UNIVERSITAT DE BARCELONA

Jordi Vitrià

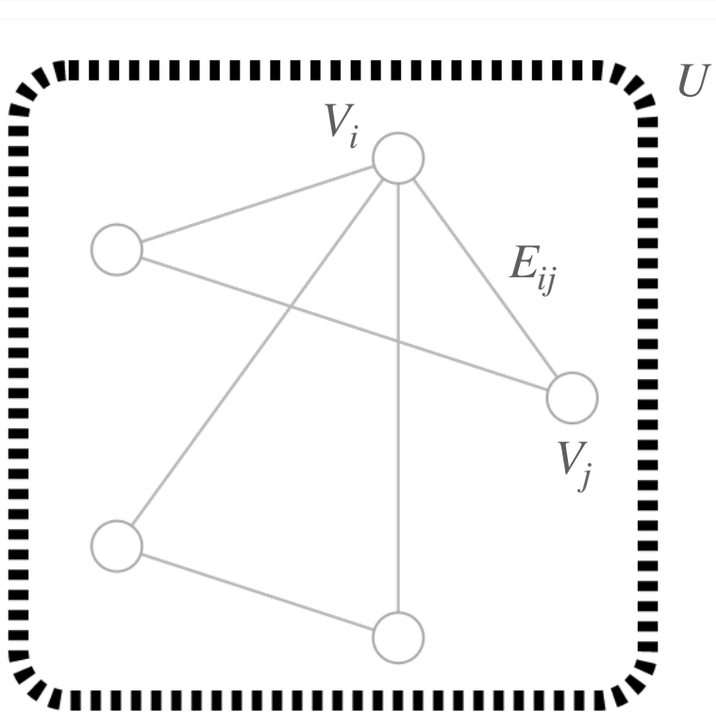
Deep Learning

Deep Learning and Graphs

Graphs

A set of objects, and the connections between them, are naturally expressed as a *graph*.

Researchers have developed neural networks that operate on graph data (called **graph neural networks**, or GNNs)



V Vertex (or node) attributes
e.g., node identity, number of neighbors

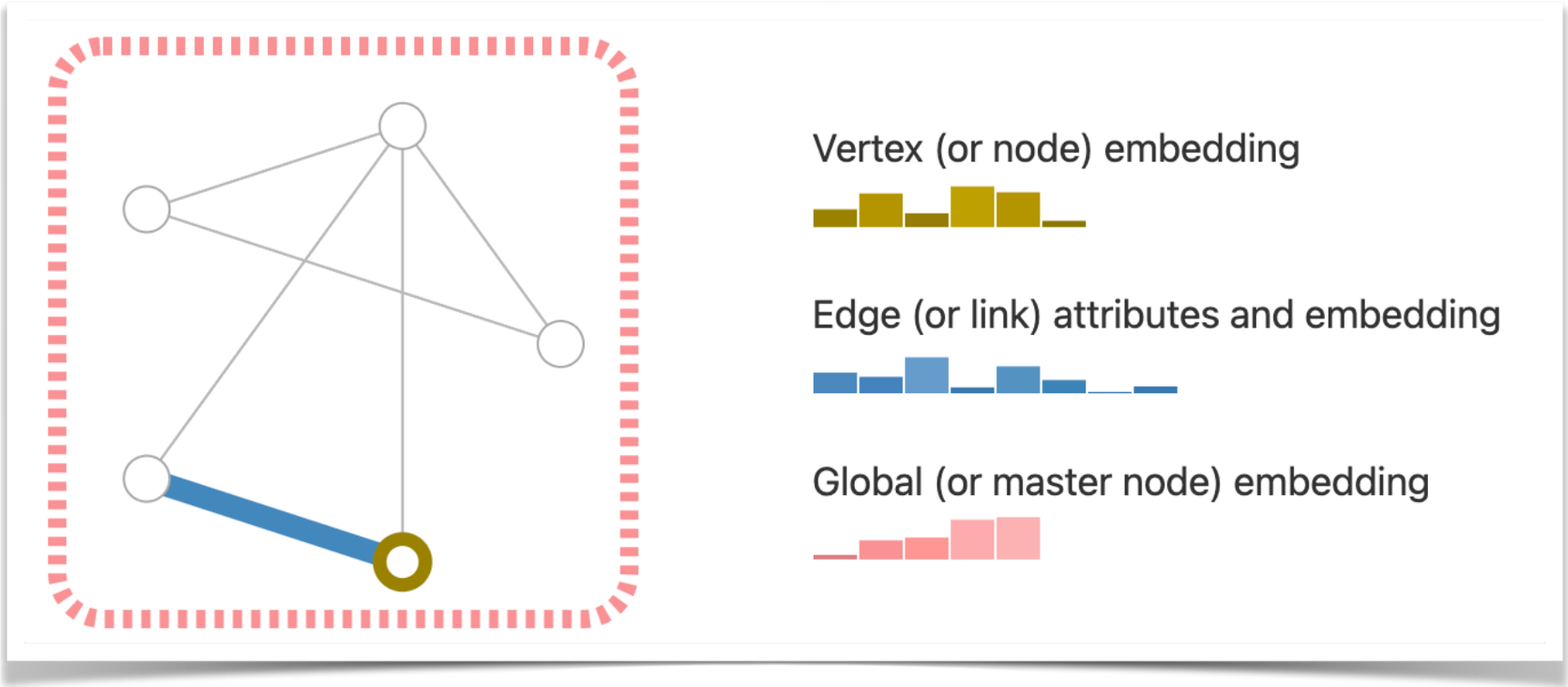
E Edge (or link) attributes and directions
e.g., edge identity, edge weight

U Global (or master node) attributes
e.g., number of nodes, longest path

https://distill.pub/2021/gnn-intro/

Graph

To further describe each node, edge or the entire graph, we can **store information in each of these pieces of the graph**.



https://distill.pub/2021/gnn-intro/

Images as Graphs

We typically think of images as rectangular grids.

Another way to think of images is as graphs with regular structure, where each pixel represents a node and is connected via an edge to adjacent pixels. Each non-border pixel has exactly 8 neighbors, and the information stored at each node is a 3-dimensional vector representing the RGB value of the pixel.

Adjacency matrix. We order the nodes, in this case each of 25 pixels in a simple 5x5 image of a smiley face, and fill a matrix with an entry if two nodes share an edge.

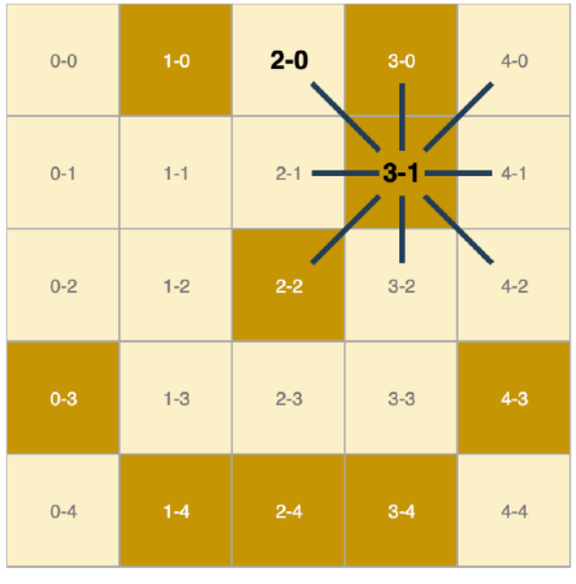
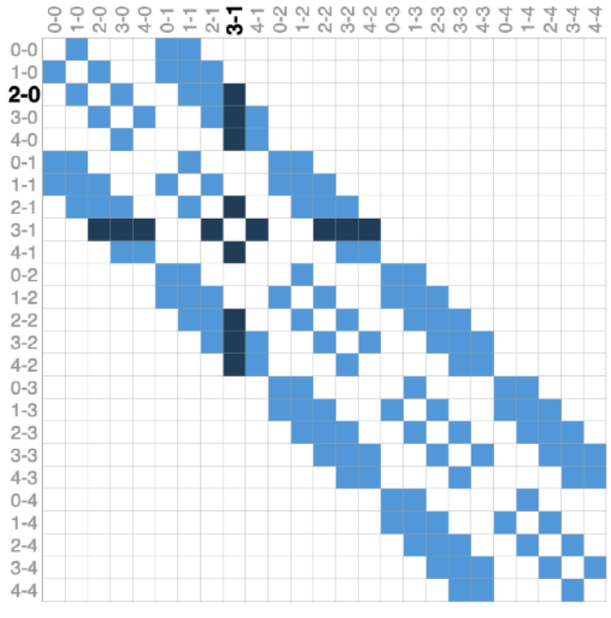
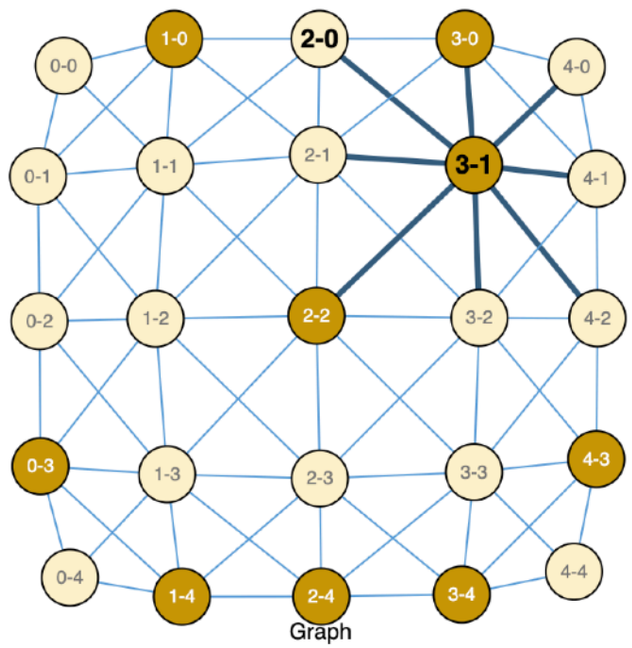


Image Pixels



Adjacency Matrix

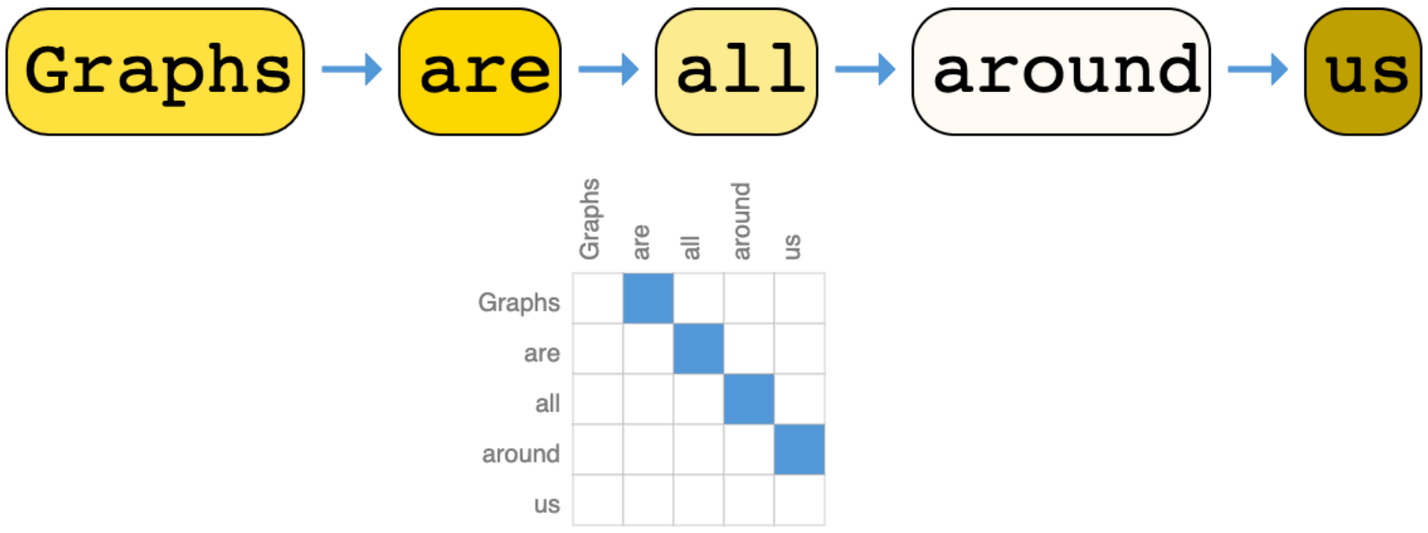


Graph

https://distill.pub/2021/gnn-intro/

Text as Graphs

We can digitize text by associating indices to each character, word, or token, and representing text as a sequence of these indices.



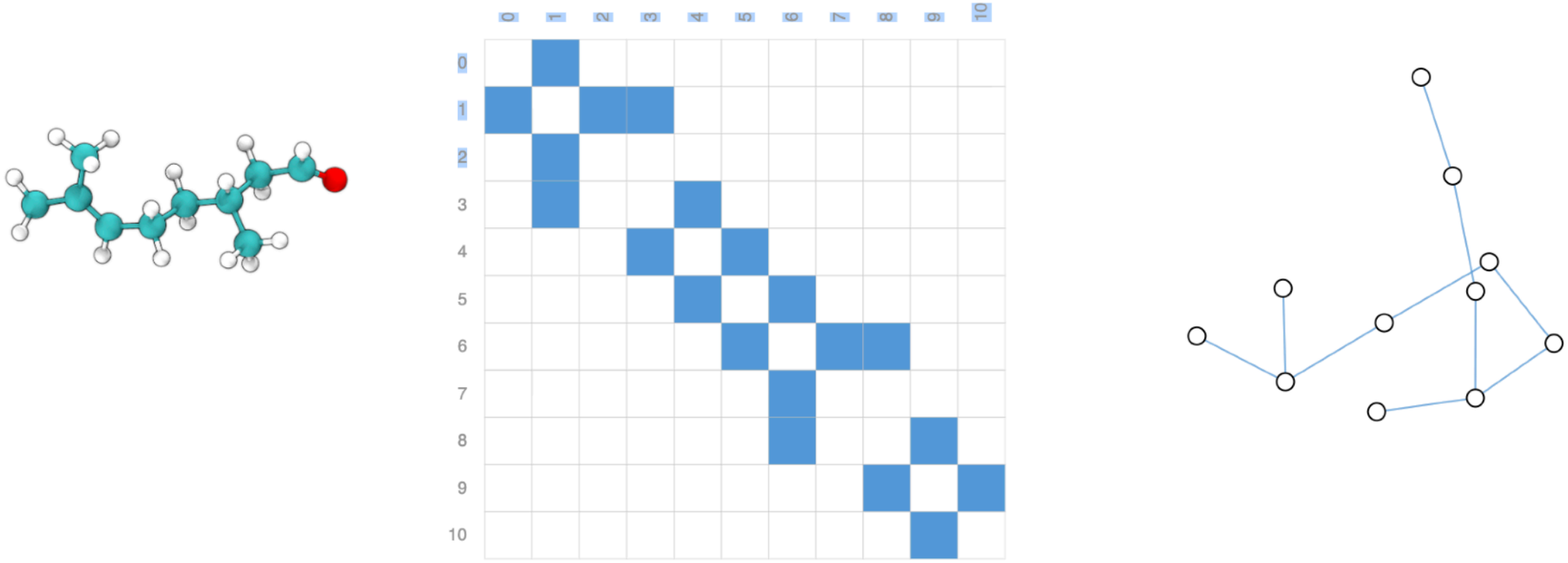
https://distill.pub/2021/gnn-intro/

Of course, in practice, this is not usually how text and images are encoded: these graph representations are redundant since all images and all text will have very regular structures.

Graph Data

Molecules are the building blocks of matter, and are built of atoms and electrons in 3D space. All particles are interacting, **but when a pair of atoms are stuck in a stable distance from each other**, we say they share **a covalent bond**.

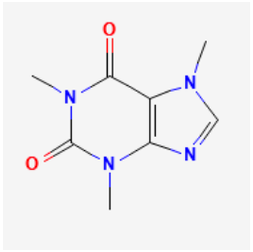
It's a very convenient and common abstraction to describe this 3D object as a graph, where nodes are atoms and edges are covalent bonds.



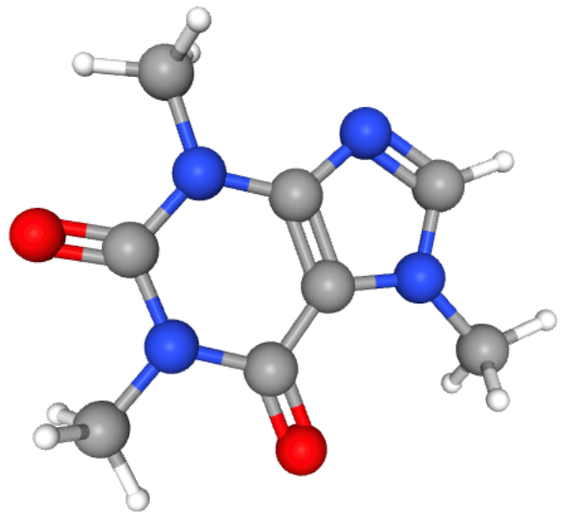
(Left) 3d representation of the Citronellal molecule (Center) Adjacency matrix of the bonds in the molecule (Right) Graph representation of the molecule.

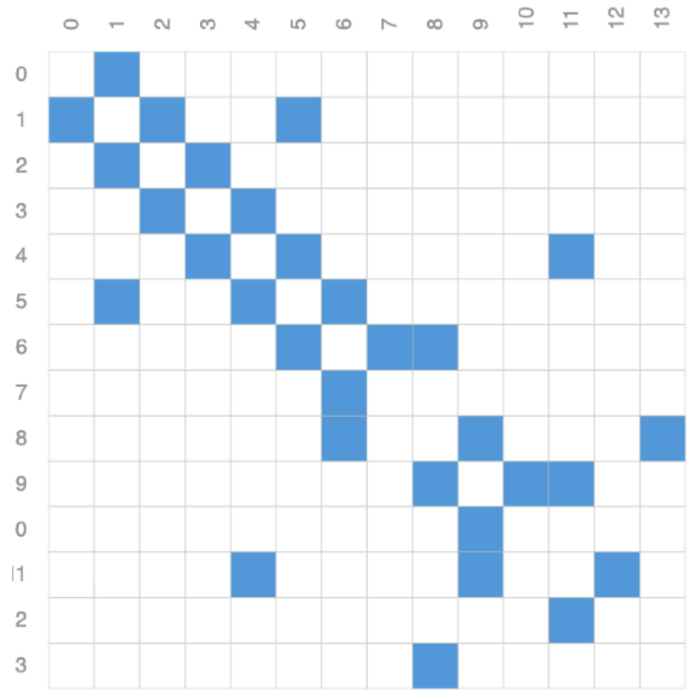
https://distill.pub/2021/gnn-intro/

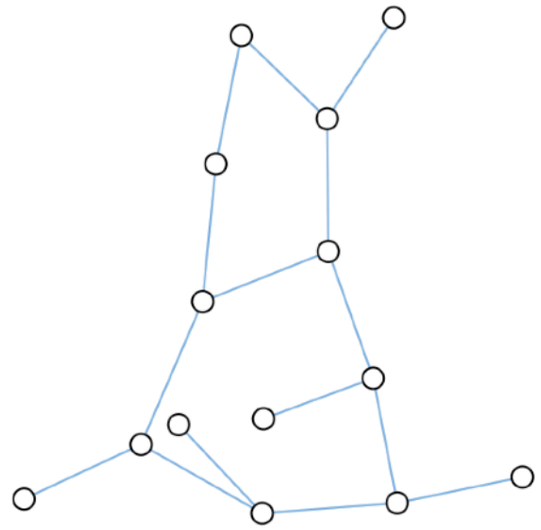
Graph Data



Smile:
CN1C=NC2=C1C(=O)N(C(=O)N2C)C





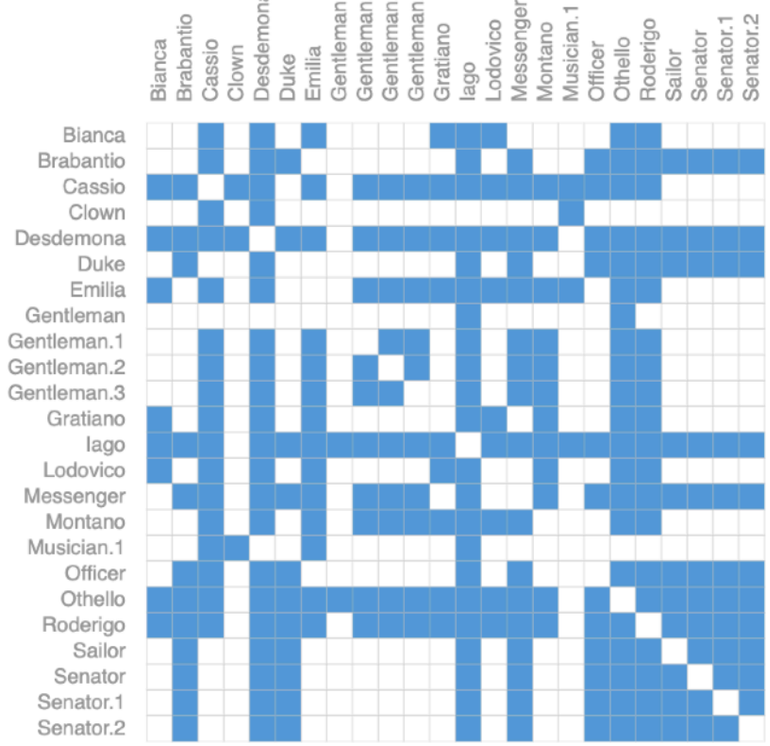


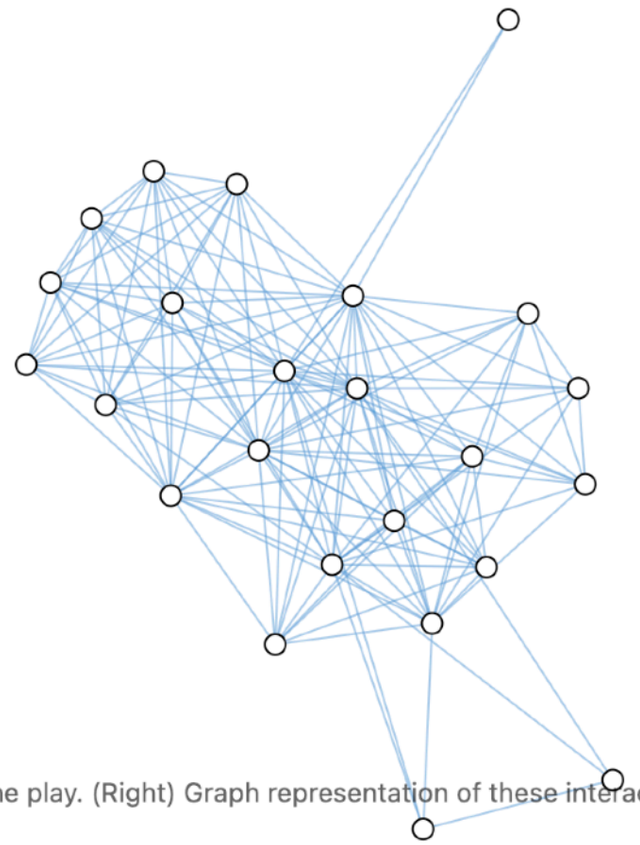
(Left) 3d representation of the Caffeine molecule (Center) Adjacency matrix of the bonds in the molecule (Right) Graph representation of the molecule.

https://distill.pub/2021/gnn-intro/

Graph Data

Social networks are tools to study patterns in collective behaviour of people, institutions and organizations. We can build a graph representing groups of people by modelling individuals as nodes, and their relationships as edges.





(Left) Image of a scene from the play “Othello”. (Center) Adjacency matrix of the interaction between characters in the play. (Right) Graph representation of these interactions.

https://distill.pub/2021/gnn-intro/

Graph Data

The structure of real-world graphs can vary greatly between different types of data — some graphs have many nodes with few connections between them, or vice versa.
Graph datasets can vary widely (both within a given dataset, and between datasets) in terms of the number of nodes, edges, and the connectivity of nodes.

					Edges per node (degree)		
Dataset	**Domain**	**graphs**	**nodes**	**edges**	**min**	**mean**	**max**
karate club	Social network	1	34	78		4.5	17
qm9	Small molecules	134k	≤ 9	≤26	1	2	5
Cora	Citation network	1	23,166	91,500	1	7.8	379
Wikipedia links, English	Knowledge graph	1	12M	378M		62.24	1M

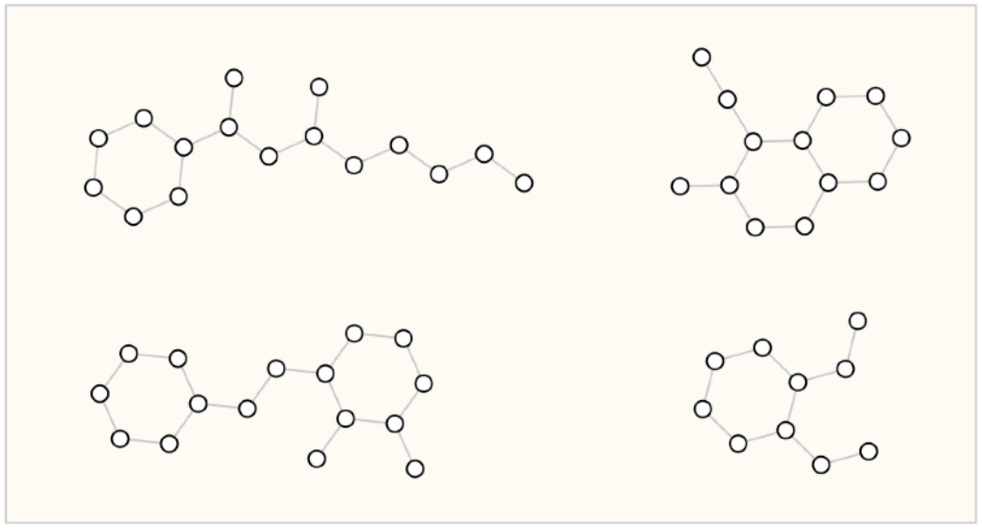
Summary statistics on graphs found in the real world. Numbers are dependent on featurization decisions. More useful statistics and graphs can be found in KONECT [14]

https://distill.pub/2021/gnn-intro/

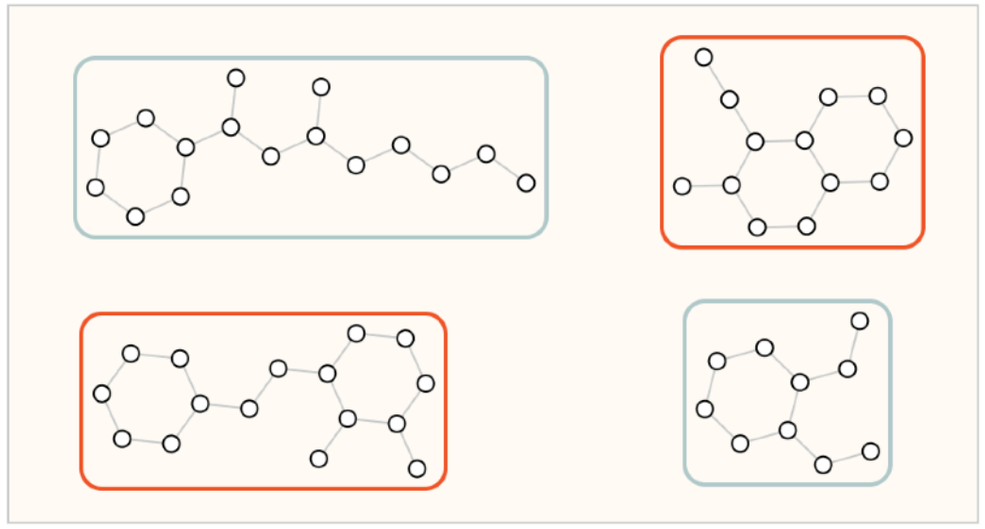
Graph-level Task

In a graph-level task, our goal is to predict the property of an entire graph.

For example, for a molecule represented as a graph, we might want to predict what the molecule smells like, or whether it will **bind to a receptor implicated in a disease**.



Input: graphs

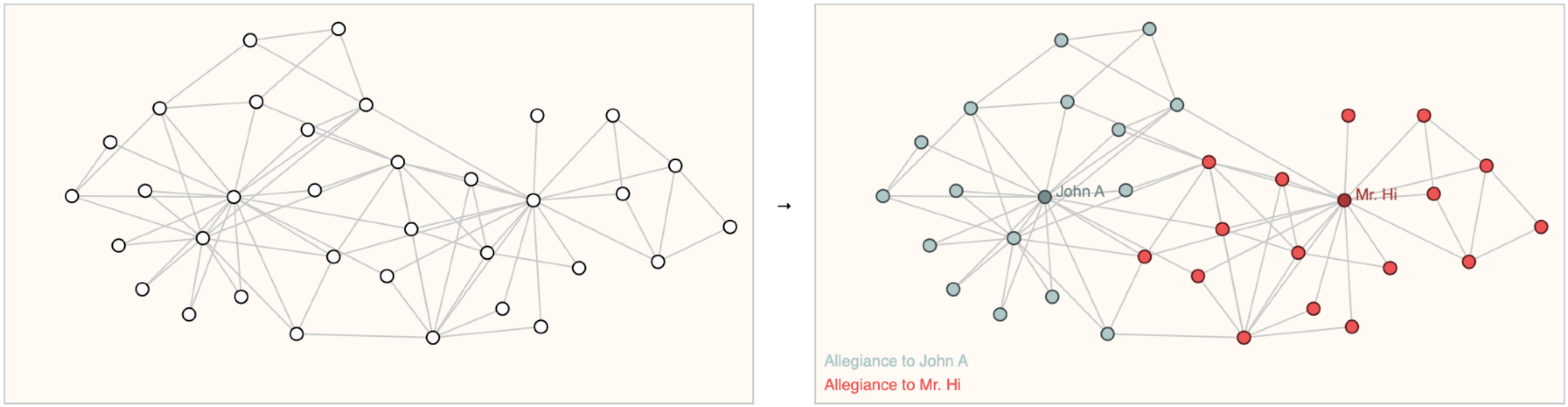


Output: labels for each graph, (e.g., "does the graph contain two rings?")

https://distill.pub/2021/gnn-intro/

Node-level Task

Node-level tasks are concerned with predicting the identity or role of each node within a graph.



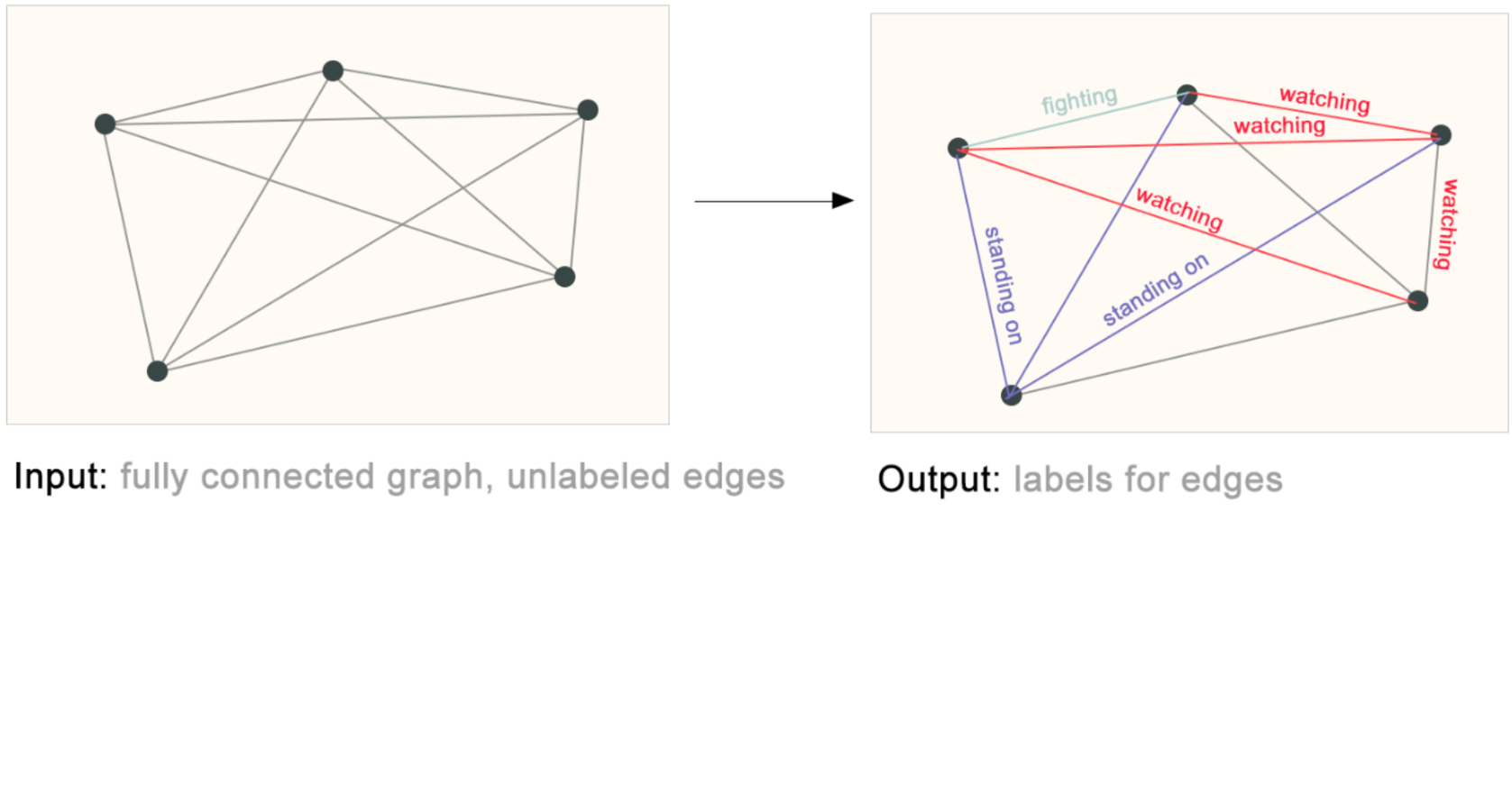
Input: graph with unlabled nodes

Output: graph node labels

https://distill.pub/2021/gnn-intro/

Edge-level Task

Given nodes that represent the objects in the image, we wish to predict which of these nodes share an edge or what the value of that edge is.



Input: fully connected graph, unlabeled edges

Output: labels for edges

https://distill.pub/2021/gnn-intro/

Challenges

How do we go about solving these different graph tasks with neural networks?

The first step is to think about how we will represent graphs to be compatible with neural networks.

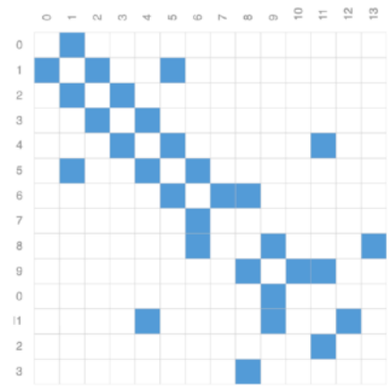
Graphs have up to four types of information that we will potentially want to use to make predictions: **nodes**, **edges**, **global-context and connectivity**.

The first three are relatively straightforward (embeddings).

However, representing a graph's connectivity is more complicated.

How to represent connectivity

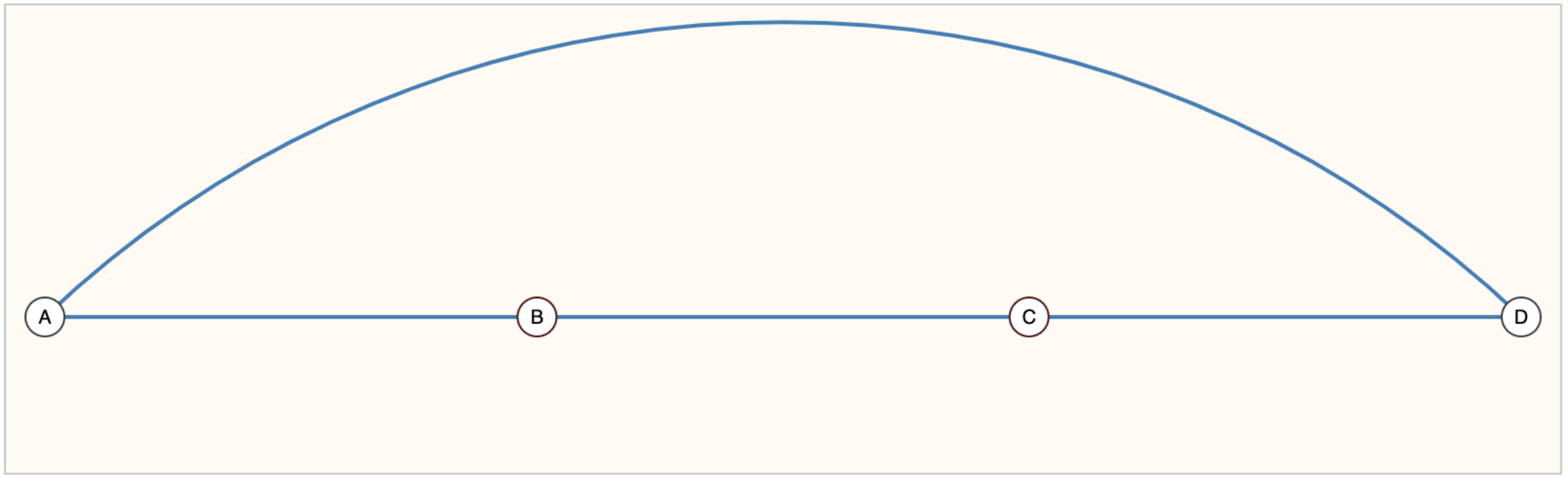
The most obvious choice would be to use an **adjacency matrix**, since this is easily tensorisable.

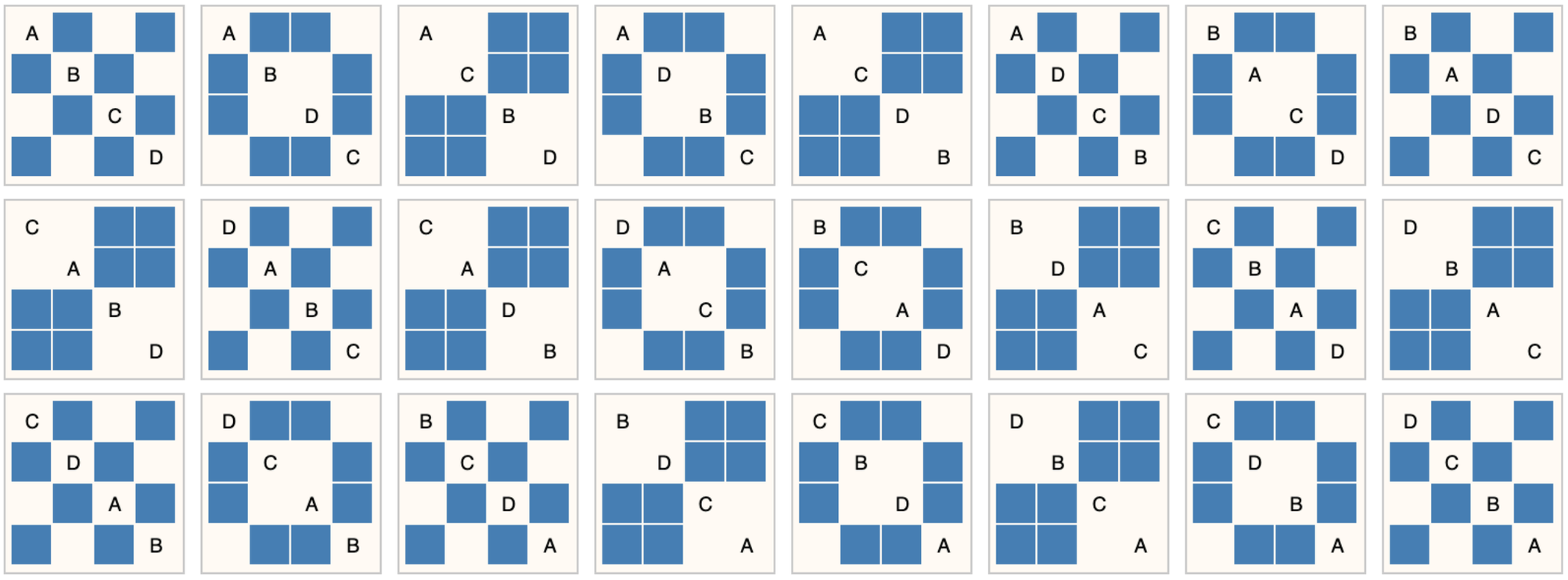


However, this representation has a few drawbacks:

- The number of nodes in a graph can be on the order of millions, and the number of edges per node can be highly variable. Often, this leads to very **sparse adjacency matrices**, which are space-inefficient.
- Another problem is that there are many adjacency matrices that can encode the same connectivity, and there is no guarantee that these different matrices would produce the same result in a deep neural network (that is to say, **they are not permutation invariant**).

How to represent connectivity

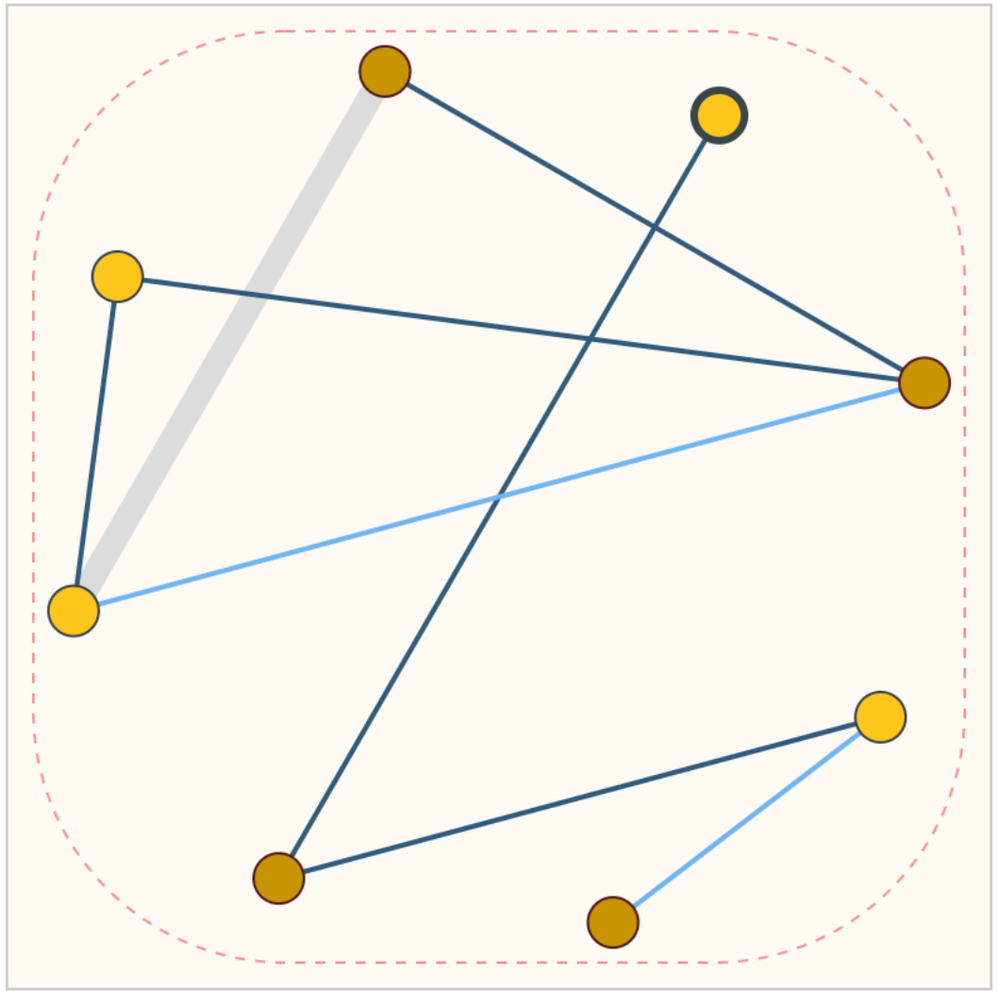




All these matrices represent the same graph

How to represent connectivity

One elegant and memory-efficient way of representing sparse matrices is as **adjacency lists**.



Nodes

```
[ 0 , 1 , 1 , 0 , 0 , 1 , 0 , 1 ]
```

Edges

```
[ 2 , 1 , 1 , 1 , 2 , 1 , 1 ]
```

Adjacency List

```
[ [1, 0] , [2, 0] , [4, 3] , [6, 2] ,
  [7, 3] , [7, 4] , [7, 5] ]
```

Global

```
0
```

It should be noted that the figure uses scalar values per node/edge/global, but most practical tensor representations have vectors per graph attribute.

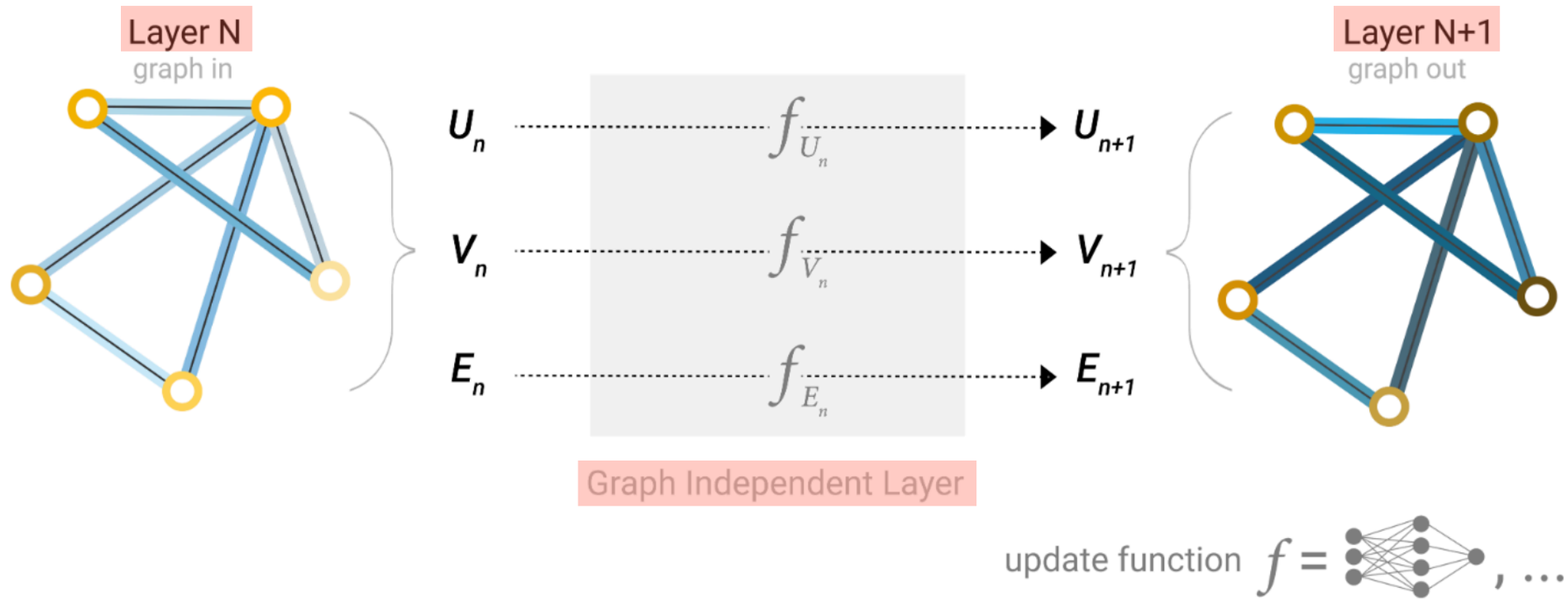
Graph Neural Networks

A GNN is an optimizable **transformation on all attributes** of the graph (nodes, edges, global-context) that preserves graph symmetries (**permutation invariances**).

GNNs adopt a "graph-in, graph-out" architecture meaning that these model types accept a graph as input, with information loaded into its nodes, edges and global-context, and progressively **transform these embeddings, without changing the connectivity of the input graph**.

Graph Neural Networks

We will start with the simplest GNN architecture, one where we learn new embeddings for all graph attributes (nodes, edges, global), but where we do not yet use the connectivity of the graph.



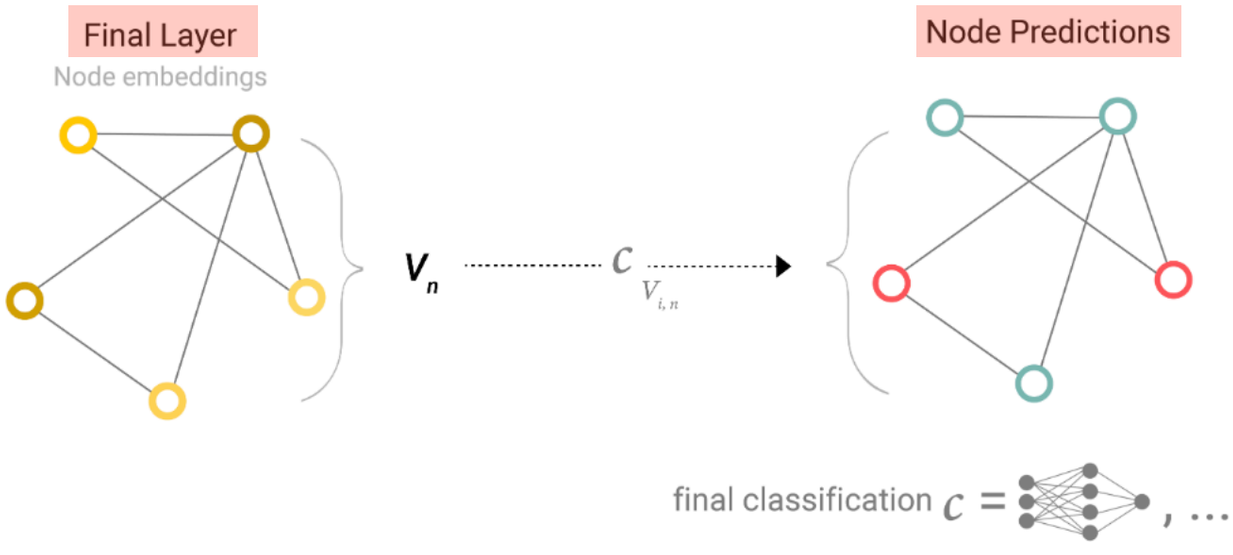
A single layer of a simple GNN. A graph is the input, and each component (V,E,U) gets updated by a MLP to produce a new graph. Each function subscript indicates a separate function for a different graph attribute at the n-th layer of a GNN model.

As is common with neural networks modules or layers, we can stack these GNN layers together.

Graph Neural Networks

How do we make predictions in any of the tasks we described before?

If the task is, for example, to make binary **predictions on nodes**, and the graph **already contains node information**, the approach is straightforward — for each node embedding, apply a classifier.



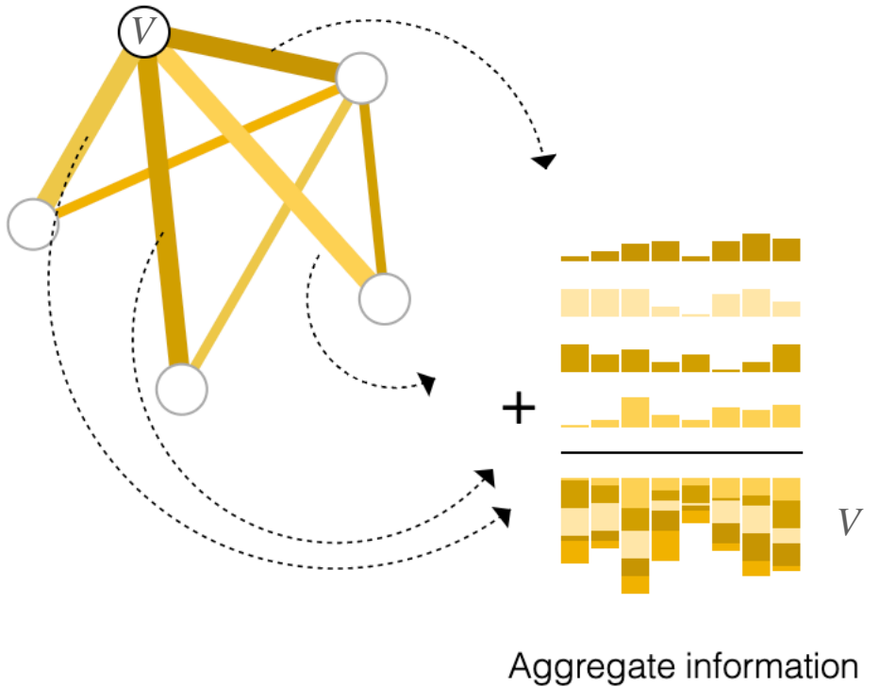
Graph Neural Networks

If we have information in the graph **stored in edges**, **but no information in nodes**, we need a way to collect information from edges and give them to nodes for prediction.

We can do this by ***pooling***.

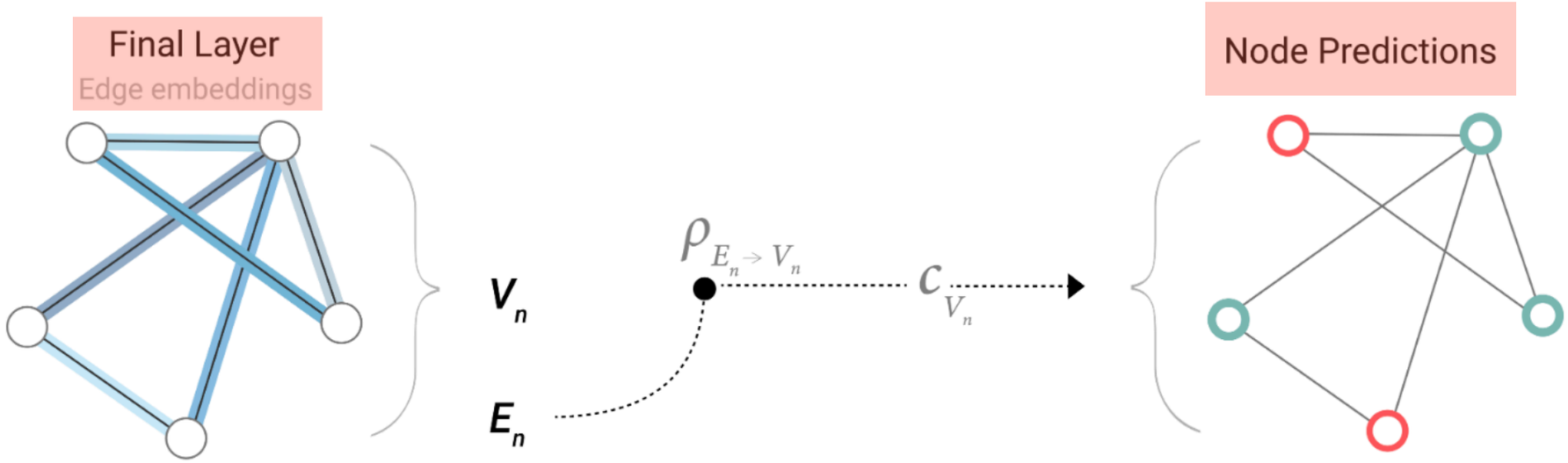
Pooling (ρ) proceeds in two steps:

- For each item to be pooled, gather each of their embeddings and concatenate them into a matrix.
- The gathered embeddings are then **aggregated**, usually via a sum operation.



Graph Neural Networks

The model looks like this.

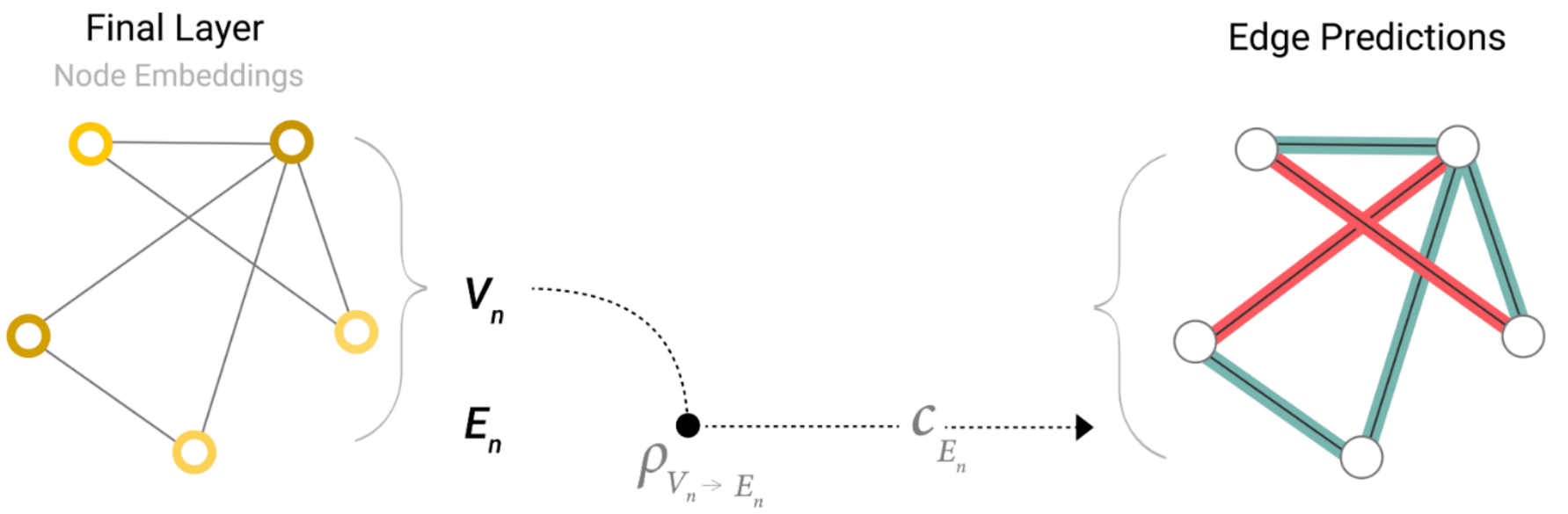


pooling function ρ

final classification c =  , ...

Graph Neural Networks

If we only have **node-level features**, and are trying to **predict edge-level information**, the model looks like this.



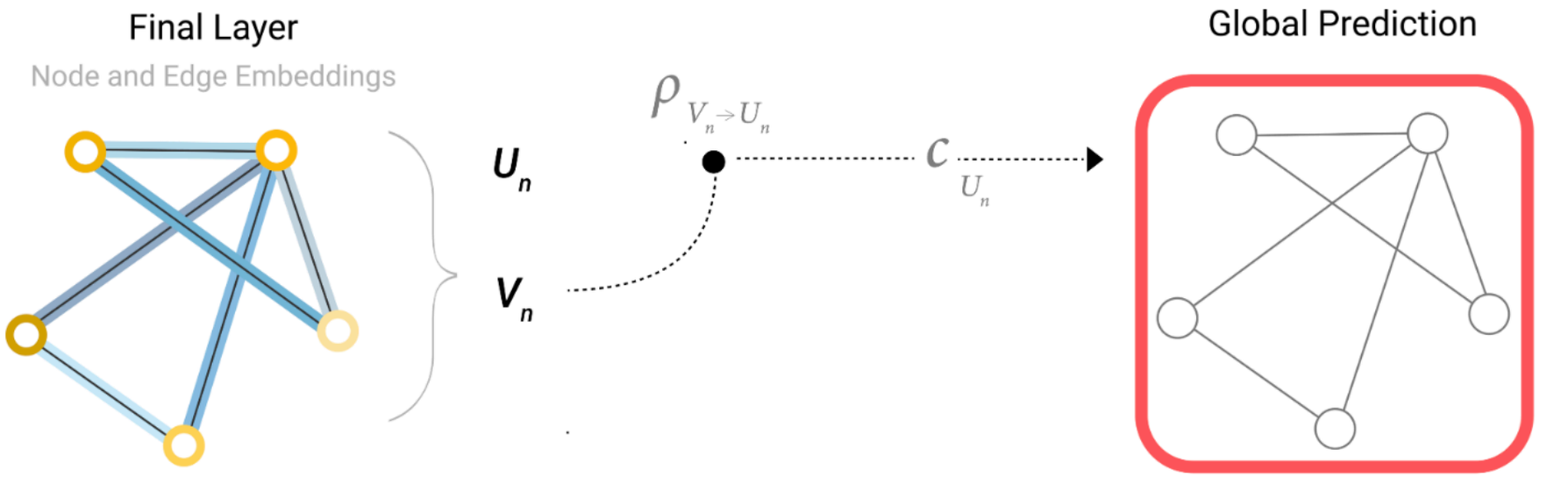
pooling function ρ

final classification c =  , ...

Graph Neural Networks

If we only have **node-level features**, and need to predict a **global property**, we need to gather all available node information together and aggregate them.

The same can be done for edges.

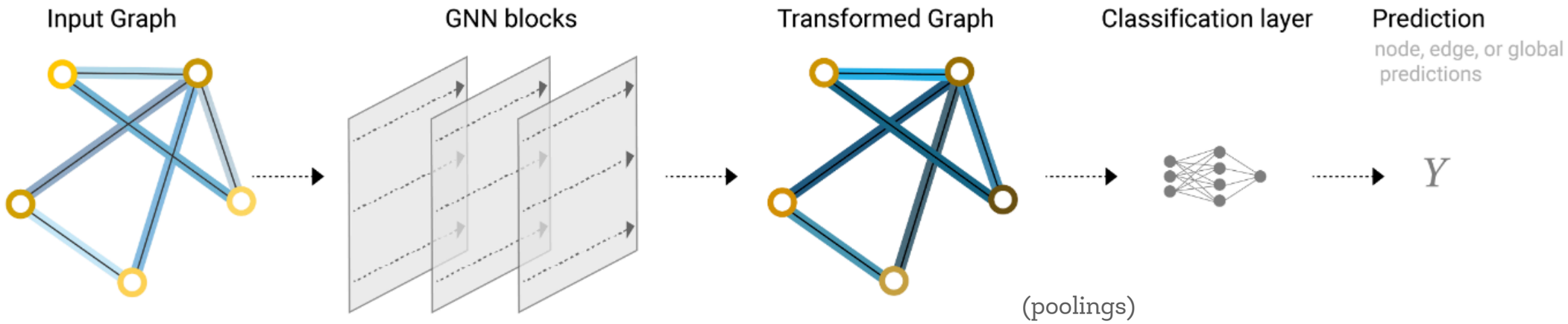


pooling function ρ

final classification $c =$, ...

Graph Neural Networks

The classification model c can be any differentiable model.



An end-to-end prediction task with a GNN model.

Note that in this simplest GNN formulation, **we're not using the connectivity of the graph** at all inside the **GNN layer**:

- We are using graph independent layers for computing embeddings. Each node is processed independently, as is each edge, as well as the global context.

We **only use connectivity when pooling information** for prediction.

Passing messages between parts of the graph

We could make more sophisticated predictions by using **pooling within the GNN layer**, in order to make our learned embeddings aware of graph connectivity.

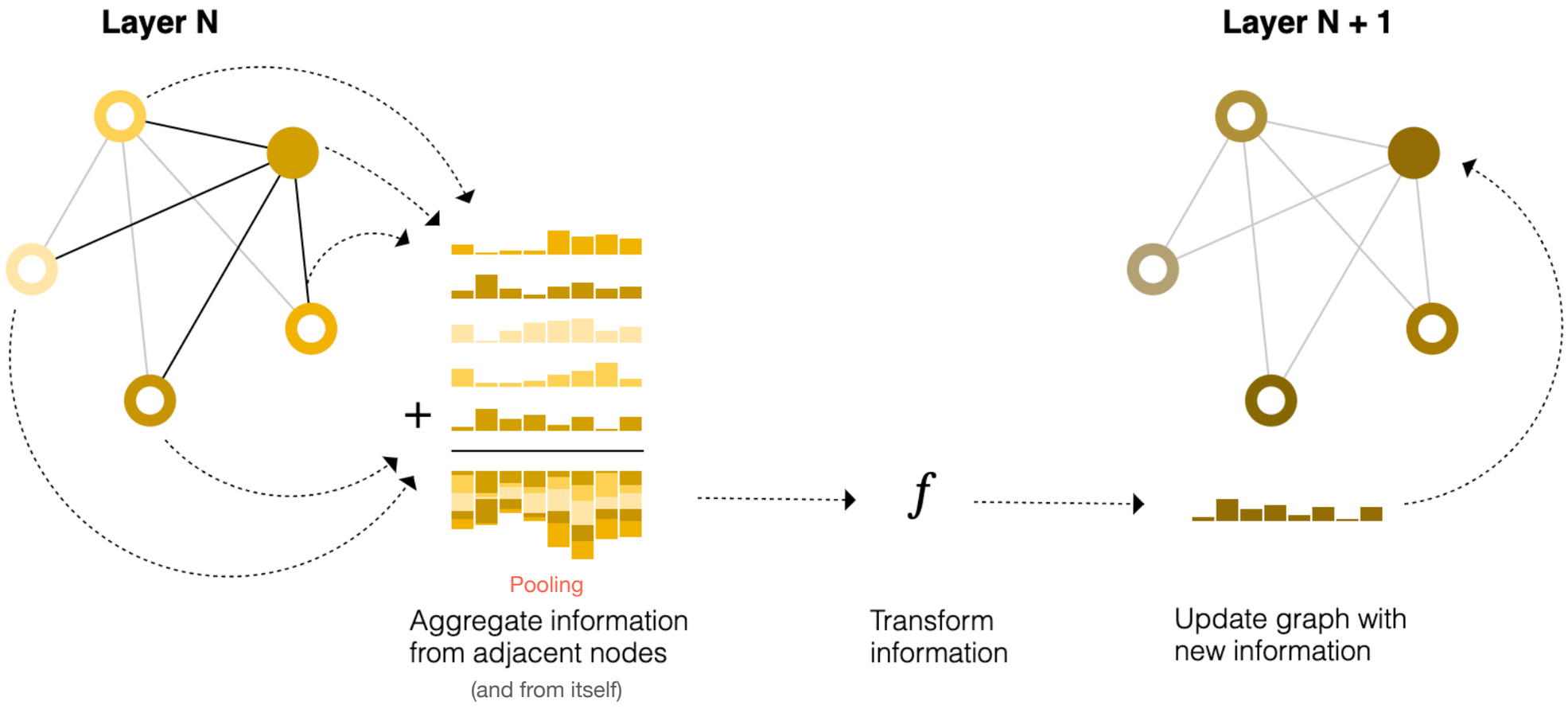
We can do this using **message passing**, where neighboring nodes or edges **exchange information** and **influence each other's updated embeddings**.

Message passing works in three steps:

1. For each node/edge in the graph, gather all the neighboring node/edge embeddings (or messages).
2. Aggregate all messages via an aggregate function (like sum).
3. All pooled messages are passed through an update function, usually a learned neural network.

Just as pooling can be applied to either nodes or edges, message passing can occur between either nodes or edges.

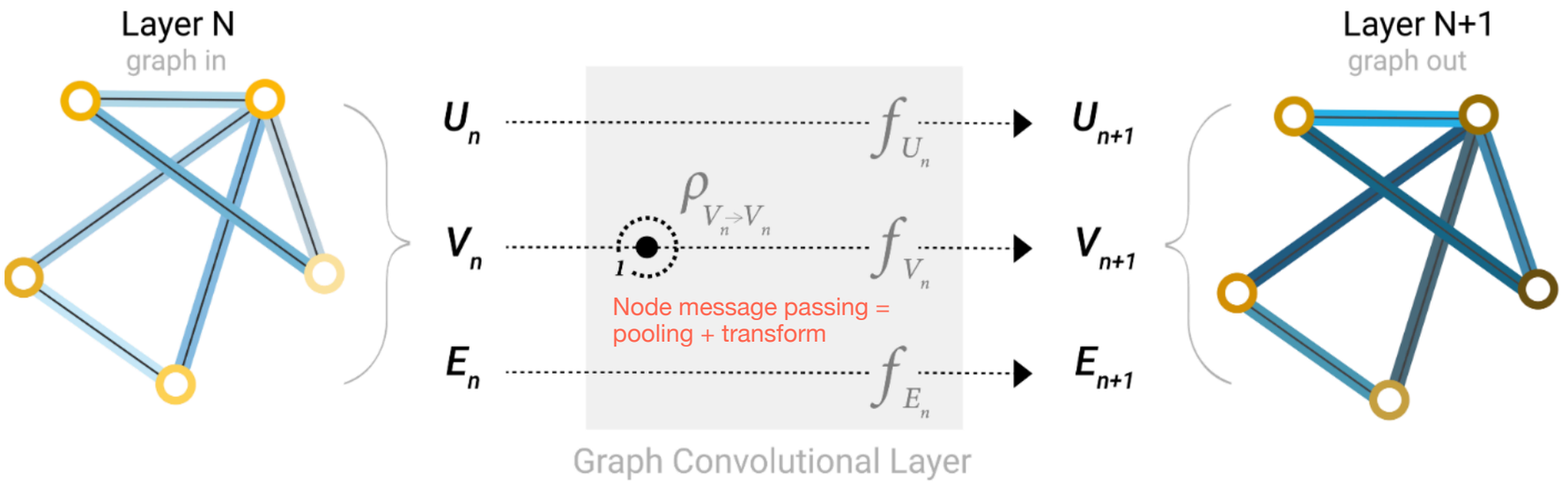
Passing messages between parts of the graph



By stacking message passing GNN layers together, **a node can eventually incorporate information from across the entire graph**: after three layers, a node has information about the nodes three steps away from it.

Passing messages between parts of the graph

We can update our architecture diagram to include this new source of information for nodes:



Graph Convolutional Layer

update function $f =$  , ...

pooling function ρ

Learning edge representations

We can also incorporate the information from **neighboring edges** in the same way we used neighboring node information earlier, by first pooling the edge information, transforming it with an update function, and storing it.

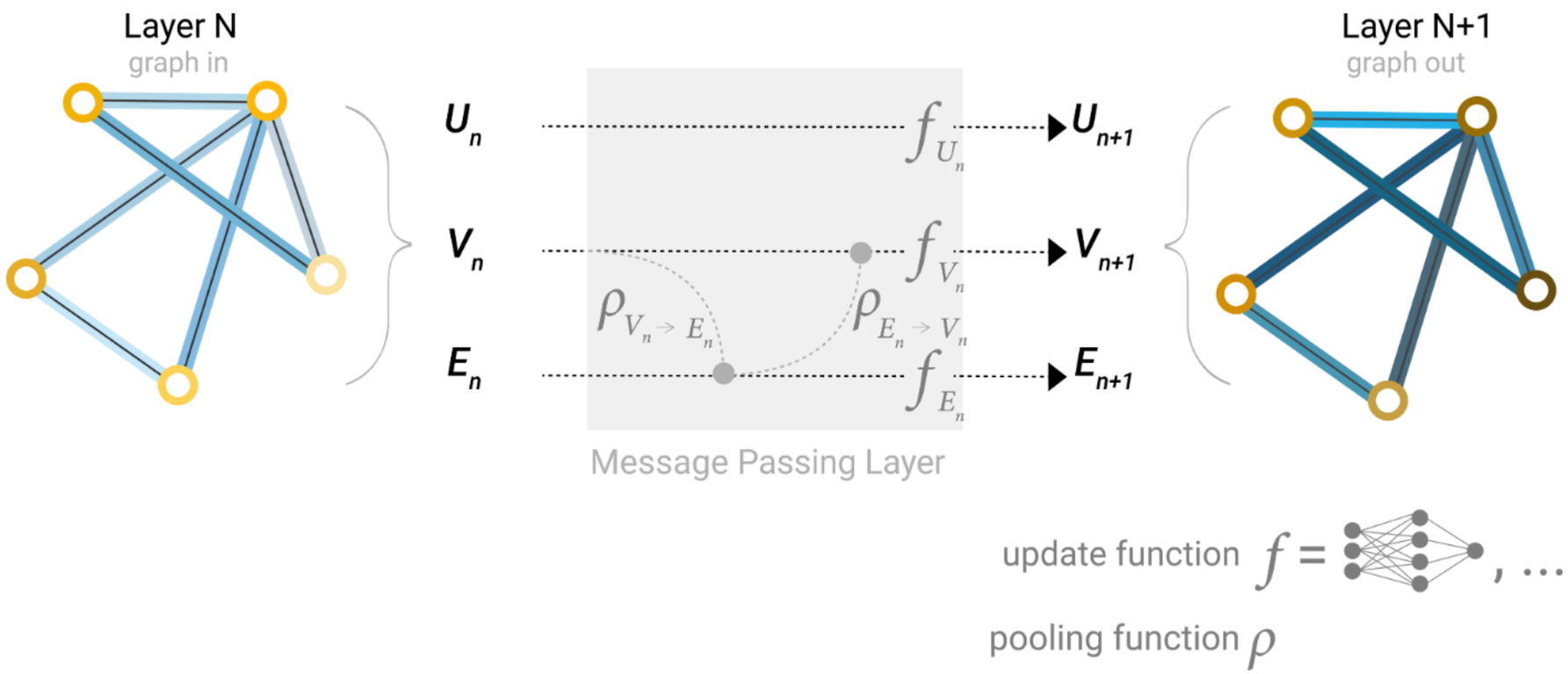
Now we could think of **sharing information between nodes and edges within the GNN layer** using message passing.

However, the node and edge information stored in a graph are not necessarily the **same size or shape**, so it is not immediately clear how to combine them.

Learning edge representations

One way is to learn a linear mapping from the space of edges to the space of nodes, and vice versa.

Alternatively, one may concatenate them together before the update function.

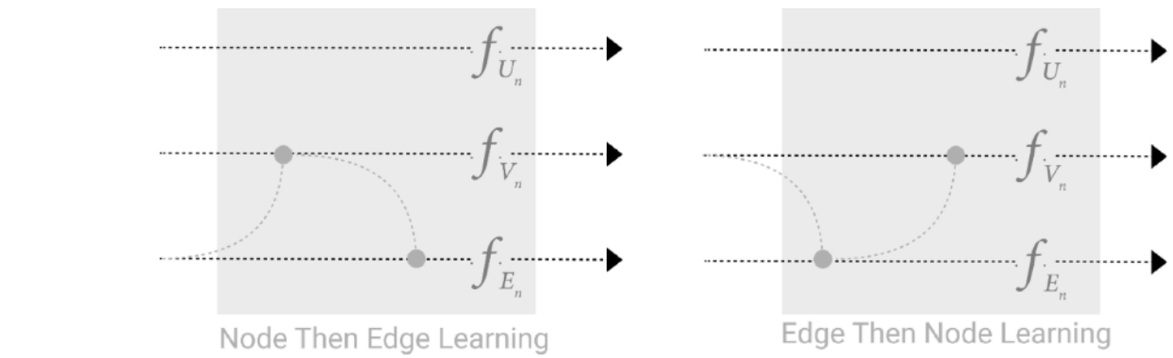


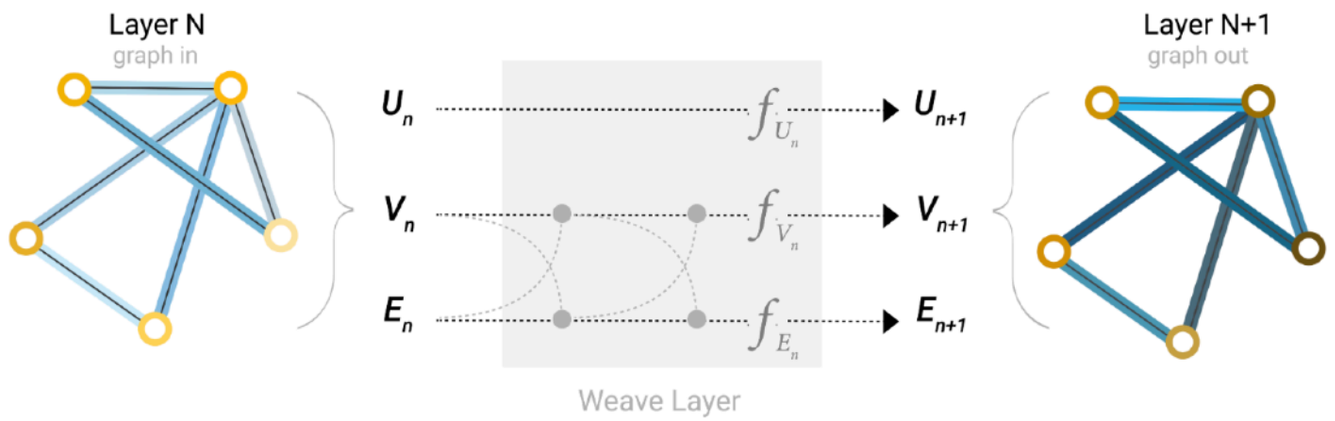
Architecture schematic for Message Passing layer. The first step “prepares” a message composed of information from an edge and it’s connected nodes and then “passes” the message to the node.

Learning edge representations

Which graph attributes we update and in which order we update them is one design decision when constructing GNNs.

We could choose whether to update node embeddings before edge embeddings, or the other way around.





update function $f =$, ...

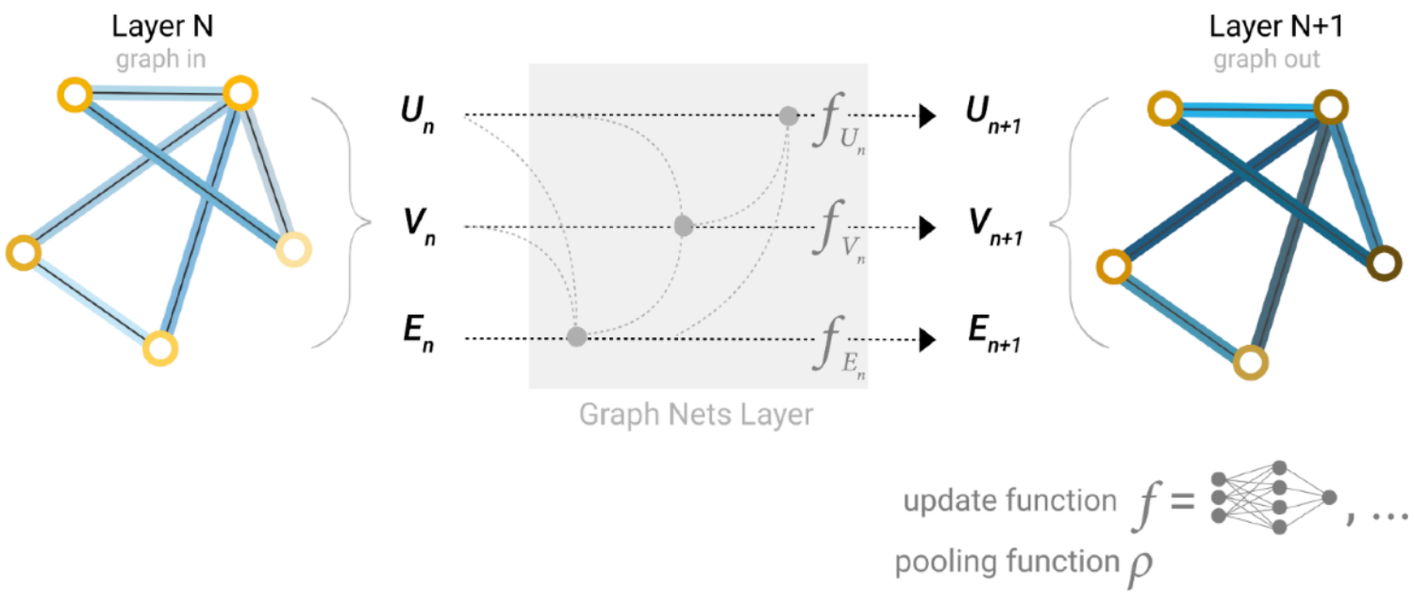
pooling function ρ

Adding global representations

There is one flaw with the networks we have described so far: nodes that are far away from each other in the graph may never be able to efficiently transfer information to one another, even if we apply message passing several times.

One solution to this problem is by using the global representation of a graph (U) which is sometimes called a master node or context vector.

This global context vector is connected to all other nodes and edges in the network, and can act as a bridge between them to pass information, building up a representation for the graph as a whole.



Some empirical GNN design lessons

- GNNs are a very parameter-efficient model type: for even a **small number of parameters** we can already find models with high performance.
- GNN with a **higher number of layers** will broadcast information at a higher distance and can risk having their node representations 'diluted' from many successive iterations.
- Regarding aggregation operations, *sum* has a very slight improvement, but *max* or *mean* can give equally good models.
- The more graph attributes are communicating, the better the performance of the model.

Embedding Computation

Message-passing forms the backbone of many GNN **architectures** today. The most popular is **Graph Convolutional Networks** (GCN).

$h_v^{(0)}$	$=$	x_v	for all $v \in V$.
Node v's initial embedding.		... is just node v's original features.	

Embedding Computation

Message-passing forms the backbone of many GNN **architectures** today. The most popular is **Graph Convolutional Networks** (GCN).

and for $k = 1, 2, \ldots$ upto K:

$$h_v^{(k)} = f^{(k)}\left(W^{(k)} \cdot \frac{\sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}}{|\mathcal{N}(v)|} + B^{(k)} \cdot h_v^{(k-1)} \right) \quad \text{for all } v \in V.$$

This is a self-connection

Non linear function

Node v's embedding at step k.

Mean of v's neighbour's embeddings at step $k-1$.

Node v's embedding at step $k-1$.

Color Codes:

- Embedding of node v.
- Embedding of a neighbour of node v.
- (Potentially) Learnable parameters.

Embedding Computation

Predictions can be made at each node by using the final computed embedding:

$$\hat{y_v} = \text{PREDICT}(h_v^{(K)})$$

where PREDICT is generally another neural network, learnt together with the GCN model.

For each step k, the function $f^{(k)}$, matrices $W^{(k)}$ and $B^{(k)}$ are shared across all nodes.

This allows the GCN model to scale well, because the number of parameters in the model is not tied to the size of the graph.

- GCNs are much more computationally effective than their predecessors bu they don’t directly support edge features.

Embedding Computation