Day 3 - Graph neural networks and Transformers

May 30, 2024

Darius Schaub

Graph neural networks (GNNs)

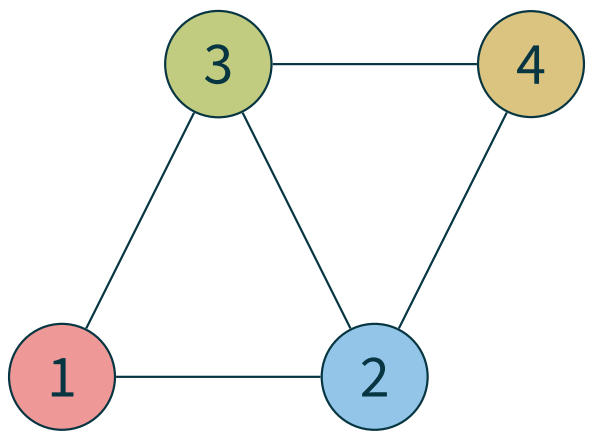
What are graphs?

We define a graph as a mathematical object $G(V, E, X, D)$***, where***

- $V = \{1, \ldots, n\}$ is the set of nodes (node indices).
- $E \subset \{(i, j) \| i, j \in V\}$ is the set of edges, connecting the nodes.
- $X \in \mathbb{R}^{n \times d}$ are node features attached to each node.
- $E \in \mathbb{R}^{|E| \times p}$ are edge features attached to each edge.

For the adjacency matrix A of G it holds $a_{ij} = 1$ if $(i, j) \in E$ and $a_{ij} = 0$ otherwise.

Note: the numbering of the nodes inside the graph is arbitrary.



Graphs in practice

- Knowledge graphs: this includes gene-regulatory networks, protein-protein interaction networks, pathways, customer-product relationships, etc.
- Molecular graphs: graph representations of molecules (e.g., drugs, DNA, RNA, proteins) can capture and represent interactions between different atoms or sub-molecules
- Spatial graphs: this can include train connections, road networks, brain circuits, **cellular networks**, and more. Google Maps uses this representation and even utilizes GNNs to find the best connections.

Graph neural networks (GNNs)

GNNs are currently our best answer to the question of how to represent a function that maps from a space of possible graphs to some desired output space

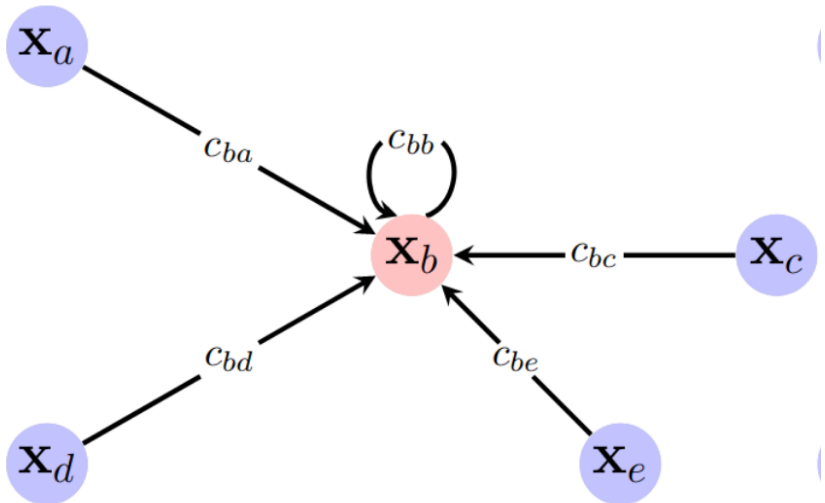
Recently also a geometric rationale for this architecture (and others) has been established, by requiring that the function is permutation equivariant/invariant

$$f_\theta : \mathcal{G} \to \mathcal{H}$$

Three different "flavors" of GNNs

1. Convolutional GNNs
2. Attentional GNNs
3. Message-passing GNNs

Convolutional GNNs



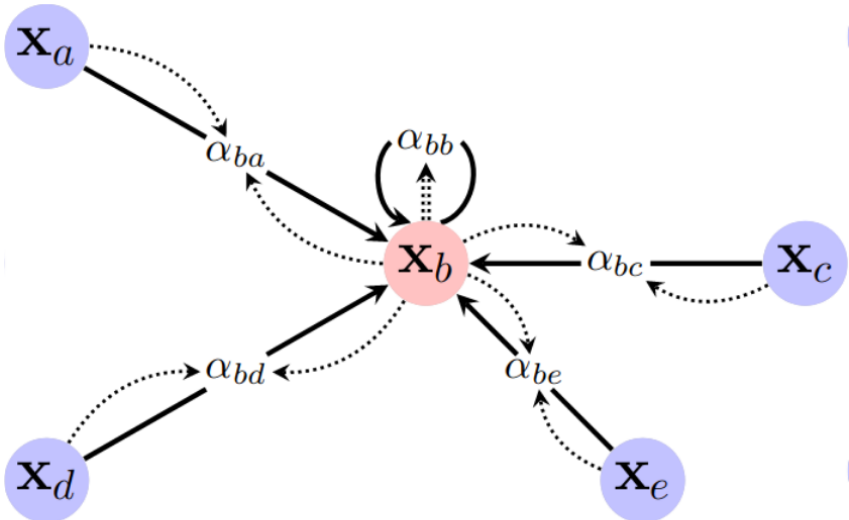
Convolutional

$$h_v^{i+1} = \phi\left(h_v, \bigoplus_{u \in \mathcal{N}(v)} c_{uv}\psi(h_u)\right)$$

Usually we can express this as a series of matrix products:

$$H^{i+1} = \sigma(C(A)H^i\Theta^i) \in \mathbb{R}^{n \times l}$$

Attentional GNNs

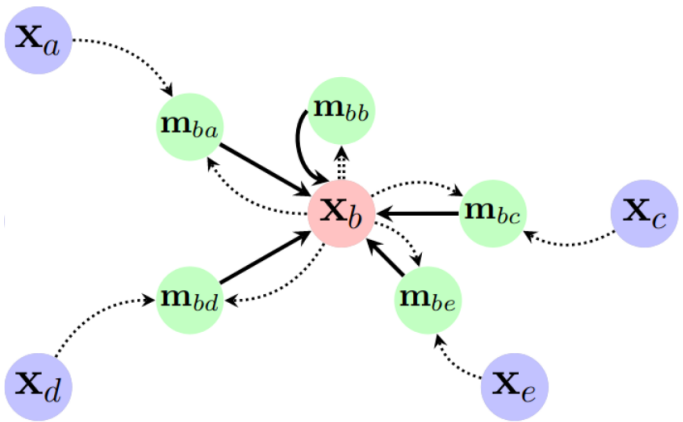


Attentional

$$h_v^{i+1} = \phi\left(h_v, \bigoplus_{u \in \mathcal{N}(v)} a(h_u, h_v, e_{uv})\psi(h_u)\right)$$

As we will see this architecture is closely related to the well-known Transformer architecture.

Message-passing GNNs



Message-passing

$$h_v^{i+1} = \phi\left(h_v, \bigoplus_{u \in \mathcal{N}(v)} \psi(h_u, h_v, e_{uv})\right)$$

This is the most expressive GNN architecture in terms of possible functions it can represent. The expressiveness is usually characterized through the ability to distinguish certain graphs.

GNNs in practice

- Drug discovery: molecular property prediction, drug-target docking/interaction prediction
- Spatial organization of cells: **spatial domain identification**, imputation, cell-interaction analysis
- Industry: predicting favorable customer product relationships

GNN hands-on

Overview

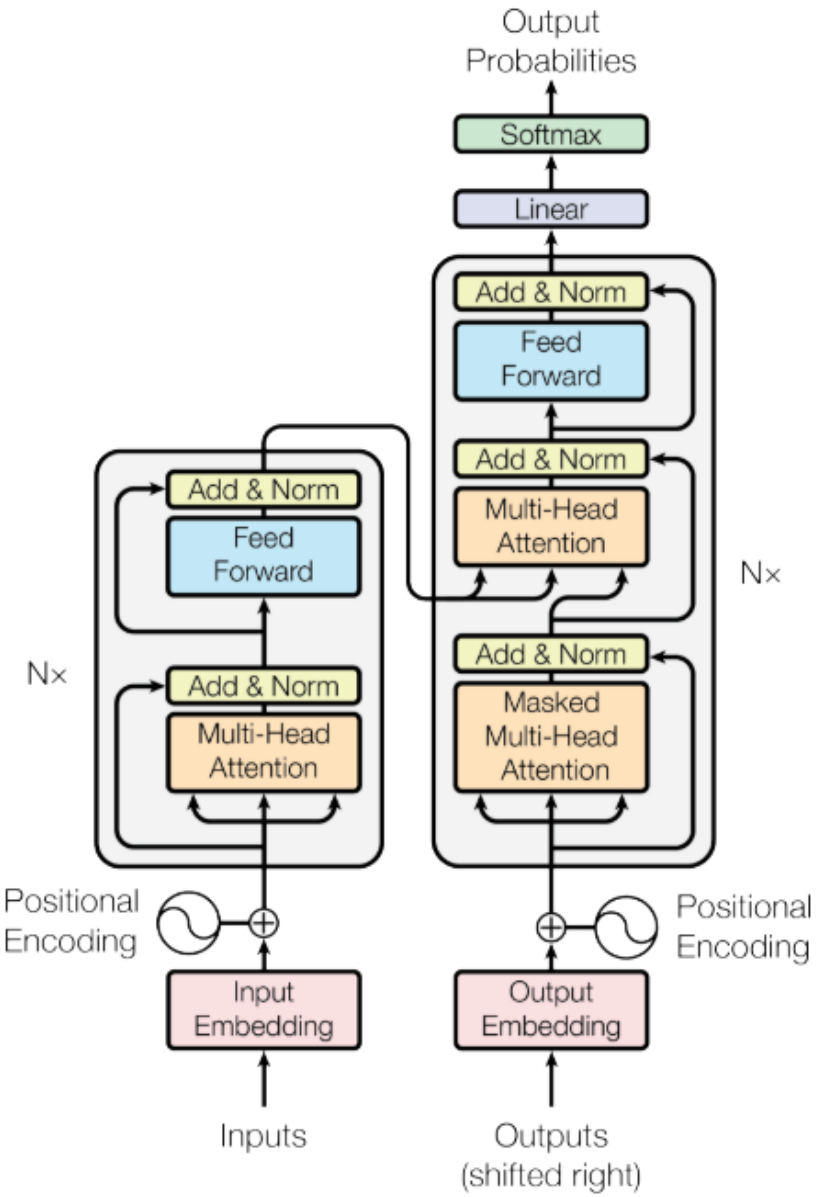
> ***Your task is to train a graph convolutional autoencoder to identify spatial domains in a spatial transcriptomics (MERFISH) sample.***

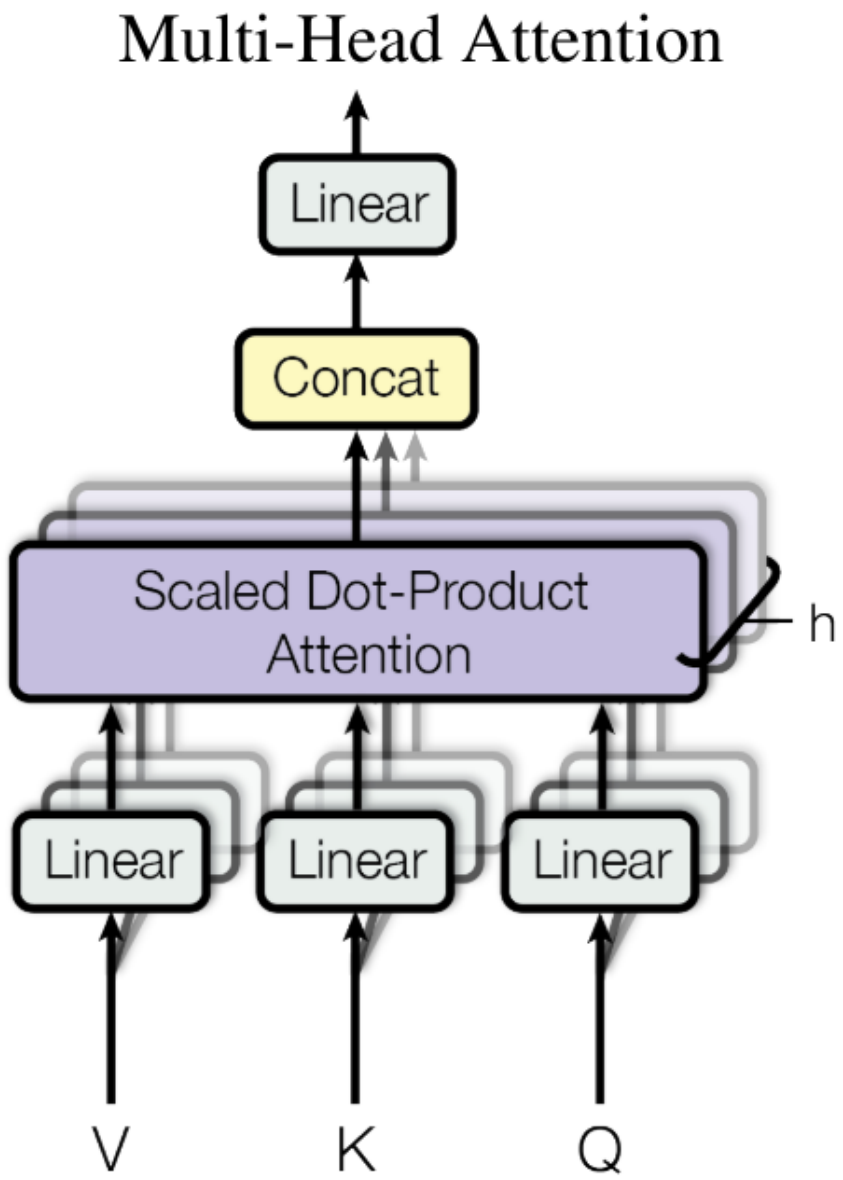
1. Install all necessary packages
2. Download and preprocess spatial transcriptomics sample
3. Construct the spatial cell graph
4. Aggregate the gene expression per cellular neighborhood
5. Implement a convolutional GNN
6. Build and train a graph autoencoder
7. Cluster embeddings and visualize spatial domains

Link to lab: https://learn.baiome.org/

From GNNs to Transformers

The Transformer architecture





Single-head computation

First, we linearly project the input embeddings with three different learnable weight matrices to obtain query tensor Q, key tensor K, and value tensor V:

$$Q = H^0 W_q$$
$$K = H^0 W_k$$
$$V = H^0 W_v$$

Next, we calculate the attention weights between all nodes:

$$A = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$$

Finally, we obtain the output embeddings:

$$H^1 = AV$$

Single-head computation as attentional GNN layer

Assume we are given a fully connected graph, then we can translate the previous computation into the GNN framework as follows:

$$h_v^{i+1} = \phi\left(h_v, \bigoplus_{u \in \mathcal{N}(v)} a(h_u, h_v, e_{uv})\psi(h_u)\right)$$

Single-head computation as attentional GNN layer

Assume we are given a fully connected graph, then we can translate the previous computation into the GNN framework as follows:

$$h_v^{i+1} = \phi\left(h_v, \bigoplus_{u \in \mathcal{N}(v)} a(h_u, h_v, e_{uv})\psi(h_u)\right)$$

$$h_v^{i+1} = \bigoplus_{u \in V} \frac{h_v^i W_q (h_u^i W_k)^\top}{\sqrt{d_k}} h_u^i W_k$$

Single-head computation as attentional GNN layer

Assume we are given a fully connected graph, then we can translate the previous computation into the GNN framework as follows:

$$h_v^{i+1} = \phi\left(h_v, \bigoplus_{u \in \mathcal{N}(v)} a(h_u, h_v, e_{uv})\psi(h_u)\right)$$

$$h_v^{i+1} = \bigoplus_{u \in V} \frac{h_v^i W_q (h_u^i W_k)^\top}{\sqrt{d_k}} h_u^i W_k$$

$$h_v^{i+1} = \sum_{u \in V} \frac{\exp\left(\frac{h_v^i W_q (h_u^i W_k)^\top}{\sqrt{d_k}}\right)}{\sum_{u \in V} \exp\left(\frac{h_v^i W_q (h_u^i W_k)^\top}{\sqrt{d_k}}\right)} h_u^i W_k$$

Multi-head attention

> ***Multi-head attention can now be understood as the parallel application of multiple single-head layers, whose outputs are concatenated in the end.***

Transformers in practice

- Large Language Models: basically all current state-of-the-art LLMs are decoder-only Transformers.
- Vision Transformers: ViTs are essentially encoder-only Transformers and perform as well or even outperform CNN-based architectures. They are utilized in almost all state-of-the-art models in the medical imaging domain.

> ***The big strength of Transformers is their very good scaling behavior with large amounts of data.***
>
> However, the architecture also has weaknesses, e.g., quadratic complexity with the sequence length (graph size) and limited context lengths.

Transformer hands-on

Overview

> ***Here your task is to implement a simple Transformer encoder from scratch and to apply it as a Vision Transformer to an image dataset.***

1. Implement Transformer encoder
2. Expand into a Vision Transformer
3. Train ViT on MNIST data

Link to lab: https://learn.baiome.org/