Ciência das Redes

Redes Neurais de Grafos

Thiago H Silva

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Node embeddings

Limitações dos métodos de shallow encoders (como o node2vec):

- Não incorpora *features* de nó:
  - Os nós em muitos gráficos têm recursos que podemos e devemos usar

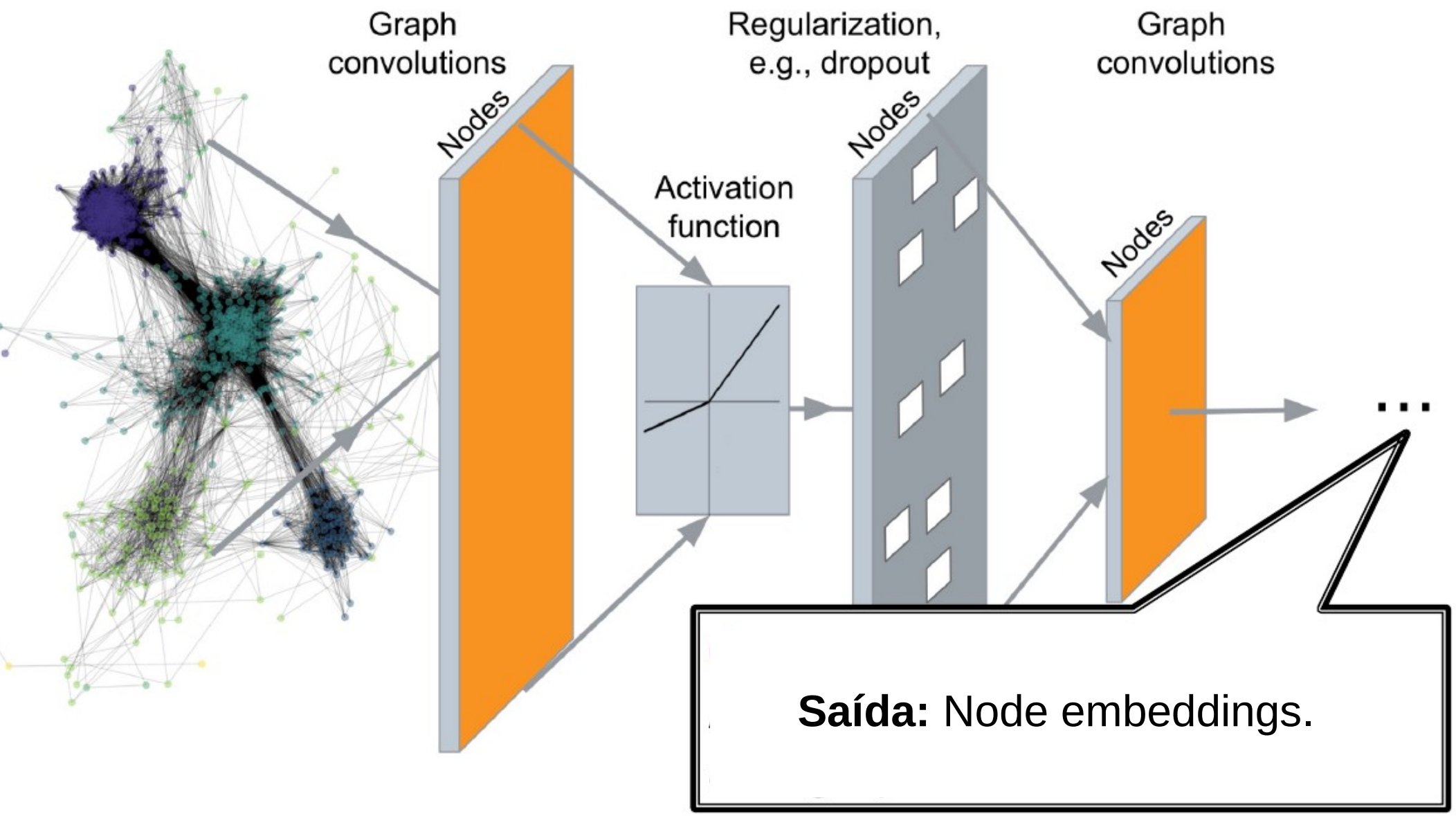
Deep graph encoders

Podemos usar métodos de aprendizagem profunda com base em redes neurais de grafos (GNNs):

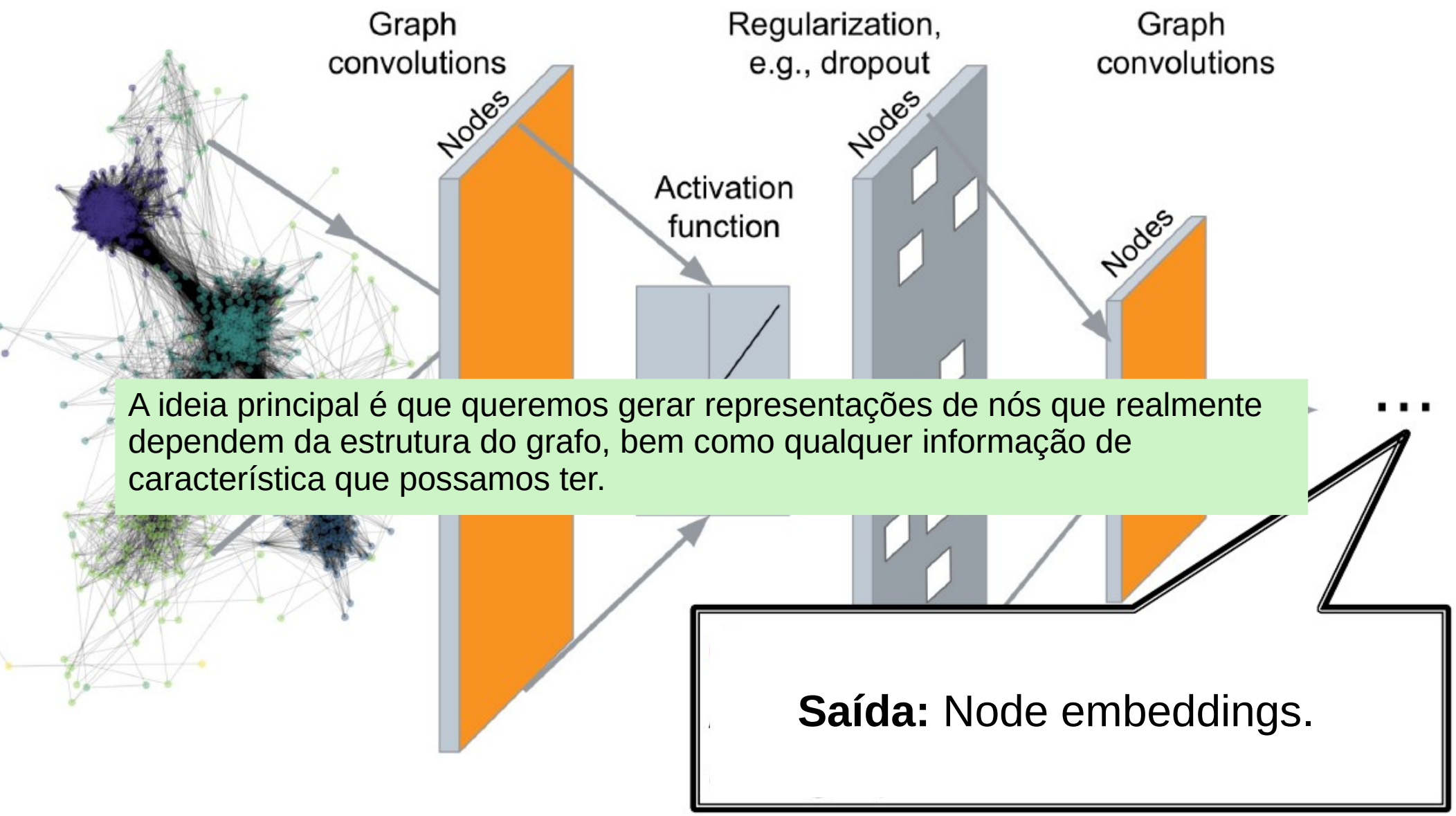
$$\mathrm{ENC}(v) =$$

Múltiplas camadas de transformações não lineares baseadas na estrutura do grafo

Deep graph encoders



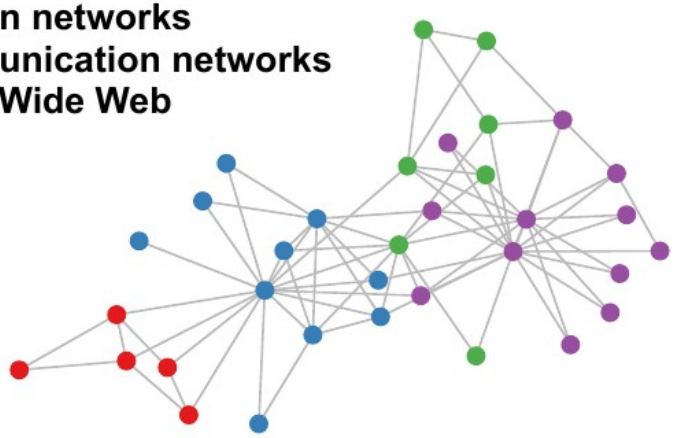
Deep graph encoders

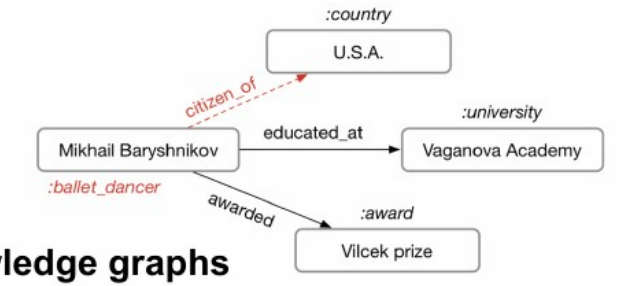


A ideia principal é que queremos gerar representações de nós que realmente dependem da estrutura do grafo, bem como qualquer informação de característica que possamos ter.

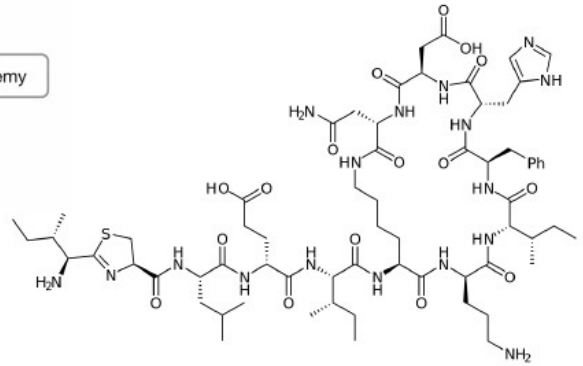
Graph Neural Networks

Social networks
Citation networks
Communication networks
World Wide Web

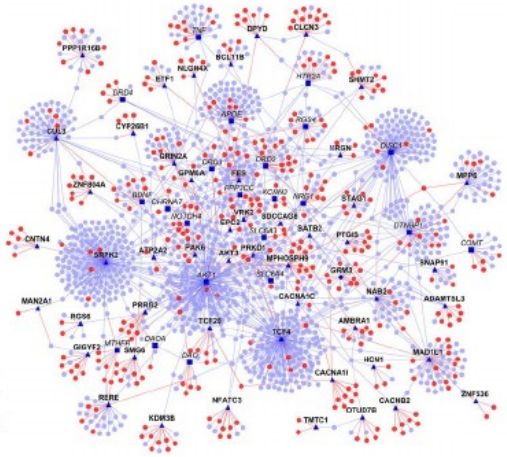




Knowledge graphs



Molecules

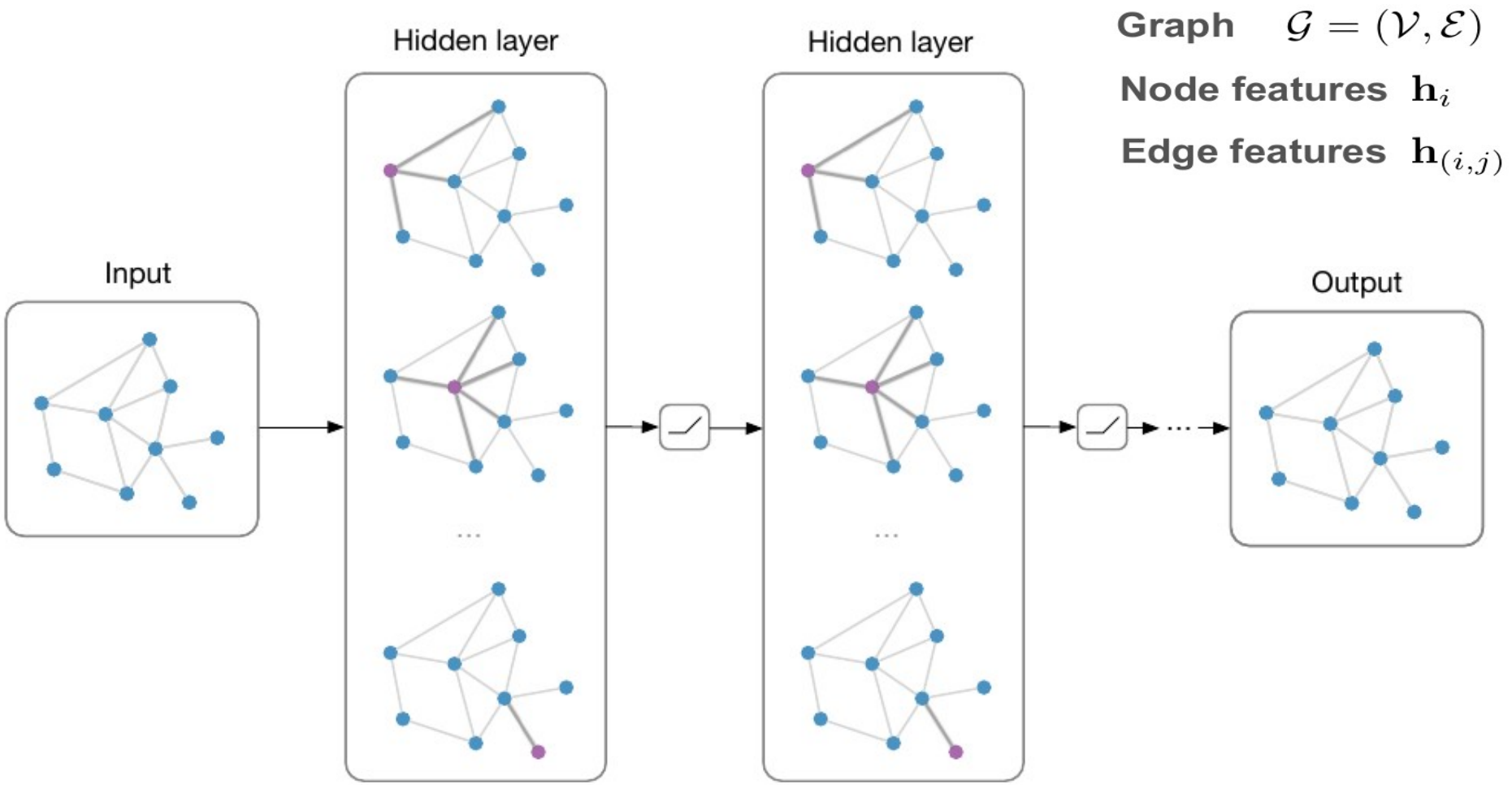


Protein interaction networks

Road maps

Challenging for standard deep neural net architectures (CNNs / RNNs)

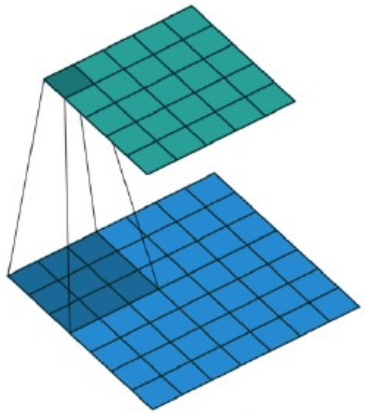
Ideia geral

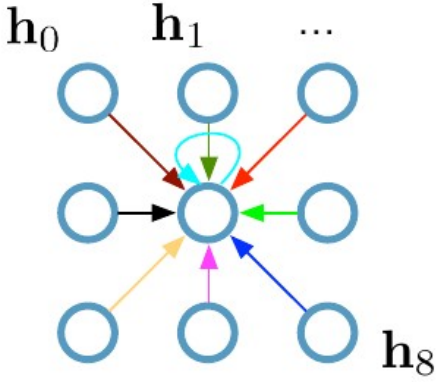


Ideia principal: transmitir mensagens ao longo das arestas do grafo, aglomerar e transformar

CNNs (em grades) como transmissora de mensagens

Single CNN layer with 3x3 filter:





Update for a single pixel:

- Transform messages individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

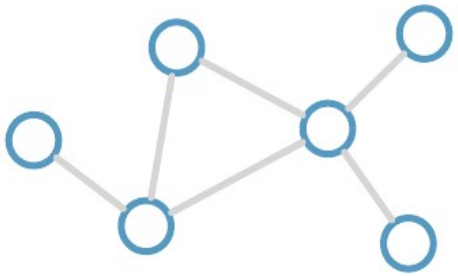
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Full update:

$$\mathbf{h}'_4 = \sigma(\mathbf{W}_0\mathbf{h}_0 + \mathbf{W}_1\mathbf{h}_1 + \cdots + \mathbf{W}_8\mathbf{h}_8)$$

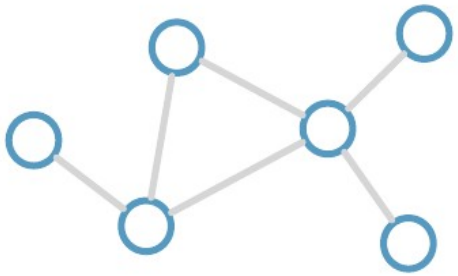
Pytorch Geometric (PyG)

Considere este grafo
não direcionado

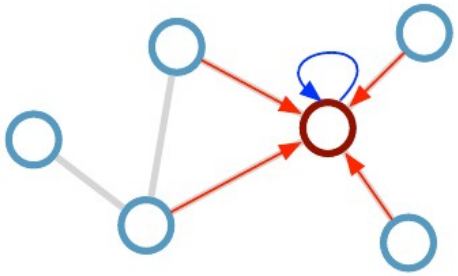


Pytorch Geometric (PyG)

Considere este grafo não direcionado



Calcular a atualização para o nó em vermelho



Update rule:

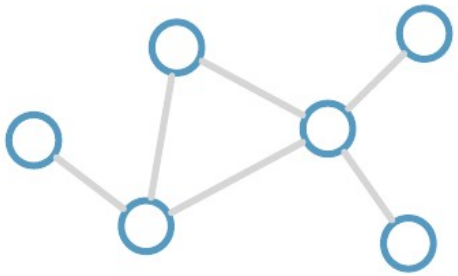
$$\mathbf{h}_i' = \sigma\left(\mathbf{W}_0\mathbf{h}_i + \sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}_1\mathbf{h}_j\right)$$

Índices dos vizinhos

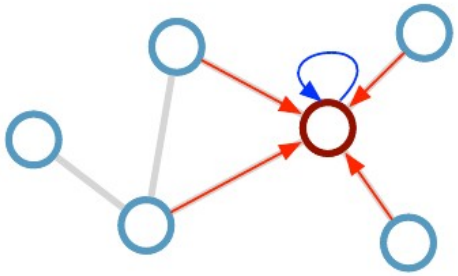
Constante de normalização

Pytorch Geometric (PyG)

Considere este grafo não direcionado



Calcular a atualização para o nó em vermelho



Update rule:

$$\mathbf{h}_i' = \sigma\left(\mathbf{W}_0\mathbf{h}_i + \sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}_1\mathbf{h}_j\right)$$

Índices dos vizinhos

Constante de normalização

Propriedades desejáveis:

- Compartilhamento de pesos em todas as localidades
- Complexidade linear O(E)
- Aplicáveis em cenários indutivos (novos nós e arestas)

Limitações

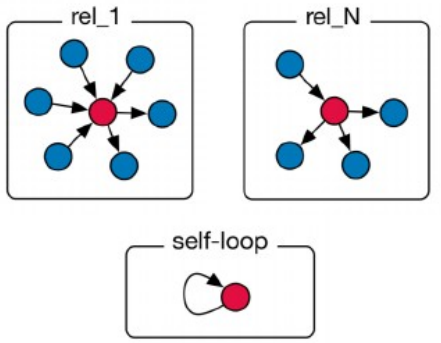
- Não suporta a interação entre pares
- Não suporta features de arestas

GNNs mais expressivas

MoNet & Relational GCN for modeling relational data

Monti et al. (CVPR 2017), Schlichtkrull & Kipf et al. (arXiv 2017, ESWC 2018)

$$\mathbf{h}_i' = \sigma\left(\sum_{r=1}^{R}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^r\mathbf{W}_r\mathbf{h}_j\right)$$



α_{ij}^r **based on:**

- Edge type (Relational GCN)
- Auxiliary features (MoNet), e.g. node degree

Self-attention and Graph Attention Networks

Vaswani et al. (NIPS 2017), Veličković et al. (ICLR 2018)

$$\alpha_{ij}^r = \frac{\exp(\mathbf{h}_i^T\mathbf{W}_r'\mathbf{h}_j)}{\sum_{k\in\mathcal{N}_i}\exp(\mathbf{h}_i^T\mathbf{W}_r'\mathbf{h}_k)}$$

Multi-head dot product attention:

- Activation-dependent α_{ij}^r
- Bi-linear scoring

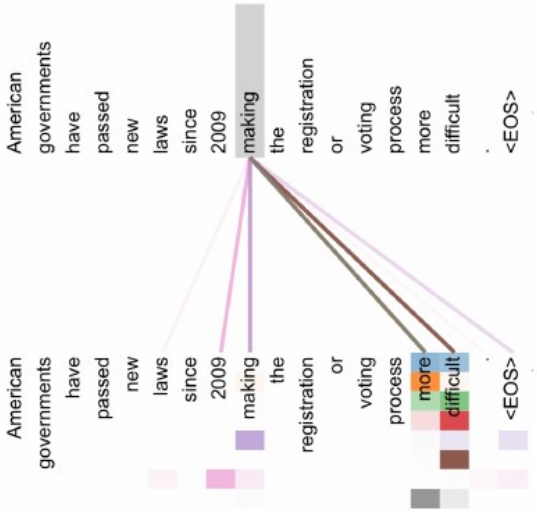
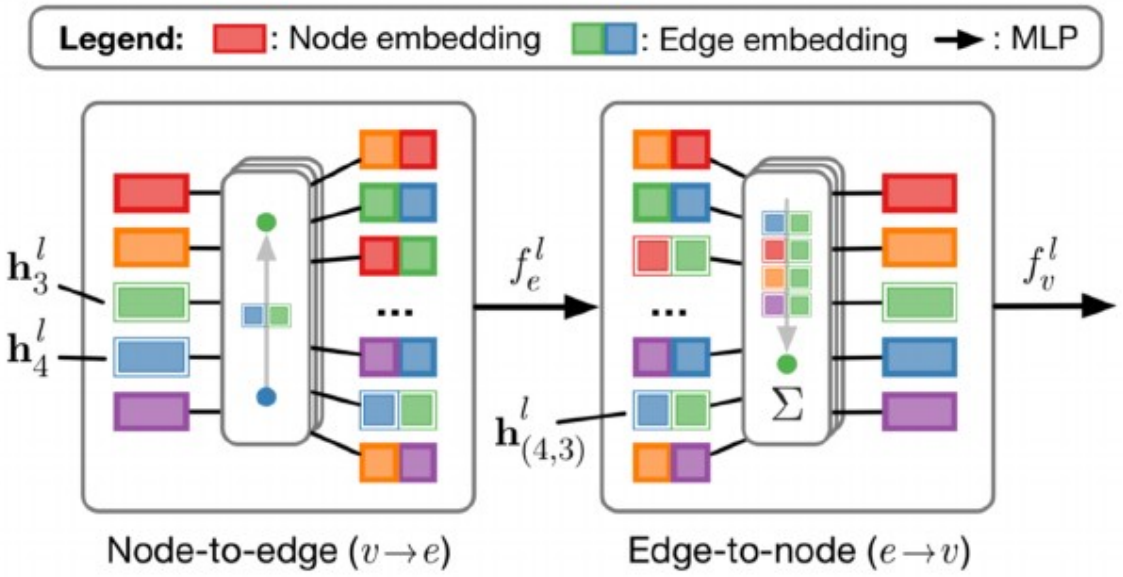


Figure from Vaswani et al. (NIPS 2017)

GNNs com Edge Embeddings



Formally:

$$v \to e: \quad \mathbf{h}_{(i,j)}^l = f_e^l([\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{x}_{(i,j)}])$$

$$e \to v: \quad \mathbf{h}_j^{l+1} = f_v^l([\textstyle\sum_{i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^l, \mathbf{x}_j])$$

Pros:

- Suporte a features de arestas
- Mais expressivas do que GCN
- Bastante flexível

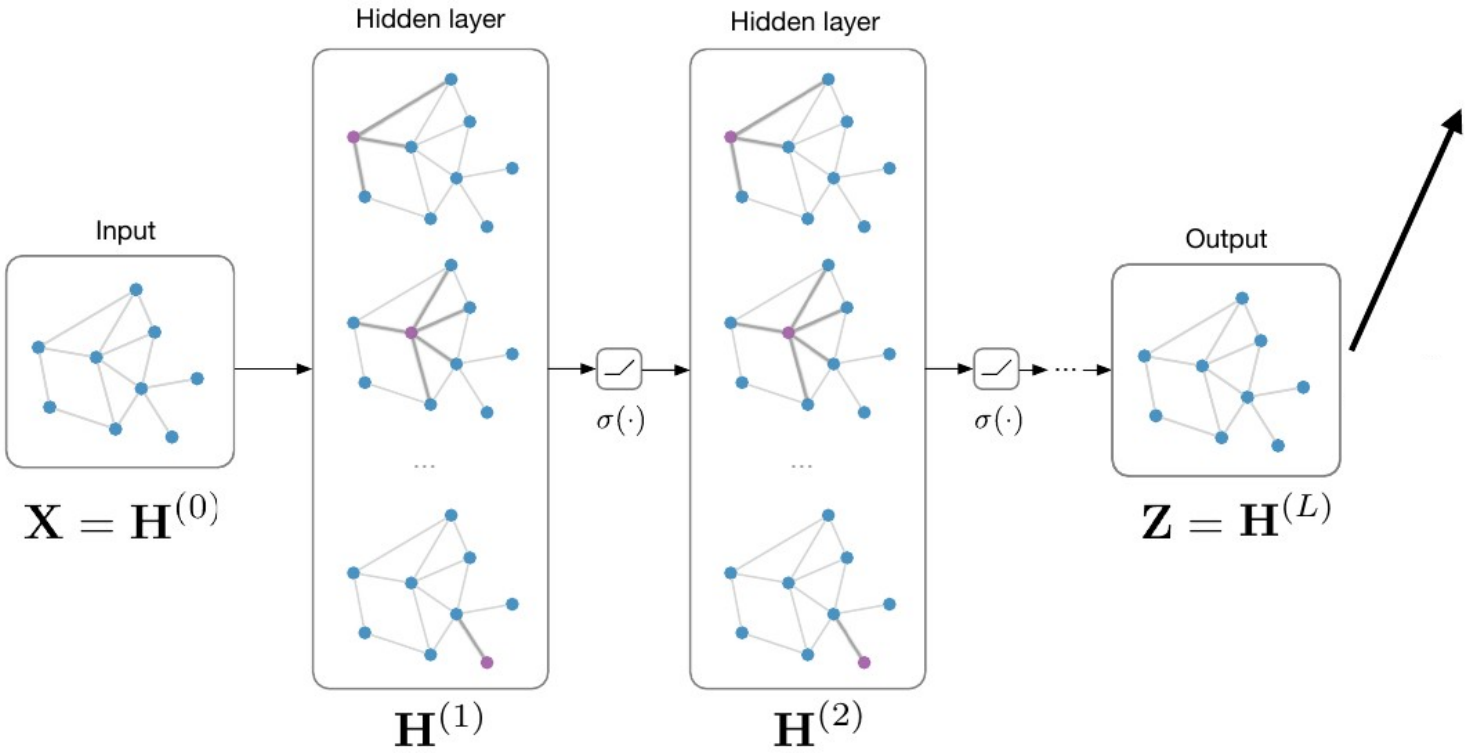
Cons:

- Precisa armazenar ativações baseadas em arestas intermediárias
- Na prática mais lentas do que GCN / Self-attention models

Battaglia et al. (NIPS 2016), Gilmer et al. (ICML 2017), Kipf et al. (ICML 2018)

Classificação e Predição de links com GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$, graph adjacency matrix $\mathbf{A}$



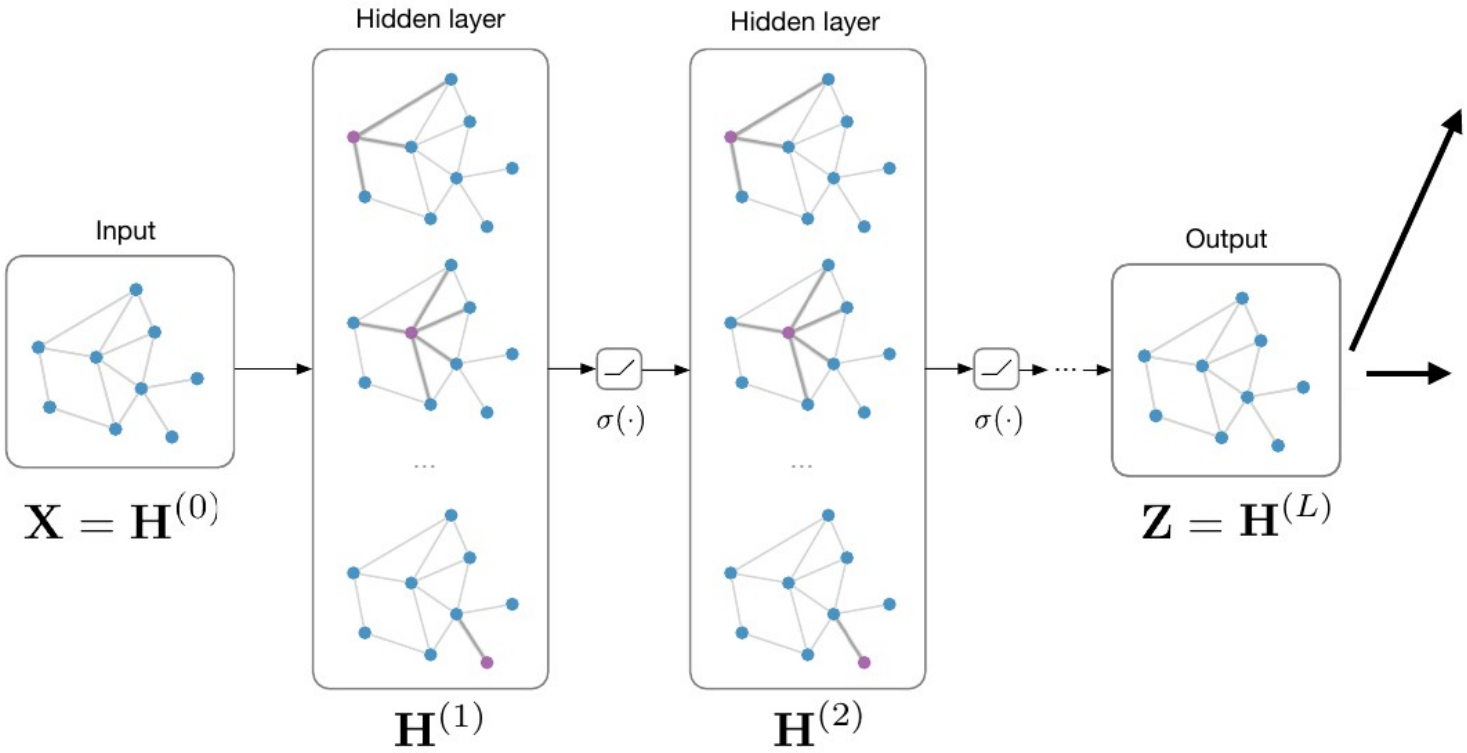
Node classification:

$\mathrm{softmax}(\mathbf{z_n})$

e.g. Kipf & Welling (ICLR 2017)

Classificação e Predição de links com GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$, graph adjacency matrix $\mathbf{A}$



Node classification:

$\mathrm{softmax}(\mathbf{z_n})$

e.g. Kipf & Welling (ICLR 2017)

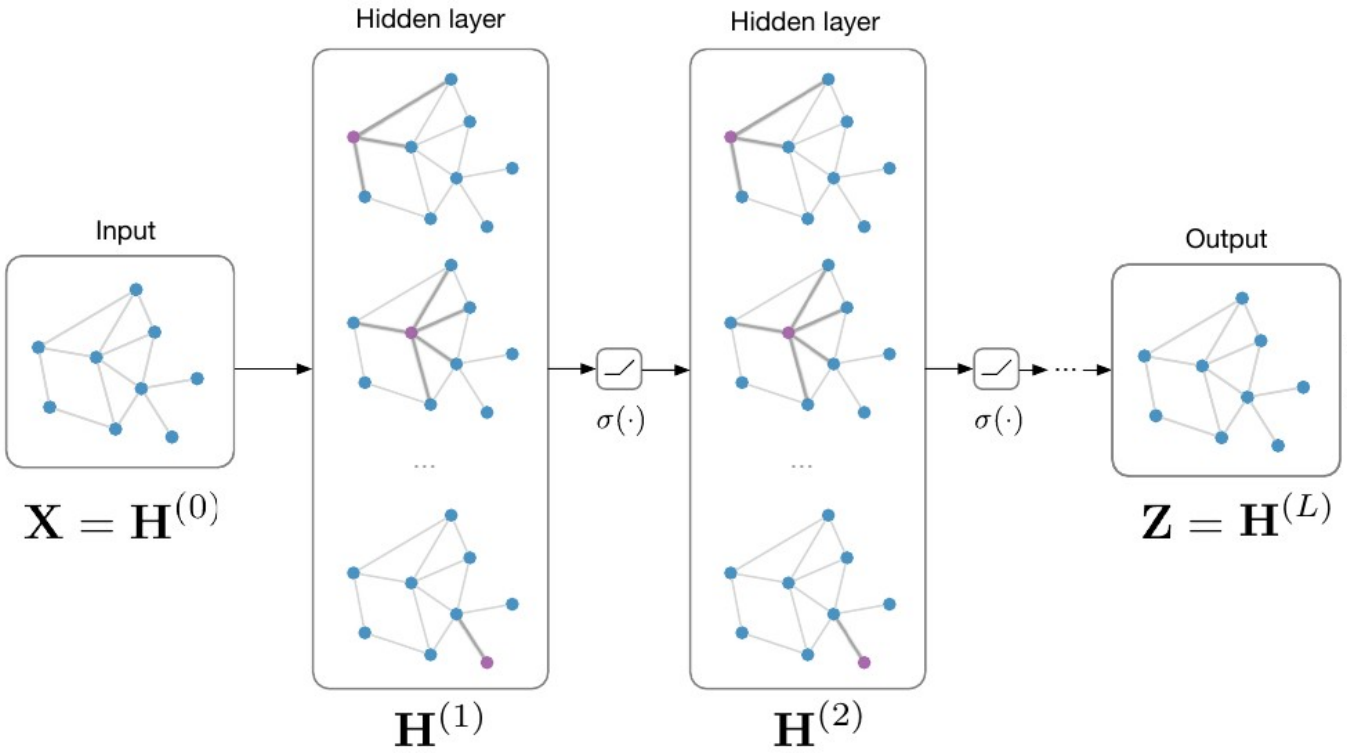
Graph classification:

$\mathrm{softmax}(\sum_n \mathbf{z_n})$

e.g. Duvenaud et al. (NIPS 2015)

Classificação e Predição de links com GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$, graph adjacency matrix $\mathbf{A}$



Node classification:

$\mathrm{softmax}(\mathbf{z_n})$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$\mathrm{softmax}(\sum_n \mathbf{z_n})$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

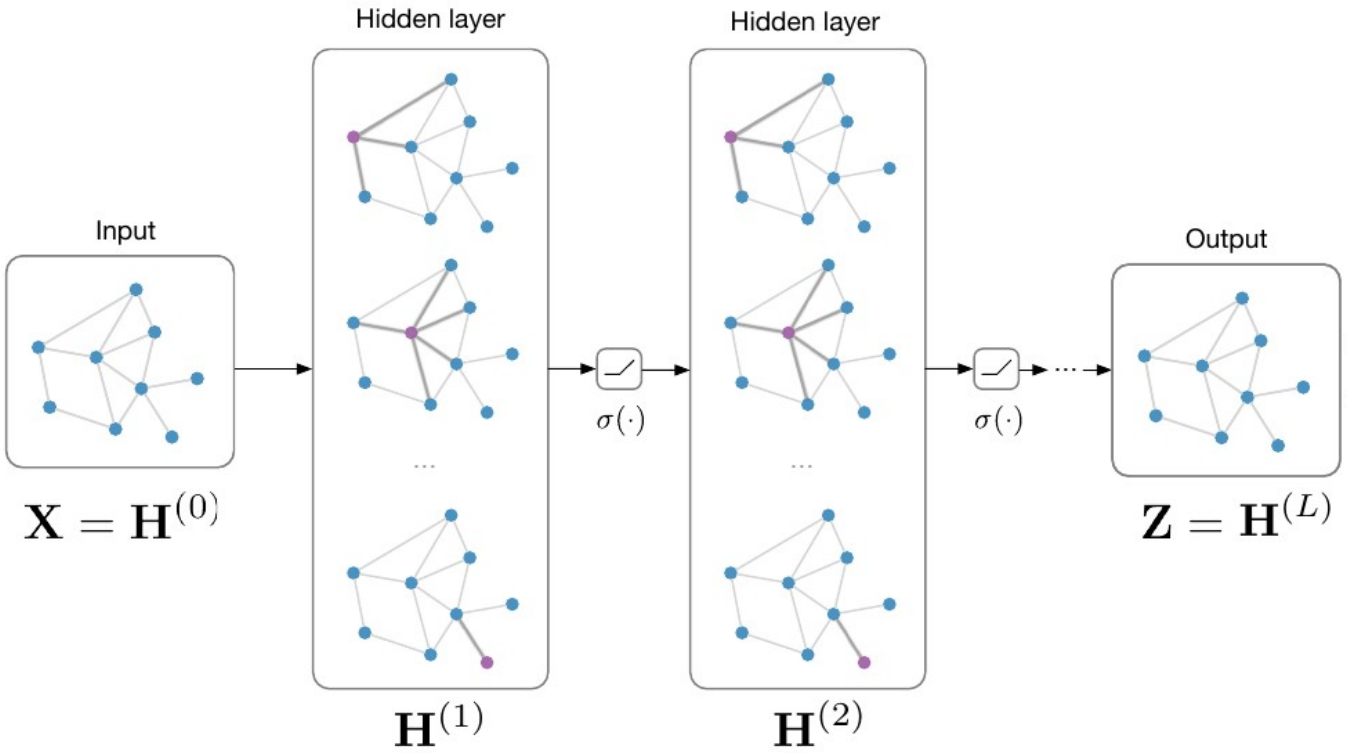
$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$

Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

Classificação e Predição de links com GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$, graph adjacency matrix $\mathbf{A}$



Node classification:

$\mathrm{softmax}(\mathbf{z_n})$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$\mathrm{softmax}(\sum_n \mathbf{z_n})$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

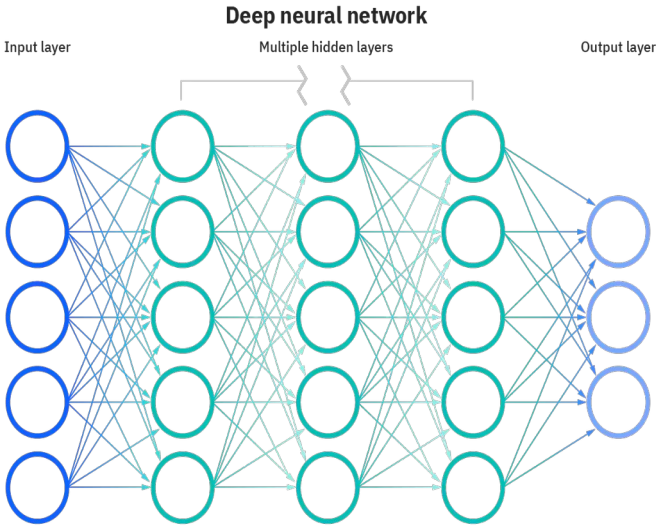
$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$

Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

AutoEncoders

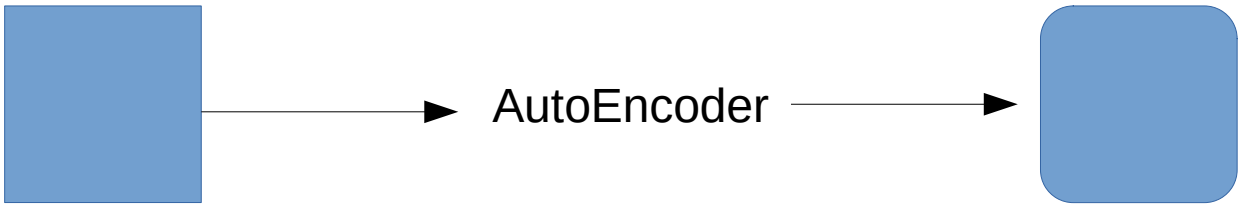
- Relembrando – O que as redes neurais profundas fazem?



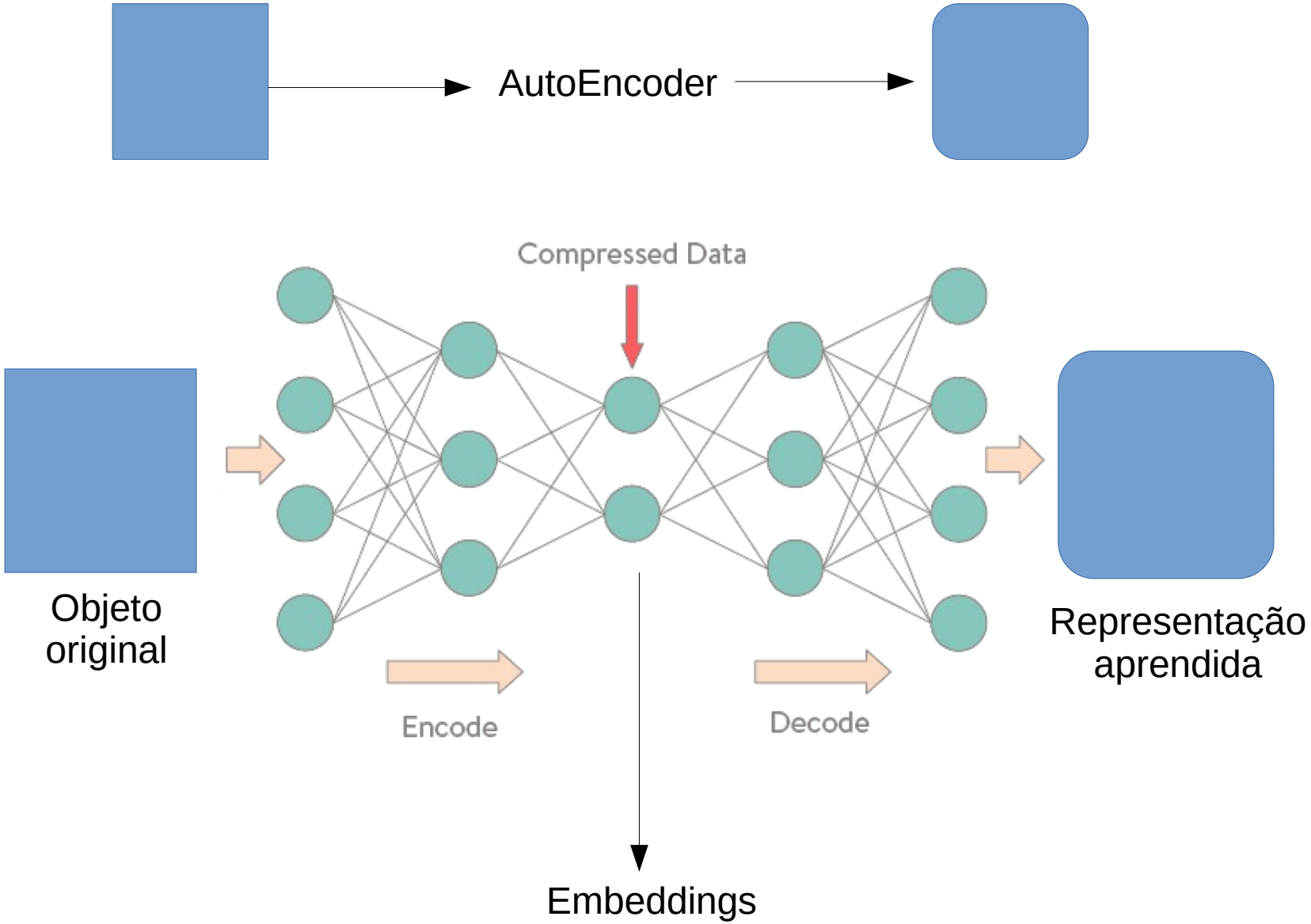
- Aprendem *features* importantes do *input*
  - Permitem fazer tarefas variadas
- Usar esse recurso para transformar o input (codificar - encode)

AutoEncoders

- AutoEcoders são redes neurais que trabalham no regime **não supervisionado**
- Ou seja não precisamos de rótulos

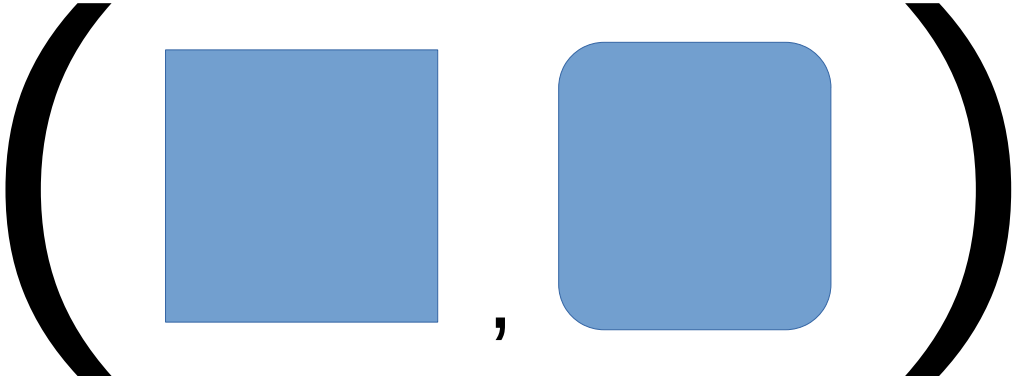


AutoEncoders

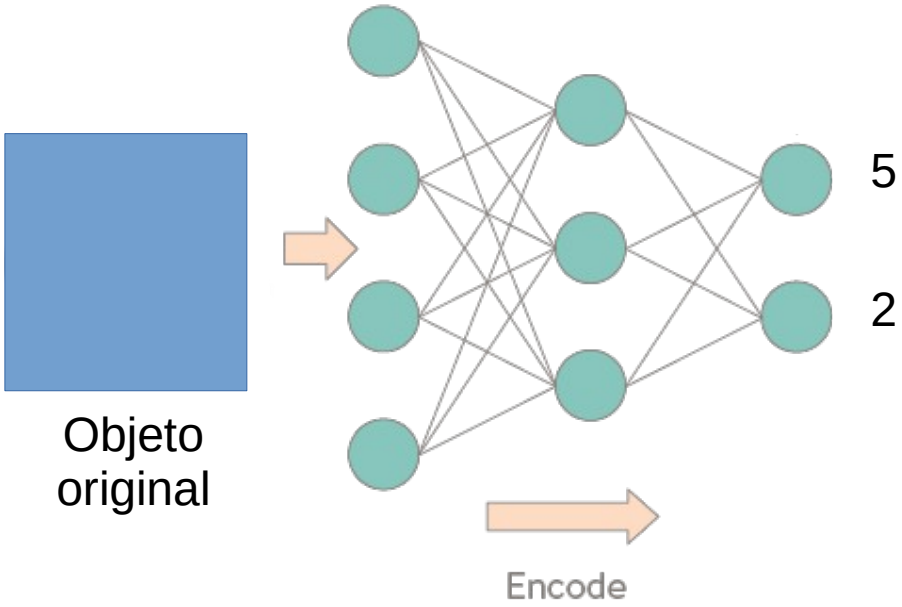


AutoEncoders

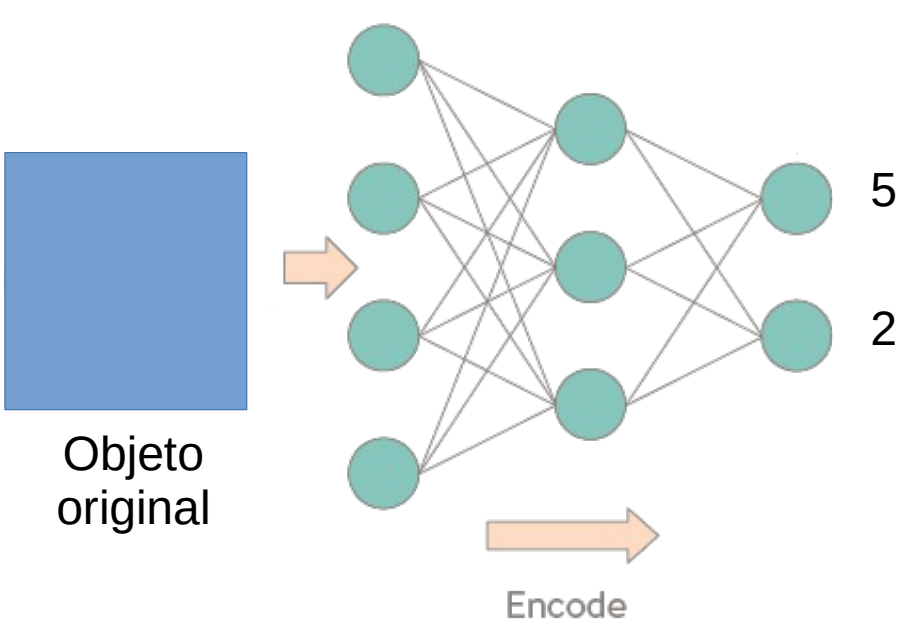
Erro = similaridade

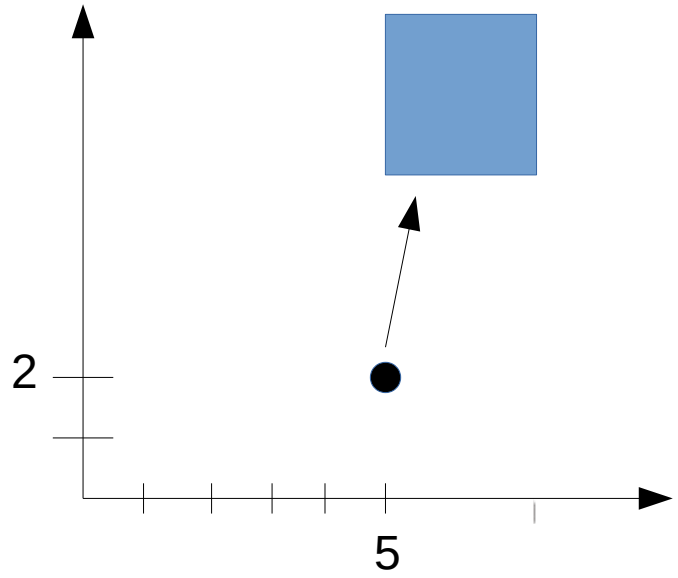


AutoEncoders

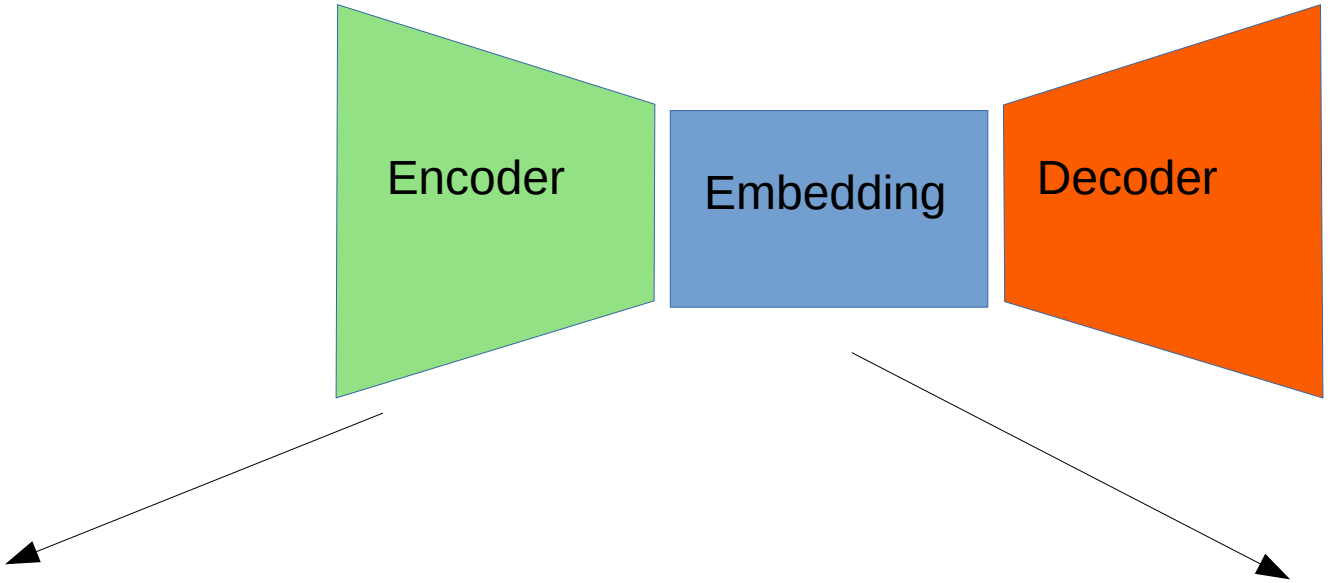


AutoEncoders





Graph AutoEncoders (GAE)



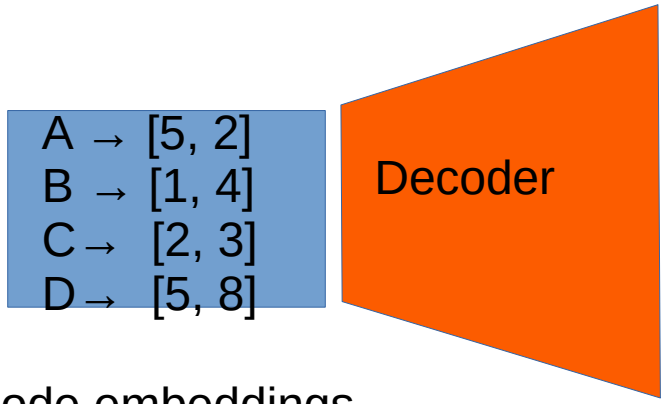
Necessário uma rede convolucional de grafo (graph convolution network)

Que produz uma representação de *embeddings* em baixa dimensão

Saída da rede neural de grafo

Graph AutoEncoders (GAE)

Reconstrói o
grafo de entrada



Node embeddings
com duas dimensões

Produto interno (Inner product)

$Adj_{(A,B)} = sigmoid([5,2] * [1,4]^T)$

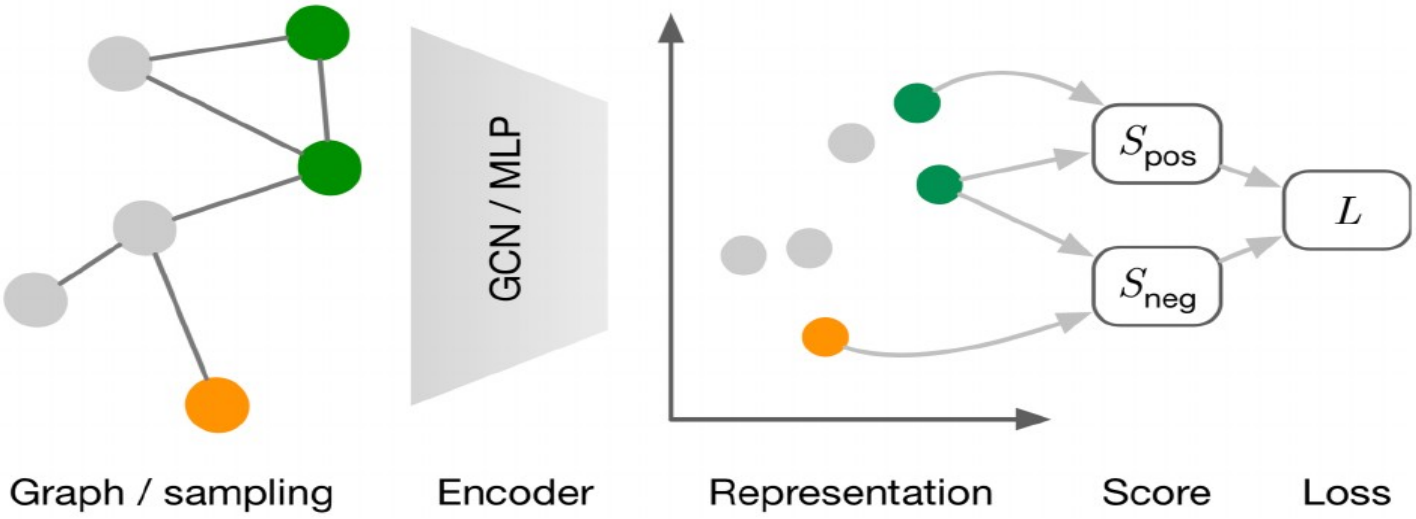
$Adj_{(A,C)} = sigmoid([5,2] * [2,3]^T)$

...

Usado por “Variational Graph Auto-Encoders” 2016

Graph AutoEncoders (GAE)

- Também pode ser usada para classificação de nó:
  - Cenário: alguns nós possuem labels
  - Objetivo: predizer os labels de nós que não possuem labels



Graph AutoEncoders (GAE)

CLASS GAE (**encoder**, **decoder**=None) [source]

The Graph Auto-Encoder model from the "Variational Graph Auto-Encoders" paper based on user-defined encoder and decoder models.

PARAMETERS:

- **encoder** (*Module*) – The encoder module.
- **decoder** (*Module*, *optional*) – The decoder module. If set to `None`, will default to the `torch_geometric.nn.models.InnerProductDecoder`. (default: `None`)

reset_parameters ( ) [source]

encode (***args**, ****kwargs**) [source]

Runs the encoder and computes node-wise latent variables.

decode (***args**, ****kwargs**) [source]

Runs the decoder and computes edge probabilities.

recon_loss (**z**, **pos_edge_index**, **neg_edge_index**=None) [source]

Given latent variables `z`, computes the binary cross entropy loss for positive edges `pos_edge_index` and negative sampled edges.

PARAMETERS:

- **z** (*Tensor*) – The latent space $\mathbf{Z}$.
- **pos_edge_index** (*LongTensor*) – The positive edges to train against.
- **neg_edge_index** (*LongTensor*, *optional*) – The negative edges to train against. If not given, uses negative sampling to calculate negative edges. (default: `None`)

Demonstração usando a implementação do PyG (Pytorch geometric)

Outras abordagens para aprendizado não supervisionado

Table 1. Components of unsupervised learning methods on graphs in existing algorithms: DeepWalk (Perozzi et al., 2014), GAE (Kipf & Welling, 2016b), $\mathcal{S}$-VGAE (Davidson et al., 2018), DGI (Veličković et al., 2018), and G2G (Bojchevski & Günnemann, 2017).

Method	Encoder	Representation	Score	Loss	Sampling
DeepWalk	LUT	$\mathbf{z}_i \in \mathbb{R}^D$	$\sigma(\mathbf{z}_i^\top \mathbf{z}_j)$	$-\log S - \log(1-\tilde{S})$	(+) random walk neighbors (-) non-neighbors
GAE	GCN	$\mathbf{z}_i \in \mathbb{R}^D$	$\sigma(\mathbf{z}_i^\top \mathbf{z}_j)$	$-\log S - \log(1-\tilde{S})$	(+) 1st order neighbors (-) non-neighbors
$\mathcal{S}$-VGAE	GCN	$\mathbf{z}_i \sim \text{vMF}(\mathbf{z})$	$\sigma(\mathbf{z}_i^\top \mathbf{z}_j)$	$-\log S - \log(1-\tilde{S})$	(+) 1st order neighbors (-) non-neighbors
DGI	GCN	$\mathbf{z}_i \in \mathbb{R}^D$	$\sigma\left(\mathbf{z}_i^T \mathbf{W} \mathbf{s}\right)$	$-\log S - \log(1-\tilde{S})$	(+) original graph (-) corrupted graph
G2G	MLP	$\mathbf{z}_\mu \in \mathbb{R}^D$ $\mathbf{z}_\Sigma \in \mathbb{R}^D$	$\exp(-\text{KL}(\mathcal{N}_i \| \mathcal{N}_j))$	$(\log S)^2 + \tilde{S}$	(+) 1st order neighbors (-) higher order neighbors

GAE: Graph Auto-Encoders, G2G: Graph2Gauss, DGI: Deep Graph Infomax $\mathbf{s} = \sum_{i=1}^{N} \mathbf{z}_i$

Pytorch Geometric (PyG)

Um grafo em PyG é descrito por uma instância de *torch_geometric.data.Data*, que contém os seguintes atributos por padrão:

- data.x: Node feature matrix with shape [num_nodes, num_node_features]
- data.edge_index: Graph connectivity in COO format with shape [2, num_edges] and type torch.long
- data.edge_attr: Edge feature matrix with shape [num_edges, num_edge_features]
- data.y: Target to train against (may have arbitrary shape), e.g., node-level targets of shape [num_nodes, *] or graph-level targets of shape [1, *]
- data.pos: Node position matrix with shape [num_nodes, num_dimensions]

Nenhum desses atributos é obrigatório.

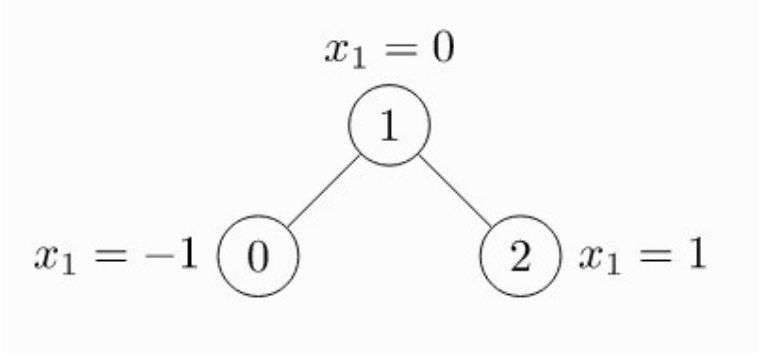
O objeto Data não está restrito a esses atributos.

Pytorch Geometric (PyG)

```python
import torch
from torch_geometric.data import Data

edge_index = torch.tensor([[0, 1, 1, 2],
                           [1, 0, 2, 1]], dtype=torch.long)
x = torch.tensor([[-1], [0], [1]], dtype=torch.float)

data = Data(x=x, edge_index=edge_index)
```



Quatro arestas definidas

Embora o gráfico tenha apenas duas arestas, precisamos definir quatro tuplas de índice para explicar as duas direções de uma aresta.

Pytorch Geometric (PyG)

Data
DataLoader

Classes que organizam e facilitam o trabalho com um dataset de grafos

Pytorch Geometric (PyG)

- Mini-batches

  Oferece recursos para facilitar o trabalho com mini-batches

- Recursos para gerenciar treinamento e teste (mesmo que com um grafo apenas)

Graph Neural Networks

Área nova e difícil de acompanhar

Inúmeras aplicações e sucesso

Ainda muitos desafios

Agradecimentos e referências

Alguns slides foram derivados de Thomas Kipf – University of Amsterdam

Paper “Variational Graph Auto-Encoders” 2016

Biblioteca oficial do Pytorch Geometric