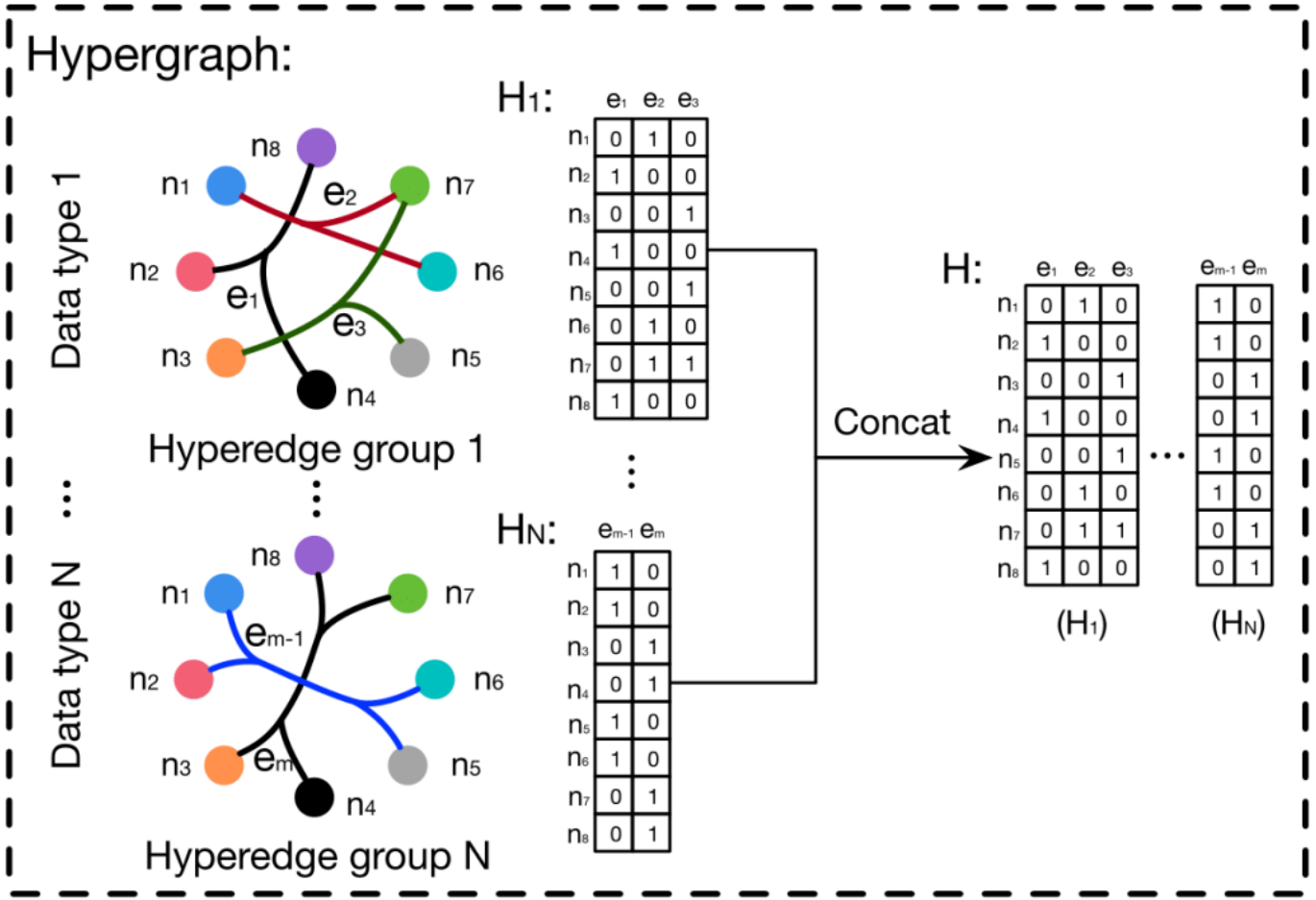
IIC-3641 Aprendizaje Automático basado en Grafos

https://github.com/marcelomendoza/IIC3641

- RECAPITULACIÓN -

Hypergraph Neural Networks

En la AF10 trabajamos con HGNNs, o cual nos permitiía modelar de forma nativa hiperenlaces.



Las HGNNs son flexibles y permiten modelas datos multimodales.

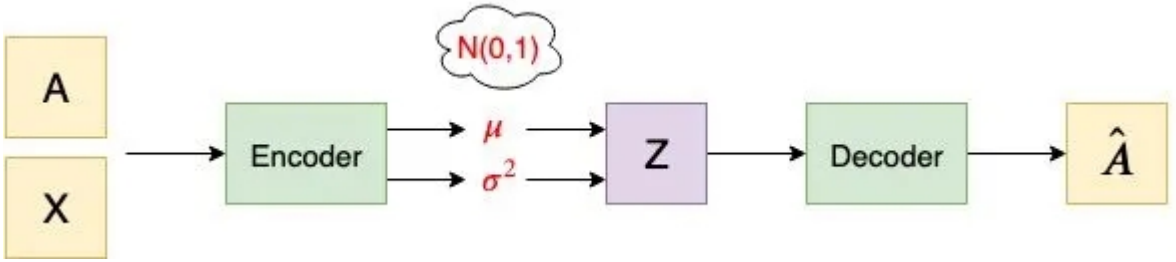
- VARIATIONAL GRAPH AUTOENCODER -

Variational Graph Autoencoder

Algo que se dificulta mucho hacer en el graph transformer o en el graphormer es construir un encoder a nivel de grafos. El graph transformer hace readouts sobre embeddings de nodos y enlaces para trabajar con propiedades de grafos, pero nativamente no tienen representaciones del grafo.

Si quisiéramos saber que aprende una máquina para codificar grafos, necesitariamos un autoencoder que opere con el grafo completo.

Esta es la idea del Variational Graph Autoencoder (VGAE).



Al igual que el VAE, el VGAE usa los datos para generar los parametros de un kernel Gaussiano a partir del cual se recupera el Z. El hecho de que el espacio latente esté condicionado a Gaussianas permite trabajar mejor en la fase de decodificación.

Vamos a revisar las ideas del VAE para entender que hace el VGAE.

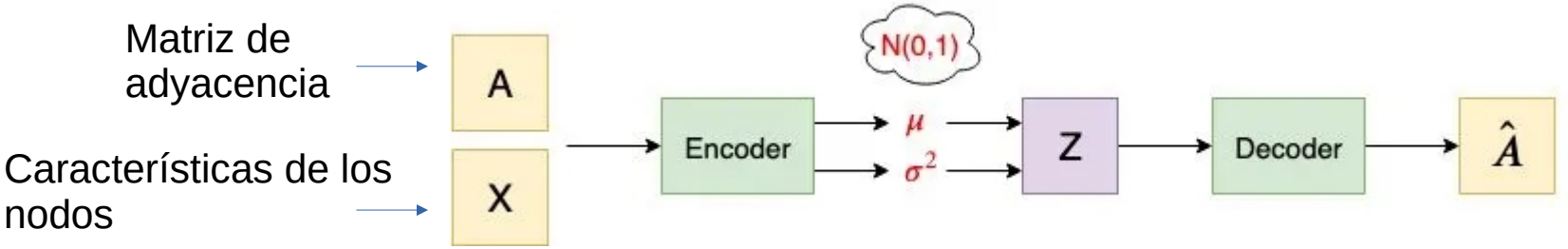
👉 Kipf, T., Welling, M. Variational Graph Auto-Encoders, NeurIPS 2016.
https://arxiv.org/abs/1611.07308

Variational Graph Autoencoder

El paper original propone la VGAE en base a capas GCN. Existen extensiones que reemplazan la GCN por capas lineales.

Vamos a seguir al paper original para entender que hace la VGAE.

Notación:



Encoder: $$\boldsymbol{\mu} = \mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A}) \quad \log \boldsymbol{\sigma} = \mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$$

A nivel de un ejemplo: $$q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i \mid \boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2))$$

El espacio latente: $$q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A})$$

Variational Graph Autoencoder

El VGAE usa una GCN para el encoder sobre la matriz de adyacencia normalizada:

$$\mathrm{GCN}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}\left(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0\right)\mathbf{W}_1$$

donde: $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$

estos parámetros son compartidos entre

$$\mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A})$$

$$\mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$$

Decoder (modelo generativo): la generación se basa en el producto interno entre variables latentes

A nivel de cada entrada de A: $p\left(A_{ij} = 1 \,|\, \mathbf{z}_i, \mathbf{z}_j\right) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j)$

Función logística

A nivel de A: $p\left(\mathbf{A} \,|\, \mathbf{Z}\right) = \prod_{i=1}^{N} \prod_{j=1}^{N} p\left(A_{ij} \,|\, \mathbf{z}_i, \mathbf{z}_j\right)$

Variational Graph Autoencoder

El VGAE se entrena sobre la siguiente función de pérdida:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}\left[\log p\left(\mathbf{A} \mid \mathbf{Z}\right)\right] - \mathrm{KL}\left[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \,||\, p(\mathbf{Z})\right]$$

La primera parte se enfoca en la reconstrucción de A desde el VGAE. La segunda corresponde a la divergencia Kullback-Leibler entre la distribución empírica en el espacio latente (q()), y un prior Gaussiano no informativo:

$$p(\mathbf{Z}) = \prod_i p(\mathbf{z_i}) = \prod_i \mathcal{N}(\mathbf{z}_i \mid 0, \mathbf{I})$$

Como va con signo -, la divergencia KL aminora la pérdida en la medida que aumenta. Maximizar la divergencia fuerza a que el **q** empírico tenga divergencia de un prior no informativo.

Variational Graph Autoencoder

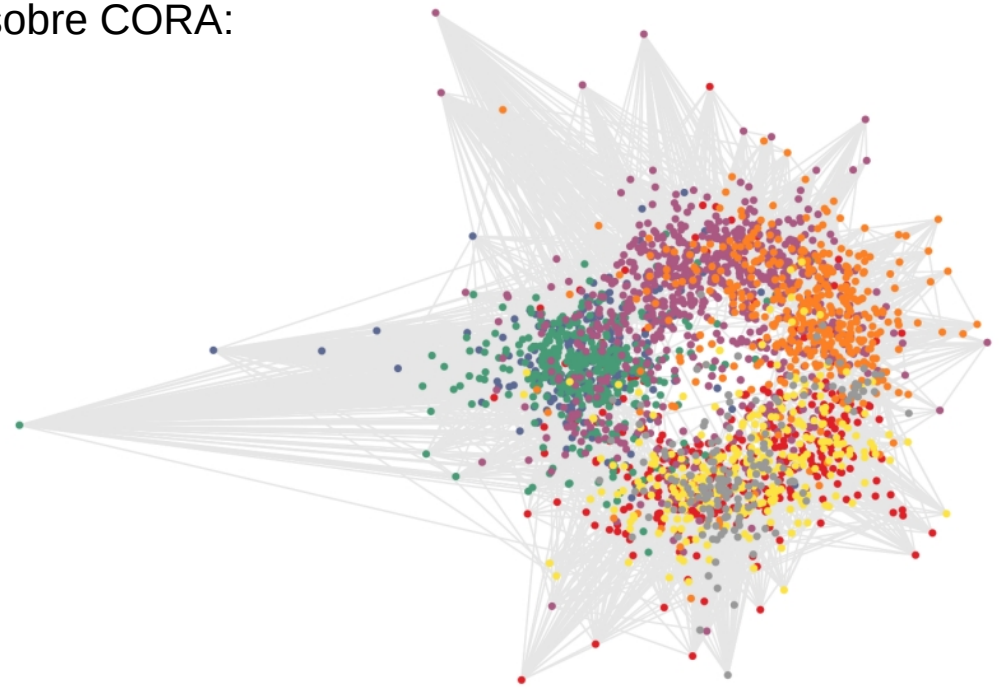
¿Qué debiera aprender el VGAE?

Miremos que es lo que reconstruye:

$$p\left(A_{ij}=1 \mid \mathbf{z}_i, \mathbf{z}_j\right)=\sigma\left(\mathbf{z}_i^{\top} \mathbf{z}_j\right)$$

Para que la función logística se active, el z_i y z_j deben estar cerca en el espacio latente.
Es decir, el espacio latente debe cumplir la hipótesis de clustering.

Veamos la VGAE sobre CORA:



La VGAE es un learner no supervisado de representaciones. Observemos que el atributo de clase no fue usado durante el entrenamiento. Podemos usar los embeddings para **link prediction** y **node classification**.

Variational Graph Autoencoder

Se puede trabajar con una versión no probabilística de la VGAE, calculando los embeddings Z y reconstruyendo la matriz de adyacencia:

$$\mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A})$$

$$\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^{\top}\right)$$

A esta variante se le llama GAE. El paper original trae experimentos sobre tres datasets en link prediction midiendo AUC y Average Precision (AP). Usar las features de los nodos produce mejoras (* indica sin features).

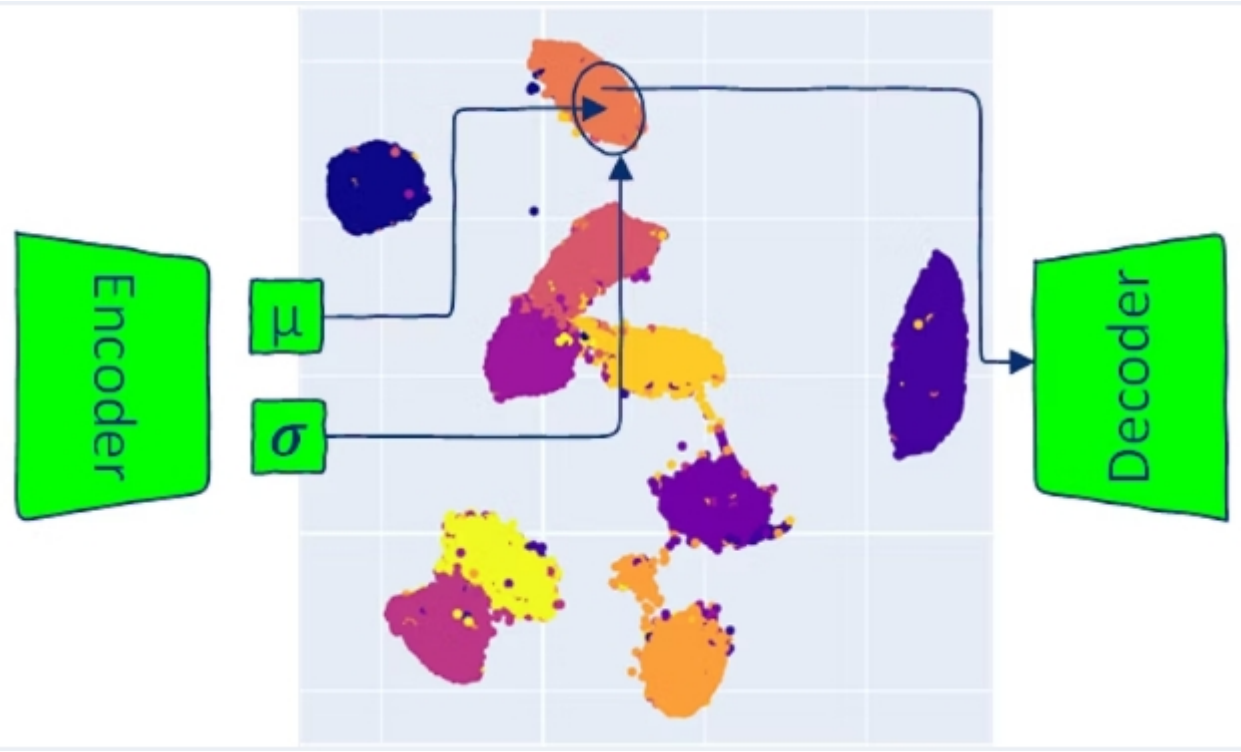
Method	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01	84.2 ± 0.02	87.8 ± 0.01
DW [6]	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01	84.4 ± 0.00	84.1 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01	78.7 ± 0.02	84.1 ± 0.02	82.2 ± 0.01	87.4 ± 0.00
VGAE*	84.0 ± 0.02	87.7 ± 0.01	78.9 ± 0.03	84.1 ± 0.02	82.7 ± 0.01	87.5 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	$\mathbf{96.4} \pm 0.00$	$\mathbf{96.5} \pm 0.00$
VGAE	$\mathbf{91.4} \pm 0.01$	$\mathbf{92.6} \pm 0.01$	$\mathbf{90.8} \pm 0.02$	$\mathbf{92.0} \pm 0.02$	94.4 ± 0.02	94.7 ± 0.02

- EXPLORANDO EL ESPACIO LATENTE -

El espacio latente del auto-encoder

Es de interés explorar que aprende el auto-encoder a partir de los datos, dado que al forzar la compresión debe identificar patrones no evidentes que le permitan reconstruir A luego de haber pasado por el cuello de botella.

El mecanismo básico de exploración del espacio latente es la interpolación entre pares de vectores. Para obtener estos vectores, muestreamos desde el espacio latente usando los parámetros Gaussianos:



El espacio latente del auto-encoder

Si tenemos dos vectores en el espacio latente, podemos explorar el espacio usando interpolación esférica (slerp):

1. Normalización de los vectores:

Los vectores de entrada $\mathbf{a}$ y $\mathbf{b}$ se normalizan:

$$\mathbf{a}_{\text{norm}} = \frac{\mathbf{a}}{\|\mathbf{a}\|}$$

$$\mathbf{b}_{\text{norm}} = \frac{\mathbf{b}}{\|\mathbf{b}\|}$$

donde $\|\mathbf{a}\|$ y $\|\mathbf{b}\|$ son las normas de los vectores $\mathbf{a}$ y $\mathbf{b}$, respectivamente.

2. Ángulo entre los vectores:

Se calcula el ángulo ω entre los vectores $\mathbf{a}_{\text{norm}}$ y $\mathbf{b}_{\text{norm}}$ utilizando el coseno del ángulo:

$$\omega = \arccos\left(\mathbf{a}_{\text{norm}} \cdot \mathbf{b}_{\text{norm}}\right) + \epsilon$$

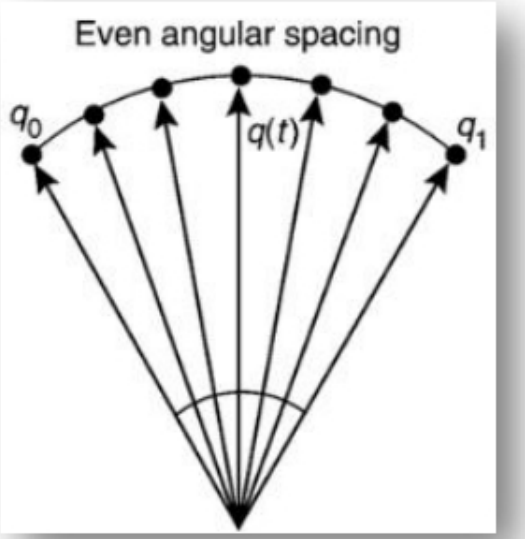
Aquí $\cdot$ es el producto punto entre $\mathbf{a}_{\text{norm}}$ y $\mathbf{b}_{\text{norm}}$, y ϵ es un pequeño valor para evitar divisiones por cero.

3. Interpolación en la esfera:

SLERP usa el ángulo ω y las funciones seno para interpolar entre los dos vectores. La ecuación de interpolación es:

$$\text{SLERP}(\mathbf{a}, \mathbf{b}, n) = \frac{\sin((1-n)\omega)}{\sin(\omega)}\mathbf{a} + \frac{\sin(n\omega)}{\sin(\omega)}\mathbf{b}$$

donde $n \in [0, 1]$ es el parámetro de interpolación. Si $n = 0$, el resultado es $\mathbf{a}$; si $n = 1$, el resultado es $\mathbf{b}$; y para otros valores de n, se obtiene un punto intermedio en la esfera.



- FIN -

- FIN -

… *de las clases, faltan los proyectos y la AF11* ...