FACULTAD DE
INGENIERÍA

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Aprendizaje Automático para Datos en Grafos

Graph Neural Networks: Definición

Federico 'Larroca' La Rocca

Muy basado en transparencias de **Fernando Gama**

flarroca@fing.edu.uy

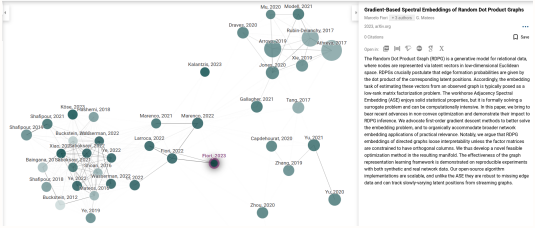
http://iie.fing.edu.uy/personal/flarroca

Aprendizaje Automático en Grafos

- Grafos son modelos para datos estructurados y útiles para aprender en varios problemas prácticos
- **Ejemplo 1**: ¿A qué área pertenece una nueva submission en arXiv?

Subjects: **Machine Learning (cs.LG)**; Optimization and Control (math.OC)
Cite as: arXiv:2307.13818 **[cs.LG]**
(or arXiv:2307.13818v1 **[cs.LG]** for this version)
https://doi.org/10.48550/arXiv.2307.13818

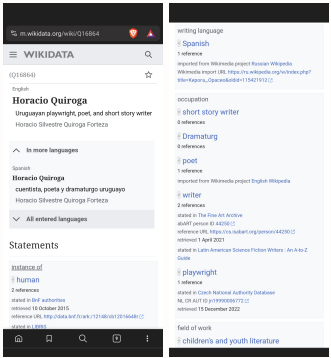
  - El sistema hace sugerencias, aliviando la tarea de los moderadores. ¿Basado en qué?
  - Idea: Usar los papers (ya categorizados) citados por la nueva submission!

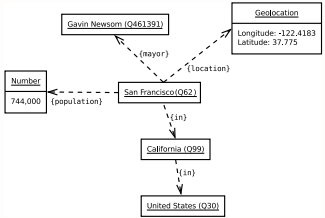


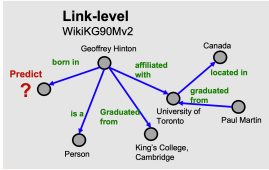
  - Complementar cada nodo con algunos datos extra: año, representación vectorial de título y abstract, etc.

Aprendizaje Automático en Grafos

- Grafos son modelos para datos estructurados y útiles para aprender en varios problemas prácticos
- **Ejemplo 2**: Análisis de bases de conocimiento. O en este caso grafo de conocimiento





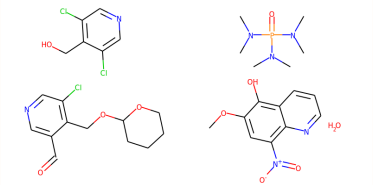


- ¿Es posible completar datos faltantes? Es decir, ¿predecir enlaces?

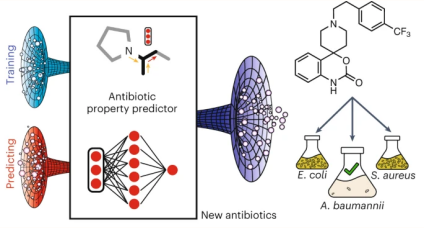
Open Graph Benchmark, https://ogb.stanford.edu/docs/lsc/wikikg90mv2/

Aprendizaje Automático en Grafos

- Grafos son modelos para datos estructurados y útiles para aprender en varios problemas prácticos
- **Ejemplo 3**: Descubrimiento de medicamentos



- ¿Tiene alguna de estas moléculas propiedades antibióticas? Predicción de propiedades del grafo



Liu *et al.*, "Deep learning-guided discovery of an antibiotic targeting Acinetobacter baumannii", Nature Chemical Biology 2023

Empirical Risk Minimization

- En este curso, machine learning $\equiv$ empirical risk minimization (ERM).
- Los ingredientes de ERM son:
  - $\Rightarrow$ Un conjunto de entrenamiento $\mathcal{T}$ conteniendo observaciones $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}$ (no-supervisados: sólo $\mathbf{x}$)
  - $\Rightarrow$ Una clase de funciones $\boldsymbol{\Phi}(\mathbf{x}) = \hat{\mathbf{y}}$
  - $\Rightarrow$ Una función de pérdida $\mathsf{J}(\mathbf{y}, \hat{\mathbf{y}})$ que evalúa la bondad del ajuste de $\hat{\mathbf{y}}$ (respecto a $\mathbf{y}$)
- Aprender significa hallar la función $\boldsymbol{\Phi}^\star \in \boldsymbol{\Phi}$ que minimiza la pérdida $\mathsf{J}\Big(\mathbf{y}, \boldsymbol{\Phi}(\mathbf{x})\Big)$ promediada en el conjunto de entrenamiento

$$\boldsymbol{\Phi}^\star = \underset{\boldsymbol{\Phi} \in \boldsymbol{\Phi}}{\arg\min} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{T}} \mathsf{J}\Big(\mathbf{y}, \boldsymbol{\Phi}(\mathbf{x})\Big)$$

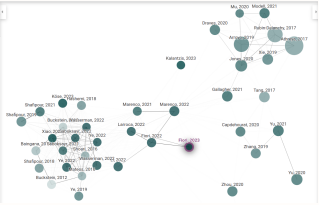
- Usamos $\boldsymbol{\Phi}^\star(\mathbf{x})$ para estimar las salidas $\hat{\mathbf{y}} = \boldsymbol{\Phi}^\star(\mathbf{x})$ en entradas $\mathbf{x}$ no observadas en el conjunto de entrenamiento $\mathcal{T}$

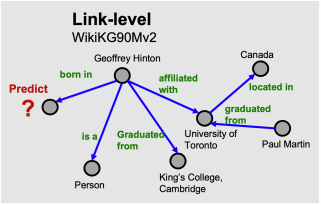
Empirical Risk Minimization

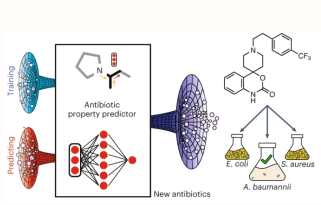
- En ERM, la clase de funciones $\mathbf{\Phi}$ es el grado de libertad disponible a la hora de diseñar el sistema

$$\mathbf{\Phi}^\star = \arg\min_{\mathbf{\Phi} \in \mathbf{\Phi}} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \mathsf{J}\Big(\mathbf{y}, \mathbf{\Phi}(\mathbf{x})\Big)$$

- Diseñar un algoritmo de Machine Learning $\equiv$ hallar la clase de funciones $\mathbf{\Phi}$ adecuada
- Pero antes, ¿quién es nuestro dato en este caso?
- Tres ingredientes:
  - El grafo $G = (V, E)$
  - Atributos (señales) en los nodos
  - "Opcional": Atributos en las aristas

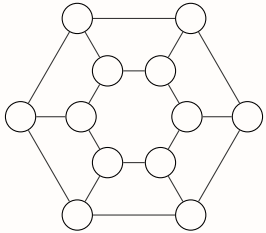






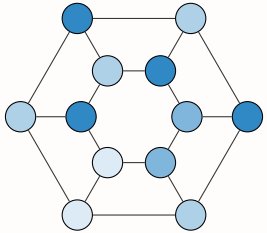
Representando Datos en Grafos

- Ingredientes:
  - $G = (V, E)$



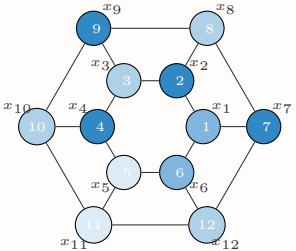
Representando Datos en Grafos

- Ingredientes:
  - $G = (V, E)$
  - Valor asociado a cada nodo (señal)



Representando Datos en Grafos

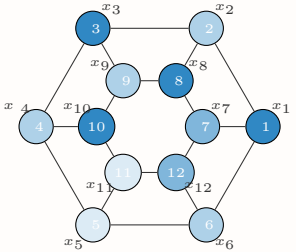
- Ingredientes:
  - $G = (V, E)$
  - Valor asociado a cada nodo (señal)
  - Ordenemos los nodos:
    - matriz de adyacencia $\mathbf{A}$
    - un vector de señal $\mathbf{x} = [x_1, \ldots, x_{12}]^\top$



- Ejemplo: predecir propiedad del grafo. Podemos usar una función $\Phi : \mathbb{R}^{12\times 12} \times \mathbb{R}^{12} \to \mathbb{R}$ $(\Phi(\mathbf{A}, \mathbf{x}))$

Representando Datos en Grafos

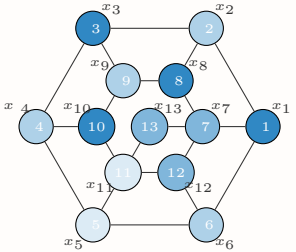
- Ingredientes:
  - $G = (V, E)$
  - Valor asociado a cada nodo (señal)
  - Ordenemos los nodos:
    - matriz de adyacencia $\mathbf{A}$
    - un vector de señal $\mathbf{x} = [x_1, \ldots, x_{12}]^\top$



- Ejemplo: predecir propiedad del grafo. Podemos usar una función $\Phi : \mathbb{R}^{12\times 12} \times \mathbb{R}^{12} \to \mathbb{R}$ $(\Phi(\mathbf{A}, \mathbf{x}))$
  - ¿Podemos?

Representando Datos en Grafos

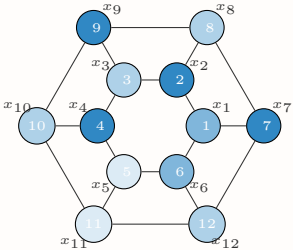
- Ingredientes:
  - $G = (V, E)$
  - Valor asociado a cada nodo (señal)
  - Ordenemos los nodos:
    - matriz de adyacencia $\mathbf{A}$
    - un vector de señal $\mathbf{x} = [x_1, \ldots, x_{12}]^\top$

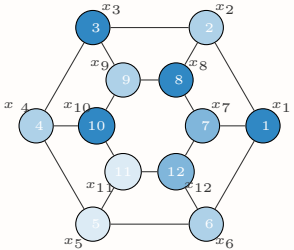


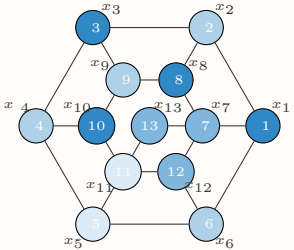
- Ejemplo: predecir propiedad del grafo. Podemos usar una función $\Phi : \mathbb{R}^{12\times 12} \times \mathbb{R}^{12} \to \mathbb{R}$ $(\Phi(\mathbf{A}, \mathbf{x}))$
  - ¿Podemos?
  - Claramente no deberíamos...

Familia de funciones

- Wishlist sobre Φ:
  - No dependa del orden de los nodos
  - Aplicable a grafos de distinto tamaño



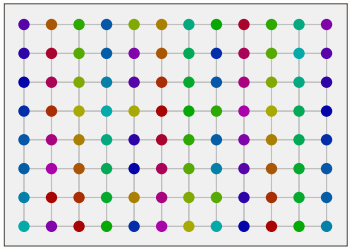




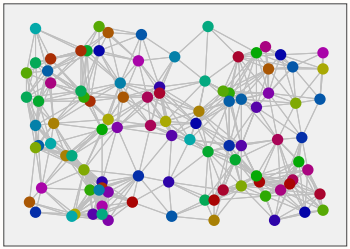
Aprendizaje Automático en Grafos

- Aprendizaje automático exitoso aprovecha estructura $\Rightarrow$ Convolutional neural networks (CNNs)

Somos buenos aprendiendo en esto



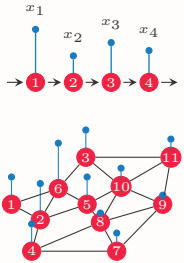
Desafío: queremos aprender sobre esto



- Escala, aprovecha la estructura de los datos, y tiene una implementación eficiente (distribuida)
- Graph Convolutions $\Rightarrow$ Graph Signal Processing $\Rightarrow$ Graph Filtering

Datos Estructurados como Grafos

- Señales en tiempo discreto $\Rightarrow$ Cercanía contiene información
  - $\Rightarrow$ Explicitemos la estructura de los datos $\Rightarrow$ Relación temporal
  - $\Rightarrow$ Dos elementos de Proc. de señales: estructura y valores de la señal
- Grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ $\Rightarrow$ $\mathcal{V}$: nodos, $\mathcal{E}$: aristas
- Señal en el grafo $\Rightarrow$ asociar un valor a cada nodo $\mathbf{x} : \mathcal{V} \to \mathbb{R}$
- Representación matricial $\Rightarrow$ Matriz de adyacencia (traspuesta) $\mathbf{A}$, Laplaciano $\mathbf{L}$
  - $\Rightarrow$ Fija el orden de los nodos $\Rightarrow$ Permutaciones
  - $\Rightarrow$ Matriz $\mathbf{S}$ genérica (Graph Shift Operator)



Sandryhaila, Moura, "Discrete Signal Processing on Graphs", IEEE TSP, 2013

Shuman, Narang, Frossard, Ortega, Vandergheynst, "The Emerging Field of Signal Processing on Graphs", IEEE SPM, 2013

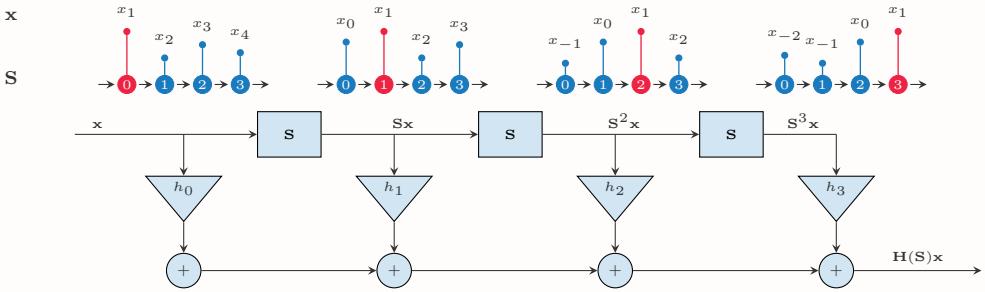
Graph Convolutions

- Graph convolution $\Rightarrow$ Combinación lineal de versiones desplazadas de la señal

$$\mathbf{x} *_{\mathbf{S}} \mathbf{h} = \sum_{k=0}^{K-1} h_k \, \mathbf{S}^k \mathbf{x}$$

- Noción de desplazamiento $\mathbf{S}$ $\Rightarrow$ $\mathbf{Sx}$ desplaza la señal $\mathbf{x}$

$$\begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \\ \cdots & 0 & 0 & 0 & \cdots \\ \cdots & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 1 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & \cdots \\ & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ x_1 \\ x_2 \\ x_3 \\ \vdots \end{bmatrix}$$



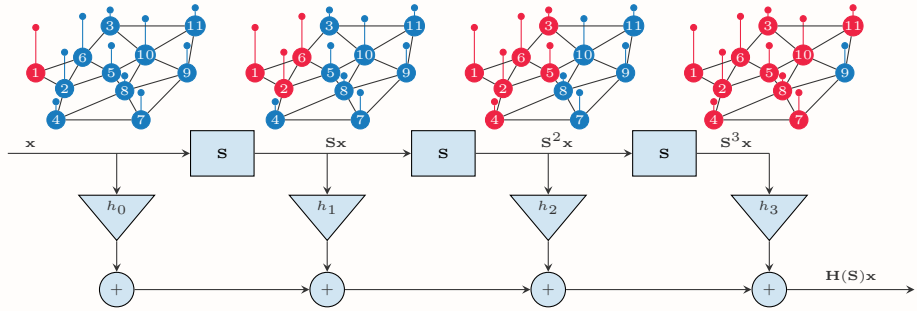
Sandryhaila, Moura, "Discrete Signal Processing on Graphs", IEEE TSP, 2013

Graph Convolutions

- Graph convolution $\Rightarrow$ Combinación lineal de versiones desplazadas de la señal

$$\mathbf{x} *_{\mathbf{S}} \mathbf{h} = \sum_{k=0}^{K-1} h_k \, \mathbf{S}^k \mathbf{x} = \mathbf{H}(\mathbf{S})\mathbf{x}$$

- Noción de desplazamiento $\mathbf{S}$ $\Rightarrow$ Descripción matricial del grafo (e.g. adyacencia)
- Combinación lineal de las señales vecinas $\Rightarrow$ Operación local

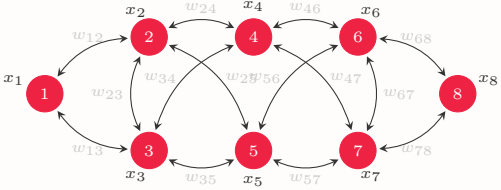


Sandryhaila, Moura, "Discrete Signal Processing on Graphs", IEEE TSP, 2013

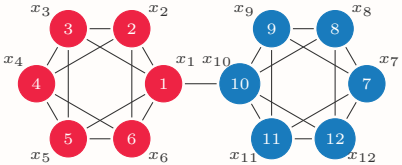
De Información Local a Global

- La convolución en grafos agrega información desde lo local a lo global
  $\Rightarrow$ Igual que la clásica convolución en el tiempo o espacio

Graph Filter en un grafo



Mismo Graph Filter en otro grafo

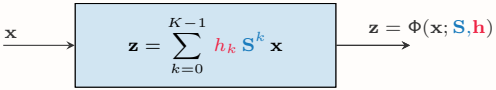


- Usando coeficientes $\mathbf{h} = \{h_k\}_{k=0}^{\infty}$ $\Rightarrow \mathbf{z} = h_0\mathbf{S}^0\mathbf{x} + h_1\mathbf{S}^1\mathbf{x} + h_2\mathbf{S}^2\mathbf{x} + h_3\mathbf{S}^3\mathbf{x} + \ldots = \sum_{k=0}^{\infty} h_k\mathbf{S}^k\mathbf{x}$
- La salida de un grafo depende de los coeficientes $\mathbf{h}$ y el GSO $\mathbf{S}$

Aprendiendo usando Filtros sobre Grafos

- Asumamos que las entradas $\mathbf{x}$ son señales en un mismo grafo con GSO $\mathbf{S}$
- Elegimos como clase de funciones los filtros en grafos de orden K en $\mathbf{S}$

  $\Rightarrow \Phi(\mathbf{x}) = \Phi(\mathbf{x}; \mathbf{S}, \mathbf{h}) = \sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}$

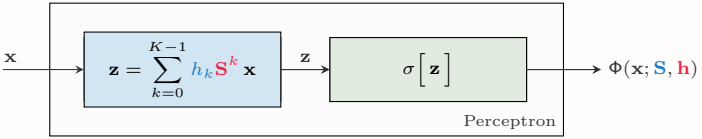


- Aprender: la solución al ERM restringido a la clase de filtros $\Rightarrow \mathbf{h}^\star = \arg\min_{\mathbf{h}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\Phi(\mathbf{x}; \mathbf{S}, \mathbf{h}), \mathbf{y}\Big)$

  $\Rightarrow$ La optimización es sobre los coeficientes del filtro $\mathbf{h}$ con el GSO $\mathbf{S}$ dado

Aprendiendo usando un Graph Perceptron

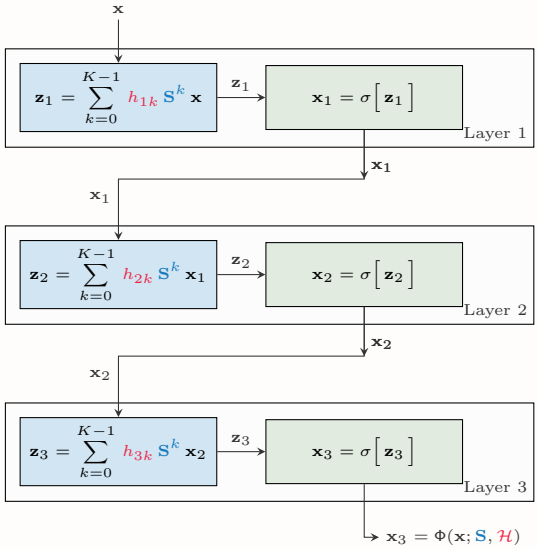
- Los filtros tienen expresividad limitada: sólo pueden aprender funciones lineales
- Otra elección para $\mathbf{\Phi}$ es el conjunto de graph perceptrons $\Rightarrow \mathbf{\Phi}(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathbf{h}) = \sigma\left[\sum_{k=0}^{K-1} h_k \mathbf{S}^k \mathbf{x}\right]$



- El óptimo ahora está restringido a la clase de los graph perceptrons
  $\Rightarrow \mathbf{h}^\star = \arg\min_{\mathbf{h}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathbf{h}), \mathbf{y}\Big)$
  $\Rightarrow$ Los perceptrones permiten aprender funciones no-lineales $\Rightarrow$ Mayor expresividad.

Graph Neural Networks (GNNs)

- Se aumenta la expresividad aún más con una GNN
- Apilo algunos graph perceptrons (3 en la fig.)
  - ⇒ Alimento la Capa 1 con $\mathbf{x}$
  - ⇒ Conecto la salida de la Capa 1 a la entrada de la 2
  - ⇒ y la salida de la Capa 2 a la entrada de la 3
- Salida de la última capa: salida de la GNN
  ⇒ $\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H})$
  - ⇒ El parámetro es el tensor $\mathcal{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3]$



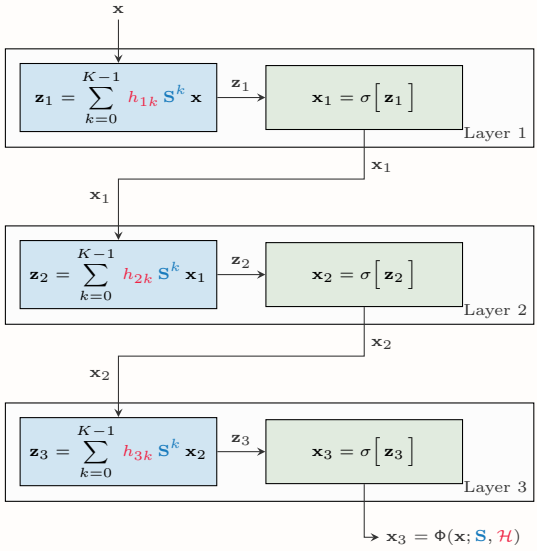
Aprendiendo con una Graph Neural Network

- Aprender el tensor $\mathcal{H}^\star = (\mathbf{h}_1^\star, \mathbf{h}_2^\star, \mathbf{h}_3^\star)$ óptimo:

$$\mathcal{H}^\star = \arg\min_{\mathcal{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathrm{J}\Big(\mathbf{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H}),\mathbf{y}\Big)$$

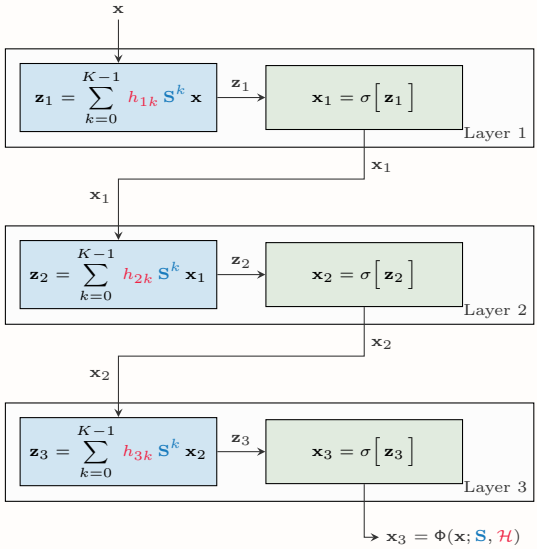
- La optimización es sobre el tensor únicamente. El GSO $\mathbf{S}$ está dado

  $\Rightarrow$ Información a priori que le damos a la GNN



Graph Neural Networks y Graph Filters

- GNNs son una variación sencilla de un graph filter
- LA diferencia es la no-linealidad punto-a-punto
  - ⇒ Las entradas se procesan individualmente
  - ⇒ No se combinan
- Pero las GNNs funcionan (mucho) mejor
  - ⇒ Merece una explicación
  - ⇒ La intentaremos mediante análisis de estabilidad



Transferencia de GNNs a otros Grafos

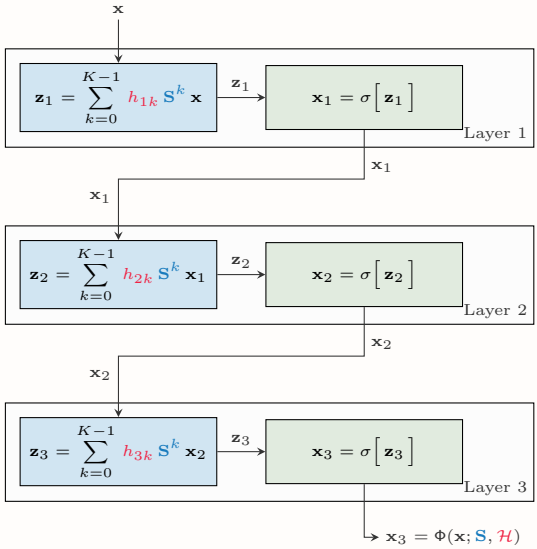
- Salida de la GNN depende del GSO $\mathbf{S}$.
- Podemos interpretar $\mathbf{S}$ como un parámetro

  $\Rightarrow$ Codifica información a priori. Como hasta ahora

- O podemos reinterpretar $\mathbf{S}$ como una entrada

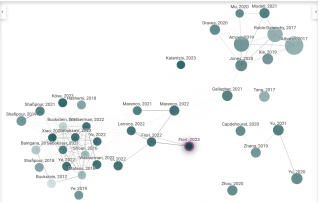
  $\Rightarrow$ Permite transferencia a otros grafos

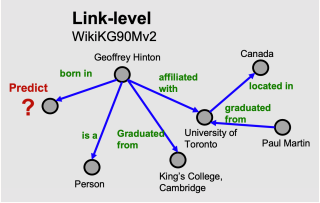
  $\Rightarrow$ Una GNN entrenada es simplemente el tensor $\mathcal{H}^{\star}$

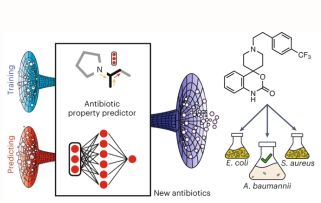


El camino no explorado: Fully Connected Neural Networks

- Elegimos graph filters y GNNs porque nos interesaban las señales en el grafo
- Nos pareció una buena idea porque son generalizaciones de filtros y CNNs
- Hay otras alternativas que vale la pena explorar $\Rightarrow$ Fully connected neural networks

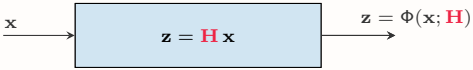






Aprendiendo con una Función Lineal

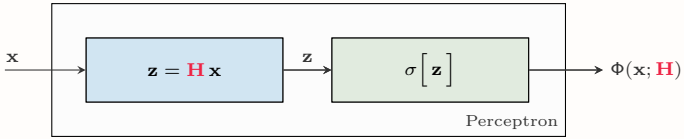
- En vez de filtros, elijamos funciones lineales $\Rightarrow \mathbf{\Phi}(\mathbf{x}) = \mathbf{\Phi}(\mathbf{x}; \mathbf{H}) = \mathbf{H}\,\mathbf{x}$



- Óptimo del ERM es la solución restringida a la clase lineal $\Rightarrow \mathbf{H}^\star = \arg\min_{\mathbf{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\mathbf{\Phi}(\mathbf{x}; \mathbf{H}), \mathbf{y}\Big)$

Aprendiendo con un Perceptrón

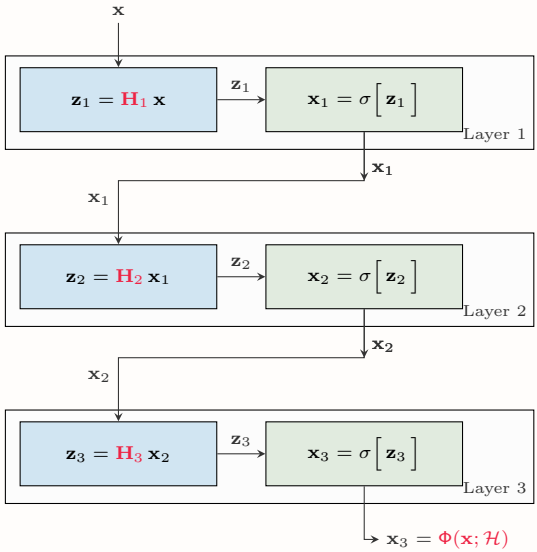
- Podemos aumentar la expresividad agregando una no-linealidad: perceptrón
  $\Rightarrow \Phi(\mathbf{x}) = \Phi(\mathbf{x}; \mathbf{H}) = \sigma\Big[\mathbf{H}\mathbf{x}\Big]$



- Óptimo del ERM es la solución restringida a la clase de perceptrones
  $\Rightarrow \mathbf{H}^\star = \arg\min_{\mathbf{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\Phi(\mathbf{x}; \mathbf{H}), \mathbf{y}\Big)$

Fully Connected Neural Network (NN)

- Aumentemos aún más la expresividad con una NN
- Apilo algunos perceptrones (3 en la fig.)
  - ⇒ Alimento la Capa 1 con $\mathbf{x}$
  - ⇒ Conecto la salida de la Capa a la entrada de la 2
  - ⇒ y la salida de la Capa 2 a la entrada de la 3
- Salida de la última capa: salida de la NN ⇒ $\Phi(\mathbf{x}; \mathcal{H})$
  - ⇒ El parámetro es el tensor $\mathcal{H} = [\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3]$



¿Cuál es mejor? ¿GNN o NN?

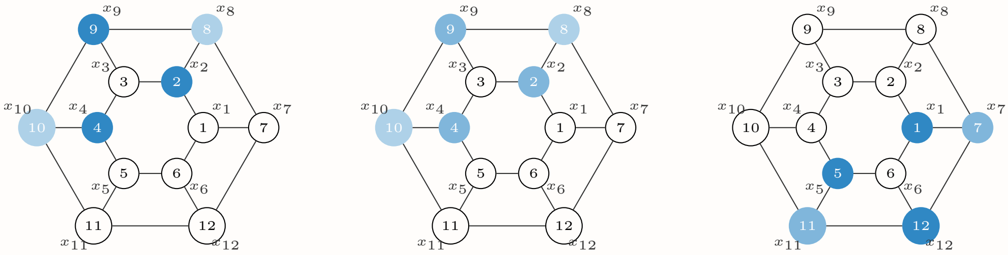
- Como la GNN es un caso particular de una fully connected NN, ésta obtiene menor costo

$$\min_{\mathcal{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\boldsymbol{\Phi}(\mathbf{x};\mathcal{H}),\mathbf{y}\Big) \leq \min_{\mathcal{H}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\boldsymbol{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H}),\mathbf{y}\Big)$$

- La NN anda mejor, pero sólo en el conjunto de entrenamiento
- La GNN funciona mejor porque generaliza mejor a señales no vistas
  - $\Rightarrow$ Aprovecha simetrías internas de la señal del grafo codificadas en el GSO
  - $\Rightarrow$ Posible interpretación: aumenta el dataset de entrenamiento

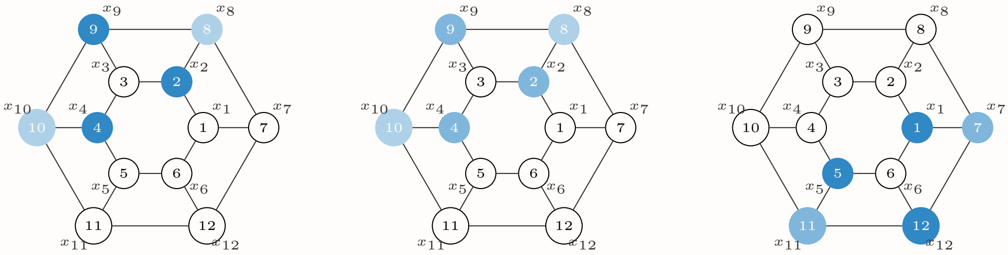
Generalización con una Neural Network

- Supongamos que el grafo representa un sistema de recomendación:
  - Nodos son usuarios, aristas gustos similares y señal un rating
  - Queremos rellenar los valores faltantes
- Observamos los rating de la izquierda (entrenamiento). Pero no observamos datos como los otros dos
- Del ejemplo de la izquierda, la NN aprende a predecir el ejemplo del medio pero NO el de la derecha



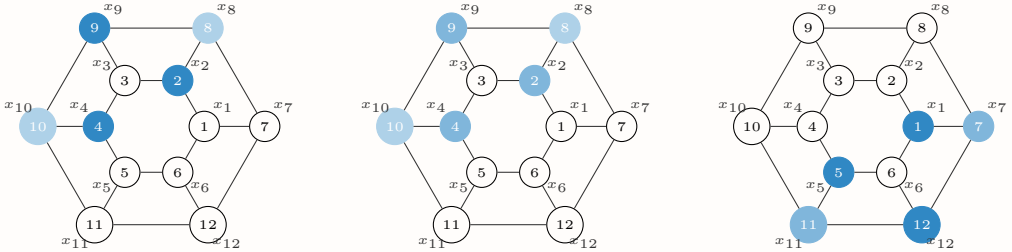
Generalización con una Graph Neural Network

- Supongamos que el grafo representa un sistema de recomendación:
  - Nodos son usuarios, aristas gustos similares y señal un rating
  - Queremos rellenar los valores faltantes
- Observamos los rating de la izquierda (entrenamiento). Pero no observamos datos como los otros dos
- La GNN también aprende a predecir el ejemplo del medio y además el ejemplo de la derecha



Equivarianza a permutaciones en Graph Neural Networks

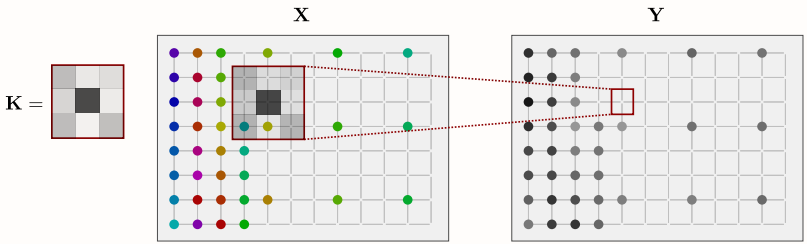
- Lo anterior lo vamos a formalizar como la equivarianza a permutaciones de las GNNs
- Es análogo a la invarianza a traslaciones de las convolutional neural networks. El cual es un caso particular



CNNs como GNNs

- Recordemos que una CNN (básicamente) toma una matriz y genera otra usando la convolución

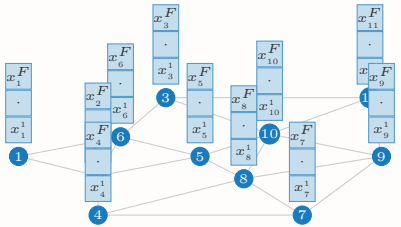
$$y_{i,j} = \sum_{h=-W}^{W} \sum_{v=-W}^{W} K_{h,v} x_{i+h,j+v}$$



- Ejercicio ¿Cómo podemos construir esto usando una GNN?

Múltiples Features

- Graph signal $\Rightarrow$ Un escalar asociado a cada nodo $\Rightarrow$ $\mathbf{x}$: nodos $\to \mathbb{R}$
- Extendamos el poder descriptivo $\Rightarrow$ Asignamos un vector a cada nodo
  $\Rightarrow$ $\mathbf{X}$: nodos $\to \mathbb{R}^2$ $\Rightarrow$ $\mathbf{X}$: nodos $\to \mathbb{R}^3$ $\Rightarrow$ $\mathbf{X}$: nodos $\to \mathbb{R}^F$
- Multiple feature graph signal $\Rightarrow$ Matriz $\mathbf{X}$ de tamaño N (nodos) $\times$ F (features)
  $\Rightarrow$ Fila i contiene los features del nodo i $\Rightarrow$ Información local del nodo i
  $\Rightarrow$ Columna f representa el feature f de todos los nodos $\Rightarrow$ $\mathbf{x}^f$ es una graph signal



$$\mathbf{X} = \begin{bmatrix} x_1^1 & x_1^2 & x_1^3 & \cdots & x_1^F \\ x_2^1 & x_2^2 & x_2^3 & \cdots & x_2^F \\ x_3^1 & x_3^2 & x_3^3 & \cdots & x_3^F \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{10}^1 & x_{10}^2 & x_{10}^3 & \cdots & x_{10}^F \\ x_{11}^1 & x_{11}^2 & x_{11}^3 & \cdots & x_{11}^F \end{bmatrix}$$

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2018

Extendiendo la Convolución

- Convolución $\Rightarrow$ Operación lineal, información local, implementación distribuida

$$\mathbf{Y} = \sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X} \mathbf{H}_k$$

- Multiplicación por $\mathbf{S}$ $\Rightarrow$ Desplaza los graph signals de los features $\mathbf{S}\mathbf{x}^f$
- Multiplicación por $\mathbf{H}$ $\Rightarrow$ Combinación lineal de los features en cada nodo (no hay intercambios)

- La convolución es equivalente a la aplicación de un banco de filtros
- Por cada feature $\mathbf{x}^f$ el filtro (f, g) para obtener $\mathbf{x}^{fg}$ $\Rightarrow$ Lineal, local, distribuido

$$\mathbf{x}^{fg} = \sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{x}^f h_k^{fg}$$

$\Rightarrow$ Hay G filtros aplicados a cada entrada $\mathbf{x}^f$ $\Rightarrow$ Hay un total de $F \times G$ filtros (tamaño de $\mathbf{H}_k$)

- Agregamos la salida de cada g-avo filtro en todas las F features $\Rightarrow \mathbf{y}^g = \sum_{f=1}^{F} \mathbf{x}^{fg}$ $\Rightarrow \mathbf{Y} = [\mathbf{y}^1, \ldots, \mathbf{y}^G]$

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2018

Graph Neural Networks

- Single feature graph neural networks $\Rightarrow \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H}) = \mathbf{x}_L$

$$\mathbf{x}_\ell = \sigma\Big[\sum_{k=0}^{K_\ell - 1} \mathbf{S}^k \mathbf{x}_{\ell-1} h_{\ell k} \Big]$$

  $\mathbf{x}_\ell$ graph signal de la capa ℓ $\Rightarrow$ Tamaño $N \times 1$

  $h_{\ell k}$ k-ésimo tap de la capa ℓ $\Rightarrow$ Tamaño 1×1 $\Rightarrow$ Aprender $\sum_\ell K_\ell$ filter taps $\mathcal{H} = \{h_{\ell k}\}$

  $\mathbf{S}$ shift operator (dado por el problema) $\Rightarrow$ Tamaño $N \times N$

- Hiperparámetros (a elección) $\Rightarrow$ Afecta el número de parámetros aprendibles

  $\Rightarrow$ L: número de capas

  $\Rightarrow$ K_ℓ: número de taps en cada capa

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2018

Graph Neural Networks

- Multi feature graph neural networks $\Rightarrow \Phi(\mathbf{x}; \mathbf{S}, \mathcal{H}) = \mathbf{X}_L$

$$\mathbf{X}_\ell = \sigma\Big[\sum_{k=0}^{K_\ell - 1} \mathbf{S}^k \mathbf{X}_{\ell-1} \mathbf{H}_{\ell k} \Big]$$

  $\mathbf{X}_\ell$ graph signal de la capa ℓ $\Rightarrow$ Tamaño $N \times F_\ell$

  $\mathbf{H}_{\ell k}$ k-ésimo tap de la capa ℓ $\Rightarrow$ Tamaño $F_{\ell-1} \times F_\ell$ $\Rightarrow$ Aprender $\sum_\ell K_\ell F_\ell F_{\ell-1}$ taps $\mathcal{H} = \{\mathbf{H}_{\ell k}\}$

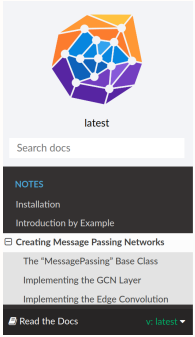
  $\mathbf{S}$ shift operator (dado por los datos) $\Rightarrow$ Tamaño $N \times N$

- Hiperparámetros (a elección) $\Rightarrow$ Afecta el número de parámetros aprendibles
  - $\Rightarrow$ L: número de capas
  - $\Rightarrow$ K_ℓ: número de taps en cada capa
  - $\Rightarrow$ F_ℓ: número de features en cada capa

Gama, Marques, Leus, Ribeiro, "Convolutional Neural Network Architectures for Signals Supported on Graphs", IEEE TSP 2018

Message Passing

- Te instalaste PyG y no habla en ningún lado de convoluciones ...
  - Es más, habla mucho de Message Passing



» Creating Message Passing Networks

CREATING MESSAGE PASSING NETWORKS

Generalizing the convolution operator to irregular domains is typically expressed as a *neighborhood aggregation or message passing* scheme. With $\mathbf{x}_i^{(k-1)} \in \mathbb{R}^F$ denoting node features of node i in layer $(k-1)$ and $\mathbf{e}_{j,i} \in \mathbb{R}^D$ denoting (optional) edge features from node j to node i, message passing graph neural networks can be described as

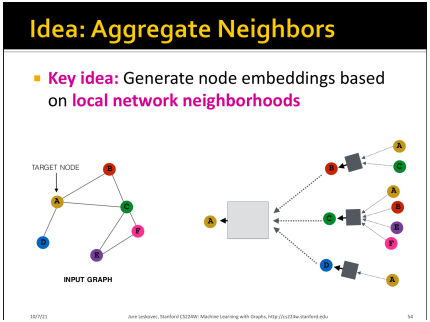
$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \, \phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i} \right) \right),$$

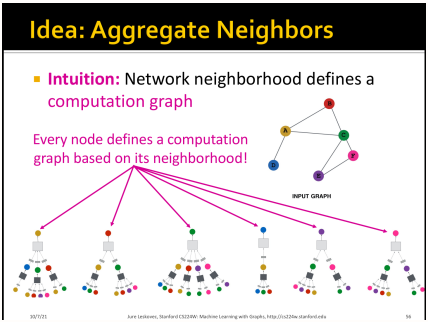
where $\square$ denotes a differentiable, permutation invariant function, *e.g.*, sum, mean or max, and γ and ϕ denote differentiable functions such as MLPs (Multi Layer Perceptrons).

- The "MessagePassing" Base Class
- Implementing the GCN Layer
- Implementing the Edge Convolution
- Exercises

Message Passing

- Te instalaste PyG y no habla en ningún lado de convoluciones ...
  - Es más, habla mucho de Message Passing
- Y también en varios papers e incluso en cursos...

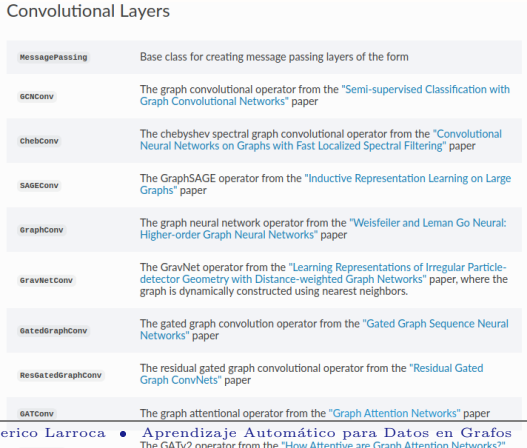




- Pues no es más que lo que ya vimos pero con $K_l = 1$

Message Passing

- En general, casi todas las arquitecturas se pueden implementar con las herramientas que vimos. Ejemplos:
  - Spectral GCNNs: $K_l = N - 1$, $\mathbf{S} = \mathbf{L}$ normalizada [Bruna et al, '14] (distintos autovalores)
  - ChebNets: $\mathbf{S} = \mathbf{L}$ normalizada, $\mathbf{H}_{lk}$ = polinomio de Chebyshev [Defferrard et al, '16]
  - Diffusion CNNs: $\mathbf{S} = \mathbf{A}$, $L = 1$, $F_1 = NF_0$, $\mathbf{H}_{1k}$ = único valor para toda la fila [Atwood et al, '16]
  - GCNs: $\mathbf{S} = (\mathbf{I} + \mathbf{A})$ normalizada, $K_l = 2$, $\mathbf{H}_{l0} = \mathbf{0}$ [Kipf et al, '17]
  - SGCs: $\mathbf{S} = (\mathbf{I} + \mathbf{A})$ normalizada, $\mathbf{H}_{lk} = \mathbf{0}$ para todo $k < K_l$ [Wu et al, '19]
  - GINs: $\mathbf{S} = \mathbf{A}$ binaria, $K_l = 1$, $\mathbf{H}_{l0} = (1 + \epsilon_l)\mathbf{H}_{l1}$, usar $K_l = 0$ para algunos l [Xu et al, '19]



Convolutional Layers

`MessagePassing`	Base class for creating message passing layers of the form
`GCNConv`	The graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper
`ChebConv`	The chebyshev spectral graph convolutional operator from the "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" paper
`SAGEConv`	The GraphSAGE operator from the "Inductive Representation Learning on Large Graphs" paper
`GraphConv`	The graph neural network operator from the "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks" paper
`GravNetConv`	The GravNet operator from the "Learning Representations of Irregular Particle-detector Geometry with Distance-weighted Graph Networks" paper, where the graph is dynamically constructed using nearest neighbors.
`GatedGraphConv`	The gated graph convolution operator from the "Gated Graph Sequence Neural Networks" paper
`ResGatedGraphConv`	The residual gated graph convolutional operator from the "Residual Gated Graph ConvNets" paper
`GATConv`	The graph attentional operator from the "Graph Attention Networks" paper
	The GATv2 operator from the "How Attentive are Graph Attention Networks?"

Funciones de Pérdida

- Aprender con una GNN:

$$\min_{\mathcal{H}} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\boldsymbol{\Phi}(\mathbf{x};\mathbf{S},\mathcal{H}),\mathbf{y}\Big)$$

  $\Rightarrow$ Conjunto de entrenamiento $\mathcal{T}$ de observaciones $\mathbf{x} \in \mathcal{T}$ (generalmente parejas $(\mathbf{x},\mathbf{y})$)

  $\Rightarrow$ Una función de pérdida $\mathsf{J}(\hat{\mathbf{y}})$ que evalúa la bondad del ajuste de $\hat{\mathbf{y}}$ (generalmente $\mathsf{J}(\hat{\mathbf{y}},\mathbf{y})$)

  $\Rightarrow$ Los parámetros $\mathcal{H}$ que definen la GNN

- Ejemplo para el caso de regresión: L2 loss

$$\mathsf{J}(\hat{\mathbf{y}},\mathbf{y}) = \|\hat{\mathbf{y}}-\mathbf{y}\|^2$$

- Ejemplo para el caso de clasificación: Cross-entropy (CE)
  - La salida es categórica (one-hot encoding). Ej: $\mathbf{y} = [0,0,1,0,0]$
  - Además $\hat{\mathbf{y}} = \text{softmax}(\mathbf{x}_L)$ (i.e. $\hat{y}_i = \frac{e^{x_{Li}}}{\sum_j e^{x_{Lj}}}$)

$$\Rightarrow \mathsf{J}(\hat{\mathbf{y}},\mathbf{y}) = -\sum_i y_i \log(\hat{y}_i)$$

Gradient Descent

- ¿Cuál es el método más sencillo para hallar $\mathcal{H}^*$?

$$\mathcal{H}^* = \arg\min_{\mathcal{H}} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{J}\Big(\Phi(\mathbf{x};\mathbf{S},\mathcal{H}),\mathbf{y}\Big) = \arg\min_{\mathcal{H}} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{L}_{\mathbf{x},\mathbf{y}}\Big(\mathcal{H}\Big)$$

- Bajar por el gradiente!

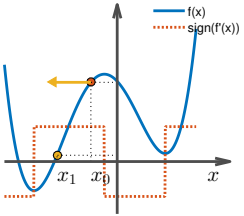
$$\hat{\mathcal{H}} \leftarrow \hat{\mathcal{H}} - \eta \nabla_{\mathcal{H}} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{L}_{\mathbf{x},\mathbf{y}}\Big(\hat{\mathcal{H}}\Big)$$

  $\Rightarrow$ η learning rate (hiperparámetro; puede ir cambiando durante el entrenamiento)

  $\Rightarrow$ Existen varios algoritmos mucho más sofisticados y mejores: Adam, Adagrad, Adadelta, RMSprop

Descenso por Gradiente

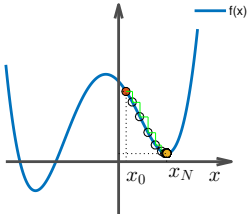
- Supongamos que estamos en el punto x_0 y queremos dar un paso de manera de que $x_1 = x_0 + \Delta x$ cumpla que $f(x_1) \leq f(x_0)$.
- Debemos movernos en la dirección contraria la derivada.
- Si η pequeño $x_1 = x_0 - \eta \text{signo}(f'(x_0))$, es tal que $f(x_1) \leq f(x_0)$



Descenso por Gradiente 1D

$$x_{t+1} = x_t - \eta f'(x_t) \quad \eta > 0 \text{ (step size / learning rate).}$$

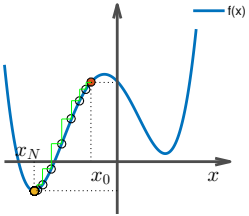
- Si función no es convexa, converge a un mínimo **local** (según x_0)



Descenso por Gradiente 1D

$$x_{t+1} = x_t - \eta f'(x_t) \quad \eta > 0 \text{ (step size / learning rate).}$$

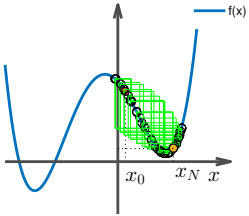
- Si función no es convexa, converge a un mínimo **local** (según x_0)



Descenso por Gradiente 1D

$$x_{t+1} = x_t - \eta f'(x_t) \quad \eta > 0 \text{ (step size / learning rate)}.$$

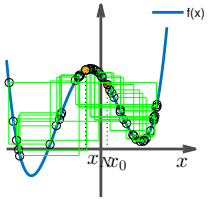
- Si función no es convexa, converge a un mínimo **local** (según x_0)
- η tiene que ser pequeño



Descenso por Gradiente 1D

$$x_{t+1} = x_t - \eta f'(x_t) \quad \eta > 0 \text{ (step size / learning rate)}.$$

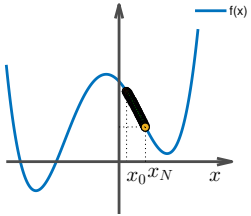
- Si función no es convexa, converge a un mínimo **local** (según x_0)
- η tiene que ser pequeño



Descenso por Gradiente 1D

$$x_{t+1} = x_t - \eta f'(x_t) \quad \eta > 0 \text{ (step size / learning rate).}$$

- Si función no es convexa, converge a un mínimo **local** (según x_0)
- η tiene que ser pequeño ... no tan pequeño



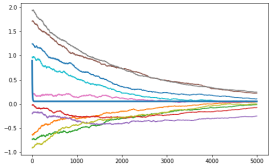
Stochastic Gradient Descent

- ¿Qué problema tiene Gradient Descent? Cada paso del GD implica evaluar el gradiente en todo el set de entrenamiento

$$\hat{\mathcal{H}} \leftarrow \hat{\mathcal{H}} - \eta \nabla_{\mathcal{H}} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{T}} \mathsf{L}_{\mathbf{x},\mathbf{y}}\left(\hat{\mathcal{H}}\right)$$

$$\hat{\mathcal{H}} \leftarrow \hat{\mathcal{H}} - \eta \nabla_{\mathcal{H}} \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{B}} \mathsf{L}_{\mathbf{x},\mathbf{y}}\left(\hat{\mathcal{H}}\right)$$

- Idea Evaluar sólo en un pequeño sub-conjunto al azar
- Intuición: $\mathbb{E}\left\{\frac{1}{|\mathcal{B}|}\sum_{(\mathbf{x},\mathbf{y})\in\mathcal{B}} \mathsf{L}_{\mathbf{x},\mathbf{y}}\left(\hat{\mathcal{H}}\right)\right\} = \frac{1}{|\mathcal{T}|}\sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \mathsf{L}_{\mathbf{x},\mathbf{y}}\left(\hat{\mathcal{H}}\right)$. Ejemplo usando $|\mathcal{B}|$ =2



- Terminología:
  - (mini) Batch: conjunto $\mathcal{B}$
  - Batch size: tamaño de $\mathcal{B}$ (hiperparámetro; típicamente 32, 64 o 128)
  - Iteración: Un paso del GD usando un mini-batch
  - Epoch: Una pasada completa por todo $\mathcal{T}$

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

Método de Backpropagation

- El método de *Backpropagation* es una forma de aplicar sistemáticamente de la regla de la cadena
- Se basa en tres ideas:
  1. Representar las operaciones de la red en un Grafo Computacional
  2. Aplicar el paradigma *Divide-and-conquer*
     - Subdividir el problema principal en subpartes que se puedan resolver eficientemente
     - La combinación de las subsoluciones también debe ser eficiente
  3. Evitar el cálculo explícito de matrices Jacobianas de gran tamaño

12 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

Grafo Computacional: Ejemplo

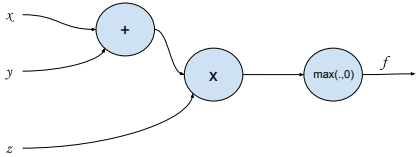
$$f(x,y,z) = \max\Big((x+y)z, 0\Big)$$

Composición de funciones elementales:

$$f_{\text{sum}}(x,y) = x+y \quad \longrightarrow \quad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

$$f_{\text{prod}}(x,y) = xy \quad \longrightarrow \quad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$f_{\max}(x) = \max(x,0) \quad \longrightarrow \quad \frac{\partial f}{\partial x} = 1_{\{x>0\}}$$



13 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

Algoritmo de *Backpropagation*

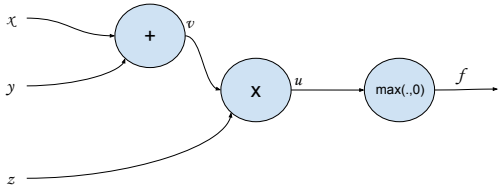
$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$v(x,y) = x+y$

$u(z,v) = zv$

$f(u) = \max(u,0)$

Objetivo: calcular $\nabla f(x,y,z)$ en $x=-2, y=3, z=2$



14 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

Algoritmo de *Backpropagation*

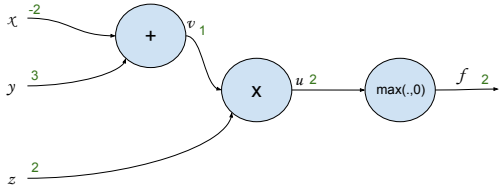
$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$$v(x,y) = x+y$$

$$u(z,v) = zv$$

$$f(u) = \max(u,0)$$

Pasada hacia delante
(Forward pass)



14 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

Algoritmo de *Backpropagation*

$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

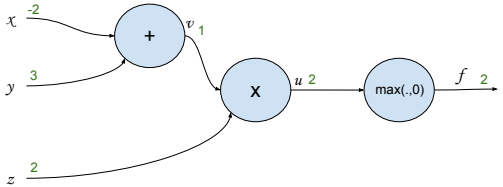
$$v(x,y) = x+y$$

$$u(z,v) = zv$$

$$f(u) = \max(u,0)$$

Pasada hacia atrás
(Backward pass)

Objetivo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



14 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

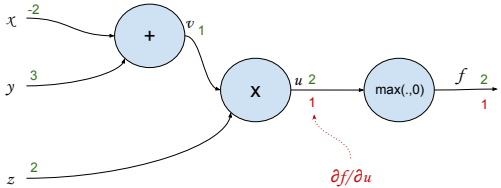
Algoritmo de *Backpropagation*

$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$v(x,y) = x+y$	$\longrightarrow$	$\frac{\partial v}{\partial x} = 1 \quad \frac{\partial v}{\partial y} = 1$
$u(z,v) = zv$	$\longrightarrow$	$\frac{\partial u}{\partial z} = v \quad \frac{\partial u}{\partial v} = z$
$f(u) = \max(u,0)$	$\longrightarrow$	$\frac{\partial f}{\partial u} = 1_{\{u>0\}}$

Pasada hacia atrás
(Backward pass)

$$\frac{\partial f}{\partial u} = 1$$



14 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

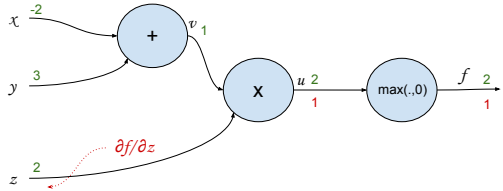
Algoritmo de *Backpropagation*

$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$$\begin{array}{lll} v(x,y) = x+y & \longrightarrow & \dfrac{\partial v}{\partial x} = 1 \quad \dfrac{\partial v}{\partial y} = 1 \\ u(z,v) = zv & \longrightarrow & \dfrac{\partial u}{\partial z} = v \quad \dfrac{\partial u}{\partial v} = z \\ f(u) = \max(u,0) & \longrightarrow & \dfrac{\partial f}{\partial u} = 1_{\{u>0\}} \end{array}$$

Pasada hacia atrás
(Backward pass)

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial u}\overbrace{\frac{\partial u}{\partial z}}^{v=1}$$



14 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

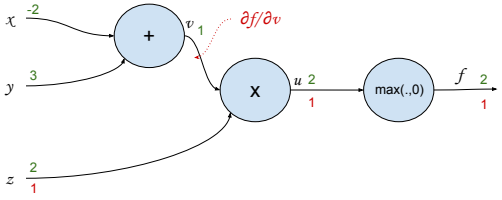
Algoritmo de *Backpropagation*

$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$$\begin{array}{lll} v(x,y)=x+y & \longrightarrow & \dfrac{\partial v}{\partial x}=1 \quad \dfrac{\partial v}{\partial y}=1 \\ u(z,v)=zv & \longrightarrow & \dfrac{\partial u}{\partial z}=v \quad \dfrac{\partial u}{\partial v}=z \\ f(u)=\max(u,0) & \longrightarrow & \dfrac{\partial f}{\partial u}=1_{\{u>0\}} \end{array}$$

Pasada hacia atrás
(Backward pass)

$$\frac{\partial f}{\partial v} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial v}$$



14 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

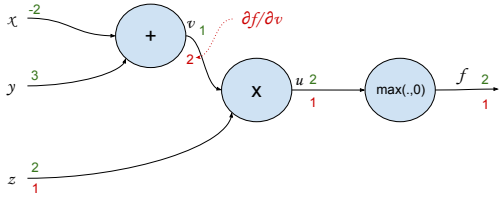
Algoritmo de *Backpropagation*

$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$v(x,y) = x+y$	$\longrightarrow$	$\frac{\partial v}{\partial x} = 1 \quad \frac{\partial v}{\partial y} = 1$
$u(z,v) = zv$	$\longrightarrow$	$\frac{\partial u}{\partial z} = v \quad \frac{\partial u}{\partial v} = z$
$f(u) = \max(u,0)$	$\longrightarrow$	$\frac{\partial f}{\partial u} = 1_{\{u>0\}}$

Pasada hacia atrás
(Backward pass)

$$\frac{\partial f}{\partial v} = \frac{\partial f}{\partial u}\overbrace{\frac{\partial u}{\partial v}}^{z=2} = 2$$



Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

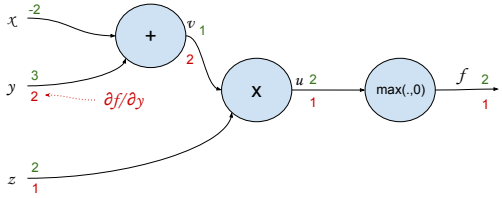
Algoritmo de *Backpropagation*

$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$v(x,y) = x+y$	$\longrightarrow$	$\frac{\partial v}{\partial x} = 1 \quad \frac{\partial v}{\partial y} = 1$
$u(z,v) = zv$	$\longrightarrow$	$\frac{\partial u}{\partial z} = v \quad \frac{\partial u}{\partial v} = z$
$f(u) = \max(u,0)$	$\longrightarrow$	$\frac{\partial f}{\partial u} = 1_{\{u>0\}}$

Pasada hacia atrás
(Backward pass)

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial v}\overbrace{\frac{\partial v}{\partial y}}^{1} = 2$$



14 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

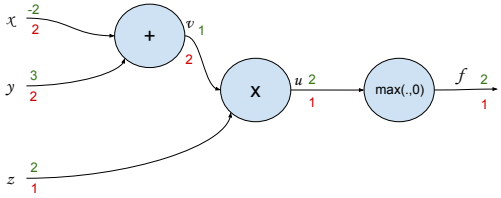
Algoritmo de *Backpropagation*

$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$v(x,y) = x+y$	$\longrightarrow$	$\frac{\partial v}{\partial x} = 1 \quad \frac{\partial v}{\partial y} = 1$
$u(z,v) = zv$	$\longrightarrow$	$\frac{\partial u}{\partial z} = v \quad \frac{\partial u}{\partial v} = z$
$f(u) = \max(u,0)$	$\longrightarrow$	$\frac{\partial f}{\partial u} = 1_{\{u>0\}}$

Pasada hacia atrás
(Backward pass)

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial v}\overbrace{\frac{\partial v}{\partial x}}^{1} = 2$$



14 / 29

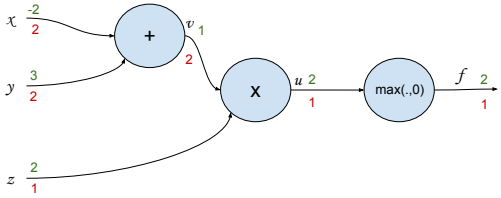
Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

Algoritmo de *Backpropagation*

$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$$\begin{array}{lll} v(x,y) = x+y & \longrightarrow & \dfrac{\partial v}{\partial x} = 1 \quad \dfrac{\partial v}{\partial y} = 1 \\ u(z,v) = zv & \longrightarrow & \dfrac{\partial u}{\partial z} = v \quad \dfrac{\partial u}{\partial v} = z \\ f(u) = \max(u,0) & \longrightarrow & \dfrac{\partial f}{\partial u} = 1_{\{u>0\}} \end{array}$$



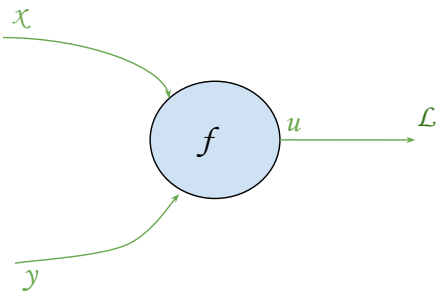
14 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

Algoritmo de Backpropagation: *divide-and-conquer*

Pasada hacia adelante (*forward pass*)



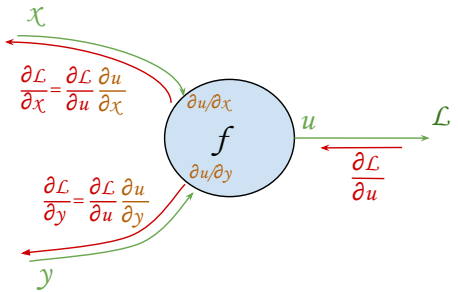
15 / 29

Backpropagation

- ¿Cómo calcular el gradiente eficientemente? Los parámetros pueden ser varios miles

Algoritmo de Backpropagation: *divide-and-conquer*

Gradiente hacia abajo: Regla de la cadena



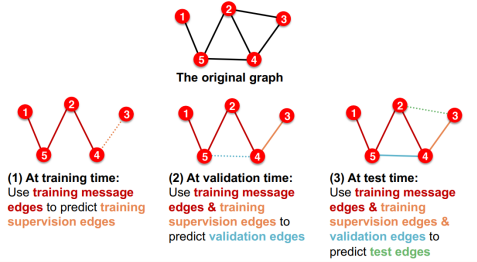
15 / 29

Problemas "canónicos" en GNNs

- Los problemas típicos que se atacan con GNNs caen en tres cateogrías:
  - Predicción a nivel de nodo
  - Predicción a nivel de arista
  - Predicción a nivel de grafo
- Para las predicciones a nivel de nodo puedo usar $(\mathbf{X}_L)_i$ (la salida de la última capa en el nodo i)
  - Decidir si un usuario de una red social es un bot
  - Clasificar el área de un documento en una red de citaciones académicas
  - Estimar la nota de un usuario ante un contenido
- Típicamente pasamos $(\mathbf{X}_L)_i \in \mathbb{R}^{F_L}$ por una capa fully-connected lineal que adapte las dimensiones
- ¿Y el training set?
  1. Semi-supervised learning: en el entrenamiento sólo escondemos las etiquetas de algunos nodos
     - Todos los nodos participan de la GNN (están presentes en $\mathbf{S}$)
     - También llamado Transductive
  2. Inductive learning: en el entrenamiento sólo uso los nodos etiquetados
     - En la GNN sólo participan los nodos del training set durante el entrenamiento ($\mathbf{S}$ recortada)
     - A la hora de predecir, sí uso el $\mathbf{S}$ completo

Problemas "canónicos" en GNNs

- Muchas veces el problema es determinar si hay aristas faltantes en una red/grafo o cuánto vale su peso
  - Recomendación de contenido a usuarios
  - Inferir información nueva en una base de datos relacional
  - Estimar la atenuación en una red inalámbrica
- Podemos usar la salida de la GNN de dos nodos para estimar si existe una arista (o cuánto vale)
- Definimos una función $f((\mathbf{X}_L)_i, (\mathbf{X}_L)_j)$ que diga si la arista existe o de qué categoría es. Ejemplos:
  - $f((\mathbf{X}_L)_i, (\mathbf{X}_L)_j) = \langle (\mathbf{X}_L)_i, (\mathbf{X}_L)_j \rangle$
  - $f((\mathbf{X}_L)_i, (\mathbf{X}_L)_j) = (\mathbf{X}_L)_i^T \mathbf{W} (\mathbf{X}_L)_j$ (fácilmente generalizable a multi-categoría usando varios $\mathbf{W}$)
- ¿Y el training set?
  1. Transductive: Separamos los enlaces en training, validation y test, y sólo los usamos en esas etapas

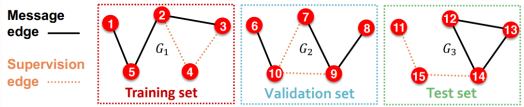


  2. Inductive: Contamos con grafos distintos para training, validación y test

Slides del curso cs224W, Leskovec, Stanford University

Problemas "canónicos" en GNNs

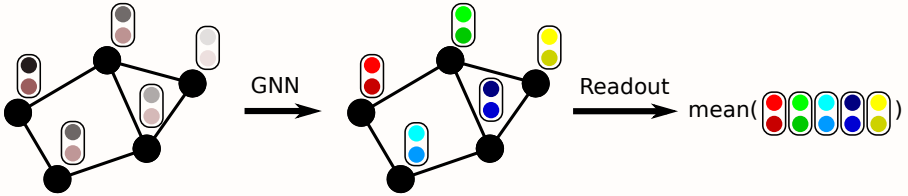
- Muchas veces el problema es determinar si hay aristas faltantes en una red/grafo o cuánto vale su peso
  - Recomendación de contenido a usuarios
  - Inferir información nueva en una base de datos relacional
  - Estimar la atenuación en una red inalámbrica
- Podemos usar la salida de la GNN de dos nodos para estimar si existe una arista (o cuánto vale)
- Definimos una función $f((\mathbf{X}_L)_i, (\mathbf{X}_L)_j)$ que diga si la arista existe o de qué categoría es. Ejemplos:
  - $f((\mathbf{X}_L)_i, (\mathbf{X}_L)_j) = \langle (\mathbf{X}_L)_i, (\mathbf{X}_L)_j \rangle$
  - $f((\mathbf{X}_L)_i, (\mathbf{X}_L)_j) = (\mathbf{X}_L)_i^T \mathbf{W} (\mathbf{X}_L)_j$ (fácilmente generalizable a multi-categoría usando varios $\mathbf{W}$)
- ¿Y el training set?
  1. Transductive: Separamos los enlaces en training, validation y test, y sólo los usamos en esas etapas
  2. Inductive: Contamos con grafos distintos para training, validación y test



Slides del curso cs224W, Leskovec, Stanford University

Problemas “canónicos” en GNNs

- Por último, existen problemas donde nos gustaría clasificar o hacer regresión sobre el grafo completo:
  - Predecir propiedades como toxicidad o solubilidad de una molécula
  - Clasificar si una obra es de cierto autor
  - Brindar la posición de un usuario basado en medidas de RSSI de varios APs Wi-Fi
- Quizá sea el más similar a aprendizaje supervisado tradicional
- Tomamos toda la salida de la GNN $\mathbf{X}_L$ y le aplicamos una capa de readout o pooling
  - Si la identidad de cada nodo no importa, tenemos que usar funciones de readout como máx o mean
  - Si la identidad de cada nodo importa, podemos concatenar las salidas



Problemas “canónicos” en GNNs

- Por último, existen problemas donde nos gustaría clasificar o hacer regresión sobre el grafo completo:
  - Predecir propiedades como toxicidad o solubilidad de una molécula
  - Clasificar si una obra es de cierto autor
  - Brindar la posición de un usuario basado en medidas de RSSI de varios APs Wi-Fi
- Quizá sea el más similar a aprendizaje supervisado tradicional
- Tomamos toda la salida de la GNN $\mathbf{X}_L$ y le aplicamos una capa de readout o pooling
  - Si la identidad de cada nodo no importa, tenemos que usar funciones de readout como máx o mean
  - Si la identidad de cada nodo importa, podemos concatenar las salidas