



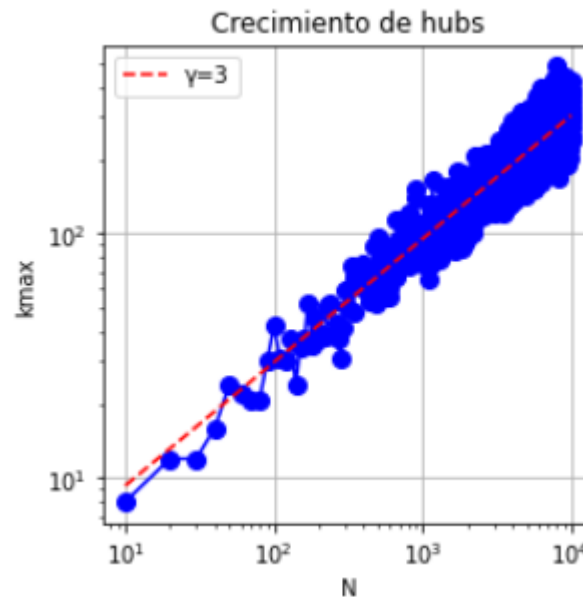
IIC-3641 Aprendizaje Automático basado en Grafos

<https://github.com/marcelomendoza/IIC3641>

- RECAPITULACIÓN -

El modelo Barabási-Albert explica la distribución de grado (AF3)

Las redes que vimos en la AF3 fueron generada por un modelo Barabási-Albert con $m=3$. Al observar el crecimiento de la conectividad de los hubs en función de N ($\sim t$), medimos:



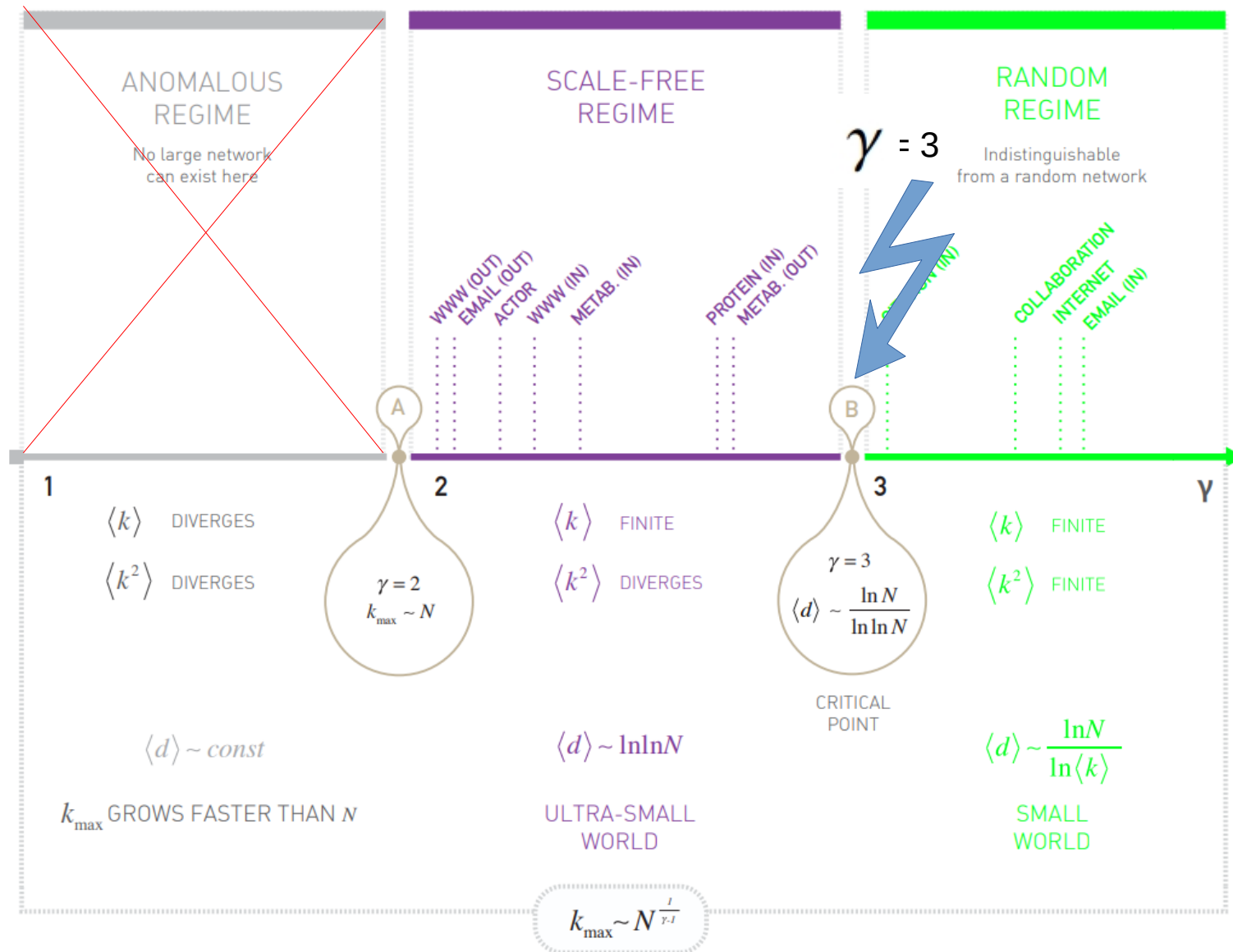
Recordar que crecimiento de hub $\sim t^{\frac{1}{\gamma-1}}$, y como la pendiente de la curva azul es $\frac{1}{2}$, entonces $\gamma = 3$.



Importante: Al simular varias redes con N distinto lo que en realidad hicimos fue simular el crecimiento de una red (**simulación de dinámica**).

El modelo Barabási-Albert explica la distribución de grado (AF3)

$\gamma = 3$ implica lo siguiente:

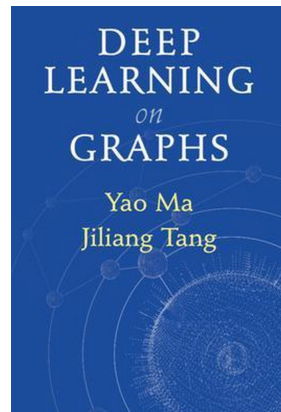


Exploren qué ocurre para otros valores de m (recordar que simulamos con $m=3$).

- VECTORIZACIÓN DE GRAFOS -

Vectorización de grafos

Capítulo 4



Vamos a vectorizar grafos para que sean *machine readable* y los podamos incorporar en modelos de *machine learning*.

Comenzaremos con vectorización de nodos. El objetivo es mapear cada nodo de un grafo en un **espacio de baja dimensionalidad que preserve información clave del nodo en el grafo original**.

Preguntas relevantes

¿Cuál información es conveniente preservar?

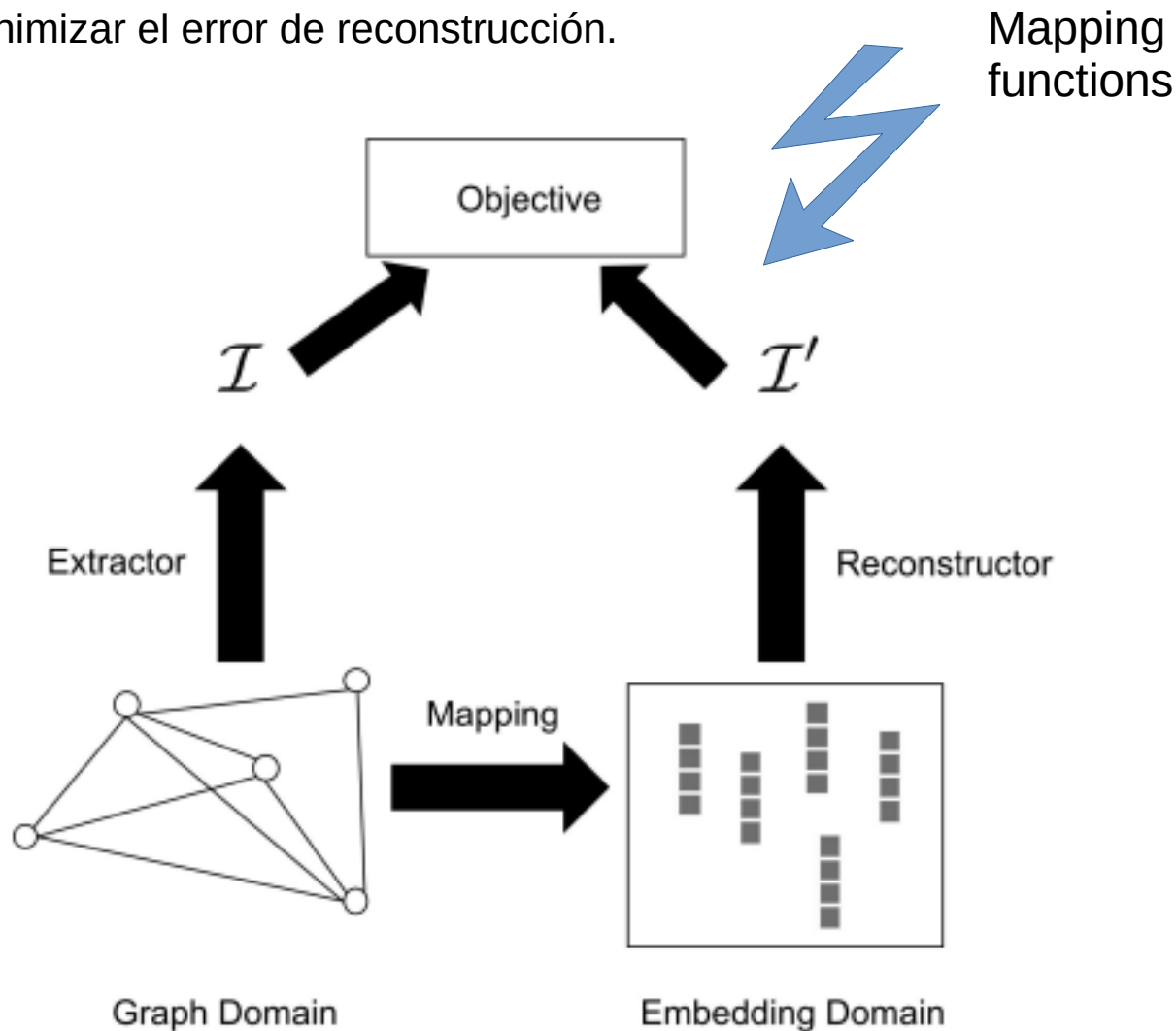
¿Cómo preservamos esta información en el espacio de representación?

Idea clave de diseño: Minimizar el error de reconstrucción del grafo original a partir de los **node embeddings**.

Vectorización de grafos

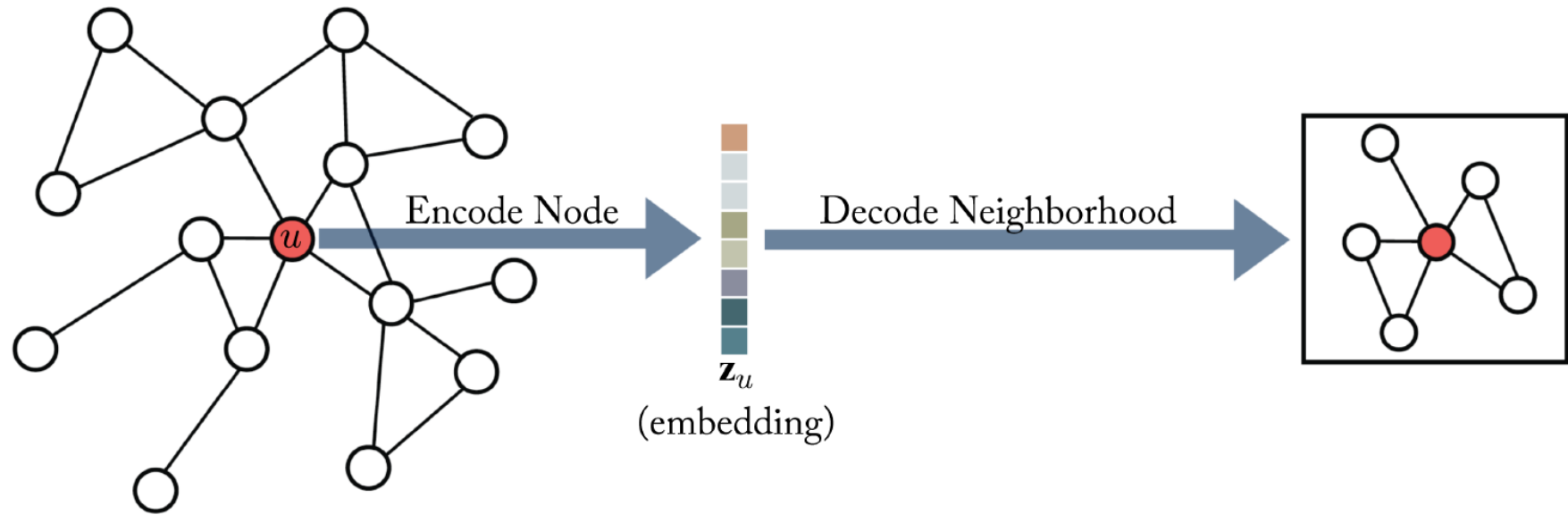
Vamos a tener:

- Information extractor (encoder) que preserva información del grafo original.
- Information reconstructor (decoder) que usa los embeddings para reconstruir el grafo original.
- El objetivo es minimizar el error de reconstrucción.



Vectorización de grafos

La idea de reconstruir un grafo es similar a pensar en un proceso encoder-decoder:



Buena parte de los modelos de node embeddings se basan en la idea de que si somos capaces de **reconstruir el vecindario de cada nodo del grafo**, tendremos una buena reconstrucción del grafo completo.

Preservación de co-ocurrencias de nodos

La idea más usada para preservar co-ocurrencia de nodos es usar **random walks**. Los nodos se considerarán similares si tienden a co-ocurrir en los random walks.

La función de mapping se optimiza de manera que las representaciones de nodos permitan reconstruir la similitud extraída desde los random walks.

Random walk: Sea $G = \{V, E\}$ un grafo conexo. Consideramos un random walk que comienza en el nodo $v^{(0)}$ en G . Asumimos que en el step t del random walk, estamos en el nodo $v^{(t)}$. La probabilidad de transición en el random walk está dada por:

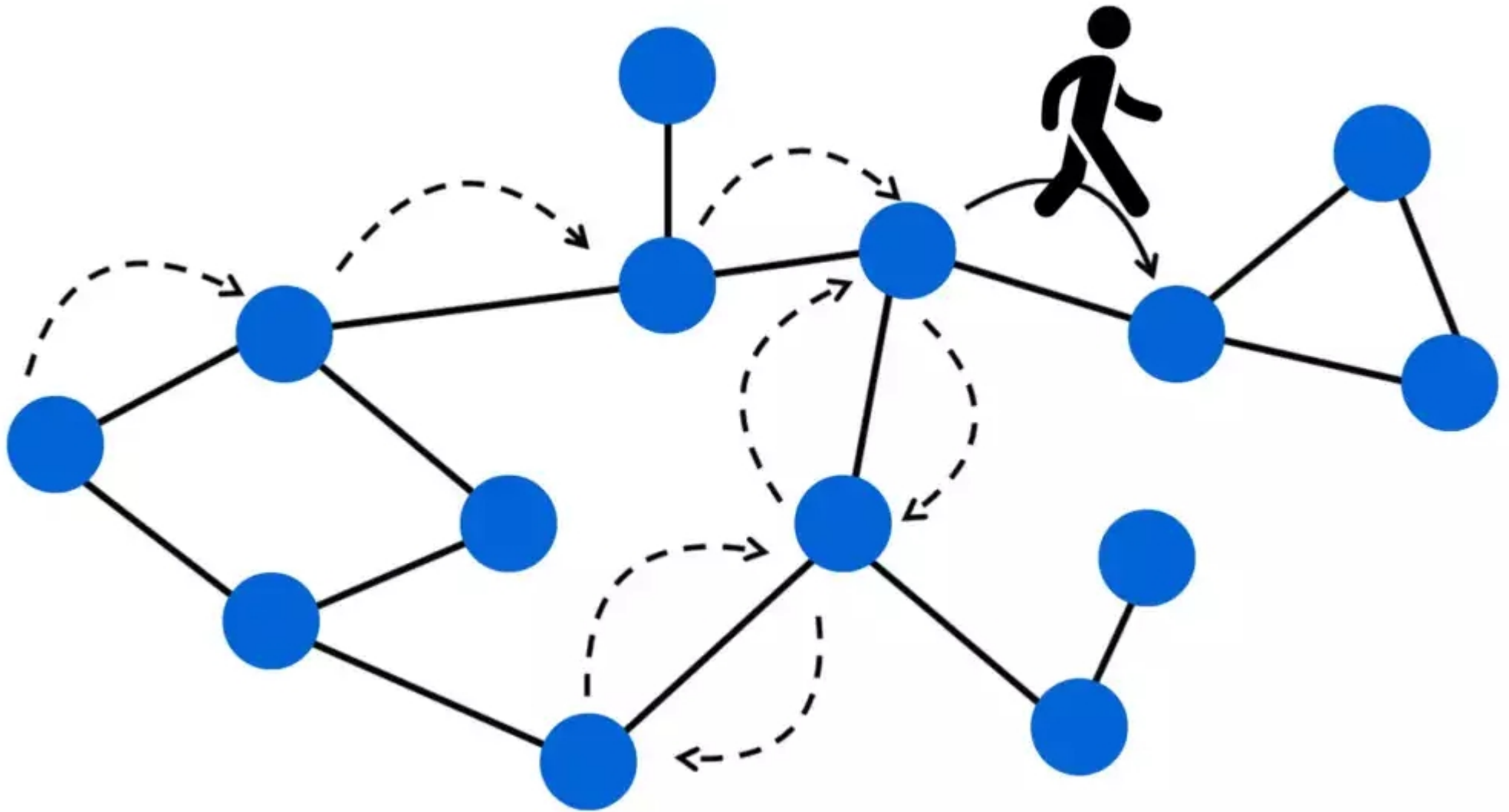
$$p(v^{(t+1)} \mid v^{(t)}) = \begin{cases} \frac{1}{k(v^{(t)})}, & \text{si } v^{(t+1)} \in \mathcal{N}(v^{(t)}) \\ 0, & \text{en caso contrario,} \end{cases}$$

donde $\mathcal{N}(v^{(t)})$ corresponde al conjunto de vecinos del nodo. En otras palabras, el siguiente nodo se toma al azar de los vecinos del nodo actual.

Los random walks quedarán parametrizados por su largo T .

Para cada nodo del grafo, se realizarán γ random walks.

Los random walks permiten formar secuencias de nodos



Preservación de co-ocurrencias de nodos

Algorithm 1: Generating Random Walks

```
1 Input:  $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}, T, \gamma$ 
2 Output:  $\mathcal{R}$ 
3 Initialization:  $\mathcal{R} \leftarrow \emptyset$ 
4 for  $i$  in  $\text{range}(1, \gamma)$  do
5   for  $v \in \mathcal{V}$  do
6      $\mathcal{W} \leftarrow RW(\mathcal{G}, v^{(0)}, T)$ 
7      $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathcal{W}\}$ 
8   end
9 end
```

Los random walks pueden ser tratados como las oraciones de word2vec, donde los nodos son el vocabulario. Al igual que en skip-grams, podemos preservar información de las oraciones capturando la co-ocurrencia de nodos en los random walks.

Básicamente la idea es la misma. Tenemos un nodo central $v^{(i)}$ y lo pareamos con los nodos dentro de una ventana de contexto w . Los pares se usan para entrenar el modelo.

Preservación de co-ocurrencias de nodos

Tenemos un nodo central $v^{(i)}$ y lo pareamos con los nodos dentro de una ventana de contexto w . Los pares se usan para entrenar el modelo.

Algorithm 2: Extracting Co-occurrence

```
1 Input:  $\mathcal{R}, w$ 
2 Output:  $\mathcal{I}$ 
3 Initialization:  $\mathcal{I} \leftarrow []$ 
4 for  $\mathcal{W}$  in  $\mathcal{R}$  do
5   for  $v^{(i)} \in \mathcal{W}$  do
6     for  $j$  in  $\text{range}(1, w)$  do
7        $\mathcal{I}.\text{append}((v^{(i-j)}, v^{(i)}))$ 
8        $\mathcal{I}.\text{append}((v^{(i+j)}, v^{(i)}))$ 
9     end
10  end
11 end
```

El modelo (una red feed-forward de dos capas) genera representaciones para los nodos, ya sea en su rol de contexto o como nodo central:

$$f_{cen}(v_i) = \mathbf{u}_i = e_i^\top \mathbf{W}_{cen}$$

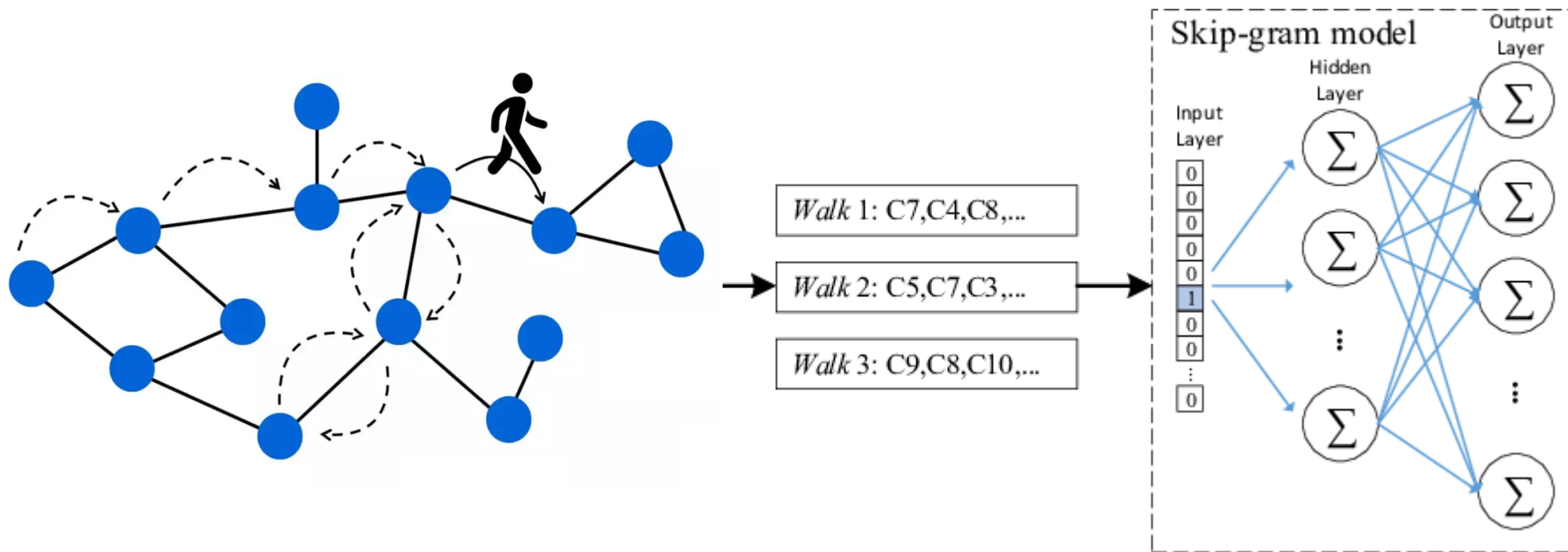
$$f_{con}(v_i) = \mathbf{v}_i = e_i^\top \mathbf{W}_{con}.$$



One hot para
posición i

Entrenando la red para preservar localidad de nodos

Una vez tenemos los pares de nodos, digamos nodo central y nodo de contexto, entrenamos una red neuronal con una capa softmax de salida.



Dado un nodo central a la entrada, en forward, la red debiera entregar una mayor probabilidad a los nodos que co-ocurren en los random walks. Esta tarea ayuda a que los embeddings que aprende la red preservan los vecindarios del nodo central.

Preservación de co-ocurrencias de nodos

La red se entrena para aprender por medio de una capa softmax lo siguiente:

$$p(v_{con}|v_{cen}) = \frac{\exp(f_{con}(v_{con})^\top f_{cen}(v_{cen}))}{\sum_{v \in \mathcal{V}} \exp(f_{con}(v) f_{cen}^\top(v_{cen}))}$$

que puede ser interpretado como la información reconstruida desde el espacio latente de embeddings de nodos del grafo.

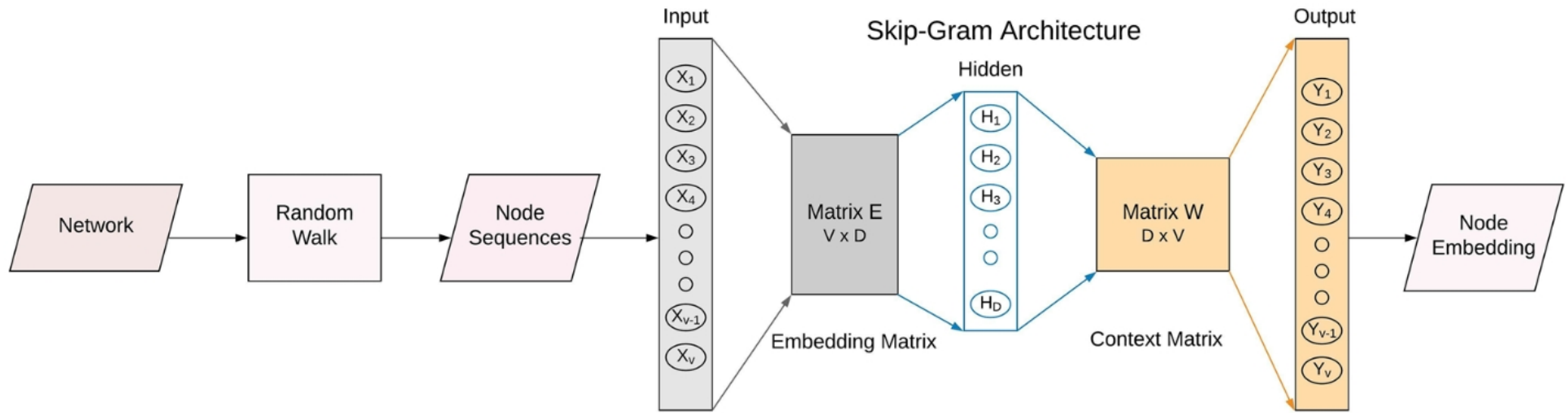
Para que la información reconstruida desde el espacio latente sea precisa, se deben reconstruir con alta probabilidad tuplas extraídas desde el grafo y con muy baja probabilidad tuplas generadas al azar. Al igual que en **SkipGrams**, se introduce negative sampling para que esto ocurra. En consecuencia, la función de pérdida del modelo es:

$$\mathcal{L}(\mathbf{W}_{con}, \mathbf{W}_{cen}) = \sum_{(v_{con}, v_{cen}) \in \text{set}(\mathcal{I})} \#(v_{con}, v_{cen}) \cdot \left(\log \sigma \left(f_{con}(v_{con})^\top f_{cen}(v_{cen}) \right) + \sum_{i=1}^k E_{v_n \sim P_n(v)} \left[\log \sigma \left(-f_{con}(v_n)^\top f_{cen}(v_{cen}) \right) \right] \right).$$

Este método para construcción de embeddings se conoce como **DeepWalk**.

Preservación de co-ocurrencias de nodos

La idea de **DeepWalk** es muy parecida a la de SkipGrams de NLP:



Analogía: Cada nodo del grafo corresponde a una palabra del vocabulario. Luego, se usan los pesos de la red como lookup tables.


Básicamente, estos embeddings son estáticos, es decir, siguiendo con la analogía de NLP, no se considera un contexto.

¿Qué es el contexto en el grafo? Volveremos a esto más adelante.

Preservación de co-ocurrencias de nodos

Existen variantes de DeepWalk basadas en la misma idea de preservación de co-ocurrencias de nodos por medio de random walks. Las más célebres son node2vec y LINE. En esta clase veremos node2vec.

Node2vec modifica el random walk definiendo un *biased random walk* (random walk de segundo orden) con parámetros p y q . En lugar de decidir el siguiente nodo del random walk de manera uniforme sobre los vecinos de $v^{(t)}$, la probabilidad se define en base tanto a $v^{(t)}$ como a $v^{(t-1)}$:

$$\alpha_{pq}(v^{(t+1)} \mid v^{(t-1)}, v^{(t)}) = \begin{cases} \frac{1}{p}, & \text{if } d(v^{(t-1)}, v^{(t+1)}) = 0 \\ 1, & \text{if } d(v^{(t-1)}, v^{(t+1)}) = 1 \\ \frac{1}{q}, & \text{if } d(v^{(t-1)}, v^{(t+1)}) = 2 \end{cases}$$


Se aplica normalización para que sume 1.

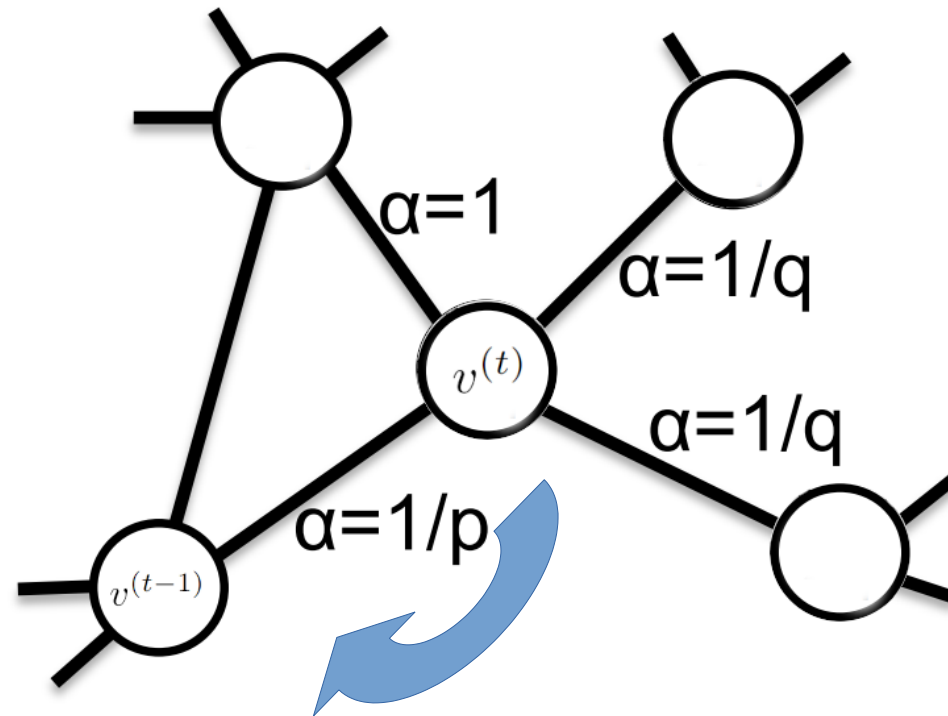
El parámetro p controla la probabilidad de visitar el nodo $v^{(t-1)}$. Un valor grande de p asegura que el random walk no vuelva a visitar nodos ya visitados.

Cuando $q > 1$ el random walk se sesga a visitar nodos cercanos a $v^{(t-1)}$ (\sim BFS). Si $q < 1$, el random walk tenderá a visitar nodos lejanos a $v^{(t-1)}$ (\sim DFS). Observemos que en el fondo esto controla si el recorrido es BFS o DFS.

Preservación de co-ocurrencias de nodos

Node2vec

$$\alpha_{pq}(v^{(t+1)} \mid v^{(t-1)}, v^{(t)}) = \begin{cases} \frac{1}{p}, & \text{if } d(v^{(t-1)}, v^{(t+1)}) = 0 \\ 1, & \text{if } d(v^{(t-1)}, v^{(t+1)}) = 1 \\ \frac{1}{q}, & \text{if } d(v^{(t-1)}, v^{(t+1)}) = 2 \end{cases}$$



Una vez se calculan los alpha, se dividen por su suma para que nos de una probabilidad.

Preservación de co-ocurrencias de nodos

Node2vec: Si aprendemos node embeddings, los podemos usar para aprender edge embeddings.

$$u \longleftrightarrow v$$

Operator	Symbol	Definition
Average	\boxplus	$[f(u) \boxplus f(v)]_i = \frac{f_i(u) + f_i(v)}{2}$
Hadamard	\boxdot	$[f(u) \boxdot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\ \cdot\ _{\bar{1}}$	$\ f(u) \cdot f(v)\ _{\bar{1}i} = f_i(u) - f_i(v) $
Weighted-L2	$\ \cdot\ _{\bar{2}}$	$\ f(u) \cdot f(v)\ _{\bar{2}i} = f_i(u) - f_i(v) ^2$

¿Cómo accionar el modelo?

Tareas: clasificación de nodos. Usamos los embeddings para entrenar un clasificador.

Preservación de co-ocurrencias de nodos

Node2vec: Sensibilidad sobre BlogCatalog (d : dimensionalidad del embedding).

- BlogCatalog: red social. N = 10,312 nodos, 333,983 enlaces, 39 clases (intereses de los usuarios).

