Deep Learning

Graph Neural Networks

Lucas Silveira Kupssinskü

Machine Learning Theory and Applications Laboratory
PUCRS

agosto de 2026

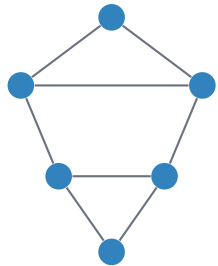
Visão Geral

Visão Geral

O Mundo é Cheio de Grafos

Muitos dados naturais não vivem em uma grade regular:

- **Redes sociais**: usuários e suas conexões
- **Moléculas**: átomos ligados por ligações químicas
- **Citações científicas**: artigos referenciando uns aos outros
- **Sistemas de recomendação**: bipartido usuário–item
- **Grafos de conhecimento**: entidades e relações
- **Tráfego e logística**: cidades conectadas por rotas



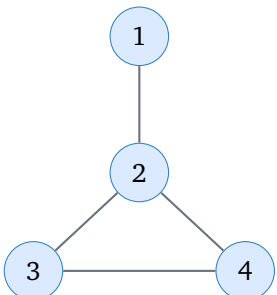
Notação e Tipos de Grafos

Definição básica

Um grafo é $G = (V, E)$ com $|V| = n$ nós e $|E| = m$ arestas. A *adjacency matrix* $A \in \{0,1\}^{n \times n}$ codifica E. Atributos de nó vivem em $X \in \mathbb{R}^{n \times d}$.

- **Dirigido vs não-dirigido**: A simétrica ou não.
- **Ponderado vs não-ponderado**: $A_{ij} \in \mathbb{R}$ ou $\{0,1\}$.
- **Homogêneo vs heterogêneo**: tipos únicos de nó/aresta ou múltiplos (e.g., paciente, doença, fármaco em uma KG médica).
- **Estático vs dinâmico**: G fixo ou evoluindo no tempo.

Exemplo: Matriz de Adjacência



$V = \{1, 2, 3, 4\}$
$E = \{(1, 2), (2, 3), (2, 4), (3, 4)\}$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

- Linha i: vizinhança de i.
- $A_{ij} = 1$ se $(i, j) \in E$.
- Não-dirigido $\Rightarrow$ A **simétrica**.
- Soma da linha i = grau d_i.

Esse mesmo grafo será o exemplo numérico para GCN mais à frente.

O Que Queremos Aprender

- Representações vetoriais $h_v \in \mathbb{R}^{d'}$ para cada nó v.
- Que essas representações sejam úteis em tarefas *downstream*: classificar nós, prever arestas, classificar o grafo inteiro.
- E que respeitem a estrutura do grafo: vizinhança, simetrias, escala.

O desafio

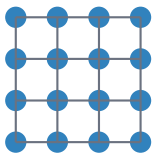
Diferente de imagens (grade regular) e texto (sequência), um grafo **não tem ordem canônica** nem **vizinhança de tamanho fixo**. Não dá para empilhar uma CNN diretamente em cima.

Visão Geral

Domínio Euclidiano vs Não-Euclidiano

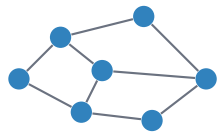
Imagens, áudio, séries temporais

- Grade regular, ordem natural.
- Vizinhança de tamanho fixo (e.g., 3×3).
- Translação é uma simetria do domínio.
- CNNs/RNNs aproveitam isso.



Grafos

- Estrutura irregular.
- Cada nó tem grau diferente.
- Sem ordem canônica entre os nós.
- Translação não faz sentido.



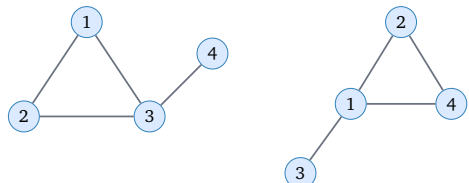
Sem Ordem Canônica

Um mesmo grafo G pode ser representado por matrizes de adjacência A **diferentes**, dependendo de como rotulamos os nós.

Se P é qualquer permutação dos nós, então A e $PAP^\top$ descrevem o **mesmo grafo**, apenas com etiquetas trocadas.

Implicação

Qualquer modelo $f(A, X)$ que veja G deve ser indiferente a essa permutação: $f(PAP^\top, PX) = f(A, X)$ (no caso invariante).



Mesmo grafo, duas rotulagens, duas matrizes A distintas.

Invariância e Equivariância à Permutação

Seja P uma matriz de permutação $n \times n$.

- Função **invariante** (saída no nível-grafo, e.g., classe da molécula):

$$f(PAP^\top, PX) = f(A, X)$$

- Função **equivariante** (saída no nível-nó, e.g., embedding por nó):

$$f(PAP^\top, PX) = P\, f(A, X)$$

Princípio de projeto

Uma GNN deve ser **equivariante por construção** no estágio de message passing, e tornar-se **invariante** apenas no *readout* final quando a tarefa exige saída por grafo.

Paralelo: CNNs codificam invariância/equivariância à translação. GNNs codificam invariância/equivariância à permutação.

(*Message passing* e *readout* são detalhados adiante; por ora: "trocar informação entre vizinhos" e "produzir a saída final da tarefa".)

Visão Geral

Três Tarefas Canônicas

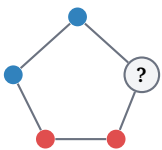
Tarefa	**O que se prevê**	**Exemplo**
Node classification	Rótulo de um nó	Categoria de artigo em rede de citações
Link prediction	Existe/existirá uma aresta (u, v)?	Recomendar item para usuário; completar KG
Graph classification	Rótulo do grafo inteiro	Toxicidade de molécula; propriedade química

Cada tarefa demanda uma estratégia de *readout* diferente:

- Nó: usar diretamente h_v.
- Aresta: combinar h_u e h_v (concatenação, produto interno, MLP).
- Grafo: agregação (*pooling*) sobre todos os h_v (soma, média, máx, atenção).

As Três Tarefas, Visualmente

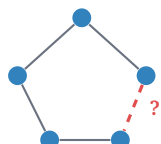
Node classification



De qual classe é o nó “?”

readout: usar h_v direto

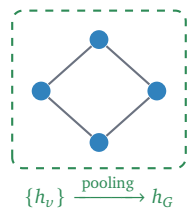
Link prediction



A aresta tracejada existe ou existirá?

readout: combinar h_u e h_v

Graph classification



O grafo inteiro é tóxico?

readout: pooling sobre $\{h_v\}$

Transdutivo vs Indutivo

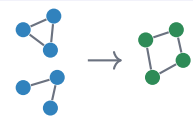
Transdutivo

- Treino e teste no **mesmo grafo**.
- Vemos todos os nós, mas só rótulos de alguns.
- Caso clássico: rede de citações.

• treino ○ teste: **mesmo grafo**

Indutivo

- Treino em um conjunto de grafos, teste em **grafos novos** (ou subgrafos não vistos).
- Casos: moléculas novas; usuários novos em uma rede que cresce.



• grafos de treino • **grafo novo** no teste

Por que importa

GCN precisa do grafo todo; GraphSAGE nasce indutivo via *sampling*.

Visão Geral

Open Graph Benchmark[1]

- Conjunto de **benchmarks padronizados** para tarefas em grafos.
- Splits, métricas e protocolos de avaliação **fixos e públicos** ⇒ leaderboard reproduzível.
- Três famílias, uma por tarefa:
  - `ogbn-*`: classificação de nó.
  - `ogbl-*`: predição de aresta (*link prediction*).
  - `ogbg-*`: classificação de grafo.
- Escala realista: `ogbn-papers100M` tem ~111 M nós e ~1.6 B arestas. **Não cabe em GPU**.

Por que precisamos de OGB

Sem benchmarks padronizados, splits arbitrários e datasets pequenos geram rankings instáveis: comparações entre modelos viram ruído.

[1] Weihua Hu et al. "Open Graph Benchmark: Datasets for Machine Learning on Graphs". Em: NeurIPS. 2020.

Um Exemplo OGB Por Tarefa

Família	Dataset	Tarefa	Métrica
`ogbn-*`	`ogbn-arxiv`	Classificar artigo do arXiv por subárea	Accuracy
`ogbl-*`	`ogbl-ddi`	Prever interação fármaco–fármaco	Hits@20
`ogbg-*`	`ogbg-molhiv`	Prever se molécula inibe HIV	ROC-AUC

- Cada dataset vem com **split fixo**: treino, validação, teste.
- Para link prediction, o split também inclui um **conjunto fixo de negativos amostrados**, sem isso comparações não seriam reproduzíveis.
- Pipeline padrão em *PyTorch Geometric* (PyG) ou *DGL*.

Métricas por Tarefa

Tarefa	Métrica padrão	Específica de grafo?
Node classification	Accuracy, F1 (macro/micro)	Não
Graph classification	ROC-AUC, Accuracy, AP (multi-task)	Não
Link prediction	**Hits@K, MRR**	**Sim**

Por que link prediction é diferente

Em um grafo com n nós existem $O(n^2)$ arestas possíveis. Calcular AUC contra todas as não-arestas é caro e o desbalanceamento é absurdo ($> 99.99\%$ de negativos). A solução é **negative sampling**: avaliamos cada positivo contra um subconjunto fixo de negativos amostrados.

Hits@K e MRR
Definições

Setup: para cada query (nó ou par), o modelo recebe 1 positivo v^* e N negativos amostrados, produz um *score* para cada candidato e ordena de modo decrescente. Seja r a posição (*rank*) do positivo nesse ranking ($r = 1$ no topo).

Hits@K

$$\text{Hits@K}_q = \mathbb{1}[r_q \leq K]$$

Média sobre todas as queries:

$$\text{Hits@K} = \frac{1}{|Q|} \sum_{q \in Q} \mathbb{1}[r_q \leq K]$$

Mean Reciprocal Rank

$$\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{r_q}$$

Sensível ao topo: cair de rank $1 \to 2$ vale 0.5; de $10 \to 11$ vale apenas ~ 0.009.

Exemplo Trabalhado: 3 Queries

Estilo `ogbl-ddi`: cada query tem 1 positivo e 4 negativos.

Query	Score v^*	Scores neg.	Rank v^*	Hits@1	Hits@3	RR
u_1	0.70	0.90, 0.80, 0.50, 0.40	3	0	1	1/3
u_2	0.95	0.60, 0.50, 0.30, 0.10	1	1	1	1
u_3	0.20	0.90, 0.80, 0.70, 0.60	5	0	0	1/5

- Hits@1 = $(0 + 1 + 0)/3 \approx \mathbf{0.333}$
- Hits@3 = $(1 + 1 + 0)/3 \approx \mathbf{0.667}$
- MRR = $(1/3 + 1 + 1/5)/3 \approx \mathbf{0.511}$

Note como a query u_2 (rank 1) puxa o MRR para cima sozinha, enquanto Hits@K apenas conta acertos binários.

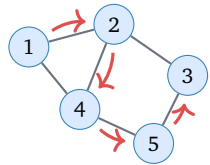
Visão Geral

Random Walks como Sinal Supervisor[2]

Antes das GNNs, a abordagem dominante era **aprender embeddings** de nós via *random walks*.

DeepWalk (Perozzi, Al-Rfou e Skiena, 2014)

1. Para cada nó v, gere γ *random walks* de comprimento ℓ.
2. Trate cada walk como uma "sentença" de nós.
3. Aplique *skip-gram* (Word2Vec) sobre essas sentenças: maximize $P(v_j \mid v_i)$ para nós que coocorrem em uma janela.



"sentença": $(v_1, v_2, v_4, v_5, v_3)$

skip-gram: maximizar $P(v_4 \mid v_2)$, $P(v_5 \mid v_4)$, ...

Resultado: vizinhanças similares $\Rightarrow$ embeddings próximos.

[2] Bryan Perozzi, Rami Al-Rfou e Steven Skiena. "DeepWalk: Online Learning of Social Representations". Em: KDD. 2014, pp. 701–710.

node2vec e o Limite dos Embeddings Pré-GNN

node2vec (Grover e Leskovec, 2016) generaliza DeepWalk com random walks parametrizados por (p, q):

- p (*return*) controla a probabilidade de voltar ao nó anterior.
- q (*in-out*) interpola entre busca local (tipo BFS, homofilia) e exploração distante (tipo DFS, equivalência estrutural).

Permite balancear *homofilia* vs *equivalência estrutural*.

Limites fundamentais

- **Transdutivo**: embeddings só existem para nós vistos no treino.
- **Sem atributos**: ignora X.
- **Não generaliza entre grafos**.
- Sem fim-a-fim com tarefas *downstream*.

Para onde queremos ir

Um modelo que (i) seja **indutivo**, (ii) use atributos X, (iii) treine fim-a-fim, e (iv) compartilhe pesos entre grafos. $\Rightarrow$ **GNNs**.

Visão Geral

O Framework de Message Passing[3]

Praticamente todas as GNNs populares se encaixam em um único esquema:

Para cada camada $k = 1, \ldots, K$ e cada nó v:

1. **Message**: cada vizinho u envia uma mensagem

$$m_{u \to v}^{(k)} = M^{(k)}\left(h_u^{(k-1)},\, h_v^{(k-1)},\, e_{uv}\right)$$

2. **Aggregate**: v combina mensagens dos vizinhos $\mathcal{N}(v)$

$$a_v^{(k)} = \mathrm{AGG}\left(\{m_{u \to v}^{(k)} : u \in \mathcal{N}(v)\}\right)$$

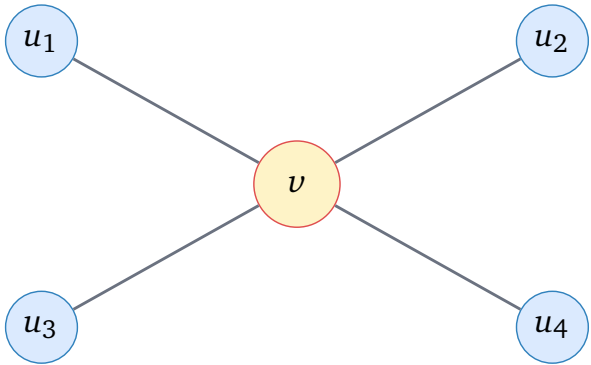
3. **Update**: v atualiza seu próprio embedding

$$h_v^{(k)} = U^{(k)}\left(h_v^{(k-1)},\, a_v^{(k)}\right)$$

Convenções: $h_v^{(0)} = X_v$ (atributos iniciais do nó), $\mathcal{N}(v)$ é a vizinhança de v e e_{uv} é o atributo (opcional) da aresta.

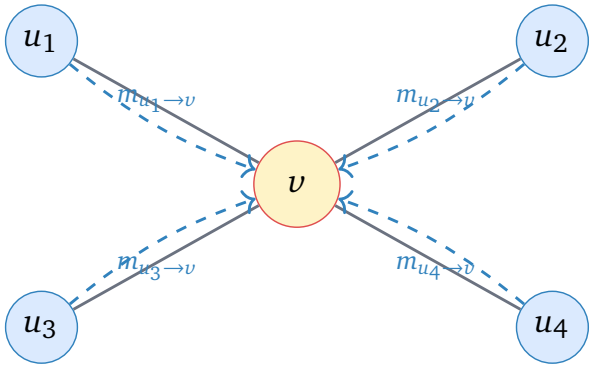
[3] Justin Gilmer et al. "Neural Message Passing for Quantum Chemistry". Em: ICML. 2017, pp. 1263–1272.

Uma Camada de Message Passing



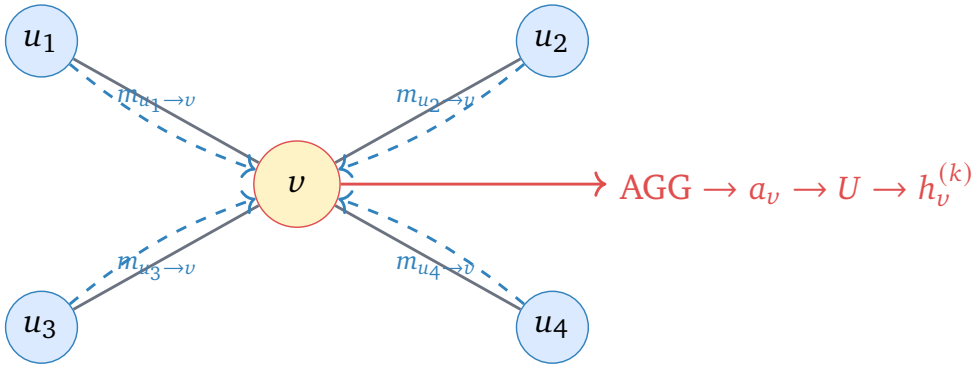
- A estrutura do grafo define **quem fala com quem**.
- Cada vizinho envia uma mensagem; v agrega e atualiza.
- Após K camadas, h_v contém informação a até K saltos: empilhar camadas $\approx$ aumentar o *receptive field*.

Uma Camada de Message Passing



- A estrutura do grafo define **quem fala com quem**.
- Cada vizinho envia uma mensagem; v agrega e atualiza.
- Após K camadas, h_v contém informação a até K saltos: empilhar camadas $\approx$ aumentar o *receptive field*.

Uma Camada de Message Passing

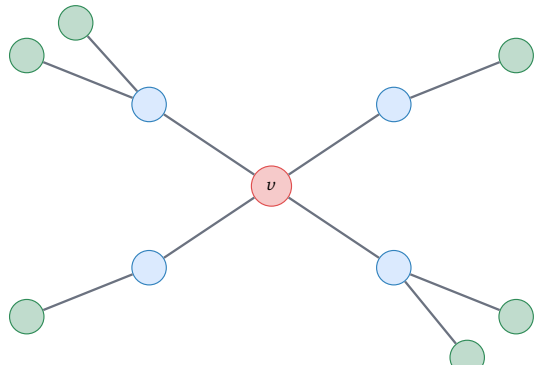


- A estrutura do grafo define **quem fala com quem**.
- Cada vizinho envia uma mensagem; v agrega e atualiza.
- Após K camadas, h_v contém informação a até K saltos: empilhar camadas $\approx$ aumentar o *receptive field*.

Receptive Field e Profundidade

- Camada 1: v vê seus vizinhos imediatos.
- Camada 2: v vê vizinhos dos vizinhos (2-hops).
- Camada K: v vê todos os nós a até K saltos.

Em grafos com pequeno diâmetro (e.g., redes sociais, “6 graus de separação”), $K = 3$ a 5 já cobre quase todo o grafo a partir de cada nó.



Visão Geral

GCN
Intuição[4]

(Kipf e Welling, 2017) propõem a forma mais simples de message passing: **média ponderada dos vizinhos**.

Forma por nó

$$h_v^{(k)} = \sigma\left(W^{(k)} \cdot \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\tilde{d}_v \, \tilde{d}_u}} h_u^{(k-1)} \right)$$

onde $\tilde{d}_v$ é o grau de v no grafo com auto-laços adicionados.

- Mensagem = identidade (h_u).
- Agregação = soma normalizada simétrica.
- Update = projeção linear $W^{(k)}$ + não-linearidade σ.

~~A normalização **simétrica** (não a média $1/\tilde{d}_v$) evita que vizinh~~os de grau alto dominem a agregação.

[4] Thomas N. Kipf e Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". Em: ICLR. 2017.

GCN
Forma Matricial

Definindo $\tilde{A} = A + I$ (auto-laços, para v manter a própria informação) e $\tilde{D} = \text{diag}(\tilde{A}\mathbf{1})$:

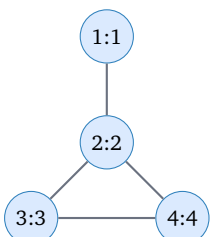
Camada GCN inteira de uma vez

$$H^{(k)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}\, H^{(k-1)}\, W^{(k)}\right)$$

- $H^{(k)} \in \mathbb{R}^{n \times d_k}$: matriz de embeddings por nó.
- $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$: **depende apenas do grafo**, é constante durante o treino.
- Implementação eficiente: multiplicação esparsa-densa.
- **Limitação**: precisa do grafo todo em memória $\Rightarrow$ transdutivo e custoso em escala.

Exemplo Numérico: GCN em 4 Nós (1/2)

Mesmo grafo da Seção 1. Atributos iniciais 1D: $h^{(0)} = (1, 2, 3, 4)^\top$. Para clareza fazemos $W^{(1)} = 1$ (escalar) e $\sigma = \mathrm{Id}$.



$$\tilde{A} = A + I = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Graus em $\tilde{A}$:

$$\tilde{d} = (2,\ 4,\ 3,\ 3)$$

$$\tilde{D}^{-1/2} = \mathrm{diag}\left(\tfrac{1}{\sqrt{2}},\ \tfrac{1}{2},\ \tfrac{1}{\sqrt{3}},\ \tfrac{1}{\sqrt{3}}\right)$$

Próximo passo: aplicar $\tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2}$ a $h^{(0)}$.

Exemplo Numérico: GCN em 4 Nós (2/2)

Matriz normalizada $\hat{A} = \tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2}$ com entrada $(i,j) = \tilde{A}_{ij}/\sqrt{\tilde{d}_i\,\tilde{d}_j}$:

$$\hat{A} \approx \begin{bmatrix} 0.50 & 0.35 & 0 & 0 \\ 0.35 & 0.25 & 0.29 & 0.29 \\ 0 & 0.29 & 0.33 & 0.33 \\ 0 & 0.29 & 0.33 & 0.33 \end{bmatrix}$$

Aplicando a $h^{(0)} = (1,2,3,4)^\top$, por exemplo $h_1^{(1)} = 0.50 \cdot 1 + 0.354 \cdot 2 \approx 1.21$:

$$h^{(1)} = \hat{A}\,h^{(0)} \approx (1.21,\ 2.87,\ 2.91,\ 2.91)^\top$$

Observação

$h_3^{(1)} = h_4^{(1)}$ porque os nós 3 e 4 são **simétricos no grafo**. GNNs não distinguem nós com a mesma estrutura local quando os atributos não os diferenciam; voltaremos a essa ideia em *expressividade*.

Visão Geral

GraphSAGE
Motivação[5]

GCN tem duas limitações que importam em escala:

- **Transdutivo**: precisa do grafo todo para computar $H^{(k)}$.
- **Custo de memória**: $\tilde{A}$ esparsa de `ogbn-papers100M` tem ~1.6 B entradas. Não cabe em GPU.

A ideia de (Hamilton, Ying e Leskovec, 2017)

Em vez de operar no grafo todo, **amostrar uma vizinhança fixa** $S_k(v) \subseteq \mathcal{N}(v)$ por nó por camada. O custo por minibatch passa a depender de $|S_k|^K$, não de n.

Bônus: como o modelo aprende uma função $f(\cdot)$ que age sobre vizinhanças locais, ele é naturalmente **indutivo**: pode ser aplicado a grafos novos.

[5] William L. Hamilton, Rex Ying e Jure Leskovec. "Inductive Representation Learning on Large Graphs". Em: NeurIPS. 2017.

GraphSAGE
Algoritmo

Funções AGG comuns

- **Mean**: média dos vizinhos.
- **Pool**: max-pool de $\mathrm{MLP}(h_u)$.
- **LSTM**: LSTM sobre vizinhos (com ordem aleatória, para preservar permutação).

Concatenação no update

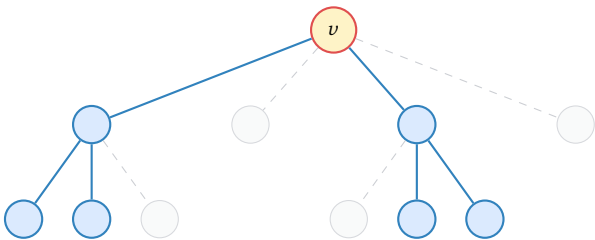
GraphSAGE **concatena** h_v com a agregação, em vez de fundi-los como GCN. Isso preserva melhor a identidade do nó central.

Require: Grafo G, atributos X, K camadas, tamanhos $S_1, \ldots, S_K$

$h_v^{(0)} \leftarrow X_v$ para todo v
for $k = 1, \ldots, K$ **do**
  for cada nó v no minibatch **do**
    $S_k(v) \leftarrow$ amostra de S_k vizinhos de $\mathcal{N}(v)$
    $a_v^{(k)} \leftarrow \mathrm{AGG}(\{h_u^{(k-1)} : u \in S_k(v)\})$
    $h_v^{(k)} \leftarrow \sigma\big(W^{(k)} \, [h_v^{(k-1)} \,\|\, a_v^{(k)}]\big)$
    $h_v^{(k)} \leftarrow h_v^{(k)} / \|h_v^{(k)}\|_2$
  end for
end for
return $\{h_v^{(K)}\}$

Neighbor Sampling, Visualmente

Árvore de computação de **um** nó v com $K = 2$ camadas e $S_1 = S_2 = 2$:



saída: $h_v^{(2)}$

camada 1: $S_1 = 2$ dos 4 vizinhos

camada 2: $S_2 = 2$ por nó amostrado

- Custo por nó: no máximo $1 + S_1 + S_1 S_2 = 7$ nós visitados, **independente de** n.
- A amostra é redesenhada a cada época: a variância extra também age como regularização.

Visão Geral

GAT
Vizinhos Não São Iguais[6]

GCN dá peso fixo $1/\sqrt{\tilde{d}_v\,\tilde{d}_u}$ a cada vizinho, depende só do grau. (Veličković et al., 2018) propõem **aprender** o peso a partir do conteúdo dos embeddings.

Coeficientes de atenção

$$e_{vu} = \text{LeakyReLU}\big(\mathbf{a}^\top\,[\,Wh_v \,\|\, Wh_u\,]\big)$$

$$\alpha_{vu} = \text{softmax}_{u\in\mathcal{N}(v)}(e_{vu}) = \frac{\exp(e_{vu})}{\sum_{u'\in\mathcal{N}(v)}\exp(e_{vu'})}$$

$$h_v^{(k)} = \sigma\left(\sum_{u\in\mathcal{N}(v)} \alpha_{vu}\, W h_u^{(k-1)}\right)$$

Os pesos α_{vu} dependem dos embeddings $\Rightarrow$ ajustam-se à tarefa.

[6] Petar Veličković et al. "Graph Attention Networks". Em: ICLR. 2018.

Multi-Head e Ligação com Transformers

Como em Transformers, **múltiplas cabeças** de atenção são empilhadas:

$$h_v^{(k)} = \Big\|_{i=1}^{H} \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(i)} W^{(i)} h_u^{(k-1)} \right)$$

Na camada final costuma-se **tirar a média** das cabeças em vez de concatenar.

Multi-head dá ao modelo a chance de aprender diferentes "tipos de relação" entre vizinhos no mesmo nível.

Ligação com Transformers

- *Self-attention* é GAT em um **grafo completamente conectado** (cada token vê todos os tokens).
- GAT é o caso onde a estrutura do grafo **limita** a atenção aos vizinhos.
- Voltaremos a esse paralelo em Graph Transformers.

Exemplo Numérico: Coeficientes de Atenção em GAT

Nó v com 3 vizinhos. Após projeção, $Wh_v = (1, 0)$ e

$$Wh_{u_1} = (1, 0), \qquad Wh_{u_2} = (0, 1), \qquad Wh_{u_3} = (0.5,\ 0.5).$$

Vetor de atenção (parâmetro aprendido): $\mathbf{a} = (1, 0, 1, 0)$, de dimensão 4 (2× a dos embeddings projetados, por causa da concatenação).

Scores $e_{v,u_k} = \mathbf{a}^\top [Wh_v \,\|\, Wh_{u_k}]$ (todos positivos, LeakyReLU é identidade):

$$e_{v,u_1} = (1, 0, 1, 0) \cdot (1, 0, 1, 0) = 2$$
$$e_{v,u_2} = (1, 0, 1, 0) \cdot (1, 0, 0, 1) = 1$$
$$e_{v,u_3} = (1, 0, 1, 0) \cdot (1, 0, 0.5, 0.5) = 1.5$$

Softmax sobre os scores:

$$\alpha \approx (\,0.51,\ 0.19,\ 0.31\,)$$

Embedding final de v

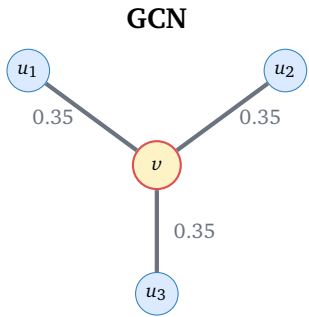
$$h_v^{\text{novo}} = \sum_k \alpha_{v,u_k} Wh_{u_k}$$

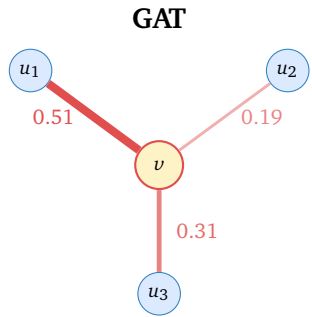
$$\approx 0.51\,(1, 0) + 0.19\,(0, 1) + 0.31\,(0.5,\ 0.5)$$

$$\approx (\,0.66,\ 0.34\,)$$

u_1 (mais "parecido" com v na 1ª coordenada) recebe maior peso α.

Visualizando: Pesos Fixos (GCN) vs Aprendidos (GAT)





Peso $1/\sqrt{\tilde{d}_v \tilde{d}_u}$ depende **só dos graus**: igual para qualquer tarefa e entrada.

Pesos α_{vu} do exemplo anterior: dependem do **conteúdo** dos embeddings.

A espessura da aresta é o quanto cada vizinho “fala” na agregação.

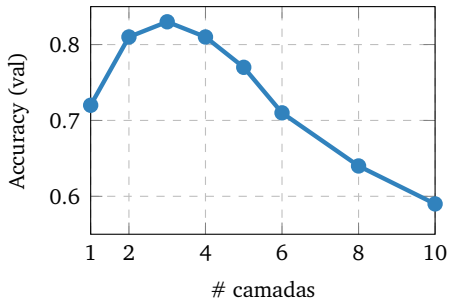
Visão Geral

O Fenômeno do *Oversmoothing*

Empilhar muitas camadas de message passing tende a fazer todos os nós convergirem para o **mesmo embedding**.

- Após K camadas, cada nó vê informação a K-hops.
- Em grafos pequenos (diâmetro $\leq K$), todos veem o grafo inteiro.
- Mensagens repetidas $\Rightarrow$ embeddings se misturam.

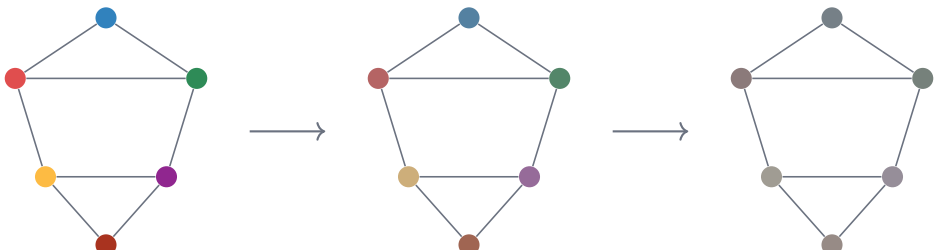
Resultado: GCN/GAT clássicos param de ganhar com mais de ~3 a 4 camadas; em alguns casos, pioram.



Tendência típica em GCN.

Visualizando o Oversmoothing

Cor do nó = embedding. O mesmo grafo após k camadas de média com os vizinhos:



$k = 0$ $k = 2$ $k = 8$

A cada camada, cada nó vira uma média dos vizinhos: a **distância entre embeddings cai** até todos ficarem indistinguíveis (ilustração esquemática).

Mitigações

- **Skip connections** (estilo ResNet): $h_v^{(k)} = h_v^{(k-1)} + \mathrm{GNN}^{(k)}(\ldots)$.
- **Jumping Knowledge** (JK-Net): concatena ou faz max-pool das ativações de *todas* as camadas no final.
- **DropEdge**: descarta arestas aleatoriamente a cada época, análogo a dropout para a estrutura.
- **Normalização**: *LayerNorm*, *GraphNorm*, *PairNorm* ajudam a estabilizar magnitudes.
- **Atenção esparsa global** (Graph Transformers): rota mensagens de longo alcance sem precisar empilhar muitas camadas locais.

Trade-off central

Profundidade ↔ *receptive field* ↔ diversidade de embeddings. Não dá para maximizar os três simultaneamente.

Visão Geral

Quão Poderosa é uma GNN?

Pergunta formal: que pares de grafos uma GNN consegue distinguir?

- Se duas estruturas produzem embeddings idênticos, a GNN não pode separá-las em nenhuma tarefa *downstream*.
- Algumas estruturas (e.g., contar triângulos, ciclos longos) são surpreendentemente difíceis para message passing puro.

Roteiro da seção

(i) Um teste clássico de isomorfismo, o **1-WL**; (ii) o resultado de que toda MPNN é **no máximo** tão poderosa quanto ele; (iii) um par de grafos concreto em que ambos falham; (iv) a arquitetura que **atinge** esse teto, a GIN.

Teste de Weisfeiler-Lehman (1-WL)

Heurística clássica (1968) para **testar isomorfismo de grafos**. Para cada nó v:

1. Inicialize com um rótulo $c_v^{(0)}$ (e.g., grau).
2. Em cada iteração ($\{\!\{\cdot\}\!\}$ é um *multiset*: um conjunto que conta repetições):

$$c_v^{(k)} = \text{HASH}\left(c_v^{(k-1)},\ \{\!\{\, c_u^{(k-1)} : u \in \mathcal{N}(v) \,\}\!\}\right)$$

3. Repita até estabilizar; compare os **multisets** de rótulos entre dois grafos. Diferentes $\Rightarrow$ não isomorfos; iguais $\Rightarrow$ **inconclusivo**.

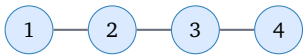
Conexão com message passing

Compare a equação acima com a forma genérica **message–aggregate–update**. É **exatamente o mesmo padrão**, com HASH substituindo $U(\cdot, \text{AGG}(\cdot))$. WL é message passing com hash injetivo.

Exemplo: Rodando 1-WL à Mão

Dois grafos com 4 nós; rotulamos todos com $c_v^{(0)} = 1$ inicialmente.

G_1: caminho

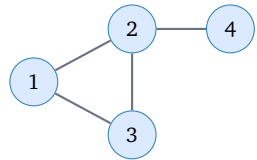


Iter 1, $\text{hash}(c_v, \{c_u\}_{u \in \mathcal{N}(v)})$:

- Nó 1: $\text{hash}(1, \{1\}) = A$
- Nó 2: $\text{hash}(1, \{1, 1\}) = B$
- Nó 3: $\text{hash}(1, \{1, 1\}) = B$
- Nó 4: $\text{hash}(1, \{1\}) = A$

Multiset: $\{A, A, B, B\}$

G_2: triângulo + pendant



Iter 1:

- Nó 1: $\text{hash}(1, \{1, 1\}) = B$
- Nó 2: $\text{hash}(1, \{1, 1, 1\}) = C$
- Nó 3: $\text{hash}(1, \{1, 1\}) = B$
- Nó 4: $\text{hash}(1, \{1\}) = A$

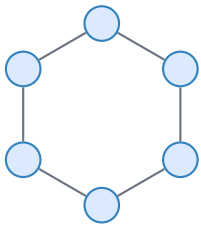
Multiset: $\{A, B, B, C\}$

Multisets diferem $\Rightarrow$ **1-WL distingue G_1 e G_2 em uma única iteração**.

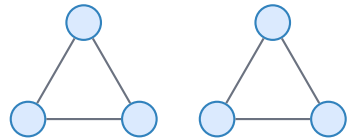
Onde o 1-WL (e Toda MPNN) Falha

Dois grafos **não isomorfos** em que todo nó tem grau 2:

G_1**: ciclo de 6**



G_2**: dois triângulos**



Por que o teste empata

Em **toda** iteração, todo nó dos dois grafos computa o mesmo $\text{HASH}(c, \{\!\{c, c\}\!\})$: os multisets nunca diferem. Com atributos iguais, nenhuma GNN de message passing produz embeddings diferentes para G_1 e G_2.

Consequência prática: MPNNs puras **não contam triângulos nem ciclos**.

GIN
Tão Poderoso Quanto WL[7]

(Xu et al., 2019) mostram:

- Toda GNN baseada em message passing é **no máximo tão poderosa** quanto o teste 1-WL.
- Para alcançar esse teto, AGG precisa ser **injetivo** sobre multisets.
- Soma é injetiva (a menos de medida zero); média e máximo **não são**.

Camada GIN

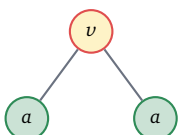
$$h_v^{(k)} = \mathrm{MLP}^{(k)}\Big((1+\epsilon^{(k)})\, h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}\Big)$$

Soma + MLP $\Rightarrow$ injetividade. Em prática, GIN frequentemente vence GCN/GAT em tarefas de classificação de grafo.

[7] Keyulu Xu et al. "How Powerful are Graph Neural Networks?" Em: ICLR. 2019.

Por Que Soma, e Não Média ou Máximo?

Dois nós com vizinhanças **diferentes**; todos os vizinhos carregam o mesmo atributo a:



$\{\!\{a,\ a\}\!\}$

$\{\!\{a\}\!\}$

AGG	v	v'	distingue?
média	a	a	não
máximo	a	a	não
soma	**2a**	**a**	**sim**

Média e máximo **descartam a multiplicidade** do multiset; a soma a preserva. É exatamente a injetividade que o teto 1-WL exige.

Visão Geral

Graph Transformers
Motivação[8]

Message passing local tem dois limites estruturais:

- Mensagens de longo alcance exigem muitas camadas ⇒ *oversmoothing*.
- *Bottlenecks* topológicos (*over-squashing*): metades do grafo se comunicam por poucas arestas.

Tudo entre as metades é espremido em uma aresta.

Ideia: atenção global, mas estruturada

Tratar cada nó como um *token* e aplicar *self-attention* sobre o grafo todo. Risco: perde-se a estrutura, é equivalente a um grafo completo.

Solução: injetar a estrutura via **positional encoding** ou **bias** de atenção.

[8] Ladislav Rampášek et al. "Recipe for a General, Powerful, Scalable Graph Transformer". Em: NeurIPS. 2022.

Positional Encodings em Grafos

Em sequências, PE(t) é função do índice t. Em grafos, não há índice canônico; precisamos de PEs invariantes à permutação:

- **Laplacian PE**: usar os k primeiros autovetores do Laplaciano $L = D - A$ como coordenadas espectrais. Capturam estrutura global do grafo.
- **Random Walk PE** (RWPE): probabilidade de estar em v após t passos partindo de v (1 a T passos).
- **Shortest-path bias**: adicionar termo na atenção proporcional à distância $d(u, v)$ no grafo.

GPS (Rampášek et al., 2022)

Combina **message passing local** + **atenção global** + **PE**, em camadas paralelas. Vence vários benchmarks OGB.

Geração de Grafos
GraphRNN[9]

Tarefa: aprender uma distribuição $p(G)$ e amostrar grafos novos (e.g., moléculas com propriedades desejadas).

Abordagem autorregressiva de (You et al., 2018)

1. Defina uma ordem dos nós (BFS a partir de um nó aleatório).
2. Para $i = 1, \ldots, n$:
   - Gere o nó v_i via uma RNN "nó-a-nó".
   - Para cada $j < i$, decida se a aresta (v_j, v_i) existe via uma segunda RNN "aresta-a-aresta".

- Vantagem: trata a geração como sequência, reaproveita arquitetura RNN.
- Desafio: a ordenação introduz viés. Modelos mais recentes (e.g., difusão sobre grafos) tentam contornar isso.

[9] Jiaxuan You et al. "GraphRNN: Generating Realistic Graphs with Deep Auto-Regressive Models". Em: ICML. 2018.

LLMs + Grafos
Tendência Atual

Linha de pesquisa em forte expansão (2023–2026):

- **Text-attributed graphs**: nós têm conteúdo textual (artigos, descrições de produto). LLM produz embeddings, GNN agrega via grafo.
- **GraphRAG**: extrair grafos de conhecimento de coleções de documentos para alimentar *retrieval-augmented generation*.
- **Reasoning sobre grafos com LLMs**: descrever o grafo em linguagem natural ou via tokens estruturais e deixar o LLM "raciocinar" sobre conexões.
- **Foundation models para grafos**: pré-treinar uma única GNN em muitos domínios e adaptar via *fine-tuning* ou *in-context learning*.

O curso Stanford CS224W dedica cerca de 3 aulas inteiras a esse tópico.

Aplicações em Destaque

- **AlphaFold** (Jumper et al., 2021): proteínas como grafos com message passing especializado; referência mundial em biologia computacional.
- **Descoberta de fármacos**: `ogbg-molhiv` e variantes, screening em escala industrial.
- **Recomendação**: PinSage (Pinterest), grafos usuário–item com bilhões de arestas.
- **Detecção de fraude**: redes de transações como grafos, detecção de comunidades suspeitas.

Onde aprofundar

- **Stanford CS224W** (Leskovec): curso semestral, vídeos públicos no YouTube.
- **OGB**: datasets, leaderboards e código de referência.
- **PyTorch Geometric (PyG)** e **DGL**: bibliotecas de implementação.
- **Hamilton**, Graph Representation Learning (livro 2020).

Resumo

- Grafos são domínio não-euclidiano sem ordem canônica; **permutação** é a simetria central.
- Tarefas: **node / link / graph classification**; transdutivo vs indutivo muda o que o modelo precisa.
- **OGB** é o benchmark padrão; **Hits@K** e **MRR** são as métricas específicas para link prediction.
- Quase toda GNN é uma instância de **message passing**: M, AGG, U.
- Modelos canônicos: **GCN** (média normalizada), **GraphSAGE** (indutivo + amostragem), **GAT** (atenção sobre vizinhos).
- **Oversmoothing** limita profundidade; mitigações: skip, JK, DropEdge, atenção global.
- **GIN** alcança o teto do teste 1-WL; soma + MLP é o que dá expressividade.
- **Graph Transformers** e **LLMs + grafos** são as fronteiras ativas atuais.

Leituras Recomendadas

- **Curso**: Stanford CS224W, *Machine Learning with Graphs* (Jure Leskovec). Slides e vídeos em `web.stanford.edu/class/cs224w`.
- **Livro**: Hamilton, *Graph Representation Learning* (2020), introdução acessível e abrangente.
- **Artigos seminais** (em ordem sugerida): GCN (Kipf e Welling, 2017); GraphSAGE (Hamilton, Ying e Leskovec, 2017); GAT (Veličković et al., 2018); MPNN (Gilmer et al., 2017); GIN (Xu et al., 2019).
- **Benchmark**: OGB (Hu et al., 2020), em `ogb.stanford.edu`.
- **Bibliotecas**: *PyTorch Geometric* (PyG) e *Deep Graph Library* (DGL).

Referências I

[1] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. Em: ICML. 2017, pp. 1263–1272.

[2] Aditya Grover e Jure Leskovec. “node2vec: Scalable Feature Learning for Networks”. Em: KDD. 2016, pp. 855–864.

[3] William L. Hamilton, Rex Ying e Jure Leskovec. “Inductive Representation Learning on Large Graphs”. Em: NeurIPS. 2017.

[4] Weihua Hu et al. “Open Graph Benchmark: Datasets for Machine Learning on Graphs”. Em: NeurIPS. 2020.

[5] John Jumper et al. “Highly Accurate Protein Structure Prediction with AlphaFold”. Em: Nature 596 (2021), pp. 583–589.

[6] Thomas N. Kipf e Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. Em: ICLR. 2017.

[7] Bryan Perozzi, Rami Al-Rfou e Steven Skiena. “DeepWalk: Online Learning of Social Representations”. Em: KDD. 2014, pp. 701–710.

[8] Ladislav Rampášek et al. “Recipe for a General, Powerful, Scalable Graph Transformer”. Em: NeurIPS. 2022.

[9] Petar Veličković et al. “Graph Attention Networks”. Em: ICLR. 2018.

[10] Keyulu Xu et al. “How Powerful are Graph Neural Networks?” Em: ICLR. 2019.

[11] Jiaxuan You et al. “GraphRNN: Generating Realistic Graphs with Deep Auto-Regressive Models”. Em: ICML. 2018.