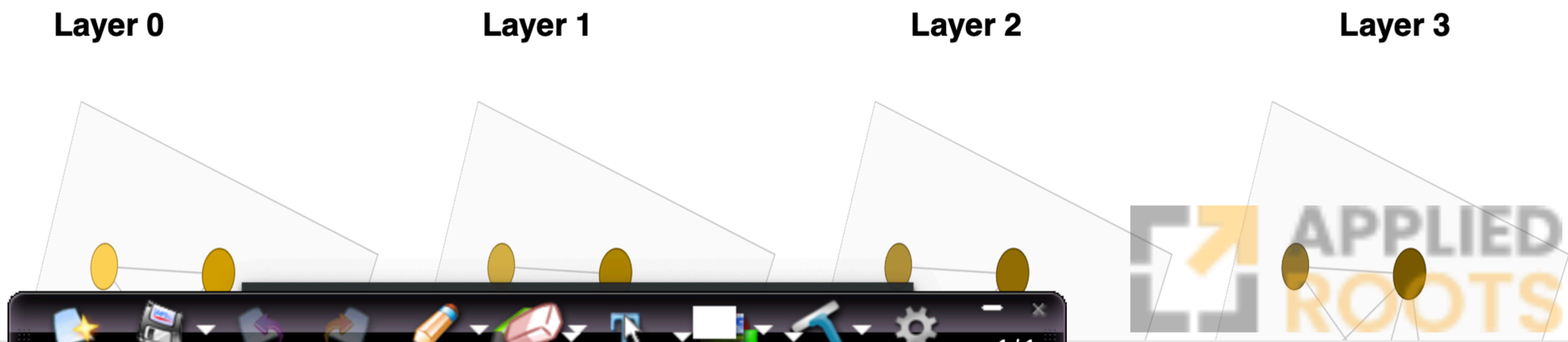
A Gentle Introduction to Graph Neural Networks

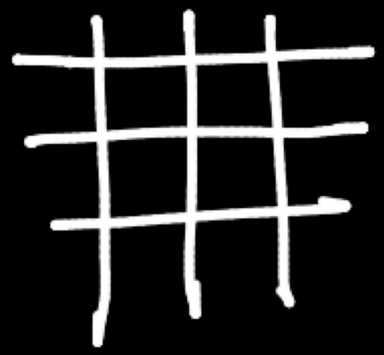
Neural networks have been adapted to leverage the structure and properties of graphs. We explore the components needed for building a graph neural network - and motivate the design choices behind them.



Geometric Learning

DL on 5G's

↳ Grids



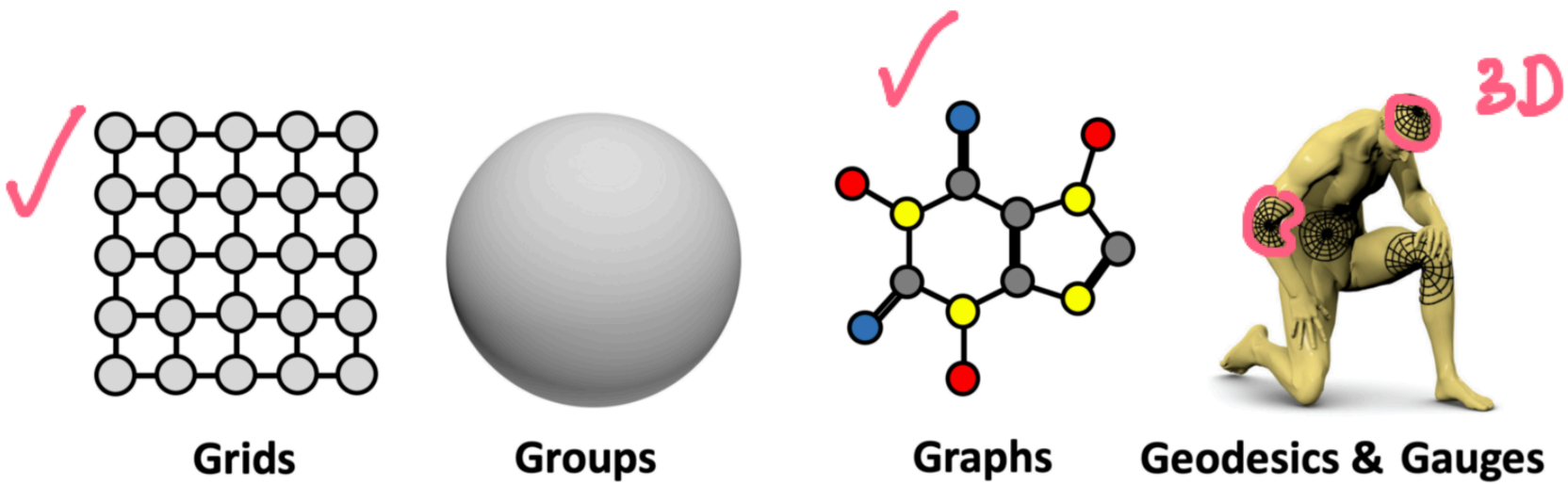
Group (Thy)

✓ { Graphs → Graph NN

Geodesics

Gauges (Thy) ← X

most popular architectures that exist today in deep learning: Convolutional Networks (CNNs), emerging from *translational symmetry*, Graph Neural Networks, DeepSets [11], and Transformers [12], implementing *permutation invariance*, gated RNNs (such as LSTM networks) that are invariant to *time warping* [13], and Intrinsic Mesh CNNs [14] used in computer graphics and vision, that can be derived from *gauge symmetry*.



The “5G” of Geometric Deep Learning: Grids, Group (homogeneous spaces with global symmetries), Graphs (and sets as a particular case), and Manifolds, where geometric priors are manifested through global isometry invariance (which can be expressed using Geodesics) and local Gauge symmetries.

In future posts, we will be exploring in further detail the instances of a final

Graph-NN

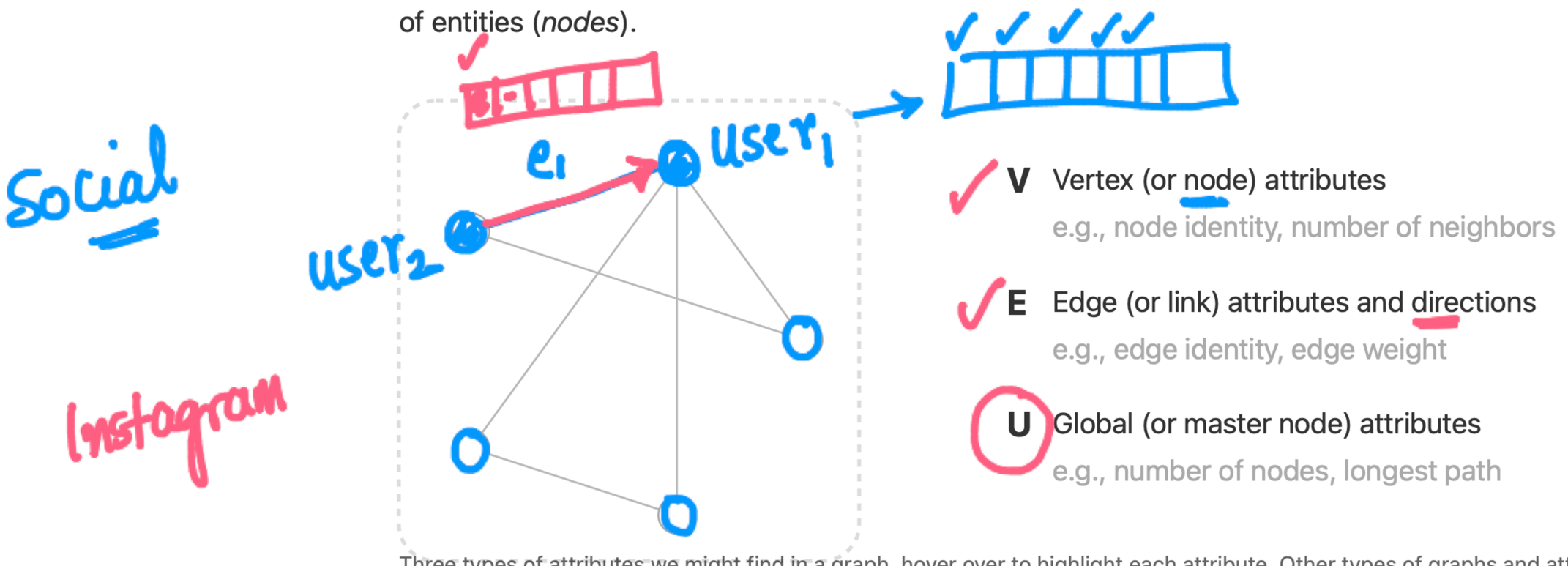
- 10 years [~2018 , ~~2021~~/~~2020~~]

- minimalist GNN ✓

⋮

SOTA GNN ✓

of entities (*nodes*).



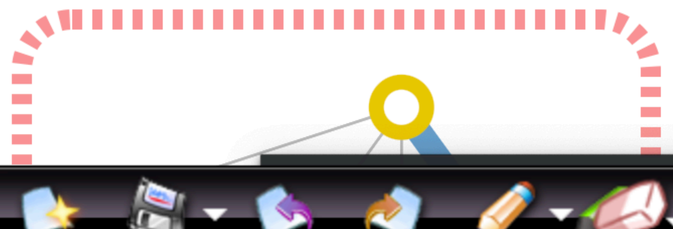
V Vertex (or node) attributes
e.g., node identity, number of neighbors

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

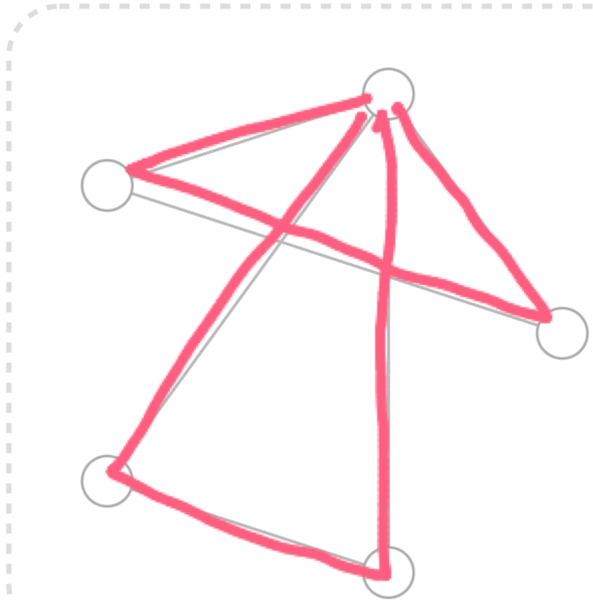
Three types of attributes we might find in a graph, hover over to highlight each attribute. Other types of graphs and attributes are explored in the Other types of graphs section.

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



Vertex (or node) embedding

of entities (*nodes*).



V Vertex (or node) attributes

e.g., node identity, number of neighbors

E Edge (or link) attributes and directions

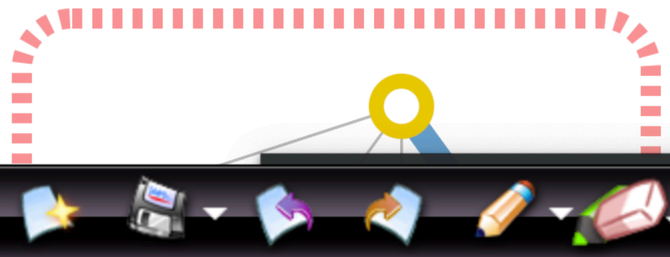
e.g., edge identity, edge weight

U Global (or master node) attributes

e.g., number of nodes, longest path

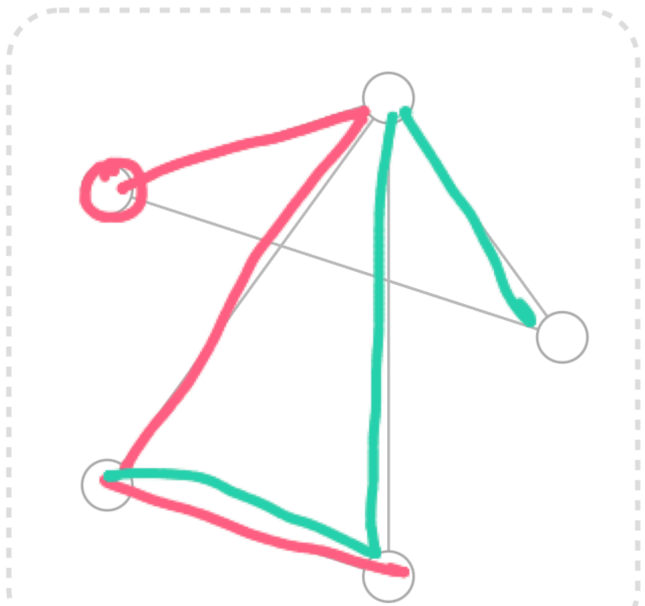
Three types of attributes we might find in a graph, hover over to highlight each attribute. Other types of graphs and attributes are explored in the Other types of graphs section.

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



Vertex (or node) embedding

of entities (*nodes*).



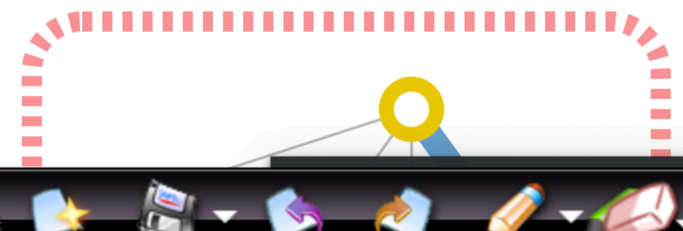
V Vertex (or node) attributes
e.g., node identity, number of neighbors

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

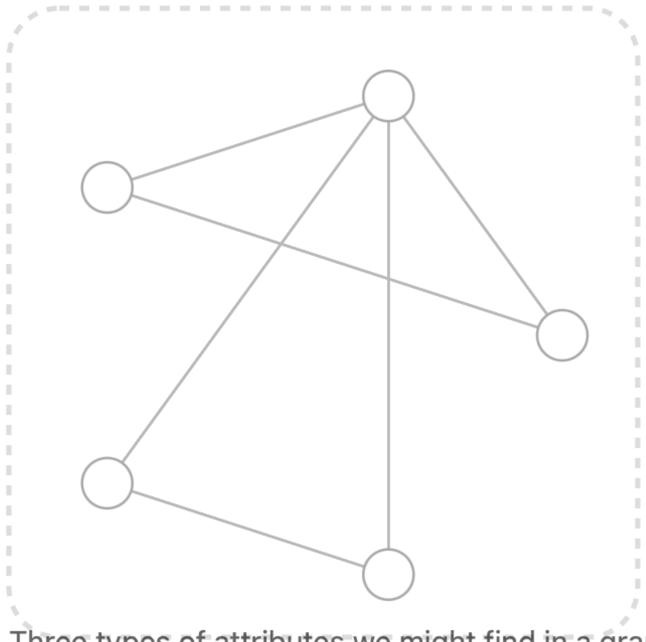
Three types of attributes we might find in a graph, hover over to highlight each attribute. Other types of graphs and attributes are explored in the Other types of graphs section.

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



Vertex (or node) embedding

of entities (*nodes*).



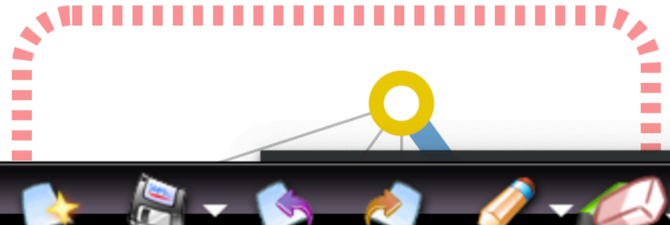
✓ **V** Vertex (or node) attributes
e.g., node identity, number of neighbors

✓ **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

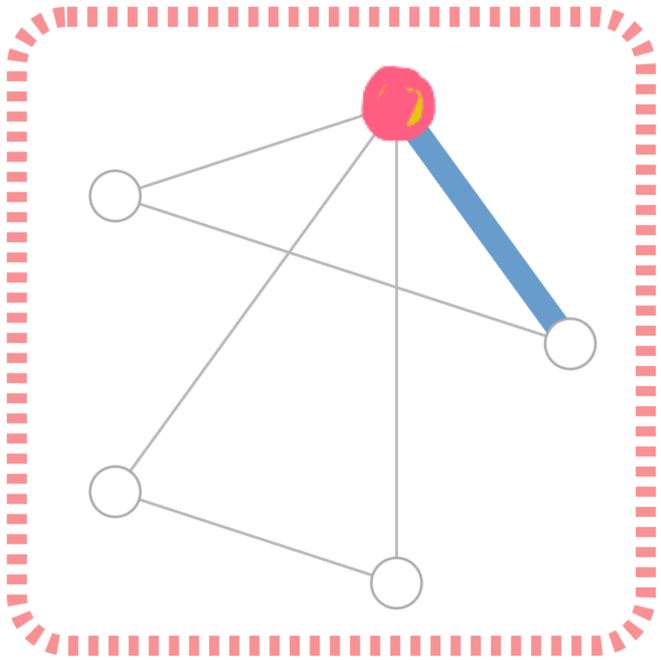
Three types of attributes we might find in a graph, hover over to highlight each attribute. Other types of graphs and attributes are explored in the Other types of graphs section.

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



Vertex (or node) embedding

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

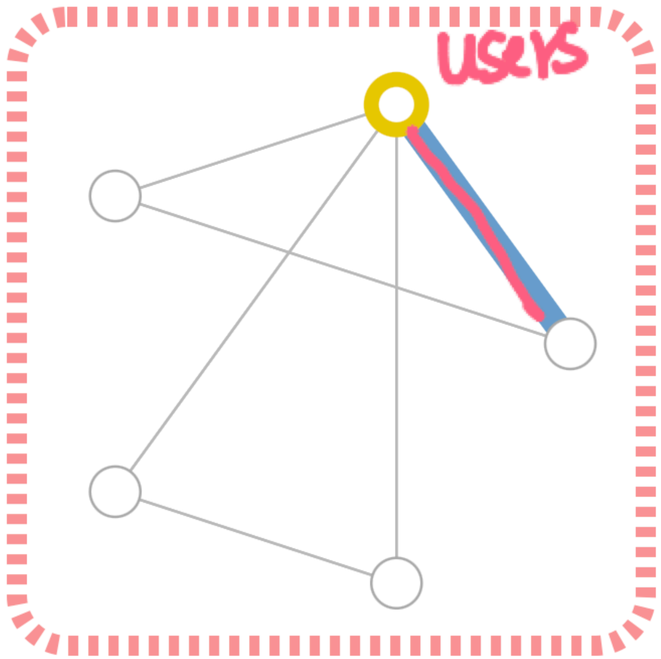


Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right).

We can additionally specialize graphs by associating directionality to edges (*directed, undirected*).

Directed edge

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



Vertex (or node) embedding

Edge (or link) attributes and embedding

Global (or master node) embedding

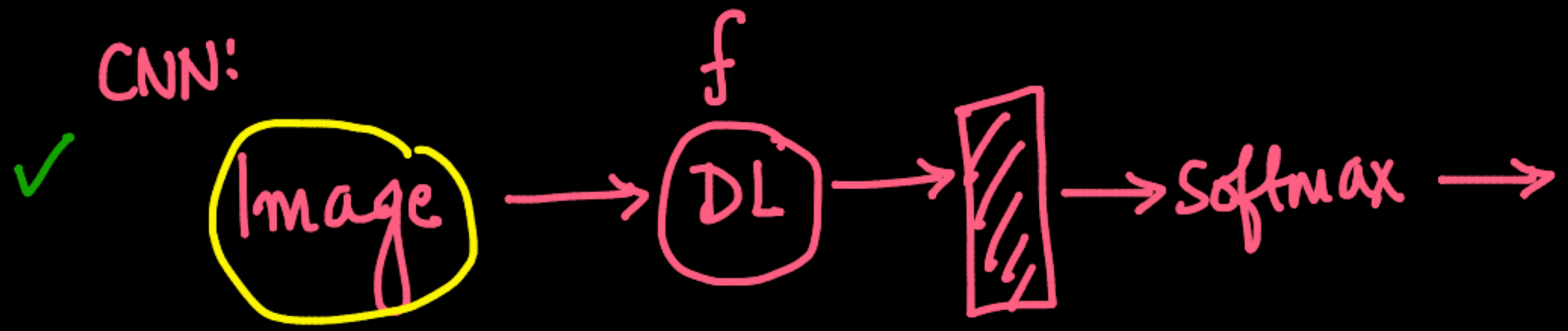
Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right).

We can additionally specialize graphs by associating directionality to edges (*directed, undirected*).

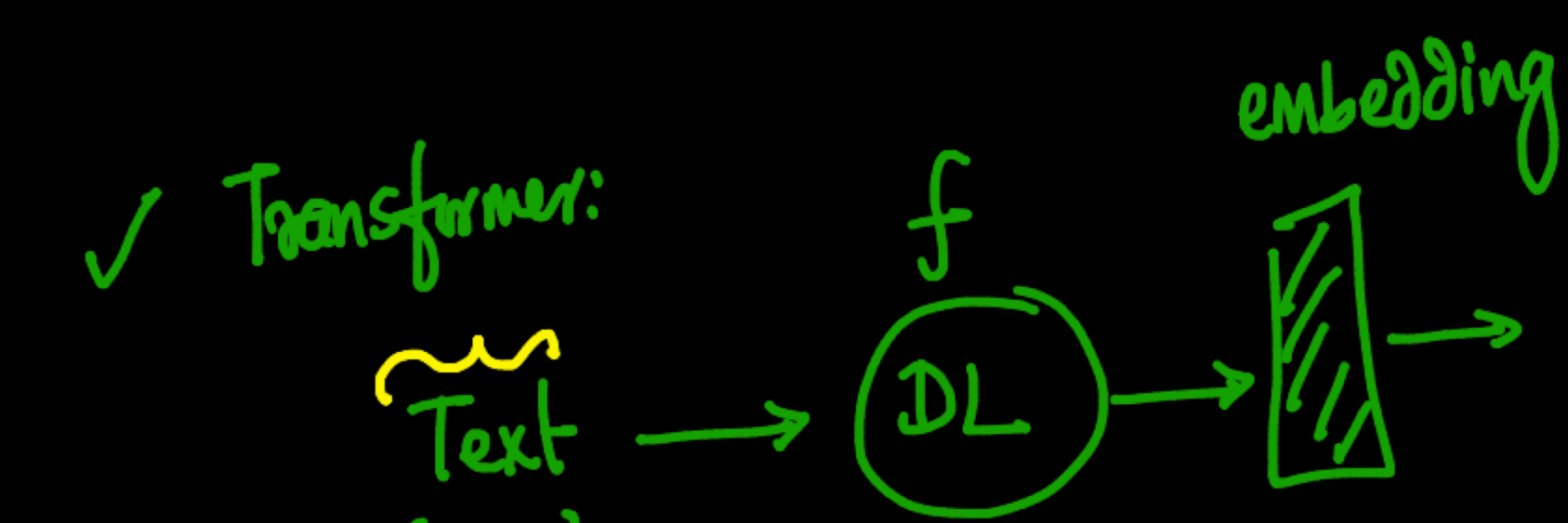
Directed edge

CNN:

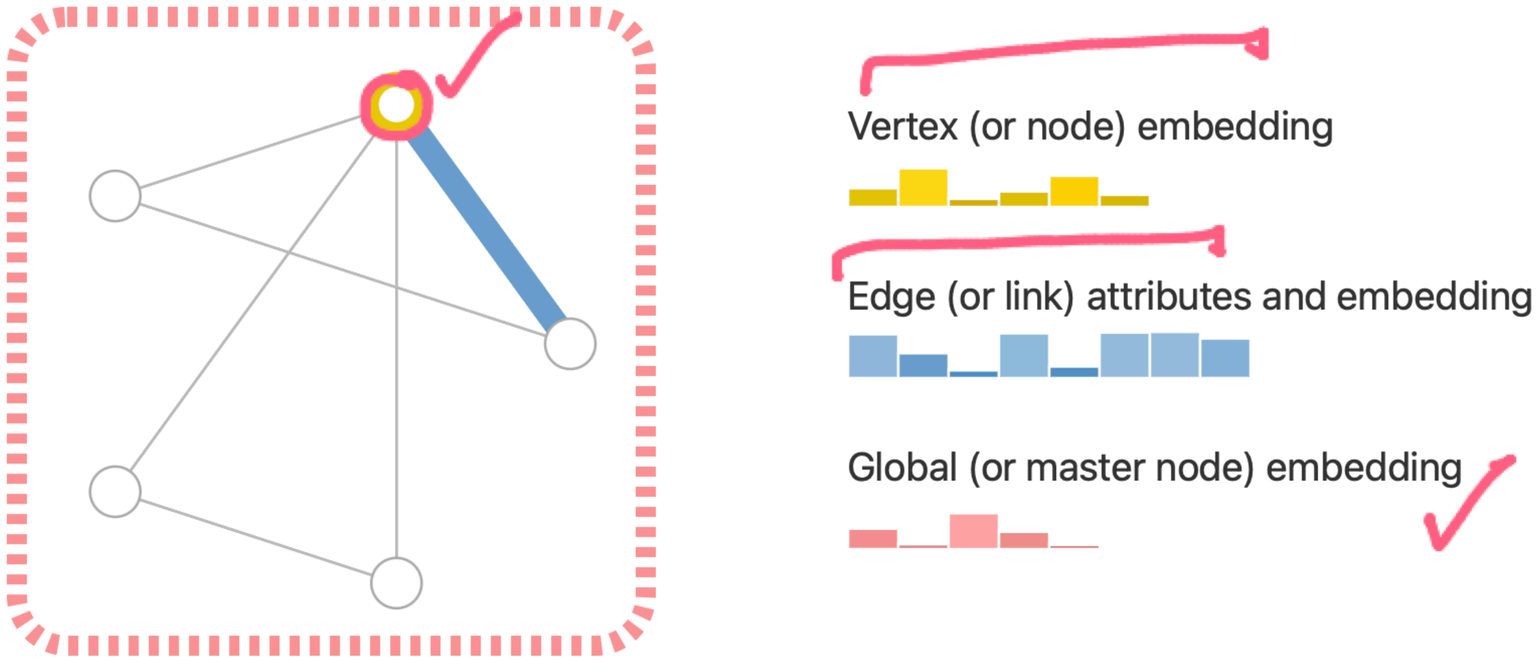
✓



✓ Transformer:



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



Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right).

We can additionally specialize graphs by associating directionality to edges (*directed, undirected*).

Directed edge

A way of visualizing the connectivity of a graph is through its *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 of $n_{nodes} \times n_{nodes}$ with an entry if two nodes share an edge. Note that each of these three representations below are different views of the same piece of data.

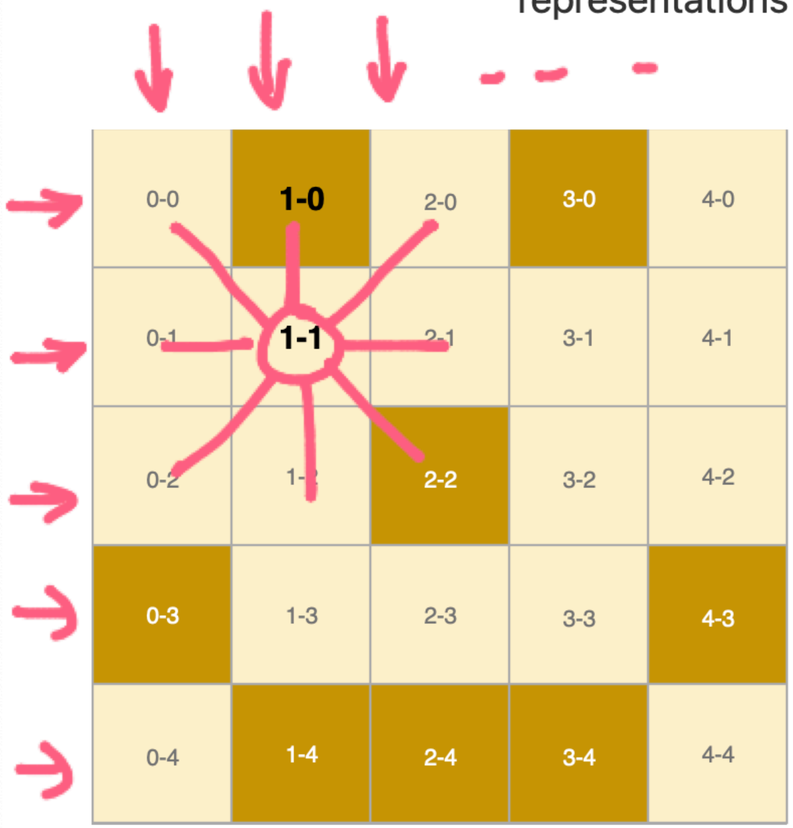
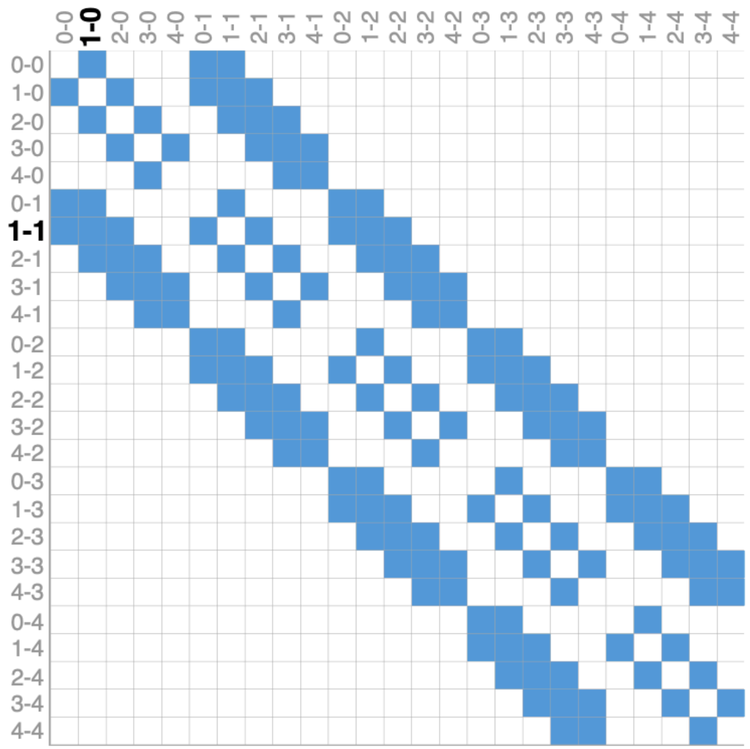
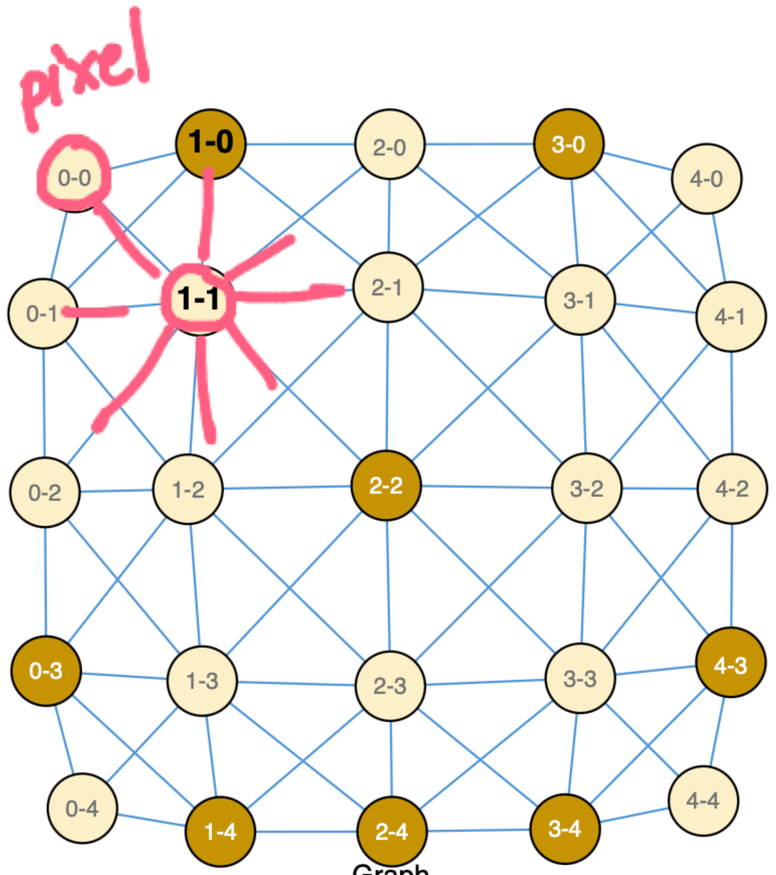


Image Pixels



Adjacency Matrix

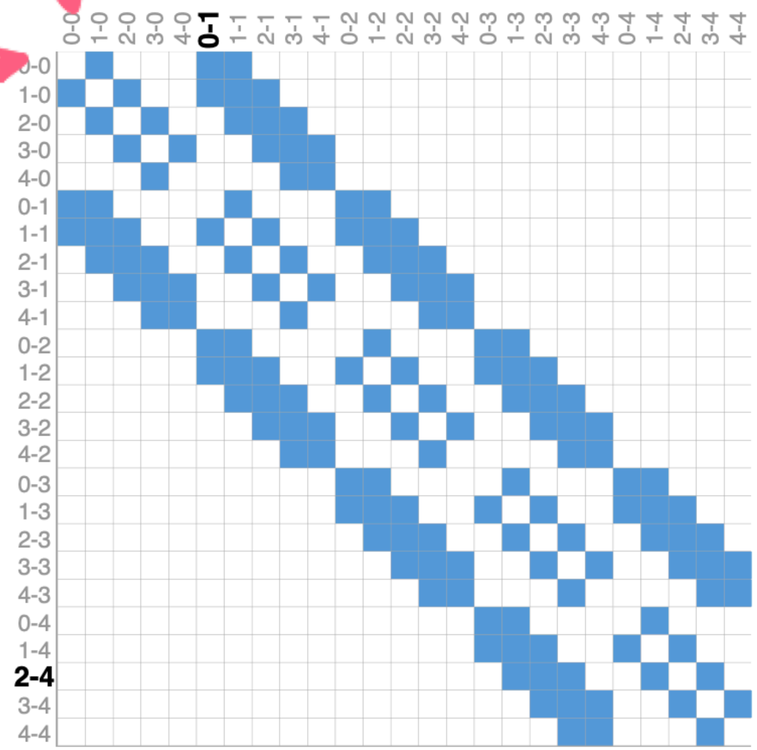


Graph

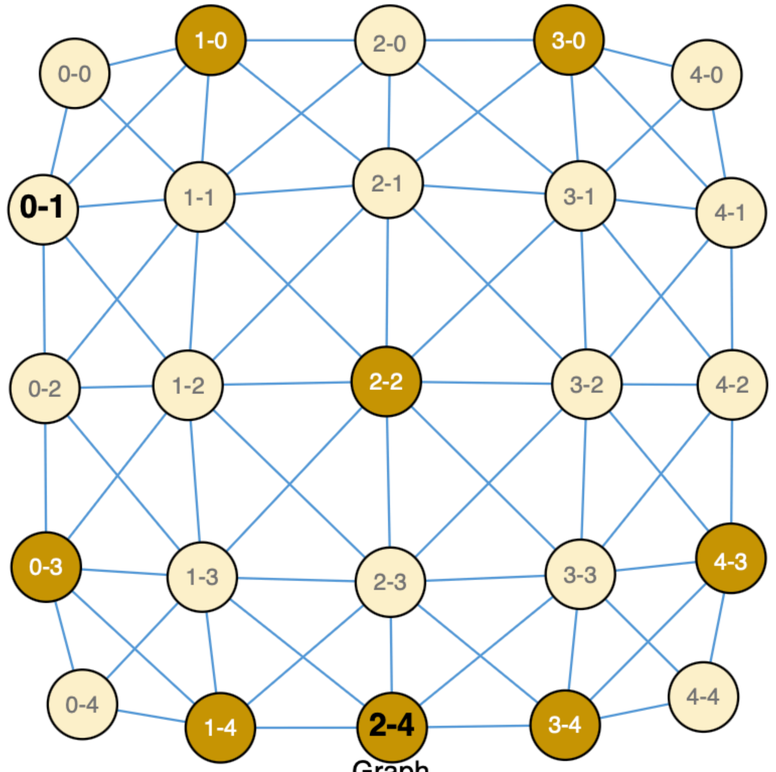
Click on an image pixel to toggle its value, and see how the graph representation changes.

A way of visualizing the connectivity of a graph is through its *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 of $n_{nodes} \times n_{nodes}$ with an entry if two nodes share an edge. Note that each of these three representations below are different views of the same piece of data.

Image Pixels



Adjacency Matrix



Graph

Click on an image pixel to toggle its value, and see how the graph representation changes.

A way of visualizing the connectivity of a graph is through its *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 of $n_{nodes} \times n_{nodes}$ with an entry if two nodes share an edge. Note that each of these three representations below are different views of the same piece of data.

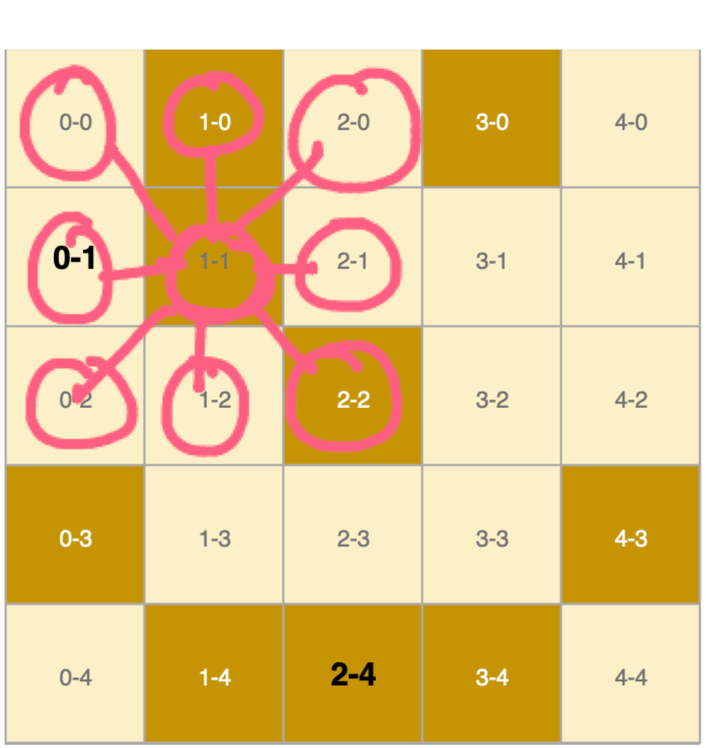
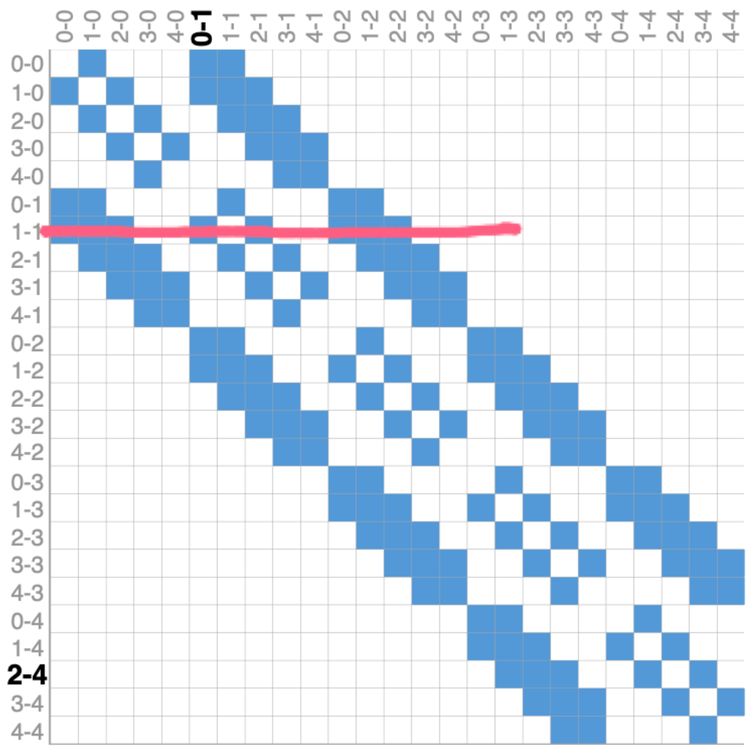
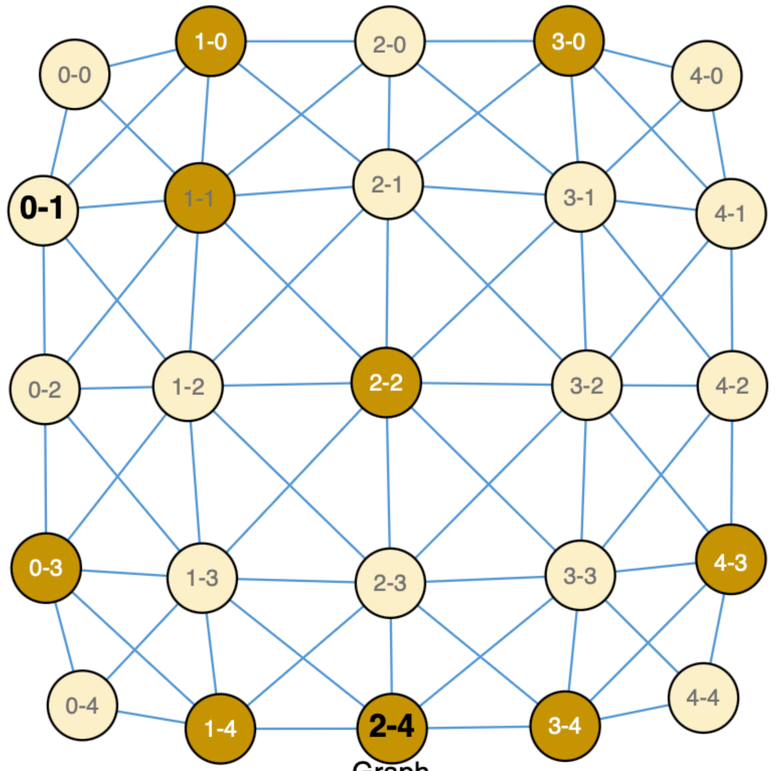


Image Pixels



Adjacency Matrix



Graph

Click on an image pixel to toggle its value, and see how the graph representation changes.

A way of visualizing the connectivity of a graph is through its *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 of $n_{nodes} \times n_{nodes}$ with an entry if two nodes share an edge. Note that each of these three representations below are different views of the same piece of data.

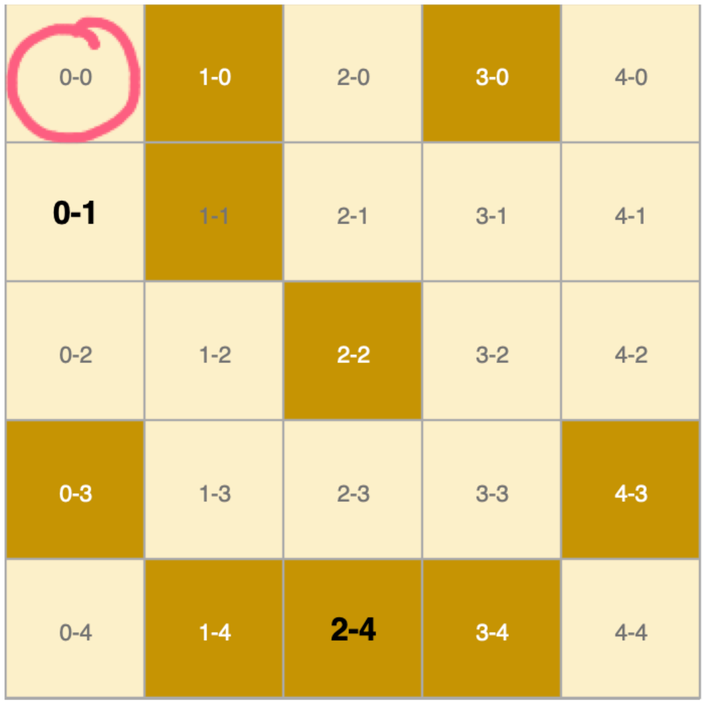
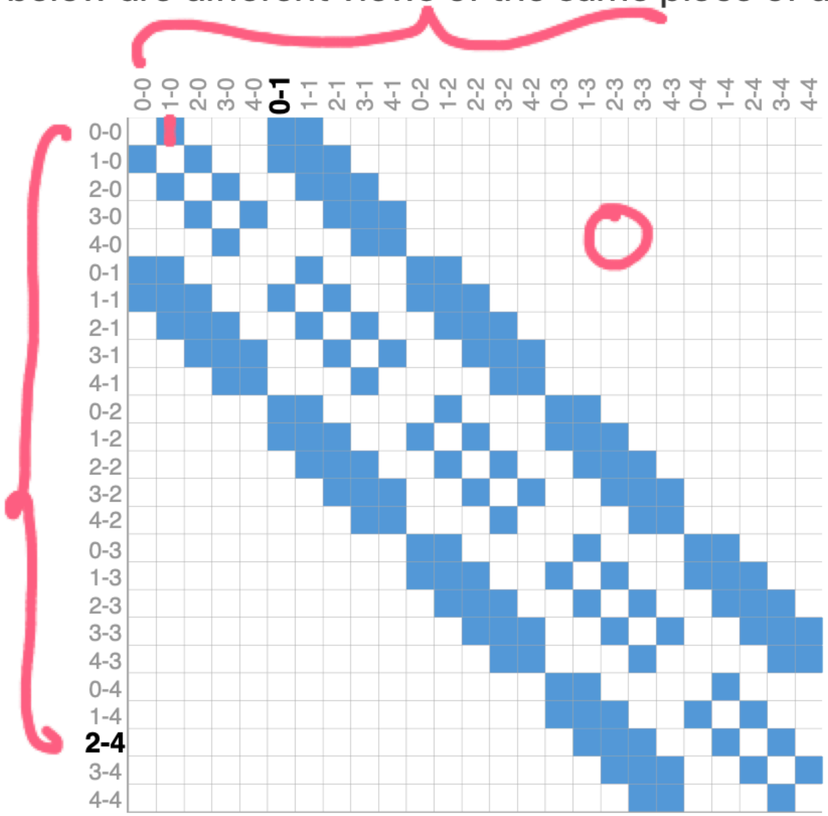
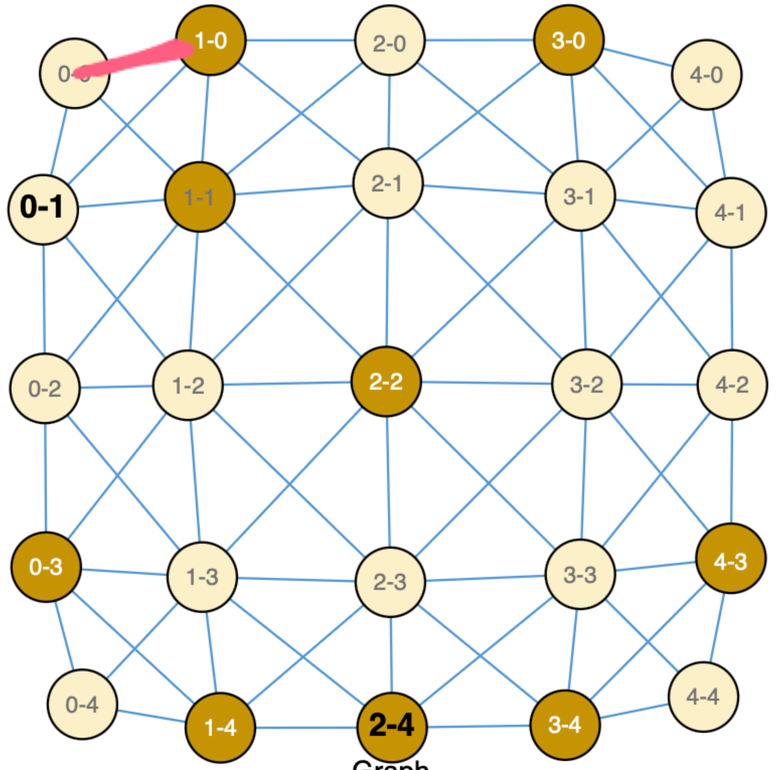


Image Pixels



Adjacency Matrix



Graph

Click on an image pixel to toggle its value, and see how the graph representation changes.

A way of visualizing the connectivity of a graph is through its *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 of $n_{nodes} \times n_{nodes}$ with an entry if two nodes share an edge. Note that each of these three representations below are different views of the same piece of data.

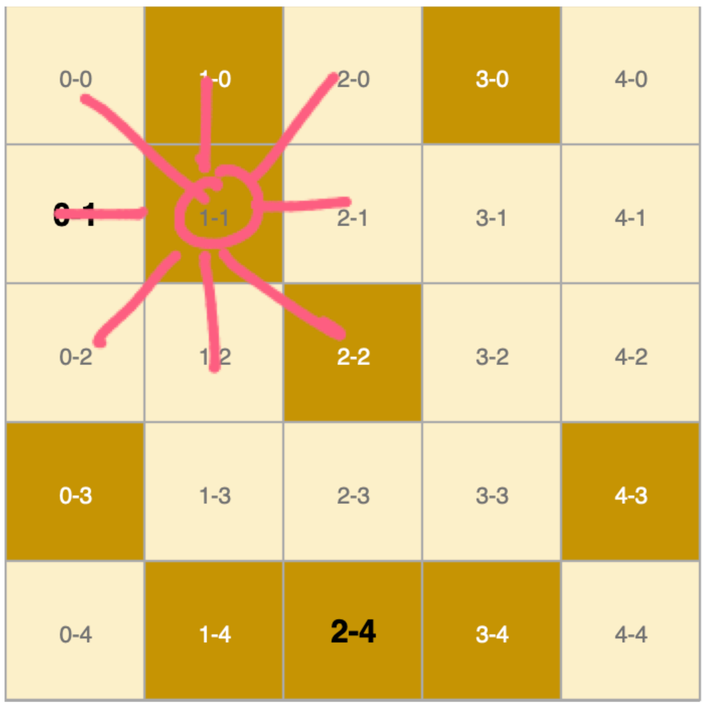
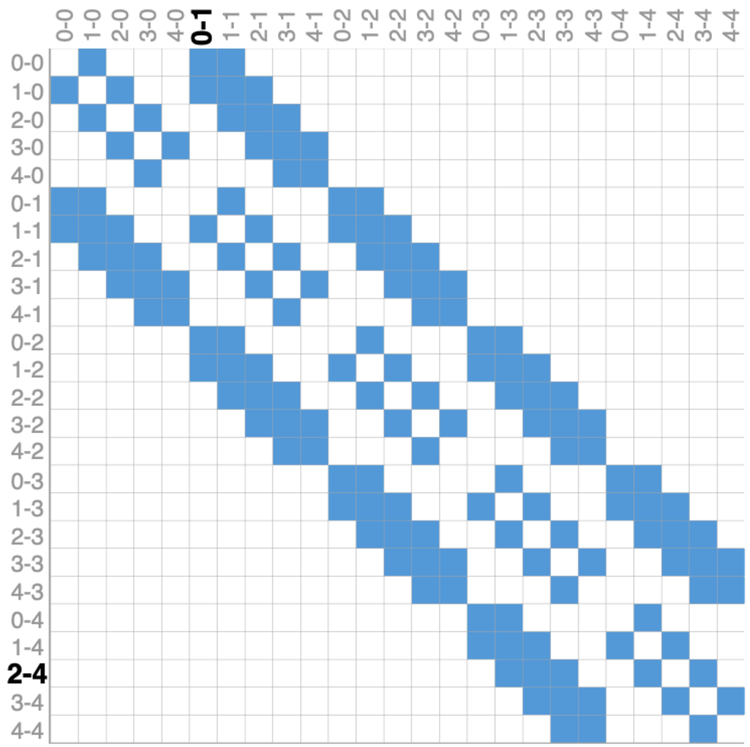
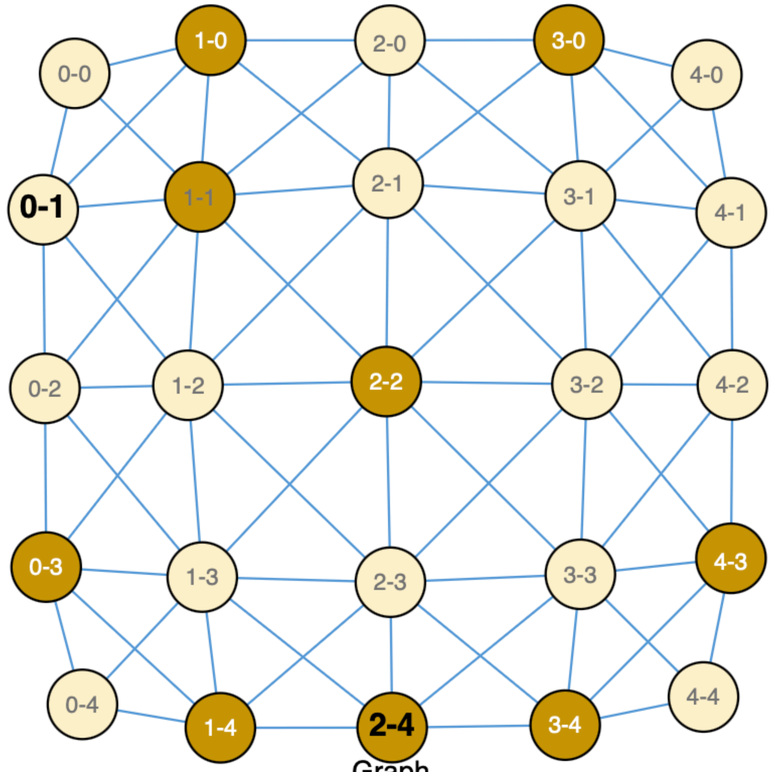


Image Pixels



Adjacency Matrix



Graph

Click on an image pixel to toggle its value, and see how the graph representation changes.

A way of visualizing the connectivity of a graph is through its *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 of $n_{nodes} \times n_{nodes}$ with an entry if two nodes share an edge. Note that each of these three representations below are different views of the same piece of data.

Grid

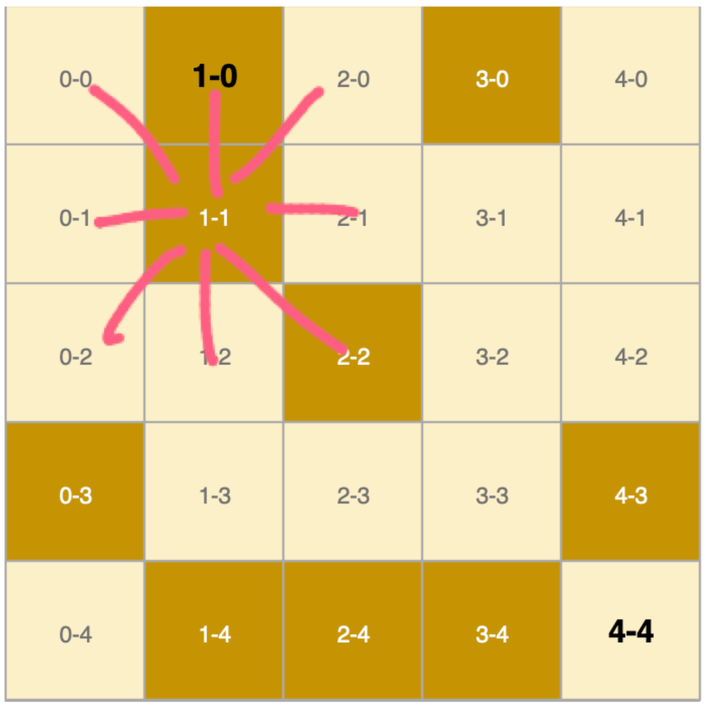
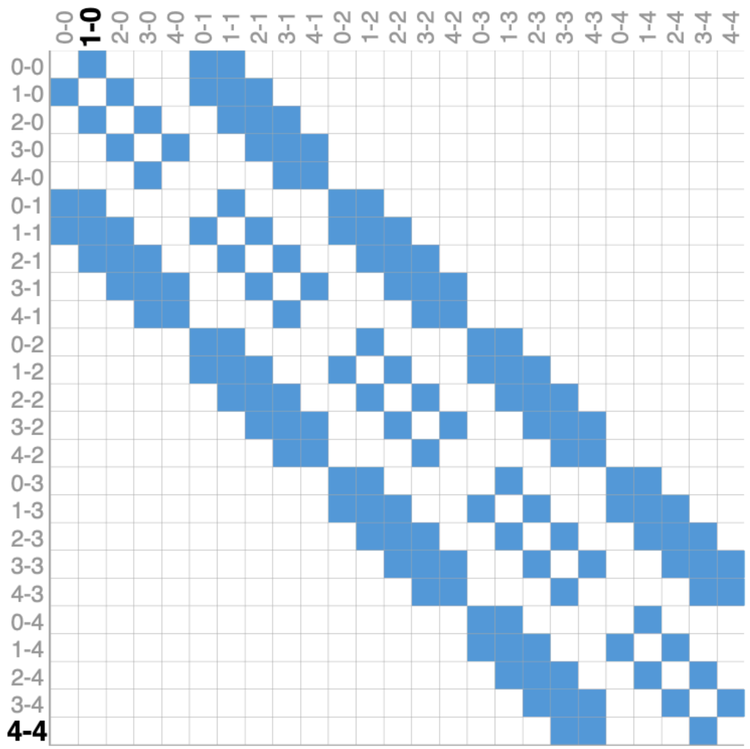
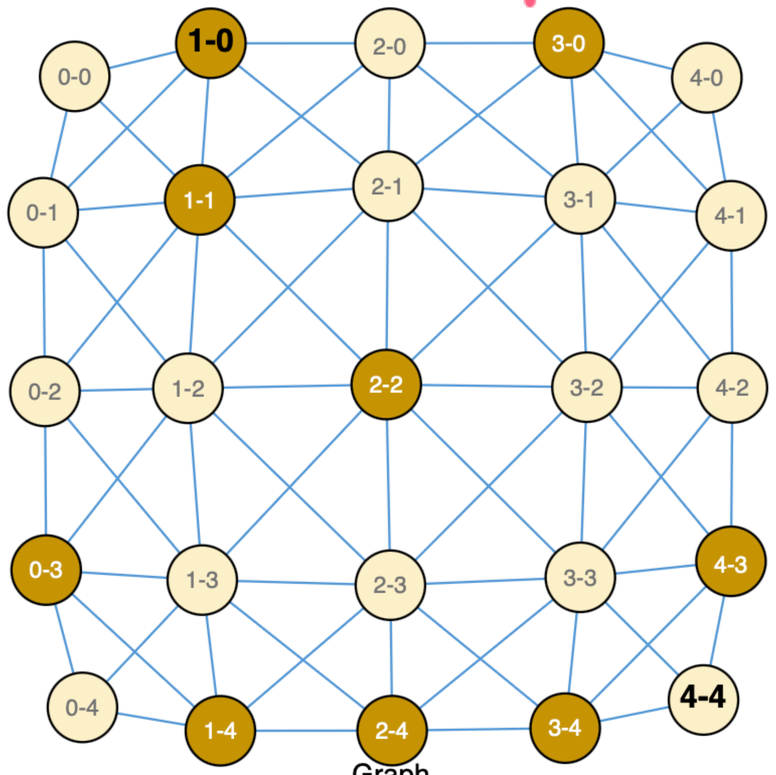


Image Pixels



Adjacency Matrix

Graph



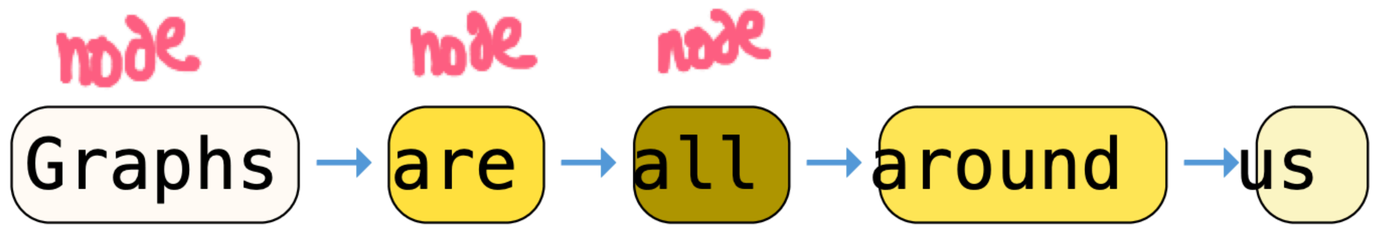
Graph

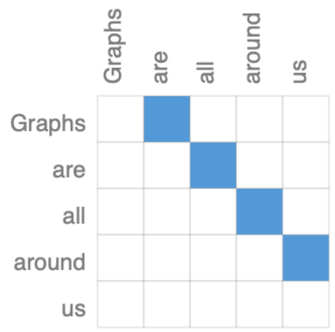
Click on an image pixel to toggle its value, and see how the graph representation changes.

Text as graphs

Attention : Special GNN

We can digitize text by associating indices to each character, word, or token, and representing text as a sequence of these indices. This creates a simple directed graph, where each character or index is a node and is connected via an edge to the node that follows it.



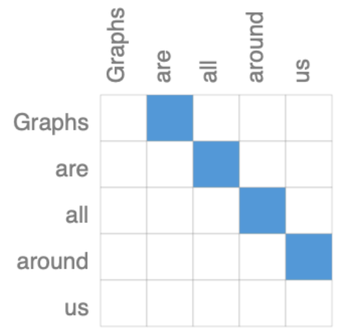


Linear Graph
Line Graph

Edit the text above to see how the graph representation changes.

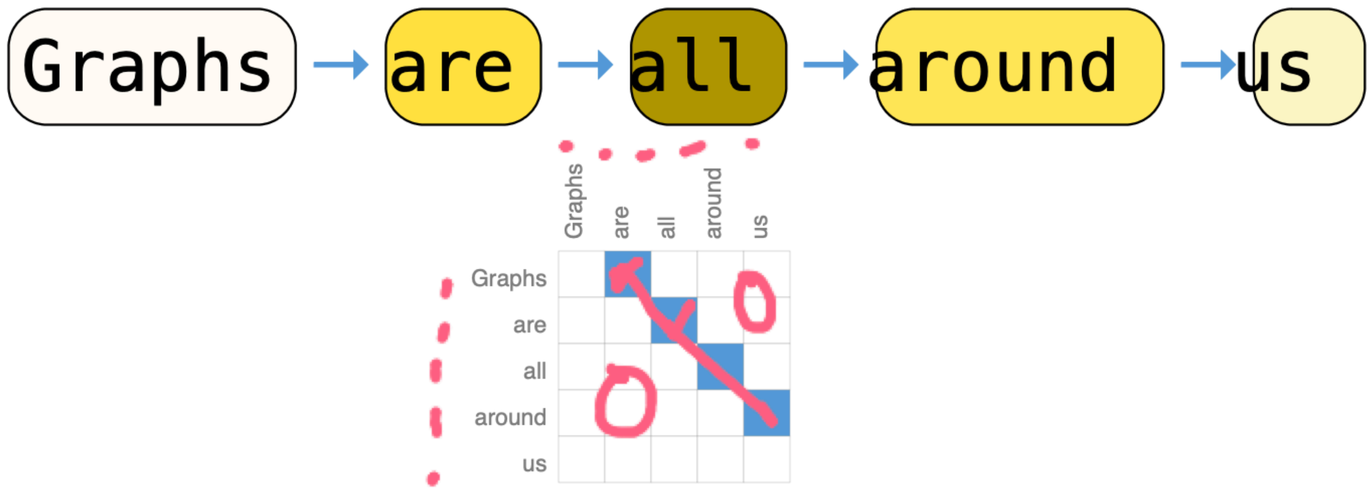
Text as graphs

We can digitize text by associating indices to each character, word, or token, and representing text as a sequence of these indices. This creates a simple directed graph, where each character or index is a node and is connected via an edge to the node that follows it.



Edit the text above to see how the graph representation changes.

text as a sequence of these indices. This creates a simple directed graph, where each character or index is a node and is connected via an edge to the node that follows it.

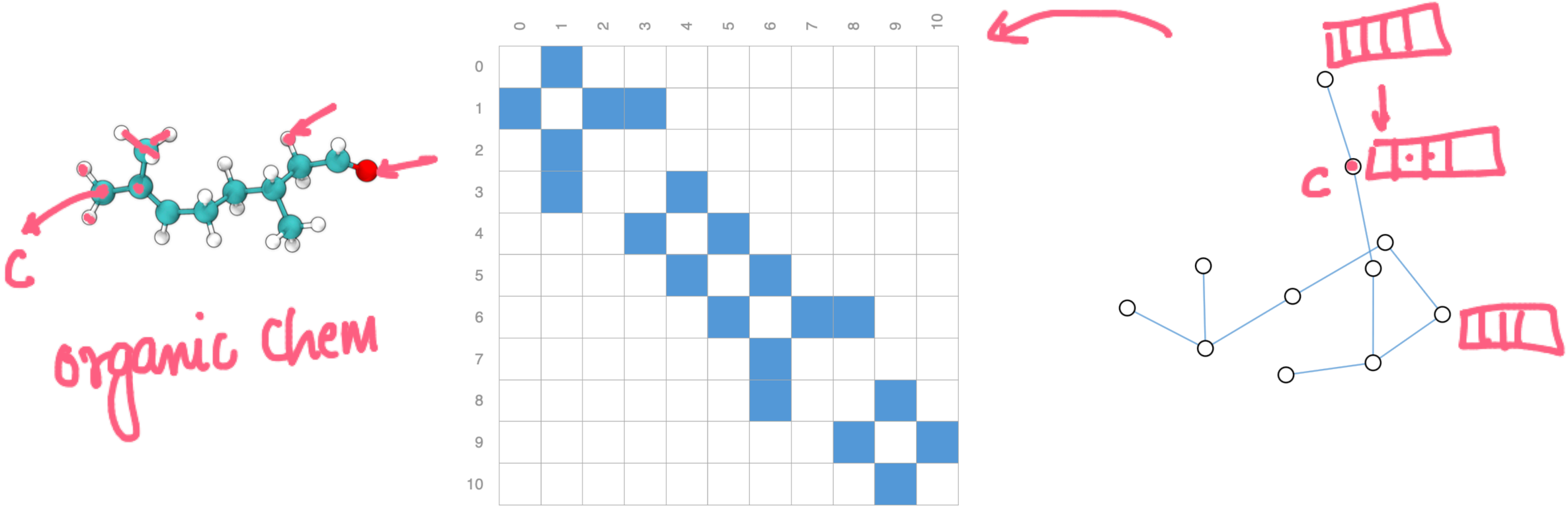


Edit the text above to see how the graph representation changes.

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. For instance, images have a banded structure in their adjacency matrix because all nodes (pixels) are connected in a grid. The adjacency matrix for text is just a diagonal line, because ea

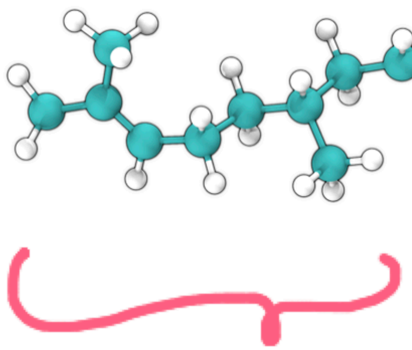
This representation (a sequence of character tokens) refers to the way text is often represented RNNs; other models, such as Transformers, can be considered to view text a

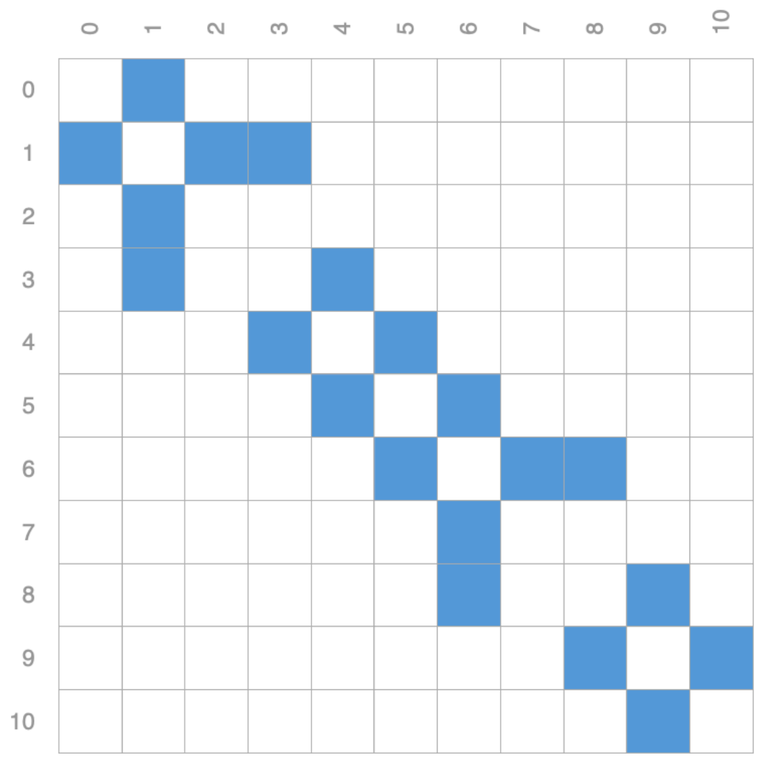
and bonds have different distances (e.g. single-bonds, double-bonds). It's a very convenient and common abstraction to describe this 3D object as a graph, where nodes are atoms and edges are covalent bonds. [8] Here are two common molecules, and their associated graphs.

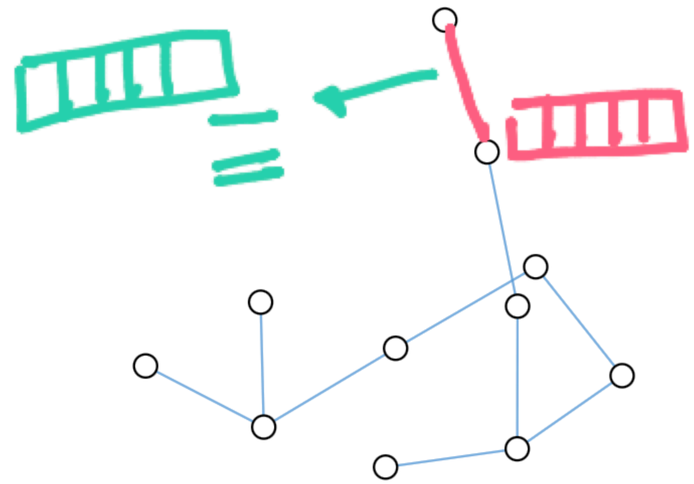


(Left) 3d representation of the Citronellal molecule (Center) Adjacency matrix of the bonds in the molecule (Right) Graph representation of the molecule.

and bonds have different distances (e.g. single-bonds, double-bonds). 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. [8] Here are two common molecules, and their associated graphs.

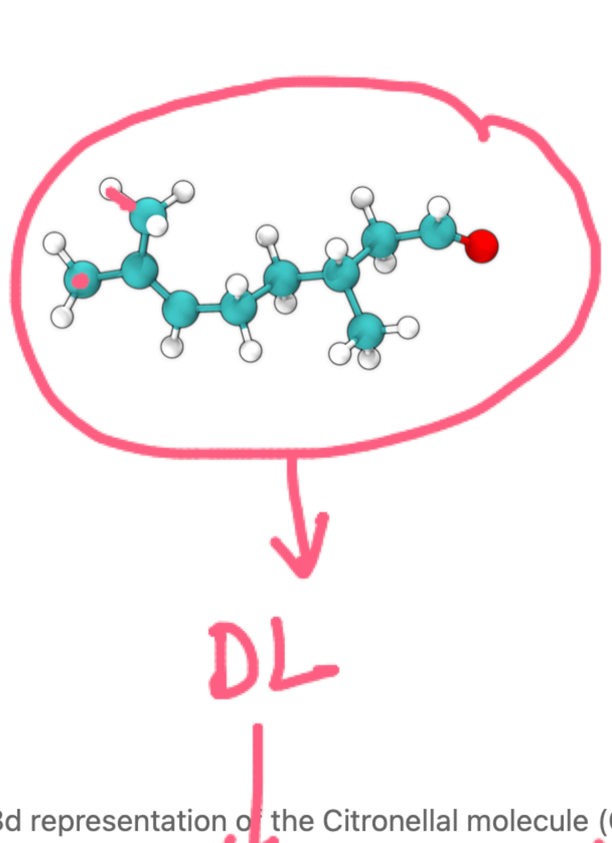


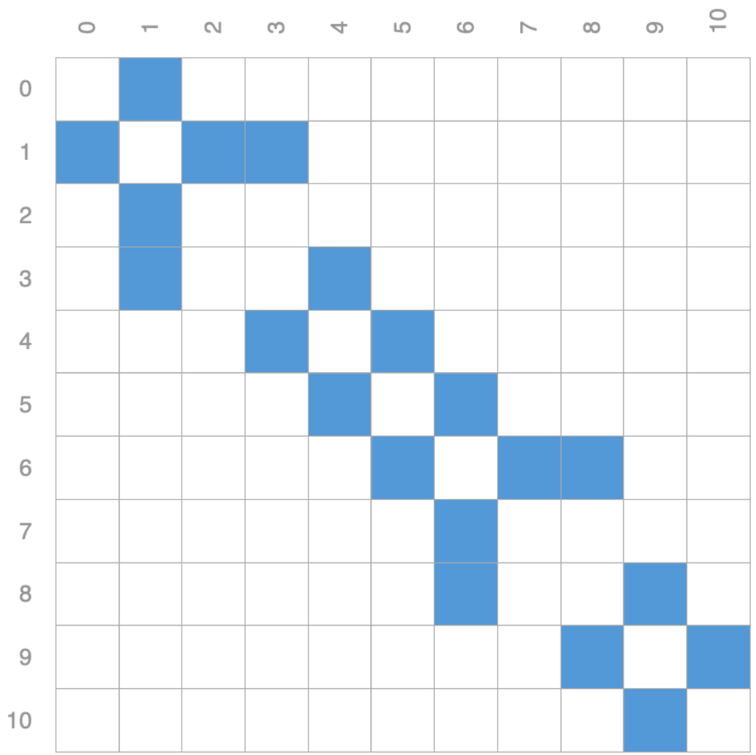


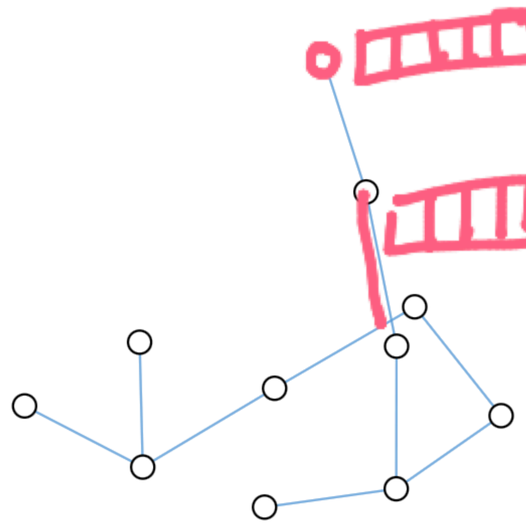


(Left) 3d representation of the Citronellal molecule (Center) Adjacency matrix of the bonds in the molecule (Right) Graph representation of the molecule.

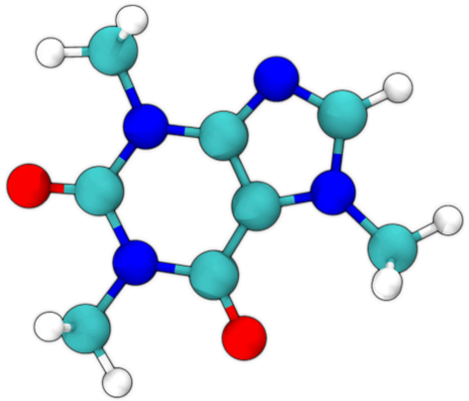
and bonds have different distances (e.g. single-bonds, double-bonds). It's a very convenient and common abstraction to describe this 3D object as a graph, where nodes are atoms and edges are covalent bonds. [8] Here are two common molecules, and their associated graphs.

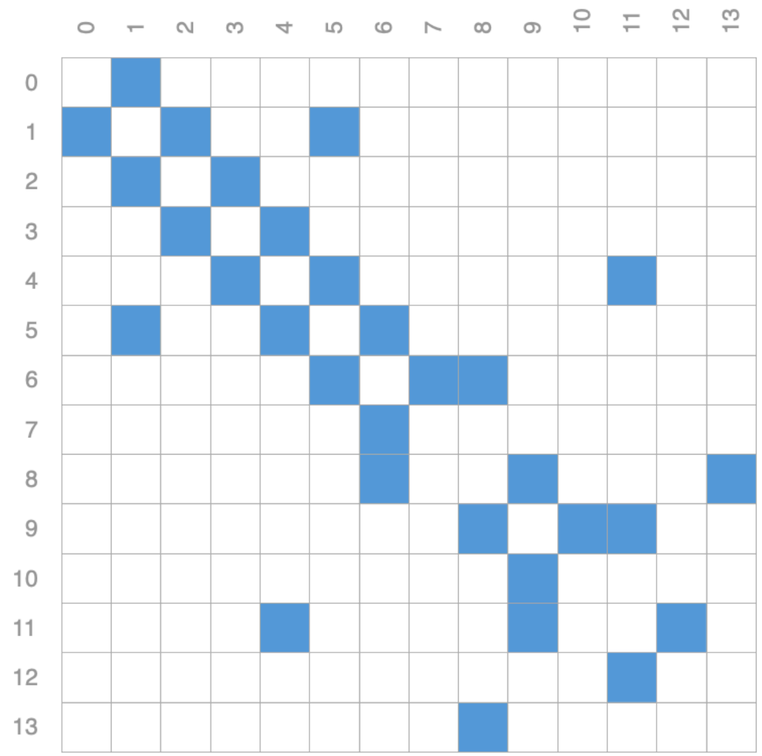


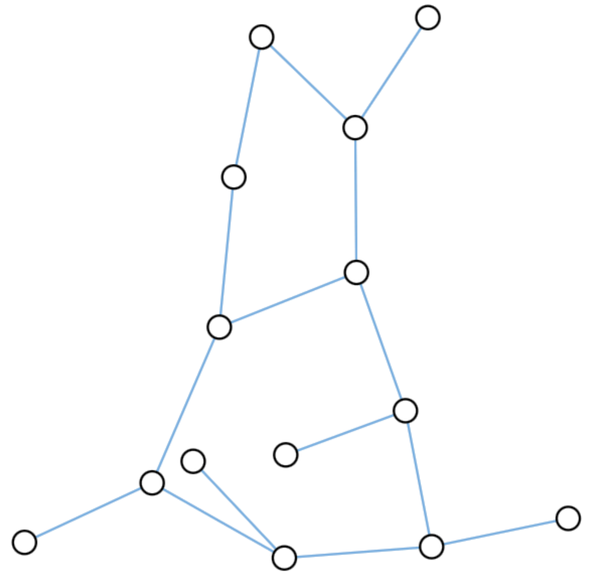




(Left) 3d representation of the Citronellal molecule (Center) Adjacency matrix of the bonds in the molecule (Right) Graph representation of the molecule.



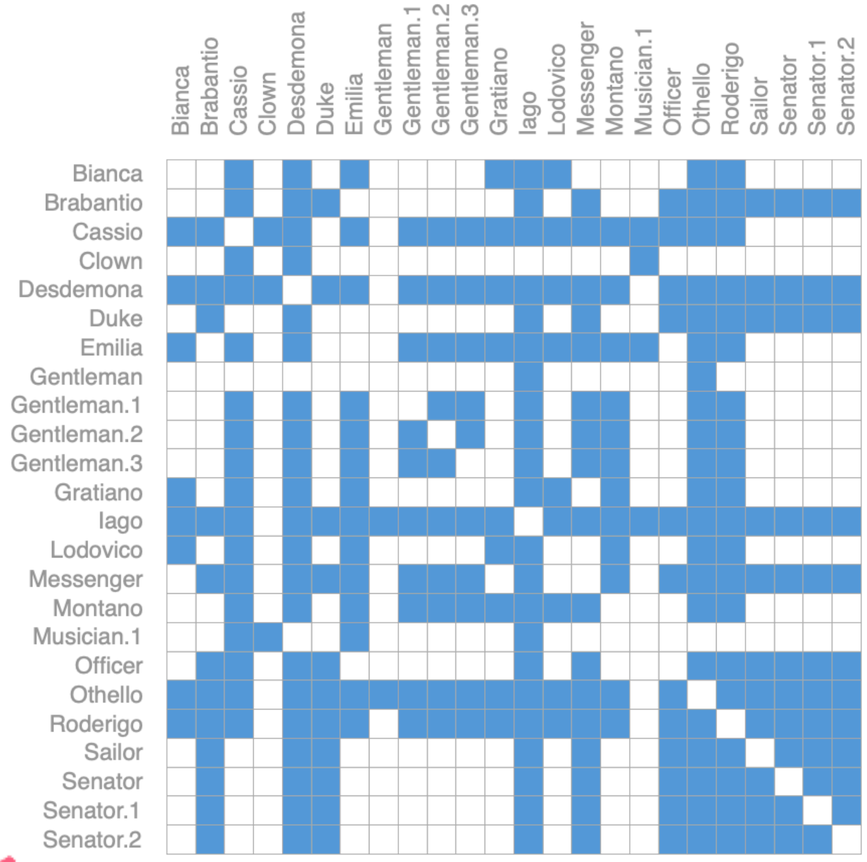


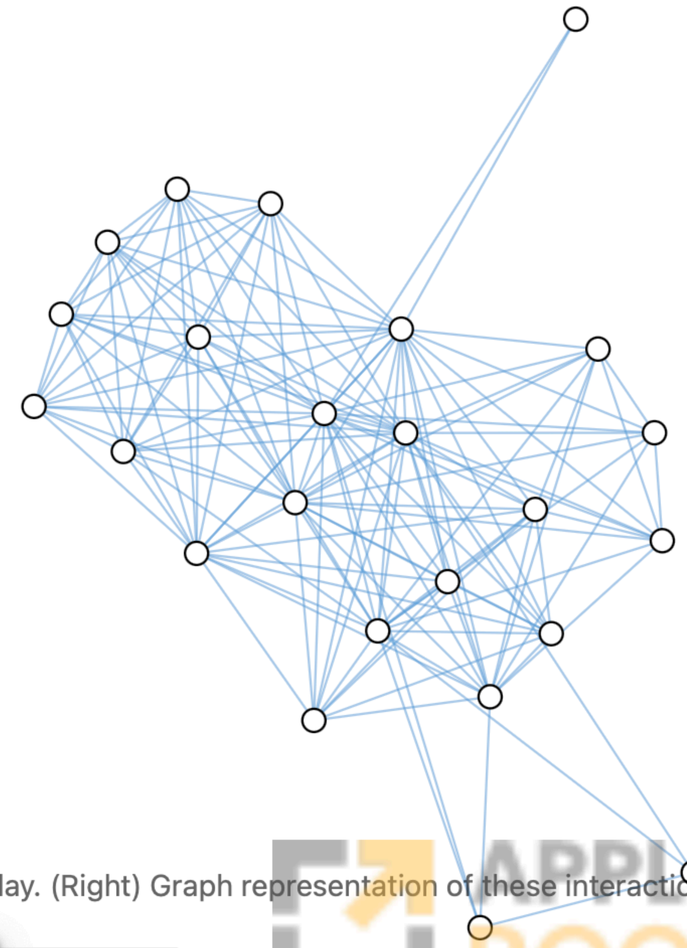


(Left) 3d representation of the Caffeine molecule (Center) Adjacency matrix of the bonds in the molecule (Right) Graph representation of the molecule.

Social networks as graphs. 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.

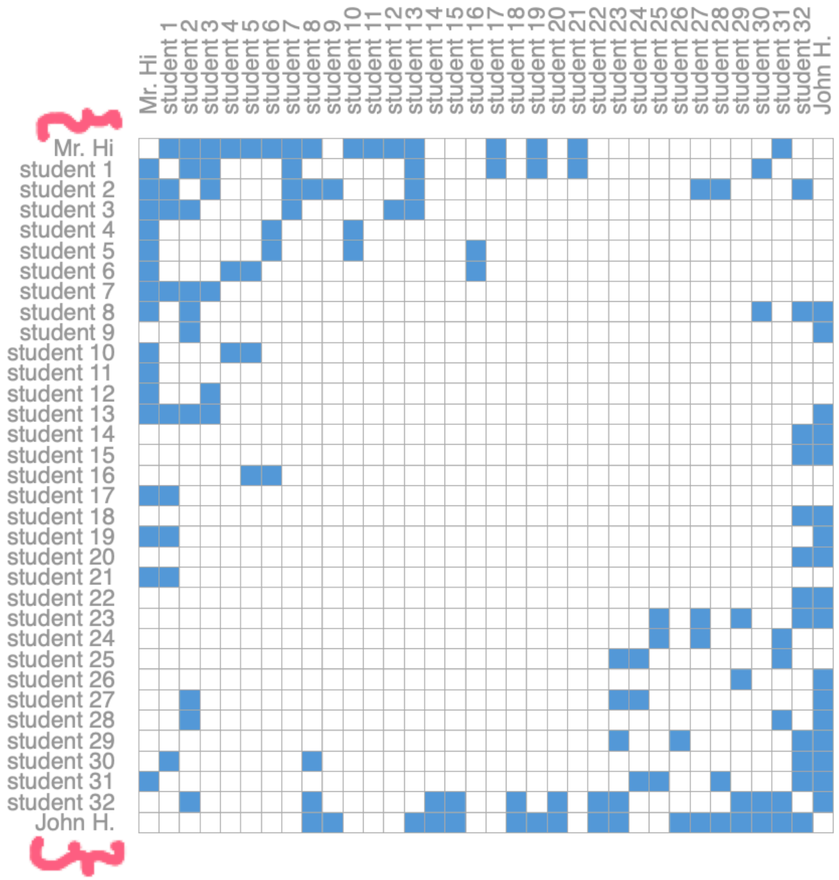
Social networks as graphs. Social networks are tools to study patterns in collective behaviour of people, institutions and organizations. We can build a graph representing groups of people by modelling individuals as nodes, and their relationships as edges.

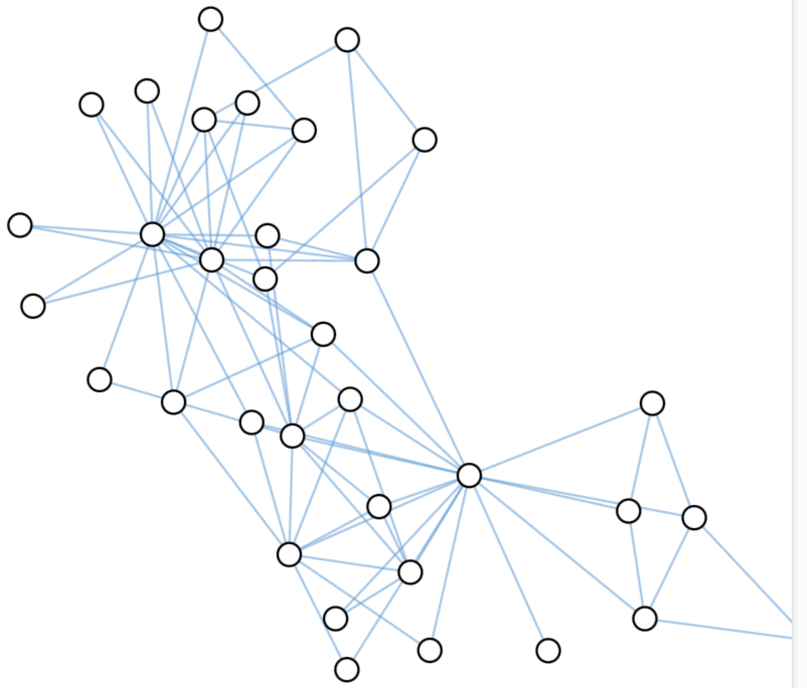




(Left) Image of a scene from the play “Othello”. (Center) Adjacency matrix of the interaction between characters in the play. (Right) Graph representation of these interactions.

Unlike image and text data, social networks do not have identical adjacency matrices.



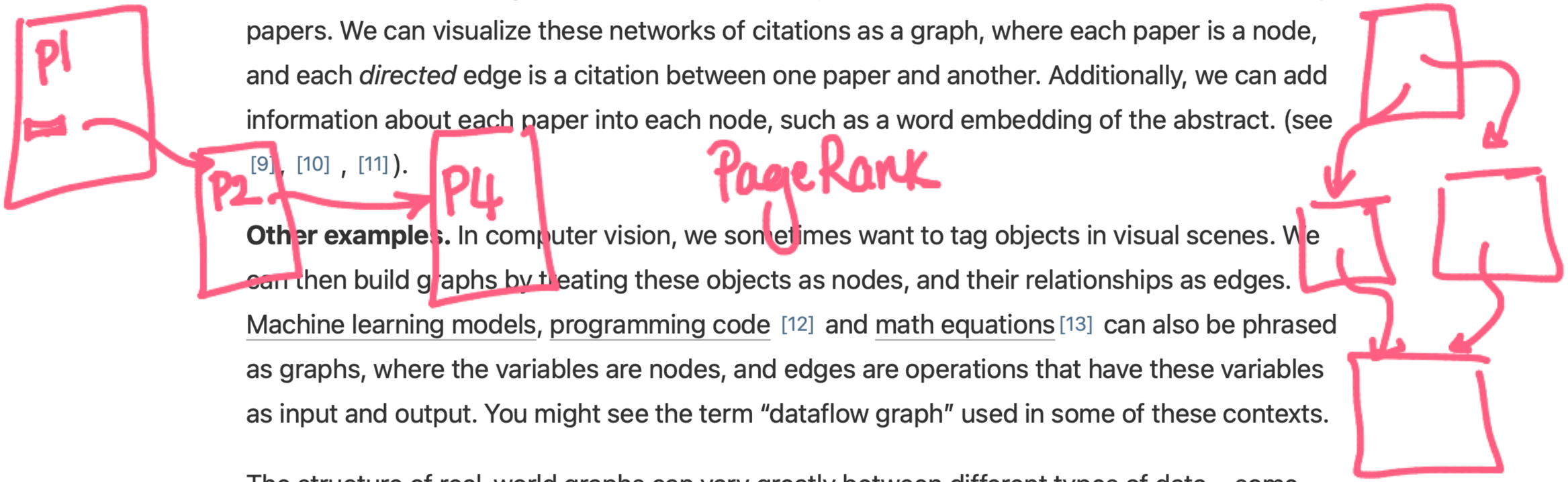


(Left) Image of karate tournament. (Center) Adjacency matrix of the interaction between people in a karate club. (Right) Graph representation of these interactions.

Citation networks as graphs. Scientists routinely cite other scientists' work when publishing papers. We can visualize these networks of citations as a graph, where each paper is a node, an[illegible] we can add

(Left) Image of karate tournament. (Center) Adjacency matrix of the interaction between people in a karate club. (Right) Graph representation of these interactions.

Citation networks as graphs. Scientists routinely cite other scientists' work when publishing papers. We can visualize these networks of citations as a graph, where each paper is a node, and each *directed* edge is a citation between one paper and another. Additionally, we can add information about each paper into each node, such as a word embedding of the abstract. (see [9], [10], [11]).

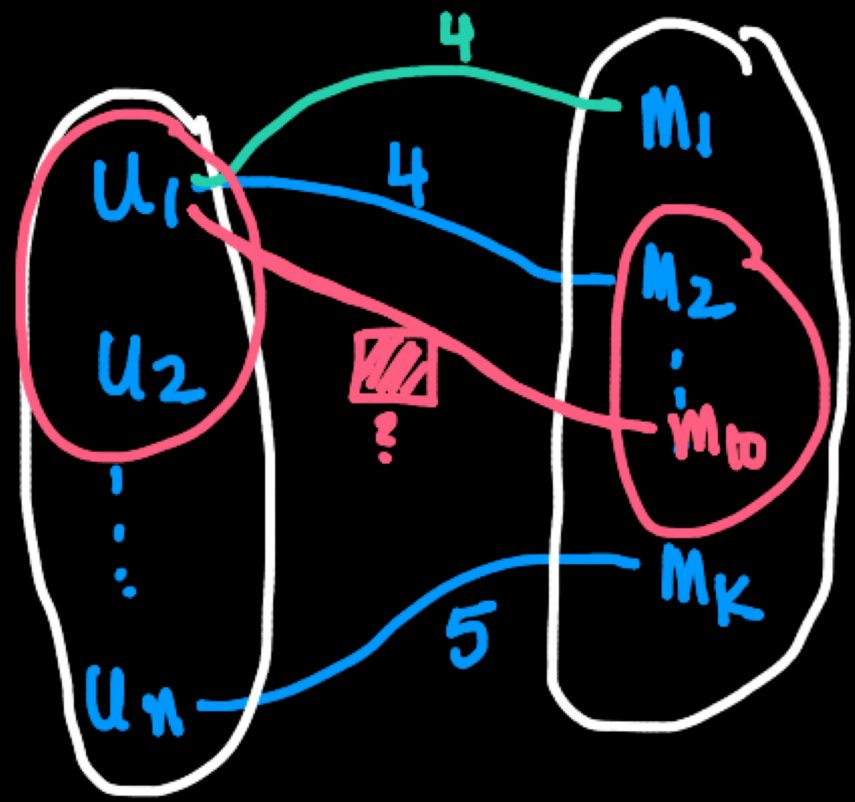


Other examples. In computer vision, we sometimes want to tag objects in visual scenes. We can then build graphs by treating these objects as nodes, and their relationships as edges. Machine learning models, programming code [12] and math equations [13] can also be phrased as graphs, where the variables are nodes, and edges are operations that have these variables as input and output. You might see the term "dataflow graph" used in some of these contexts.

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.

(Q) Rec-Sys (movie-rec) as a Graph problem:

(u, movies)



✓ { bi-partite graphs

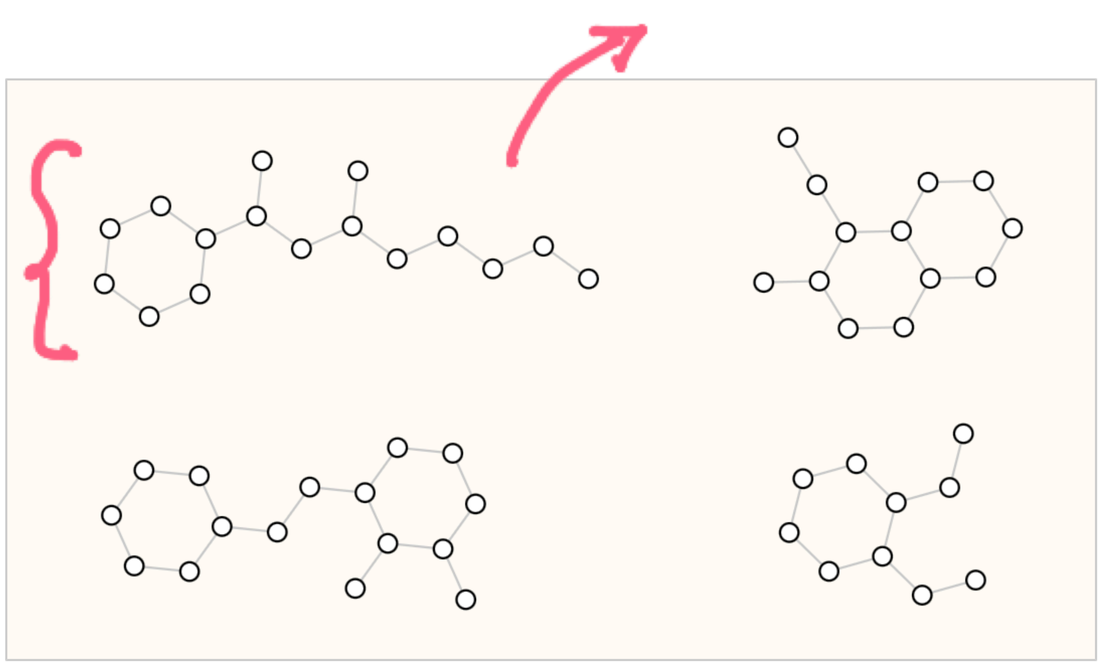
2-parts

edge-basic

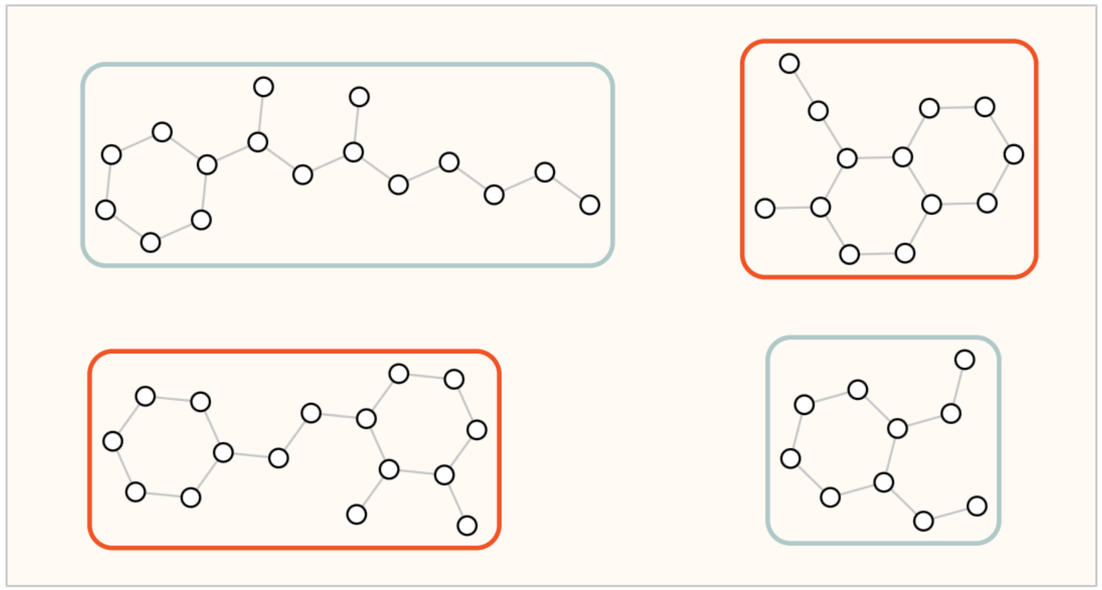
Graph-level task

U, V, E

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.



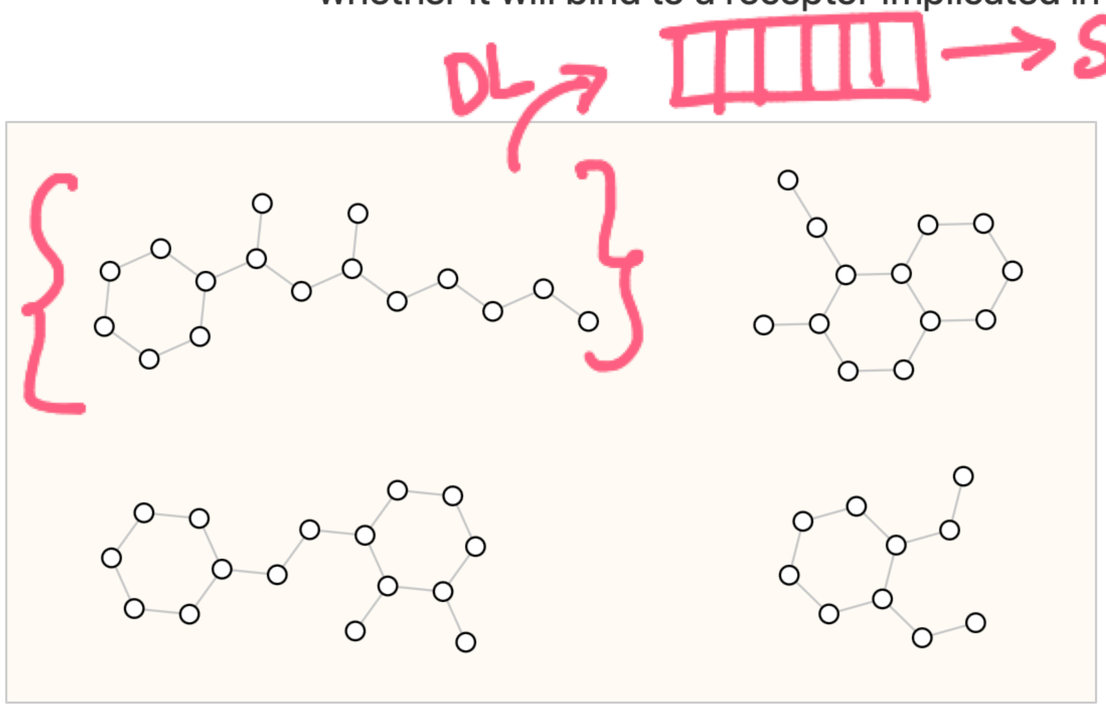
Input: graphs



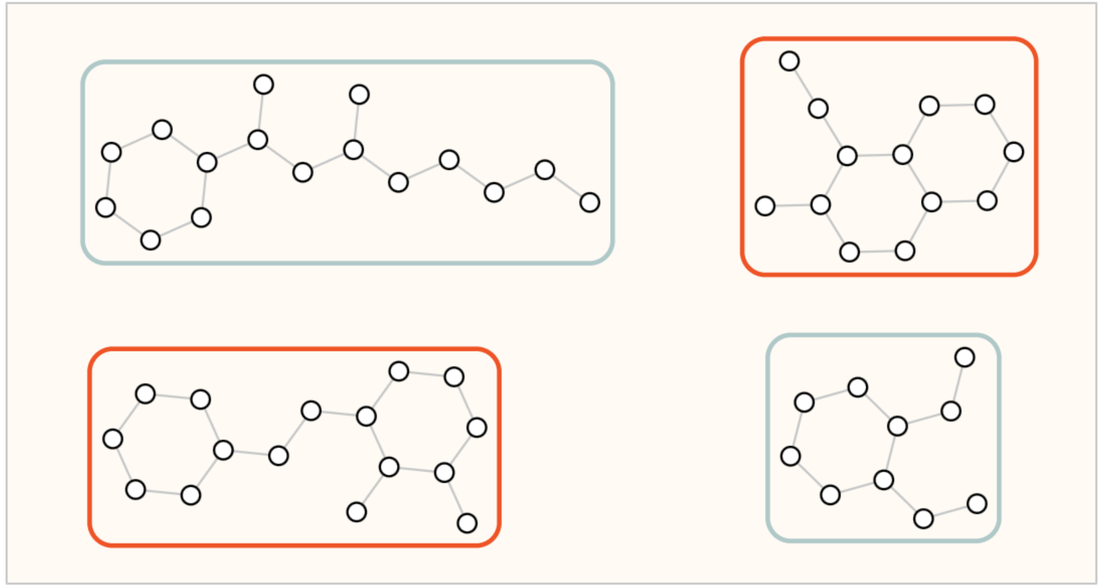
Output: labels for each graph, (e.g., "does the graph contain two rings?")

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.

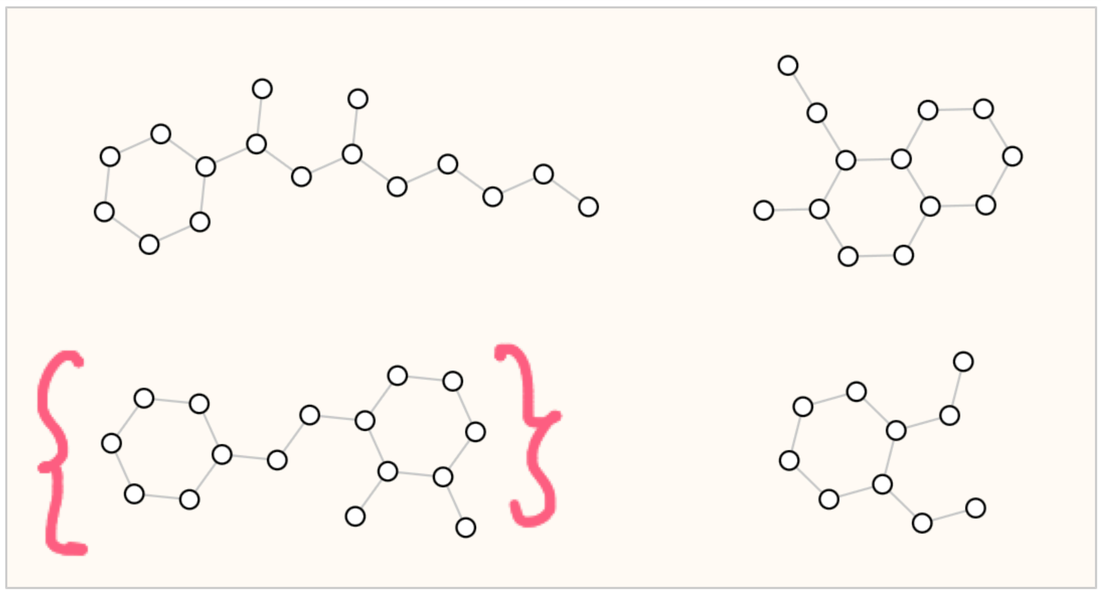


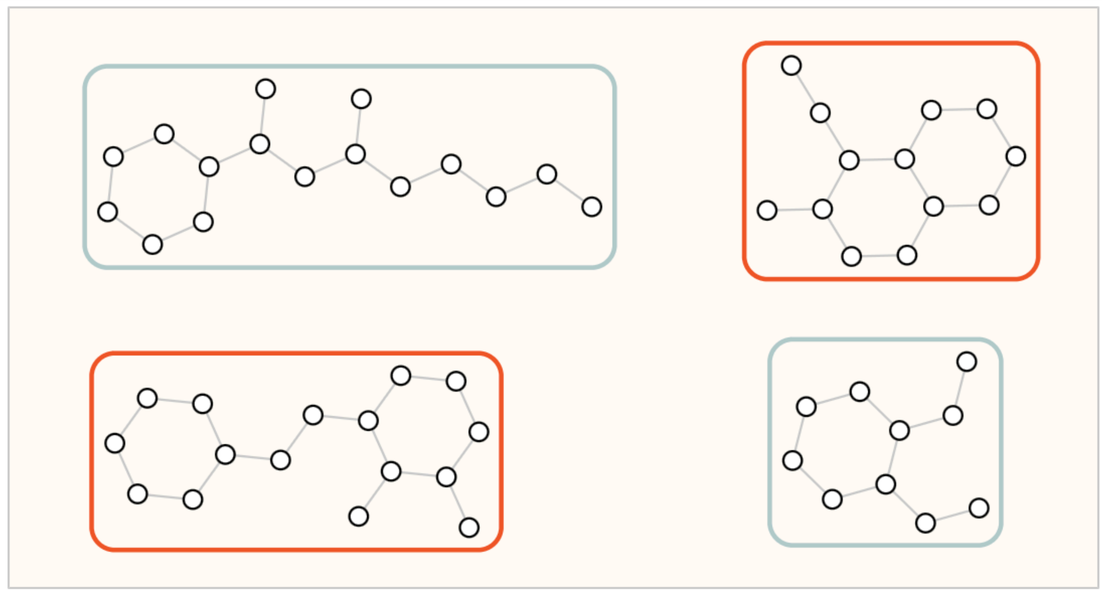
Input: graphs



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

Th we want to





Input: graphs

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

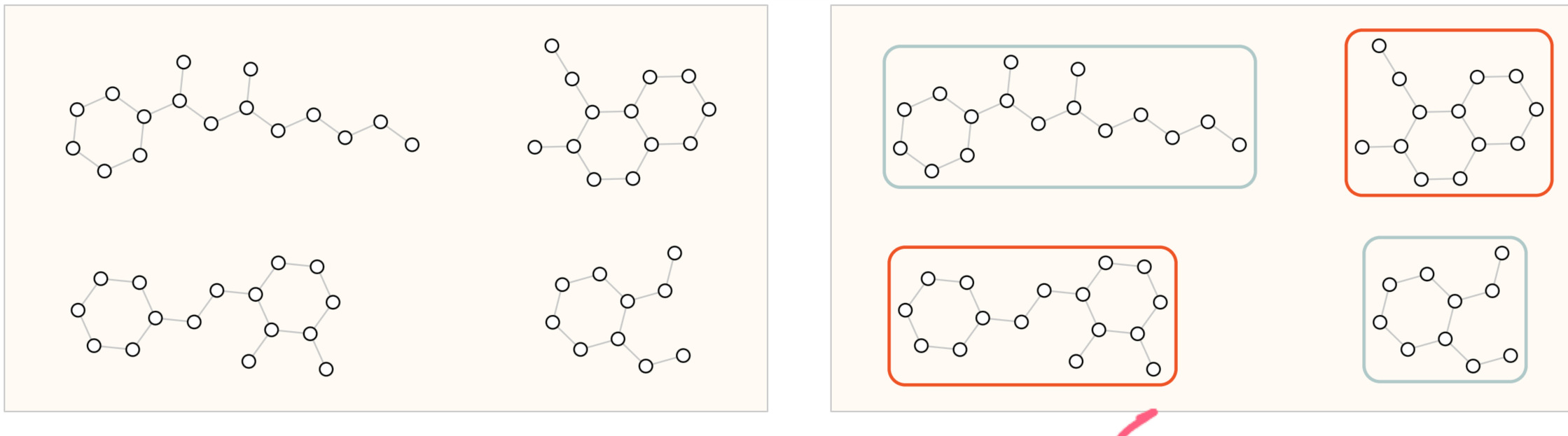
{ Graph-level - task

✓ Image → Face/Not

This is analogous to image classification problems with MNIST and CIFAR, where we want to associate a label to an entire image. With text, a similar problem is sentiment analysis where we want to identify the mood or emotion of an entire sentence at once.

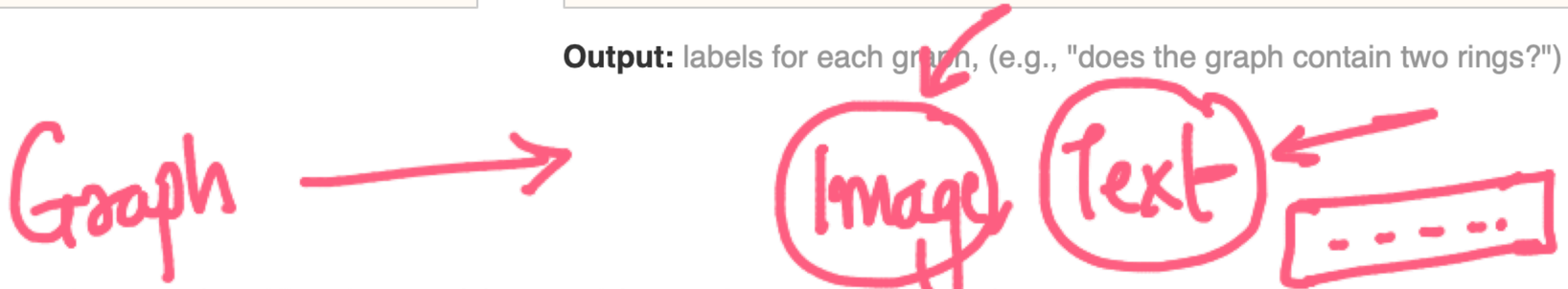
Node-level task

No within a



Input: graphs

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



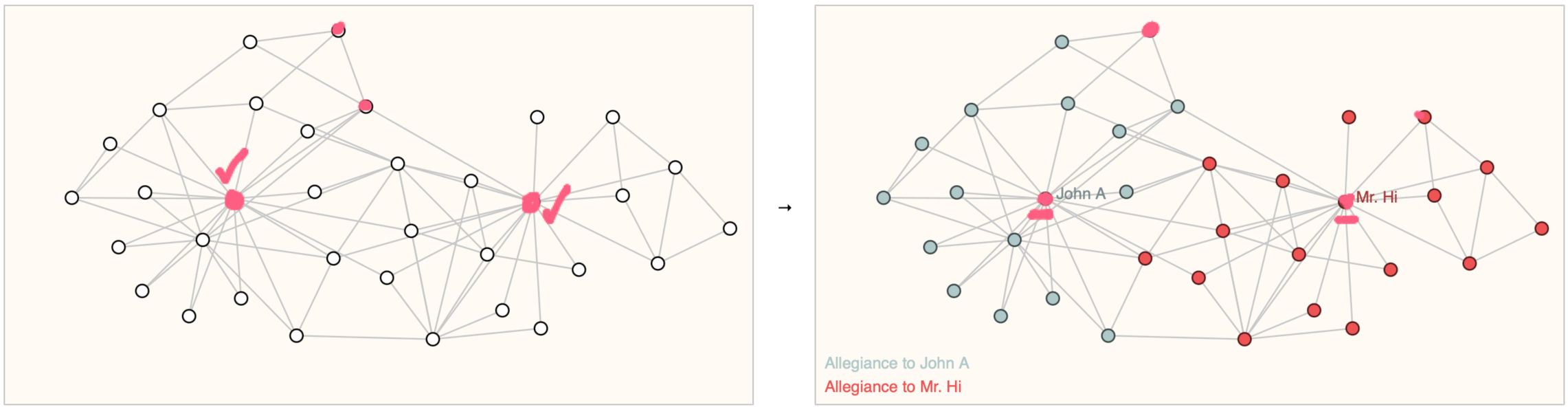
This is analogous to image classification problems with MNIST and CIFAR, where we want to associate a label to an entire image. With text, a similar problem is sentiment analysis where we want to identify the mood or emotion of an entire sentence at once.

Node-level task

No within a

H (Administrator) creates a schism in the karate club. The nodes represent individual karate practitioners, and the edges represent interactions between these members outside of karate. The prediction problem is to classify whether a given member becomes loyal to either Mr. Hi or John H, after the feud. In this case, distance between a node to either the Instructor or Administrator is highly correlated to this label.

node-level-task

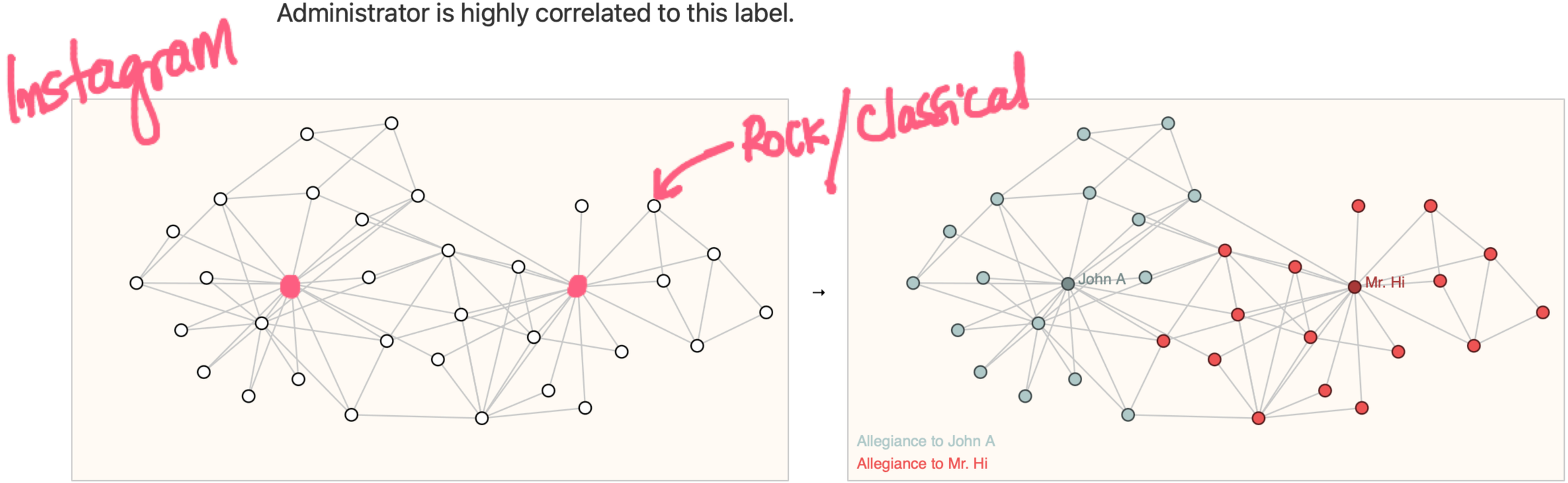


Input: graph with unlabled nodes

Output: graph node labels

On the left we have the initial conditions of the problem, on the right we have a possible solution, where each node has been classified based on the alliance. The dataset can be used in other graph problems like unsupervised learning.

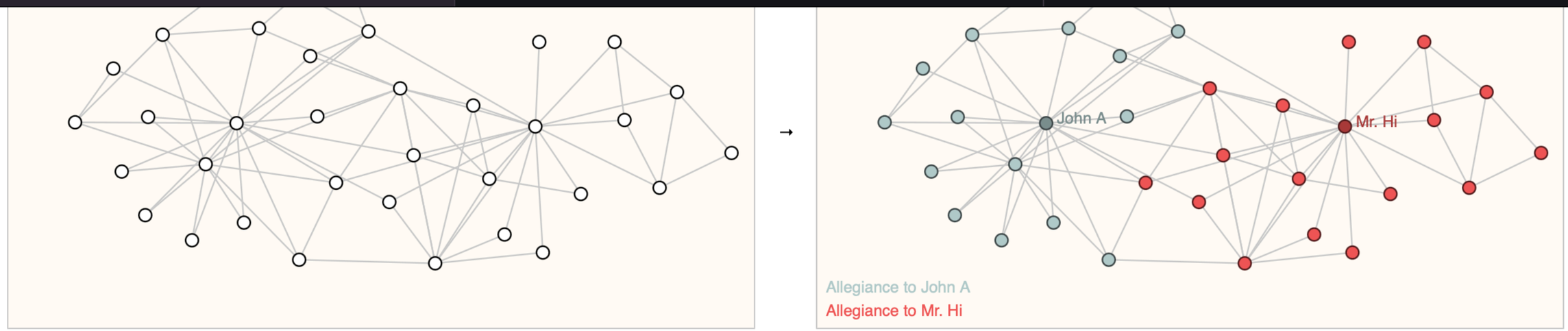
H (Administrator) creates a schism in the karate club. The nodes represent individual karate practitioners, and the edges represent interactions between these members outside of karate. The prediction problem is to classify whether a given member becomes loyal to either Mr. Hi or John H, after the feud. In this case, distance between a node to either the Instructor or Administrator is highly correlated to this label.



Input: graph with unlabled nodes

Output: graph node labels

On the left we have the initial conditions of the problem, on the right we have a possible solution, where each node has been classified based on the alliance. The dataset can be used in other graph problems like unsupervised learning.



Input: graph with unlabled nodes

Output: graph node labels

On the left we have the initial conditions of the problem, on the right we have a possible solution, where each node has been classified based on the alliance. The dataset can be used in other graph problems like unsupervised learning.

Following the image analogy, node-level prediction problems are analogous to *image segmentation*, where we are trying to label the role of each pixel in an image. With text, a similar task would be predicting the parts-of-speech of each word in a sentence (e.g. noun, verb, adverb, etc).

Edge-level task

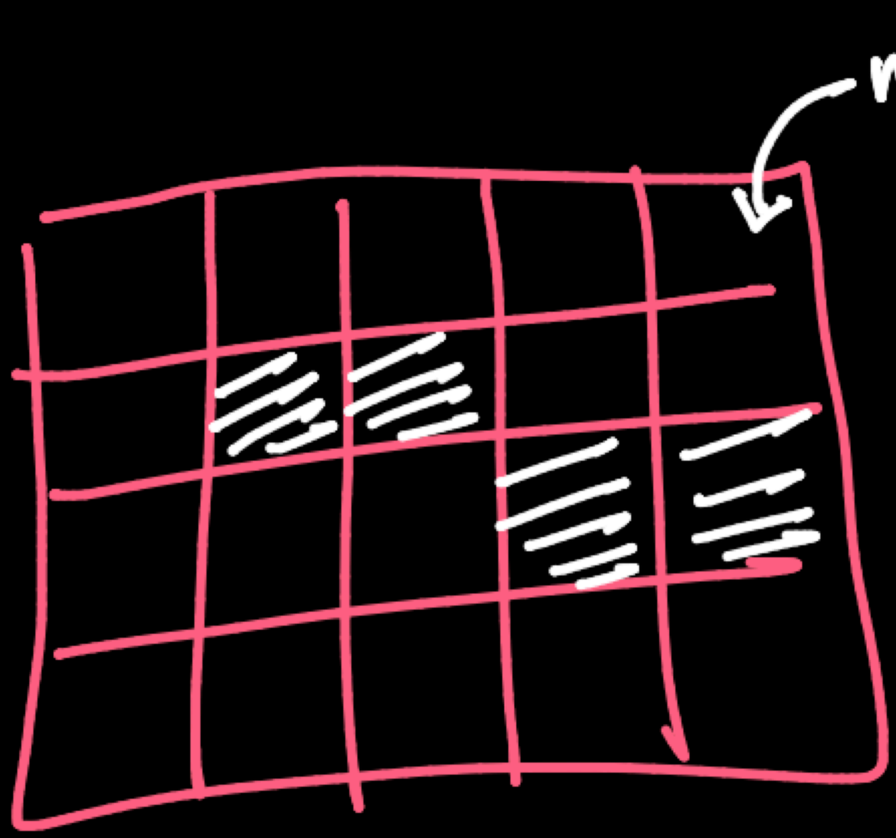
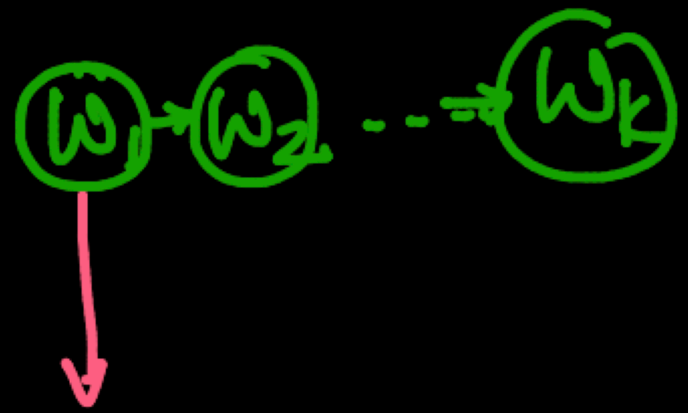


Image - Seg

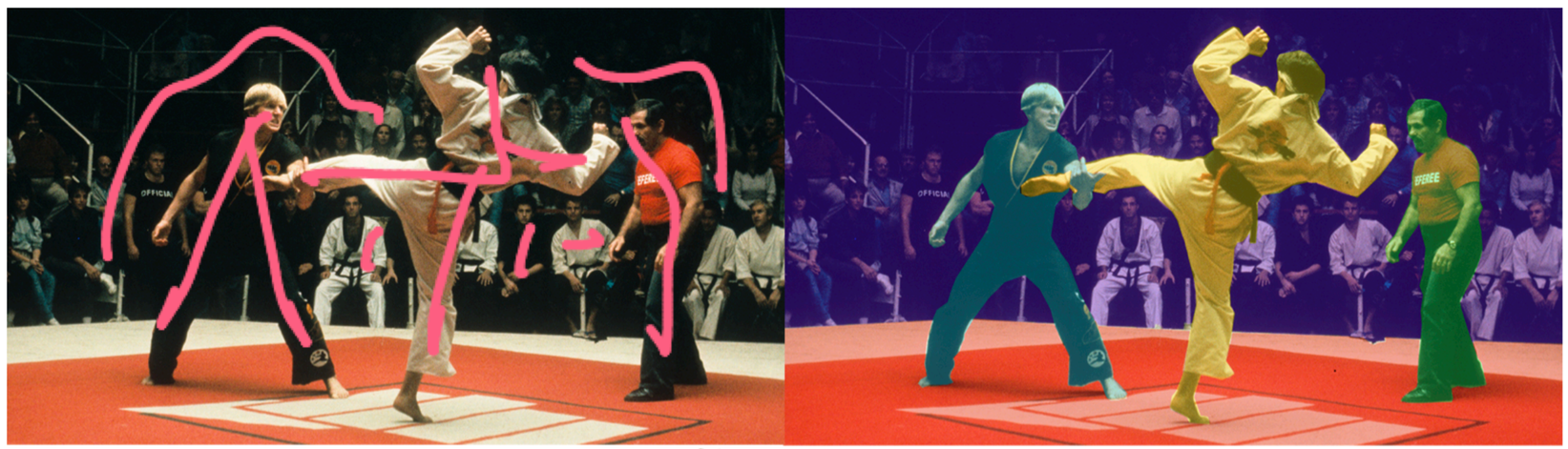
node-level-task

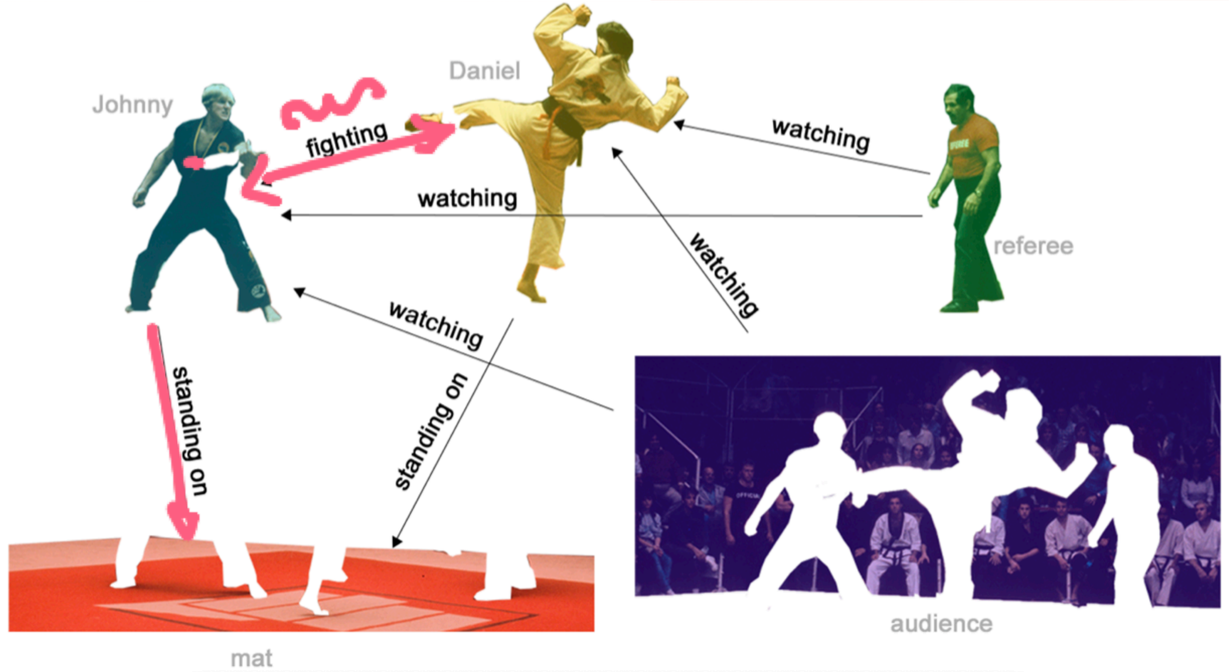
G: Text



node-level- task

graph.



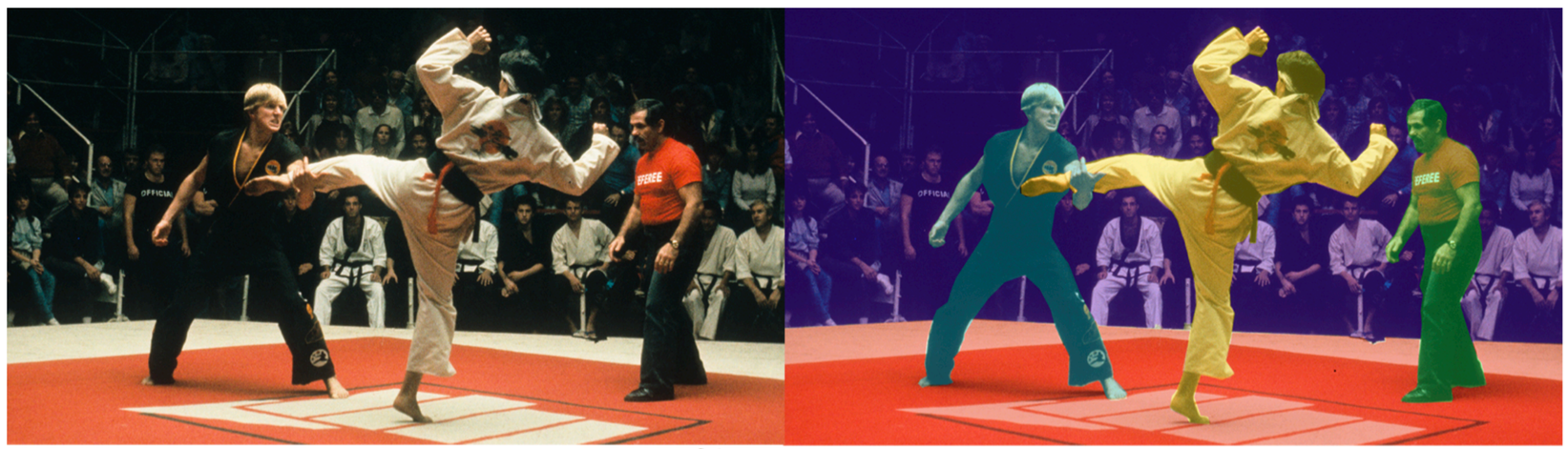


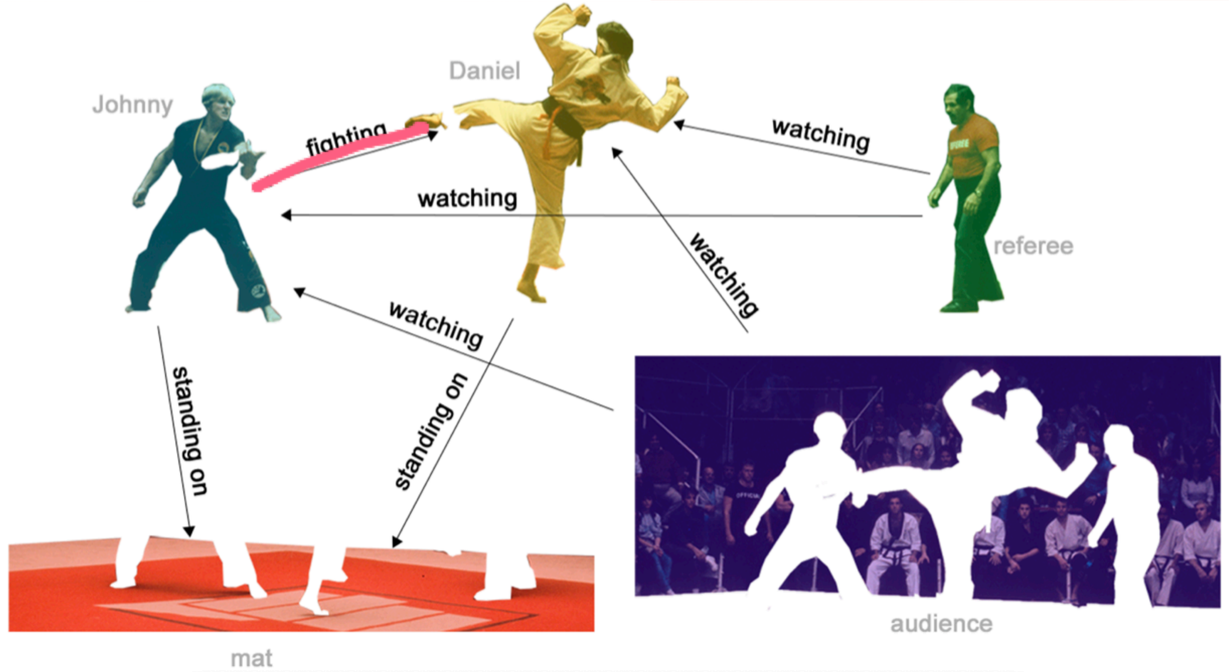
Video

In (

ree, the audience

graph.

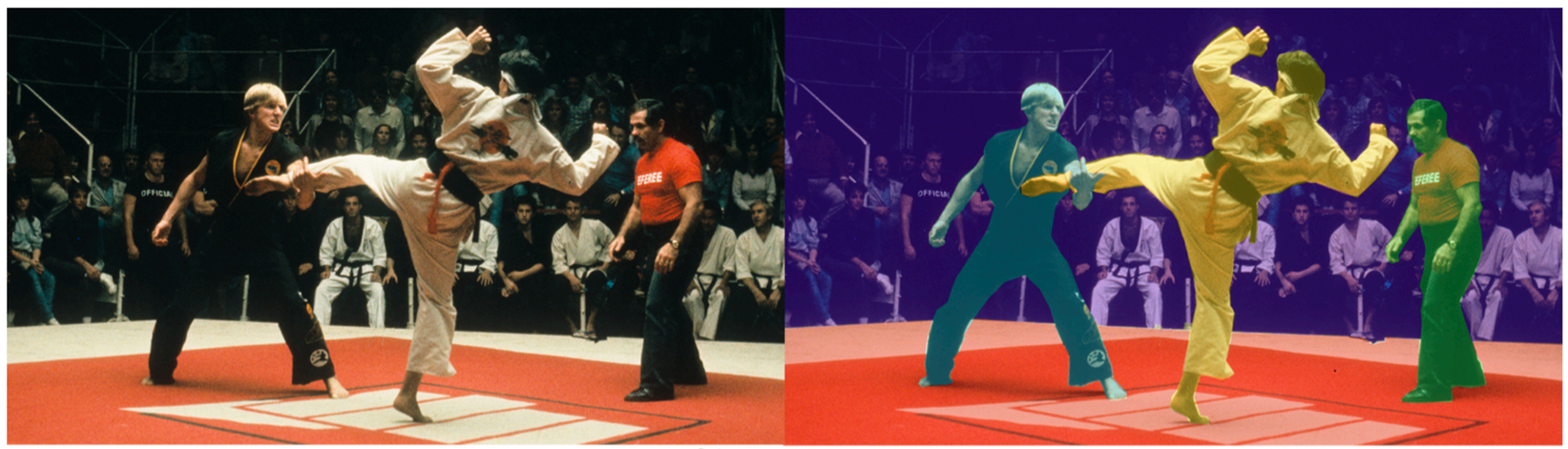


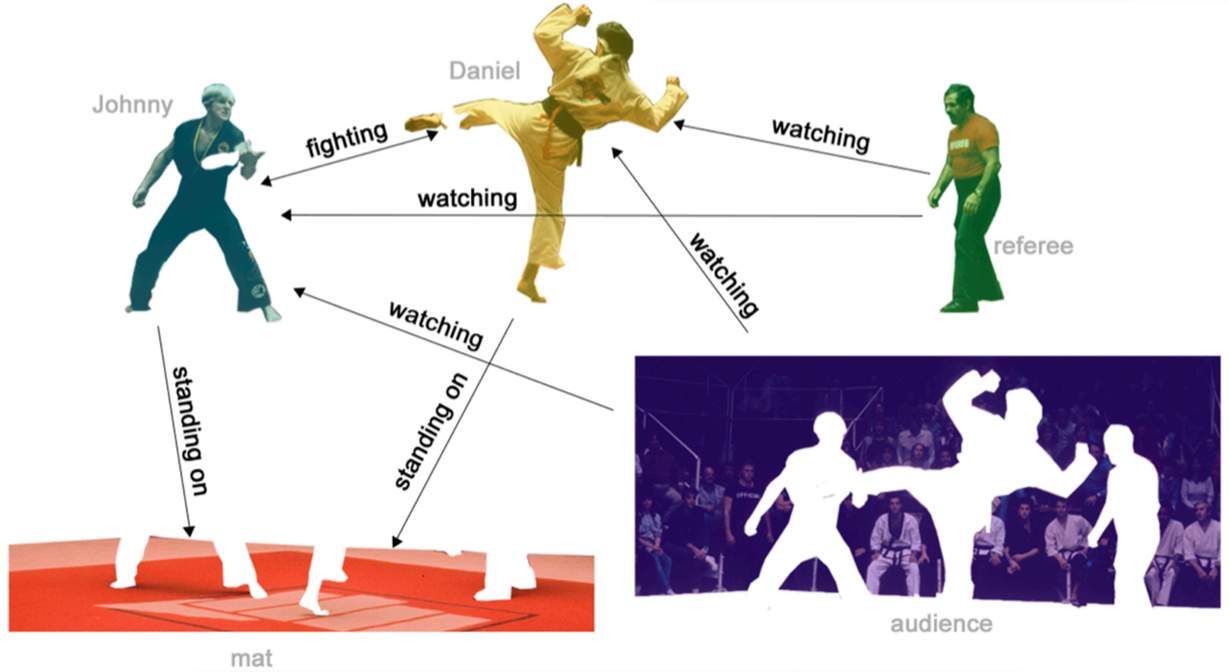


In (

ree, the audience

graph.





In (

ree, the audience

The challenges of using graphs in machine learning

So, 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.

Machine learning models typically take rectangular or grid-like arrays as input. So, it's not immediately intuitive how to represent them in a format that is compatible with deep learning. 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: for example, with nodes we can form a node feature matrix N by assigning each node an index i and storing the feature for $node_i$ in N. While these matrices have a variable number of examples, they can be processed without any special techniques.

However, representing a graph's connectivity is more complicated. Perhaps the most obvious choice would be to use an adjacency matrix, since this is easily tensorisable. However, this representation has a few drawbacks. From the example dataset table, we see the number of nodes in a graph can be on the order of millions, and the number of edges per node can be hig ace-

The challenges of using graphs in machine learning

So, 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.

Machine learning models typically take rectangular or grid-like arrays as input. So, it's not immediately intuitive how to represent them in a format that is compatible with deep learning. 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: for example, with nodes we can form a node feature matrix N by assigning each node an index i and storing the feature for $node_i$ in N. While these matrices have a variable number of examples, they can be processed without any special techniques.

However, representing a graph's connectivity is more complicated. Perhaps the most obvious choice would be to use an adjacency matrix, since this is easily tensorisable. However, this representation has a few drawbacks. From the example dataset table, we see the number of nodes in a graph can be on the order of millions, and the number of edges per node can be hig ace-

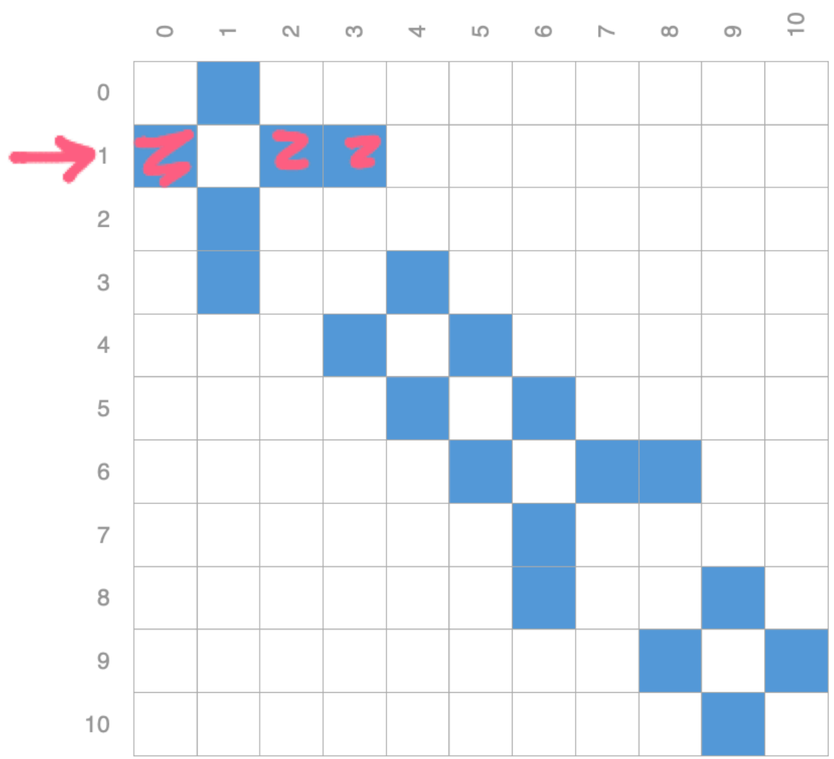
The challenges of using graphs in machine learning

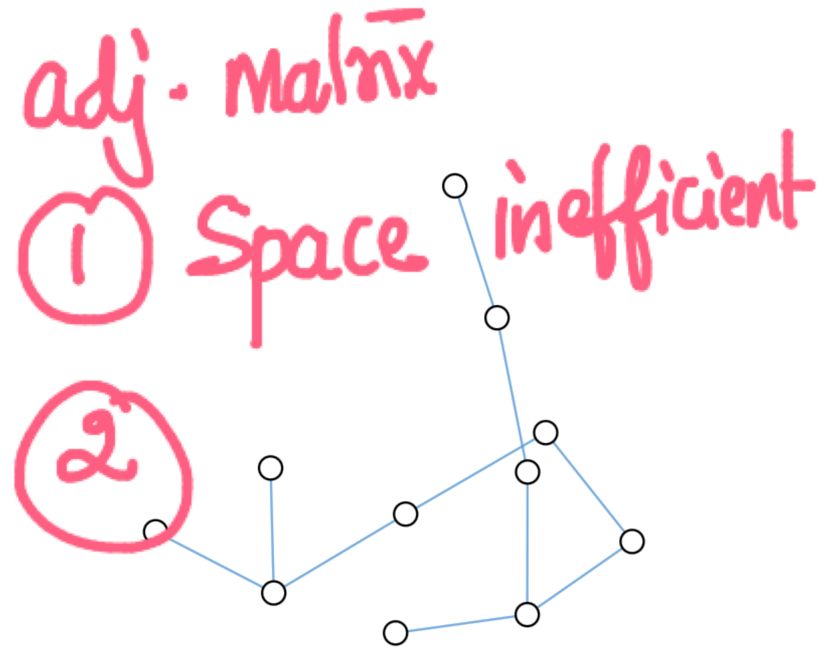
So, how do we go about solving these different graph tasks with neural networks? The first step is to think about how we will represent graphs to be compatible with neural networks.

Machine learning models typically take rectangular or grid-like arrays as input. So, it's not immediately intuitive how to represent them in a format that is compatible with deep learning. 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: for example, with nodes we can form a node feature matrix N by assigning each node an index i and storing the feature for $node_i$ in N. While these matrices have a variable number of examples, they can be processed without any special techniques.

However, representing a graph's connectivity is more complicated. Perhaps the most obvious choice would be to use an adjacency matrix, since this is easily tensorisable. However, this representation has a few drawbacks. From the example dataset table, we see the number of nodes in a graph can be on the order of millions, and the number of edges per node can be high

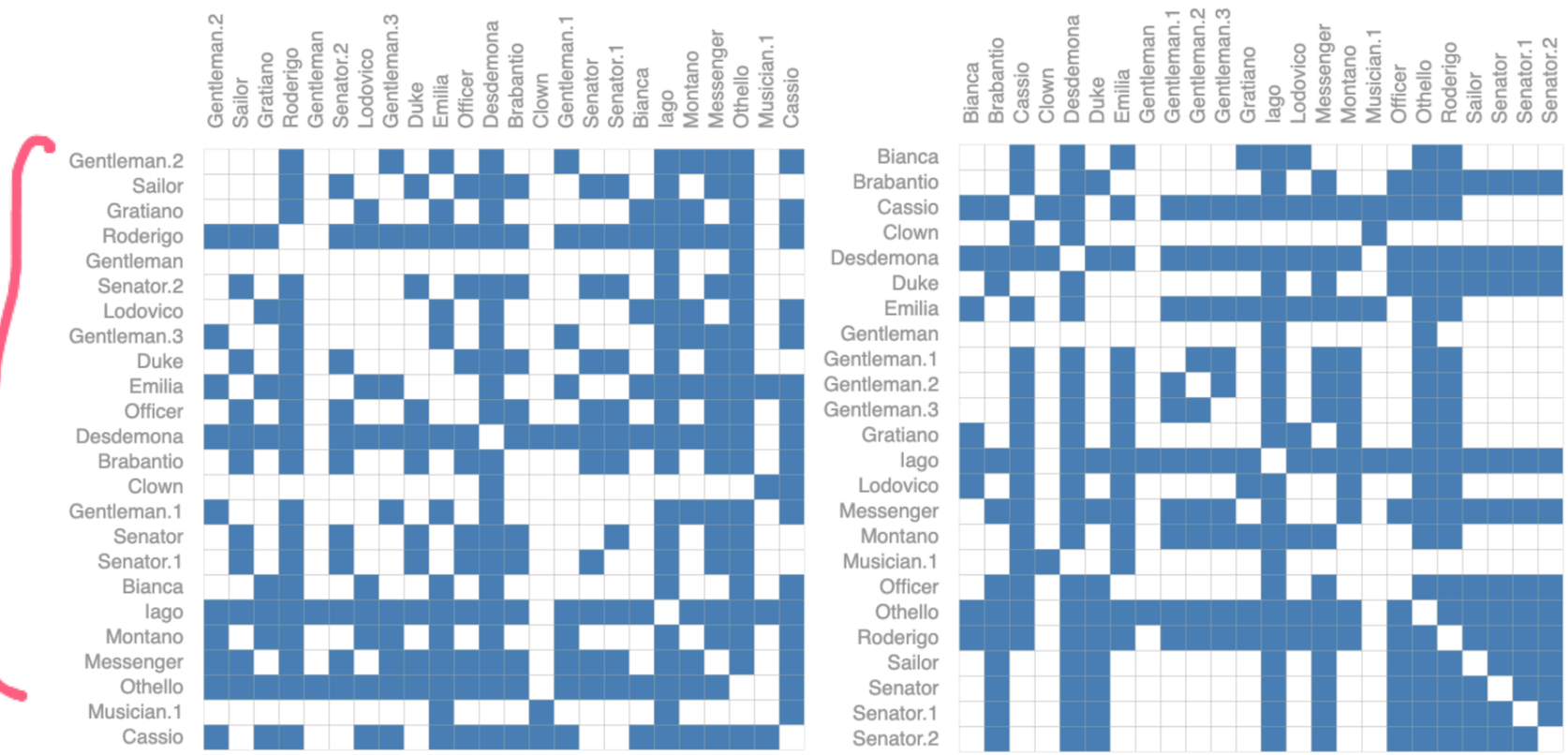
electrons in 3D space. All particles are interacting, but when a pair of atoms are stuck in a stable distance from each other, we say they share a covalent bond. Different pairs of atoms and bonds have different distances (e.g. single-bonds, double-bonds). It's a very convenient and common abstraction to describe this 3D object as a graph, where nodes are atoms and edges are covalent bonds. [8] Here are two common molecules, and their associated graphs.





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

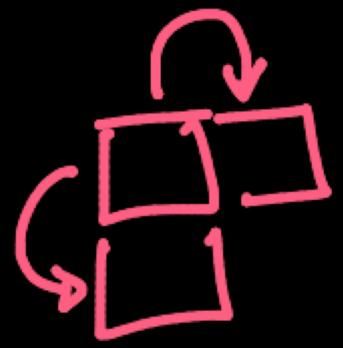
For example, the Othello graph from before can be described equivalently with these two adjacency matrices. It can also be described with every other possible permutation of the nodes.



Two adjacency matrices representing the same graph.

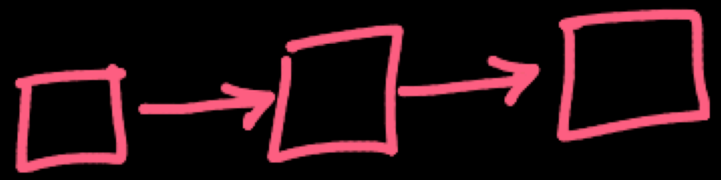
Th aph of 4

Image:

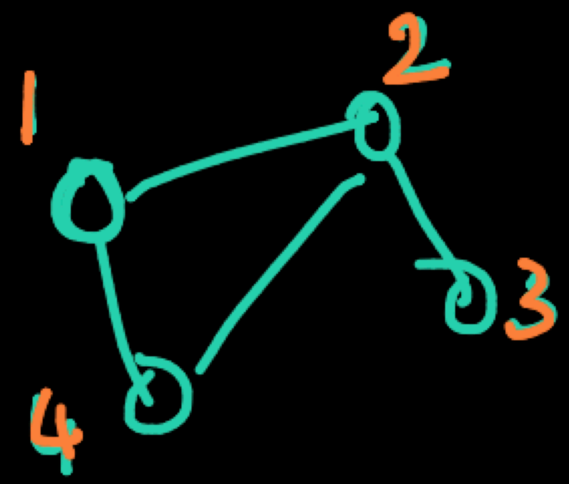


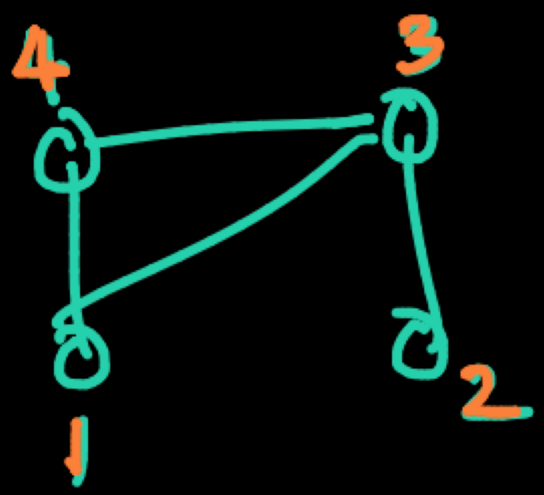
natural ordering

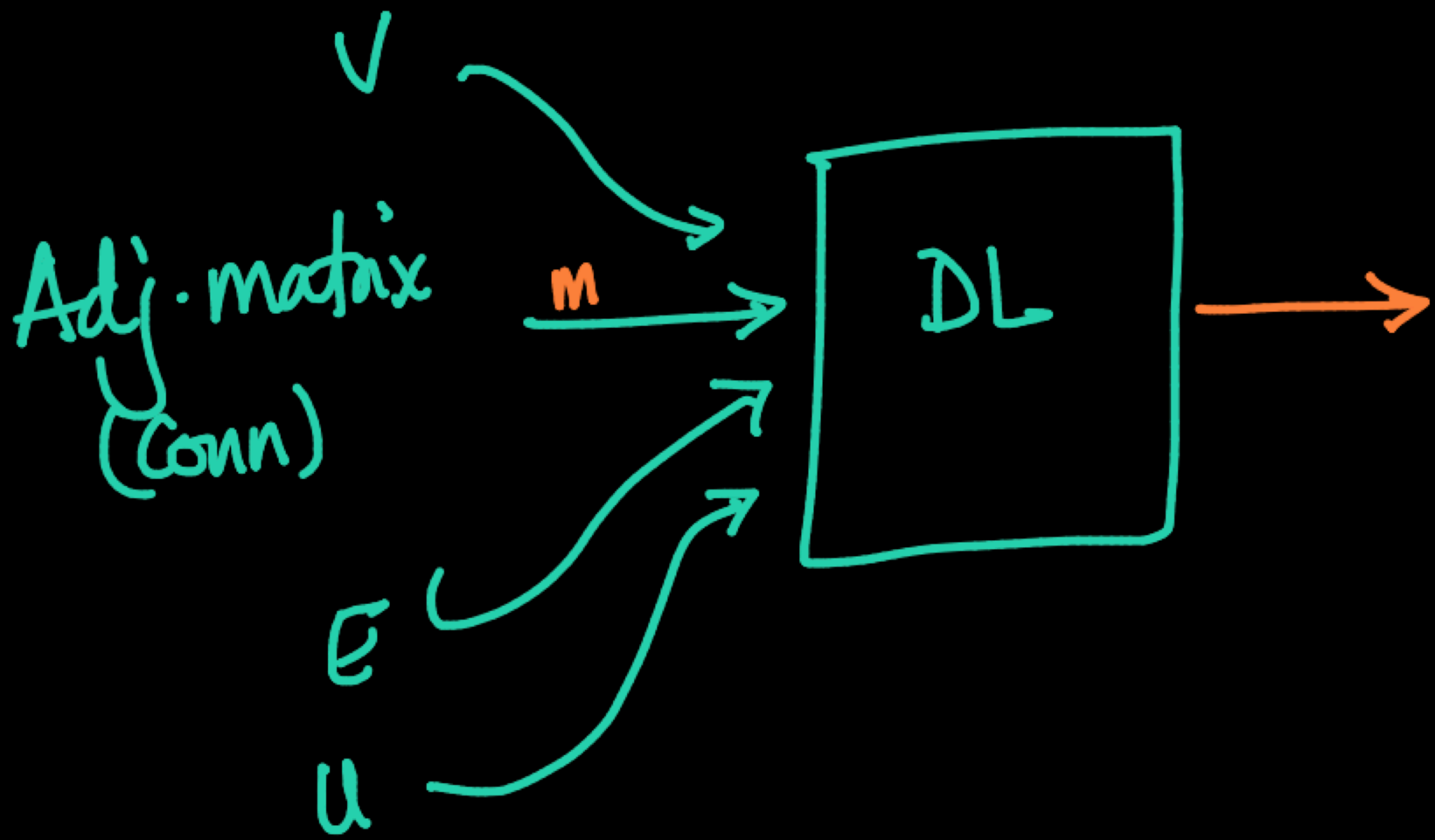
Text:

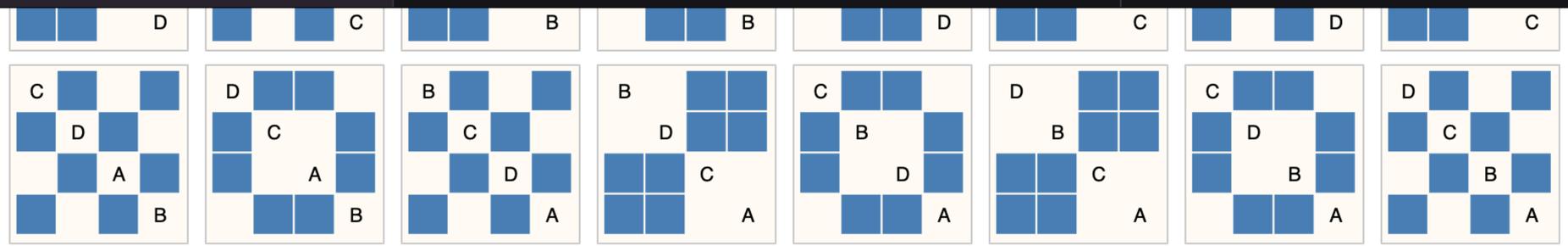


Graph:









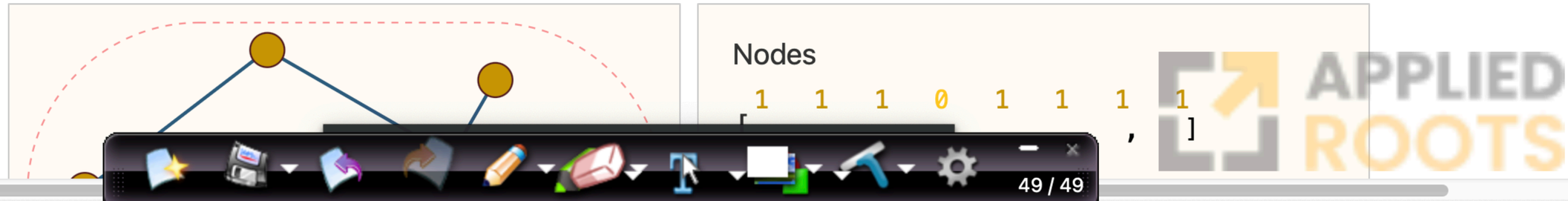
All of these adjacency matrices represent the same graph. Click on an edge to remove it on a "virtual edge" to add it and the matrices will update accordingly.

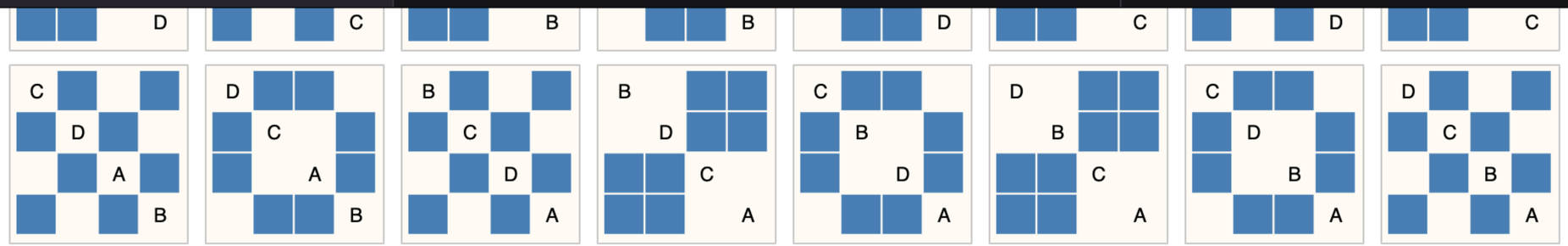
One elegant and memory-efficient way of representing sparse matrices is as adjacency lists.

These describe the connectivity of edge e_k between nodes n_i and n_j as a tuple (i,j) in the k-th entry of an adjacency list. Since we expect the number of edges to be much lower than the number of entries for an adjacency matrix (n^2_{nodes}), we avoid computation and storage on the disconnected parts of the graph.

To make this notion concrete, we can see how information in different graphs might be represented under this specification:

Another way of stating th is with Big-O notation, it i preferable to have $O(n_{edges})$, rather than $O(n^2_{nodes})$.





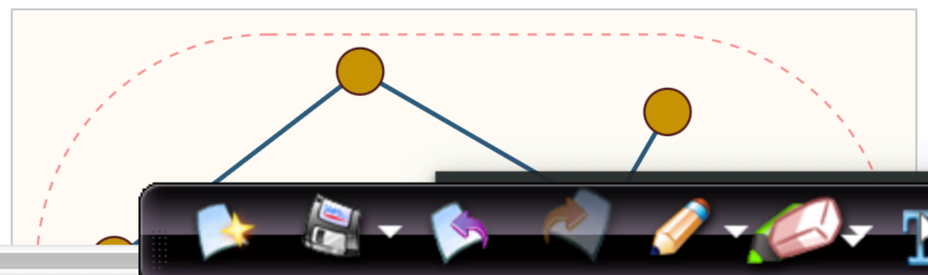
All of these adjacency matrices represent the same graph. Click on an edge to remove it on a "virtual edge" to add it and the matrices will update accordingly.

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

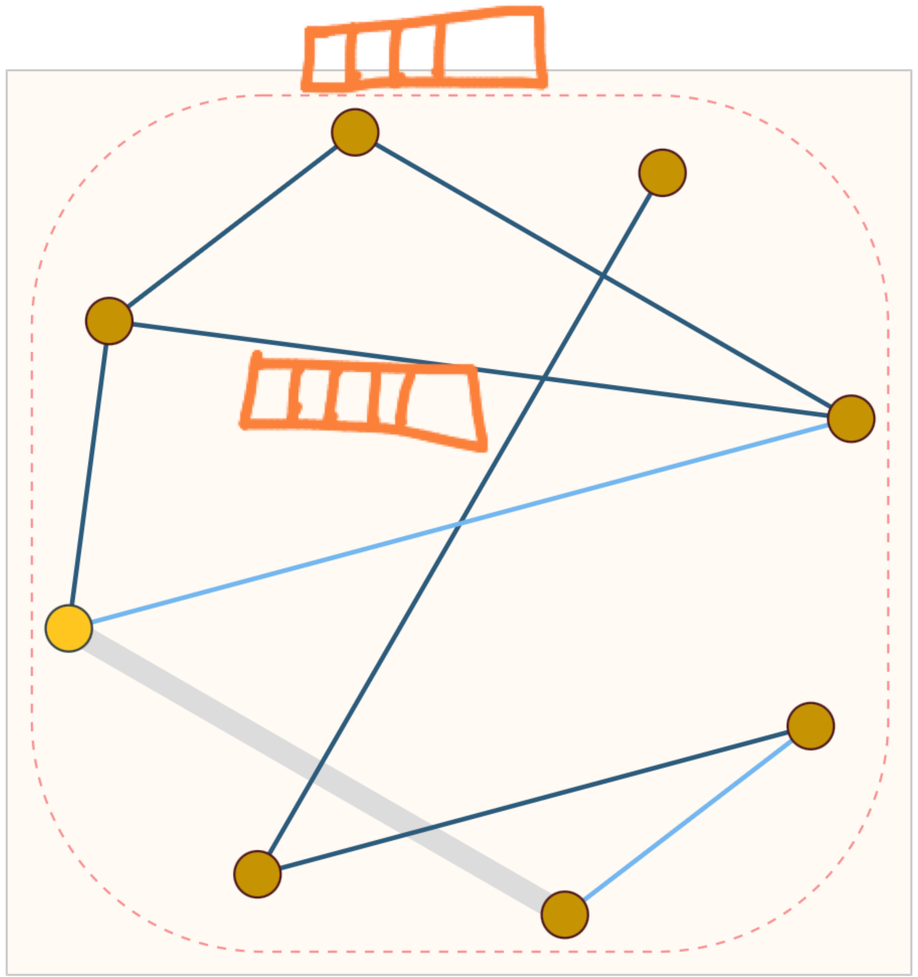
These describe the connectivity of edge e_k between nodes n_i and n_j as a tuple (i,j) in the k-th entry of an adjacency list. Since we expect the number of edges to be much lower than the number of entries for an adjacency matrix (n^2_{nodes}), we avoid computation and storage on the disconnected parts of the graph.

To make this notion concrete, we can see how information in different graphs might be represented under this specification:

Another way of stating th is with Big-O notation, it i preferable to have $O(n_{edges})$, rather than $O(n^2_{nodes})$.



To make this notion concrete, we can see how information in different graphs might be represented under this specification:



Nodes

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

Edges

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

Adjacency List (Conn)

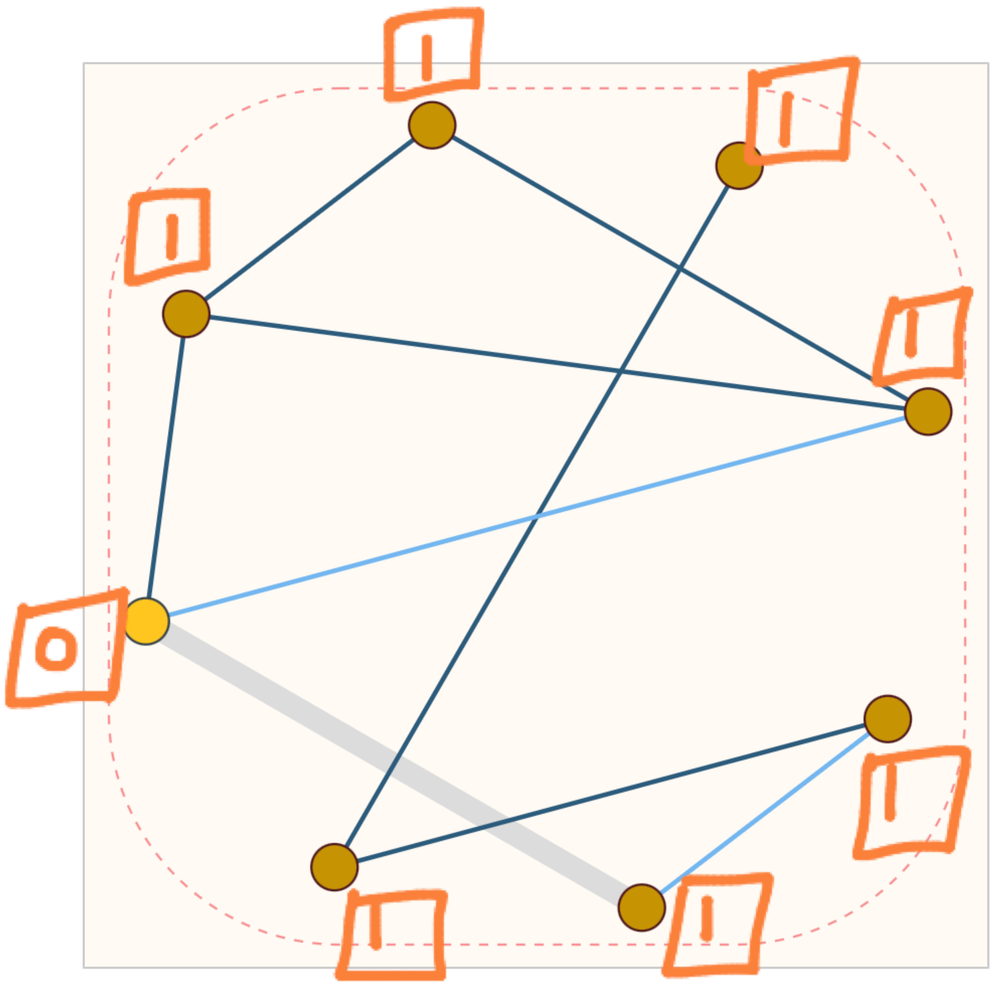
```
[ [1, 0] , [2, 0] , [4, 3] , [5, 4] ,
  [6, 2] , [7, 3] , [7, 4] , [7, 5] ]
```

Global

```
0
```

Hover and click on the edges, nodes, and global graph marker to view and change attribute representations. On one side we have a small graph and on the other the information of th graph in a tensor representation.

To make this notion concrete, we can see how information in different graphs might be represented under this specification:



Nodes

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

Edges

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

Adjacency List

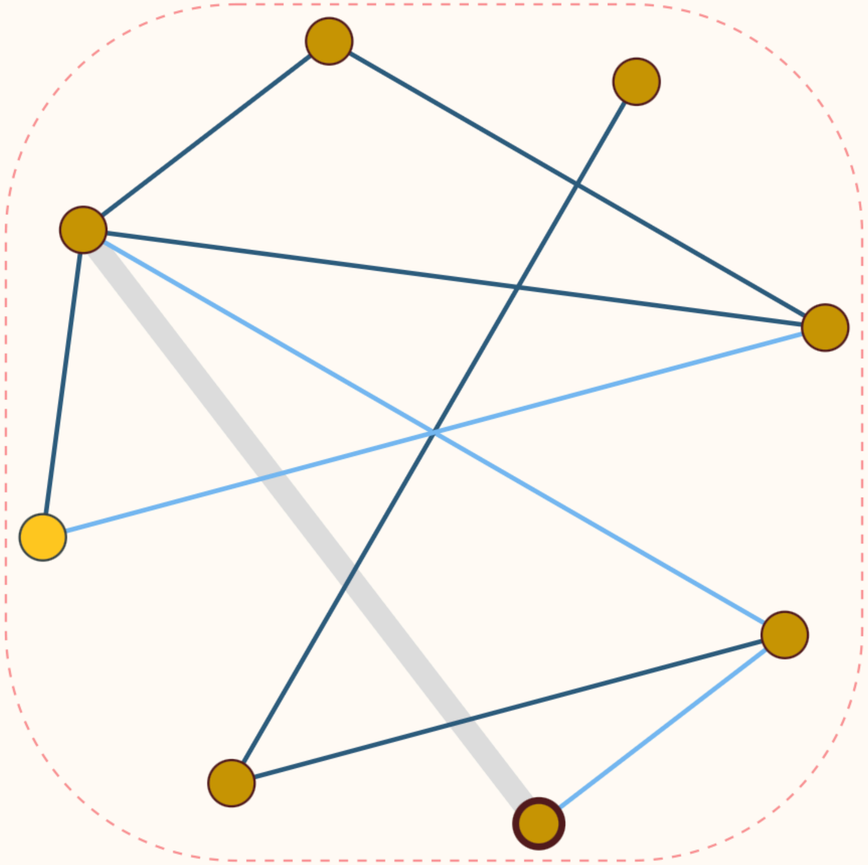
```
[ [1, 0] , [2, 0] , [4, 3] , [5, 4] ,
  [6, 2] , [7, 3] , [7, 4] , [7, 5] ]
```

Global

```
0
```

Hover and click on the edges, nodes, and global graph marker to view and change attribute representations. On one side we have a small graph and on the other the information of th graph in a tensor representation.

To make this notion concrete, we can see how information in different graphs might be represented under this specification:



Nodes

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

Edges

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

Adjacency List

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

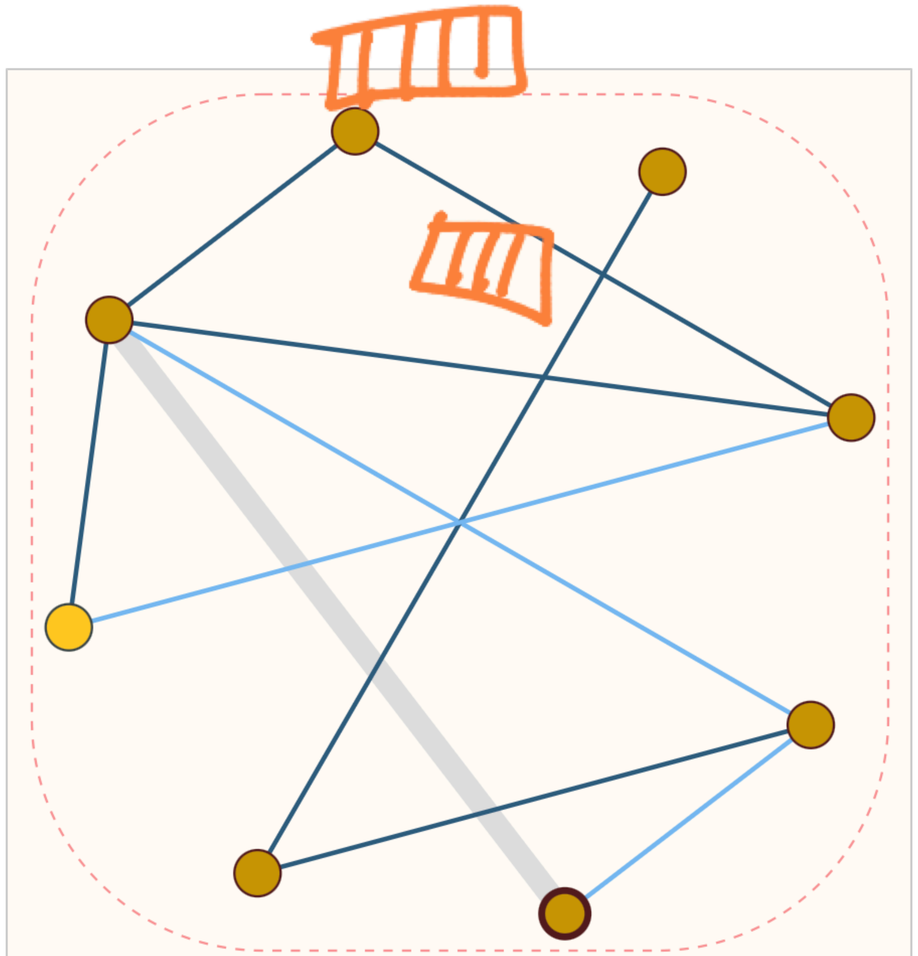
List

Global

```
0
```

Hover and click on the edges, nodes, and global graph marker to view and change attribute representations. On one side we have a small graph and on the other the information of th graph in a tensor representation.

To make this notion concrete, we can see how information in different graphs might be represented under this specification:



Nodes

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

Edges

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

Adjacency List

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

Global

```
0
```

Hover and click on the edges, nodes, and global graph marker to view and change attribute representations. On one side we have a small graph and on the other the information of th graph in a tensor representation.

Graph Neural Networks

Now that the graph's description is in a matrix format that is permutation invariant we will describe using graph neural networks (GNNs) to solve graph prediction tasks. **A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global-context) that preserves graph symmetries (permutation invariances).** We're going to build GNNs using the "message passing neural network" framework proposed by Gilmer et al. [18] using the Graph Nets architecture schematics introduced by Battaglia et al. [19] 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.

The simplest GNN

With the numerical representation of graphs that we've constructed above (with vectors instead of scalars), we are now ready to build a GNN. 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.

Th able model)

For simplicity, the previou diagrams used scalars to represent graph attribute in practice feature vector or embeddings, are much more useful.

You could also call it a

Graph Neural Networks

Now that the graph's description is in a matrix format that is permutation invariant, we will describe using graph neural networks (GNNs) to solve graph prediction tasks. **A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global-context) that preserves graph symmetries (permutation invariances).** We're going to build GNNs using the "message passing neural network" framework proposed by Gilmer et al. [18] using the Graph Nets architecture schematics introduced by Battaglia et al. [19] 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.

The simplest GNN

With the numerical representation of graphs that we've constructed above (with vectors instead of scalars), we are now ready to build a GNN. 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.

Th able model)

For simplicity, the previou diagrams used scalars to represent graph attribute in practice feature vector or embeddings, are much more useful.

You could also call it a

Graph Neural Networks

Now that the graph's description is in a matrix format that is permutation invariant, we will describe using graph neural networks (GNNs) to solve graph prediction tasks. **A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global-context) that preserves graph symmetries (permutation invariances).** We're going to build GNNs using the "message passing neural network" framework proposed by Gilmer et al. [18] using the Graph Nets architecture schematics introduced by Battaglia et al. [19] 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.

The simplest GNN

With the numerical representation of graphs that we've constructed above (with vectors instead of scalars), we are now ready to build a GNN. 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.

Th able model)

For simplicity, the previou diagrams used scalars to represent graph attribute in practice feature vector or embeddings, are much more useful.

You could also call it a

Vertices

Graph

$\{ V, E, U \}$

adj. list

DL

Softmax

Graph

$\{ V', E', U' \}$

adj. list

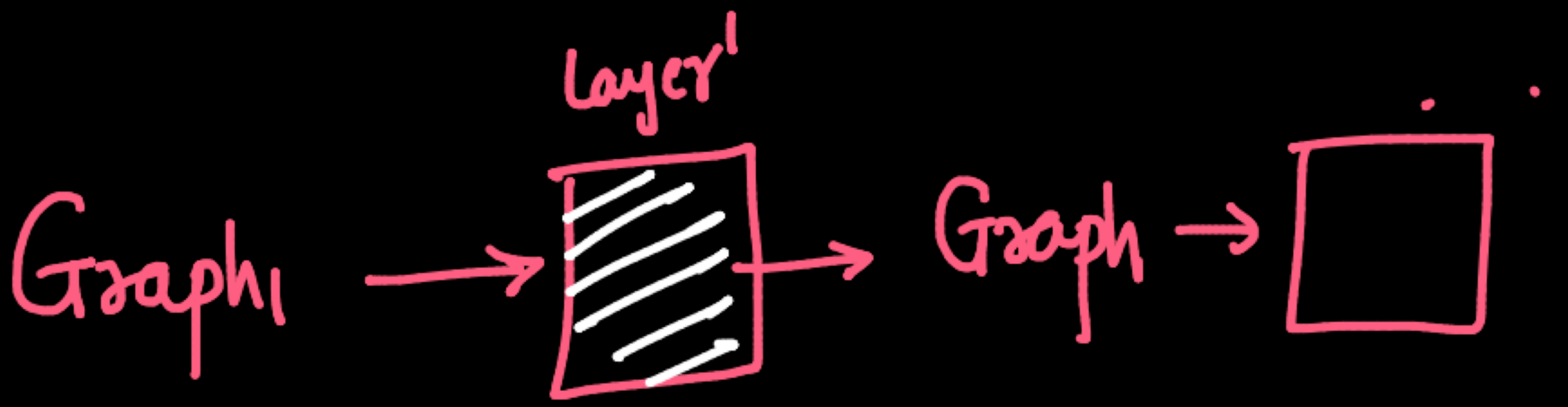
10 vertices

V_1 10

V_2 10

V_{10}

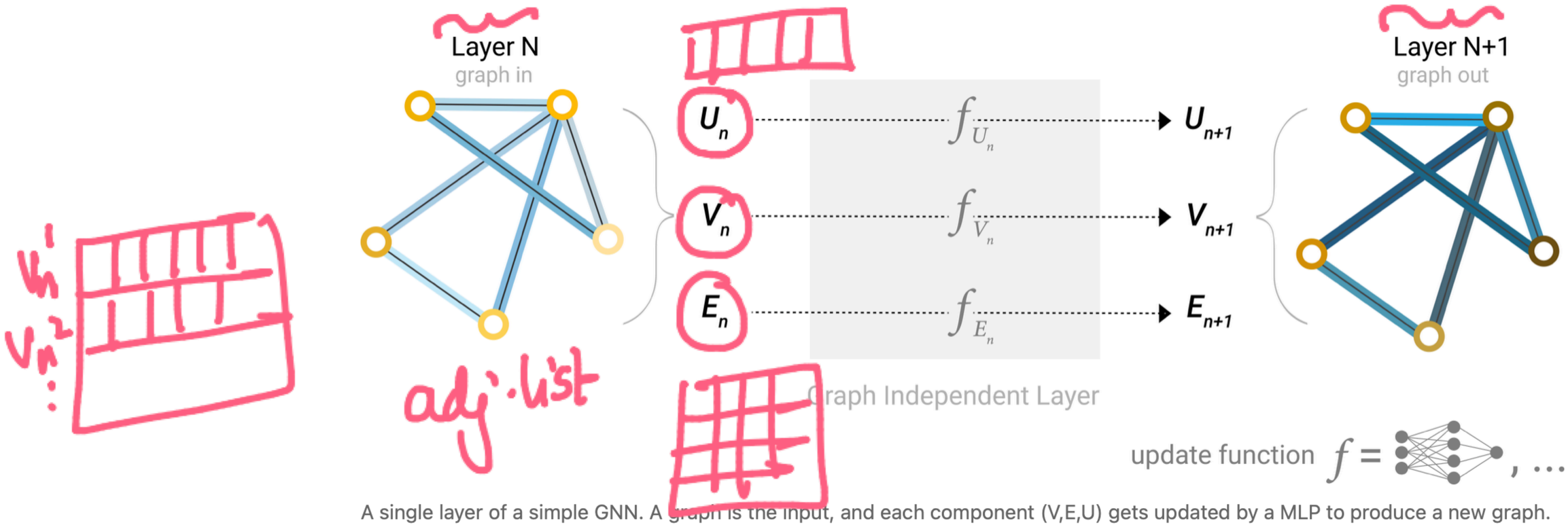
V_1' : 100



Images: Conv ; pooling

Text: Self-attention

component of a graph; we call this a GNN layer. For each node vector, we apply the MLP and get back a learned node-vector. We do the same for each edge, learning a per-edge embedding, and also for the global-context vector, learning a single embedding for the entire graph.

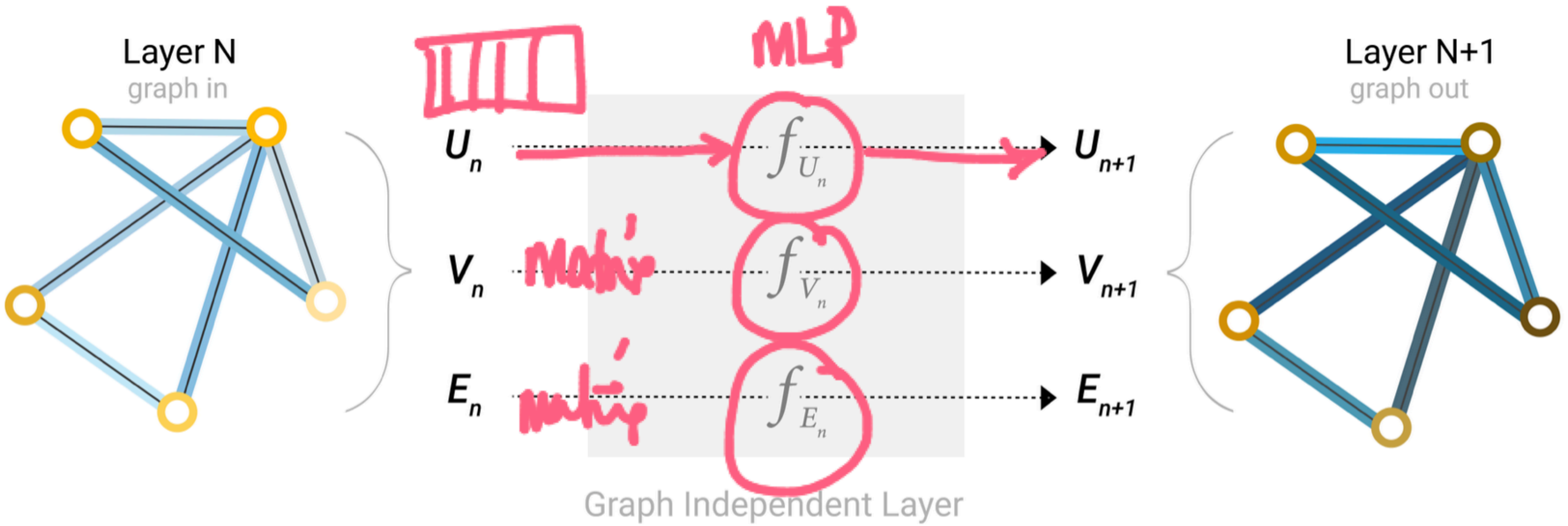


A single layer of a simple GNN. A graph is the input, and each component (V,E,U) gets updated by a MLP to produce a new graph. Each function subscript indicates a separate function for a different graph attribute at the n-th layer of a GNN model.

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

Because a GNN does not update the connectivity of the input graph, we can describe the output vectors as the input

component of a graph; we call this a GNN layer. For each node vector, we apply the MLP and get back a learned node-vector. We do the same for each edge, learning a per-edge embedding, and also for the global-context vector, learning a single embedding for the entire graph.

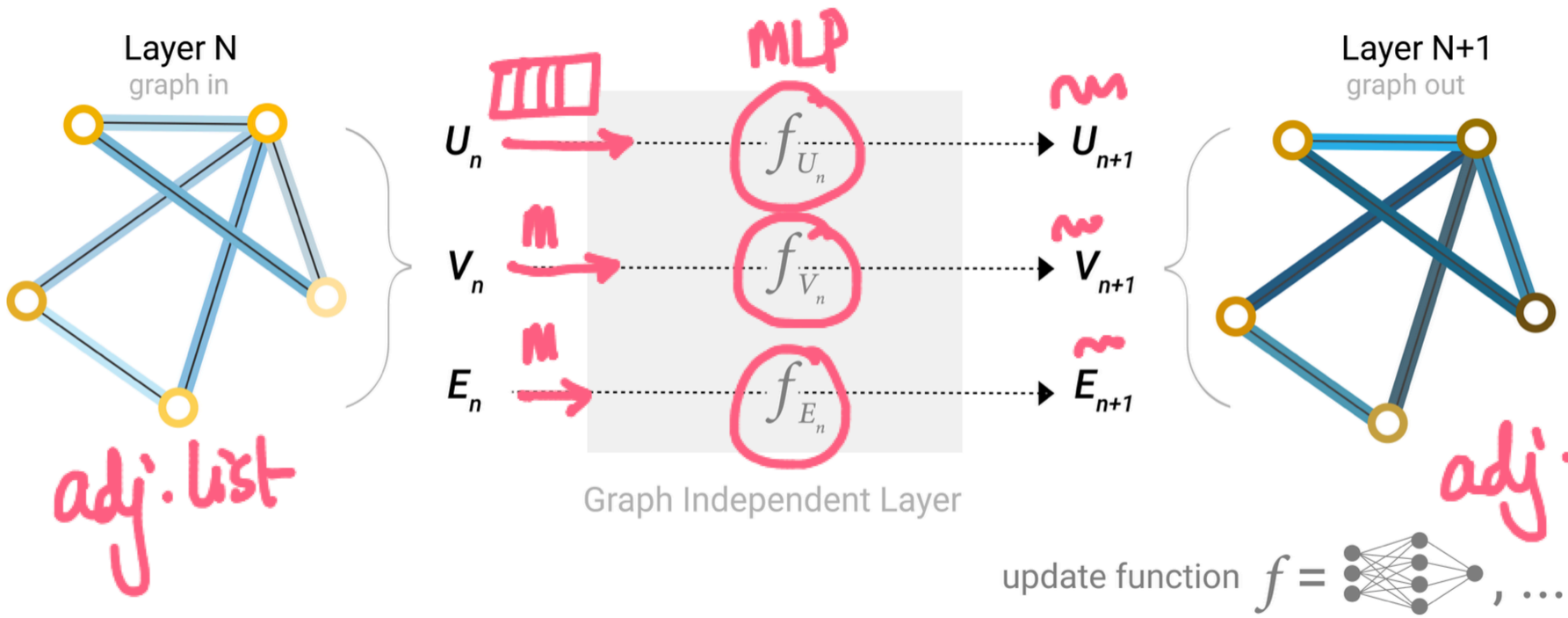


A single layer of a simple GNN. A graph is the input, and each component (V,E,U) gets updated by a MLP to produce a new graph. Each function subscript indicates a separate function for a different graph attribute at the n-th layer of a GNN model.

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

Because a GNN does not update the connectivity of the input graph, we can describe the output

component of a graph; we call this a GNN layer. For each node vector, we apply the MLP and get back a learned node-vector. We do the same for each edge, learning a per-edge embedding, and also for the global-context vector, learning a single embedding for the entire graph.

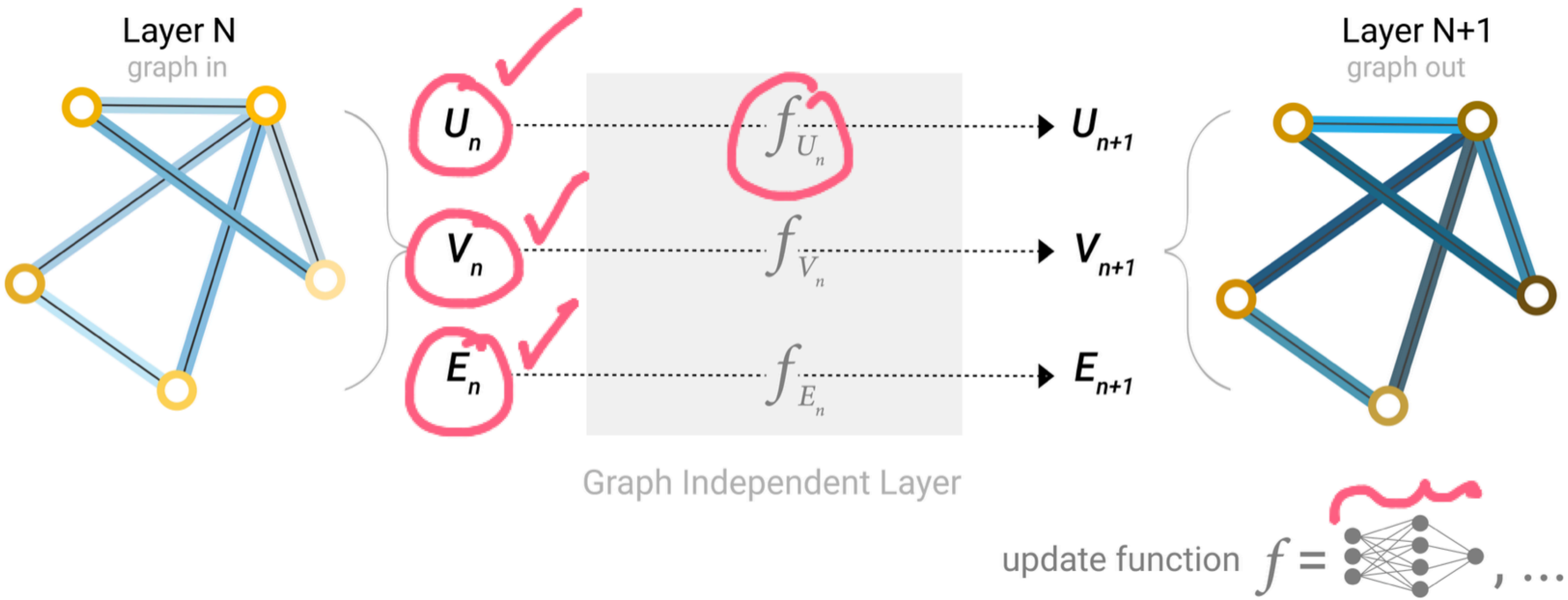


A single layer of a simple GNN. A graph is the input, and each component (V,E,U) gets updated by a MLP to produce a new graph. Each function subscript indicates a separate function for a different graph attribute at the n-th layer of a GNN model.

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

Because a GNN does not update the connectivity of the input graph, we can describe the output vectors as the input

component of a graph; we call this a GNN layer. For each node vector, we apply the MLP and get back a learned node-vector. We do the same for each edge, learning a per-edge embedding, and also for the global-context vector, learning a single embedding for the entire graph.

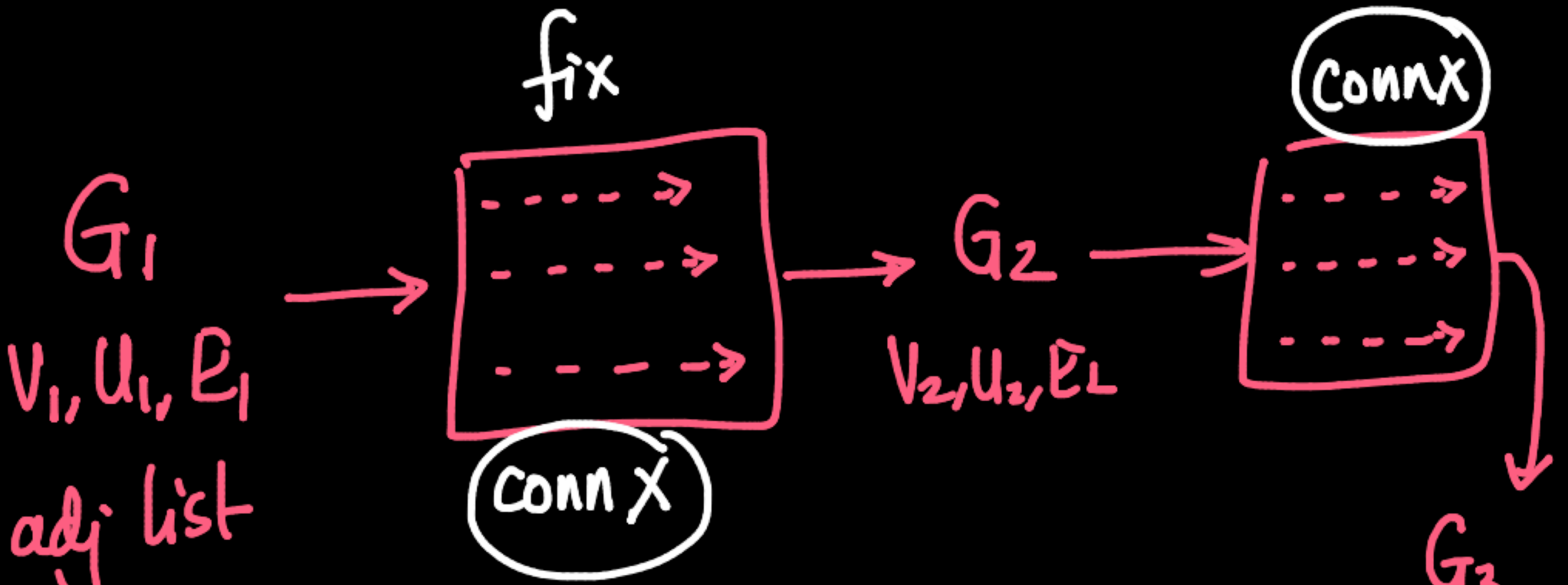


NOT using connectivity (adj. list)

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.

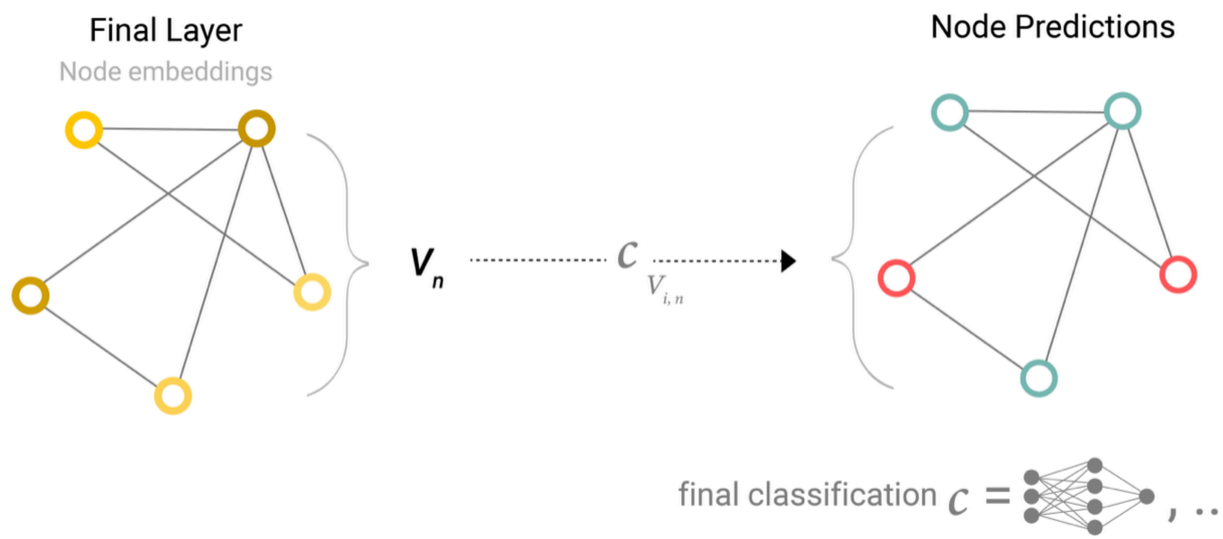
Because a GNN does not update the connectivity of the input graph, we can describe the output vectors as the input



GNN Predictions by Pooling Information

We have built a simple GNN, but how do we make predictions in any of the tasks we described above?

We will consider the case of binary classification, but this framework can easily be extended to the multi-class or regression case. If the task is to make binary predictions on nodes, and the graph already contains node information, the approach is straightforward—for each node embedding, apply a linear classifier.



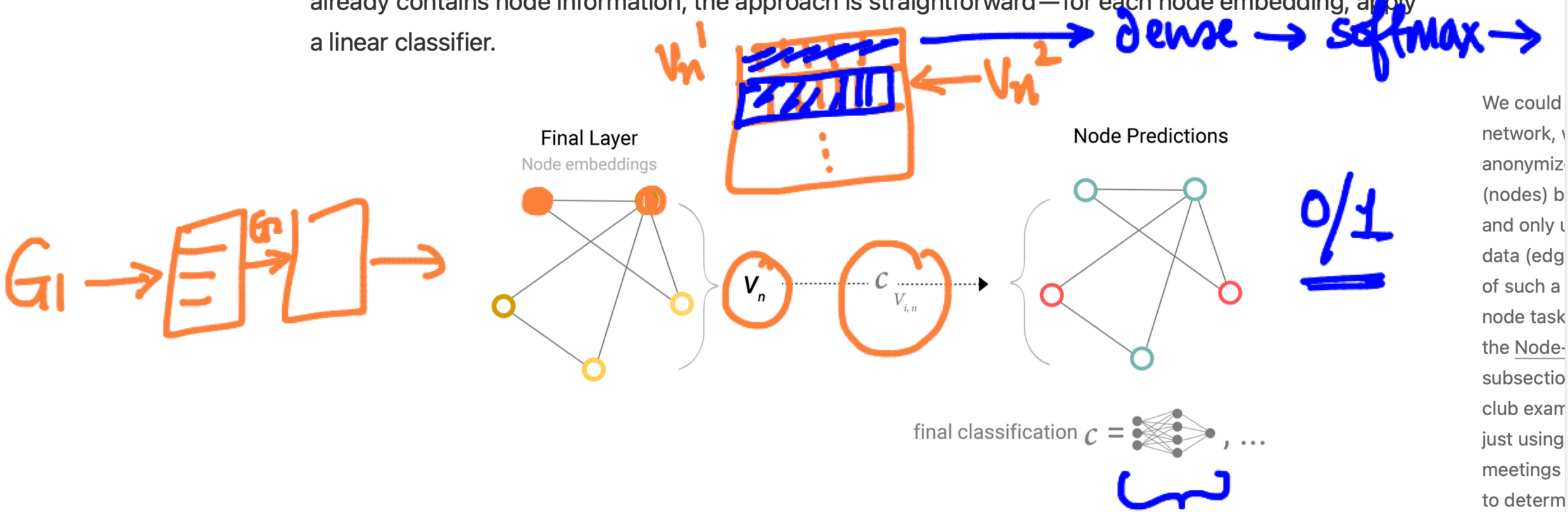
However, it is not always so simple. For instance, you might have information in the graph stored in s. We need a way to

We could network, anonymiz (nodes) b and only u data (edg of such a node task the Node- subsectio club exam just using meetings to determ Mr. Hi or

GNN Predictions by Pooling Information

We have built a simple GNN, but how do we make predictions in any of the tasks we described above?

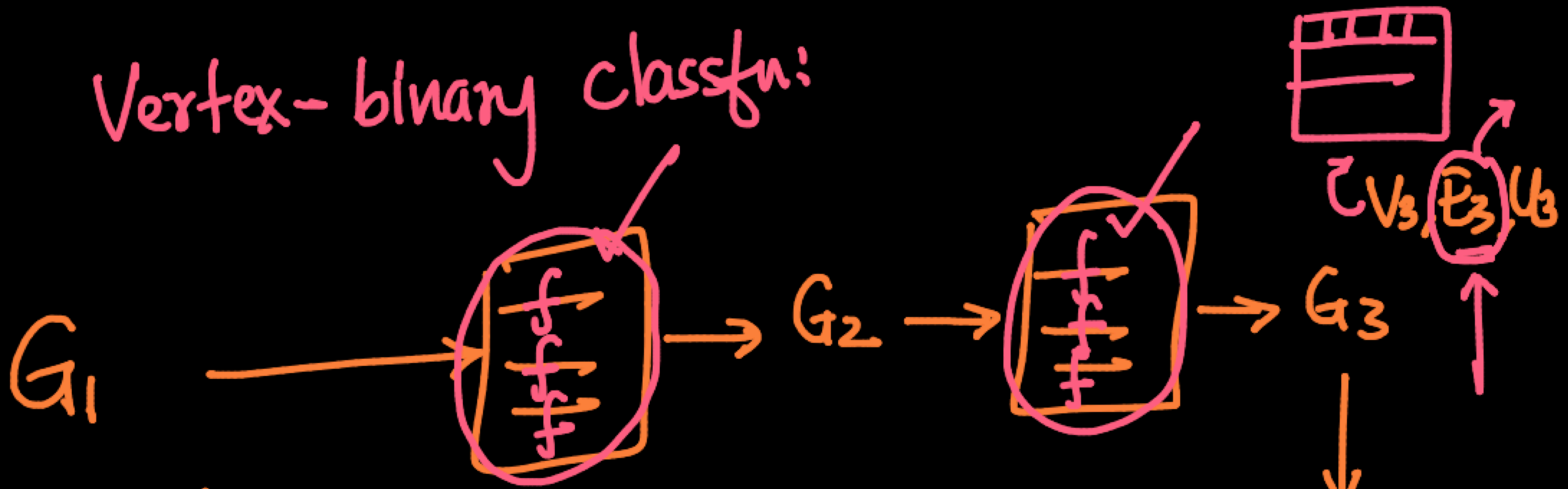
We will consider the case of binary classification, but this framework can easily be extended to the multi-class or regression case. If the task is to make binary predictions on nodes, and the graph already contains node information, the approach is straightforward—for each node embedding, apply a linear classifier.



However, it is not always so simple. For instance, you might have information in the graph stored in s. We need a way to

We could network, anonymiz (nodes) b and only u data (edg of such a node task the Node- subsectio club exam just using meetings to determ Mr. Hi or

Vertex- binary classfn:



end-end

node-classfn-

Simplest/dumbest GNN

collect information from edges and give them to nodes for prediction. We can do this by *pooling*.

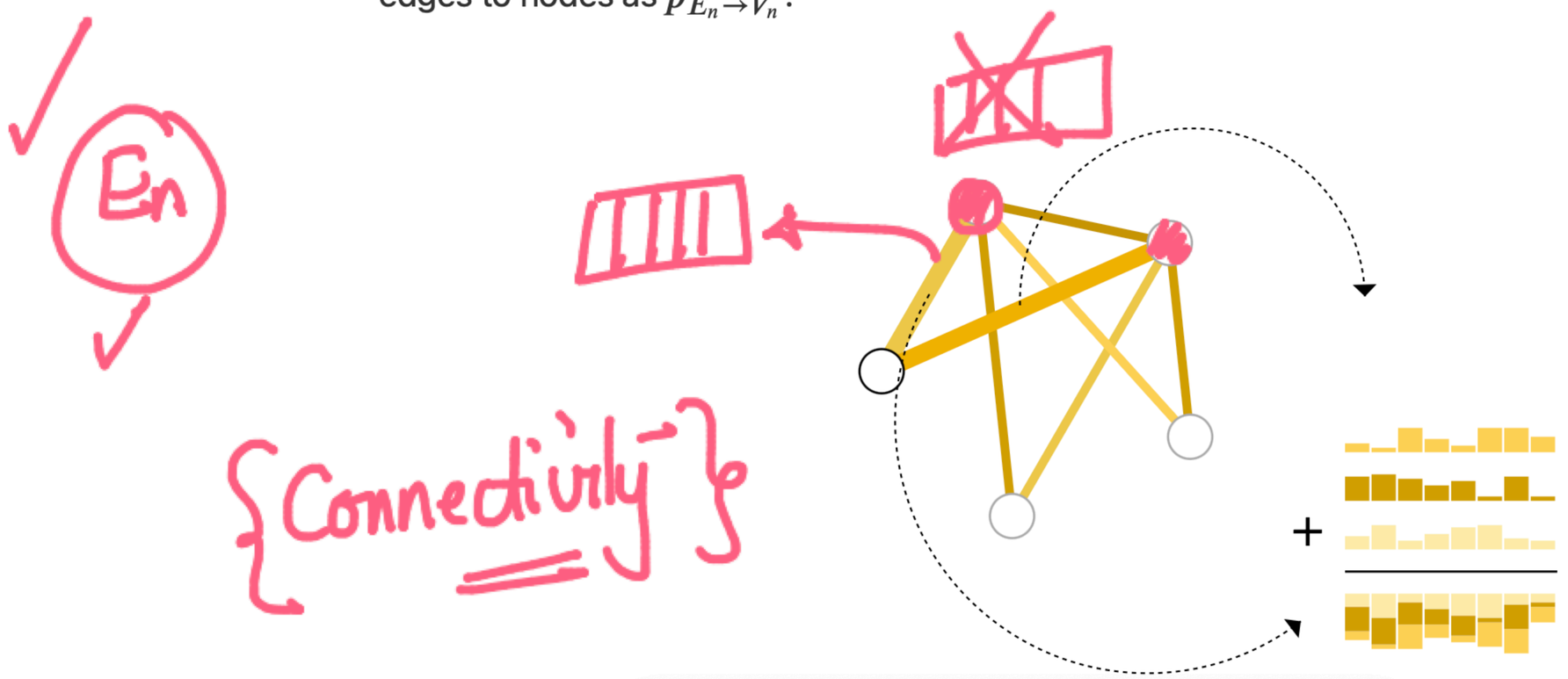
Pooling proceeds in two steps:

Case 1:

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

vertex -level- task

We represent the *pooling* operation by the letter ρ, and denote that we are gathering information from edges to nodes as $p_{E_n \to V_n}$.

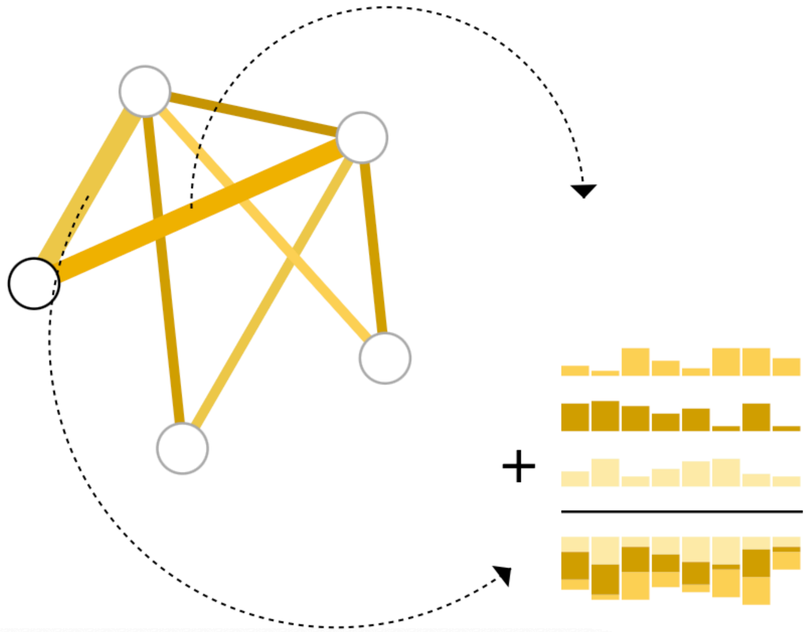


collect information from edges and give them to nodes for prediction. We can do this by *pooling*. Pooling proceeds in two steps:

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

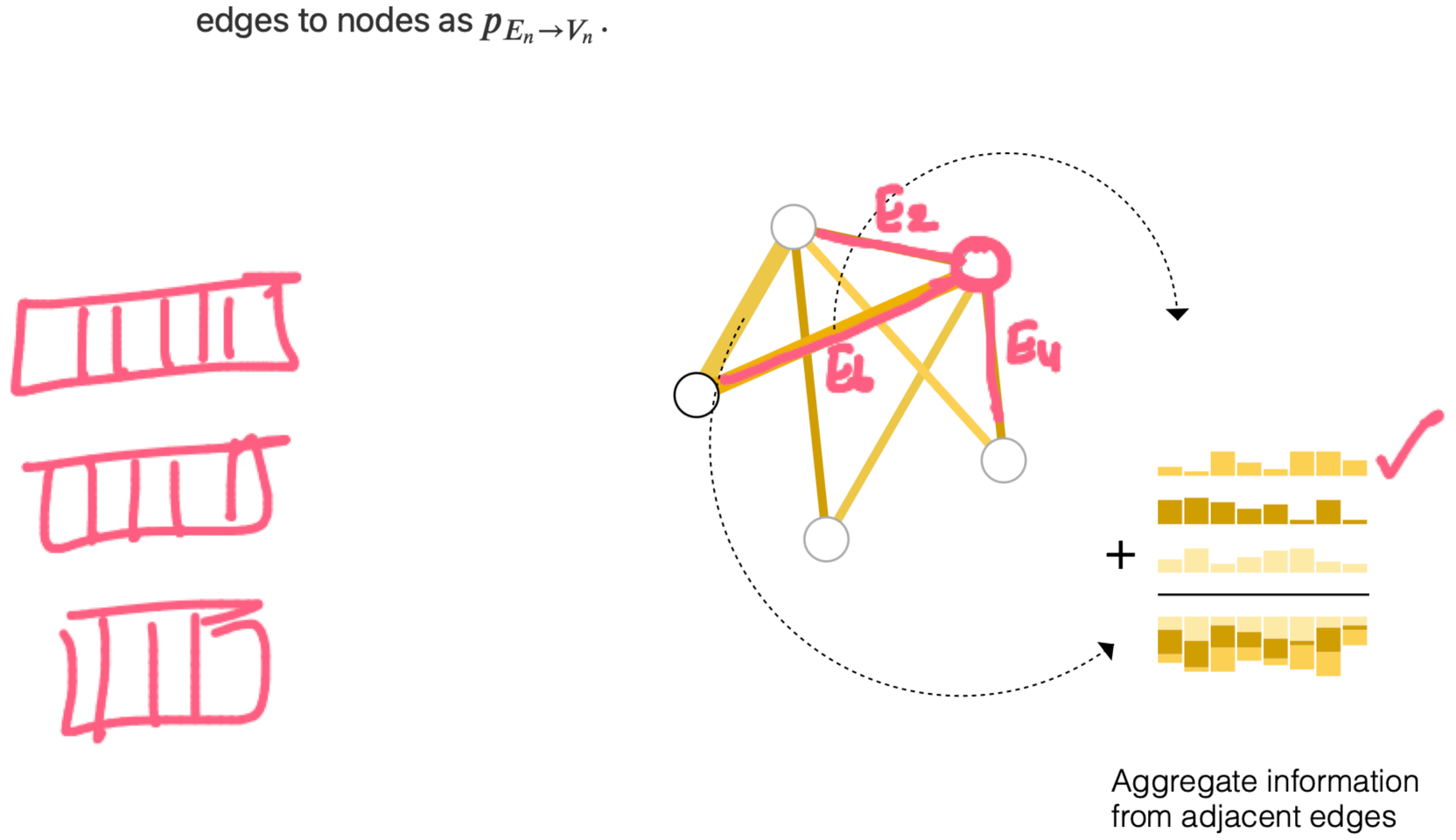
For a mor
discussio
operation
Comparin
operation

We represent the *pooling* operation by the letter ρ, and denote that we are gathering information from edges to nodes as $p_{E_n \to V_n}$.



operation

We represent the *pooling* operation by the letter ρ, and denote that we are gathering information from edges to nodes as $p_{E_n \to V_n}$.



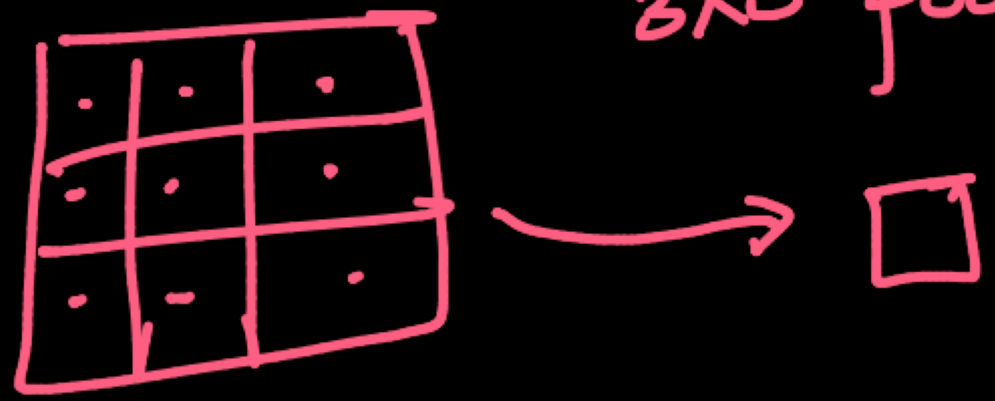
Hover over a node (black node) to visualize which edges are gathered and aggregated to produce an embedding for that target node.

formation, we can use

Max pooling

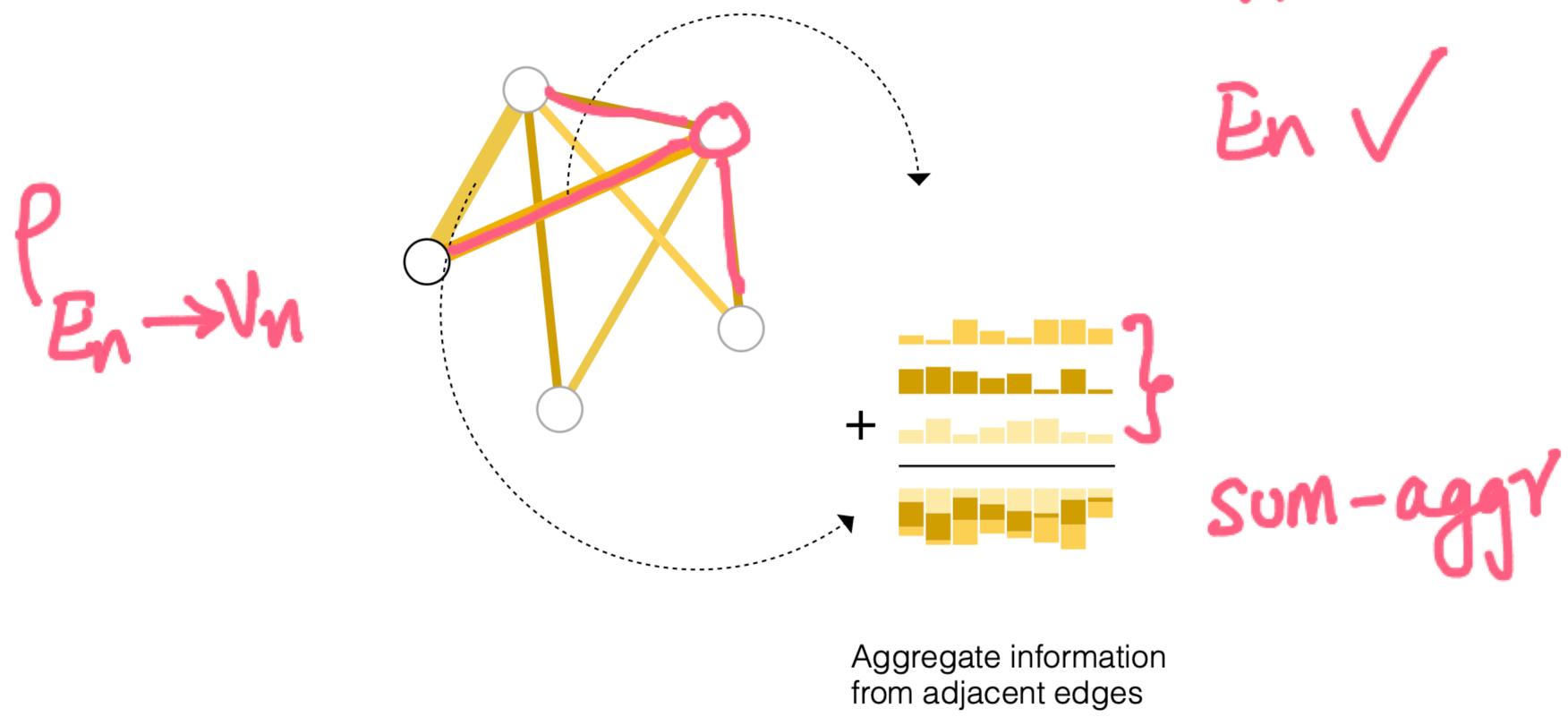
mean pooling

3x3 pooling



operation

We represent the *pooling* operation by the letter ρ, and denote that we are gathering information from edges to nodes as $p_{E_n \to V_n}$.

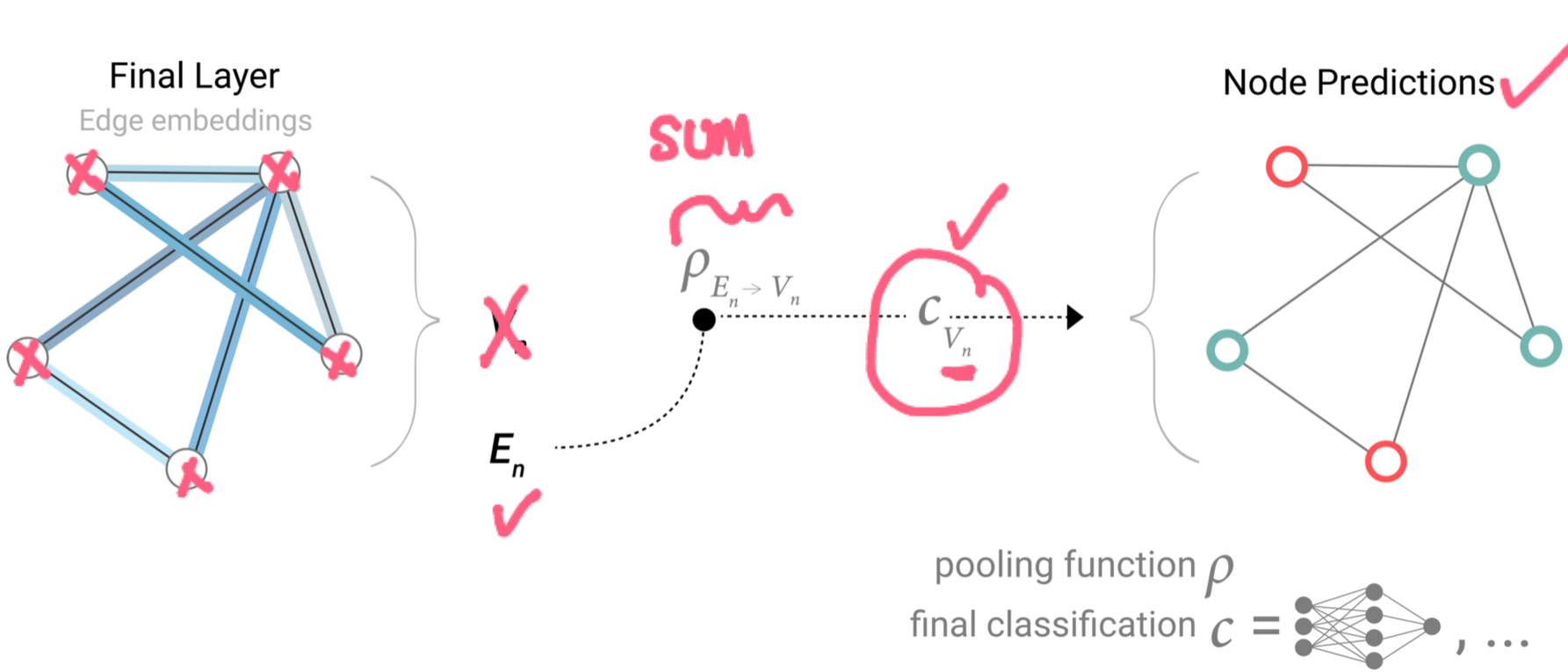


Hover over a node (black node) to visualize which edges are gathered and aggregated to produce an embedding for that target node.

formation, we can use

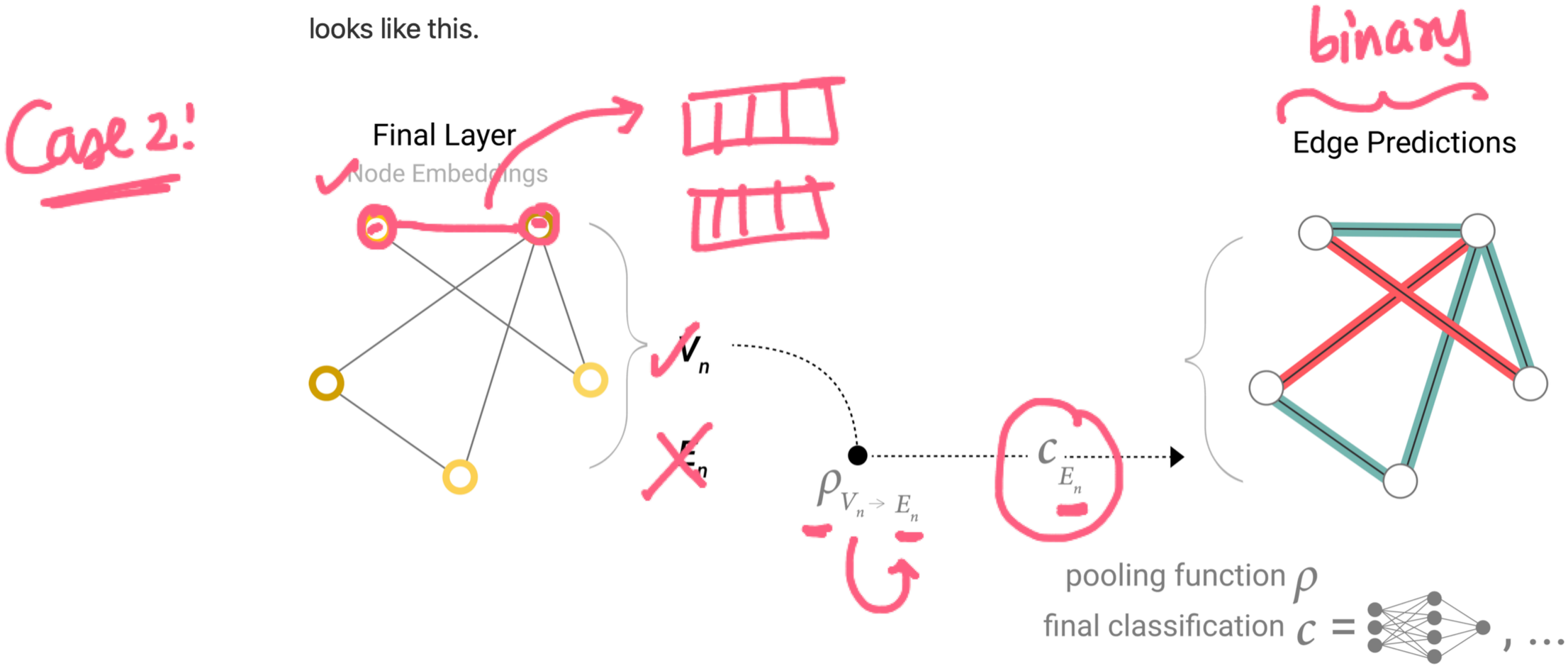
Hover over a node (black node) to visualize which edges are gathered and aggregated to produce an embedding for that target node.

So If we only have edge-level features, and are trying to predict binary node information, we can use pooling to route (or pass) information to where it needs to go. The model looks like this.



information, the model

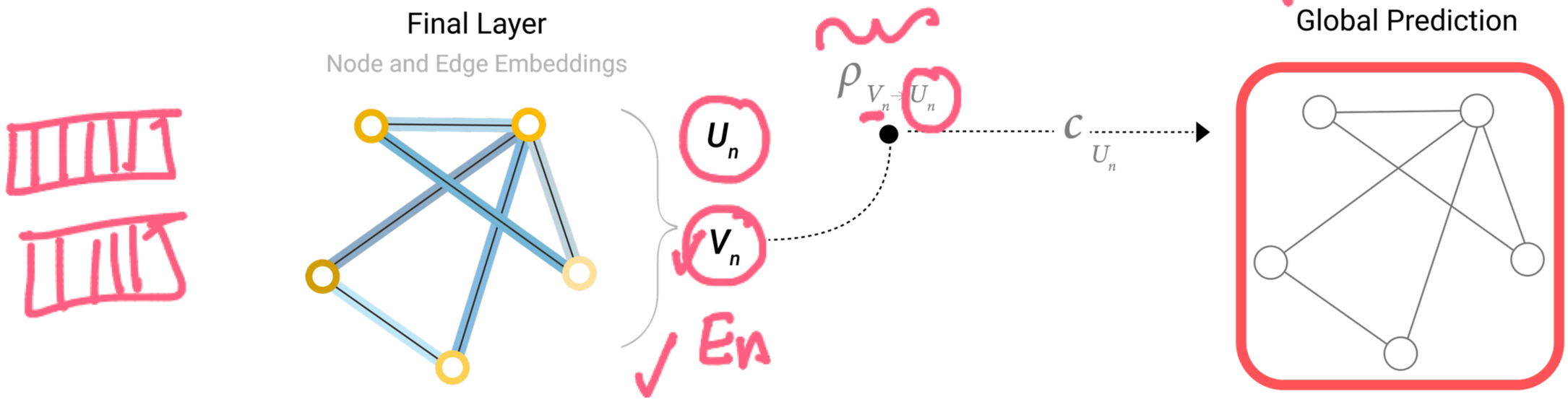
If we only have node-level features, and are trying to predict binary edge-level information, the model looks like this.



One exam
scenario i
we specif
task sub s
can be re
entities, a
to predict
share a re
edges).

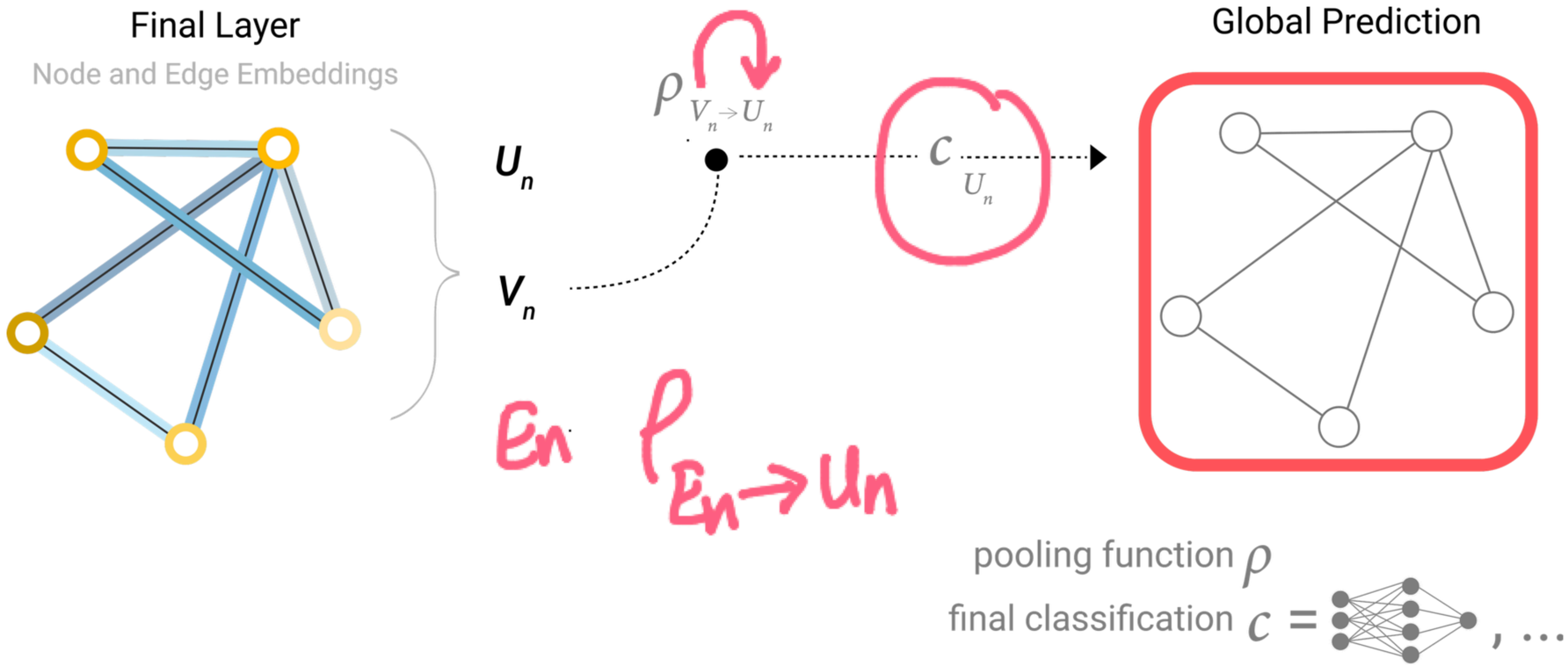
If we only have node-level features, and need to predict a binary global property, we need to gather all available node information together and aggregate them. This is similar to *Global Average Pooling* layers in CNNs. The same can be done for edges.

available node information together and aggregate them. This is similar to *Global Average Pooling* layers in CNNs. The same can be done for edges.



In our examples, the classification model c can easily be replaced with any differentiable model, or adapted to multi-class classification using a generalized linear model.

available node information together and aggregate them. This is similar to *Global Average Pooling* layers in CNNs. The same can be done for edges.



In our examples, the classification model c can easily be replaced with any differentiable model, or adapted to multi-class classification using a generalized linear model.

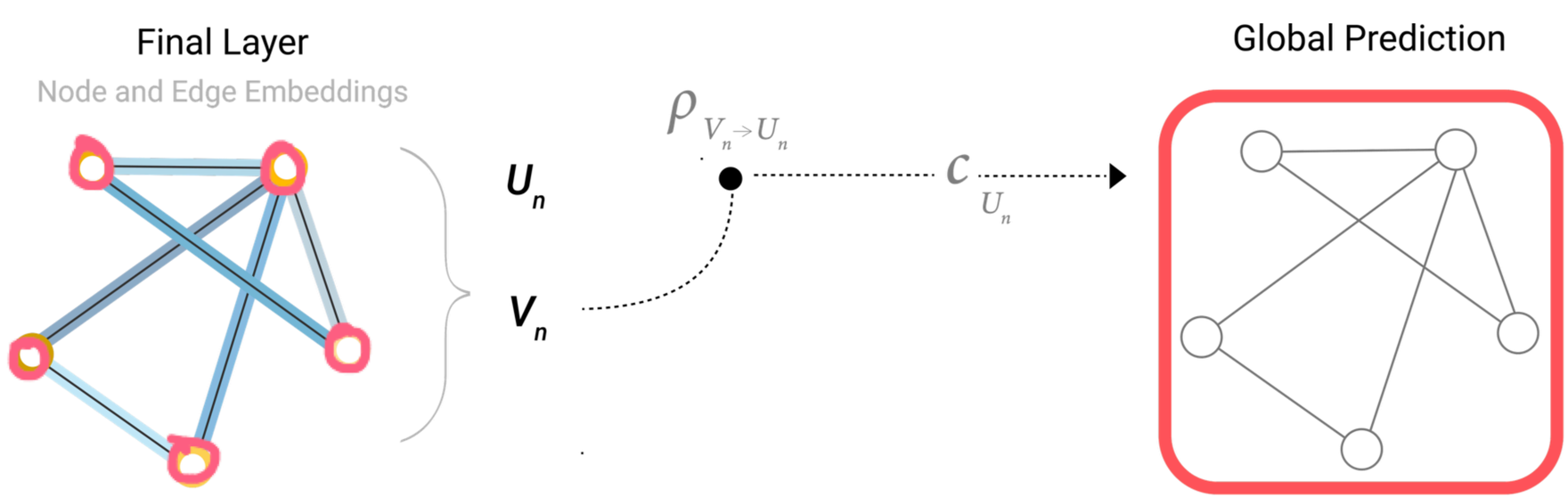
This is a c
for predic
properties
have atom
connectiv
like to kno
molecule
or if it has
(rose/not

pooling function ρ

final classification $c =$  , ...

If we only have node-level features, and need to predict a binary global property, we need to gather all available node information together and aggregate them. This is similar to *Global Average Pooling* layers in CNNs. The same can be done for edges.

This is a c
for predic
properties
have atom
connectiv
like to kno
molecule
or if it has
(rose/not

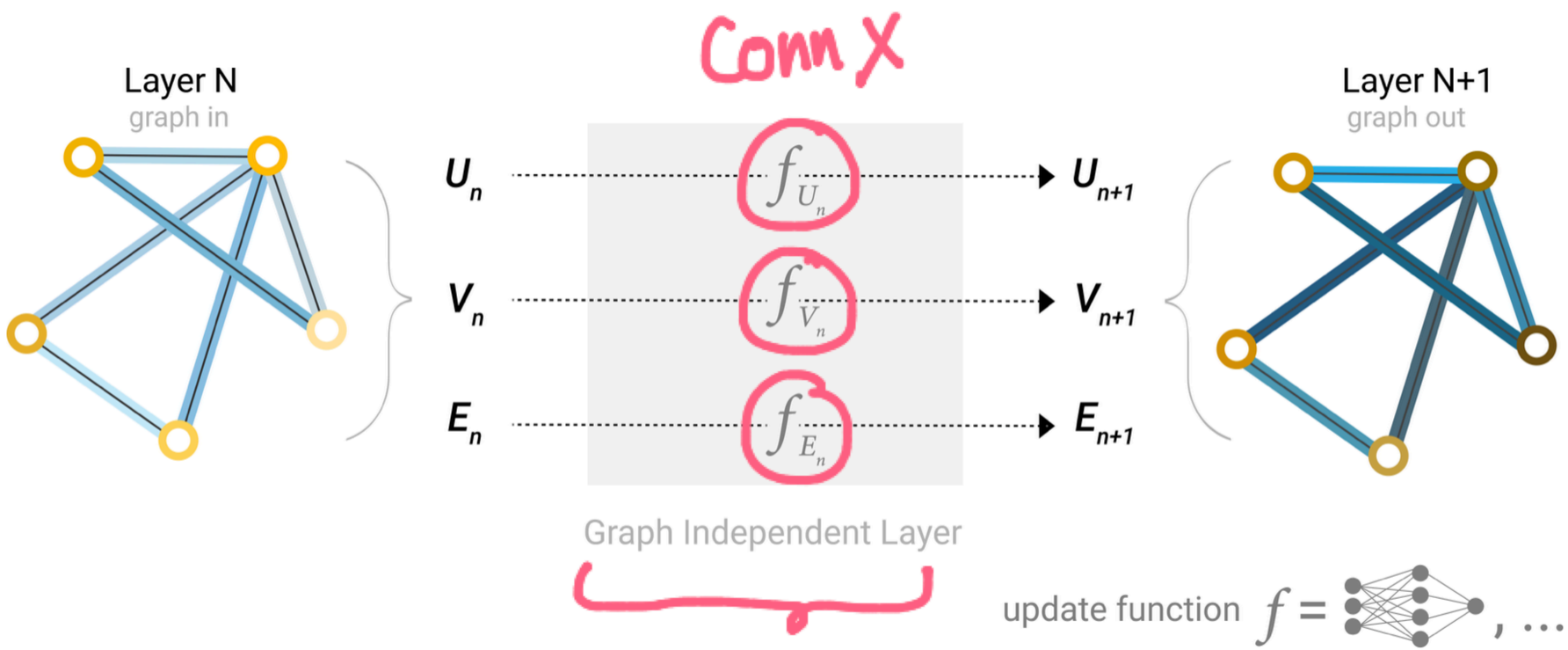


pooling function ρ

$c =$  , ...

component of a graph; we call this a GNN layer. For each node vector, we apply the MLP and get back a learned node-vector. We do the same for each edge, learning a per-edge embedding, and also for the global-context vector, learning a single embedding for the entire graph.

multiple o
(like a Res

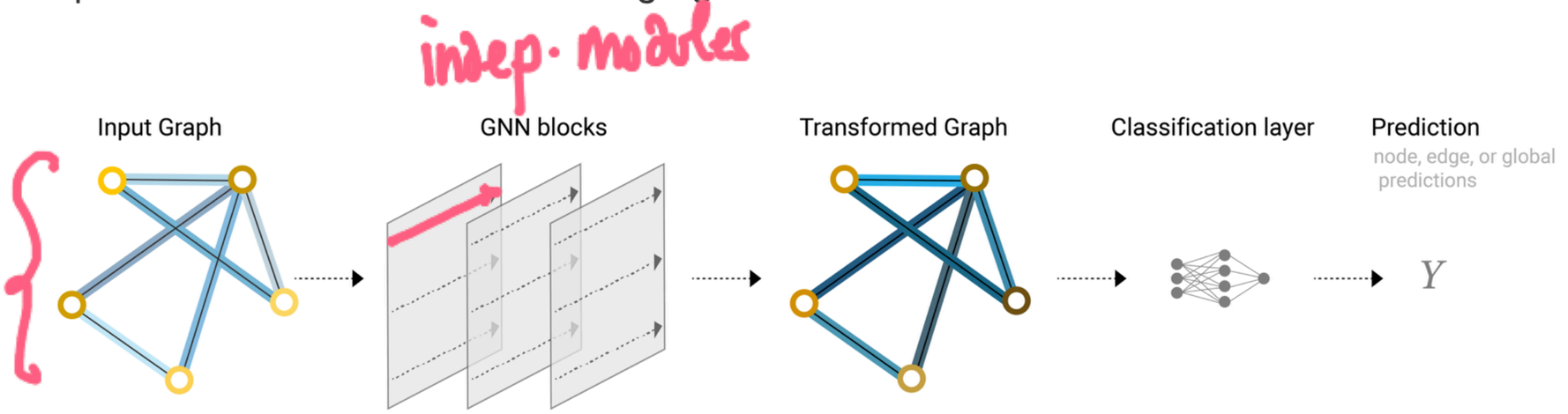


A single layer of a simple GNN. A graph is the input, and each component (V,E,U) gets updated by a MLP to produce a new graph. Each function subscript indicates a separate function for a different graph attribute at the n-th layer of a GNN model.

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

Because a GNN does not update the connectivity of the input graph, we can describe the output vectors as the input

In our examples, the classification model c can easily be replaced with any differentiable model, or adapted to multi-class classification using a generalized linear model.



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

Now we've demonstrated that we can build a simple GNN model, and make binary predictions by routing information between different parts of the graph. This pooling technique will serve as a building block for constructing more sophisticated GNN models. If we have new graph attributes, we just have to define how to pass information from one attribute to another.

Note that in this simplest GNN formulation, we're not using the connectivity of the graph at all inside the GNN layer. Each node is processed independently, as is each edge, as well as the global context.

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* [18], where neighboring nodes or edges exchange information and influence each other's updated embeddings.

Message passing works in three steps:

1. For each node in the graph, *gather* all the neighboring node embeddings (or messages), which is the g function described above.
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.

You could
messages
and 2) ag
still have
invariant

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

These steps are key for leveraging the connectivity of graphs. We will build more elaborate variants of
veness and power.

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* [18], where neighboring nodes or edges exchange information and influence each other's updated embeddings.

Message passing works in three steps:

1. For each node in the graph, *gather* all the neighboring node embeddings (or messages), which is the g function described above.
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.

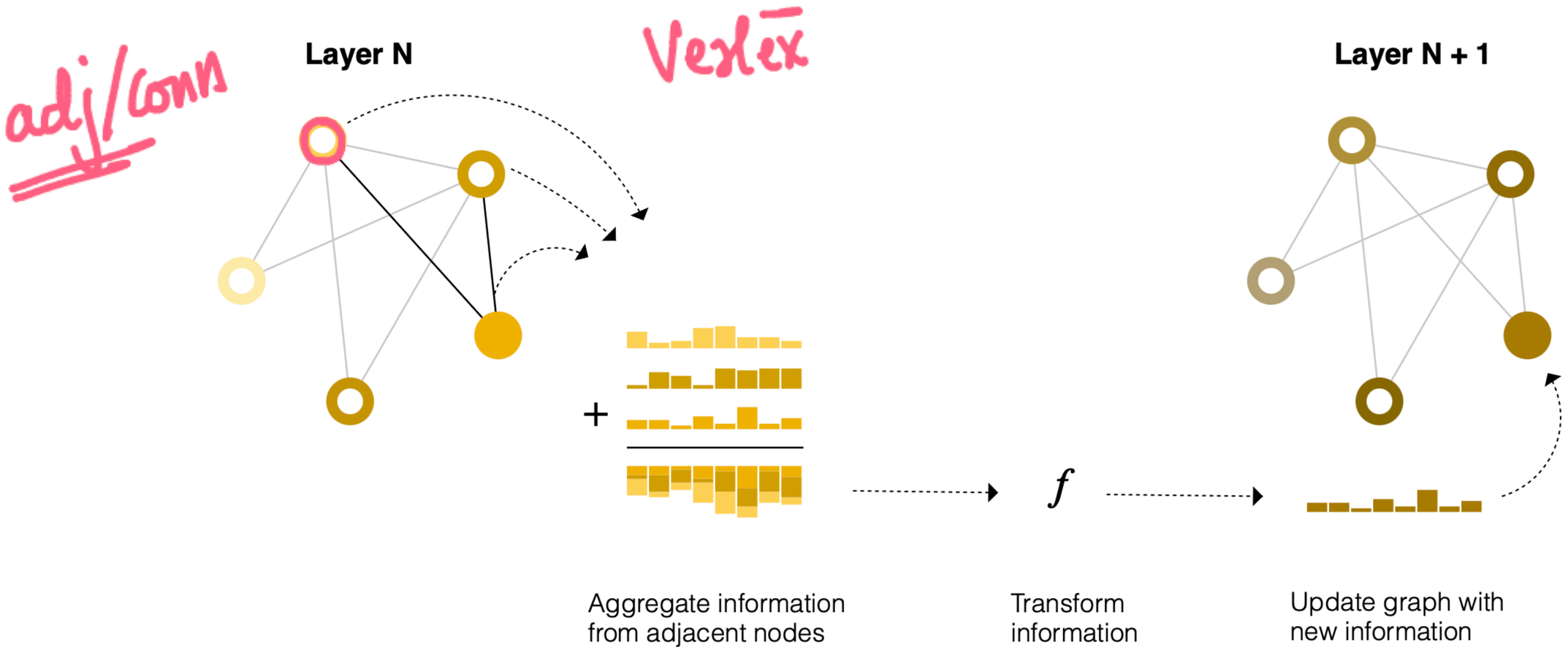
Just as pooling can be applied to either nodes or edges, message passing can occur between either nodes or edges.

These steps are key for leveraging the connectivity of graphs. We will build more elaborate variants of ...eness and power.

You could
messages
and 2) ag
still have
invariant

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

These steps are key for leveraging the connectivity of graphs. We will build more elaborate variants of message passing in GNN layers that yield GNN models of increasing expressiveness and power.

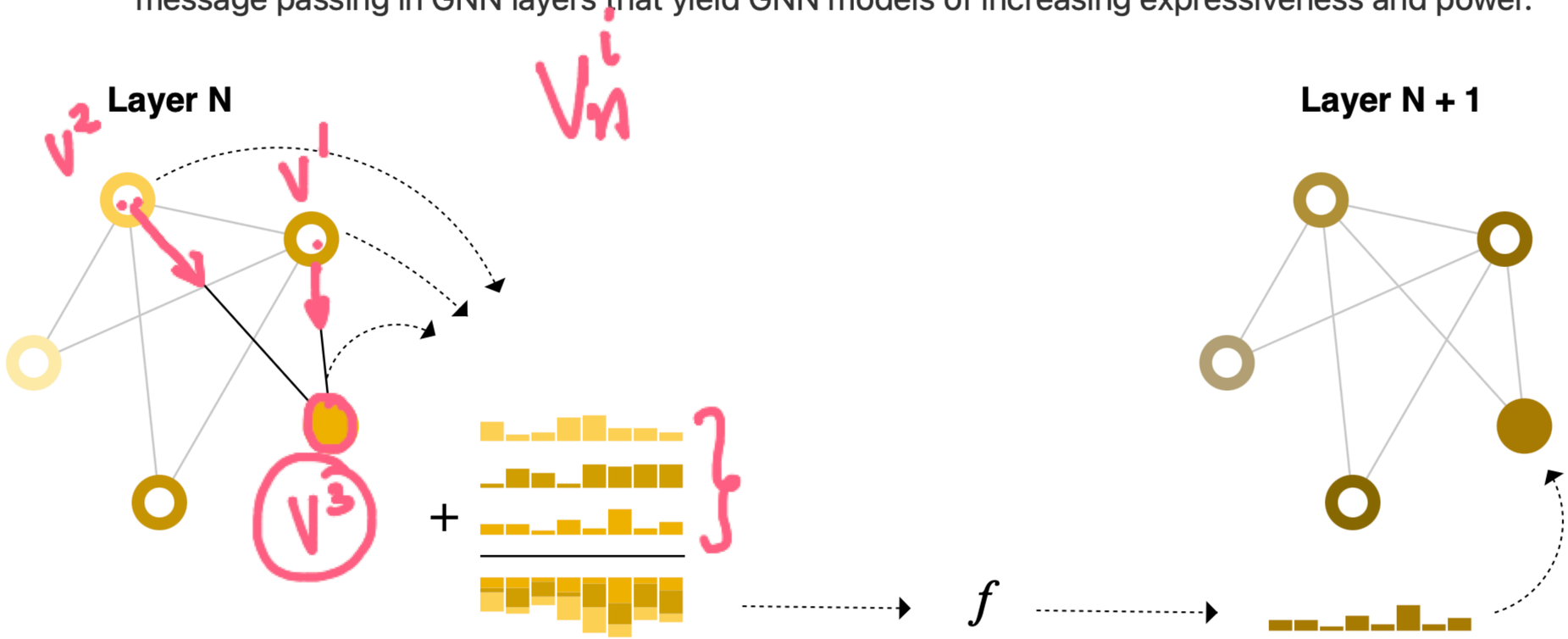


Hover over a node, to highlight adjacent nodes and visualize the adjacent embedding that would be pooled, updated and stored.

age-passing GNN layer

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

These steps are key for leveraging the connectivity of graphs. We will build more elaborate variants of message passing in GNN layers that yield GNN models of increasing expressiveness and power.

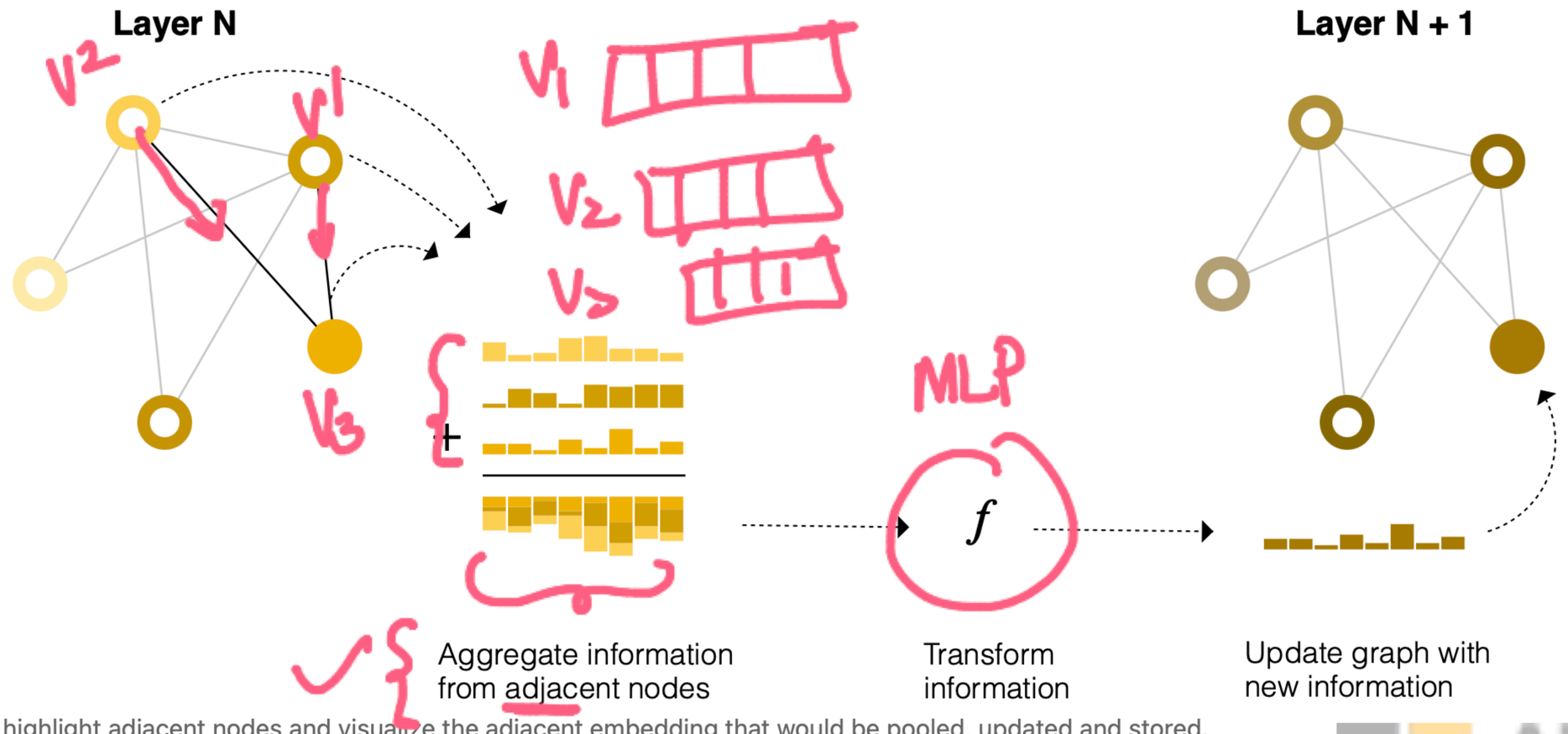


Hover over a node, to highlight adjacent nodes and visualize the adjacent embedding that would be pooled, updated and stored.

age-passing GNN layer

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

These steps are key for leveraging the connectivity of graphs. We will build more elaborate variants of message passing in GNN layers that yield GNN models of increasing expressiveness and power.



Hover over a node, to highlight adjacent nodes and visualize the adjacent embedding that would be pooled, updated and stored.

age-passing GNN layer

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* [18], where neighboring nodes or edges exchange information and influence each other's updated embeddings.

Message passing works in three steps:

1. For each node in the graph, *gather* all the neighboring node embeddings (or messages), which is the g function described above.
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.

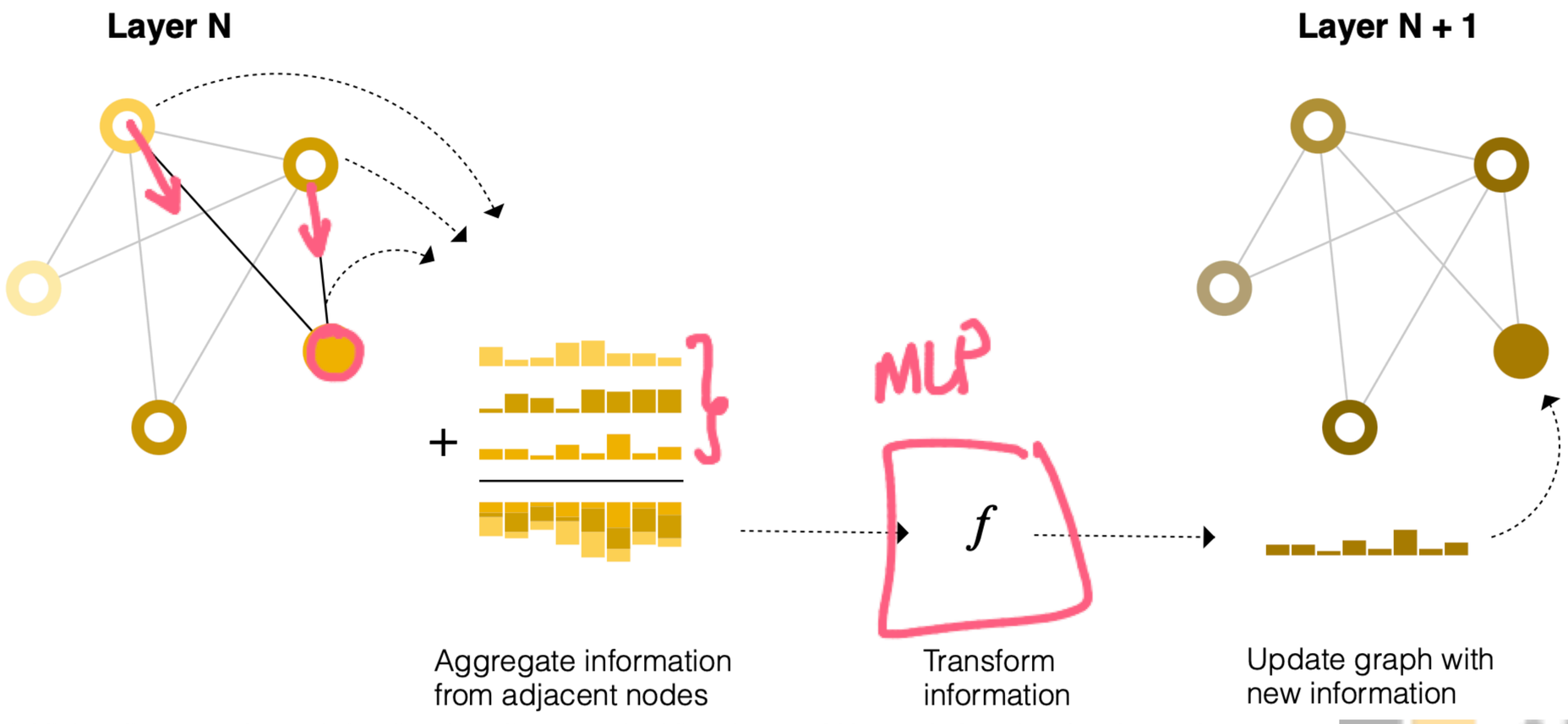
Just as pooling can be applied to either nodes or edges, message passing can occur between either nodes or edges.

These steps are key for leveraging the connectivity of graphs. We will build more elaborate variants of message passing in GNN layers that yield GNN models of increasing expressiveness and power.

You could
messages
and 2) ag
still have a
invariant o

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

These steps are key for leveraging the connectivity of graphs. We will build more elaborate variants of message passing in GNN layers that yield GNN models of increasing expressiveness and power.



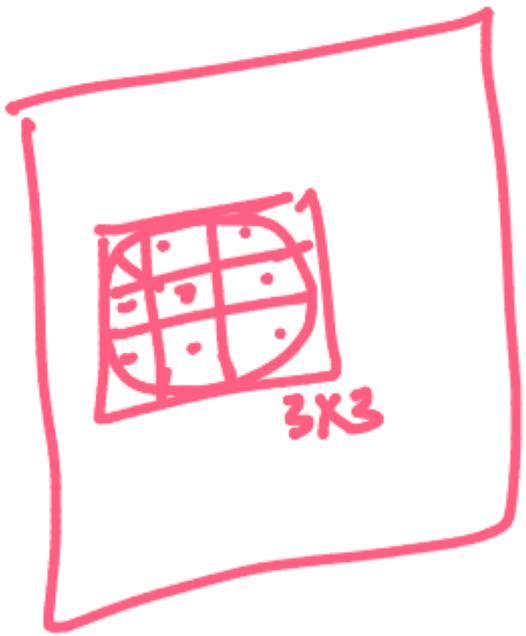
Hover over a node, to highlight adjacent nodes and visualize the adjacent embedding that would be pooled, updated and stored.

Aggregate information from adjacent nodes

Transform information

Update graph with new information

Hover over a node, to highlight adjacent nodes and visualize the adjacent embedding that would be pooled, updated and stored.

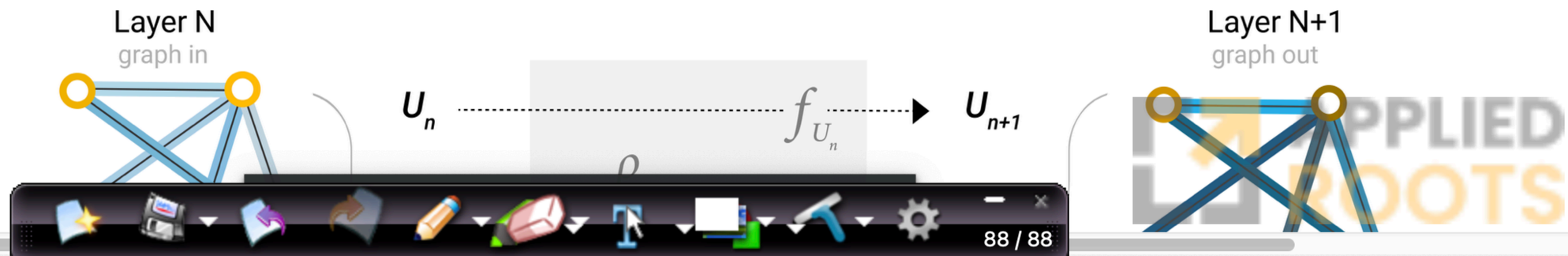


This sequence of operations, when applied once, is the simplest type of message-passing GNN layer.

This is reminiscent of standard convolution: in essence, message passing and convolution are operations to aggregate and process the information of an element's neighbors in order to update the element's value. In graphs, the element is a node, and in images, the element is a pixel. However, the number of neighboring nodes in a graph can be variable, unlike in an image where each pixel has a set number of neighboring elements.

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.

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



Aggregate information from adjacent nodes

Transform information

Update graph with new information

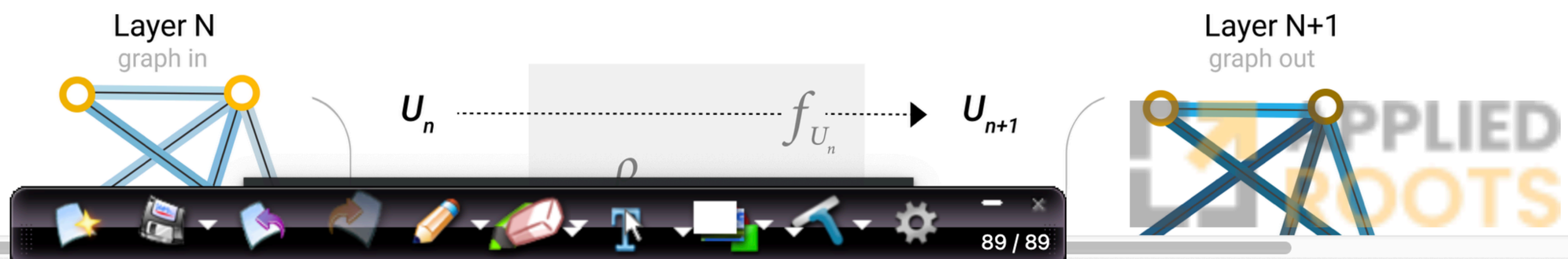
Hover over a node, to highlight adjacent nodes and visualize the adjacent embedding that would be pooled, updated and stored.

This sequence of operations, when applied once, is the simplest type of message-passing GNN layer.

This is reminiscent of standard convolution: in essence, message passing and convolution are operations to aggregate and process the information of an element's neighbors in order to update the element's value. In graphs, the element is a node, and in images, the element is a pixel. However, the number of neighboring nodes in a graph can be variable, unlike in an image where each pixel has a set number of neighboring elements.

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.

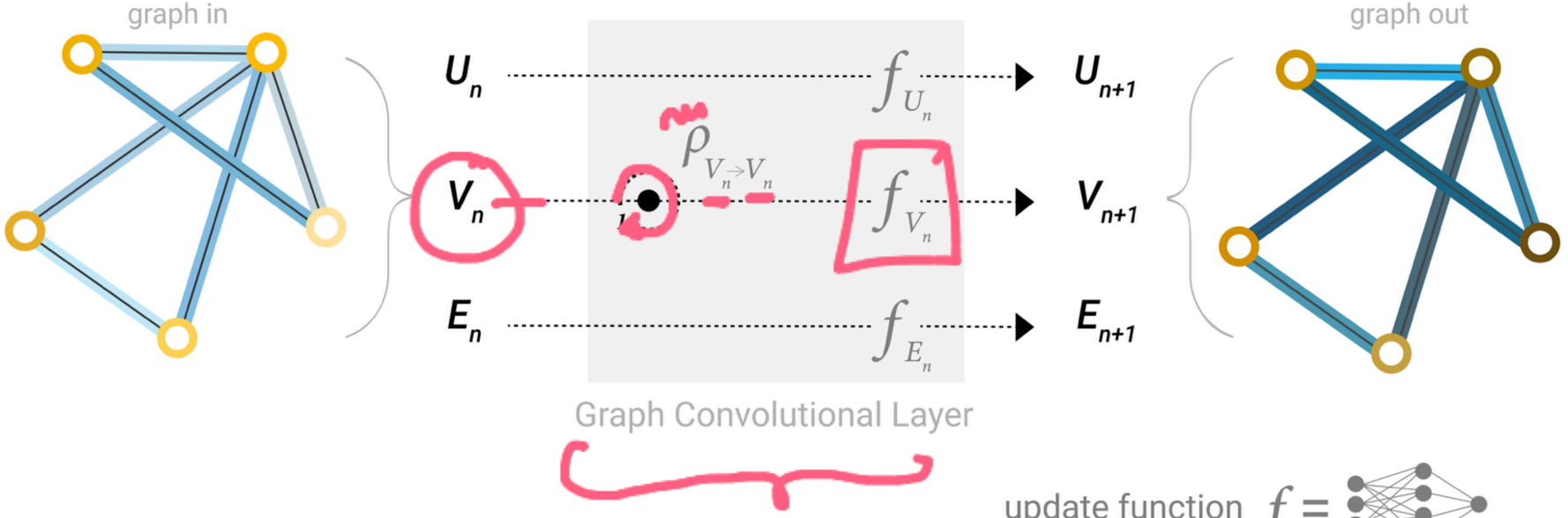
We can update our architecture diagram to include this new source of information for nodes:



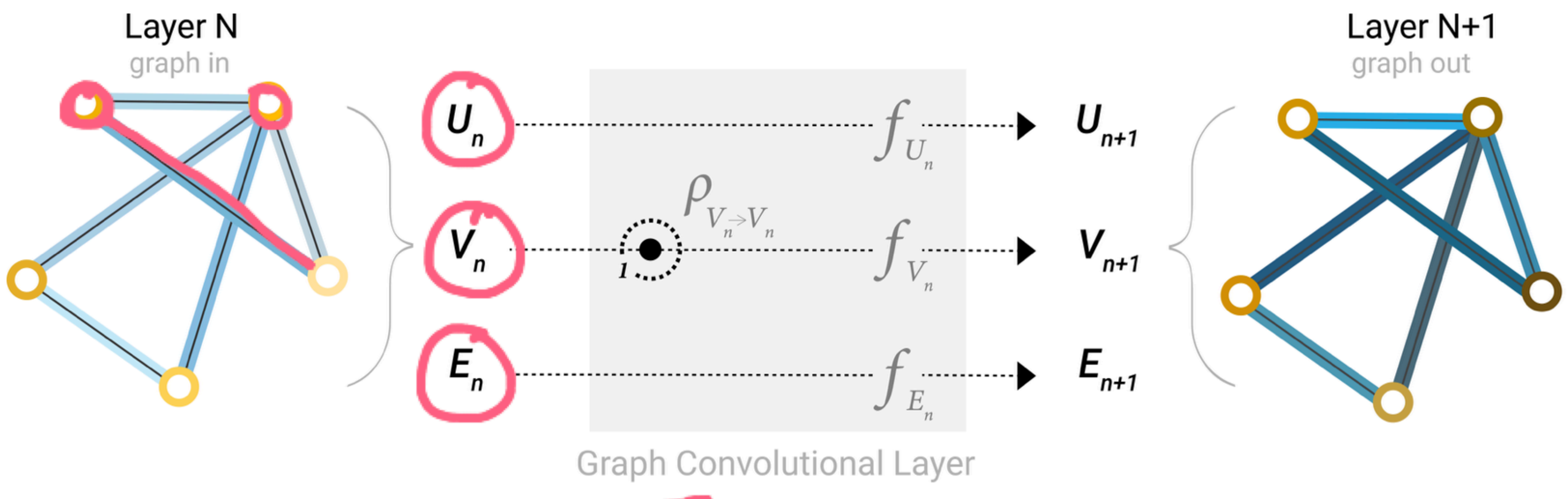
number of neighboring nodes in a graph can be variable, unlike in an image where each pixel has a set number of neighboring elements.

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.

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



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



Graph Convolutional Layer

update function $f = $, ...

pooling function ρ

Schematic for a GCN architecture, which updates node representations of a graph by pooling neighboring nodes at a distance of one degree.

Learning edge representations

global context). When

Airmeet

Graph Neural Nets & Geometric Learning Completed

View Airmeet

- Basic info
- Event Entry
- Schedule
- Speakers & Hosts
- Booths
- Sponsors
- Stage Backdrop
- Videos
- Live stream
- Branding
- Tickets

Keep the participants engaged even after the event is over with 'Event Replay'

Showcase your session recordings after the event has ended. Interactive features like chats, polls, Q&A, lounge, etc will be disabled. You have to end this Airmeet to enable Event Replay.

Setup Event Replay

Registrations BETA

Your current plan has 2000 registrations limit per event

44 Registrations

1956 registrations left as per your plan limit

Allow extra registrations Contact Sales

2%

Basic Info

Meetup

Graph Neural Nets & Geometric Learning

Edit

Starting on	Ending on	Timezone
Sunday, 26 Dec 2021	Sunday, 26 Dec 2021	UTC +05:30
07:00 pm	09:00 pm	IST