seleccionado.
5. Si ya hemos llegado a la longitud fijada **terminamos el algoritmo**, si no, **volvemos al paso 2**.



Simulemos un node2vec con p=5, q=2 empezando en el nodo 1...

5. Embeddings para aristas

Embeddings para aristas: enfoque trivial



1. Obtenemos **embeddings para los nodos**.
2. Los **agregamos** en función de las aristas que los conectan **mediante cierta operación**: suma, media, resta, máximo...

node	feat_0	feat_1	...	feat_n
1	0.213	-1.345		-1.324
2	1.231	-0.234		-0.324
3	-0.324	2.345		1.678
4	-0.121	1.234		2.453
5	1.234	-0.234		0.643
6	0.534	1.344		0.546

add

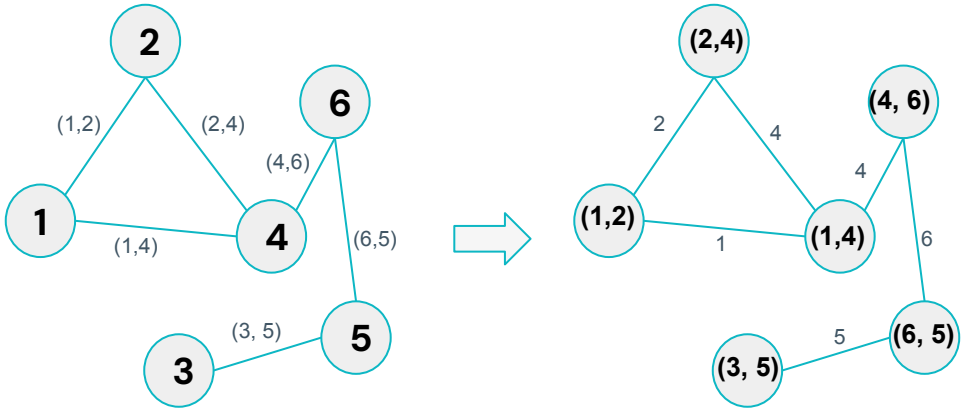
edge	feat_0	feat_1	...	feat_n
(1, 2)	0.213 + 1.231	-1.345 + (-0.234)		-1.324 + (-0.324)
(1, 4)	0.213 + (-0.121)	-1.345 + 1.234		-1.324 + 2.453
(2, 4)	1.231 + (-0.121)	-0.234 + 1.234		-0.324 + 2.453
(4, 6)	-0.121 + 0.534	1.234 + 1.344		2.453 + 0.546
(6, 5)	0.534 + 1.234	1.344 + (-0.234)		0.546 + 0.643
(3, 5)	-0.324 + 1.234	2.345 + (-0.234)		1.678 + 0.643

Embeddings para aristas: Line Graph

1. Generamos el **LineGraph**

Las **aristas pasan a ser los nodos**, estando conectados si entre las aristas existe un **nodo en común**.

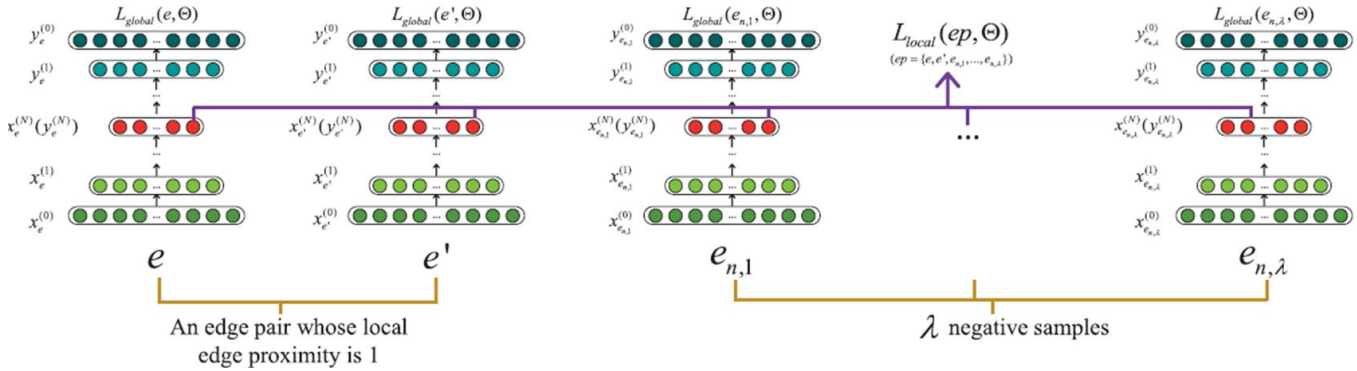
2. Calculamos los **embeddings sobre los nodos del LineGraph.**



Embeddings para aristas: Edge2Vec

Optimiza dos funciones de pérdida simultáneamente:

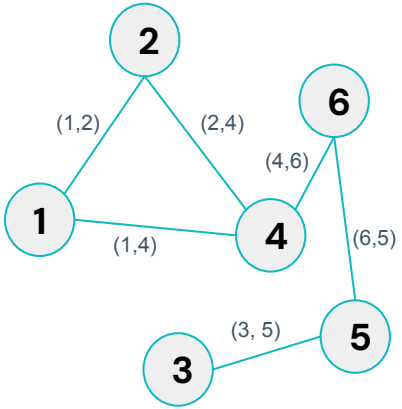
- **L global**: optimizando la matriz de adyacencia como entrada y salida de un autoencoder
- **L local**: la función de pérdida de node2vec



Wang, C., Wang, C., Wang, Z., Ye, X., & Yu, P. S. (2020). Edge2vec: Edge-based social network embedding. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *14*(4), 1-24.

6. Embeddings para grafos

Embeddings para grafos: enfoque trivial



1. Obtenemos **embeddings para los nodos**.
2. Los **agregamos** en función de **cierta operación**: suma, media, resta, máximo...

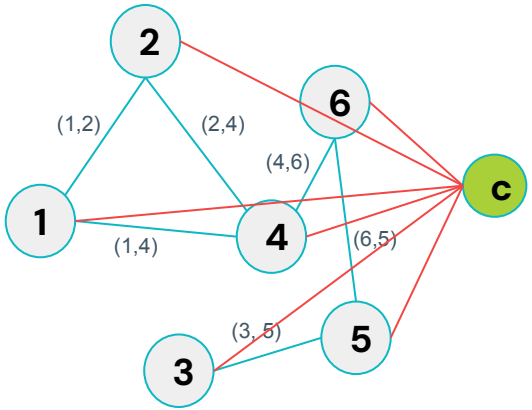
node	feat_0	feat_1	...	feat_n
1	0.213	-1.345		-1.324
2	1.231	-0.234		-0.324
3	-0.324	2.345		1.678
4	-0.121	1.234		2.453
5	1.234	-0.234		0.643
6	0.534	1.344		0.546

add

feat_0	feat_1	...	feat_n
0.213 + 1.231 + ... + 0.534	-1.345 + (-0.234) + ... + 1.344		-1.324 + (-0.324) + 0.546

Mismo enfoque que para aristas!

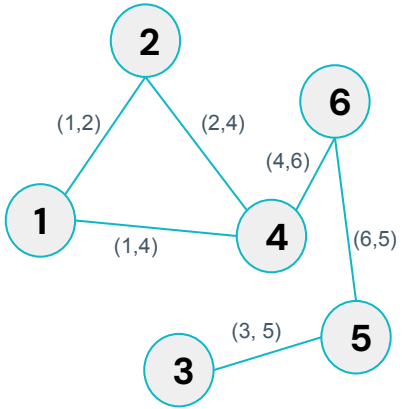
Embeddings para grafos: nodo centinela



1. Conectamos un nodo ***centinela*** a todos los nodos del grafo.
2. Obtenemos **embeddings para los nodos** incluido el **centinela**.
3. El embedding del grafo será el obtenido para el **centinela**.

Embeddings para grafos: graph kernels

Un **graph kernel** es un **vector de características** que definimos manualmente que nos permite comparar grafos.



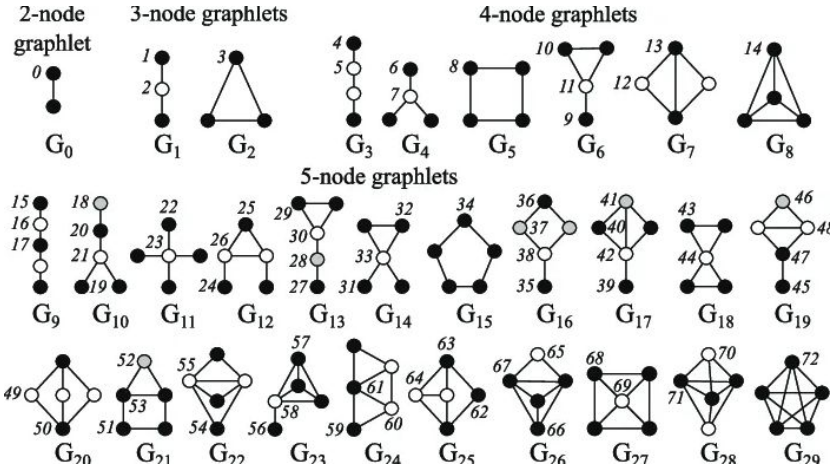
Grado: 0, 1, 2, 3

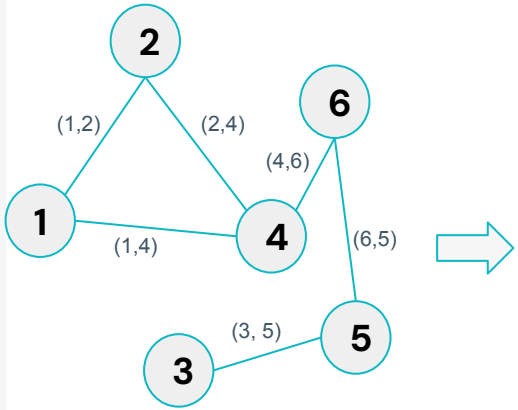
[0, 1, 4, 1]

Por ejemplo: cuantos nodos hay con cada grado

Embeddings para grafos: graph kernels

El conjunto de **graphlets** compone todas las posibles conexiones que se pueden dar con **n número de nodos.**





G0, G1, G2, G3, ..., GN

[6, 5, 1, 4, 1, 0, 1, 0, 0, 2, 1, 0, 0, 1, 0, …, 0]

Limitaciones de los Shallow Embeddings

Limitaciones

Estos métodos pertenecen al campo del ***transductive learning*** lo que significa que no pueden hacer predicciones sobre datos que no estaban presentes en el entrenamiento y **siempre que queramos predecir un elemento nuevo deberemos reentrenar**

Solo acumulan información posicional y **no integran los atributos propios de los nodos** (i.e en una red social acumularían la información de los seguidores de un usuario pero no de los datos del propio usuario)

Graph Neural Networks

Tema 2 - Shallow Embeddings

UNIVERSIDAD POLITÉCNICA DE MADRID

Máster
Deep Learning