AN ILLUSTRATED, HANDS-ON GUIDE

Graph Neural Networks

A friendly guide to AI that learns from neighbours, for buildings and cities

Plain English, pictures first, a building you turn into a graph,
five labs in plain Python, real numbers from the code, a glossary and a cheat sheet.

Six short chapters · Five labs · Three challenges · Cheat sheet

Prompted and directed by Mahmoud A.
Written and illustrated by Claude (Anthropic)



Prepared for the 2026 CCS: Urban AI + Agent Workshop · Harbin Institute of Technology · November 2026

How to Use This Guide

Read it in order: every chapter ends with something to try.

This guide teaches graph neural networks (GNNs) with one running example: a three-floor office building. You will turn the building into a graph, pass messages between rooms, and predict which rooms are too warm for rooms that nobody has reported on.

YOU WILL BE ABLE TO

- Describe a building or a city as nodes, edges and features.
- Write an adjacency matrix and read degree from it.
- Run one round of message passing by hand and in code.
- Explain the GraphSAGE and GCN layer formulas symbol by symbol.
- Show, with your own numbers, when neighbours improve a prediction.
- Name the three kinds of graph tasks and give a built-environment example of each.

WHAT YOU NEED

- Python 3 with `numpy`. No internet, no GPU.
- The file `gnn_lab.py`. Run it with `python gnn_lab.py`.
- Paper and a pen for the “On paper” exercises.

CONTENTS

1. Why graphs?
2. Graph basics · Lab 1
3. Message passing · Lab 2
4. From messages to a GNN layer
5. Predict the warm rooms · Labs 3–4
6. Two hops and beyond · Lab 5
7. Three kinds of graph tasks
8. From building to city · Challenges
9. Pitfalls, glossary, cheat sheet, reading

About the data. The building and its labels are synthetic. The rule that decides “too warm” is written in `gnn_lab.py`, so you can check every result. Real buildings are noisier; the method is the same.

1 Why Graphs?

Buildings and cities do not come in rows and grids.

Most deep learning works on data with a fixed shape. Text is a line of words, so each word has one word before it and one after it. An image is a grid of pixels, so each pixel has at most eight neighbours. Buildings and cities do not have a fixed shape. One room touches two rooms, another touches six. One street meets three streets, another meets seven.



Text: a line

each word has 2 neighbours

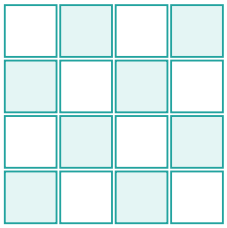
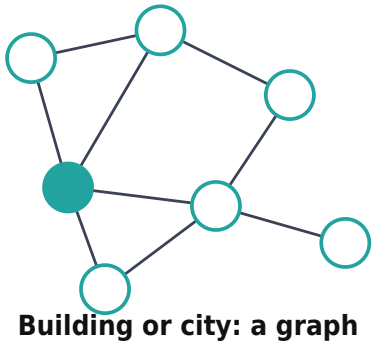


Image: a grid

each pixel has up to 8 neighbours



Building or city: a graph

each room has any number of neighbour

A **graph** handles this. A graph is a set of **nodes** (things) and **edges** (relations between things). Each node can carry **features**: numbers that describe that node. A **graph neural network** is a model that learns from the features of each node and from the features of the nodes connected to that node.

Built-environment graph	Nodes	Edges	Example features
Room adjacency (from BIM)	rooms	shared wall, door, slab	area, window orientation, occupants
Sensor network	sensors, rooms	sensor is in room	temperature, CO_2, light
Person–place	people, places	person rated or used place	age, comfort votes
Street network	intersections	street segments	traffic counts, width
City model	buildings or 3D cells	touching, nearby, contains	height, use, sun hours

KEY IDEA

In a building, what happens in one room depends on the rooms around that room. Heat leaks through walls; people move through doors. A graph lets a model see those neighbours.

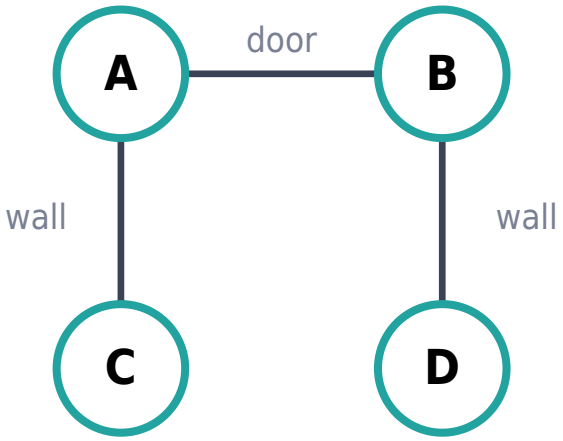
2 Graph Basics

Nodes, edges, features, and one table that holds them.

Take four rooms. Room A has a door to room B and a wall shared with room C. Room B shares a wall with room D. Each relation is an edge.

The **adjacency matrix A** stores the edges. Row i, column j is 1 if node i and node j are connected, and 0 if they are not. For an undirected graph, A is symmetric.

The **degree** of a node is the number of edges at that node: the sum of that node's row in A.



	A	B	C	D	degree
A	0	1	1	0	**2**
B	1	0	0	1	**2**
C	1	0	0	0	**1**
D	0	1	0	0	**1**

room	sun hours	occupants
A	6	2
B	2	8
C	4	4
D	0	6

The feature matrix X: one row per room, one column per feature.

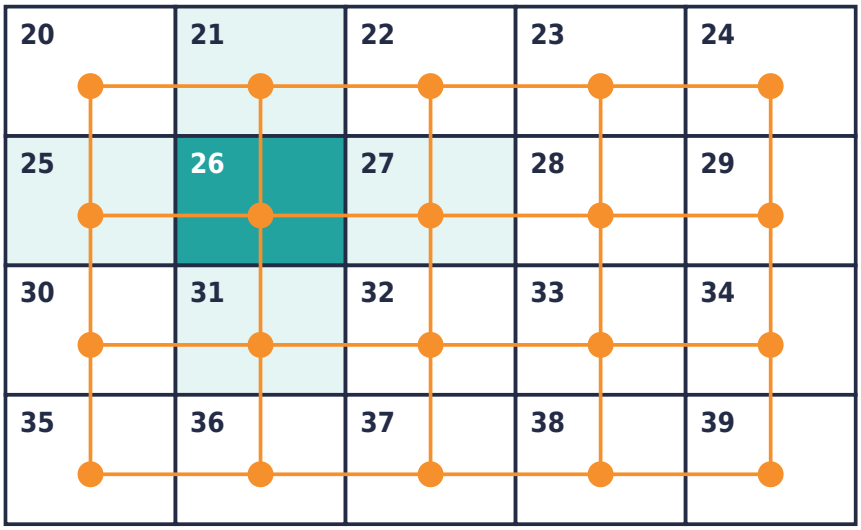
ON PAPER · EXERCISE 1

1. Add a door between room C and room D. Write the new matrix A and the new degrees.
2. Which room now has the highest degree? Why does that room matter for heat spreading?
3. Compute the mean sun hours of the neighbours of room A, using the original graph. *Answer: (2 + 4) / 2 = 3*

L1 Turn the Building into a Graph

Sixty rooms, 133 edges, ten lines of numpy.

The lab building has 3 floors. Each floor is a grid of 4 × 5 rooms, so the building has 60 rooms. Two rooms are connected if they share a wall on the same floor, or if one room is directly above the other. Room 26 is on floor 1, so room 26 touches four rooms on floor 1, room 6 below and room 46 above.



Floor 1 (rooms 20–39): walls become edges



Floor slabs are edges too

```
ROWS, COLS, FLOORS = 4, 5, 3          # 3 floors, each a 4 x 5 grid of rooms
N = ROWS * COLS * FLOORS              # 60 rooms
def rid(f, r, c):                      # room id from floor, row, column
    return f * ROWS * COLS + r * COLS + c

edges = []
for f in range(FLOORS):
    for r in range(ROWS):
        for c in range(COLS):
            if c + 1 < COLS:   edges.append((rid(f, r, c), rid(f, r, c + 1)))  # wall, east
            if r + 1 < ROWS:   edges.append((rid(f, r, c), rid(f, r + 1, c)))  # wall, south
            if f + 1 < FLOORS: edges.append((rid(f, r, c), rid(f + 1, r, c)))  # slab, above

A = np.zeros((N, N))                   # adjacency matrix
for i, j in edges:
    A[i, j] = A[j, i] = 1              # undirected: heat flows both ways
deg = A.sum(1)                         # degree of every room
```

```
rooms: 60 | edges: 133
room 26 touches rooms: [6, 21, 25, 27, 31, 46] | degree: 6
labelled rooms: [1, 5, 6, 12, 14, 15, 19, 21, 22, 31, 32, 36, 38, 41, 44, 45, 53, 54, 55, 59]
```

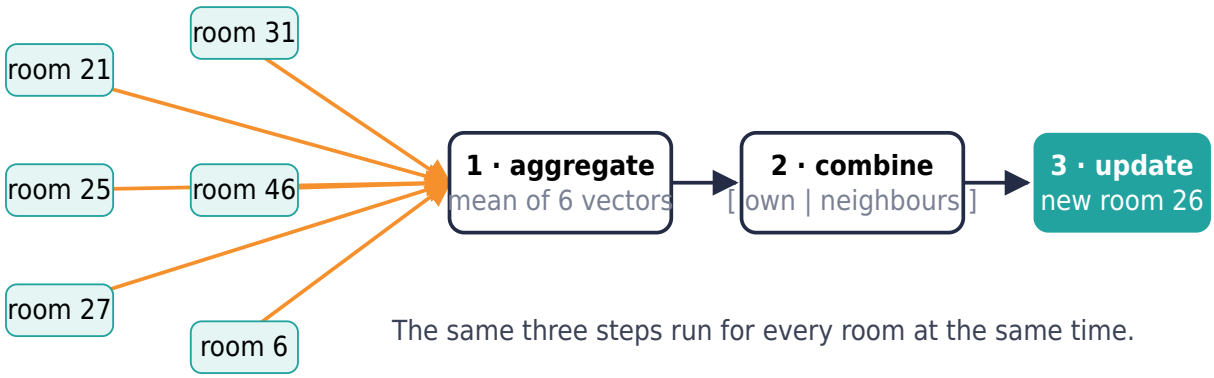
CHECK YOUR UNDERSTANDING

1. Which rooms have the lowest degree, and why? (Hint: corners on the top and bottom floors.)
2. Count the edges by hand for one floor: 4 rows × 4 east walls + 3 rows of south walls × 5. Do you get 31? Then add the slabs.
3. A door is not the same as a wall. How would you give edges different weights or types?

3 Message Passing

Every room listens to the rooms it touches.

Message passing is the core of every graph neural network. In one round, every node does three things at the same time:



1. **Aggregate:** collect the feature vectors of the neighbours and summarise them, for example with a mean.
2. **Combine:** put the node's own vector next to the summary of the neighbours.
3. **Update:** transform the combination into a new vector for the node (a weight matrix, then a non-linear function).

$$h_v^{new} = \sigma(\, W \cdot [\, h_v \,\|\, \text{mean}_{u \in N(v)}\, h_u \,]\,)$$

h_v: vector of node v · N(v): neighbours of v · ‖: put two vectors side by side · W: learned weights · σ: a non-linear function such as ReLU

Lab 2 · one round for room 26

```
def mean_of_neighbours(A, H):
    """Each room collects the features of the rooms it touches and averages them."""
    return (A @ H) / A.sum(1, keepdims=True)

M = mean_of_neighbours(A, X)          # X = [sun hours, occupants, area] per room
print(X[26], M[26])
```

```
room 26 own [sun, occupants, area]:       [ 3.73  1.   35.26]
room 26 mean of neighbours [sun, occ, area]: [ 2.58  5.83 29.3 ]
```

ON PAPER · EXERCISE 2

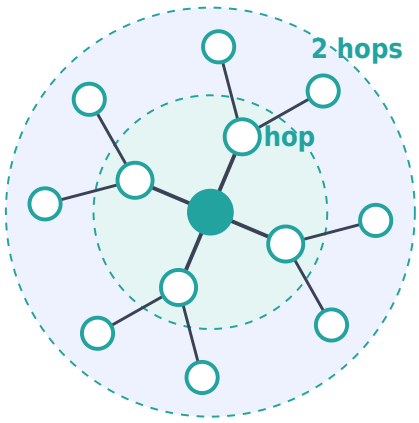
Room 26 has little sun (3.7 h) and only one occupant, but the neighbours of room 26 have 5.8 occupants on average. Before running any model: do you expect room 26 to be too warm? Why can a model that looks only at room 26 get this room wrong?

4 From Messages to a GNN Layer

Four famous layers are four ways to listen.

A **GNN layer** is one round of message passing with learned weights. Different GNNs differ mainly in how they aggregate and combine.

Layer	Formula	In words
GraphSAGE (mean) Hamilton et al., 2017	$h'_v = \sigma(W [h_v \Vert \text{mean}_{N(v)} h_u])$	Keep the node's own vector separate from the neighbours' average. Good when the node itself matters.
GCN Kipf & Welling, 2017	$H' = \sigma(S H W)$, $S = \hat{D}^{-½} (A + I) \hat{D}^{-½}$	Add a self loop, then take a weighted average of the node and the neighbours. Nodes with many neighbours are scaled down.
GAT Veličković et al., 2018	$h'_v = \sigma(\Sigma_u \alpha_{vu} W h_u)$	Learn an attention weight α for every edge: some neighbours count more than others.
R-GCN Schlichtkrull et al., 2018	a separate W_r for each edge type r	Walls, doors and slabs carry heat and people differently, so each edge type gets its own weights.



k layers = information from k hops away

Stacking layers

One layer lets a room see the rooms one step away. Two layers let a room see two steps away, because the neighbours have already collected from their own neighbours. This range is called the **receptive field**.

More layers are not always better. After many layers, every room has averaged over most of the building and all rooms start to look the same. This problem is called **over-smoothing**. In buildings and cities, two or three layers are a common starting point.

L3

Predict the Warm Rooms

Labs 3 and 4 · 20 rooms reported, 40 rooms to guess.

The task. Occupants have reported whether 20 of the 60 rooms are too warm in the afternoon. Predict the other 40 rooms. In the synthetic building, a room is warmed a little by its own sun and people, and mostly by heat leaking in from the rooms it touches.

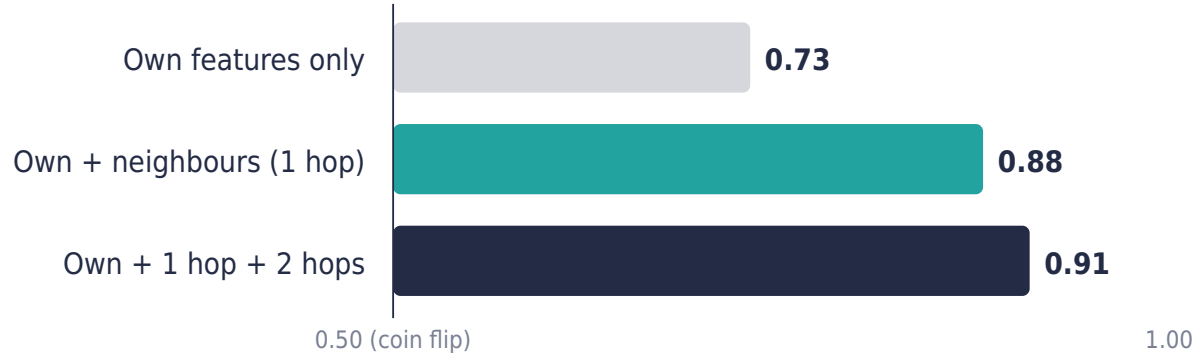
Two models, same classifier. Both use logistic regression trained on the 20 labelled rooms. Model 1 sees only each room's own features. Model 2 sees each room's own features plus the mean of its neighbours' features: one GraphSAGE-style layer.

```
def sage_features(A, H):
    """One GraphSAGE-style layer (mean aggregator): [own | mean of neighbours]."""
    return np.c_[H, mean_of_neighbours(A, H)]

pred_no_graph = logistic_regression(standardise(X), y, train)
pred_graph    = logistic_regression(standardise(sage_features(A, X)), y, train)
```

```
accuracy on 40 unlabelled rooms, own features only:       0.78
accuracy on 40 unlabelled rooms, own + neighbour features: 0.9

Lab 4b: the same test on 50 different random labelled sets
mean accuracy, own features only:       0.73
mean accuracy, own + neighbour features: 0.88
```



Mean accuracy on the 40 unlabelled rooms, over 50 different random choices of the 20 labelled rooms. Output of `gnn_lab.py`.

DISCUSS

1. Why does one random split (0.78 vs 0.90) differ from the 50-split mean (0.73 vs 0.88)? Which number would you report?
2. Look at the rooms Model 2 still gets wrong. Where are they in the building?
3. In a real building, what else should become a feature or an edge: doors, orientation, HVAC zones?

L5

Two Hops, and What a Full GNN Adds

Stretch lab · neighbours of neighbours.

Use the GCN propagation matrix S to reach two steps away. Multiplying by S once averages over the neighbours; multiplying twice reaches the neighbours of the neighbours.

```
def normalised_adjacency(A):
    A_hat = A + np.eye(len(A))                  # self loop: a room listens to itself
    D_inv_sqrt = np.diag(1 / np.sqrt(A_hat.sum(1)))
    return D_inv_sqrt @ A_hat @ D_inv_sqrt      # S in the GCN formula

S = normalised_adjacency(A)
two_hop = np.c_[X, S @ X, S @ S @ X]            # own, 1 hop, 2 hops
```

```
mean accuracy, own + 1-hop + 2-hop (GCN propagation): 0.91
```

The gain from 0.88 to 0.91 is small. In this building, heat comes mostly from the rooms directly touching each room, so the second hop adds little. Try changing the heat rule in `gnn_lab.py` so heat travels two rooms, and see whether the second hop matters more.

What we simplified

In this lab	In a full GNN
Message passing with fixed averages, then one classifier	Learned weight matrices inside every layer
No non-linearity between hops	A non-linear function (ReLU) after each layer
Features computed once	All layers trained together, end to end, by gradient descent
numpy only	Libraries such as PyTorch Geometric or DGL

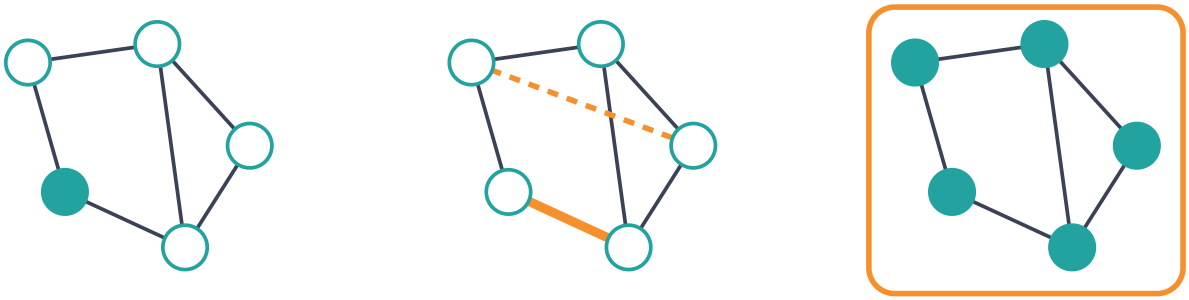
The simplified version is close to the “simplified graph convolution” of Wu et al. (2019), and the result points in the same direction: much of the gain comes from letting each node see its neighbours.

STRETCH EXERCISE

Write a 2-layer GCN in numpy: `H1 = relu(S @ X @ W1)`, `P = softmax(S @ H1 @ W2)`. Train W1 and W2 with cross-entropy on the labelled rooms only. Compare the result with Model 2. With only 20 labels, does the deeper model overfit?

5 Three Kinds of Graph Tasks

Predict a node, an edge, or the whole graph.



Node task
Is this room too warm?

Edge task
Should these two be linked?

Graph task
What energy class is this building?

Task	Predicts	Built-environment examples
Node	a label or value for each node	Is this room too warm? What is the occupancy of this space? What is the land use of this parcel?
Edge (link prediction)	whether an edge exists, or its value	Which place will this person like? Which two sensors are in the same zone? Traffic flow on a street segment.
Graph	one label for the whole graph	Energy class of a building. Type of a neighbourhood from its street pattern.

Recommendation is an edge task

Put people and places in one graph. An edge joins a person and a place when that person has rated or used that place. Predicting a missing rating is predicting a missing edge. A GNN learns a vector for every person and every place from the graph, and the dot product of a person vector and a place vector scores how well they match. This connects to the comfort recommendation exercise of this course.

From my own work

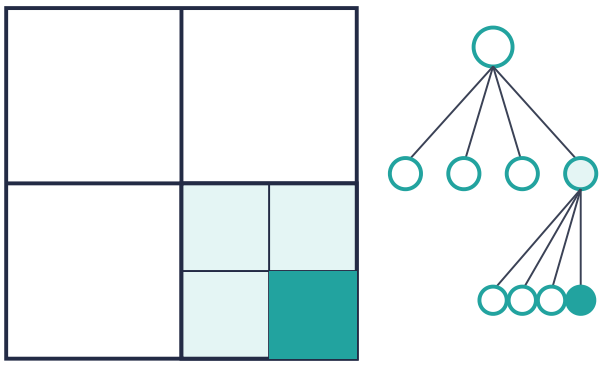
Build2Vec (Abdelrahman, Chong & Miller, *Building and Environment*, 2022) turns a building model into a graph of spaces and objects, learns a vector for each node, and uses those vectors to improve the prediction of occupants' personal thermal comfort. The idea is the one in this guide: a space is described partly by what is around that space.

6 From Building to City

Same tools, bigger graphs.

The same tools work at city scale. Only the nodes and edges change.

- **Street networks:** intersections as nodes, segments as edges. Used for traffic forecasting, for example DCRNN (Li et al., 2018).
- **Building neighbourhoods:** buildings as nodes, edges between buildings closer than a distance d. Useful for heat, energy and land-use questions.
- **3D city cells:** in OctreeCity, the city is split into cells that are small where detail matters and large in open air. Each cell is a node; edges join a parent cell to its children and join cells that touch.



cells = nodes · touching cells = edges

CHALLENGE A · A CITY GRAPH

Make 200 random building centroids in a 1 km × 1 km square. Join every pair of buildings closer than 60 m. Give each building a height. Label a building “shaded” if the mean height of its neighbours is greater than its own height. Repeat Lab 4: does adding neighbour features help?

CHALLENGE B · PERSON-PLACE GRAPH

Make 30 people and 15 places. Each person rates 4 places from 1 to 5. Build the bipartite adjacency matrix. Predict a missing rating as the mean rating that similar people gave that place, where “similar” means people who rated the same places. Which step is message passing?

CHALLENGE C · EDGE TYPES

Give walls weight 1, doors weight 0.5 and slabs weight 0.3 in the adjacency matrix. Change `mean_of_neighbours` to a weighted mean. Do the predictions change?

Pitfalls and Glossary

Read this before you use GNNs on real data.

Pitfalls checklist

- **Spatial leakage.** Rooms in the test set touch rooms in the training set, so information leaks. For a fair test, hold out whole floors, buildings or districts.
- **Missing edges.** A graph from a BIM model can miss doors, open plans or ducts. Check the graph against drawings before training.
- **Over-smoothing.** Too many layers make all nodes look alike. Start with 2 layers.
- **Scale.** A city graph can have millions of nodes. Use neighbour sampling (GraphSAGE) or work tile by tile.
- **Feature scale.** Standardise features first. Area in m² and occupants in persons have very different ranges.
- **One split is not a result.** Report the mean over many random splits, as in Lab 4b.
- **Explain.** A planner will ask why. Show which neighbours drove a prediction.

Glossary

Node
A thing in the graph: a room, a sensor, a building, a cell.

Edge
A relation between two nodes: shares a wall, is inside, is near.

Feature
A number that describes a node or an edge.

Adjacency matrix A
N × N table; entry (i, j) is 1 when node i and node j are connected.

Degree
Number of edges at a node.

Message passing
Each node collects, combines and updates information from its neighbours.

Embedding
A learned vector that describes a node.

Receptive field
How many steps away a node can see; equal to the number of layers.

Over-smoothing
After many layers, all node vectors become similar.

Semi-supervised
Labels exist for some nodes only; the graph helps predict the rest.

One-Page Cheat Sheet

Everything in this guide on one page.

GRAPH

G = (V, E). N nodes, feature matrix X (N × F), adjacency A (N × N), degree $d_i = \Sigma_j A_{ij}$.

MEAN OF NEIGHBOURS

$M = D^{-1} A X$

```
(A @ X) / A.sum(1, keepdims=True)
```

GRAPHSAGE LAYER

$h'_v = \sigma(W [h_v \| \text{mean}_{N(v)} h_u])$

GCN LAYER

$H' = \sigma(\hat{D}^{-½} (A+I) \hat{D}^{-½} H W)$

GAT LAYER

$h'_v = \sigma(\Sigma_{u \in N(v) \cup \{v\}} \alpha_{vu} W h_u)$

K LAYERS

Each node sees nodes up to k steps away. Start with k = 2.

NODE TASK

Classify or regress each node. Loss on labelled nodes only.

EDGE TASK

Score a pair: $s(u, v) = h_u \cdot h_v$. Recommendation, link prediction.

GRAPH TASK

Pool all node vectors (mean or sum), then classify the whole graph.

FAIR TEST

Split by floor, building or district, not by random node.

LAB RESULTS (SYNTHETIC)

Own features 0.73 · + 1 hop 0.88 · + 2 hops 0.91 (mean of 50 splits).

LIBRARIES

numpy to learn · PyTorch Geometric or DGL for real projects.

Further Reading

The papers behind each idea.

1. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *ICLR*.
2. Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs (GraphSAGE). *NeurIPS*.
3. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. *ICLR*.
4. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. *ICML*.
5. Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., & Welling, M. (2018). Modeling relational data with graph convolutional networks. *ESWC*.
6. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. (2019). Simplifying graph convolutional networks. *ICML*.
7. Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: online learning of social representations. *KDD*.
8. Grover, A., & Leskovec, J. (2016). node2vec: scalable feature learning for networks. *KDD*.
9. Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2018). Diffusion convolutional recurrent neural network: data-driven traffic forecasting. *ICLR*.
10. Hamilton, W. L. (2020). *Graph Representation Learning*. Morgan & Claypool.
11. Abdelrahman, M., Chong, A., & Miller, C. (2022). Build2Vec: building representation in vector space. *Building and Environment*.

Files · `gnn_lab.py` (all labs) · `gnn_lab.ipynb` (the same code as a notebook)

Part of the Fundamentals 101 series · mahmoud.work