IIC-3641 Aprendizaje Automático basado en Grafos

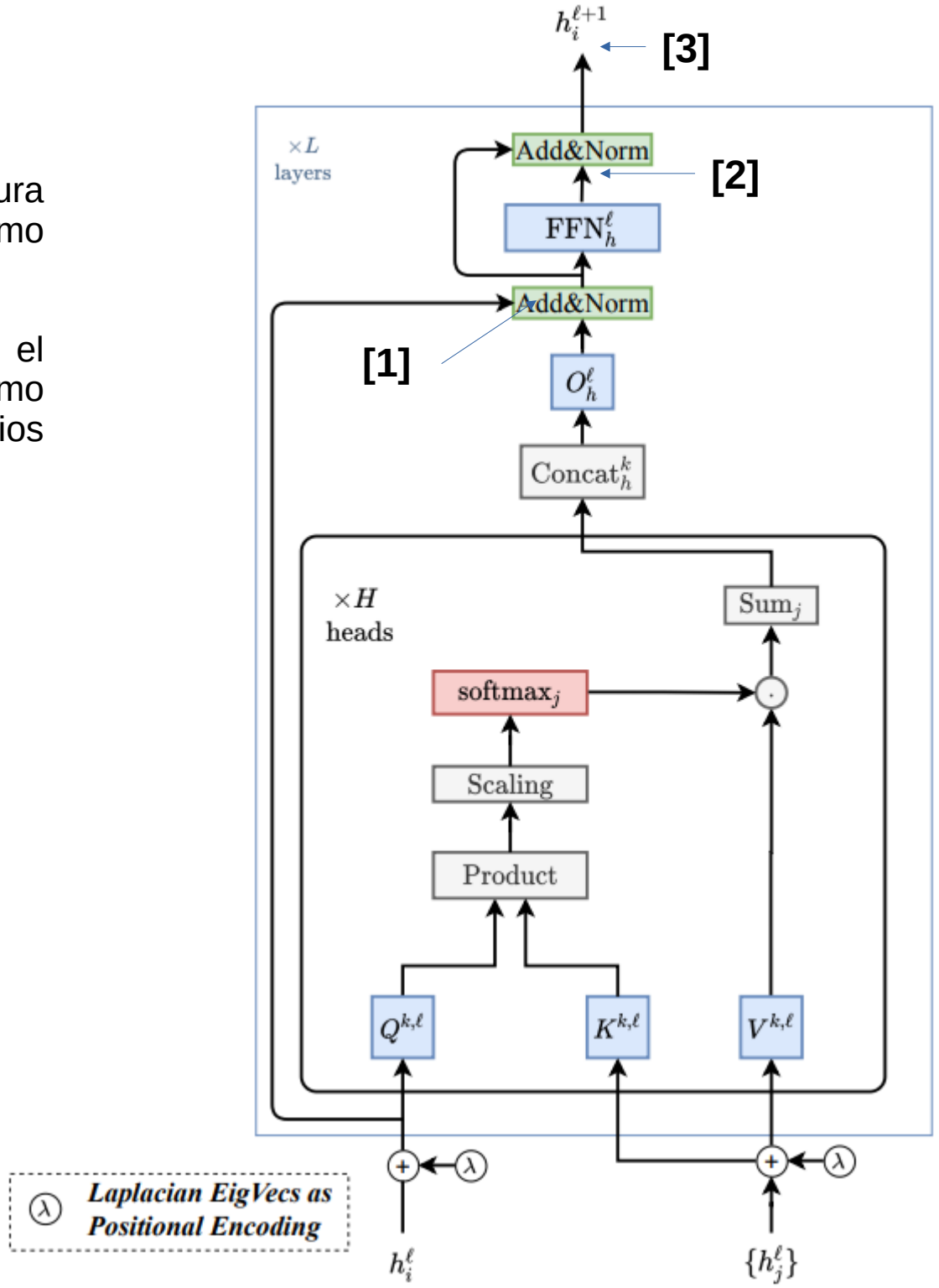
https://github.com/marcelomendoza/IIC3641

- RECAPITULACIÓN -

Graph Transformer

La clase pasada revisamos la arquitectura graph transformer, la cual opera como encoder.

Es un arquitectura que se basa en el transformer original y usa el mecanismo de atención para codificar los vecindarios de cada nodo objetivo.

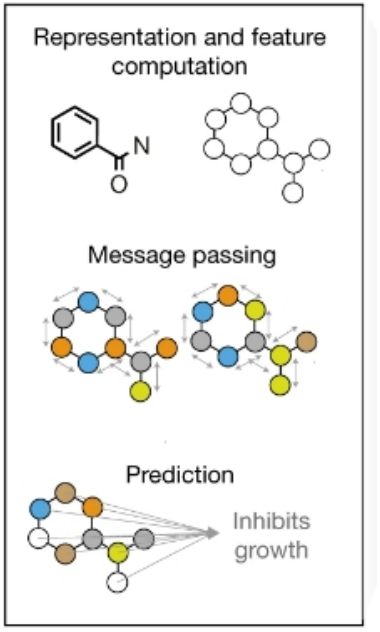


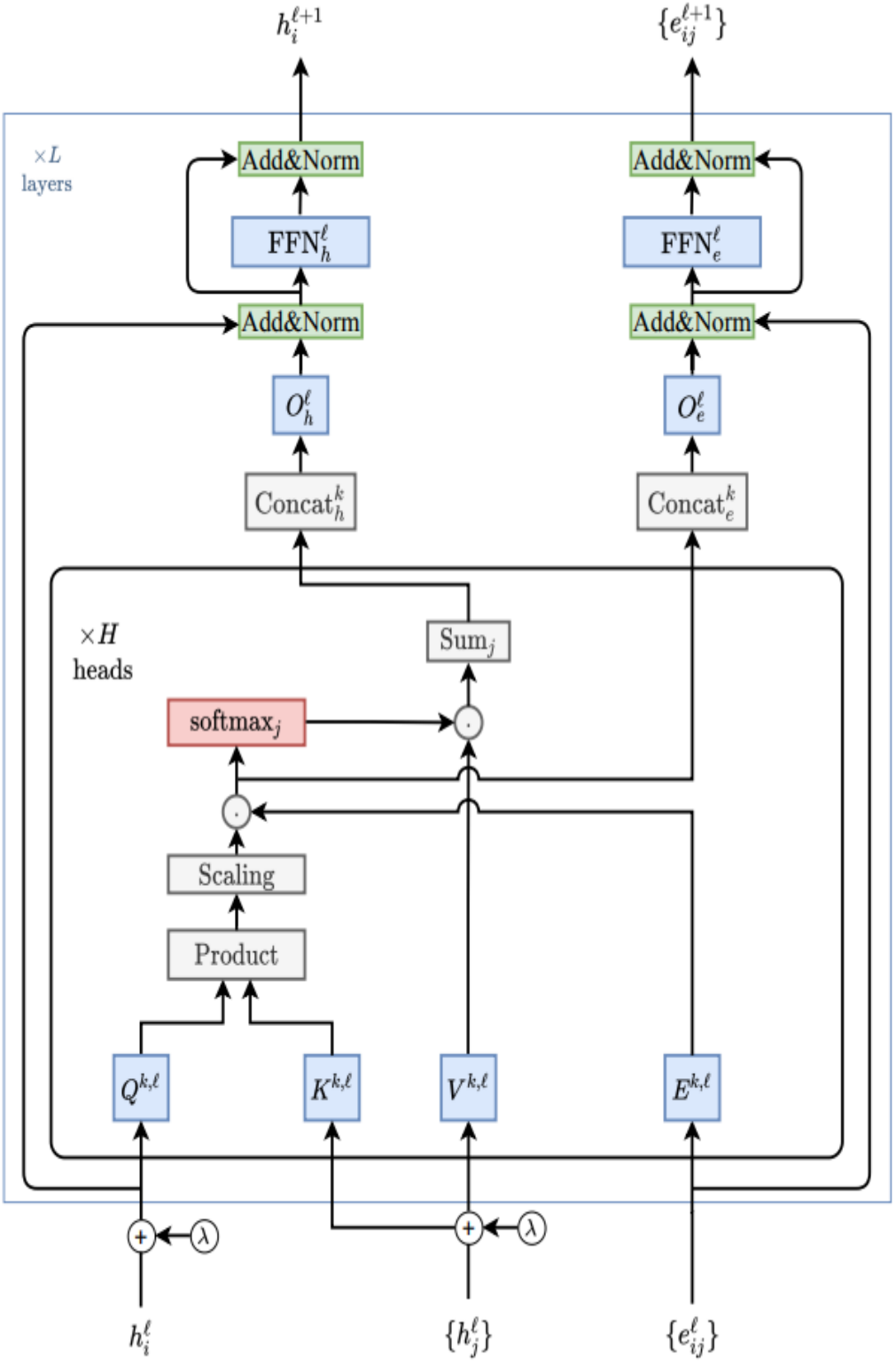
Graph Transformer

En la AF7 trabajamos con el graph transformer para construir encoders de grafos de moléculas (los nodos son átomos).

Para este caso usamos el graph transformer incluyendo embeddings de nodos y enlaces.

Aplicamos un *graph readout* para hacer regresión de grafos, usando agregación sobre nodos y enlaces. Eso nos permite explorar los efectos de una molécula.





- GRAPHORMER -

Graphormer

El graph transformer adapta el mecanismos de auto-atención para leer grafos secuenciados. Nosotros vimos como secuenciar moléculas y a partir de esto hicimos regresión de grafos.

- La atención es a los vecinos, lo cual se implementa usando convolución.
- Se introducen los embeddings derivados del Laplaciano para incluir información global.

El **graphormer** es una modificación del transformer de grafo original, el cual se basa en:

- Se introduce un *centrality encoding*, basado en alguna medida de centralidad del nodo.
- Introduce un *spatial encoding*, el que es un sesgo de atención en función de la distancia entre dos nodos (shortest path distance), generalizando la noción de vecindario directo (distancia 1).
- Atención global: el graphormer permite prestar atención a cualquier nodo del grafo, la cual está regulada por los sesgos estructurales. Requiere aumentar el tamaño de la entrada.

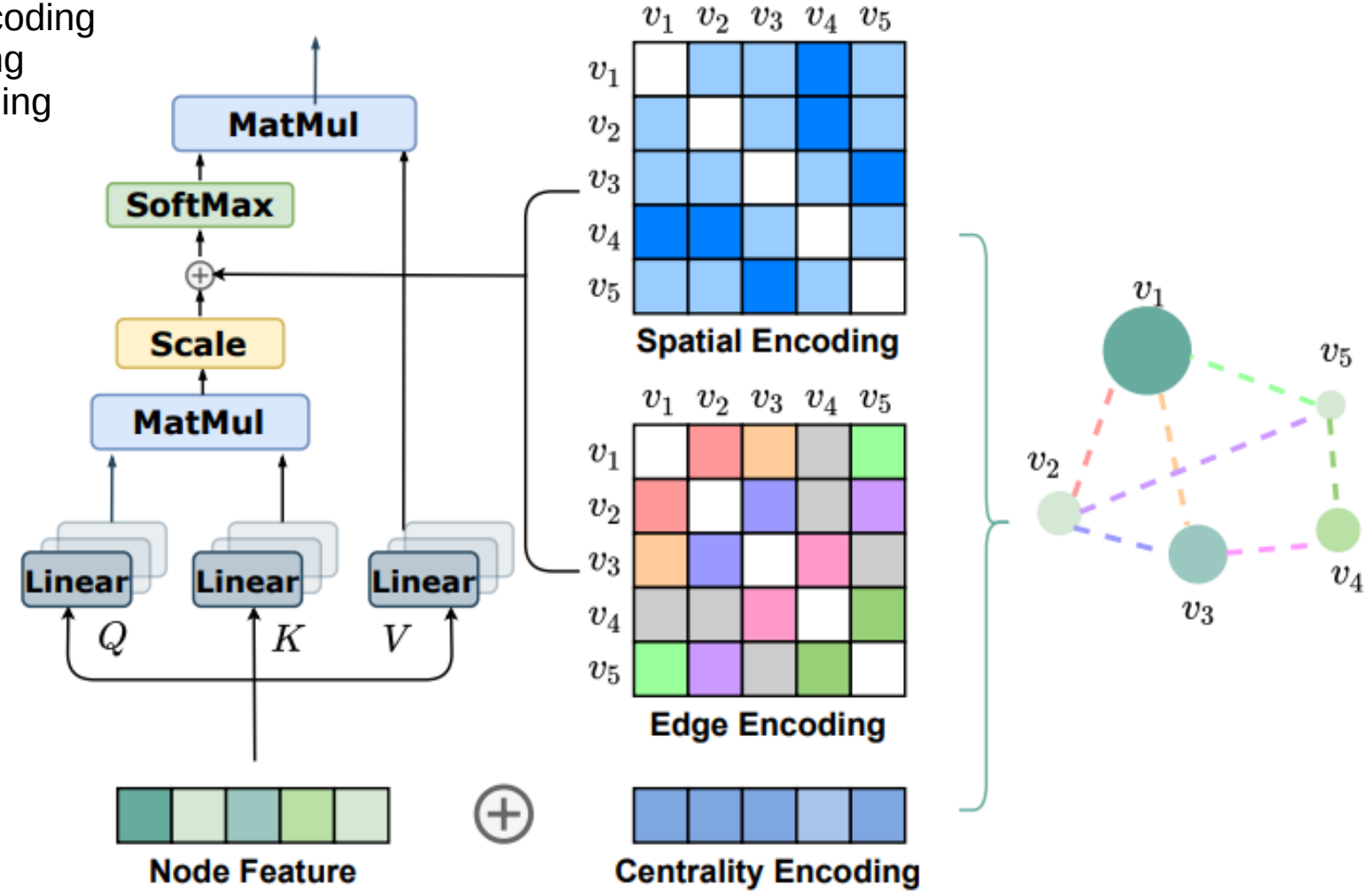
El graphormer tiene una librería y se ha usado para preentrenar modelso de grafos, en la línea de modelos fundacionales.

Ver aquí: https://graphormer.readthedocs.io/en/latest/

Graphormer

Hay tres modificaciones a las cuales prestar atención:

1) Centrality encoding
2) Edge encoding
3) Spatial encoding



Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., Schuurmans, D., & Liu, T. (2021). Do Transformers Really Perform Bad for Graph Representation? In NeurIPS 2021. https://arxiv.org/abs/2106.05234

Graphormer

Centrality encoding: se usa el grado del nodo como medida de centralidad. Luego se asigna un embedding de grado, es decir, **cada grado tiene un embedding** mediante una tabla de embedding (*look-up* por grado). La tabla es aprendible.

$$z_i = \text{Embedding}\big(\deg(i)\big)$$

Los embeddings se suman a la entrada del transformer:

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)}$$

Si la red es no dirigida, sólo se usa un embedding de centralidad.

Motivación: nodos mas centrales tienen en sesgo positivo que los hace mas influyentes en la red. Además, los nodos con grados similares tendrán em,beddings similares, lo cual ayuda a la detección de *assortativity*.

Graphormer

Spatial encoding: se usa al calcular los coeficientes de atención. Para esto se calcula un embedding asociado a cada distancia más corta (*look-up* por distancia).

- Se calculan las distancia más cortas entre los pares de nodos del grafo (preprocesamiento).

- Cada distancia se vincula a un embedding desde una tabla de embeddings aprendible:

$$\mathbf{s}_{ij} = \mathrm{Embedding}\big(d(i,j)\big)$$

- Se define una distancia máxima sobre la cual se umbraliza (clipea):

$$d(i,j) \geq d_{\max} \quad \Rightarrow \quad d(i,j) = d_{\max}$$

- En rigor, se usa una capa lineal para calcular el coeficiente de sesgo que se suma al coeficiente de atención.

$$b_{ij}^{\mathrm{spatial}} = f(\mathbf{s}_{ij})$$

Graphormer

Edge encoding: introduce información sobre las aristas que conectan a dos nodos. Introduce esta información en el coeficiente de atención.

- Si estoy codificando *i* y *j* (coeficiente de atención), tomamos la primera arista del SPD(*i*,*j*). Por ejemplo, si el camino más corto entre *i* y *j* es $i \rightarrow k \rightarrow j$, se toma la arista (*i, k*).

- Voy a hacer un *look-up* en la matriz de embeddings aprendible de los enlaces:

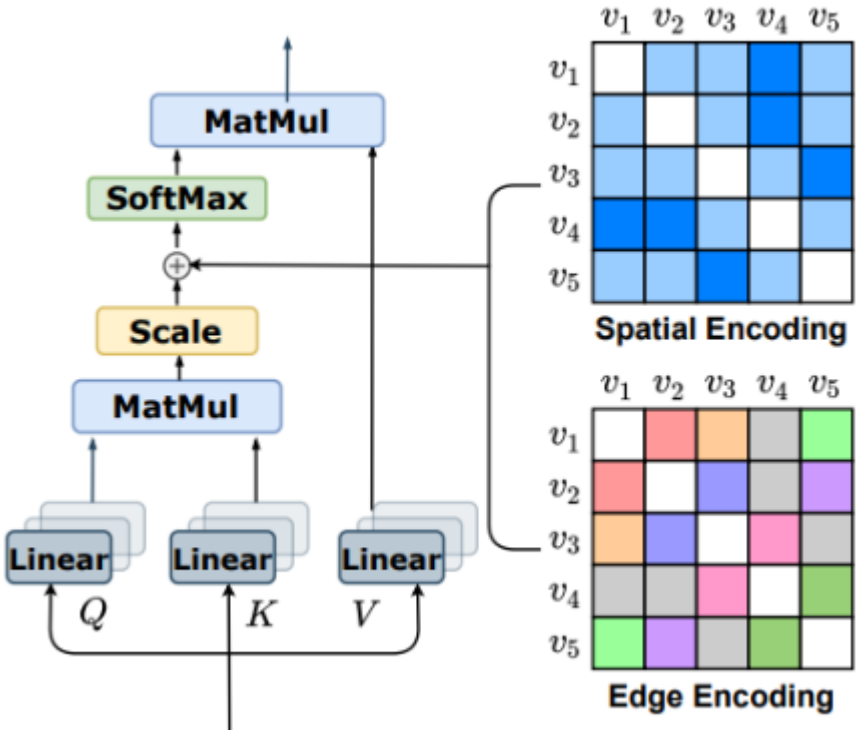
$$e_{ik} = \mathrm{Embedding}\big(e(i,k)\big)$$

- Finalmente, se usa una capa lineal para calcular el coeficiente de sesgo que se suma al coeficiente de atención:

$$b_{ij}^{\mathrm{edge}} = f(\mathbf{e}_{ij})$$

Graphormer

Coeficientes de atención: el graphormer usa self-attention con Q, K y V, pero los coeficientes de atención introducen los sesgos espaciales y de aristas.



El coeficiente de atención queda dado por:

$$\alpha_{ij} = \text{softmax}_j \left(\frac{q_i \cdot k_j}{\sqrt{d_k}} + b_{ij}^{\text{spatial}} + b_{ij}^{\text{edge}} \right)$$

Graphormer

El bloque del graphormer: hace algunos cambios, como hacer LayerNorm (LN) antes y no después. Usa multihead attention (MHA) y una FFN a la salida. También usa conexiones residuales. Las ecuaciones de transferencia quedan así:

$$h^{'(l)} = \mathbf{MHA}(\mathbf{LN}(h^{(l-1)})) + h^{(l-1)}$$
$$h^{(l)} = \mathbf{FFN}(\mathbf{LN}(h^{'(l)})) + h^{'(l)}$$

Como usa atención global, es parecido a lo que hace BERT. En este sentido, también usa un token especial, equivalente al [CLS] que en el graphormer se denomina [VNode].

En general, se usa como encoder y se entrena directo sobre la tarea específica, ya sea para clasificación de nodos, predicción de enlaces o tareas sobre grafos mediante pooling (graph readout).

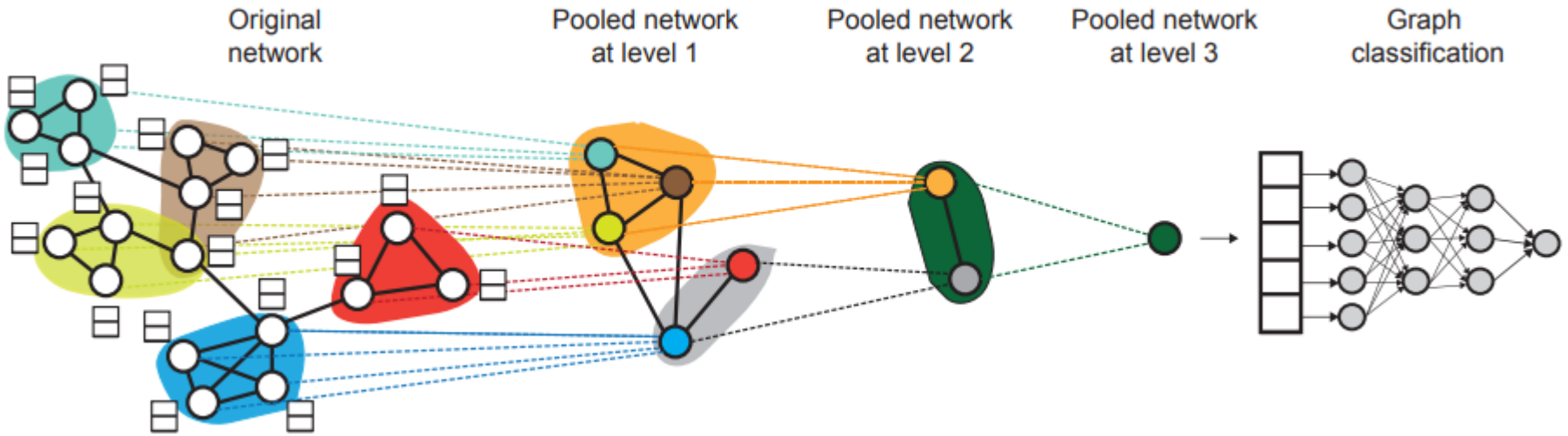
- POOLING -

Pooling

El pooling busca resumir el grafo. Se puede hacer a múltiples niveles de granularidad, a lo que se denomina, *pooling jerárquico*. Es útil para clustering (detección de comunidades en redes sociales, *motifs* funcionales en moléculas, clustering jerárquico en KG, clasificación de grafos).

DiffPool

En cada capa, obtenemos embeddings de los nodos, clusterizamos y volvemos a construir embeddings, pero ahora de los clusters, el proceso termina cuando tenemos un sólo embedding.

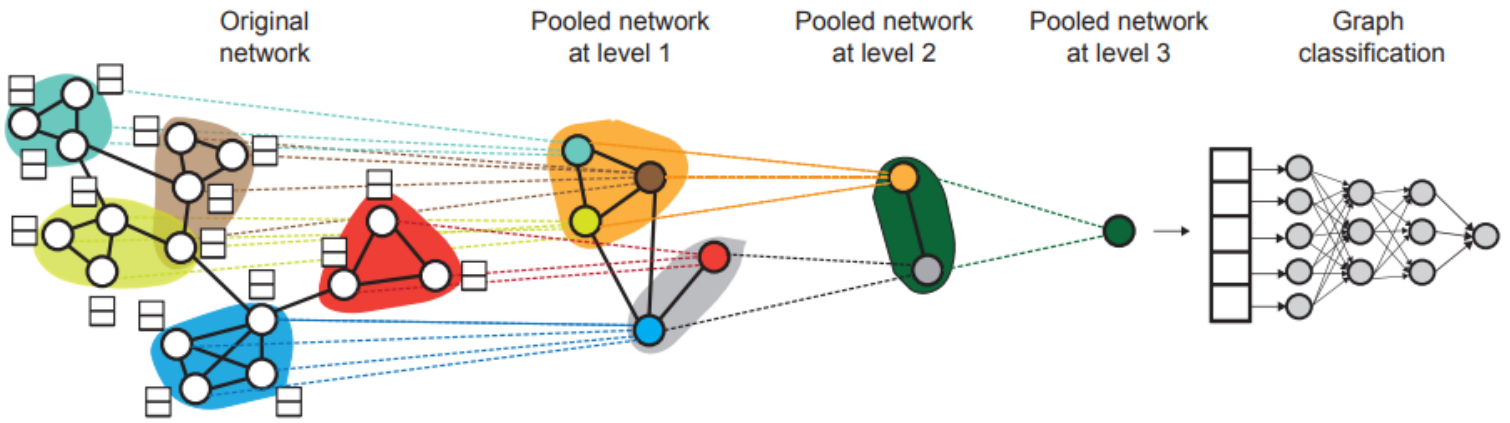


👉 Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., & Leskovec, J. (2018). Hierarchical graph representation learning with differentiable pooling. Advances in Neural Information Processing Systems, 31 (NeurIPS 2018). https://arxiv.org/abs/1806.08804

DiffPool

Se tiene una matriz de asignación *S*. Cada fila corresponde a un nodo (o cluster) en la capa *l* y cada columna a un nodo (o clúster) en la capa *l+1*. La matriz provee asignación suave de cada nodo de la capa *l* a un cluster de la capa *l+1*.

1) Para hacer *coarsening* del grafo, debe cumplirse: $S \in \mathbb{R}^{n \times k}, \quad k < n$



2) La matriz S se usa para recalcular los node embeddings:

$$X' = S^T X$$

3) También se usa para recalcular la matriz de adyacencia:

$$A' = S^T A S$$

DiffPool

Veamos un ejemplo:

$$S = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad X = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \\ -1 & 2 \end{bmatrix}.$$

$$\longrightarrow \quad X' = S^\top X = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ -1 & 3 \end{bmatrix}.$$

$$\longrightarrow \quad A' = S^\top (AS) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

La matriz de asignación es aprendible: $S^{(l)} = \text{softmax}\left(\text{GNN}_{l,\text{pool}}(A^{(l)}, X^{(l)})\right)$

MinCutPool

DiffPool tiene algunas limitaciones:

- La operación $A' = S^T A S$ tiene costo $O(n^2 k)$, es decir, no escala a grafos grandes.

- La asignación puede ser demasiado suave o degenerar (todos en un cluster).

- El *k* se elige *a priori*.

La idea central de **MinCutPool** es usar pooling jerárquico introduciendo regularización con un criterio de corte mínimo.

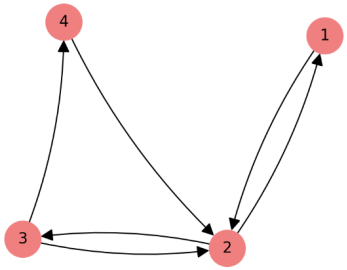
Al igual que DiffPool, trabaja haciendo *coarsening* sobre el grafo. Se usa la matriz de grado como regularizador:

$$\mathcal{L}_{\text{mincut}} = -\frac{\text{Tr}(S^T A S)}{\text{Tr}(S^T D S)}$$

donde Tr es la traza de la matriz: $\text{Tr}(M) = \sum_i M_{ii}$

Bianchi, F. M., Grattarola, D., & Alippi, C. (2020). Spectral clustering with graph neural networks for graph pooling. Advances in Neural Information Processing Systems, 33 (NeurIPS 2020), 8604–8614.https://arxiv.org/abs/1907.00481

MinCutPool



La pérdida de MinCutPool se explica así (intuición):

$\mathrm{Tr}(S^T AS)$: mide la suma de las conexiones internas a los clusters.

$\mathrm{Tr}(S^T DS)$: evita que los clusters grandes absorban clusters pequeños.

Veamos este ejemplo:

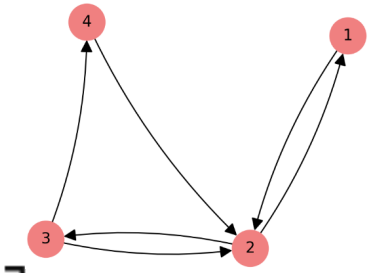
$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad D = \mathrm{diag}(1, 2, 2, 1), \quad S = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

$$\longrightarrow \quad A' = S^\top AS = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad \Rightarrow \quad \mathrm{Tr}(S^\top AS) = \mathrm{Tr}(A') = 2 + 2 = 4.$$

$$\longrightarrow \quad DS = \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} \longrightarrow \quad S^\top (DS) = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \quad \Rightarrow \quad \mathrm{Tr}(S^\top DS) = 3 + 3 = 6.$$

$$\Rightarrow \quad \mathcal{L}_{\text{mincut}} = -\frac{\mathrm{Tr}(S^\top AS)}{\mathrm{Tr}(S^\top DS)} = -\frac{4}{6} = -\tfrac{2}{3} \approx -0.6667.$$

MinCutPool



Veamos en el mismo ejemplo una asignación distinta:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad D = \mathrm{diag}(1, 2, 2, 1), \quad S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\longrightarrow \quad A' = S^\top A S = \begin{bmatrix} 0 & 3 \\ 3 & 0 \end{bmatrix} \Rightarrow \mathrm{Tr}(S^\top A S) = \mathrm{Tr}(A') = 0.$$

$$\longrightarrow \quad D' = S^\top D S = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \Rightarrow \mathrm{Tr}(S^\top D S) = \mathrm{Tr}(D') = 6.$$

Entonces, la función de pérdida está dada por:

$$\frac{\mathrm{Tr}(S^\top A S)}{\mathrm{Tr}(S^\top D S)} = \frac{0}{6} = 0 \quad \Longrightarrow \quad \mathcal{L}_{\text{mincut}} = -0.$$

Entonces $S = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$ es mejor que $S = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$

$\mathcal{L}_{\text{mincut}} = -\frac{2}{3} \approx -0.6667.$

$\mathcal{L}_{\text{mincut}} = -0.$

MinCutPool y clustering espectral

MinCutPool puede verse como una versión diferenciable de spectral clustering. En clustering espectral trabajamos sobre el Laplaciano del grafo:

$$L = I - D^{-1/2} A D^{-1/2}.$$

Si resolvemos el problema de valores propios de *L*:

$$Lu = \lambda u.$$

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

$$(L - \lambda_i I)u = 0$$

Obtenemos el espectro. Los valores propios siguientes son relevantes:

$$(\lambda_2, \ldots, \lambda_k)$$

Los valores propios contienen información sobre cómo particionar el grafo en clusters.

Ejemplo:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \longrightarrow L = I - D^{-1/2} A D^{-1/2} = \begin{bmatrix} 1 & -1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1 & -1/\sqrt{2} \\ 0 & -1/\sqrt{2} & 1 \end{bmatrix}$$

$$\longrightarrow U = \begin{bmatrix} 0.500 & 0.707 & 0.500 \\ 0.707 & 0.000 & -0.707 \\ 0.500 & -0.707 & 0.500 \end{bmatrix}$$

👉 Cada fila representa un nodo. Podemos clusterizar de forma vectorial, eso se llama clustering espectral. Este algoritmo minimiza aproximadamente el MinCut.