Graph Neural Networks

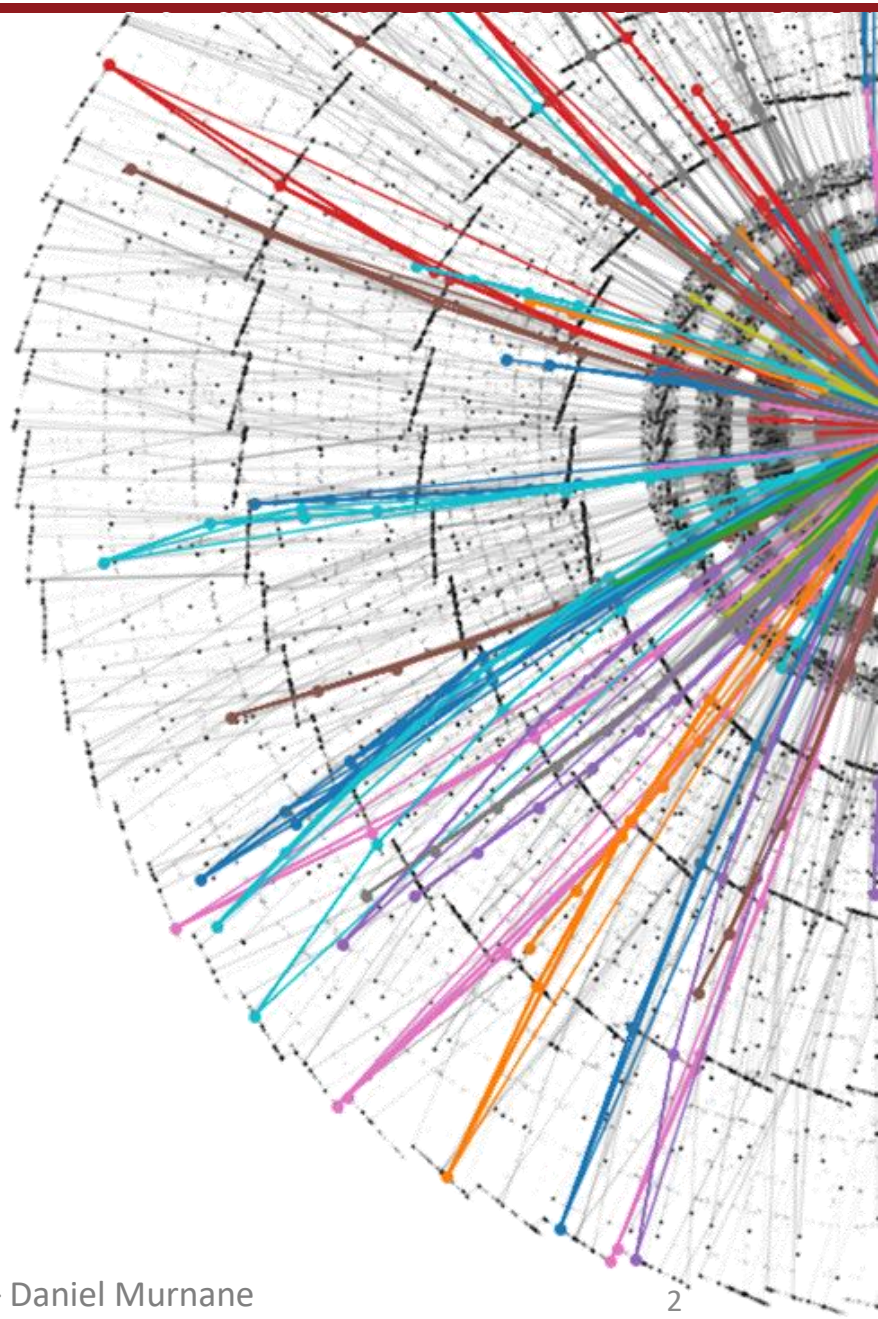
Applied Machine Learning, KU

Daniel Murnane - May 6th, 2026

UNIVERSITY OF COPENHAGEN

Introduction & Goals

- Goals for today:
  - Understand point cloud structure
  - Understand graph structure
  - Look at some real data that is represented as a graph
  - Understand how to generalize from grid (image) to a graph
  - Think about some choices of GNN architectures
  - Zoom in to the Transformer as a kind of GNN
- Have borrowed content from Troels' slides from previous years!

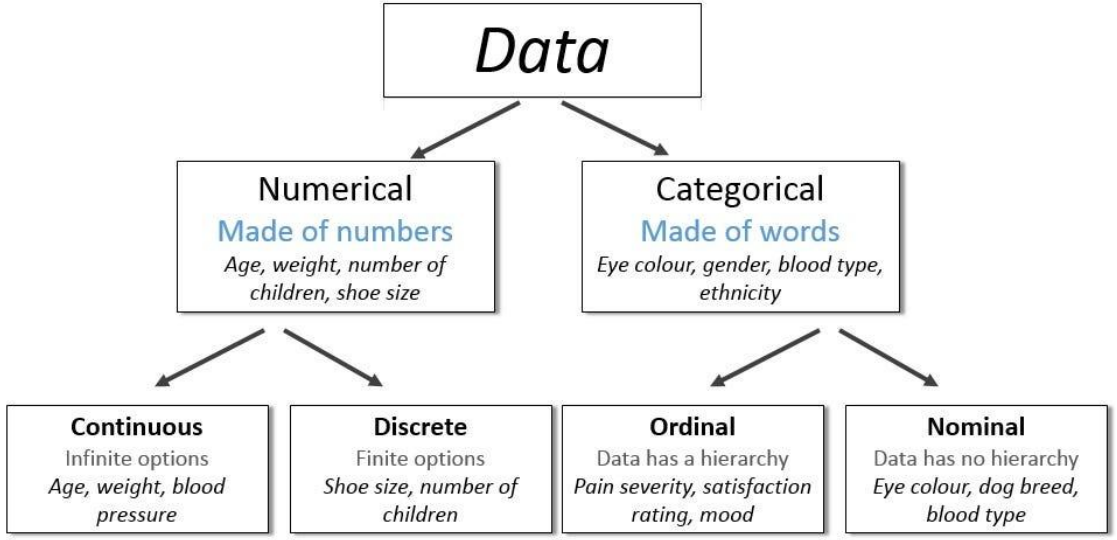


How can we represent data

UNIVERSITY OF COPENHAGEN

Data types

- Let's tidy up our language: There are data types, and data structures
- A data type is an input to our neural network, it falls into one of the following types



Data structures

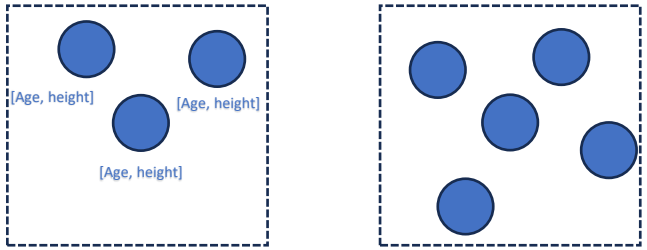
- Let’s tidy up our language: There are data types, and data structures
- A data type is an input to our neural network, it falls into one of the following types
- A data structure defines how each component of our dataset is related
- There are many data structures…

Data structures

- **Tabular:** Each sample has one entry

Data structures

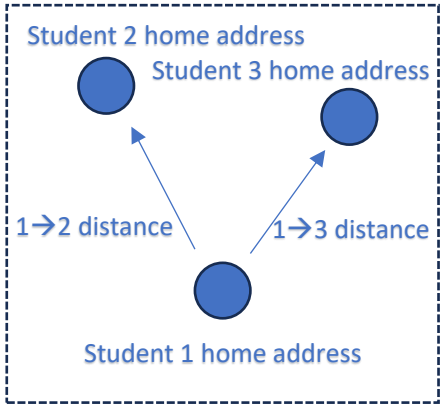
- **Tabular:** Each sample has one entry
- **Set:** Each sample has a variable number of entries



Class 1 Class 2

Data structures

- **Tabular:** Each sample has one entry
- **Set:** Each sample has a variable number of entries
- **Point Cloud:** Each sample has a variable number of entries, with some notion of distance



Class 1

Data structures

- Tabular: Each sample has one entry
- Set: Each sample has a variable number of entries
- Point Cloud: Each sample has a variable number of entries, with some notion of distance
- Grid: Each sample has a fixed number of entries, binned into a grid, with a natural distance

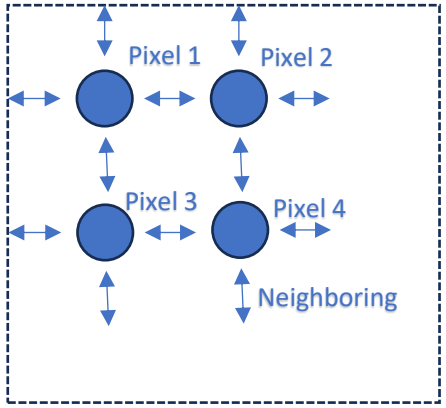
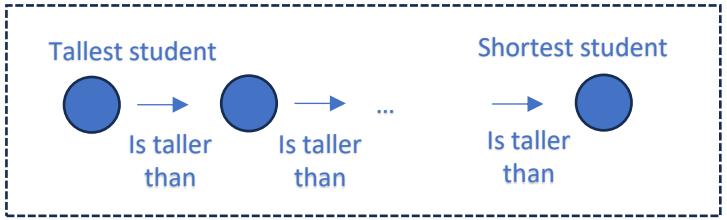


Image 1

Data structures

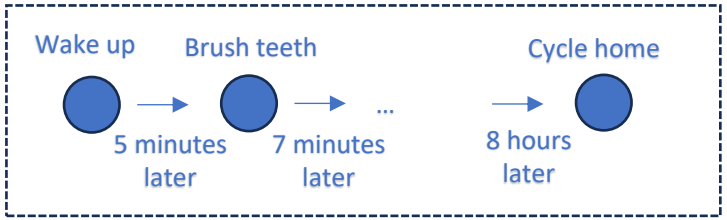
- Tabular: Each sample has one entry
- Set: Each sample has a variable number of entries
- Point Cloud: Each sample has a variable number of entries, with some notion of distance
- Grid: Each sample has a fixed number of entries, binned into a grid, with a natural distance
- Sequence: Each sample has an ordered list of variable number of entries



Class 1

Data structures

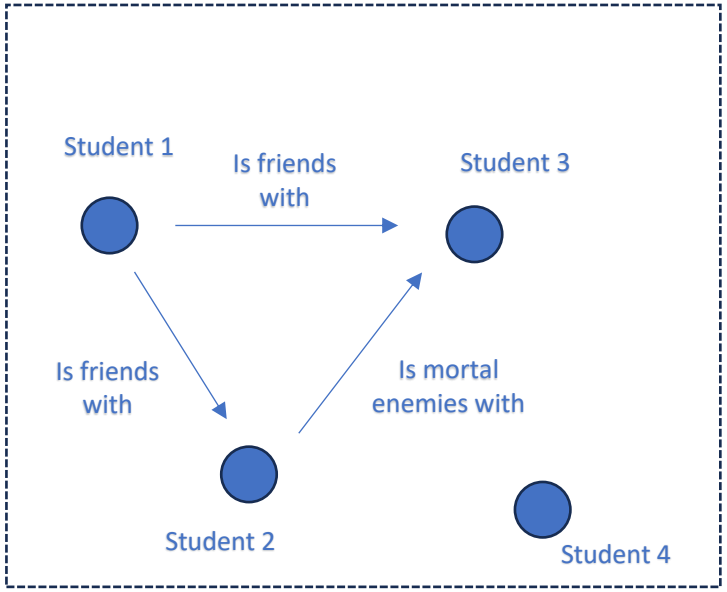
- Tabular: Each sample has one entry
- Set: Each sample has a variable number of entries
- Point Cloud: Each sample has a variable number of entries, with some notion of distance
- Grid: Each sample has a fixed number of entries, binned into a grid, with a natural distance
- Sequence: Each sample has an ordered list of variable number of entries
- Time series: Each sample has an ordered list of variable number of entries, with a neighbour distance given by time



Day 1

Data structures

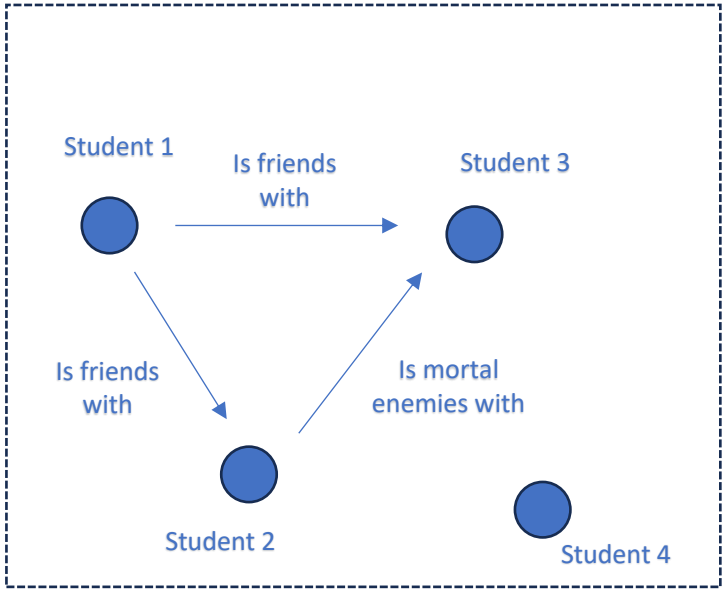
- Tabular: Each sample has one entry
- Set: Each sample has a variable number of entries
- Point Cloud: Each sample has a variable number of entries, with some notion of distance
- Grid: Each sample has a fixed number of entries, binned into a grid, with a natural distance
- Sequence: Each sample has an ordered list of variable number of entries
- Time series: Each sample has an ordered list of variable number of entries, with a neighbour distance given by time
- Graph: Each sample has a variable number of entries, with neighbourhoods given by explicit pairwise relationships



Class 1

Data structures

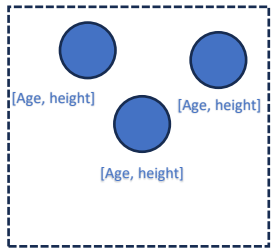
- Graph: Each sample has a variable number of entries, with neighbourhoods given by explicit pairwise relationships
- A graph is a collection of *nodes* (objects or entries) and *edges* (relationships between each object)
- Nodes can have features, edges can have features
- A graph may also have graph-level or "global" properties, e.g. `class_1["topic"] = "applied ML"`



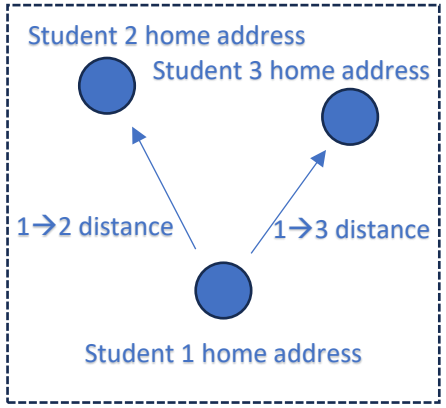
Class 1 = Applied ML

Data structures

- I encourage you to look at all data structures through the eyes of a graph



Class 1



Class 1

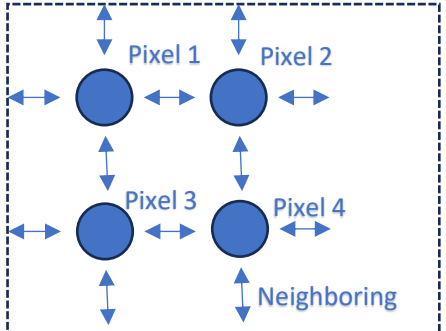
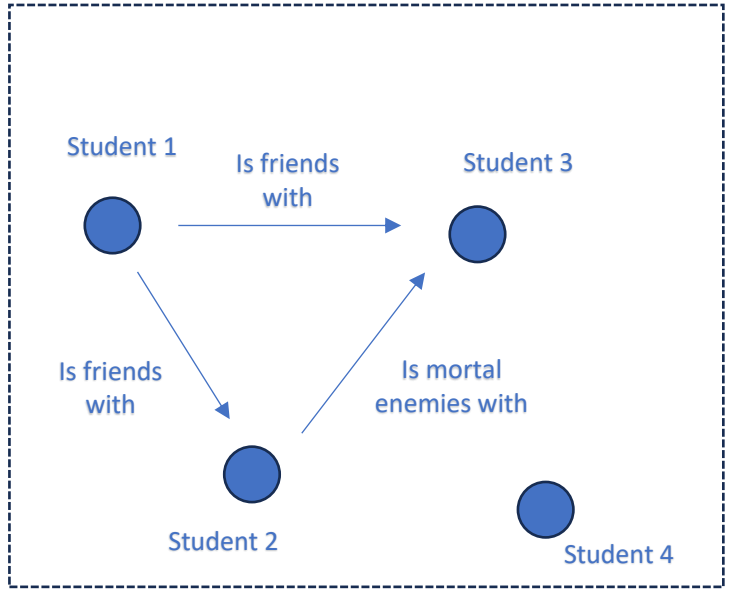
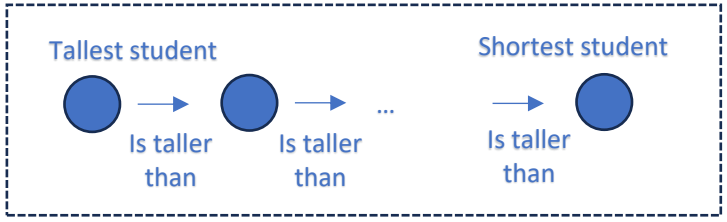


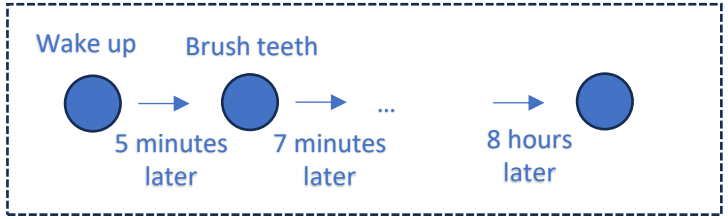
Image 1



Class 1 = Applied ML



Class 1

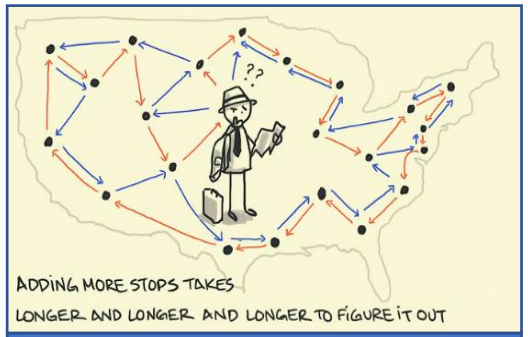


Day 1

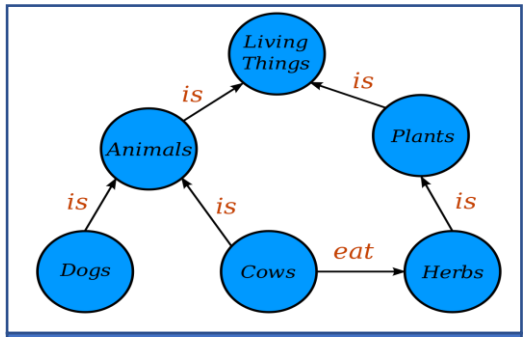
Graph Data

UNIVERSITY OF COPENHAGEN

Classic Problems with Graphs



Travelling Salesman Problem



Knowledge Graph Comprehension

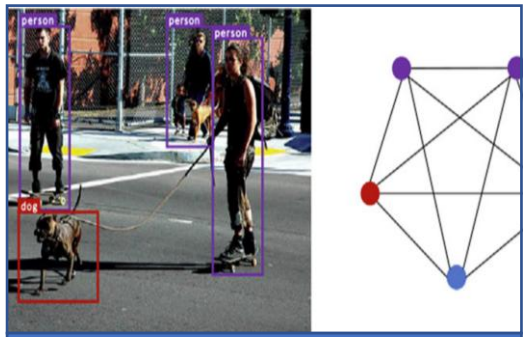
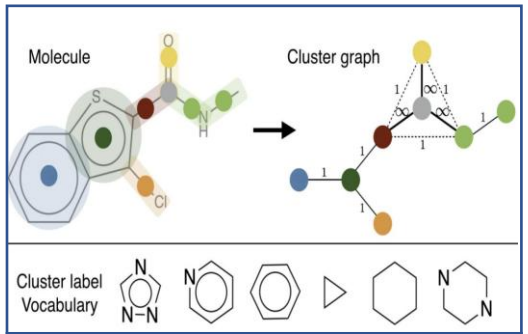
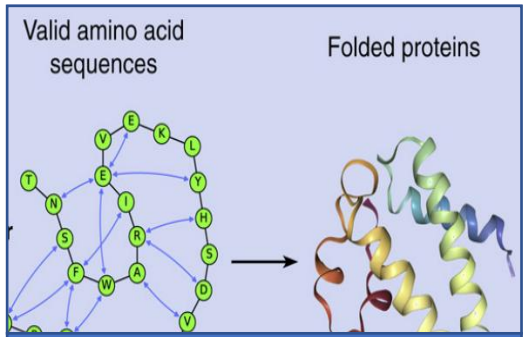


Image Comprehension

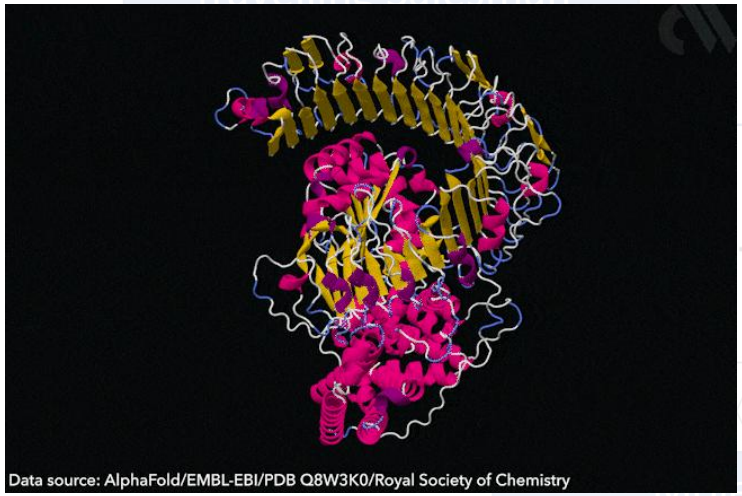


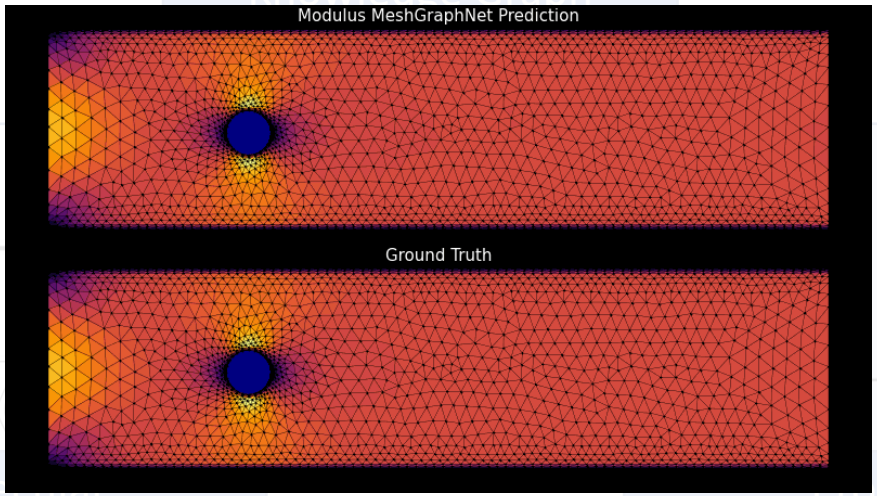
Molecular Chemistry

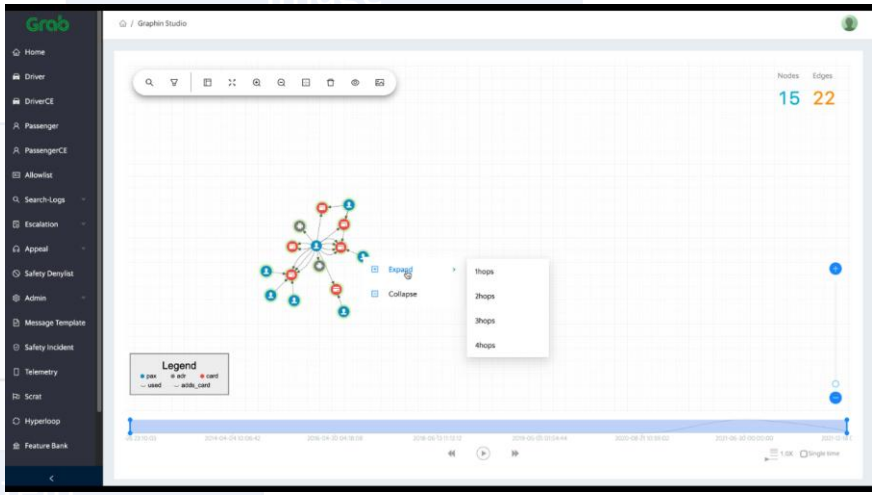


Protein Comprehension

Graph Neural Networks Eat These Up





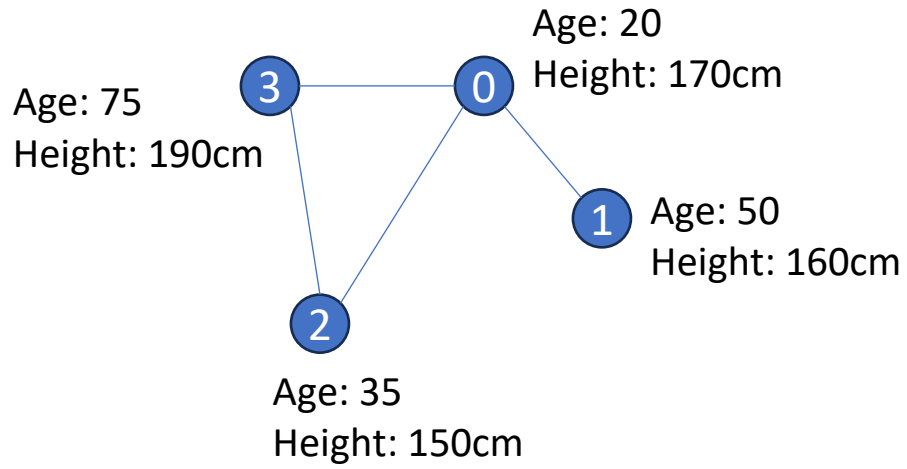


Case Study

- DeepMind worked with Google Maps developers to apply a GNN to estimate travel time
- Improved estimates by up to 50% in large cities
- Maps are *really* hard to optimize on, as they exhibit combinatorial behaviour
- GNNs can give fast, approximate solutions to map problems

Representing a Graph

- Nodes – a list of node vectors n_i^k, where $i = 0, \ldots, N_n$ up to number of nodes, $k = 0, \ldots, N_f$ up to number of features
- Edges – an adjacency matrix A_{ij}, where $A_{ij} = 1$ when there is an edge between node i and node j, and 0 otherwise
- In practice, A_{ij} is mostly 0s, so let's represent it with a more efficient structure: E_{ij}, which is a list of *pairs* of node indices



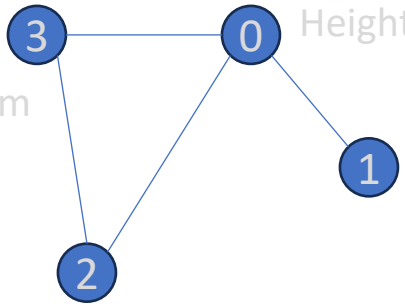
$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix} \quad A_{ij} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Representing a Graph

- Nodes – a list of node vectors n_i^k, where $i = 0, \ldots, N_n$ up to number of nodes, $k = 0, \ldots, N_f$ up to number of features
- Edges – usually represented with A_{ij}, which is 1 if there are edges between node i and node j, and 0 otherwise
- In practice, A_{ij} is mostly 0s, so let's represent with a more efficient structure: E_{ij}, which is a list of *pairs* of node indices

Happen to have the same number of entries, but A grows as N_{nodes}^2, E grows as N_{edges}



Age: 75
Height: 190cm

Age: 20
Height: 170cm

Age: 50
Height: 160cm

Age: 35
Height: 150cm

$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$A_{ij} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

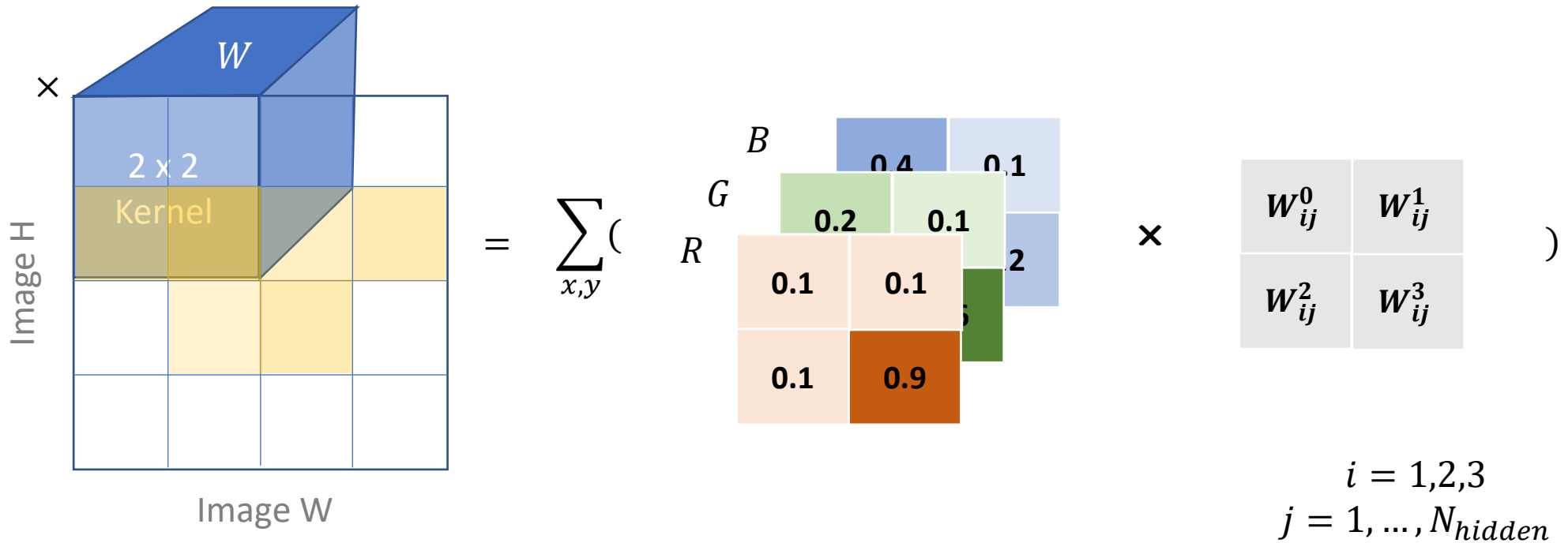
$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Removing the Grid

UNIVERSITY OF COPENHAGEN

Recall the CNN

- Remember how the image convolution worked on a $k \times k$ image, with a 2×2 window:

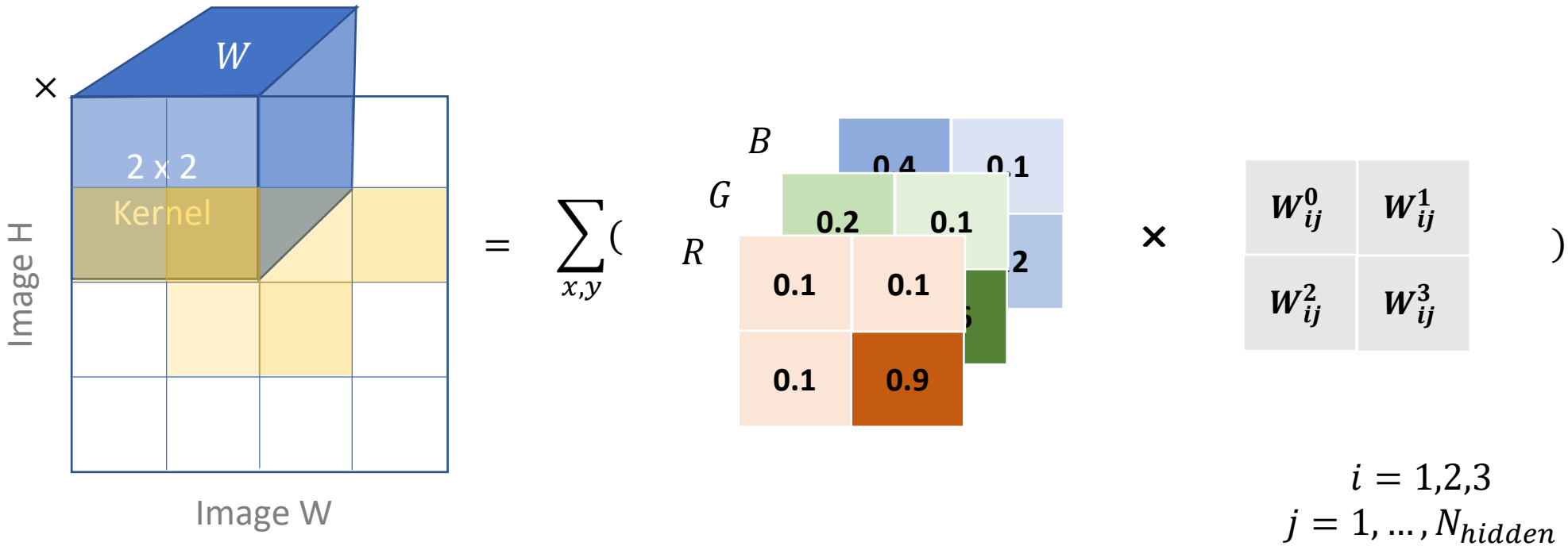


$i = 1,2,3$

$j = 1, \ldots, N_{hidden}$

Recall the CNN

- Let's rewrite this big tensor multiplication into smaller pieces…

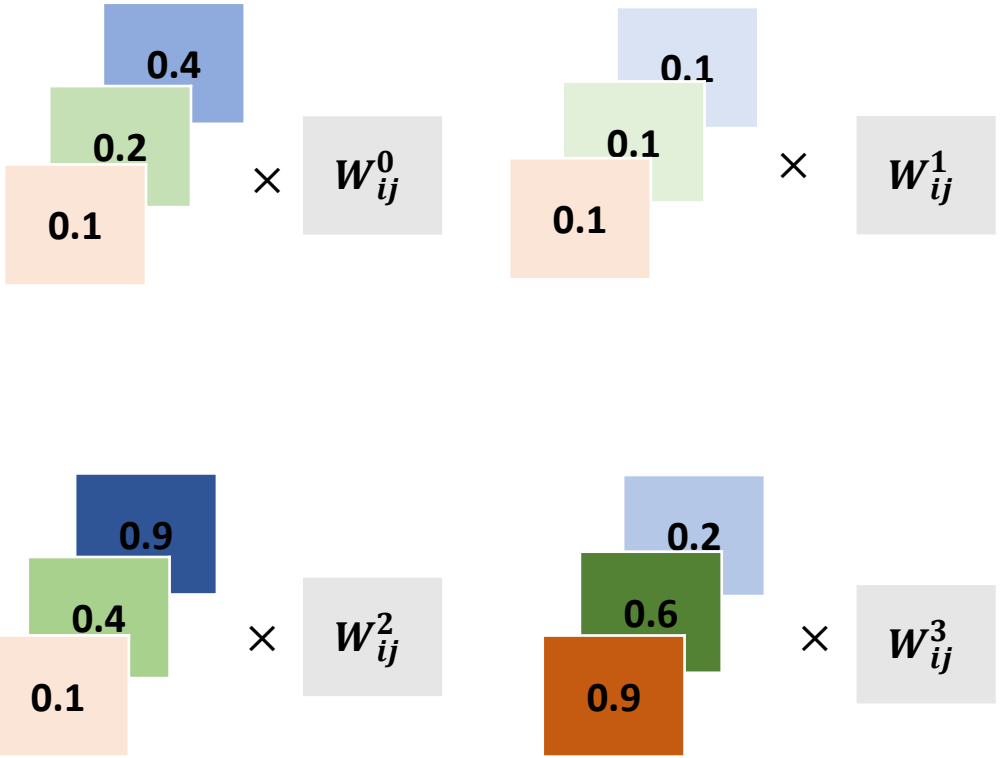


$i = 1,2,3$

$j = 1, \ldots, N_{hidden}$

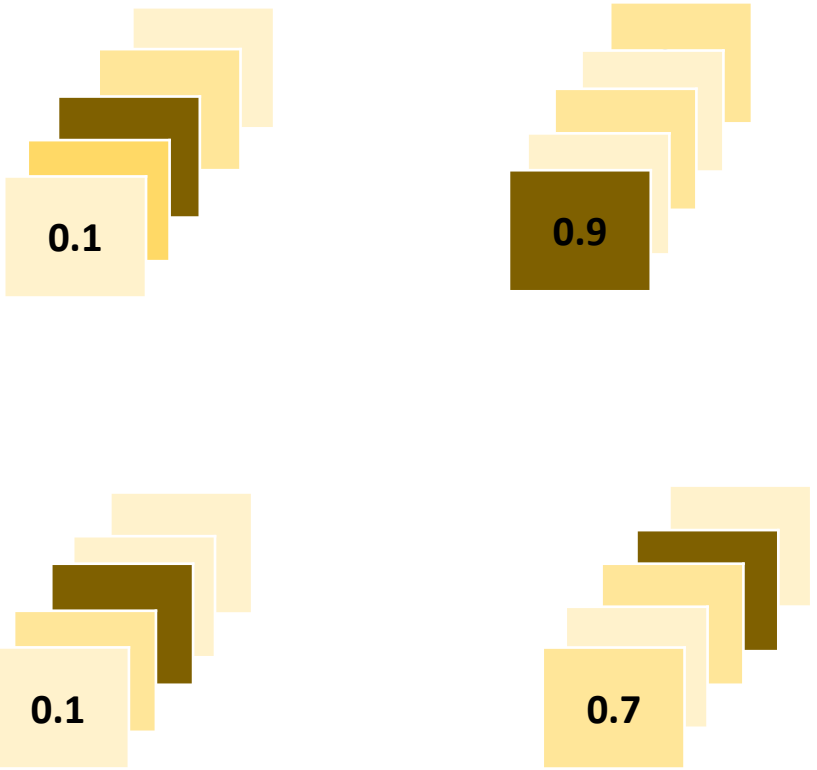
Recall the CNN

- Now this looks like each pixel in the window is passed through a simple linear layer of a feedforward NN



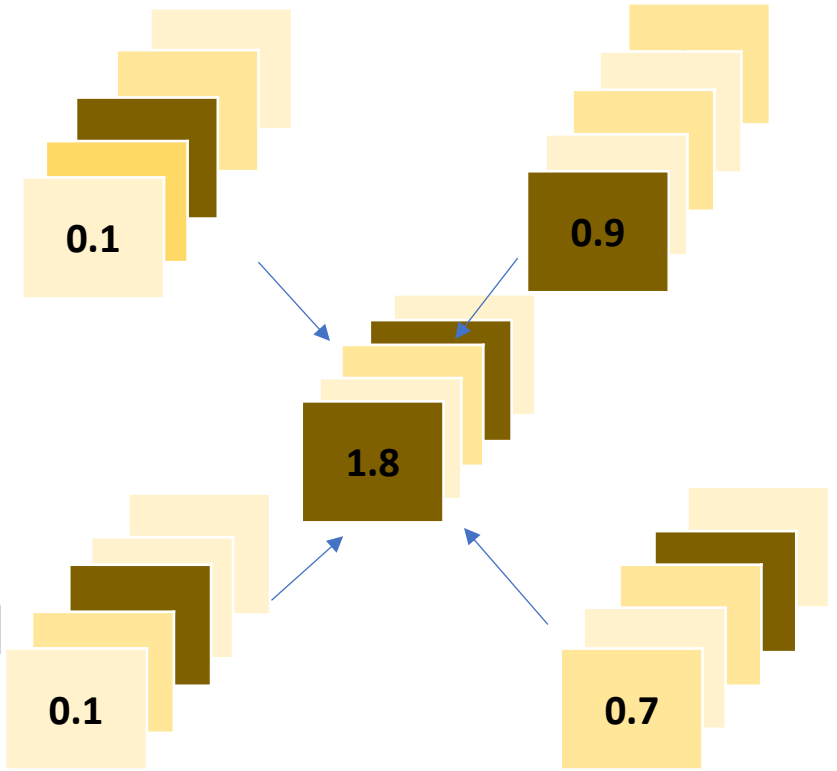
Recall the CNN

- Now this looks like each pixel in the window is passed through a simple linear layer of a feedforward NN
- Once we have passed each pixel through, we aggregate (in this case sum) the updated pixels



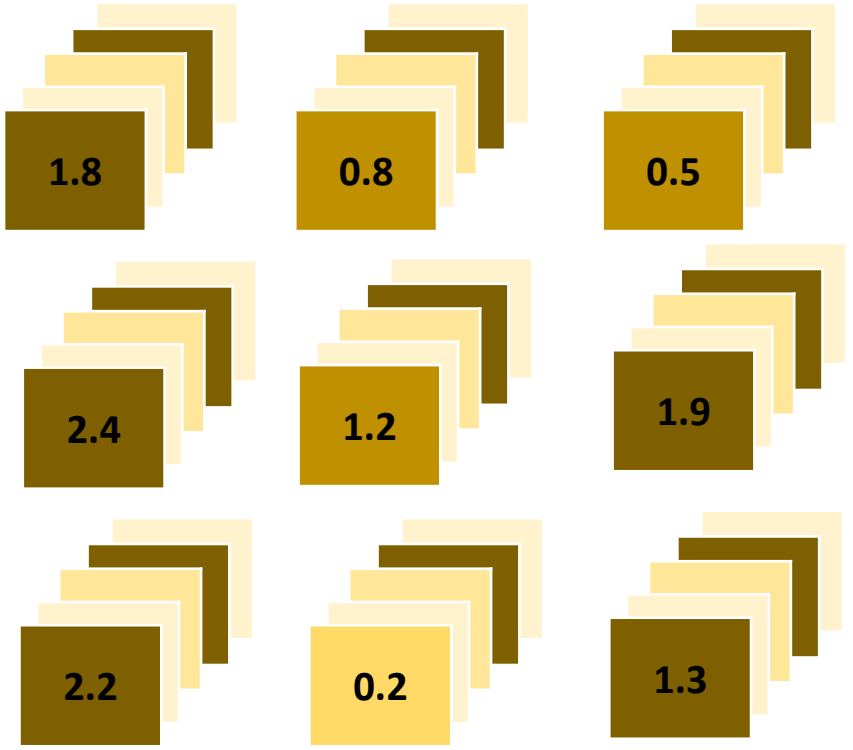
Recall the CNN

- Now this looks like each pixel in the window is passed through a simple linear layer of a feedforward NN
- Once we have passed each pixel through, we aggregate (in this case sum) the updated pixels
- The “center” of our 2x2 neighbourhood is updated with the convolution outputs



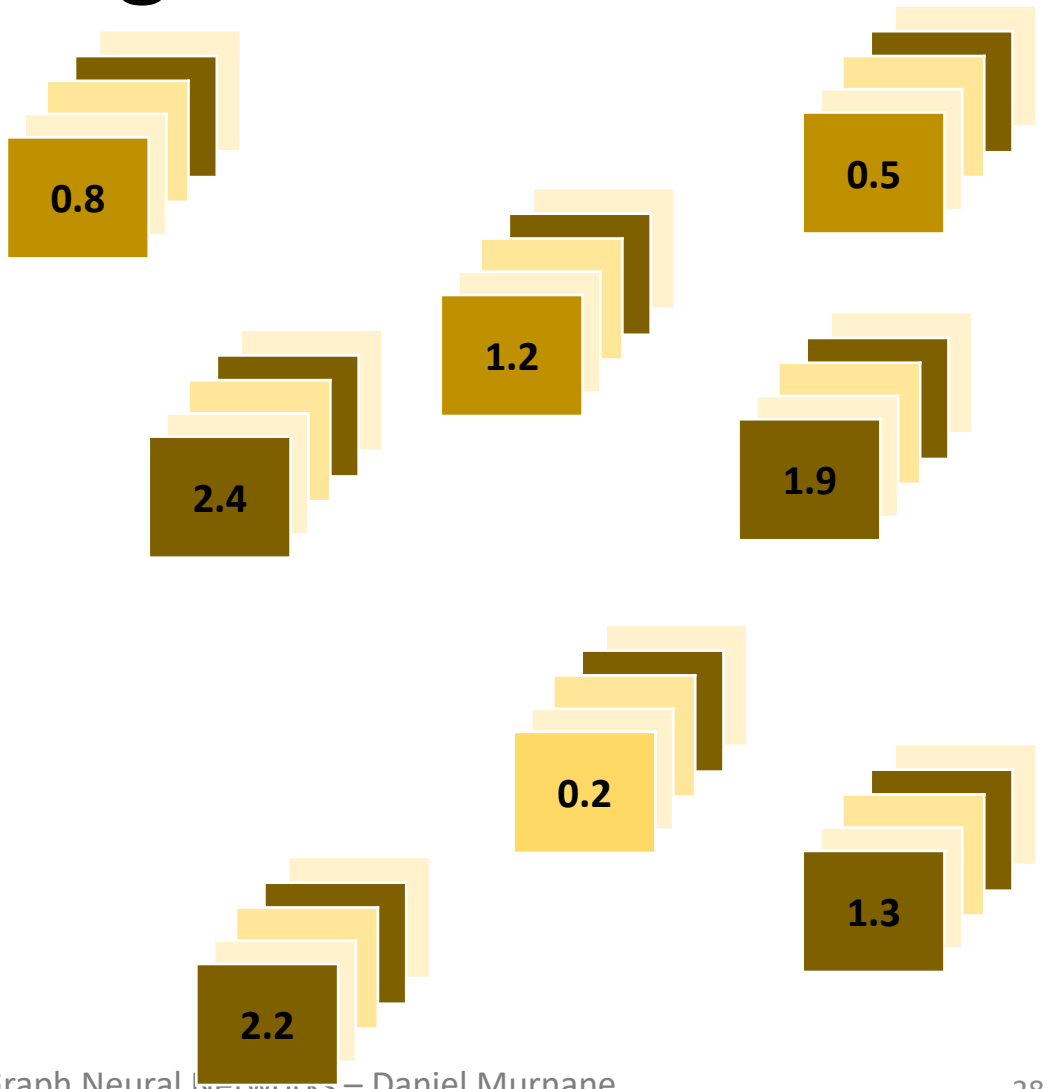
Recall the CNN

- The “center” of our 2x2 neighbourhood is updated with the convolution outputs
- These centers form the inputs for the *next* convolution, which again happens in a $(k-1) \times (k-1)$ grid neighbourhood (stride=1, no padding)



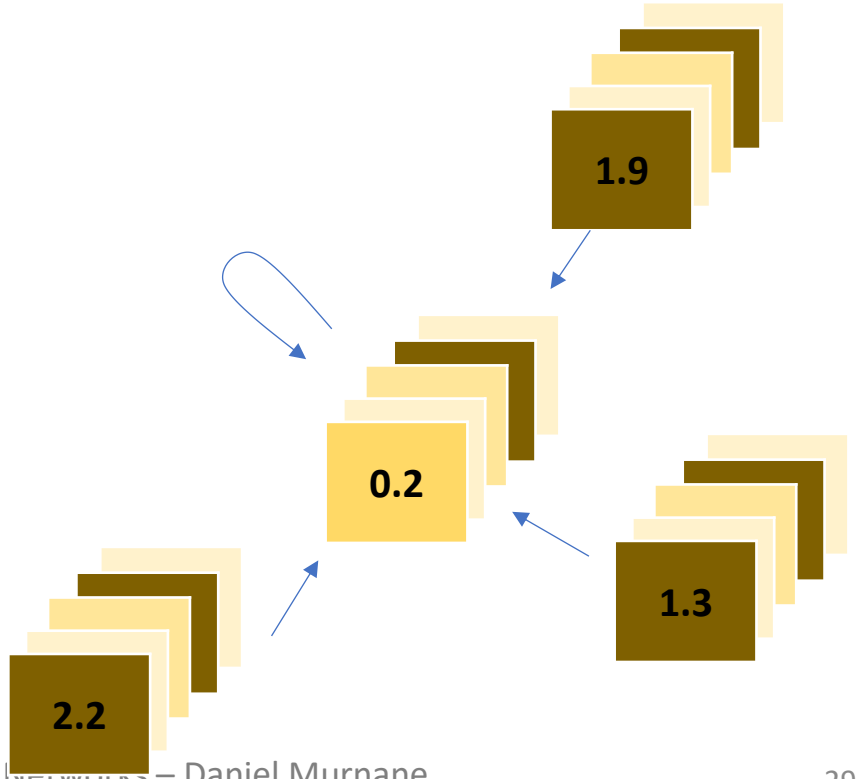
Now, let's throw away the grid structure

- Grids are convenient for images, but the world is not a grid
- Instead of "pixels", let's call each hidden vector now a "node"



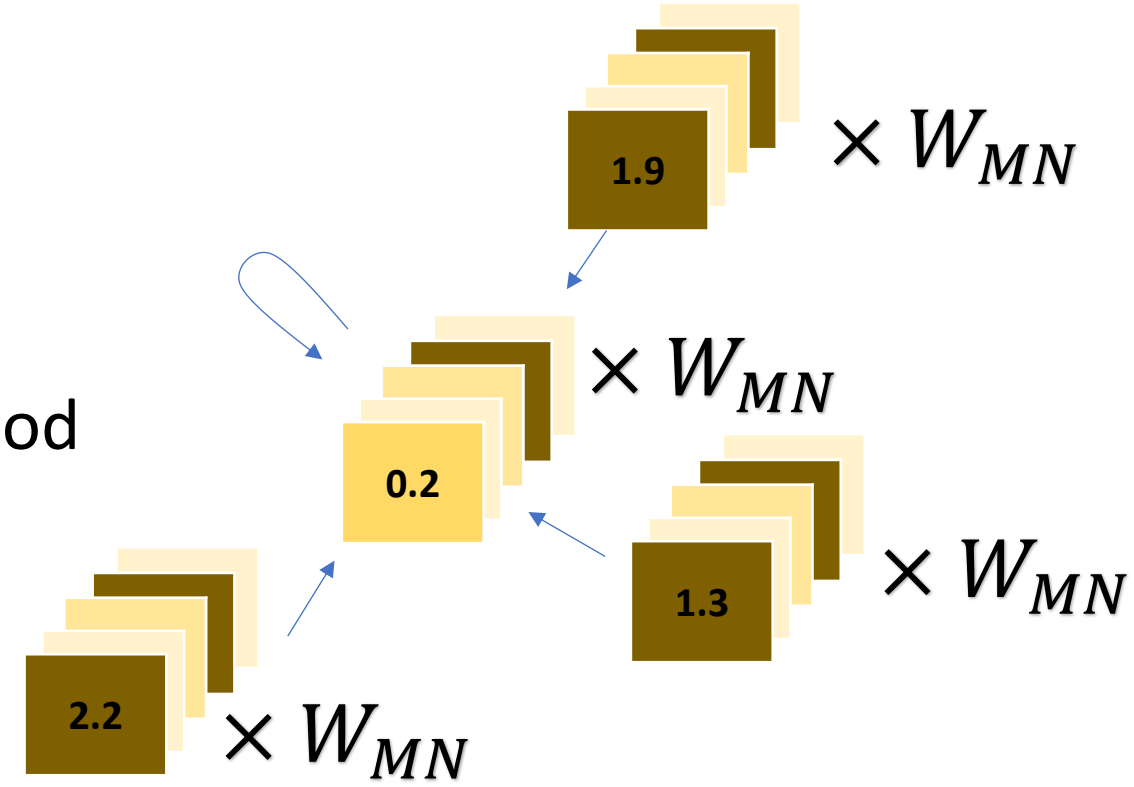
Now, let's throw away the grid structure

- Grids are convenient for images, but the world is not a grid
- Instead of "pixels", let's call each hidden vector now a "node"
- Let's also make each node the center of its own neighborhood
- And let's do our convolution over nearby nodes
- E.g. this would be one such neighborhood:



One final tweak!

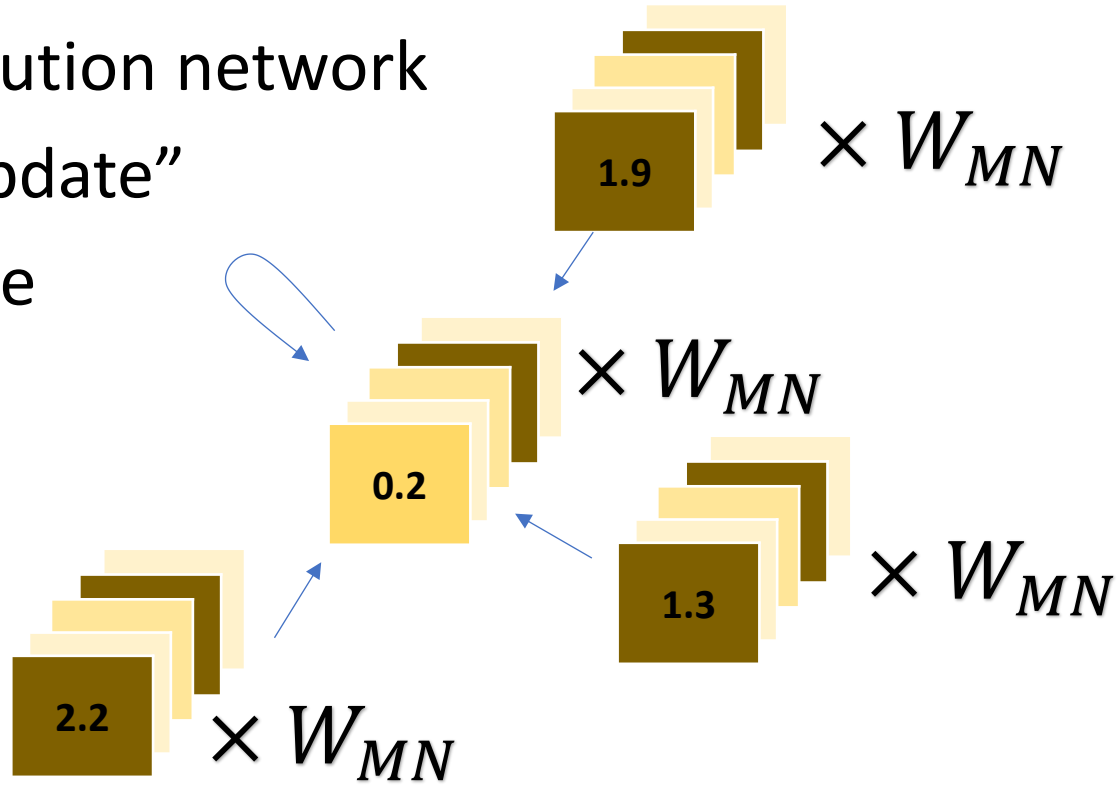
- Let's also make each node the center of its own neighborhood
- And let's do our convolution over nearby nodes
- E.g. this would be one such neighborhood
- In the CNN, each pixel had its own dedicated W. This was easy, because every neighborhood had a fixed number of pixels



- Now there are arbitrary number of nodes in each neighborhood, let's be even simpler and use the same W_{MN} for every node multiplication

This is a GNN!

- It's a simple one – a graph convolution network
- The $h_M \times W_{MN}$ step is a "node update"
- Passing these node features to the center node is called "message passing"
- Summing all the features is called "aggregation"
- Node update, message passing and aggregation are the building blocks for basically all graph neural networks

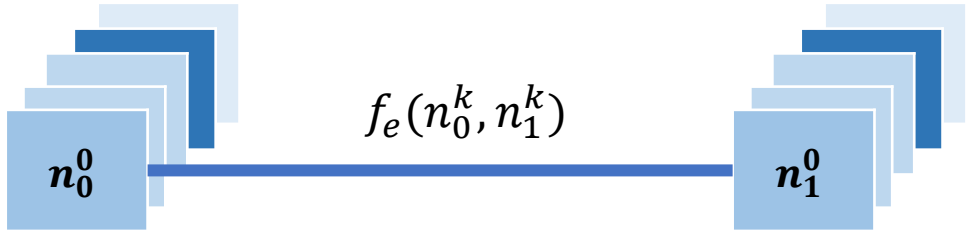


GNN Architectures

UNIVERSITY OF COPENHAGEN

Message passing

- We can create a language for convolutions that will work for almost **every other** ML model
- It's called "message passing", and it has two parts: **calculate the message** between pairs of objects, then **aggregate** the messages coming into each object's neighborhood
- For a graph, a message is the features that are passed along an edge – a learnable function that takes in the two nodes on either side of that edge

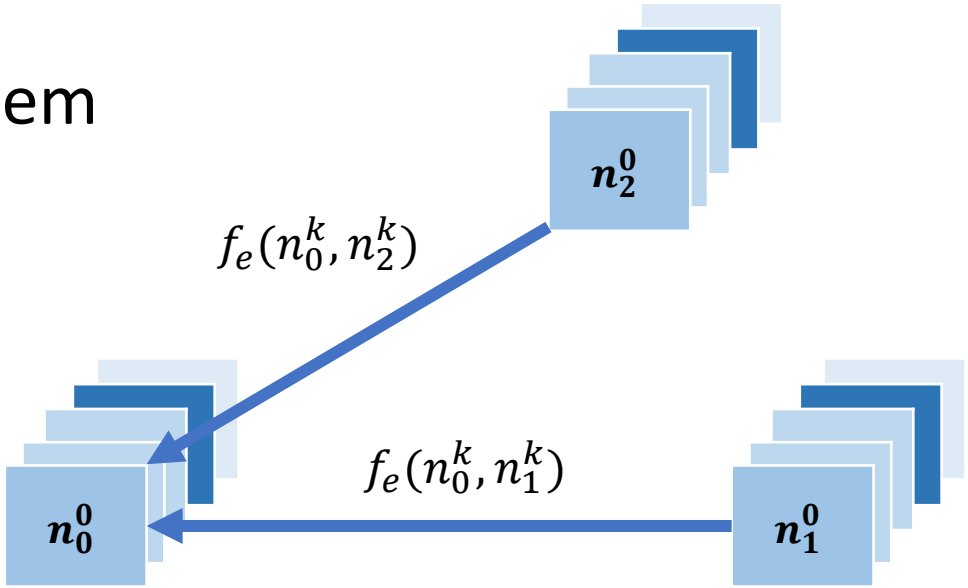


- f_e is a learnable function (feed-forward neural network)
- n_i^k are the N_k hidden channels of the N_i nodes in the graph

Aggregation & Permutation Invariance

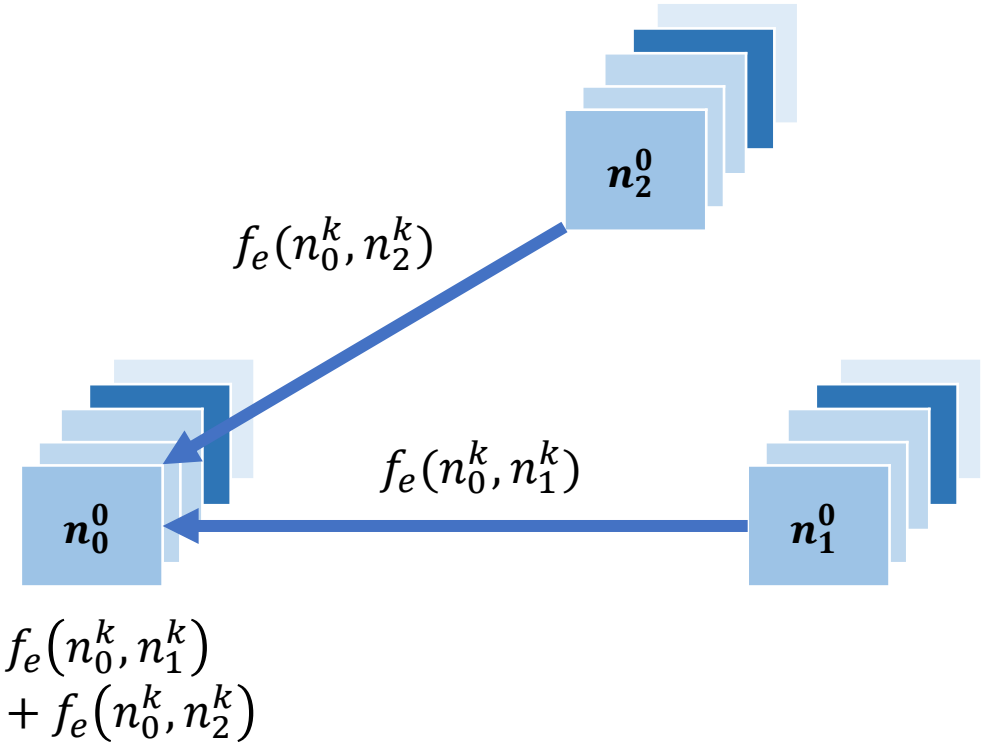
- We have a message function $m_{ij} = f_e(n_i^k, n_j^k)$, so how do we combine these messages around each node?
- We could stack them together and pass them through a FFNN? E.g. $f([m_{01}, m_{02}])$
- Two problems:

1. A FFNN has a fixed size, but we might have any number of incoming messages
2. If we switch the order that we stack the messages (which is arbitrary), the output of the FFNN will be different!



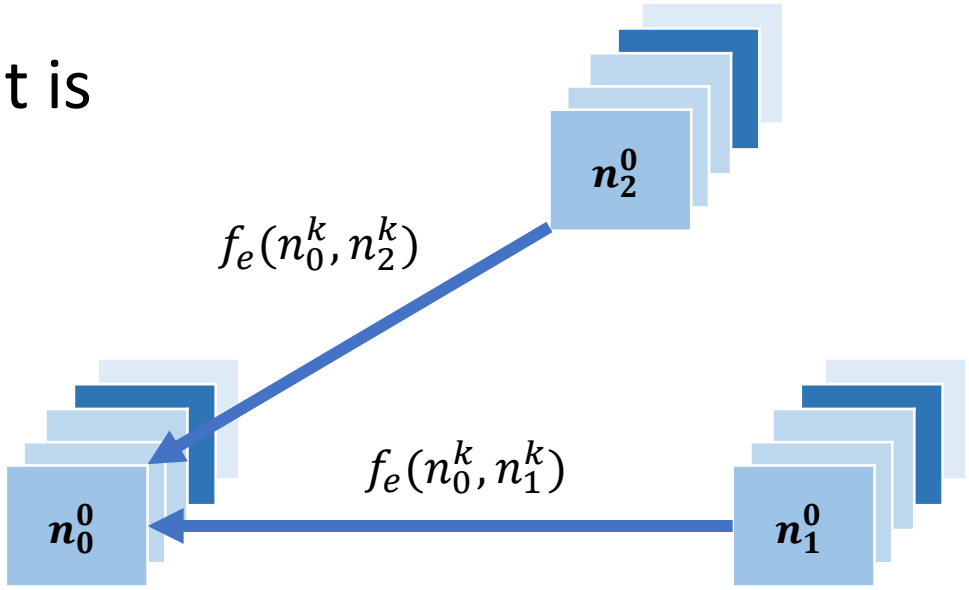
Aggregation & Permutation Invariance

- What we are looking for is a “permutation invariant” way to combine any number of incoming messages
- There are a few ways to do this, but the simplest is to take the sum (or mean/max/min)
- The choice of aggregation function is a hyperparameter (like the CNN pooling step)



Node update

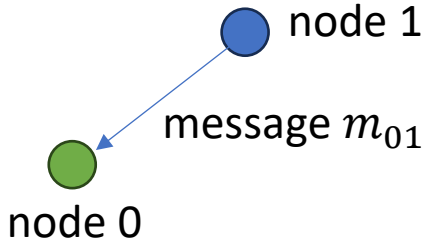
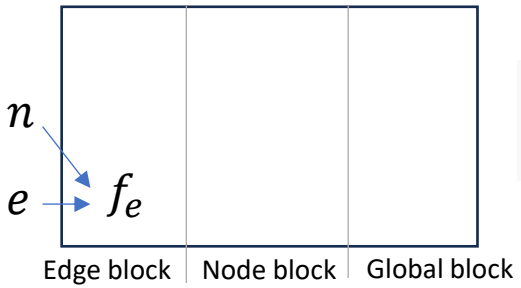
- The final step is to pass this aggregated information through a node FFNN
- The f_e is an edge-wise function/network, it is the same for every edge
- The f_n is a node function/network, it is the same for every node
- Note that even though the function is the same across all nodes or edges, the *outputs* of the function depend on the node or edge features



$$n_0'^k = f_n(f_e(n_0^k, n_1^k) + f_e(n_0^k, n_2^k))$$

Let's put it all together…

	Equations	Pictures	GNN Blocks	Code

1. Calculate messages	$m_{ij}^{l} = f_e(n_i^k, n_j^k)$, where there are edges between $i \rightarrow j$			`def message_passing(self, x, start, end):`
				`  edge_features = torch.cat([x[start], x[end]], dim=1)`
				`  e = self.edge_network(edge_features)`
				`  return e`

Let's put it all together…

Equations

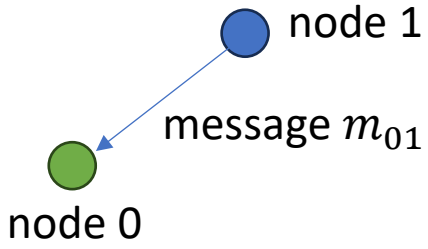
Pictures

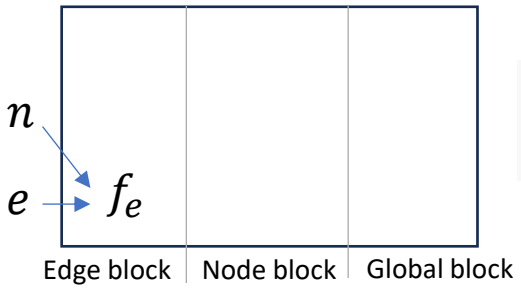
GNN Blocks

Code

1. Calculate messages

$m_{ij}^l = f_e(n_i^k, n_j^k)$,
where there are edges between $i \rightarrow j$



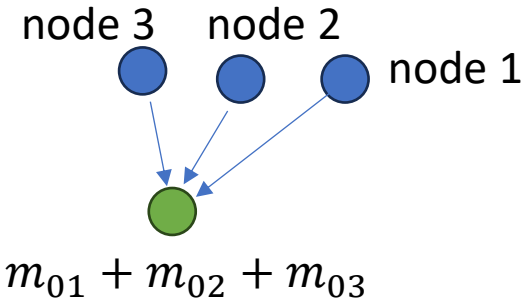


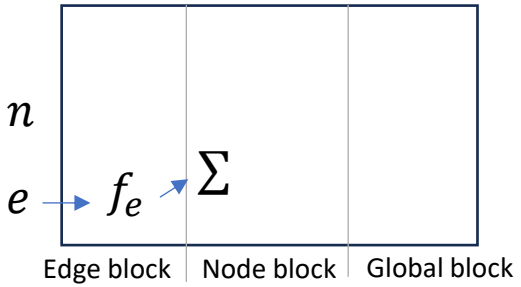

```
def message_passing(self, x, start, end):

    edge_features = torch.cat([x[start], x[end]], dim=1)
    e = self.edge_network(edge_features)
    return e
```

2. Aggregate messages

$$a_i^l = \sum_j (m_{ij}^l)$$

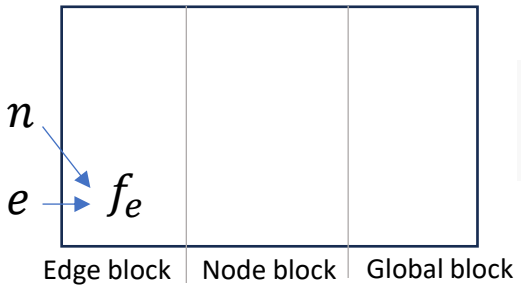


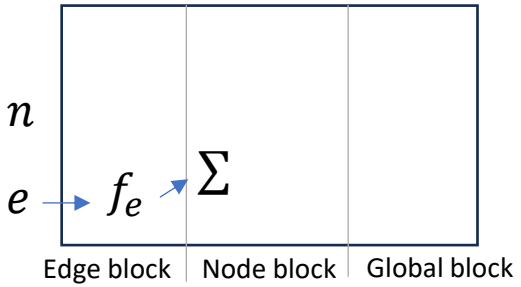
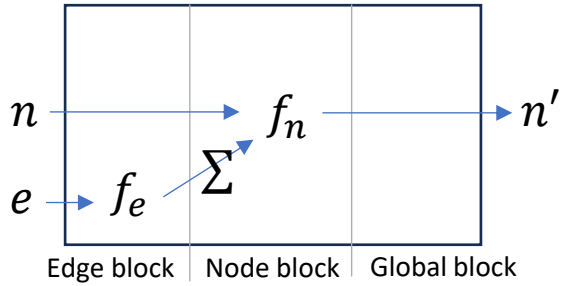



```
x = scatter_add(e, end)
```

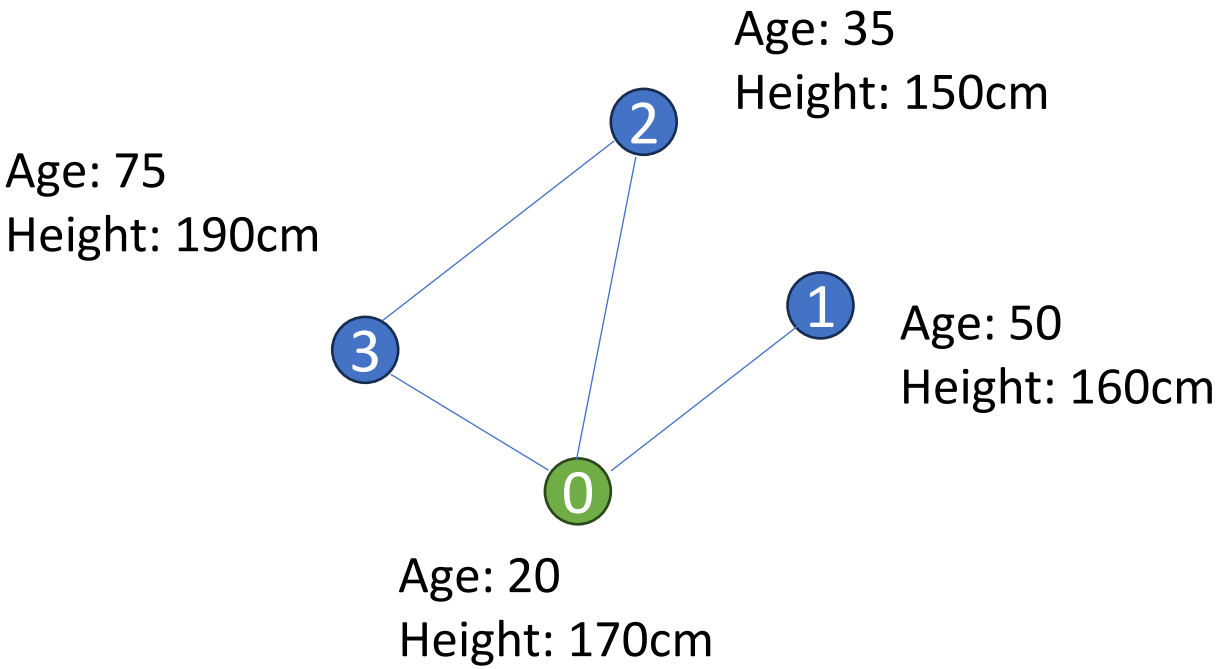
Let's put it all together…

	Equations	Pictures	GNN Blocks	Code

1. Calculate messages	$m_{ij}^l = f_e(n_i^k, n_j^k)$, where there are edges between $i \rightarrow j$	node 1, message m_{01}, node 0		`def message_passing(self, x, start, end):`
				`  edge_features = torch.cat([x[start], x[end]], dim=1)`
				`  e = self.edge_network(edge_features)`
				`  return e`

2. Aggregate messages	$a_i^l = \sum_j (m_{ij}^l)$	node 3, node 2, node 1, $m_{01} + m_{02} + m_{03}$		`x = scatter_add(e, end)`
3. Update nodes	$n'^k_i = f_n(a_i^l, n_i^k)$	Updated node 0		`x = self.node_network(x)`

Let's put it all together… with an example!



$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

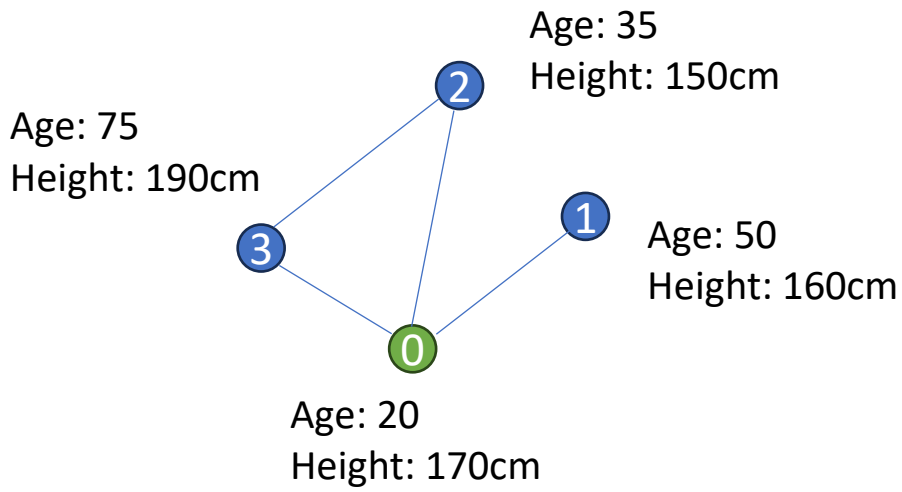
Let's put it all together… with an example!

Let's assume our "learned" message function is just:

$$f_e(n_i^k, n_j^k) = n_i^k - n_j^k$$

and our "learned" node update function is just:

$$f_n(a_i^k, n_i^k) = \frac{1}{2}(a_i^k + n_i^k)$$



$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

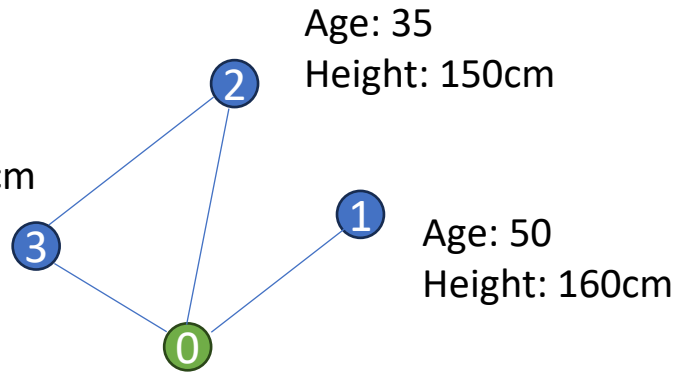
Let's put it all together… with an example!

Let's assume our "learned" message function is just:

$$f_e(n_i^k, n_j^k) = n_i^k - n_j^k$$

and our "learned" node update function is just:

$$f_n(a_i^k, n_i^k) = \frac{1}{2}(a_i^k + n_i^k)$$



$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

This is a toy, just like the hand-made filter from last lecture. When you train your GNNs, f_e will be a neural network like $\sigma(W_{kl}(n_i^k - n_j^k))$

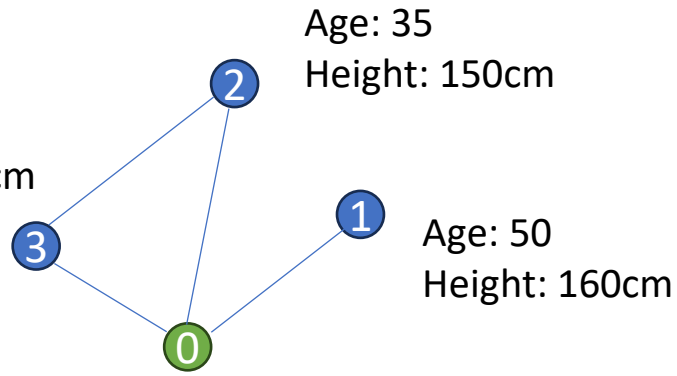
Let's put it all together… with an example!

Let's assume our "learned" message function is just:

$$f_e(n_i^k, n_j^k) = n_i^k - n_j^k$$

and our "learned" node update function is just:

$$f_n(a_i^k, n_i^k) = \frac{1}{2}(a_i^k + n_i^k)$$



$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

1. Calculate messages

$$m_{ij}^l = f_e(n_i^k, n_j^k)$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$m_{01}^l = [20 - 50, 170 - 160] = [-30, 10]$$

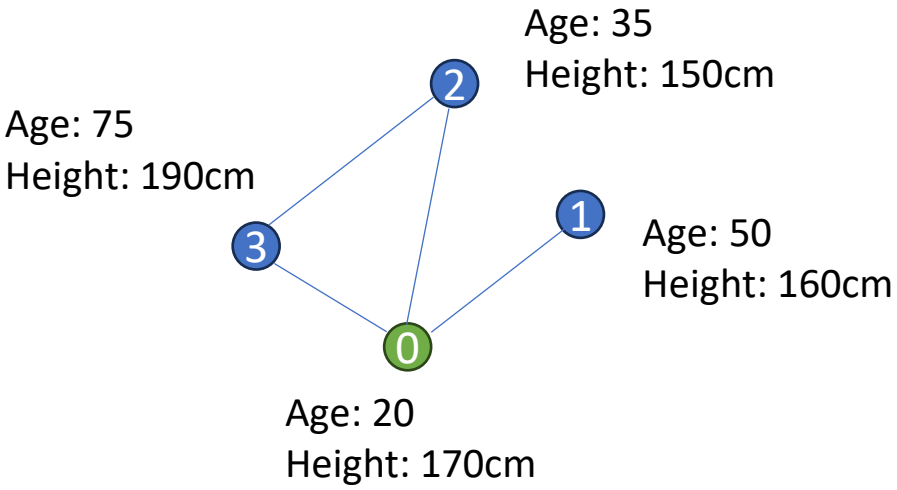
Let's put it all together… with an example!

Let's assume our "learned" message function is just:

$$f_e(n_i^k, n_j^k) = n_i^k - n_j^k$$

and our "learned" node update function is just:

$$f_n(a_i^k, n_i^k) = \frac{1}{2}(a_i^k + n_i^k)$$



$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

1. Calculate messages

$$m_{ij}^l = f_e(n_i^k, n_j^k)$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$m_{ij}^l = [\begin{bmatrix} -30 \\ 10 \end{bmatrix}, \begin{bmatrix} -15 \\ 20 \end{bmatrix}, \begin{bmatrix} -55 \\ -20 \end{bmatrix}, \begin{bmatrix} -40 \\ -40 \end{bmatrix}, \begin{bmatrix} 30 \\ -10 \end{bmatrix}, \begin{bmatrix} 15 \\ 20 \end{bmatrix}, \begin{bmatrix} 55 \\ 20 \end{bmatrix}, \begin{bmatrix} 40 \\ 40 \end{bmatrix}]$$

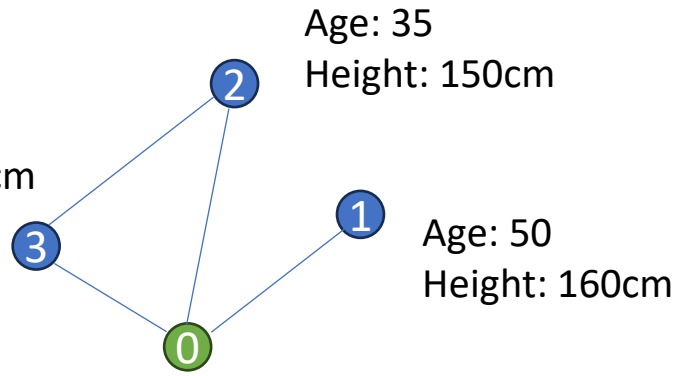
Let's put it all together… with an example!

Let's assume our "learned" message function is just:

$$f_e(n_i^k, n_j^k) = n_i^k - n_j^k$$

and our "learned" node update function is just:

$$f_n(a_i^k, n_i^k) = \frac{1}{2}(a_i^k + n_i^k)$$



$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

1. Calculate messages

$$m_{ij}^l = f_e(n_i^k, n_j^k)$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$m_{ij}^l = \left[\begin{bmatrix} -30 \\ 10 \end{bmatrix}, \begin{bmatrix} -15 \\ 20 \end{bmatrix}, \begin{bmatrix} -55 \\ -20 \end{bmatrix}, \begin{bmatrix} -40 \\ -40 \end{bmatrix}, \begin{bmatrix} 30 \\ -10 \end{bmatrix}, \begin{bmatrix} 15 \\ 20 \end{bmatrix}, \begin{bmatrix} 55 \\ 20 \end{bmatrix}, \begin{bmatrix} 40 \\ 40 \end{bmatrix}\right]$$

2. Aggregate messages

$$a_i^l = \sum_j (m_{ij}^l)$$

$$\begin{aligned} a_0^l &= m_{01}^l + m_{02}^l + m_{03}^l \\ &= \begin{bmatrix} -30 \\ 10 \end{bmatrix} + \begin{bmatrix} -15 \\ 20 \end{bmatrix} + \begin{bmatrix} -55 \\ -20 \end{bmatrix} \\ &= \begin{bmatrix} -100 \\ 10 \end{bmatrix} \end{aligned}$$

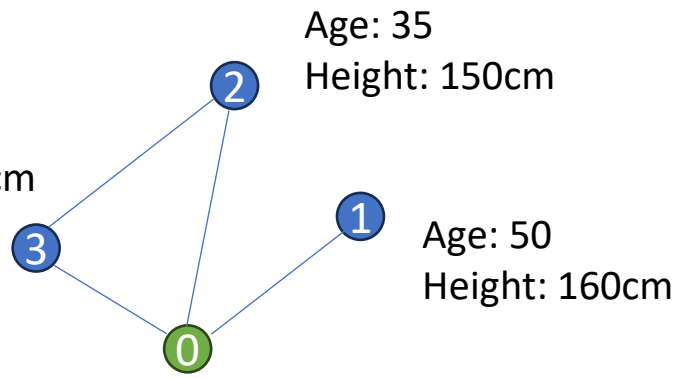
Let's put it all together… with an example!

Let's assume our "learned" message function is just:

$$f_e(n_i^k, n_j^k) = n_i^k - n_j^k$$

and our "learned" node update function is just:

$$f_n(a_i^k, n_i^k) = \frac{1}{2}(a_i^k + n_i^k)$$



$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

1. Calculate messages

$$m_{ij}^l = f_e(n_i^k, n_j^k)$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$m_{ij}^l = \left[\begin{bmatrix} -30 \\ 10 \end{bmatrix}, \begin{bmatrix} -15 \\ 20 \end{bmatrix}, \begin{bmatrix} -55 \\ -20 \end{bmatrix}, \begin{bmatrix} -40 \\ -40 \end{bmatrix}, \begin{bmatrix} 30 \\ -10 \end{bmatrix}, \begin{bmatrix} 15 \\ 20 \end{bmatrix}, \begin{bmatrix} 55 \\ 20 \end{bmatrix}, \begin{bmatrix} 40 \\ 40 \end{bmatrix}\right]$$

2. Aggregate messages

$$a_i^l = \sum_j (m_{ij}^l)$$

$$a_i^l = \left[\begin{bmatrix} -100 \\ 10 \end{bmatrix}, \begin{bmatrix} 30 \\ -10 \end{bmatrix}, \begin{bmatrix} -25 \\ -20 \end{bmatrix}, \begin{bmatrix} 95 \\ 60 \end{bmatrix}\right]$$

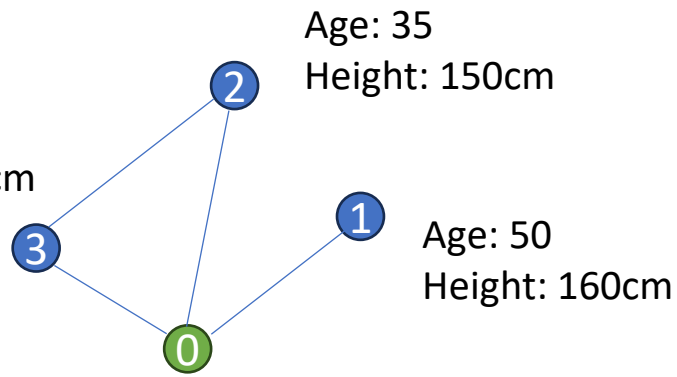
Let's put it all together… with an example!

Let's assume our "learned" message function is just:

$$f_e(n_i^k, n_j^k) = n_i^k - n_j^k$$

and our "learned" node update function is just:

$$f_n(a_i^k, n_i^k) = \frac{1}{2}(a_i^k + n_i^k)$$



$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

1. Calculate messages

$$m_{ij}^l = f_e(n_i^k, n_j^k)$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$m_{ij}^l = [\begin{bmatrix} -30 \\ 10 \end{bmatrix}, \begin{bmatrix} -15 \\ 20 \end{bmatrix}, \begin{bmatrix} -55 \\ -20 \end{bmatrix}, \begin{bmatrix} -40 \\ -40 \end{bmatrix}, \begin{bmatrix} 30 \\ -10 \end{bmatrix}, \begin{bmatrix} 15 \\ 20 \end{bmatrix}, \begin{bmatrix} 55 \\ 20 \end{bmatrix}, \begin{bmatrix} 40 \\ 40 \end{bmatrix}]$$

2. Aggregate messages

$$a_i^l = \sum_j (m_{ij}^l)$$

$$a_i^l = [\begin{bmatrix} -100 \\ 10 \end{bmatrix}, \begin{bmatrix} 30 \\ -10 \end{bmatrix}, \begin{bmatrix} -25 \\ -20 \end{bmatrix}, \begin{bmatrix} 95 \\ 60 \end{bmatrix}]$$

3. Update nodes

$$n'^k_i = f_n(a_i^l, n_i^k)$$

$$n'^k_0 = f_n(a_0^l, n_0^k) = \frac{1}{2}(\begin{bmatrix} -100 \\ 10 \end{bmatrix} + \begin{bmatrix} 20 \\ 170 \end{bmatrix}) = \begin{bmatrix} -40 \\ 90 \end{bmatrix}$$

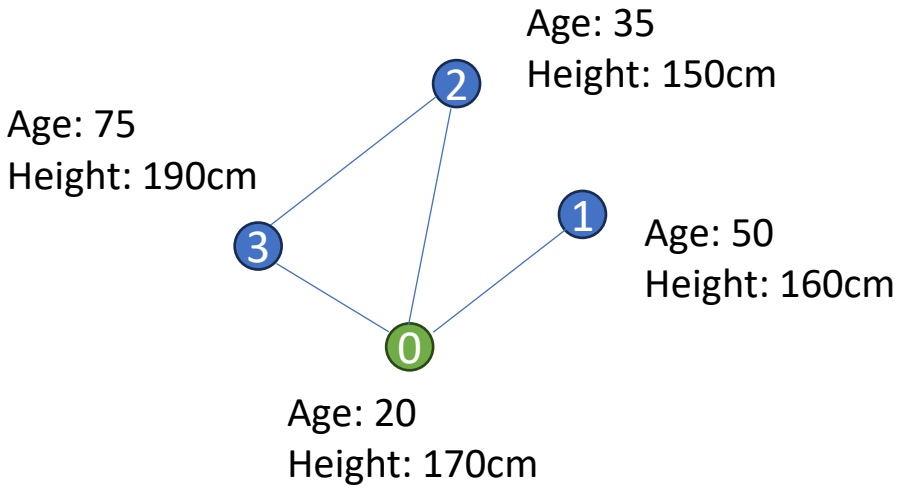
Let's put it all together… with an example!

Let's assume our "learned" message function is just:

$$f_e(n_i^k, n_j^k) = n_i^k - n_j^k$$

and our "learned" node update function is just:

$$f_n(a_i^k, n_i^k) = \frac{1}{2}(a_i^k + n_i^k)$$



$$n_i^k = \begin{bmatrix} 20 & 170 \\ 50 & 160 \\ 35 & 150 \\ 75 & 190 \end{bmatrix}$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

1. Calculate messages

$$m_{ij}^l = f_e(n_i^k, n_j^k)$$

$$E_{ij} = \begin{bmatrix} 0 & 0 & 0 & 2 & 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 3 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$m_{ij}^l = [\begin{bmatrix} -30 \\ 10 \end{bmatrix}, \begin{bmatrix} -15 \\ 20 \end{bmatrix}, \begin{bmatrix} -55 \\ -20 \end{bmatrix}, \begin{bmatrix} -40 \\ -40 \end{bmatrix}, \begin{bmatrix} 30 \\ -10 \end{bmatrix}, \begin{bmatrix} 15 \\ 20 \end{bmatrix}, \begin{bmatrix} 55 \\ 20 \end{bmatrix}, \begin{bmatrix} 40 \\ 40 \end{bmatrix}]$$

2. Aggregate messages

$$a_i^l = \sum_j (m_{ij}^l)$$

$$a_i^l = [\begin{bmatrix} -100 \\ 10 \end{bmatrix}, \begin{bmatrix} 30 \\ -10 \end{bmatrix}, \begin{bmatrix} -25 \\ -20 \end{bmatrix}, \begin{bmatrix} 95 \\ 60 \end{bmatrix}]$$

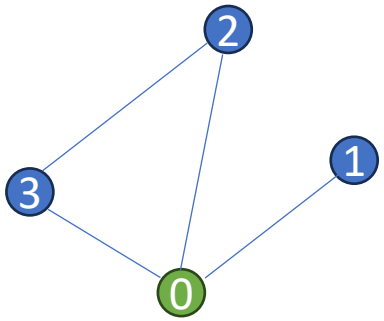
3. Update nodes

$$n'^k_i = f_n(a_i^l, n_i^k)$$

$$n'^k_i = \begin{bmatrix} -40 & 90 \\ 40 & 75 \\ 5 & 65 \\ 85 & 125 \end{bmatrix}$$

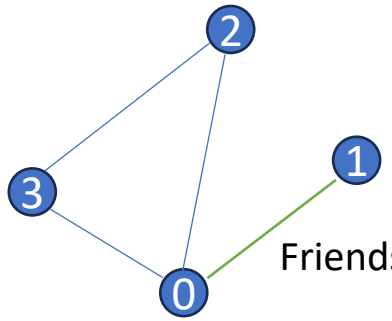
GNN tasks vs. architectures

- Just like with the CNN, the choice of GNN convolution is usually separate from the final training task
- Any GNN convolution that does message passing, and updates hidden node features, allows us to predict:



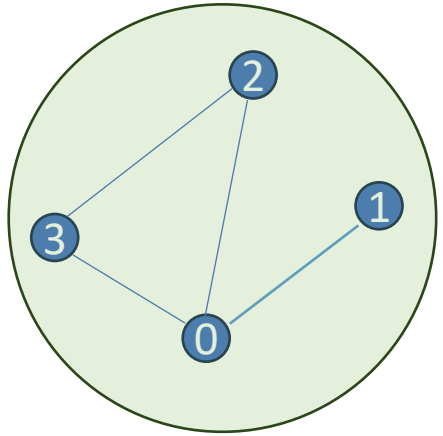
Age: ?
Height: ?

Node features



Friendship status?

Edge features



Class topic?

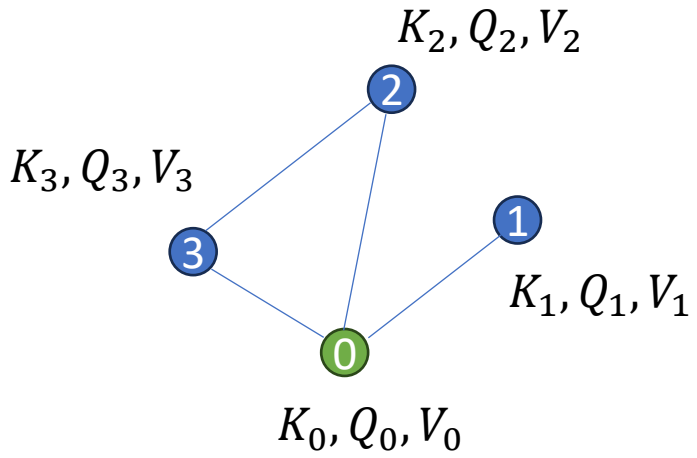
Global features

The Transformer as a GNN

UNIVERSITY OF COPENHAGEN

Let's build a ***very*** specific graph convolution…

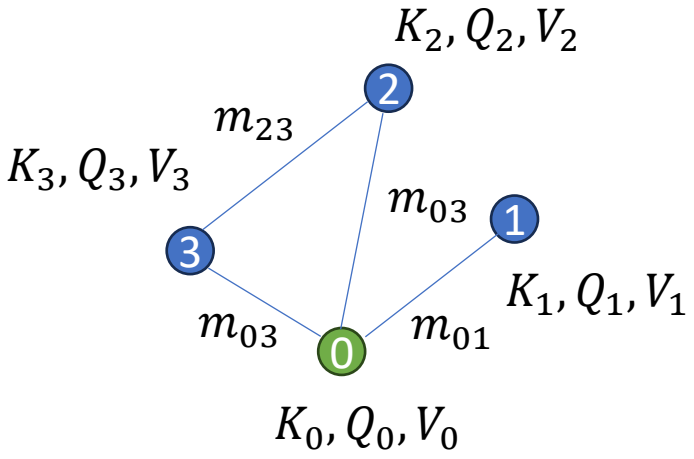
- For each node, attach *three* sets of hidden features: K, Q, V



Let's build a ***very*** specific graph convolution…

- For each node, attach *three* sets of hidden features: K, Q, V
- Define a message function on each edge:

$$m_{ij} = f_e\left(K_i^k, Q_j^k\right) = softmax\left(\sum_k K_i^k Q_j^k\right)$$



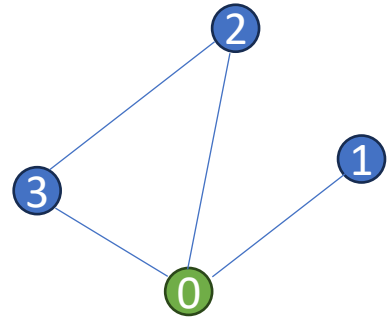
Let's build a ***very*** specific graph convolution…

- For each node, attach *three* sets of hidden features: K, Q, V
- Define a message function on each edge:

$$m_{ij} = f_e\left(K_i^k, Q_j^k\right) = softmax\left(\sum_k K_i^k Q_j^k\right)$$

- Define an aggregation function around each node:

$a_i^k = \sum_j m_{ij} V_j^k$, this is just a weighted sum

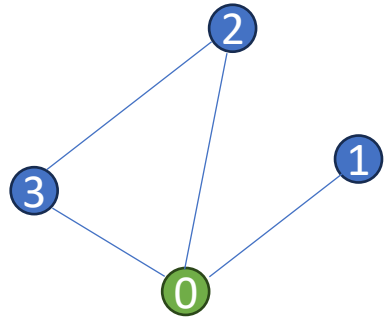


$V_0 m_{01} + V_0 m_{02} + V_0 m_{03}$

Let's build a ***very*** specific graph convolution…

- For each node, attach *three* sets of hidden features: K, Q, V
- Define a message function on each edge:

$$m_{ij} = f_e\left(K_i^k, Q_j^k\right) = softmax\left(\sum_k K_i^k Q_j^k\right)$$



$f_n^K(a_i), f_n^Q(a_i), f_n^V(a_i)$

- Define an aggregation function around each node:

$a_i^k = \sum_j m_{ij} V_j^k$, this is just a weighted sum

- The node update is just FFNNs to get the next K', Q', V'

Let's build a ***very*** specific graph convolution…

- For each node, attach *three* sets of hidden features: K, Q, V
- Define a message function on each edge:

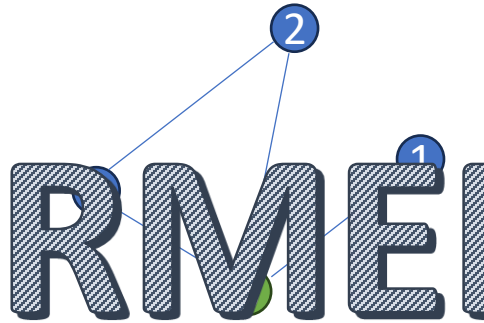
$$m_{ij} = f_e(K_i^k, Q_i^k) = softmax\left(\sum_k K_i^k Q_i^k\right)$$

THIS IS A TRANSFORMER

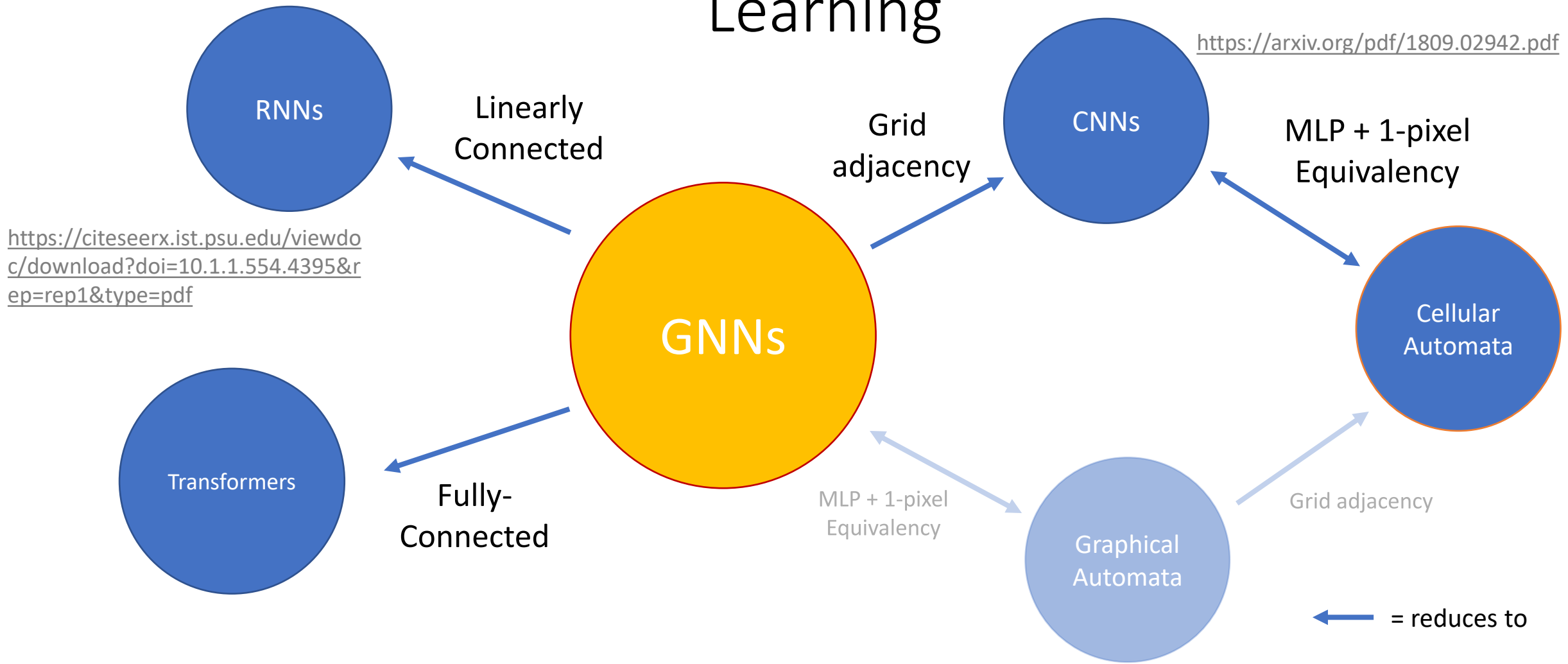
- Define an aggregation function around each node:

$a_i^k = \sum_j m_{ij} V_j^k$, this is just a weighted sum

- The node update is just FFNNs to get the next K', Q', V'



The Landscape of Geometric Deep Learning



https://github.com/murnanedaniel/GNN-as-Transformer-as-GNN/blob/main/0-Transformer_vs_GNN_Annotated.ipynb
https://arxiv.org/pdf/2012.09699.pdf

Case Studies

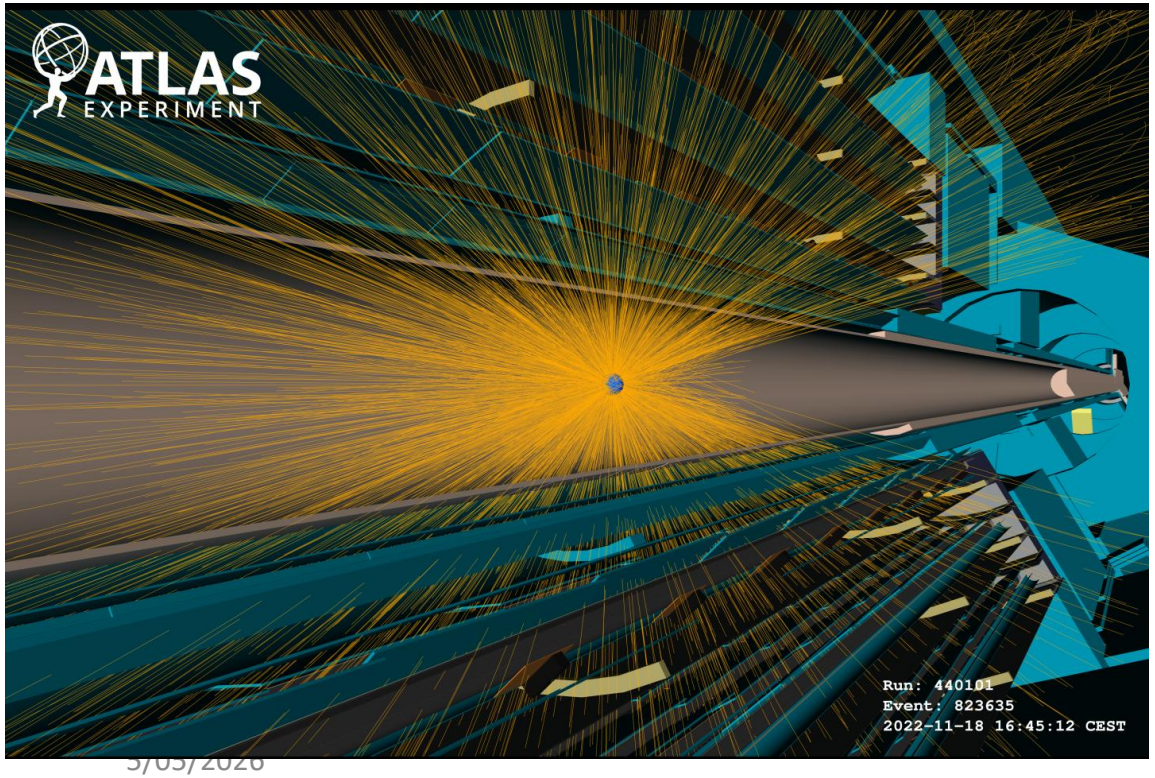
UNIVERSITY OF COPENHAGEN

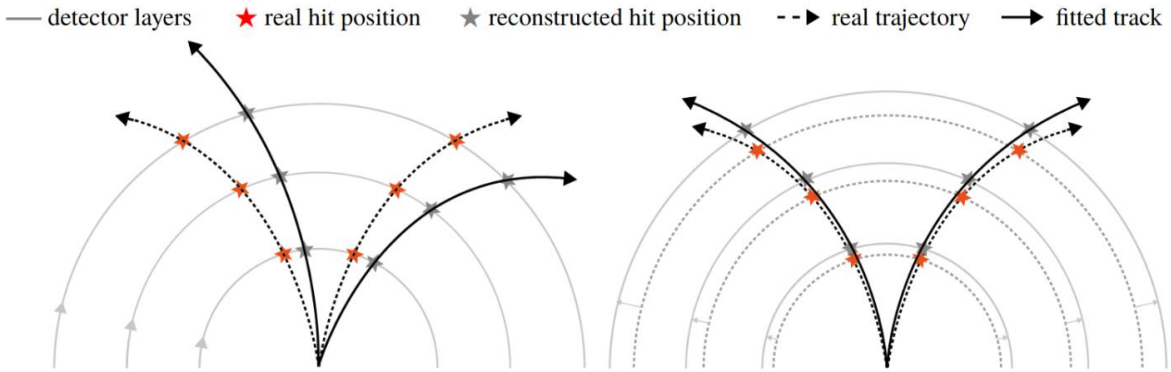
Google Maps

- DeepMind worked with Google Maps developers to apply a GNN to estimate travel time
- Improved estimates by up to 50% in large cities
- Maps are *really* hard to optimize on, as they exhibit combinatorial behavior
- GNNs can give fast, approximate solutions to map problems

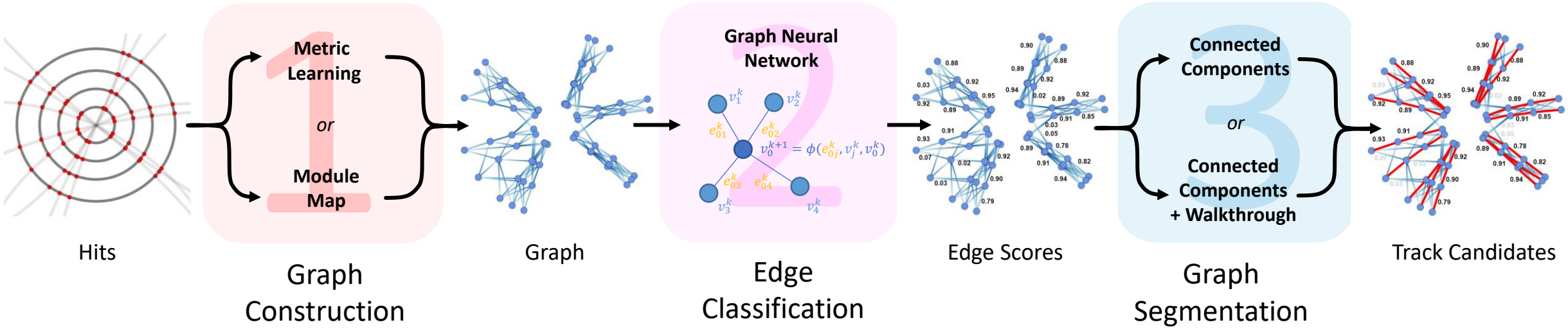
Edge Classification for Particle Tracking

- In the ATLAS experiment, we need to connect points in the detector to reconstruct particle tracks
- It's a massive connect-the-dots game, with 300,000 dots to connect, every 25 nanoseconds

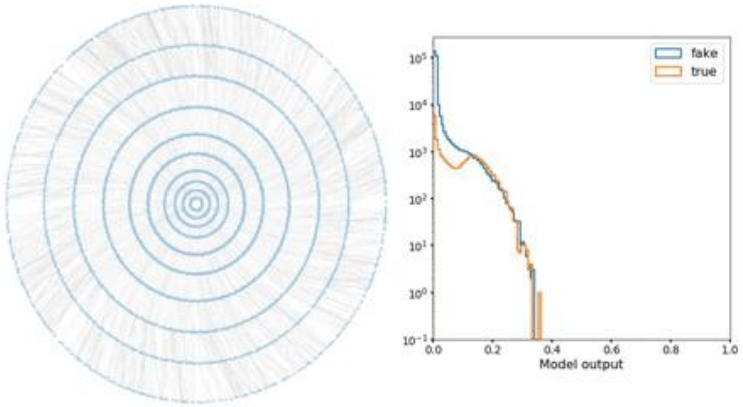




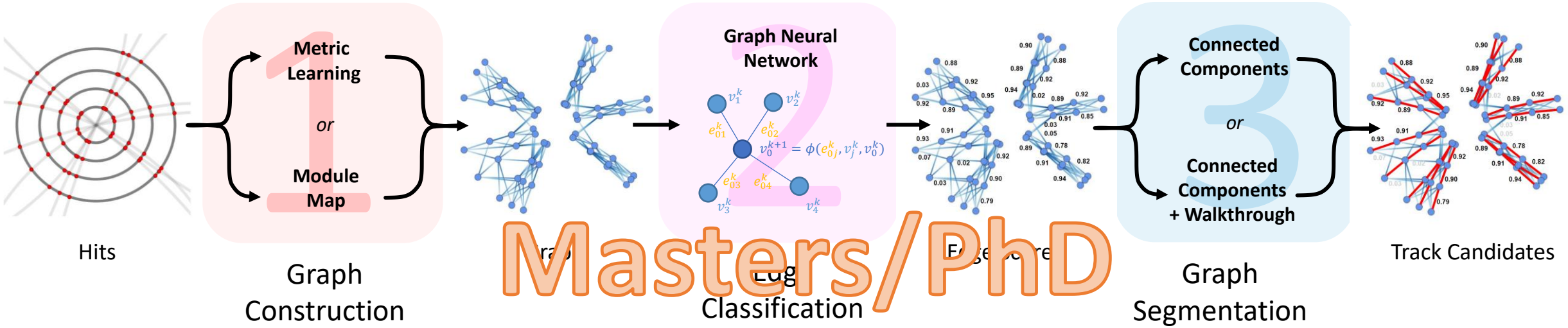
Edge Classification for Particle Tracking



- In the ATLAS experiment, we need to connect points in the detector to reconstruct particle tracks
- It's a massive connect-the-dots game, with 300,000 dots to connect, every 25 nanoseconds
- We are building graph neural networks to do this faster than traditional algorithms

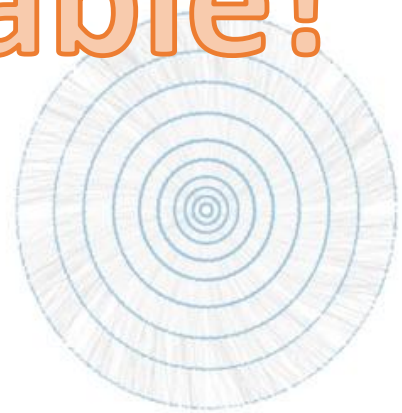


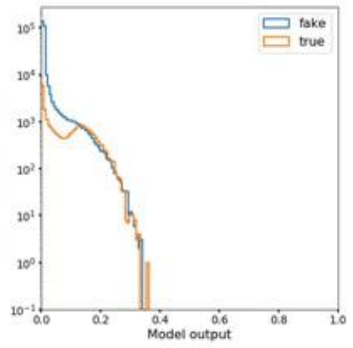
Edge Classification for Particle Tracking



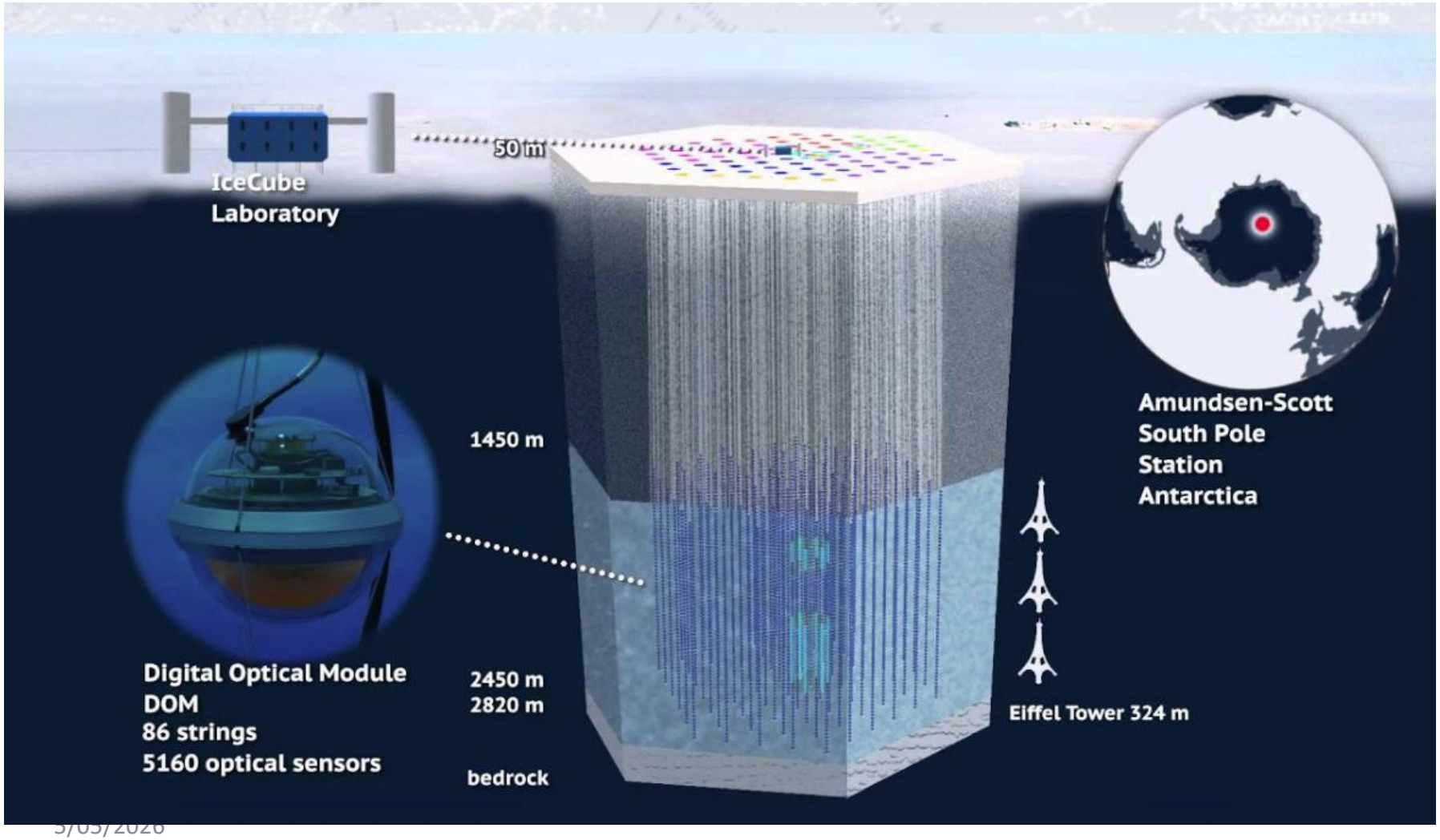
Masters/PhD projects available!

- In the ATLAS experiment, we need to connect points in the detector to reconstruct particle tracks
- It's a massive connect-the-dots game, with 300,000 dots to connect, every 25 nanoseconds
- We are building graph neural networks to do this faster than traditional algorithms



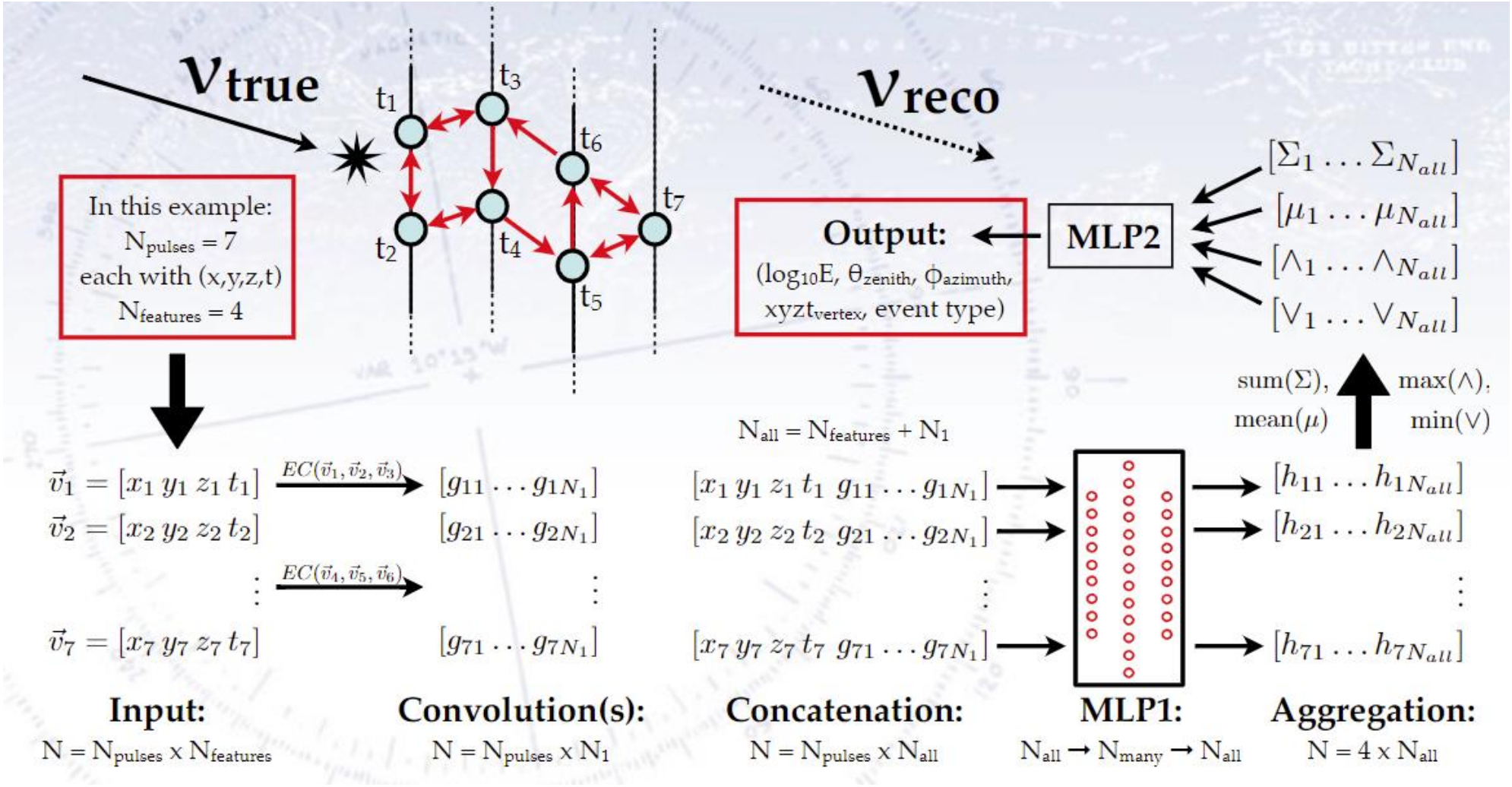


Ice Cube neutrino prediction

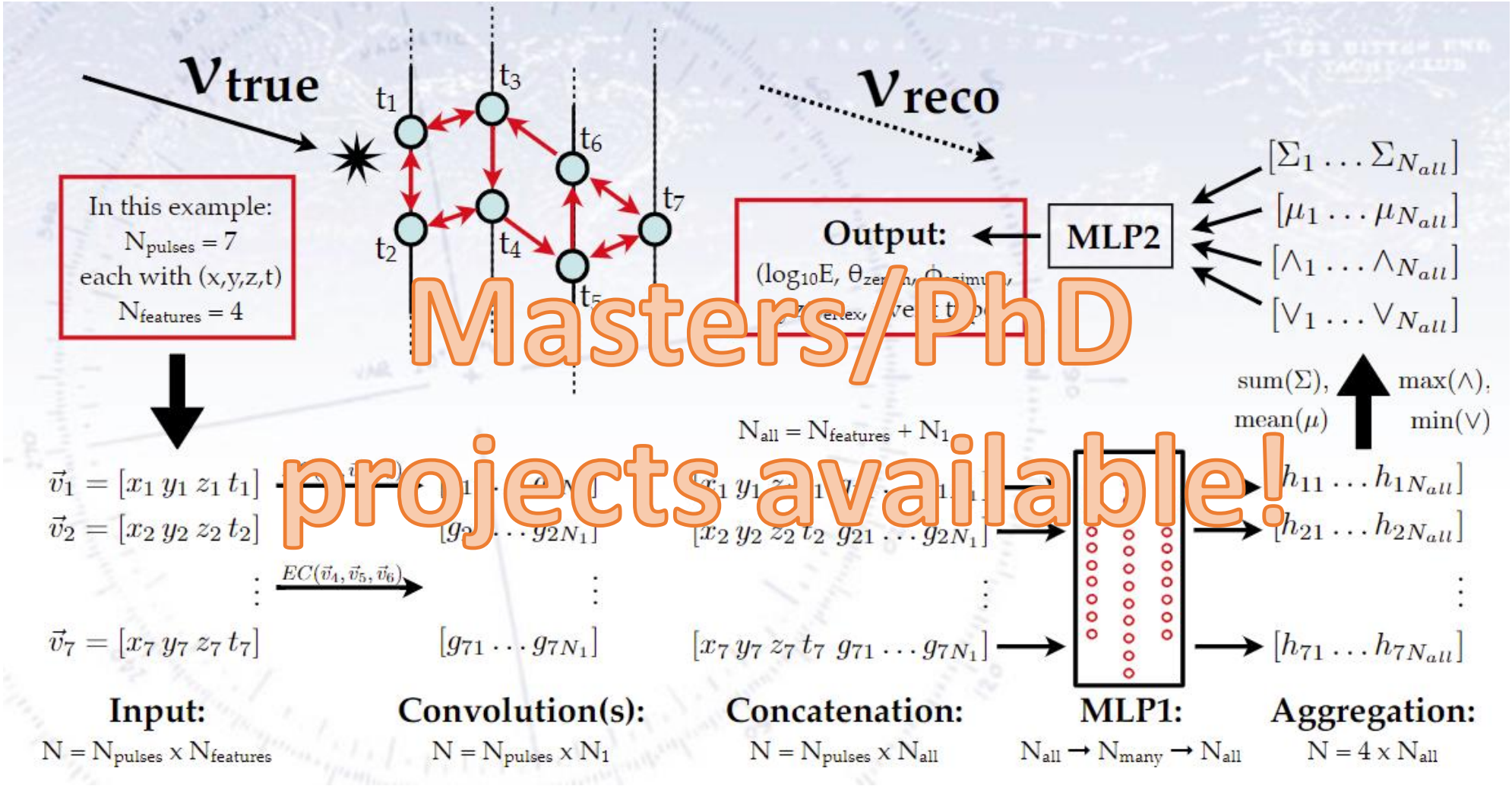


- Neutrino observatory inside the ice of Antarctica
- Goal is to detect neutrinos and measure precisely the direction they came from, their energy, what type of neutrino they are, etc.

Ice Cube neutrino prediction



Ice Cube neutrino prediction



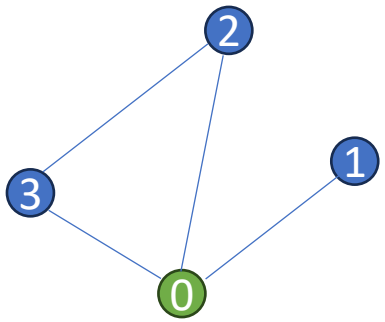
POSSIBILITIES IN GRAPH-LAND

The costs of elegance

- While technically true that many situations are naturally represented as graphs, the ML world **has not yet figured out how to handle them robustly**
- One issue: the “Topology Problem” [my paper arXiv:2307.16662] – just because you have some heuristic about the data structure, is this necessarily the right topology to use for message passing? The bitter lesson: Take a transformer, make it huge, use lots of data, and you will beat any GNN
- GNNs work great out of the box for fully supervised tasks
- CNNs and Transformers typically assume a maximum size of image or set. GNNs have some dream to be: agnostic to “cardinality” (graph size) and permutation in/equi-variant – these are very difficult requirements to satisfy for self-supervised tasks, e.g. generative models
- “Transformer People” think they are inventing things that always existed in GNNs (Set-LLM: arXiv:2505.15433), ”GNN People” refuse to just throw a transformer at a problem to see if it works. I suggest you try to be somewhere in the middle!

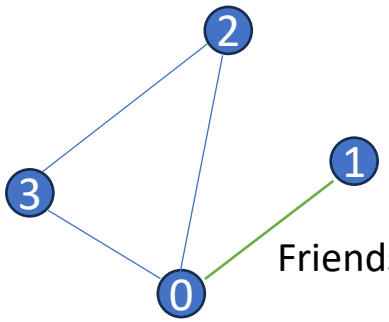
Fully Supervised Tasks

- Node, edge and graph classification are straightforward, and orthogonal to the choice of GNN architecture
- A class of students:



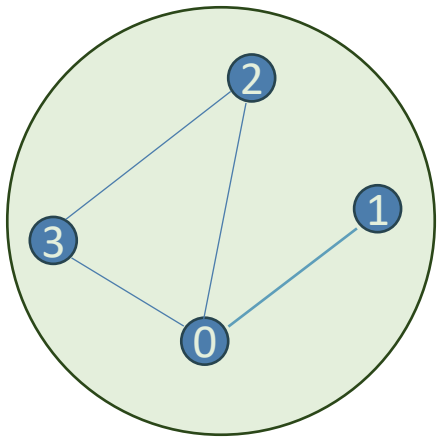
Age: ?
Height: ?

Node features



Friendship status?

Edge features



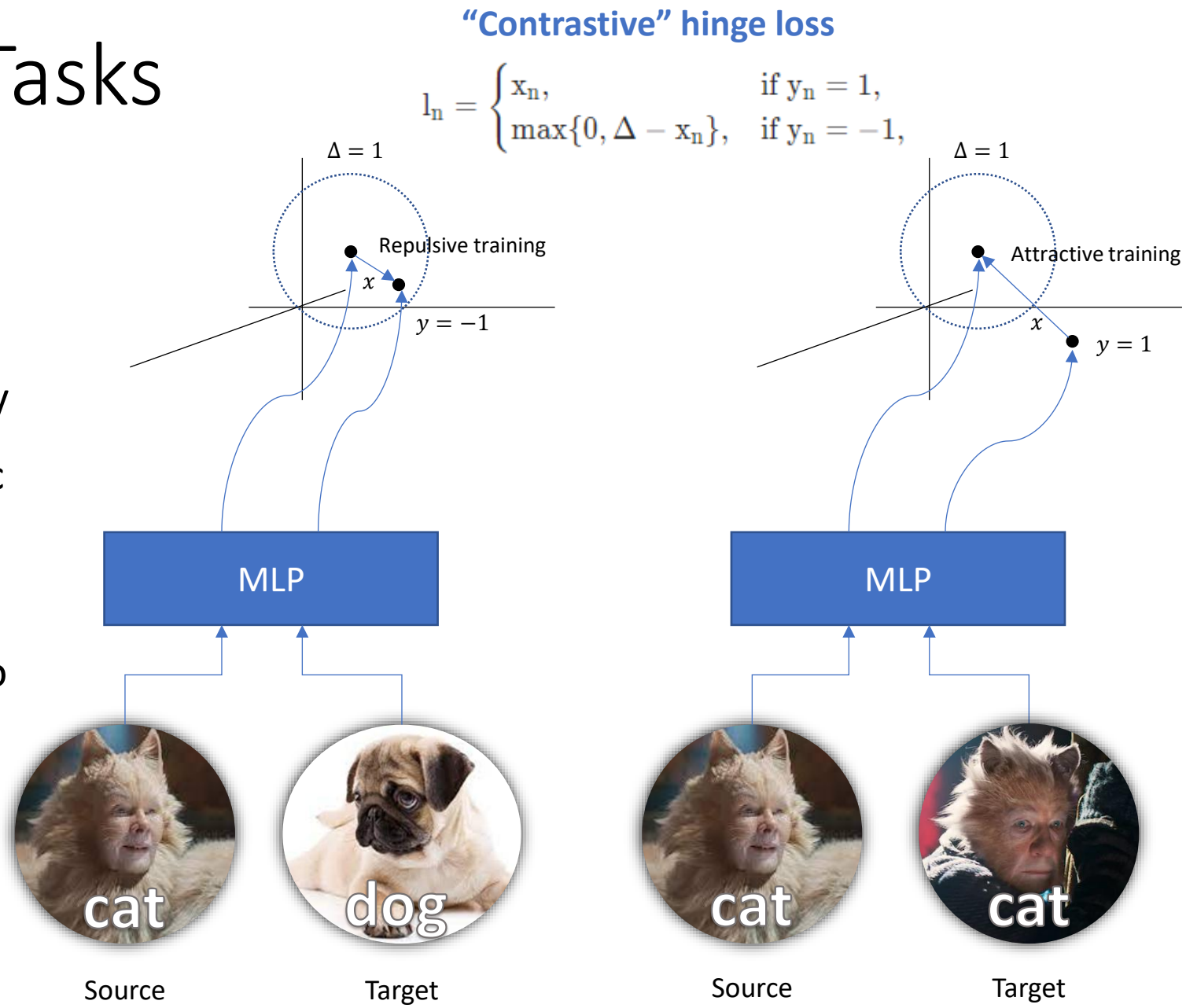
Class topic?

Global features

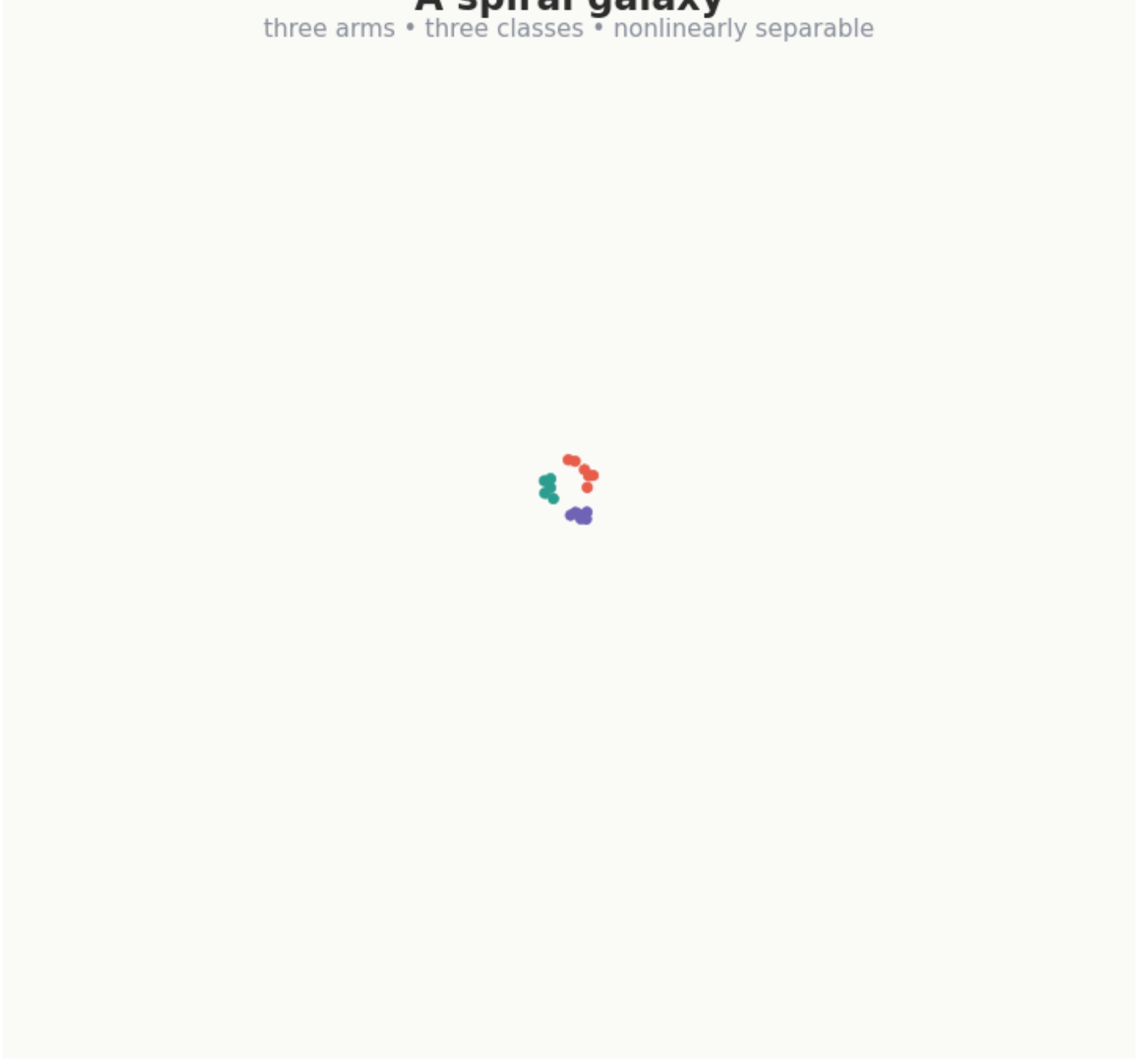
- Note: edge classification can be framed as edge prediction
  - Example: Track classification!

Fully Supervised Tasks

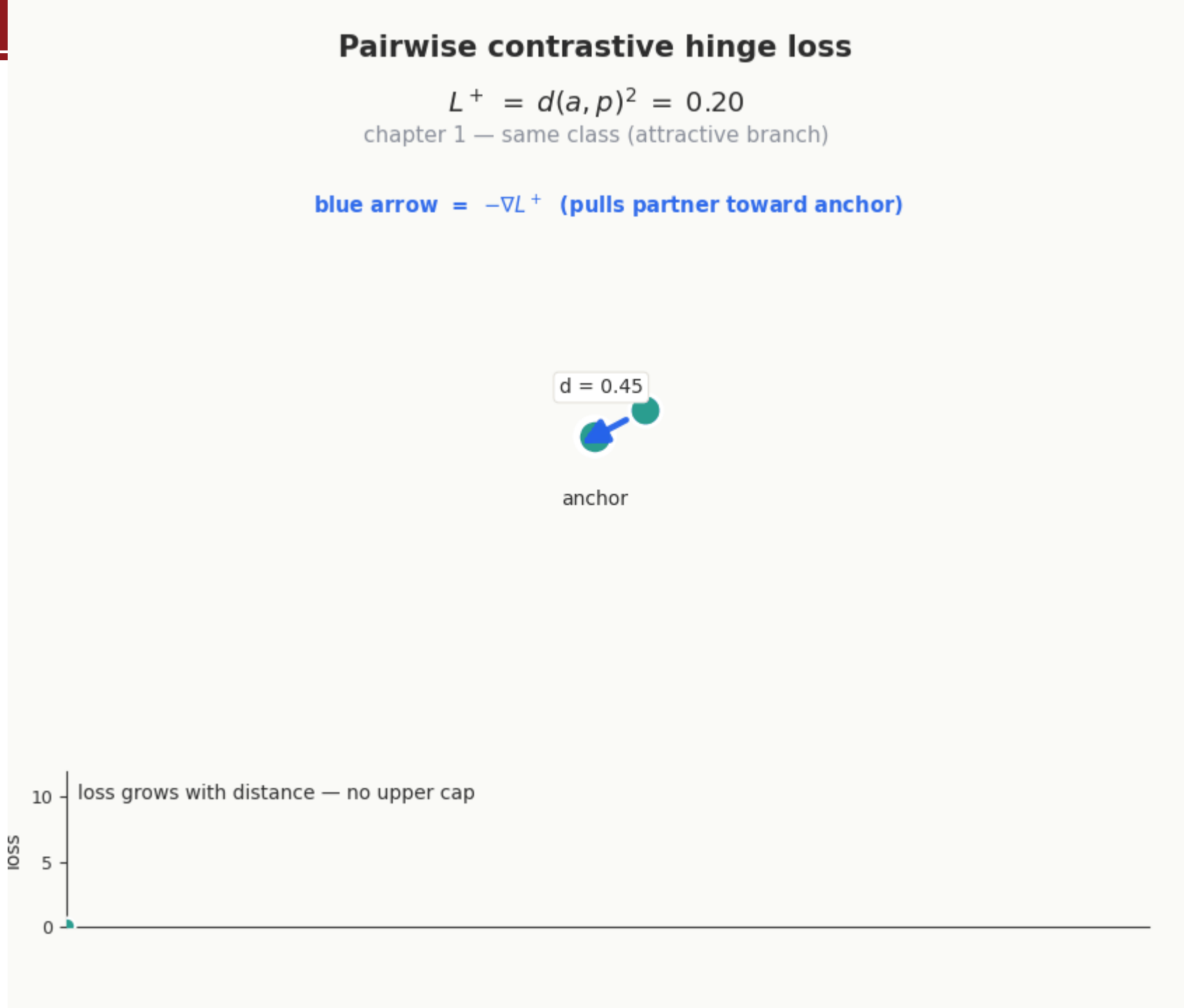
- There is also a class of geometric ML task that we will explore in the next seminar: metric learning
- Metric learning is the task of learning an “embedding space” that satisfies some criteria – clearly this is a very flexible paradigm and most ML somehow includes metric learning implicitly!
- For example, binary classify cats and docs, assigns each to a class
- We can remove this, instead just to pairwise distance with a “hinge loss”



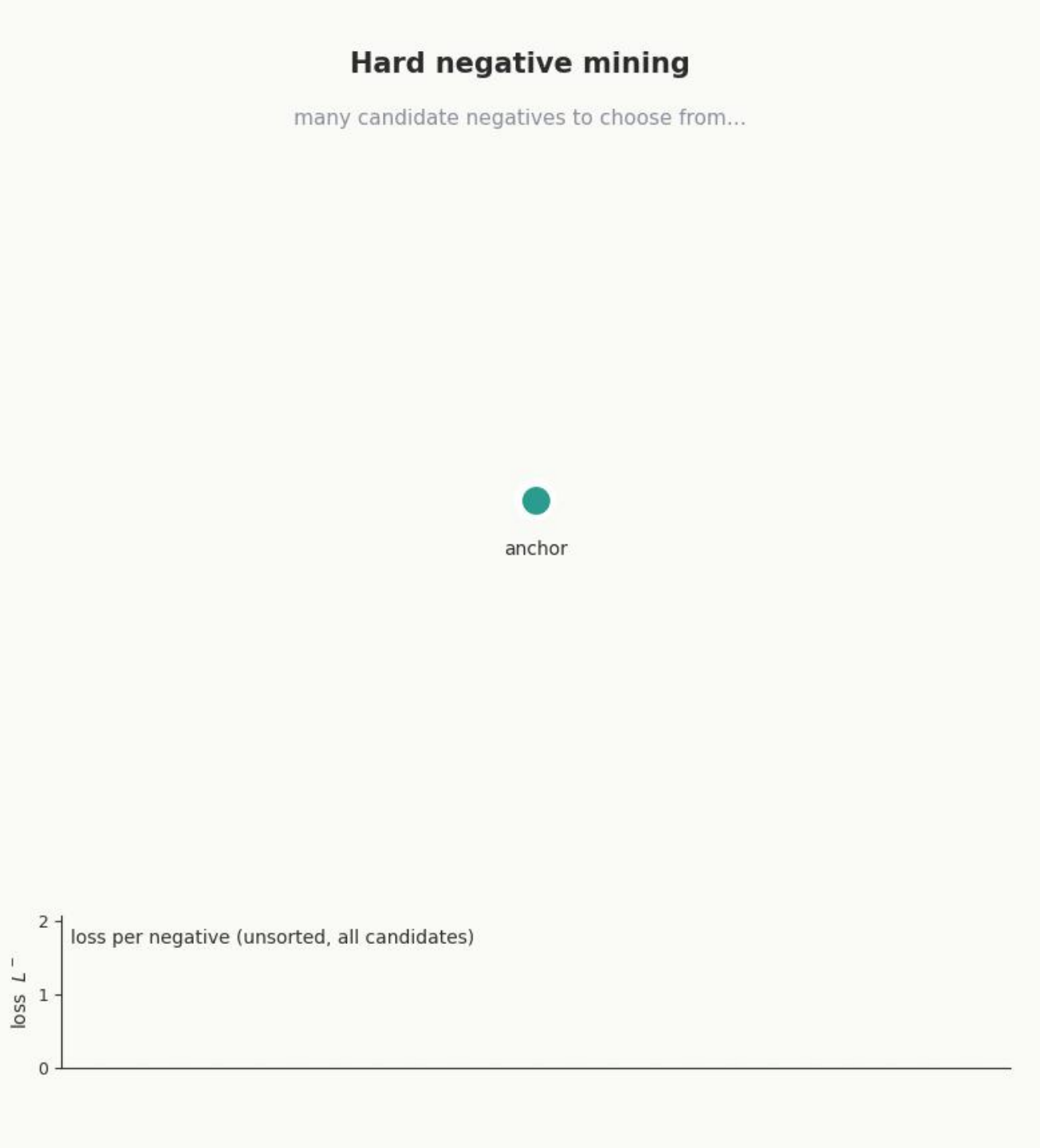
Metric Learning



Metric Learning



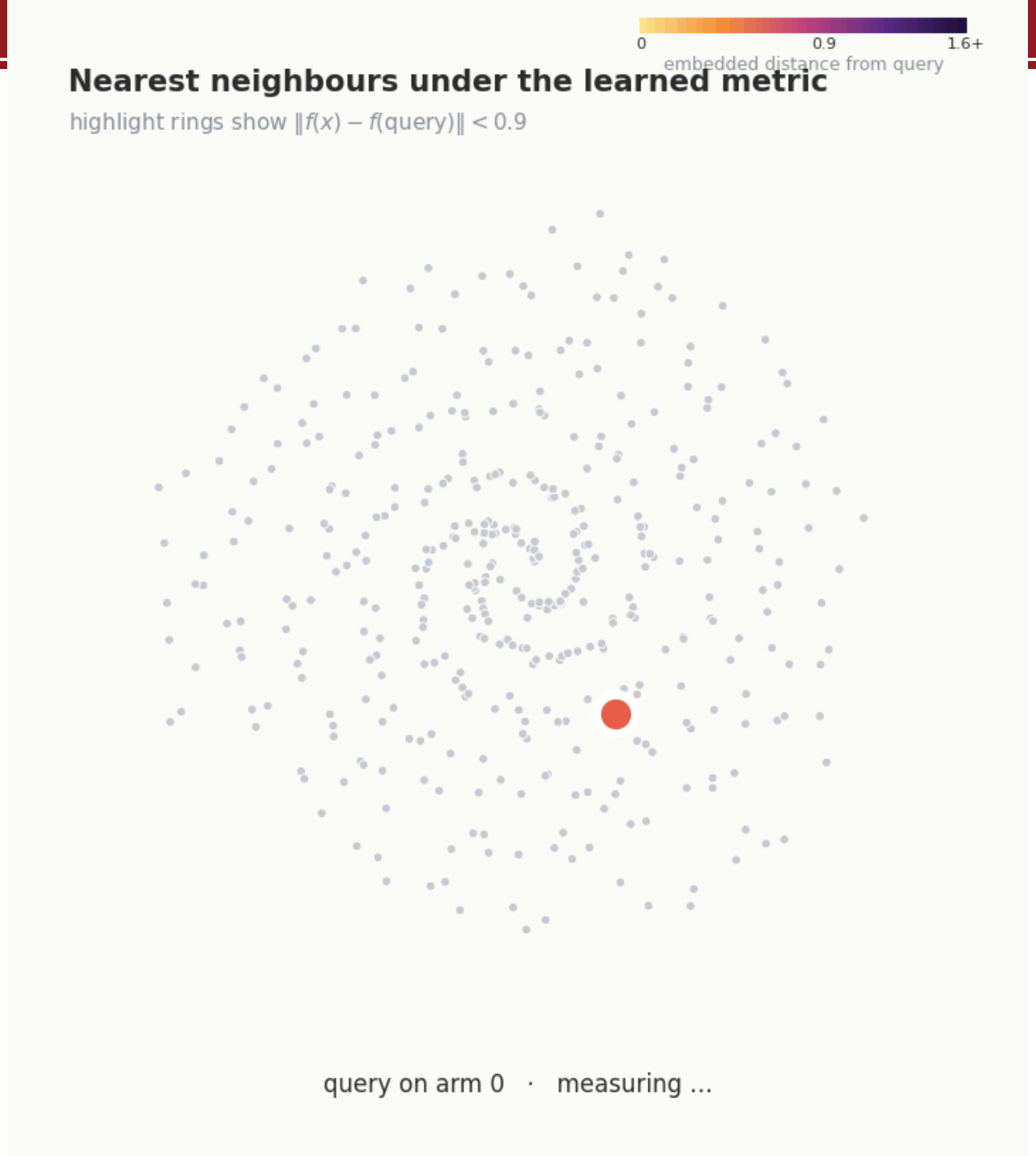
Metric Learning



Training: pairwise hinge + hard negative mining

tanh-bounded MLP $2 \to 128^3 \to 2$ · margin $m = 0.9$ · semi-hard negatives

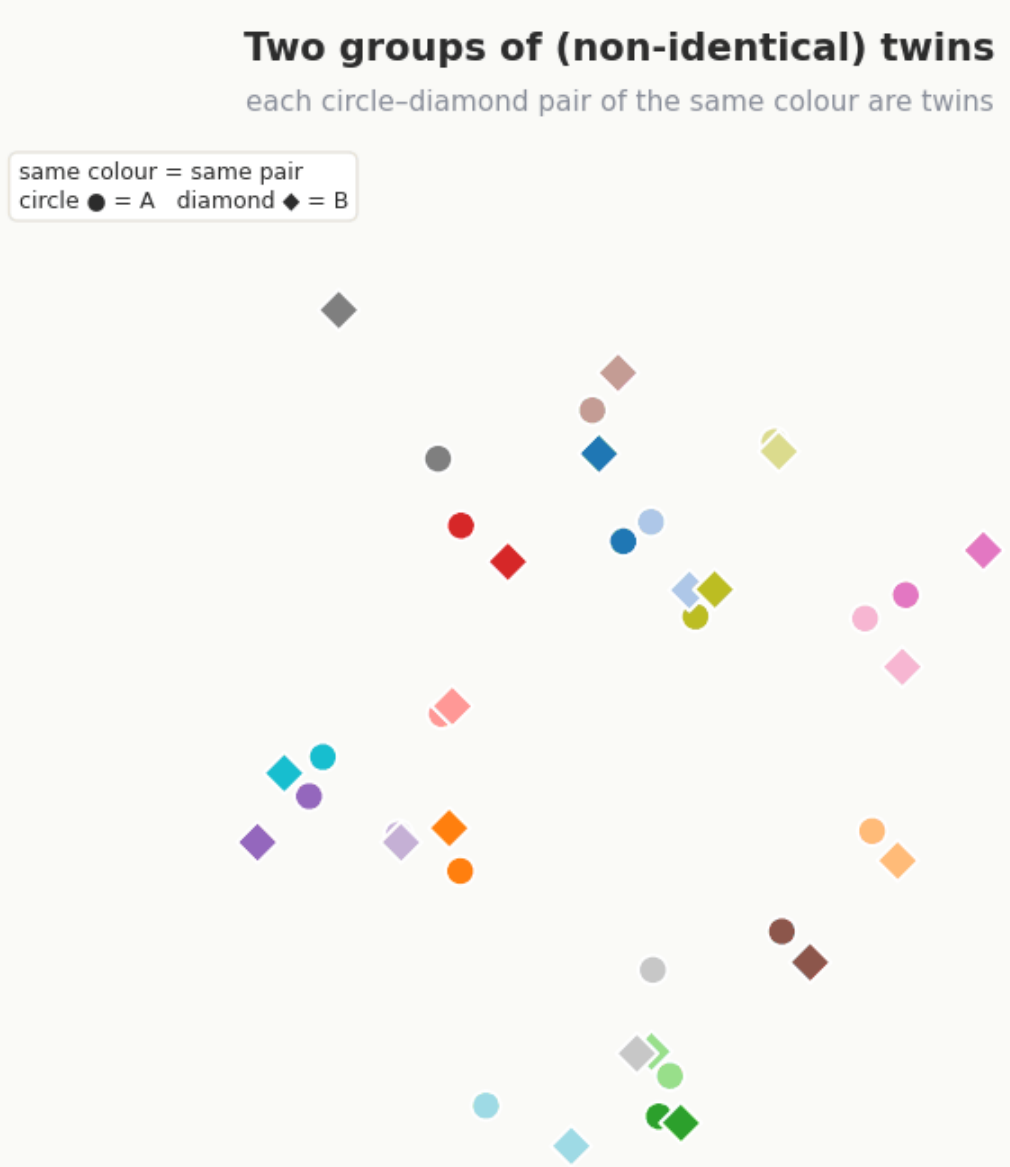
Metric Learning



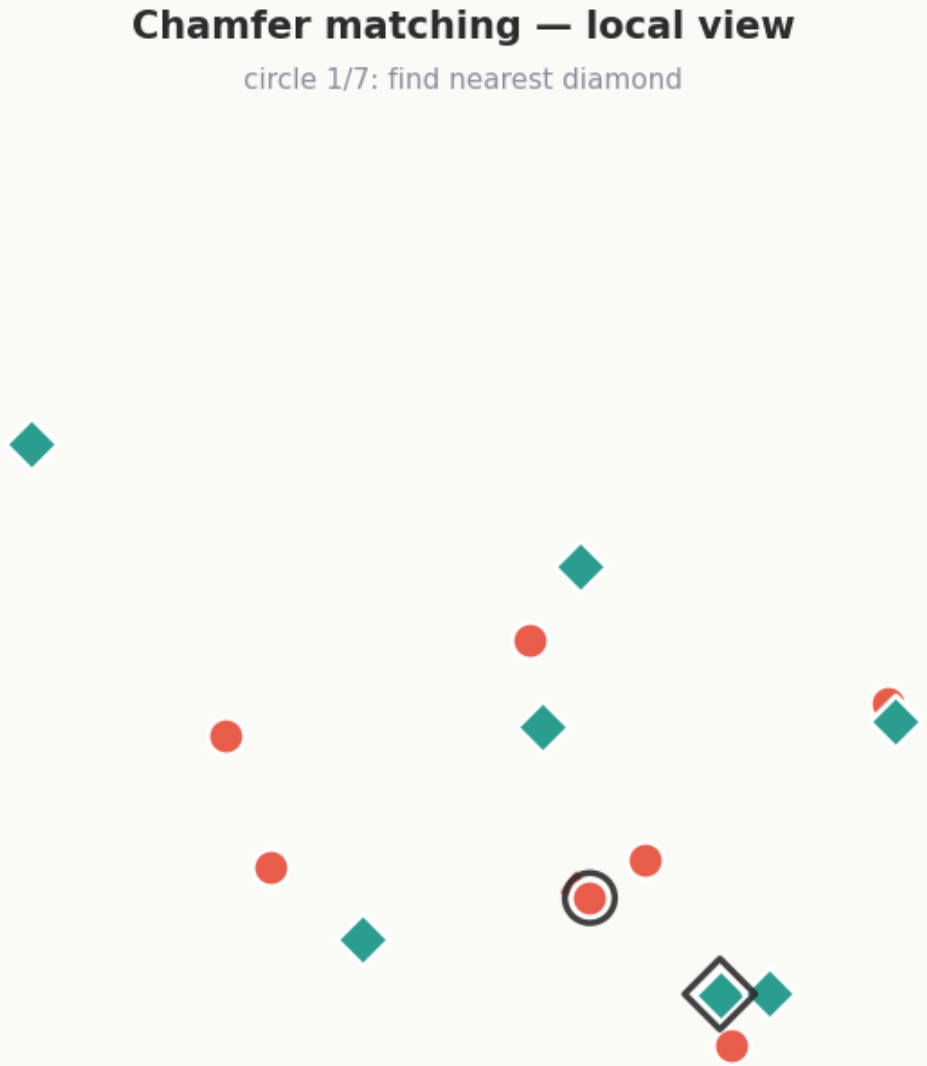
Self-supervised tasks

- Matching and cardinality – two very expensive problems…
- Cardinality is the question of how big is your set/graph
- Matching is the question of how to compare two sets/graphs
- Let’s start with matching: Consider 20 sets of (non-identical) twins, assign each twin to a group A or B, drop them into a party
- Your job is to match up twins based on appearance (e.g. two dimensions – hair colour and height)

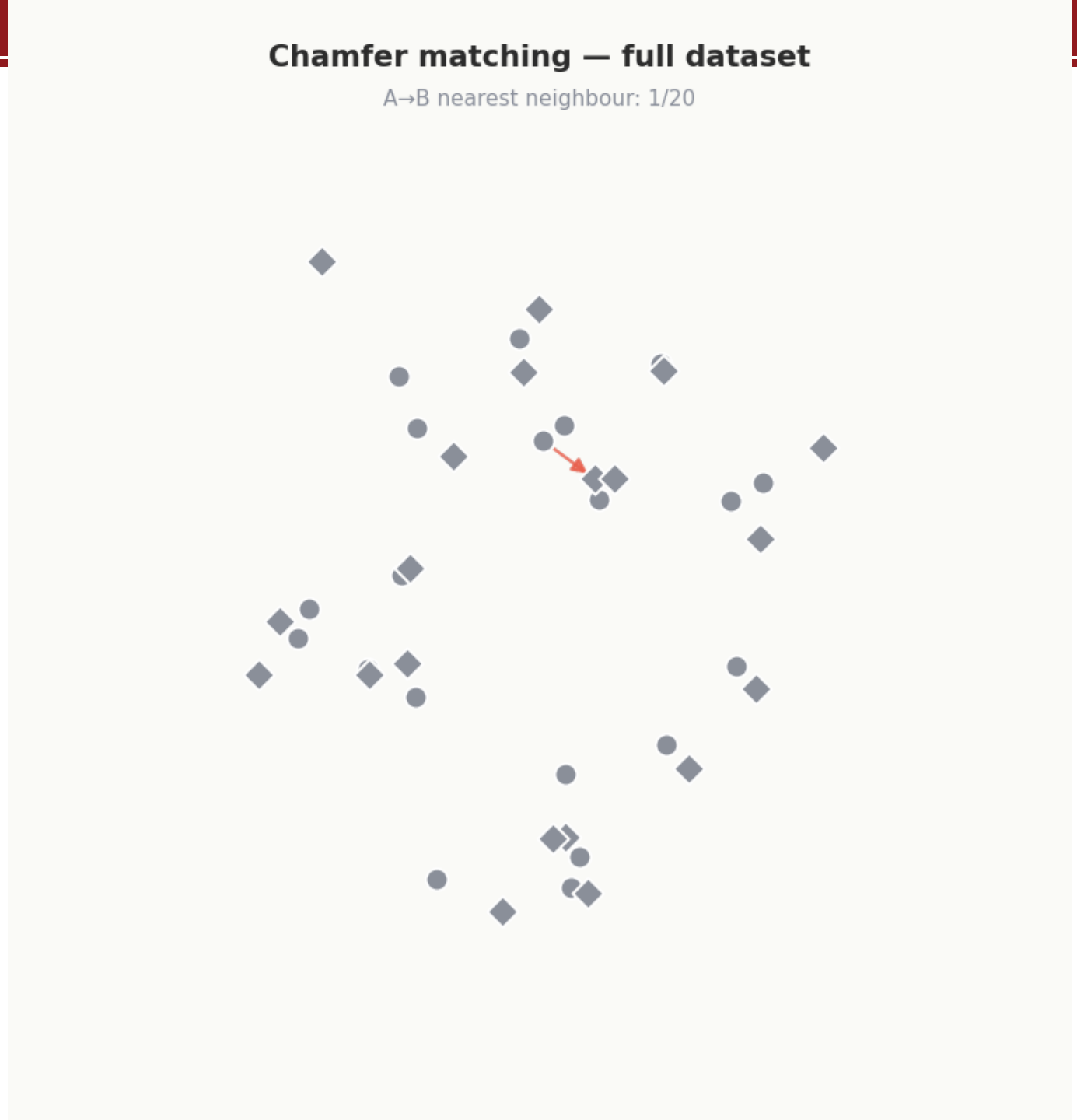
Matching Sets



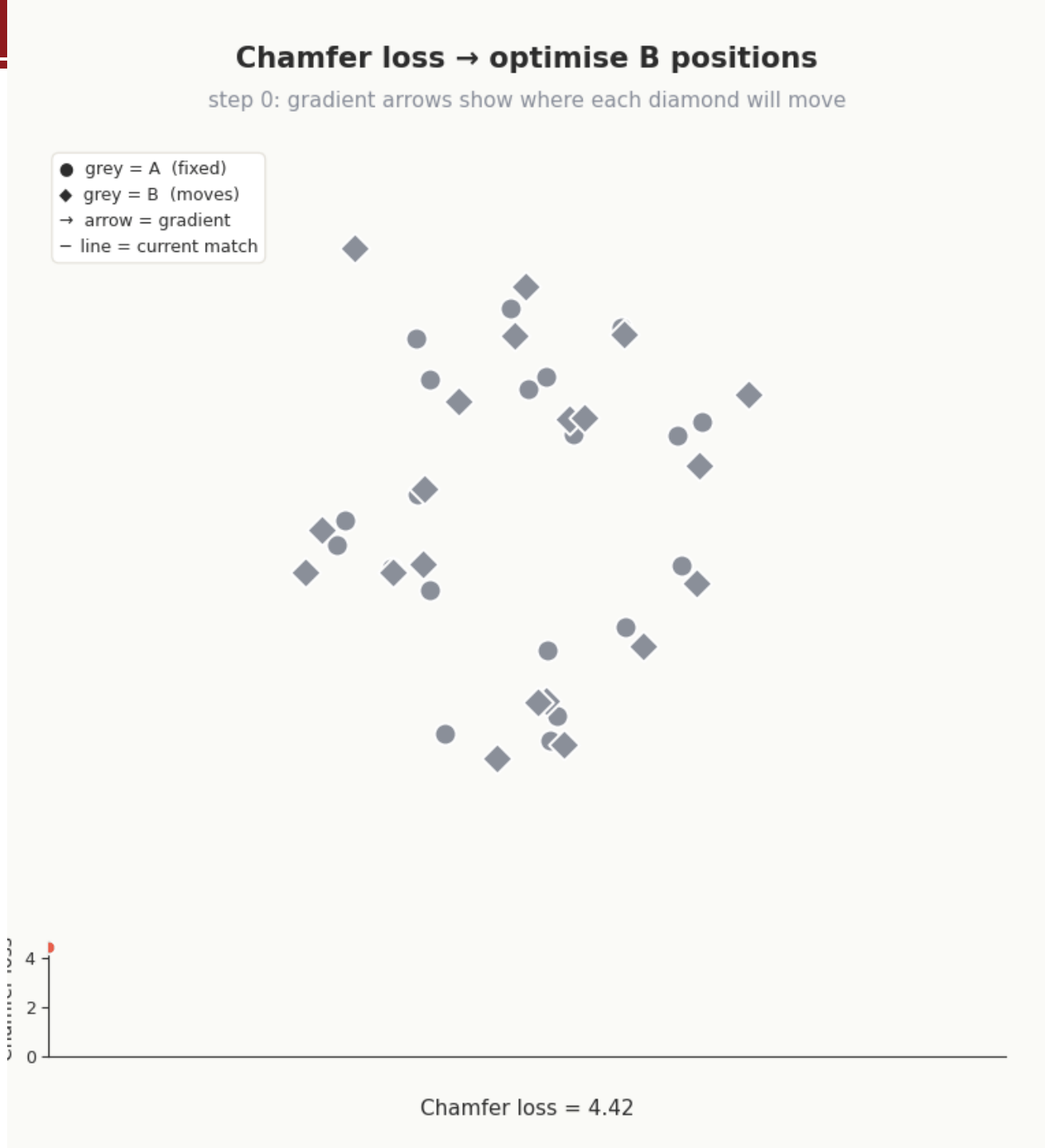
Matching Sets



Matching Sets



Matching Sets



Hungarian matching · exact 1-to-1

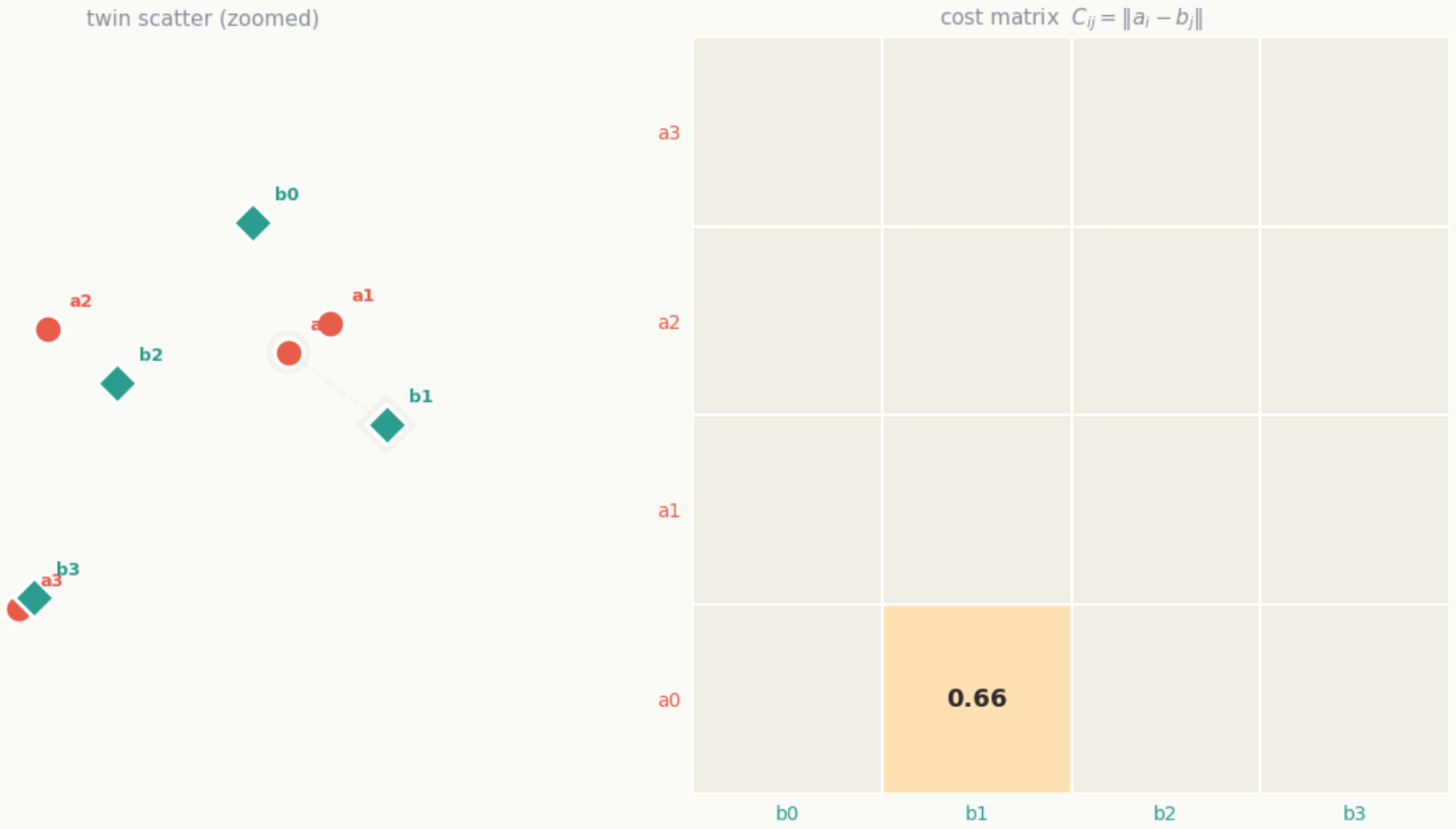
building the pairwise cost matrix

cost matrix $C_{ij} = \|a_i - b_j\|$

row 0/20

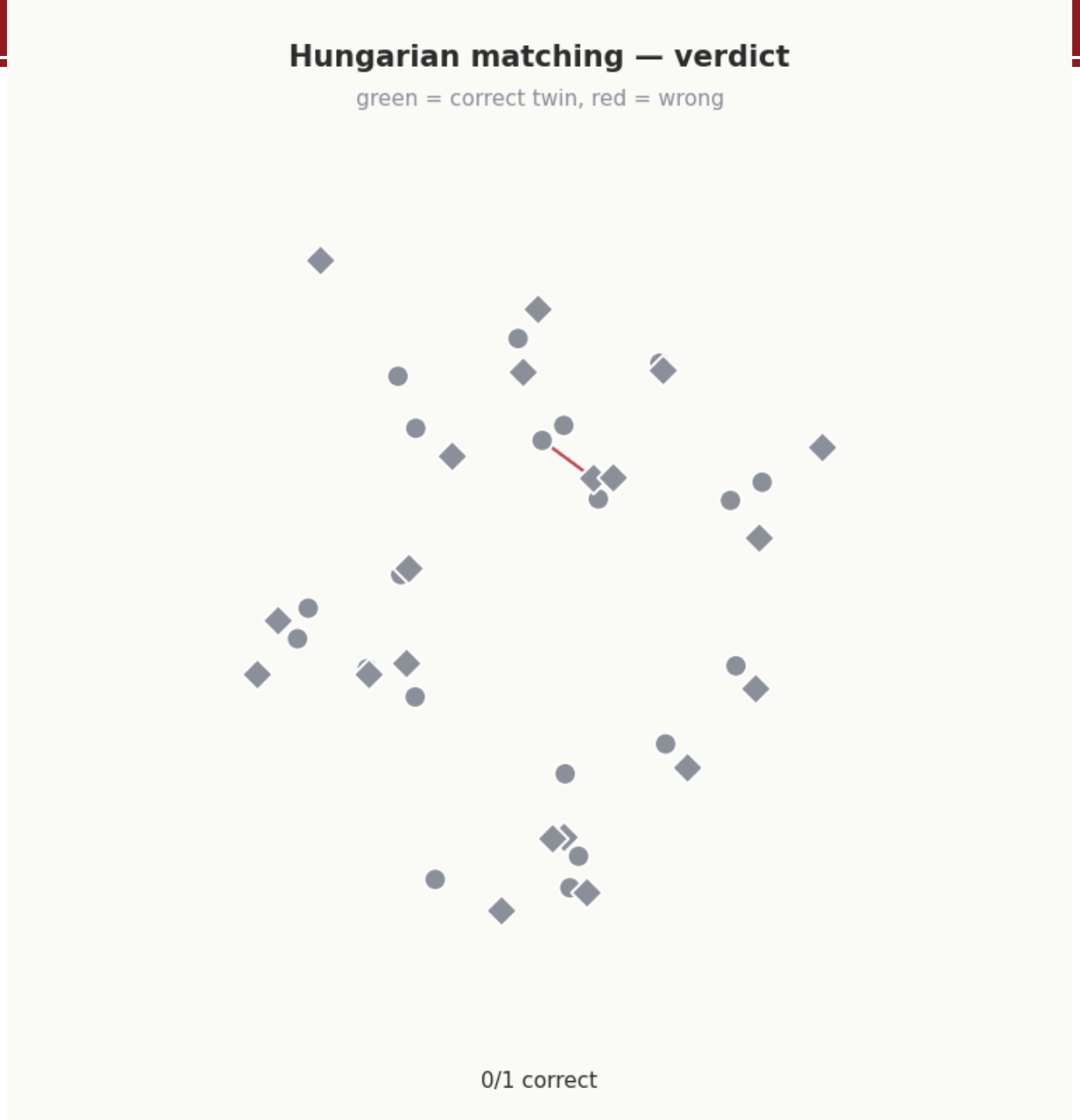
Hungarian matching — the math

$C[a_0, b_1] = \|a_0 - b_1\| = 0.66$

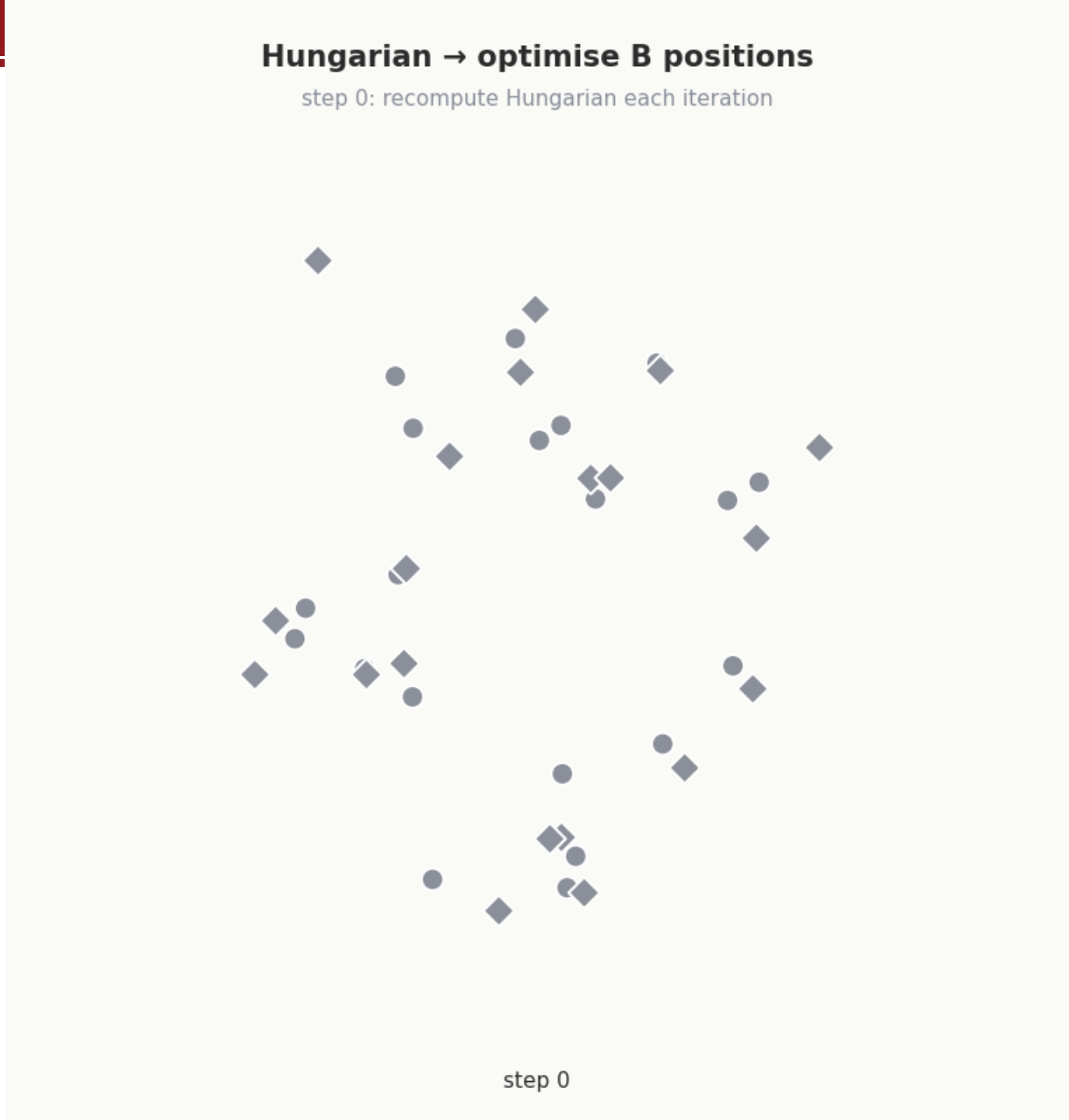


each entry = Euclidean distance between one circle and one diamond

Matching Sets



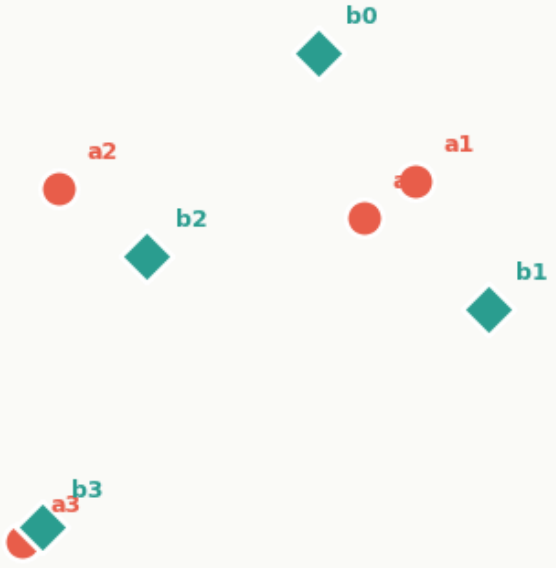
Matching Sets



Sinkhorn — the algorithm

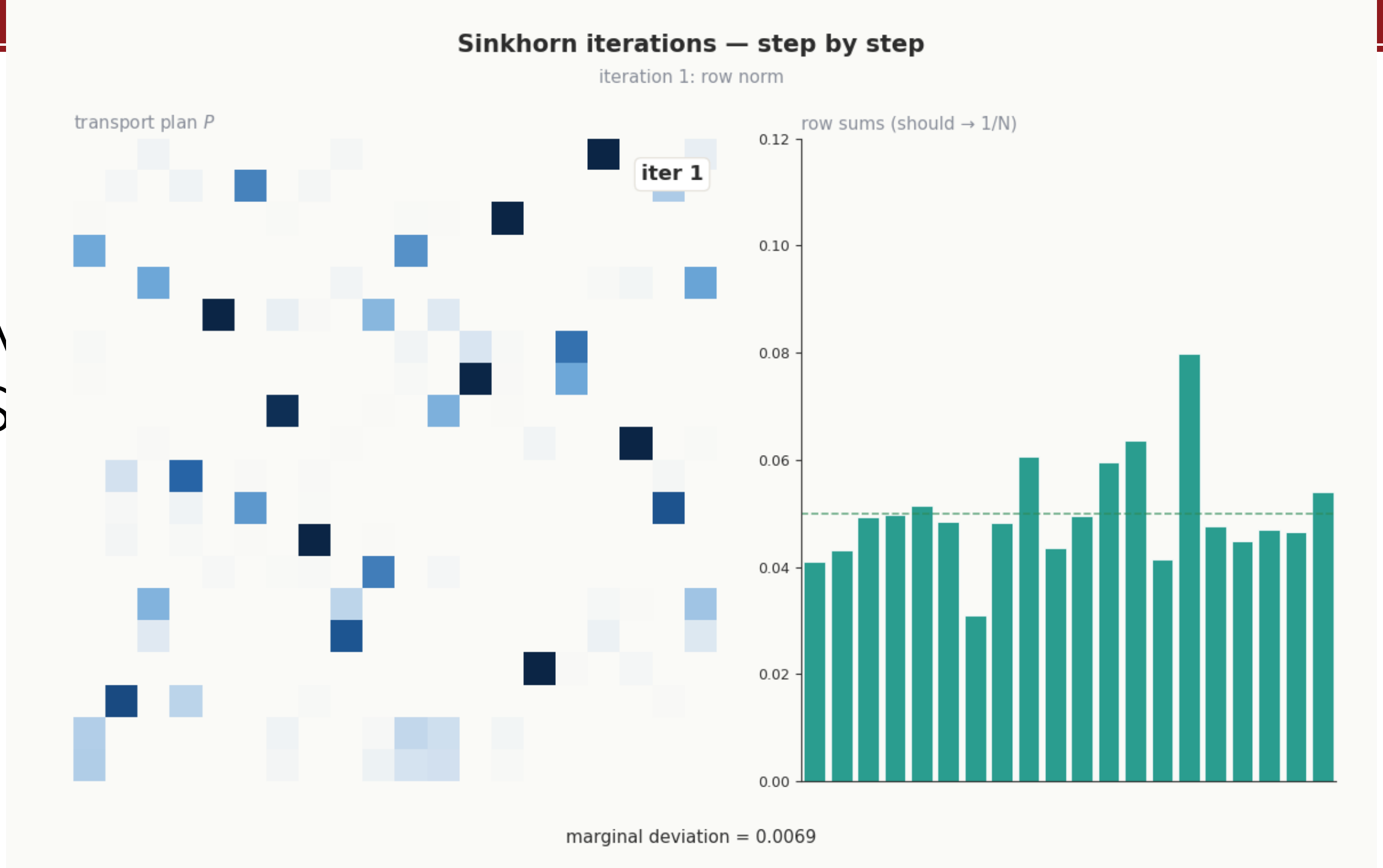
start from pairwise distances (ε = 0.4)

twin scatter

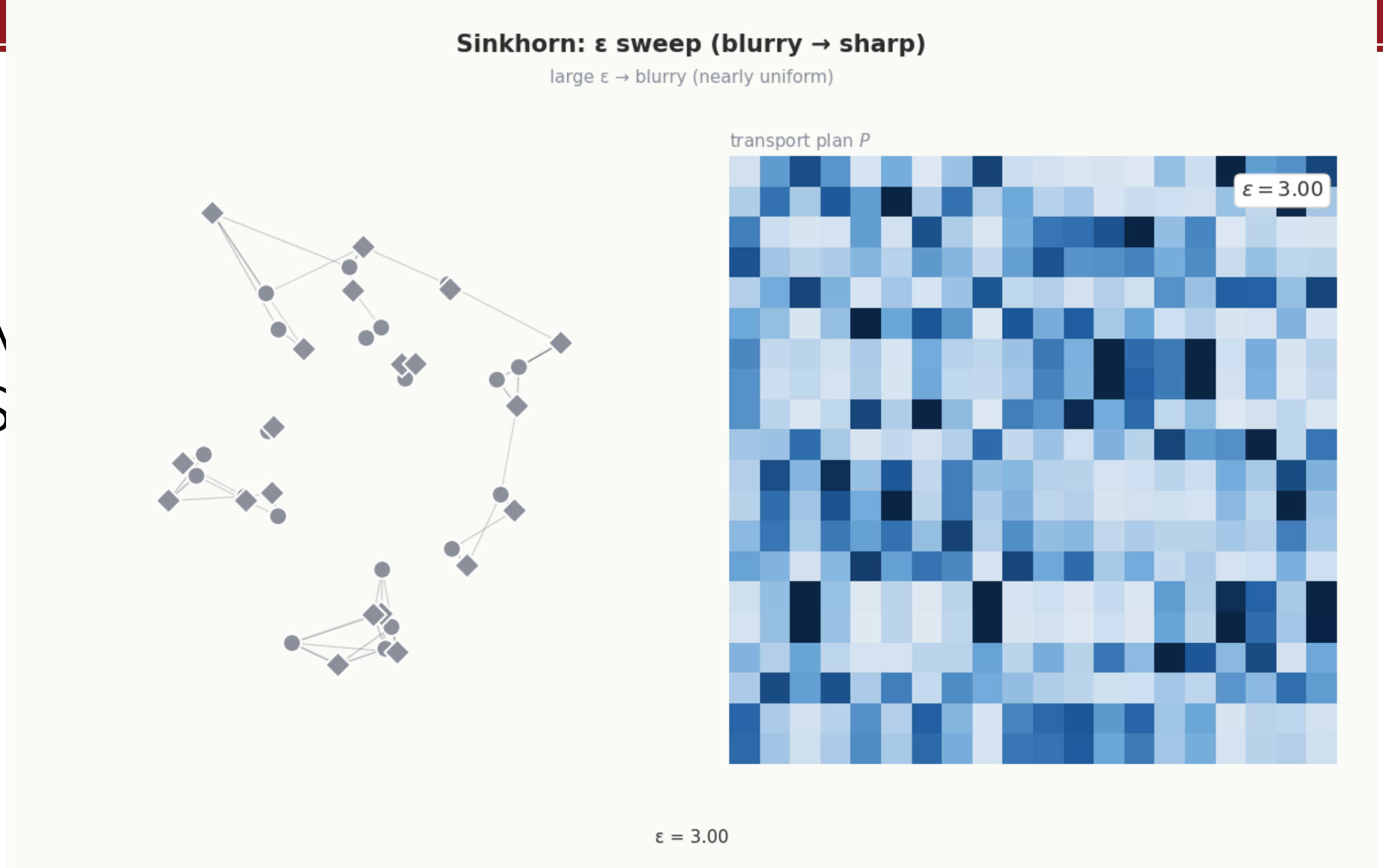


cost matrix $C_{ij} = \|a_i - b_j\|$

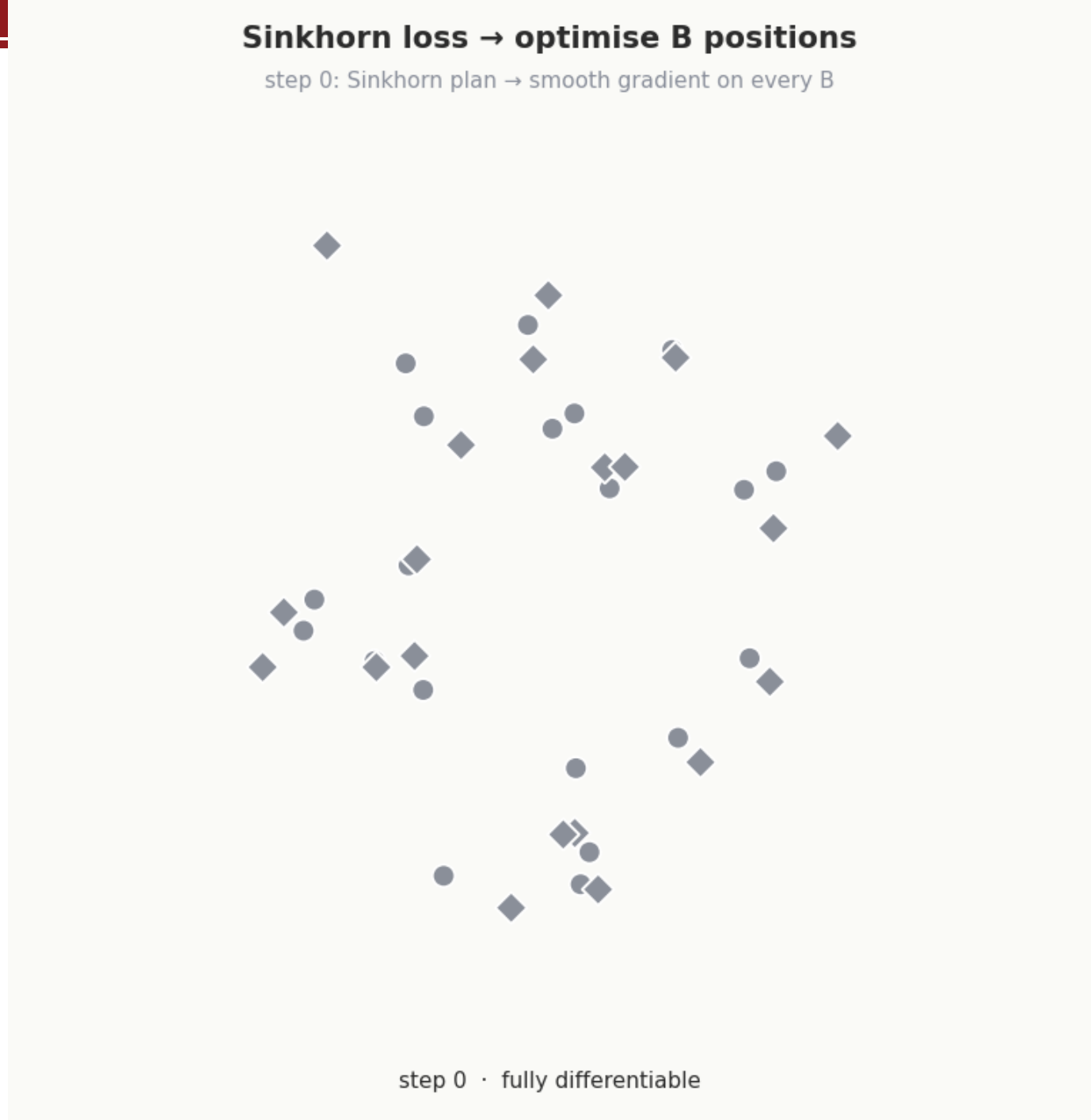
	b0	b1	b2	b3	row sum
a3	2.435	2.218	1.328	0.105	6.086
a2	1.247	1.901	0.475	1.447	5.070
a1	0.687	0.626	1.190	2.170	4.674
a0	0.727	0.658	0.939	1.903	4.227
col sum	5.097	5.404	3.932	5.625	


Sinkhorn iterations — step by step
iteration 1: row norm
transport plan P
iter 1
row sums (should → 1/N)
0.12
0.10
0.08
0.06
0.04
0.02
0.00
marginal deviation = 0.0069

N
S


Sinkhorn: ε sweep (blurry → sharp)
large ε → blurry (nearly uniform)
transport plan P
ε = 3.00
ε = 3.00

Matching Sets



Okay… but what does metric learning and matching have to do with GNNs?

- A graph is a set (a fancy set)
- If you want to do *anything* beyond classification/regression with a graph or set, you need tools like metrics, matching, cardinality prediction
- The next generation of set/graph-based models are almost certainly going to be based on large pre-trained models (i.e. foundation models), which are trained in a self-supervised way

Limitations

- Splitting message passing across GPUs is *really* expensive:
  - Large graphs (O(1 million) nodes), and medium model – gradient checkpointing!
  - Large graphs and large model – maybe switch to transformer, use windowing and model sharding
  - Huge graphs (O(1 billion) nodes) and large model – split the graph into neighborhoods
- Many tasks *seem* like they should use a GNN, but you could recast as a set or sequence or grid/voxel task (then should use e.g. transformer, state-space model, or CNN)
- Graph representations are usually expensive to operate upon (i.e. COO, CSR or CSC are either efficient for storage, row operations or columns operations, but not all three!)
- The most powerful GNNs (in my experience) materialise latent edge features, but these are very expensive to store and backpropagate through