Advanced Topic A10: *Graph Neural Networks*

Data Science with Python – BSc Advanced Lectures

Joerg Osterrieder

10 Minutes

Motivation: When Data Has Structure Beyond Rows

Many financial datasets are naturally relational – tables lose the structure

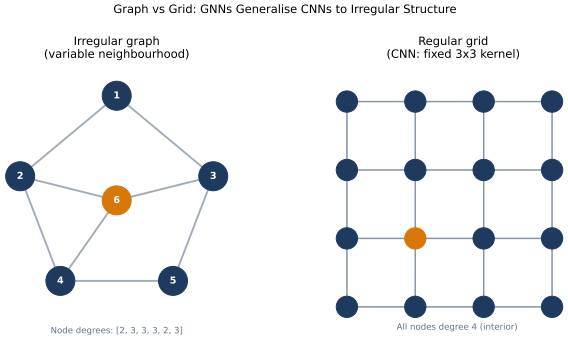
- Standard ML assumes i.i.d. rows: each sample is independent and featurised individually
- Many real systems are graphs: transaction networks, corporate ownership chains, interbank exposure matrices, supply chains, knowledge graphs
- Flattening a graph into a feature vector loses topological information (which nodes are connected, how many hops away, what cycles exist)
- GNNs operate directly on the graph: they learn representations that exploit connectivity
- Finance: a fraudulent transaction looks ordinary in isolation but anomalous in the network

GNNs are the standard tool when the data has graph structure that matters for the prediction

Graphs vs Grids

CNNs work on regular grids; GNNs work on irregular graphs

- **Grid (CNN):** fixed neighbourhood (3x3 kernel); shared weights by translation; convolution is well-defined because all nodes have the same local structure
- **Graph (GNN):** variable degree; no canonical ordering of neighbours; no notion of "left" or "right" neighbour
- GNNs solve this by aggregating over neighbours without caring about their order (permutation-invariant)
- A GNN on a grid with regular structure reduces to a CNN: they are the same family



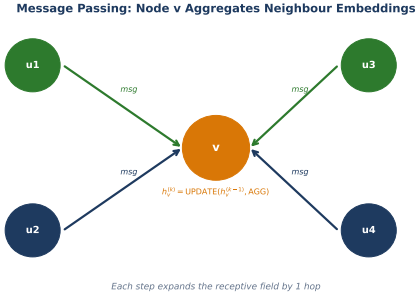
GNNs generalise CNNs: regular grids are a special case of graphs with uniform degree

Message Passing: The Core Mechanism

Each node aggregates information from its neighbours iteratively

$$h_v^{(k)} = \text{UPDATE}\Big(h_v^{(k-1)},\ \text{AGGREGATE}\Big(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\Big)\Big)$$

- $h_v^{(k)}$: embedding of node v after k message-passing steps
- $\mathcal{N}(v)$: neighbours of node v in the graph
- **AGGREGATE**: permutation-invariant function (sum, mean, max, attention)
- **UPDATE**: typically a linear layer followed by a nonlinearity
- After K steps: each node's embedding encodes information about its K-hop neighbourhood

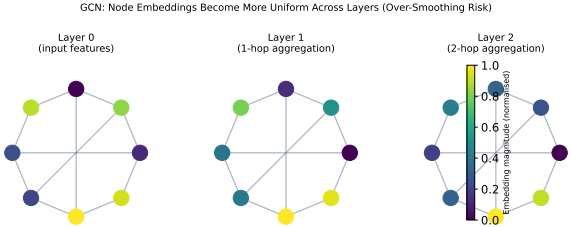


Graph Convolutional Network (GCN)

Spectral convolution simplified to a linear neighbourhood aggregation

$$H^{(k)} = \sigma\Big(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}\, H^{(k-1)}\, W^{(k)}\Big)$$

- $\tilde{A} = A + I$: adjacency matrix with self-loops added
- $\tilde{D}$: diagonal degree matrix of $\tilde{A}$; the $D^{-1/2}AD^{-1/2}$ term normalises by degree
- $W^{(k)}$: trainable weight matrix at layer k
- The normalisation prevents exploding activations for high-degree nodes
- Kipf & Welling (2017): empirically excellent on node classification benchmarks



GCN is the reference baseline; for production use GAT (attention) or GraphSAGE (inductive)

Graph Attention Networks (GAT)

Weighting neighbours by learned attention scores

$$h_v^{(k)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup \{v\}} \alpha_{vu}\, W\, h_u^{(k-1)}\right), \quad \alpha_{vu} = \frac{\exp\big(\mathrm{LeakyReLU}(a^\top [Wh_v \parallel Wh_u])\big)}{\sum_{w \in \mathcal{N}(v)} \exp(\cdots)}$$

- α_{vu}: attention weight from node u to node v; learned end-to-end
- High-attention neighbours contribute more to the updated embedding
- Multi-head attention: run K independent attention heads, concatenate or average
- Advantage: differentiates between important and unimportant neighbours automatically
- Finance: in a transaction graph, a high-attention edge signals a structurally important connection

GAT attention weights are interpretable: they identify the most influential graph neighbours

GraphSAGE: Inductive Learning on Large Graphs

Sample and aggregate: scalable message passing for unseen nodes

- Standard GCN is transductive: requires the full graph at training time; new nodes cannot be embedded without retraining
- **GraphSAGE** (Hamilton et al. 2017): sample a fixed-size neighbourhood per node; compute embedding by aggregating sampled neighbours (mean, LSTM, or max-pool)
- Inductive: learn an aggregation function, not node-specific embeddings; apply to unseen nodes at inference by sampling their neighbourhood
- Mini-batch training: sample K hops outward from target nodes; only load those subgraphs
- Finance: new accounts added daily can be embedded immediately without graph retraining

GraphSAGE is the production-scale GNN: Twitter, Pinterest, Alibaba all use it or variants

Graph-Level Tasks

Going beyond node embeddings: graph classification and regression

- **Node classification:** predict a label for each node (is this account a money mule?)
- **Edge prediction:** predict whether an edge should exist (will these two companies transact?)
- **Graph classification:** predict a label for the whole graph (is this transaction network fraudulent?)
- Graph-level tasks require a **readout** function that aggregates all node embeddings: sum, mean, max, or learned virtual node
- Hierarchical pooling (DiffPool): cluster nodes into a coarser graph; apply GNN layers at each level

Readout function choice matters: sum preserves size information; mean is size-invariant

Heterogeneous Graphs

Multiple node and edge types require specialised architectures

- A corporate ownership graph has multiple node types: companies, individuals, jurisdictions
- Edge types: owns, controls, audits, guarantees – each carries different information
- Standard GCN uses a single weight matrix: cannot distinguish edge types
- **Relational GCN (R-GCN):** separate weight matrix per relation type; $h_v^{(k)} = \sigma\big(\sum_r \sum_{u \in \mathcal{N}_r(v)} W_r h_u^{(k-1)} + W_0 h_v^{(k-1)}\big)$
- **HAN / HGT:** use type-specific attention; HGT handles dynamic heterogeneous graphs
- Finance: beneficial ownership graphs, ESG supply-chain graphs, multi-market order books

R-GCN is the standard baseline for heterogeneous graphs; HGT is state-of-the-art

Expressive Power: The Weisfeiler-Leman Limit

Message passing GNNs cannot distinguish all graphs

- The 1-Weisfeiler-Leman (1-WL) graph isomorphism test: iteratively recolour nodes by their neighbourhood's colour multiset
- Xu et al. (2019): any message-passing GNN is at most as powerful as 1-WL
- Two distinct graphs that 1-WL cannot distinguish will get the same GNN embedding
- Counter-example: regular graphs of different sizes; cycle graphs of different lengths
- Higher-order GNNs (k-WL) are more expressive but $O(n^k)$ in complexity

GNN expressiveness is an active research area; for most financial tasks 1-WL power is sufficient

Scalability Challenges

Full-graph training is impractical for large graphs

- Loading the full adjacency matrix into GPU memory: infeasible for graphs with millions of nodes
- **GraphSAGE** (Hamilton et al. 2017): sample a fixed-size neighbourhood at each layer; inductive – generalises to unseen nodes at test time
- **Cluster-GCN**: partition the graph into clusters; train on one cluster per batch; eliminates the exponential neighbour explosion
- **Graph transformers** (GPS, NAGphormer): replace message passing with attention; $O(n^2)$ but avoids multi-hop neighbourhood sampling
- Finance: payment networks have 10^8 nodes; GraphSAGE is the practical starting point

GraphSAGE is the production standard: mini-batch training, inductive, linear in nodes

Temporal GNNs: Dynamic Transaction Graphs

Handling graphs where edges and nodes appear over time

- Transaction graphs are inherently temporal: each wire transfer has a timestamp
- Static GNNs ignore time ordering: a transaction from 3 years ago is treated equally with today's
- **Temporal Graph Networks (TGN, Rossi et al. 2020):** maintain a memory state per node; update memory when new interactions arrive; use temporal attention for aggregation
- **TGAT:** extends GAT with time-encoding features that decay with elapsed time
- Finance: recent transactions should have higher weight; TGN naturally captures velocity attacks where fraud patterns emerge over a short burst window

TGN is the state-of-the-art for temporal graphs: outperforms static GNNs on fraud detection

Limitations of GNNs

Strong inductive bias but not universally superior

- GNNs can over-smooth: after many layers, all node embeddings become similar (mean-field collapse)
- Structural feature engineering often outperforms GNNs on sparse graphs (e.g. PageRank, triangle count, degree statistics)
- Heterogeneous graphs (multiple edge/node types) require separate parameter sets or heterogeneous GNN variants (HAN, HGT)
- Dynamic graphs: edges and nodes change over time; temporal GNNs add complexity
- Data requirements: GNNs need sufficient labelled nodes for supervision

Always baseline with node-level feature engineering before committing to a GNN pipeline

GNN vs Feature Engineering: When to Use Which

GNNs are not always better than hand-crafted graph features

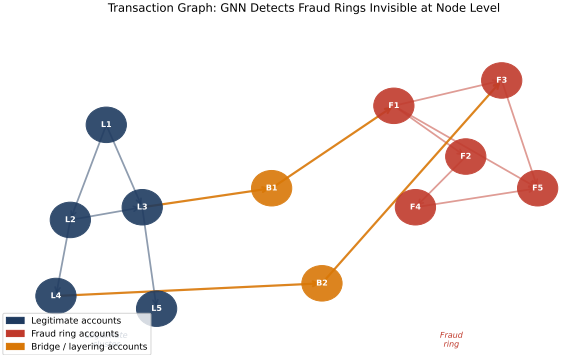
- Graph features (PageRank, clustering coefficient, degree, triangle count) are fast to compute and often match GNN accuracy on sparse, small, or well-understood graphs
- GNNs outperform when: the graph is dense, the relevant patterns are multi-hop, or there are rich node/edge features that should be combined with topology
- Practical heuristic: start with graph statistics as features in a gradient-boosted tree; upgrade to GNN only if accuracy is insufficient or multi-hop patterns are suspected
- GNNs require significantly more engineering: data pipeline, graph batching, GPU memory management
- Finance: for KYC network scoring, LightGBM + 10 graph features often beats GCN at 1/20th the cost

Feature engineering + GBM is a strong baseline; do not skip it in favour of GNNs directly

Finance Application: Transaction Graph Fraud Detection

Catching money laundering rings that look clean node by node

- Each account is a node; each wire transfer is a directed edge with amount and timestamp
- Node features: account age, historical transaction volume, geographic flag
- Edge features: transaction amount, frequency, counterparty type
- GNN task: node classification – is this account part of a money laundering ring?
- Layered aggregation: after 2 hops, each account's embedding encodes the behaviour of all accounts reachable within 2 transactions



GNNs for AML: industry standard at major banks since 2021 (J.P. Morgan, Rabobank, HSBC)

Knowledge Graphs in Finance

Structured relational knowledge enables multi-hop reasoning

- A knowledge graph (KG) encodes facts as triples: (entity, relation, entity)
- Financial KG example: (Apple, *subsidiary_of*, Apple Inc.); (Goldman Sachs, *underwrote*, BondX)
- **KG embedding models** (TransE, RotatE): learn entity and relation vectors such that $h + r \approx t$ for true triples; used for link prediction (will this company default?)
- **GNN on KGs**: combine message passing with relation-specific transformations (R-GCN)
- Finance applications: beneficial ownership tracing, conflict-of-interest detection, credit network inference, ESG supply chain analysis

Bloomberg, Refinitiv, and MSCI all maintain proprietary financial knowledge graphs

Code Pointer

GNNs in Python

- **PyTorch Geometric (PyG)**: `pip install torch_geometric`; GCN, GAT, GraphSAGE, and 50+ more; `Data` class holds node features + edge index
- **DGL (Deep Graph Library)**: alternative; strong heterogeneous graph support; `dgl.graph((src, dst))` builds the graph
- **NetworkX**: use for graph construction and feature engineering before passing to PyG/DGL
- Minimal GCN: 3 `GCNConv` layers + ReLU + softmax; train with `cross_entropy_loss` on labelled nodes; evaluate on held-out test mask
- Finance: preprocess transaction log into edge list; create node features from account aggregates

PyG is the default framework: integrates with PyTorch, extensive model zoo, active community

Summary and Open Questions

Three things to remember

- GNNs use message passing: each node iteratively aggregates neighbour embeddings; after K steps, embeddings encode K-hop neighbourhood structure
- GCN normalises by degree; GAT weights by learned attention; GraphSAGE samples fixed neighbourhoods for scalability
- Finance: GNNs detect fraud patterns that are invisible at the individual account level – money laundering rings and layering schemes are only visible in the transaction graph

Open questions:

1. How do we handle dynamic transaction graphs where edges appear and disappear in real time?
2. Can GNN embeddings satisfy GDPR right-to-explanation requirements?
3. Does graph structure supervision (link prediction) improve fraud node classification?

Further reading: Kipf & Welling (2017) GCN; Velickovic et al. (2018) GAT; Hamilton et al. (2017) GraphSAGE