



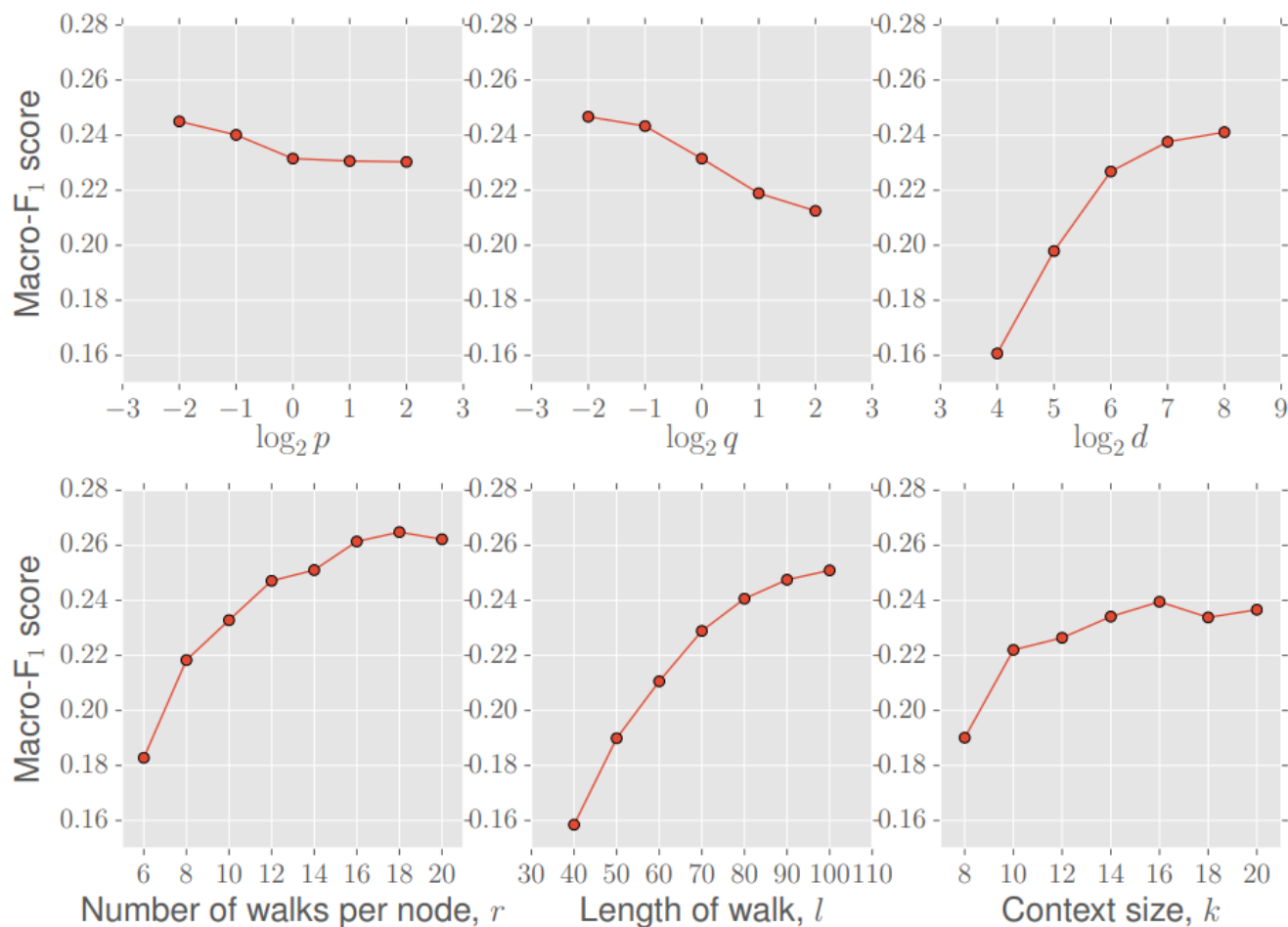
# **IIC-3641 Aprendizaje Automático basado en Grafos**

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

## - RECAPITULACIÓN -

## En Node2vec existe una dependencia de los hiperparámetros (AF4)

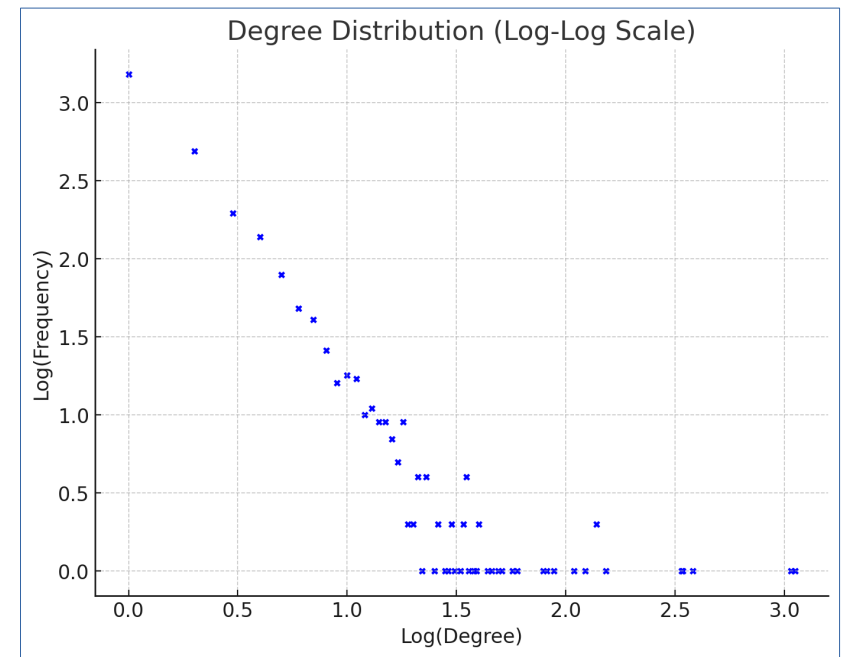
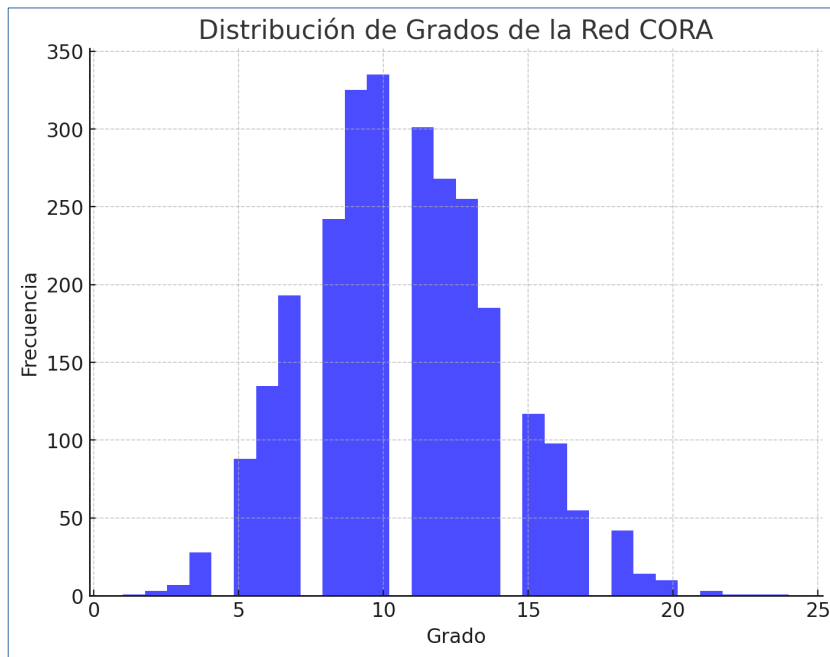
El clasificador que entrenamos sobre los nodos de CORA en la AF4 tenía dependencia de los hiperparámetros. En específico medimos dependencia de walk length y  $q$ , pero hay más hiperparámetros:



Importante: Los embeddings aprendidos desde la estructura de random walks son útiles para entrenar clasificadores a nivel de nodos.

## ¿Qué es CORA?

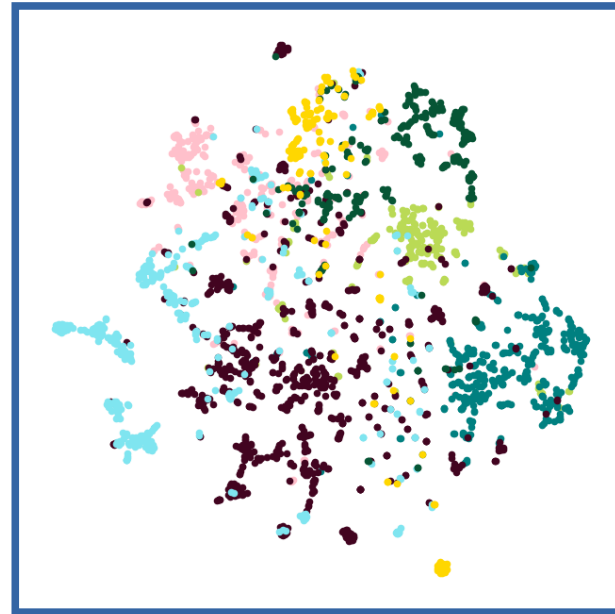
- Red dirigida: Representa las citas entre los artículos. Si el artículo A cita al artículo B, entonces existe un enlace dirigido de A a B.
- No ponderada: Los enlaces generalmente no tienen pesos asociados, solo indican la existencia de una cita.
- Número de nodos: Alrededor de 2,708 artículos.
- Número de enlaces: Aproximadamente 5,278 citas (depende de la versión).



¿Tendrá la propiedad scale-free? Verifíquelo.

## ¿Qué es CORA?

CORA es un tipo de red heterogénea, ya que los nodos tienen atributos. Es decir, CORA tiene varios tipos de nodos, dependiendo de a cuál disciplina pertenece el paper que representa.



Case\_Based  
Genetic\_Algorithms  
Neural\_Networks  
Probabilistic\_Methods  
Reinforcement\_Learning  
Rule\_Learning  
Theory

**Grafo heterogéneo**: Un grafo heterogéneo  $G$  consiste de un set de nodos  $V$  y un conjunto de enlaces  $E$ , donde cada nodo y cada enlace tienen un tipo asociado. Sea  $T_n$  el conjunto de tipos de nodos y  $T_e$  el conjunto de tipos de enlaces. Un grafo heterogéneo podría tener hasta dos tipos de funciones, uno para nodos y otro para enlaces, de manera que cada función determina el tipo de nodo y enlace, respectivamente, es decir:

$$\phi_n : \mathcal{V} \rightarrow \mathcal{T}_n \qquad \phi_e : \mathcal{V} \rightarrow \mathcal{T}_e$$

Observe que el clasificador de la AF4 aproximó  $\phi_n$ .

## - GRAPH NEURAL NETWORKS -

## GNN: Una arquitectura diseñada para grafos

La idea de las **Graph Neural Networks (GNN)** es disponer de una arquitectura de redes neuronales artificiales diseñada para grafos.

Inicialmente usaremos la arquitectura para aprender **node embeddings**, los cuales podremos usar para varias tareas, como por ejemplo clasificación y clustering de nodos.

Tenemos  $N$  nodos, la dimensionalidad de los embeddings de los nodos es  $F$  y eventualmente los nodos tienen atributos, en este caso  $C$  atributos.

Inicialmente sus valores pueden estar dados por datos, o bien pueden inicializarse al azar. En este último caso  $C = F$ .

La GNN tendrá varias capas, de  $N \times F$  parámetros. Si la red tiene  $K$  capas tenemos las siguientes matrices de parámetros de nodos:

$$H^k \in \mathbb{R}^{N \times F}, k \in \{1, 2, \dots, K\}$$

Notamos además:

Matriz de adyacencia:  $A \in \mathbb{R}^{N \times N}$

Atributos de los nodos:  $X \in \mathbb{R}^{N \times C}$

## GNN: Una arquitectura diseñada para grafos

La idea de las **Graph Neural Networks (GNN)** es actualizar iterativamente las representaciones de los nodos combinando representaciones de sus vecinos y su propia representación.

Inicialmente, la representación de cada nodo corresponde a sus atributos (al azar o bien datos), es decir:

$$H^0 = X$$

En cada capa definimos dos funciones:

AGGREGATE: agrega la información de los vecinos del nodo,

COMBINE: combina la información de AGGREGATE con la representación actual del nodo.

Formalmente:

Initialization:  $H^0 = X$

For  $k = 1, 2, \dots, K$ ,

$$a_v^k = \mathbf{AGGREGATE}^k \{H_u^{k-1} : u \in N(v)\}$$

$$H_v^k = \mathbf{COMBINE}^k \{H_v^{k-1}, a_v^k\},$$

donde  $N(v)$  corresponde al vecindario del nodo  $v$ .



## GNN: Una arquitectura diseñada para grafos

Se considera que la representación de la última capa es la representación final de los nodos. Notemos entonces que podemos usar estos embeddings para downstream tasks, como por ejemplo, clasificar nodos. Entonces:

$$\hat{y}_v = \text{Softmax}(W H_v^\top)$$

↓ fila  $v$ -ésima de la capa  $K$ -ésima

$$H^K$$

Si queremos entrenar la GNN para clasificación de nodos y tenemos etiquetas para algunos nodos del grafo, digamos  $n_l$  nodos, entrenamos en base a la función de pérdida:

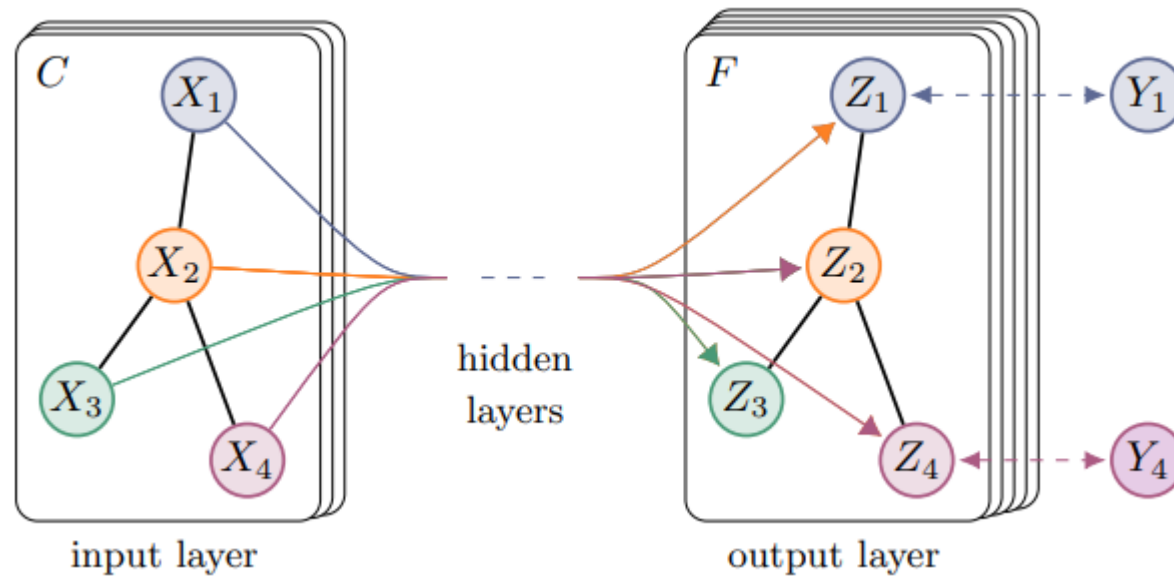
$$\frac{1}{n_l} \sum_{i=1}^{n_l} \text{loss}(\hat{y}_i, y_i)$$

↖ usualmente cross-entropy

Lo anterior representa un framework de GNN. Al definir AGGREGATE y COMBINE tenemos distintos tipos de GNN.

## GNN: Una arquitectura diseñada para grafos

Observamos que los nodos del grafo para los cuales tenemos etiquetas se consideran la partición de entrenamiento. Los otros nodos son parte del entrenamiento pero ingresan sin etiqueta ya que son de testing.



Es decir, el framework de GNN define un escenario de entrenamiento semi-supervisado.

## Graph Convolutional Networks (GCN)

Es la arquitectura más popular debido a su simplicidad y efectividad en una variedad de tareas. En las GCN, las representaciones de los nodos de cada capa se actualizan según la siguiente regla de propagación:

$$H^{k+1} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^k W^k)$$

donde:

$\tilde{A} = A + \mathbf{I}$  : matriz de adyacencia modificada con autoenlaces, lo que permite incorporar las características del mismo nodo al hacer el update.

$\tilde{D}$  : es la matriz diagonal conocida como matriz de grado. Se calcula según:  $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ .

$\sigma(\cdot)$ : función de activación. Usualmente ReLU o Tanh.

$W^k \in \mathbb{R}^{F \times F'}$  es una matriz de transformación lineal, con F y F' dims de las capas k y k+1.

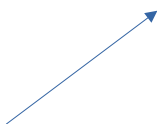


## Graph Convolutional Networks (GCN)

Veamos que pasa a nivel de nodo. Para eso separaremos la ecuación matricial en dos que operan a nivel de nodo, y que corresponden a las funciones AGGREGATE y COMBINE:

$$\text{AGGREGATE} \quad H_i^k = \sigma \left( \sum_{j \in \{N(i) \cup i\}} \frac{\tilde{A}_{ij}}{\sqrt{\tilde{D}_{ii} \tilde{D}_{jj}}} H_j^{k-1} W^k \right)$$

$$\text{COMBINE} \quad H_i^k = \sigma \left( \sum_{j \in N(i)} \frac{A_{ij}}{\sqrt{\tilde{D}_{ii} \tilde{D}_{jj}}} H_j^{k-1} W^k + \frac{1}{\tilde{D}_i} H_i^{k-1} W^k \right)$$

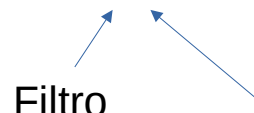


Es la función AGGREGATE que separa el vecindario del nodo mismo.



## Graph Convolutional Networks y Spectral Graph Convolutions

Las GCN tomaron inspiración de los filtros espectrales definidos para grafos. Una convolución espectral en un grafo se define como una multiplicación de una señal  $\mathbf{x}$  a nivel de nodo con un filtro convolucional en el dominio de Fourier. Formalmente:

$$g_{\theta} \star \mathbf{x} = U g_{\theta} U^T \mathbf{x}$$


Filtro

parámetros del filtro

Tenemos la matriz normalizada del Laplaciano del grafo definido por:

$$L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

En realidad, el Laplaciano corresponde a:

$$L = U \Lambda U^T$$

donde  $\Lambda$  es una matriz diagonal de valores propios. Entonces:

$$U^T \mathbf{x}$$

es la transformada de Fourier de grafo de la señal  $\mathbf{x}$ .

# Graph Convolutional Networks y Spectral Graph Convolutions

Las GCN tomaron inspiración de los filtros espectrales definidos para grafos. Una convolución espectral en un grafo se define como una multiplicación de una señal  $\mathbf{x}$  a nivel de nodo con un filtro convolucional en el dominio de Fourier. Formalmente:

$$g_{\theta} \star \mathbf{x} = U g_{\theta} U^T \mathbf{x}$$

Filtro                      parámetros del filtro

Tenemos la matriz normalizada del Laplaciano del grafo definido por:

$$L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

En realidad, el Laplaciano corresponde a:

$$L = U \Lambda U^T$$

donde  $\Lambda$  es una matriz diagonal de valores propios. Además:

$$U^T \mathbf{x}$$

es la transformada de Fourier de grafo de la señal  $\mathbf{x}$ .

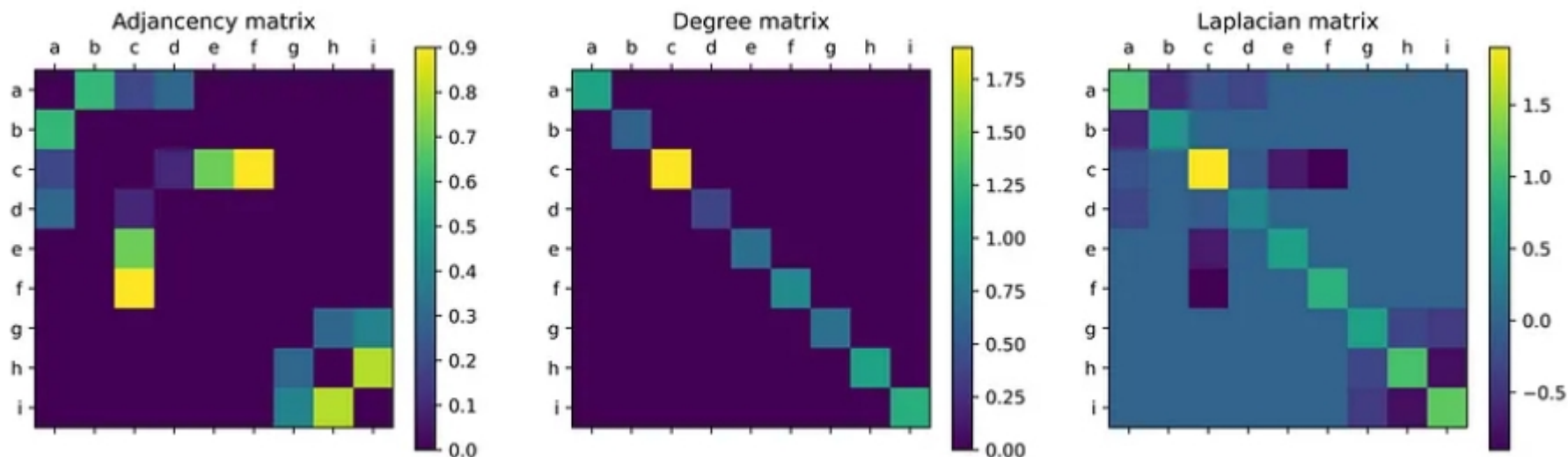


¿Esto es poco claro? Lo vamos a explicar mejor.

# Graph Convolutional Networks y Spectral Graph Convolutions

Veamos que representa cada uno de los elementos de la ecuación anterior. Comencemos con el Laplaciano de un grafo.

¿Qué es el Laplaciano de un grafo?  $L = D - A$ . Por ejemplo, en un grafo con pesos no dirigido:



Debido a la distribución de grado (power-law) de muchos grafos, los hubs pueden exhibir muchos más enlaces que el resto de los nodos (unidad 1). Para evitar el sobreajuste a los hubs, normalizamos  $L$  dividiendo por el grado de cada nodo (es decir dividiendo por  $D$ ):

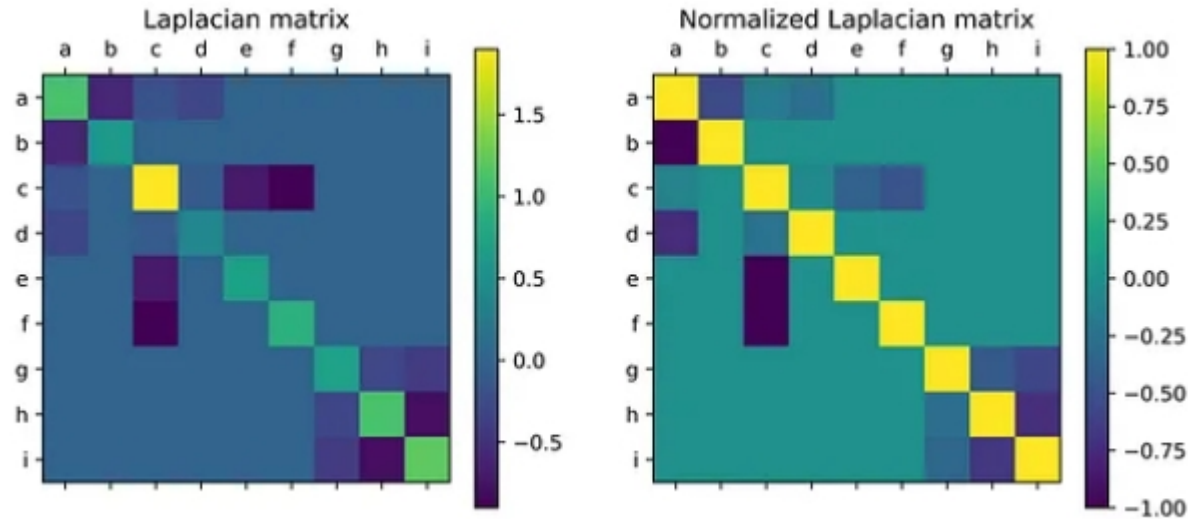
$$L = D^{-1}(D - A) = I - D^{-1}A$$

A esto lo llamamos Laplaciano normalizado del grafo.



# Graph Convolutional Networks y Spectral Graph Convolutions

Veamos que es el Laplaciano normalizado del grafo:



La normalización nos da valores entre  $[-1,1]$ .

El Laplaciano nos entrega información sobre la conectividad del grafo. El Laplaciano corrige las diferencias debidas a la distribución de grado para que las propiedades estructurales del grafo puedan ser estudiadas sin sesgar los resultados a los hubs.

¿Cómo se lee el Laplaciano?

- ~ 1 Conectividad directa
- ~ 0 No hay dependencia
- ~ -1 Conectividad indirecta (transitiva)



## Graph Convolutional Networks y Spectral Graph Convolutions

Veamos que indica el espectro de un grafo. Los valores propios se obtienen a partir de la ecuación característica aplicada al Laplaciano:

$$\det(L - \lambda I) = 0$$

Las soluciones de esta ecuación forman la matriz diagonal de valores propios:

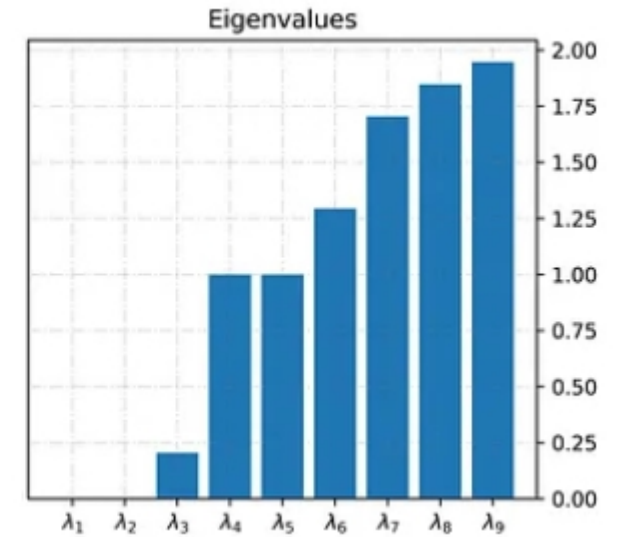
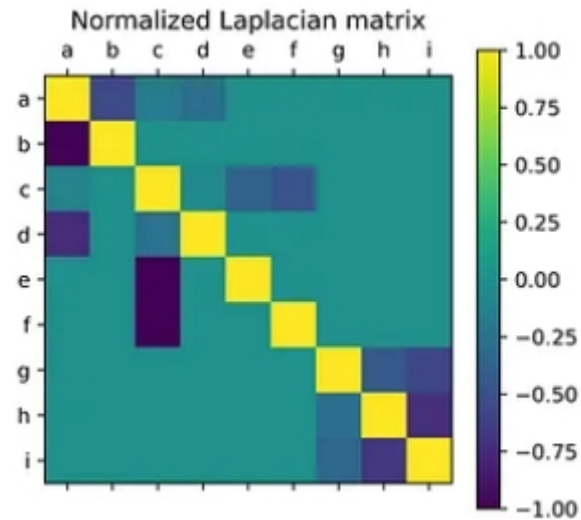
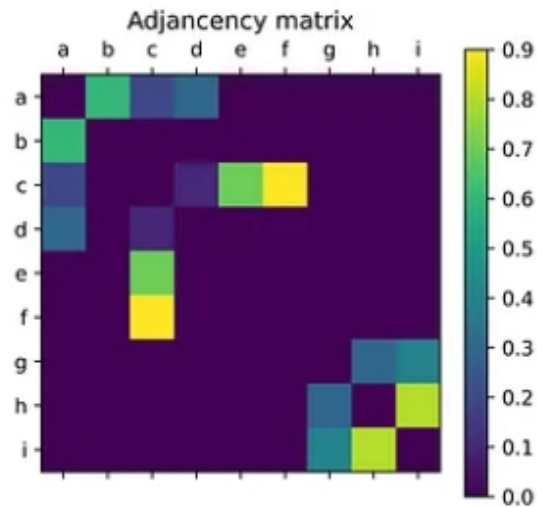
$$\Lambda = \begin{pmatrix} \lambda_1 & & & & \\ & \lambda_2 & & & \\ & & \lambda_3 & & \\ & & & \ddots & \\ & & & & \lambda_n \end{pmatrix}$$

Una propiedad del Laplaciano es que sus valores propios son reales. Además, los valores propios son interpretables y forman el espectro del grafo:

- El # de valores propios 0 indica la cantidad de particiones del grafo.

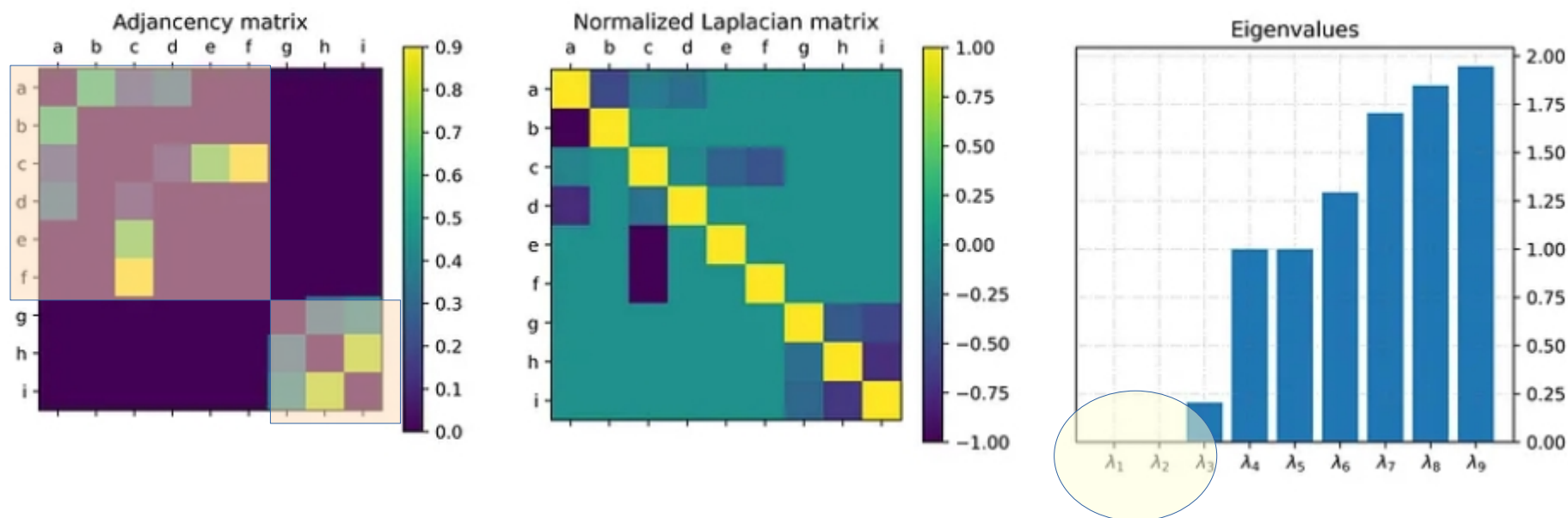
# Graph Convolutional Networks y Spectral Graph Convolutions

Volvamos al ejemplo:



# Graph Convolutional Networks y Spectral Graph Convolutions

Volvamos al ejemplo:



dos particiones

OK, y para cerrar la parte espectral del análisis del grafo, recordemos que podemos escribir la matriz original usando valores y vectores propios. En este caso, podemos reescribir el Laplaciano de la siguiente forma:

$$L = U \Lambda U^{-1} = U \Lambda U^T$$

Cada componente de cada vector propio está asociado a un nodo del grafo.

# Graph Convolutional Networks y Spectral Graph Convolutions

Vamos ahora a entender que es una Transformada de Fourier de grafo.

Transformada de Fourier clásica: descompone una señal en una suma de componentes sinusoidales.

Transformada de Fourier de grafo: descompone una señal como una combinación lineal de componentes espectrales del grafo (vectores propios de  $L$ ).

Sea  $x$  la señal a descomponer, la cual asigna un número real a cada nodo del grafo. La GFT de  $x$  es:

$$GFT_{[U]}(x) = U^T x = \hat{x}$$

Análogamente, se define la GFT inversa:

$$IGFT_{[U]}(\hat{x}) = U \hat{x} = x$$

En resumen, la GFT usa los vectores propios de  $L$  para obtener una representación de una señal en dos dominios: el dominio del grafo y el dominio del espectro del grafo.

# Graph Convolutional Networks y Spectral Graph Convolutions

**Teorema de la convolución**: la convolución entre dos señales es equivalente a la transformada inversa de Fourier del producto de sus transformadas de Fourier.

$$\mathcal{F}^{-1}\{\mathcal{F}\{f(t)\} \cdot \mathcal{F}\{g(t)\}\} = f(t) * g(t)$$

Lo vamos a aplicar a la GFT. Necesitamos el producto Hadamard:

Hadamard: 
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11} b_{11} & a_{12} b_{12} & a_{13} b_{13} \\ a_{21} b_{21} & a_{22} b_{22} & a_{23} b_{23} \\ a_{31} b_{31} & a_{32} b_{32} & a_{33} b_{33} \end{bmatrix}$$

Vamos a relacionar dos variables,  $x$  e  $y$ , que representan a  $h^{k-1}$  y  $h^k$ . Es decir,  $x \rightarrow y$  nos explica como pasar de la capa  $k-1$  a la capa  $k$ .



# Graph Convolutional Networks y Spectral Graph Convolutions

**Teorema de la convolución**: la convolución entre dos señales es equivalente a la transformada inversa de Fourier del producto Hadamard de sus transformadas de Fourier.

$$\mathcal{F}^{-1}\{\mathcal{F}\{f(t)\} \cdot \mathcal{F}\{g(t)\}\} = f(t) * g(t)$$

Lo vamos a aplicar a la GFT. Necesitamos el producto Hadamard:

Hadamard:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11} b_{11} & a_{12} b_{12} & a_{13} b_{13} \\ a_{21} b_{21} & a_{22} b_{22} & a_{23} b_{23} \\ a_{31} b_{31} & a_{32} b_{32} & a_{33} b_{33} \end{bmatrix}$$

Vamos a relacionar dos variables,  $x$  e  $y$ , que representan a  $h^{k-1}$  y  $h^k$ . Es decir,  $x \rightarrow y$  nos explica como pasar de la capa  $k-1$  a la capa  $k$ .



Vamos a definir un operador de convolución, ya que hacer operaciones directamente en el grafo es complejo. Recordemos que la convolución nos permite definir un Hadamard entre las GFT de las señales. Vamos:

$$x *_{\mathcal{G}} y = U((U^T x) \odot (U^T y))$$

## Graph Convolutional Networks y Spectral Graph Convolutions

La señal resultante,  $y$ , se obtiene aplicando el operador de convolución sobre  $x$ :

$$y = g_{\theta}(L)x = g_{\theta}(U\Lambda U^T)x = U g_{\theta}(\Lambda)U^T x$$

Si  $g_{\theta}(\Lambda) = \text{diag}(\theta)$ , donde los parámetros  $\theta \in \mathbb{R}^n$  corresponden a los coeficientes de la GFT, decimos que tenemos una **Spectral Network**.

El problema de las **Spectral Networks** es que computacionalmente es muy caro hacer forward y backward en estas redes.

Otra alternativa consiste en aproximar la convolución usando un filtro espectral de orden  $K$ :

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^K \theta'_k T_k(\tilde{\Lambda})$$

A esta red se le llama **ChebNet**. El problema es que solo retiene información local.

## Graph Convolutional Networks y Spectral Graph Convolutions

Las GCN lo que hacen es una simplificación de la operación de convolución, de manera que se pueda realizar directamente sobre la matriz de adyacencia, lo cual la hace mas eficiente:

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 (L - I_N) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$$

De hecho la versión definitiva propuesta por Kipf y Welling es aún más simple:

$$g_{\theta} \star x \approx \theta \left( I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x$$

Lo cual se puede reescribir como:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \quad \begin{array}{l} \nearrow \tilde{A} = A + I_N \\ \longrightarrow \tilde{D}_{ii} = \sum_j \tilde{A}_{ij} \end{array}$$

con  $\Theta \in \mathbb{R}^{C \times F}$  y  $Z \in \mathbb{R}^{N \times F}$

Una red de dos capas queda bastante sencilla:



$$Z = f(X, A) = \text{softmax} \left( \hat{A} \text{ReLU} \left( \hat{A} X W^{(0)} \right) W^{(1)} \right)$$

$\nearrow \hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$