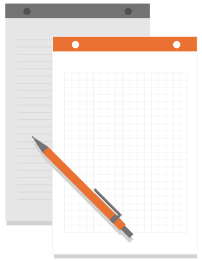
# GNNs for Recommender Systems

Cassandra Durr

# Agenda



- Types of recommender systems
- Recap of GNNs and GCNs
- A history of graph-based recommender models
    - Early collaborative filtering methods
    - Complex GNN models
    - Simplified graph-aware recommender systems
- Conclusion and references

# Recommender Systems

### Collaborative Filtering

**Goal:** Learn long-term user preferences based on historical interactions.

**Graph**: User-Item Bipartite Graph (users connect only to items)

**Use cases**: Netflix movie recommendation, Spotify music prediction

### Session-based

**Goal:** Predict the next action based on a short sequence of recent user behaviours.

**Graph**: Session graph (temporary, session-specific graphs of clicks)

**Use cases**: Online retail browsing, food delivery apps

### Heterogeneous

**Goal:** Model multiple types of entities.

**Graph**: Multiple node types, multiple edge types

**Use cases**: Social network

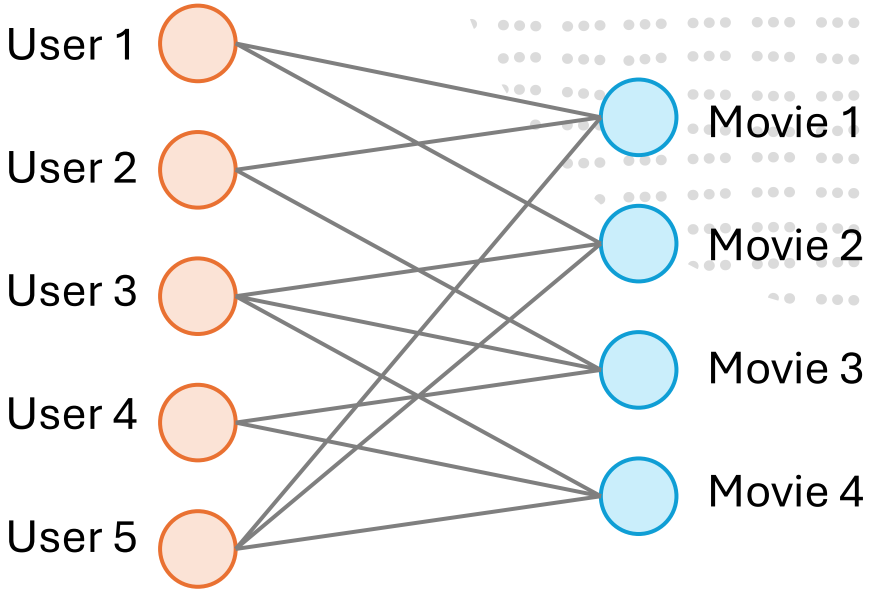
# Types of Recommender Systems

**Collaborative Filtering**

**Goal:** Learn long-term user preferences based on historical interactions.

**Graph**: User-Item Bipartite Graph (users connect only to items)

**Use cases**: Netflix movie recommendation, Spotify music prediction




**Nodes can be separated into two distinct groups.**

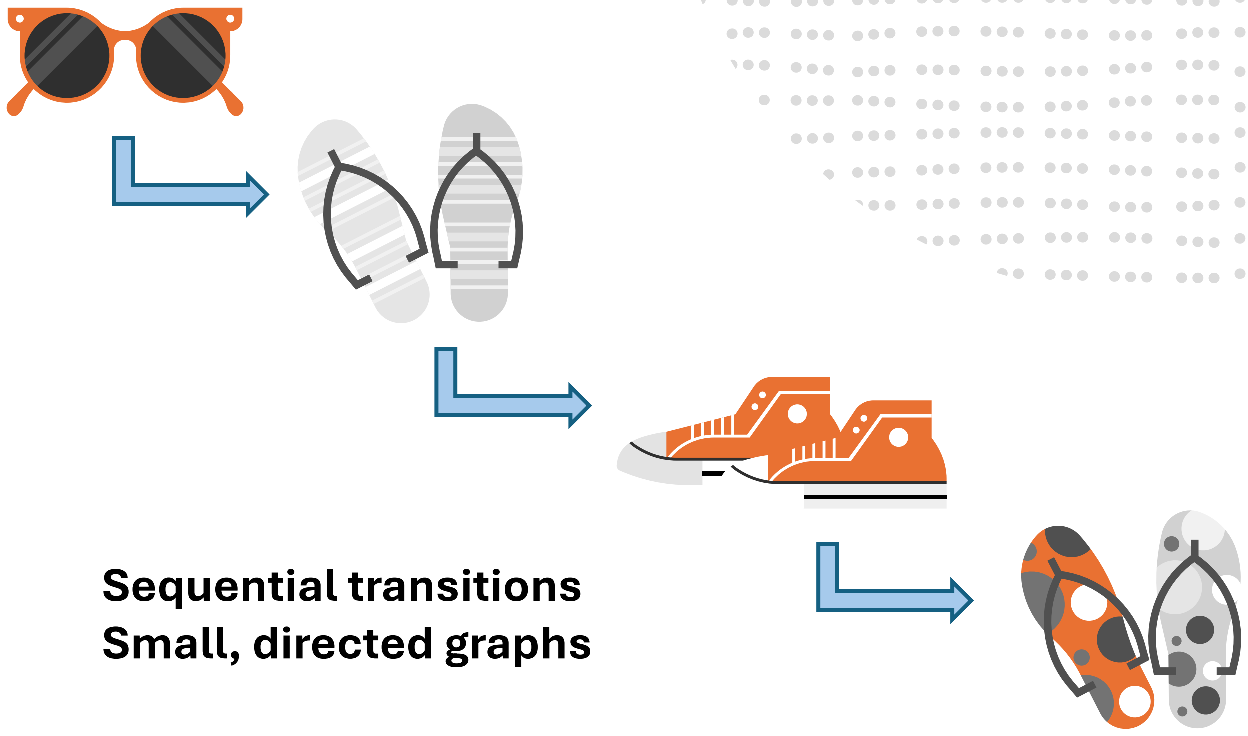
# Types of Recommender Systems

**Session-based**

**Goal:** Predict the next action based on a short sequence of recent user behaviours.

**Graph**: Session graph (temporary, session-specific graphs of clicks)

**Use cases**: Online retail browsing, food delivery apps



**Sequential transitions**
**Small, directed graphs**

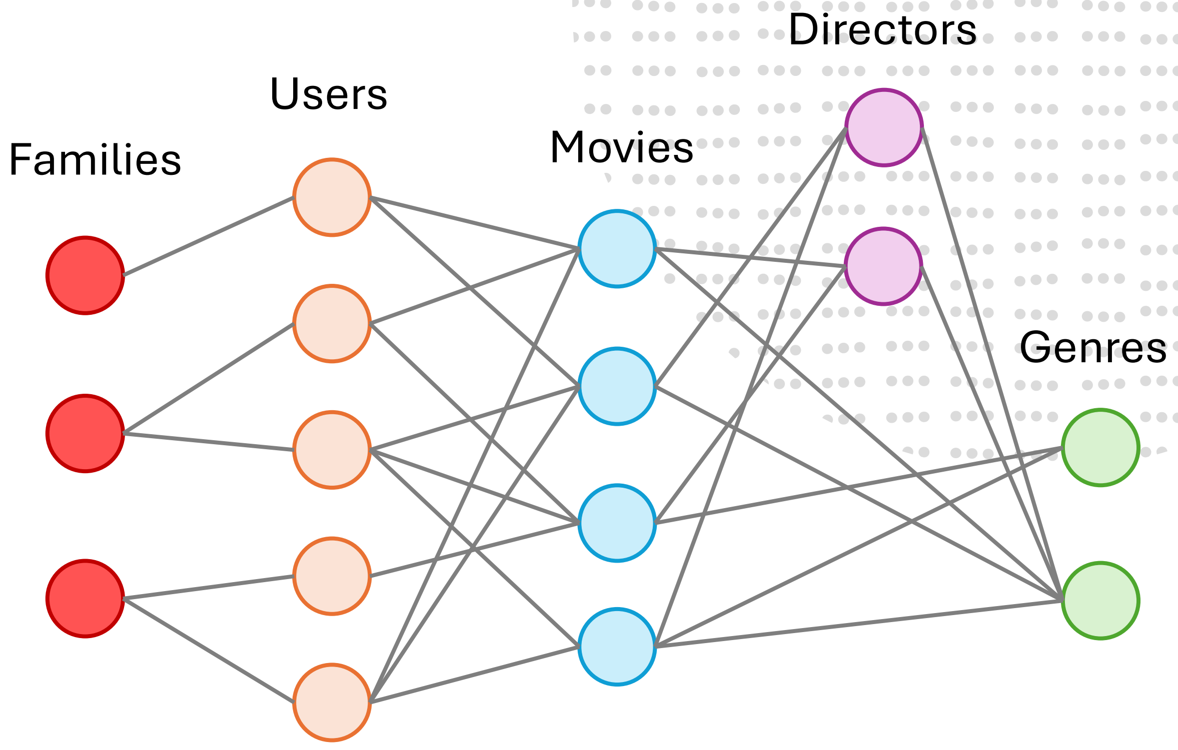
# Types of Recommender Systems

**Heterogeneous**

**Goal:** Model multiple types of entities.

**Graph**: Multiple node types, multiple edge types

**Use cases**: Social network



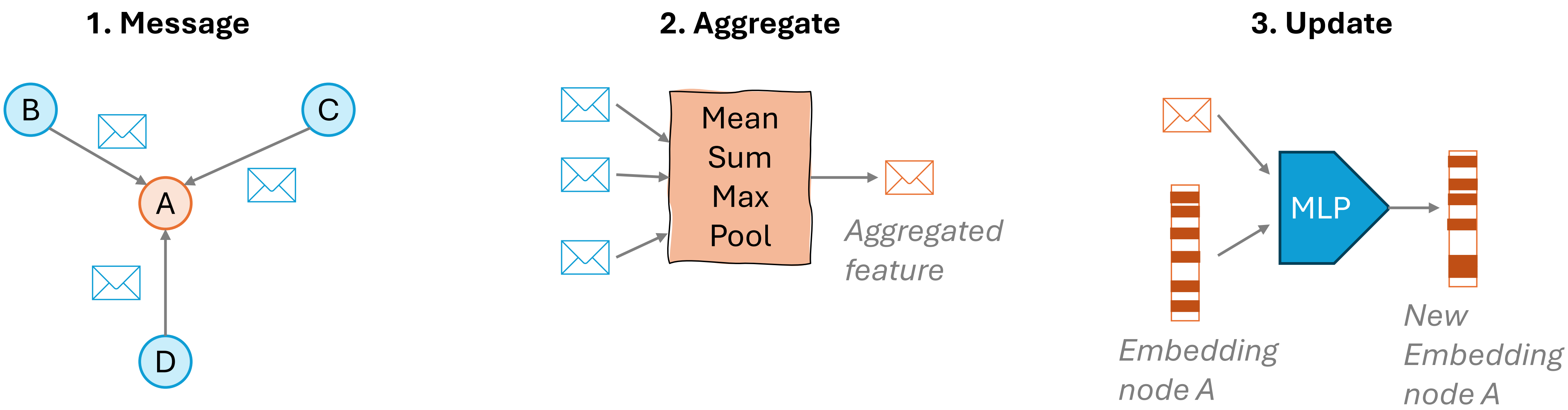

**Edges have different meanings based on node types**

# Agenda

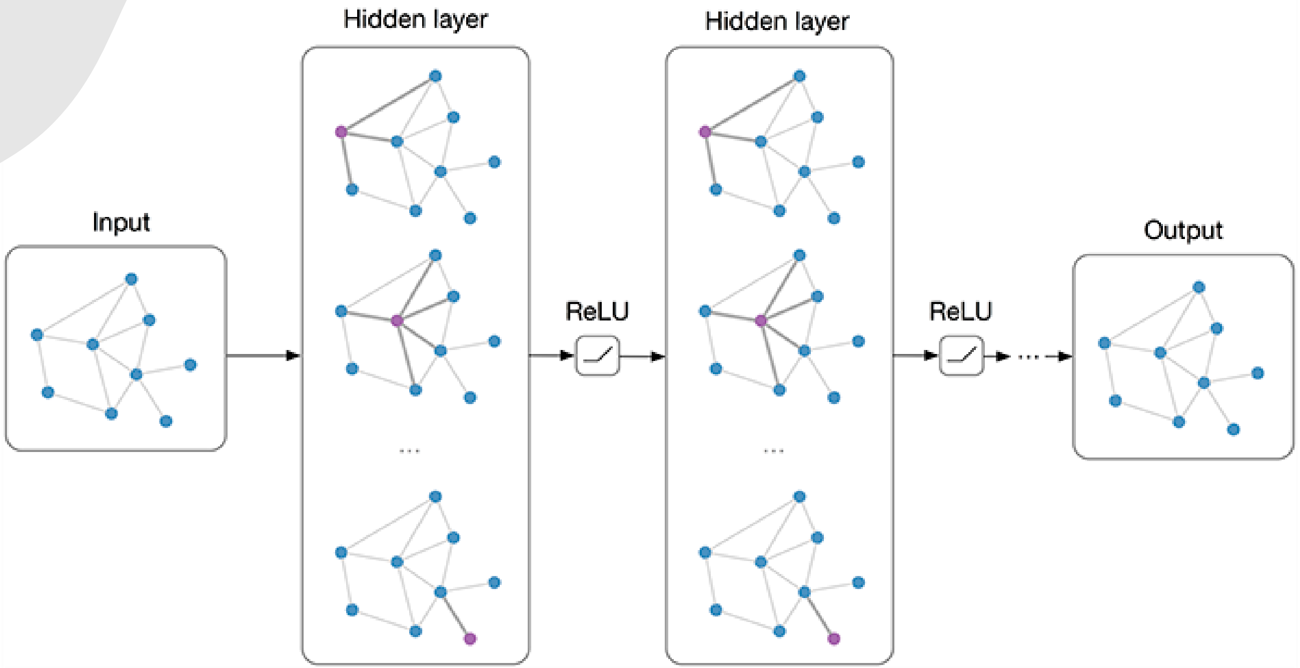
- Types of recommender systems
- **Recap of GNNs and GCNs**
- A history of graph-based recommender models
  - Early collaborative filtering methods
  - Complex GNN models
  - Simplified graph-aware recommender systems
- Conclusion and references

# Graph Neural Network (GNN)

- Neural network designed to process graphical data by learning the relationships between nodes.
- **Message passing** between nodes to learn node feature representation based on the local neighbourhood



# Graph Convolutional Network (GCN)




- Kipf & Welling (ICLR 2017)
- Type of **GNN** that performs **graph convolution** - a form of weighted averaging of neighbour features.

**Image source:** https://tkipf.github.io/graph-convolutional-networks/

# Graph Convolutional Network (GCN)

$$\boxed{H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)}$$

- Adjacency matrix with self-connections added: $\tilde{A} = A + I_n$
- Normalised adjacency matrix: $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
- Activation function: $\sigma$
- Trainable weight-matrix for layer $l$: $W^{(l)}$
- Node feature matrix for layer $l$ : $H^{(l)}$

Each node updated its representation by **aggregating information from its neighbours**.

**Without adding self-loops**, the previous representation of a node gets overwritten immediately.

# Graph Convolutional Network (GCN)

$$H^{(l+1)} = \sigma\left(\widetilde{D}^{-1/2}\tilde{A}\widetilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

- Adjacency matrix with self-connections added: $\tilde{A} = A + I_n$
- Normalised adjacency matrix: $\widetilde{D}^{-1/2}\tilde{A}\widetilde{D}^{-1/2}$, $\widetilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
- Activation function: $\sigma$
- Trainable weight-matrix for layer $l$: $W^{(l)}$
- Node feature matrix for layer $l$ : $H^{(l)}$

Without normalising the adjacency matrix with the degree matrix, high-degree nodes (nodes with lots of edges) would **overwhelm their neighbours.**

# Graph Convolutional Network (GCN)

$$\boxed{H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)}$$

- Adjacency matrix with self-connections added: $\tilde{A} = A + I_n$
- Normalised adjacency matrix: $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
- Activation function: $\sigma$
- Trainable weight-matrix for layer $l$: $W^{(l)}$ $\longrightarrow$ Linear feature transformation per layer
- Node feature matrix for layer $l$ : $H^{(l)}$ $\longrightarrow$ Message passing

# Agenda

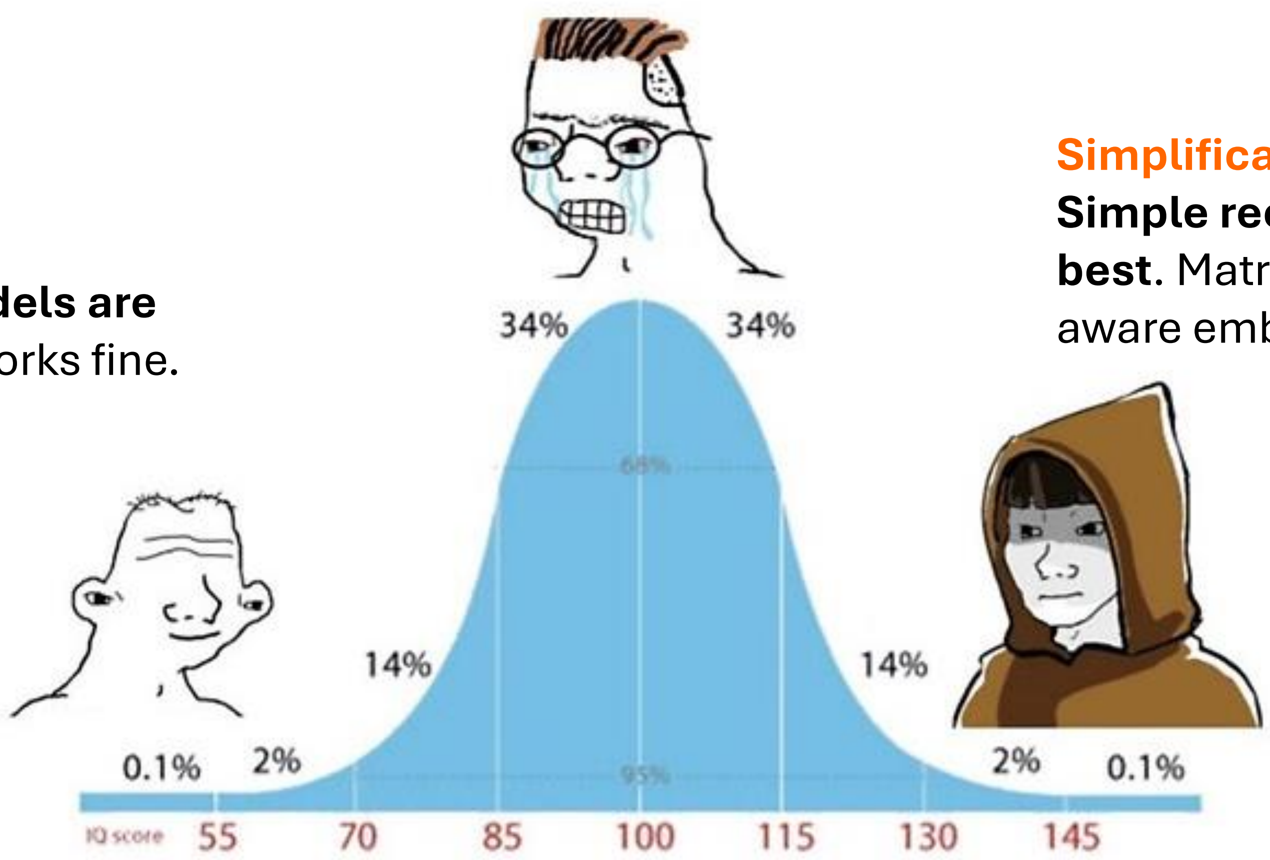
- Types of recommender systems
- Recap of GNNs and GCNs
- **A history of graph-based recommender models**
  - Early collaborative filtering methods
  - Complex GNN models
  - Simplified graph-aware recommender systems
- Conclusion and references

# Timeline of SOTA Graph-Based CF

**Early CF Era (2006 - 2015)**
**Simple recommender models are best**. Matrix factorisation works fine.

**Complex Deep Learning Era (2016 - 2019)**
We need to account for higher-order graph connections and use deep neural networks.

**Simplification Wave (2020+)**
**Simple recommender models are best**. Matrix factorisation with graph-aware embeddings.



# Early CF Era (2006 - 2015)

## Models

- Matrix factorisation
- Bayesian Personalised Ranking
- SVD ++

# Early CF Era (2006 - 2015)

***Matrix Factorisation***

User embedding matrix

$$R \approx UV^T$$

Rating matrix
(user-item)

Item embedding matrix

- **Prediction:** $\hat{r}_{ui} = \langle \boldsymbol{u}_u, \boldsymbol{v}_i \rangle$
- **Loss:** $\min \sum_{u,i} \underbrace{(r_{ui} - \hat{r}_{ui})^2}_{\text{Squared error}} + \underbrace{\lambda_1 \|\boldsymbol{u}_u\|_2 + \lambda_2 \|\boldsymbol{v}_i\|_2}_{\text{Regularisation}}$
- Requires **explicitly** provided feedback ★★☆

# Early CF Era (2006 - 2015)

*Bayesian Personalised Ranking*

- **One-class collaborative filtering system**: Does not require explicit feedback – only positive feedback.
- Positive signals assumed known, obtain negative signal by **randomly sampling** user-item pairs with no data.
- **Goal**: Rank observed positive signals higher than sampled unobserved pairs.
- **Loss**:

$$\boxed{\min - \sum_{u,i,j \in D} \ln\left(\sigma\left(\hat{x}_{ui} - \hat{x}_{uj}\right)\right) + \lambda \|\Theta\|_2}$$

Regularisation

Predicted score where the user interacted with the item

Predicted score where the user **did not** interact with the item

# Early CF Era (2006 - 2015)

***SVD++***

- **Extends matrix factorisation** model to allow for **implicit** feedback.

$$\hat{r}_{ui} = \mu + \underbrace{b_i + b_u}_{\text{Bias terms}} + \left\langle \boldsymbol{p}_u + \frac{\sum_{j \in N(u)} \boldsymbol{y}_j}{\sqrt{|N(u)|}}, \quad \boldsymbol{q}_i \right\rangle$$

Global average rating

User embedding (captures user's explicit preferences)

Item embedding

# Early CF Era (2006 - 2015)

***SVD++***

- **Extends matrix factorisation** model to allow for **implicit** feedback.

$$\hat{r}_{ui} = \mu + b_i + b_u + \left\langle \boldsymbol{p}_u + \frac{\sum_{j \in N(u)} \boldsymbol{y}_j}{\sqrt{|N(u)|}}, \quad \boldsymbol{q}_i \right\rangle$$

- $N(u)$: Set of items that user $u$ has interacted with (implicit feedback set)
- $\boldsymbol{y}_j$: Latent vector associated with **implicit item** $j$, used to enrich user $u$'s profile

# Early CF Era (2006 - 2015)

*SVD++*

- **Extends matrix factorisation** model to allow for **implicit** feedback.

$$\hat{r}_{ui} = \mu + b_i + b_u + \left\langle \underbrace{\boldsymbol{p}_u + \frac{\sum_{j \in N(u)} \boldsymbol{y}_j}{\sqrt{|N(u)|}}}_{\text{User profile}}, \quad \boldsymbol{q}_i \right\rangle$$

- User profile incorporates implicit feedback
- Why is this useful? Strong recommendations can be given even if a user has not explicitly provided ratings.

# Limitations of early CF

| Limitation | Impact |
| --- | --- |
| ❌ No **structure** or **graph reasoning** | Can't model transitive preferences (A likes B, B likes C, so maybe A likes C) |
| ❌ No **high-order connectivity** | Doesn't capture similarity through neighbours |
| ❌ Poor generalisation on **sparse** users/items | Needs many interactions to learn reliable embeddings |

# Complex Deep Learning Era (2016 - 2019)

*Developments in GNNs*

- **GCN (Kipf & Welling, 2017):** initially introduced for the purpose of semi-supervised classification. Later, applied to recommender systems.
- **GraphSAGE (Hamilton et al., 2017):** introduced as a model to learn high-quality node embeddings for downstream tasks.

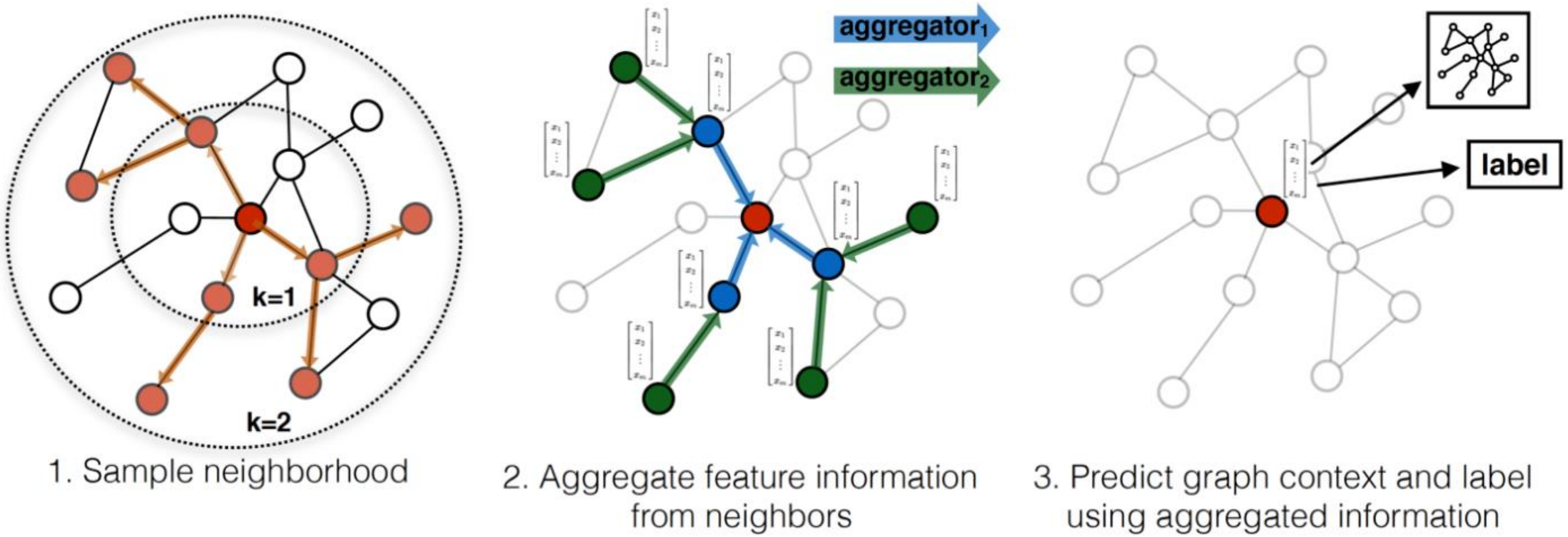



Image source: https://snap.stanford.edu/graphsage/

# Complex Deep Learning Era (2016 - 2019)

## Deep, Graph-Powered Recommender Models

- **Graph Convolutional Matrix Completion (GCMC) (Berg, Kipf & Welling, 2017):** One of the first direct applications of GCNs to CF. Uses an autoencoder-style setup.

- **Neural Graph Collaborative Filtering (NGCF) (Wang et al., 2019):** Customises GCNs for collaborative filtering.

- **PinSage (Ying et al., 2018):** first practical, large-scale GNN recommendation system.  Combines logic from GCNs and GraphSAGE. Already a simplification of GCNs.

# Complex Deep Learning Era (2016 - 2019)

*Graph Convolutional Matrix Completion (GCMC) (Berg, Kipf & Welling, 2017)*

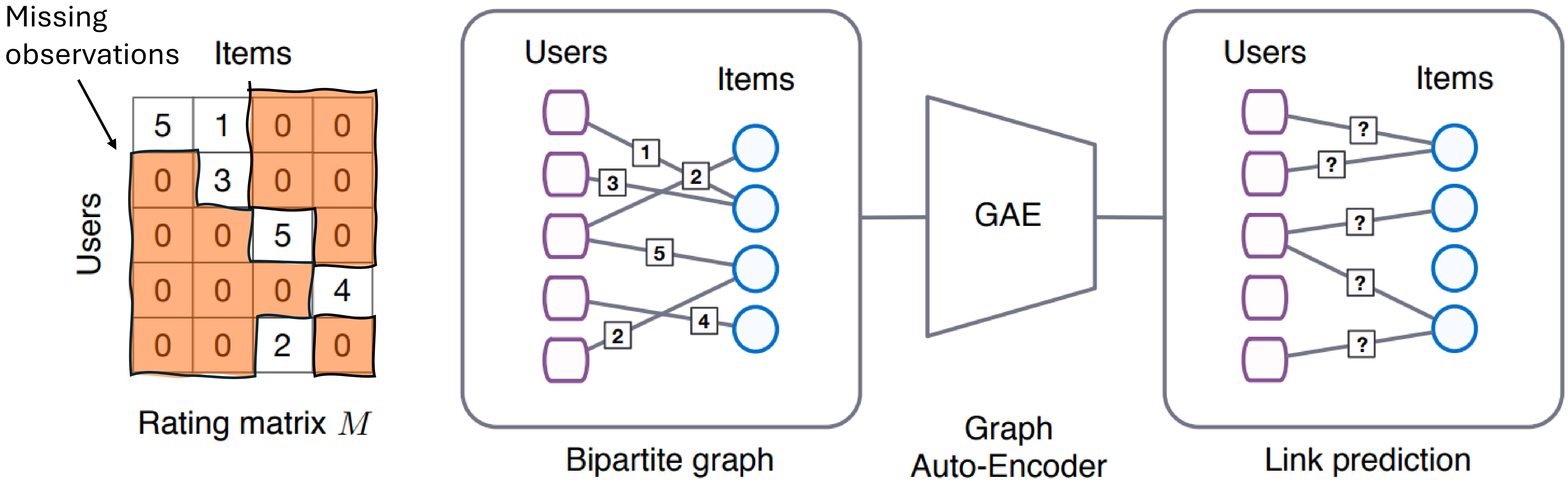
- **GCN** paper introduced at ICLR 2017 by Kipf & Welling.
- GCMC = first direct application of GCNs to collaborative filtering problem.
- **Key novelty:**
  Applying GCNs in an **autoencoder setup** for collaborative filtering, where a shared GCN encoder learns representations for users/items from interaction graphs, followed by decoding to predict ratings.

# Complex Deep Learning Era (2016 - 2019)

*Graph Convolutional Matrix Completion (GCMC) (Berg, Kipf & Welling, 2017)*

The matrix completion task can be cast as a link prediction problem.

Observed edges are input.



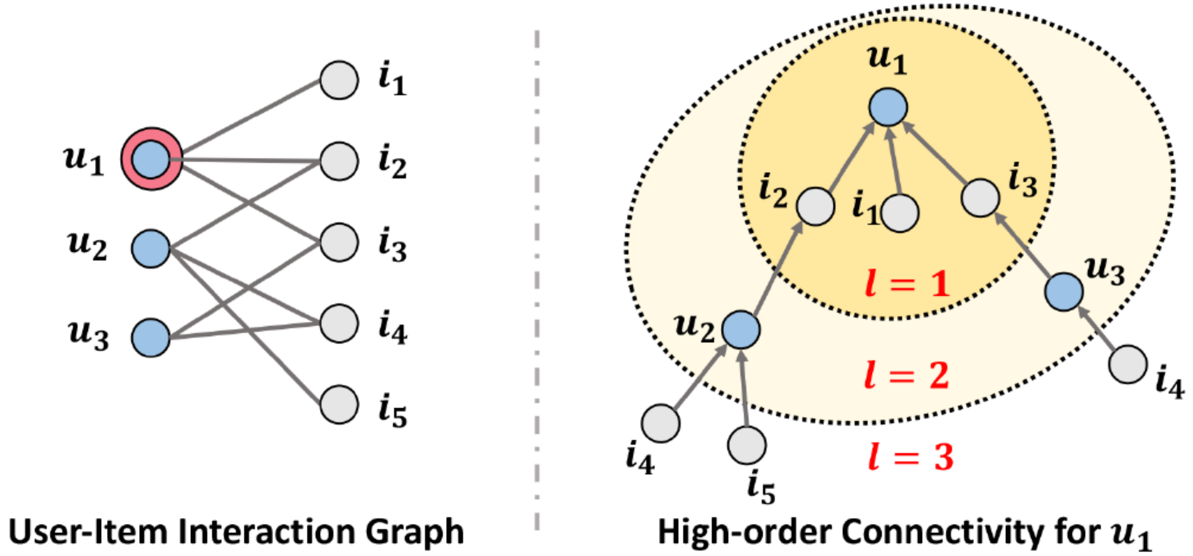

GCN encoder and MLP decoder

Image source: Berg, R. V. D., Kipf, T. N., & Welling, M. (2017). Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*.

# Complex Deep Learning Era (2016 - 2019)

***Neural Graph Collaborative Filtering (NGCF) (Wang et al., 2019)***

Multi-hop collaborative signal modelling




- **Goal: Inject collaborative signal** directly into user/item embeddings.
- **Achieves goal by** leveraging **high-order connectivity** in the **user-item interaction graph.**
- Stack multiple graph convolution layers to capture higher-order connections.

Image source: Wang, X., He, X., Wang, M., Feng, F., & Chua, T. S. (2019, July). Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval* (pp. 165-174).

# Complex Deep Learning Era (2016 - 2019)

*Neural Graph Collaborative Filtering (NGCF) (Wang et al., 2019)*

NGCF modifies **message passing** between nodes.

$$\mathbf{m}_{u \leftarrow i} = \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} \left( \underbrace{\boxed{\mathbf{W}_1 \mathbf{e}_i}}_{\text{Traditional message passing}} + \boxed{\mathbf{W}_2(\mathbf{e}_i \odot \mathbf{e}_u)} \right)$$

This term mixes the user and item embeddings together before sending the message. The message is personalised to that user.

**Example: LOTR**

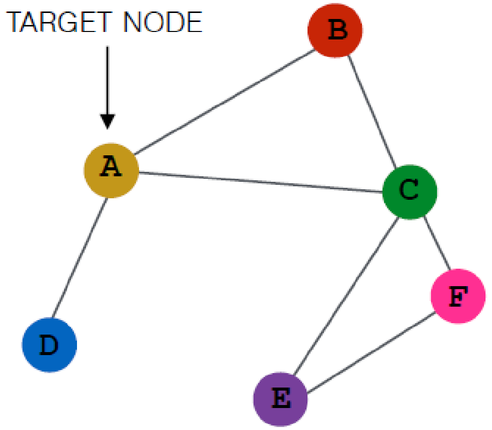
**GCN**: Alice & Ben get the same message: “you liked LOTR”

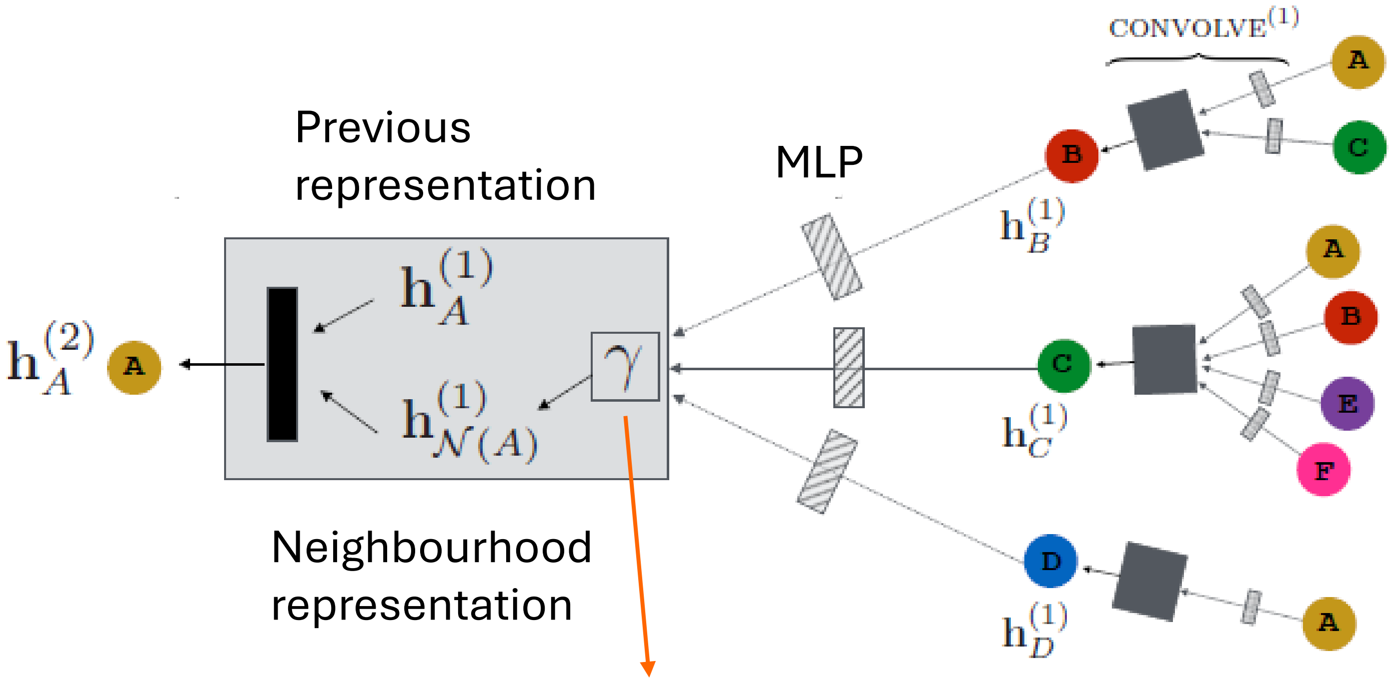
**NGCF**:

- Alice: “You liked the rich fantasy world.”
- Ben: “You liked the epic battle scenes.”

# Complex Deep Learning Era (2016 - 2019)

*PinSage (Ying et al., 2018):*




Previous representation

MLP

Neighbourhood representation

Importance pooling function

Non-linear transformation

$$\mathbf{h}_A^{(2)} \leftarrow \mathrm{ReLU}\left(\overbrace{\boldsymbol{W}\cdot \mathrm{CONCAT}\left(\mathbf{h}_A^{(1)},\mathbf{h}_{N(A)}^{(1)}\right)+\boldsymbol{w}}^{\text{Linear transformation}}\right)$$

$$\mathbf{h}_A^{(2)} \leftarrow \mathbf{h}_A^{(2)} / \left\|\mathbf{h}_A^{(2)}\right\|_2 \qquad (L_2\text{-normalisation})$$

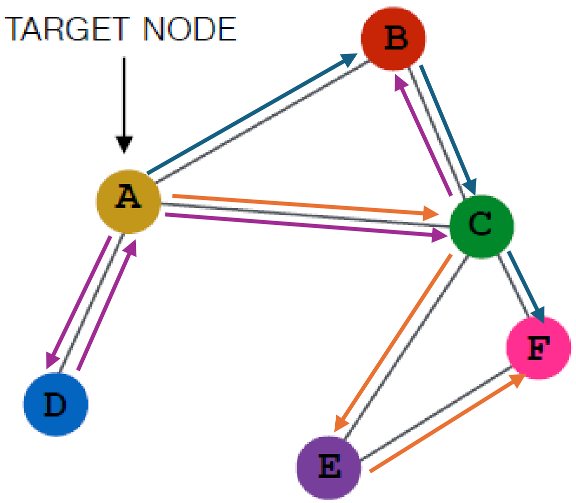
Image source: Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018, July). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 974-983).

# Complex Deep Learning Era (2016 - 2019)

*PinSage (Ying et al., 2018):*

## Efficient neighbourhood sampling

- In traditional GCNs, you compute on the **entire graph**, which is **infeasible** at large scale.
- PinSage uses **random walk-based importance sampling** to find the **top T most relevant neighbours**.




1. Start at a node
2. Perform many short random walks from the starting node
3. Count how frequently each neighbouring node is visited
4. Pick the top T most frequently visited nodes.
5. Aggregate features from these for learning.

# Complex Deep Learning Era (2016 - 2019)

*PinSage (Ying et al., 2018):*

- The **first scalable GNN recommender** system deployed at web scale (Pinterest).
- Focused on **industrial engineering challenges** like efficient neighbourhood sampling.
- Combines ideas from **GraphSAGE** (local neighbourhood sampling) and **GCN** (graph convolution updates).

**Bridge** between the **complex learning era** and the subsequent **simplification wave**.

# Simplification Wave (2020+)

## Why Simplify?

The complex GNN models of 2016 – 2019 brought rich modelling capacity, but also real-world challenges:

- × Expensive to scale (especially full-graph operations)
- × Overfitting on sparse interaction data
- × Complicated training pipelines

## New Goal: Keep the Graph Knowledge, Drop the Complexity

# Simplification Wave (2020+)

## Models

### PinSage (Ying et al., 2018)

- Bridge between the **Complex Era** and the **Simplification Wave**.

### LightGCN (He et al., 2020)

- Removes linear transformations and non-linear activation
- Just linear aggregation
- Strong performance

### UltraGCN (Mao et al., 2021)

- Removes GCN layers entirely - no message passing, back to basic embeddings
- Pure embedding model with graph regularisation
- Nearly as fast as matrix factorisation

# Simplification Wave (2020+)

*LightGCN (He et al., 2020)*

- Modifies NGCF by stripping away many features:
  - Non-linear activation
  - Self-loops
  - Personalisation in message passing
- Achieves 16% performance boost over NGCF with faster convergence

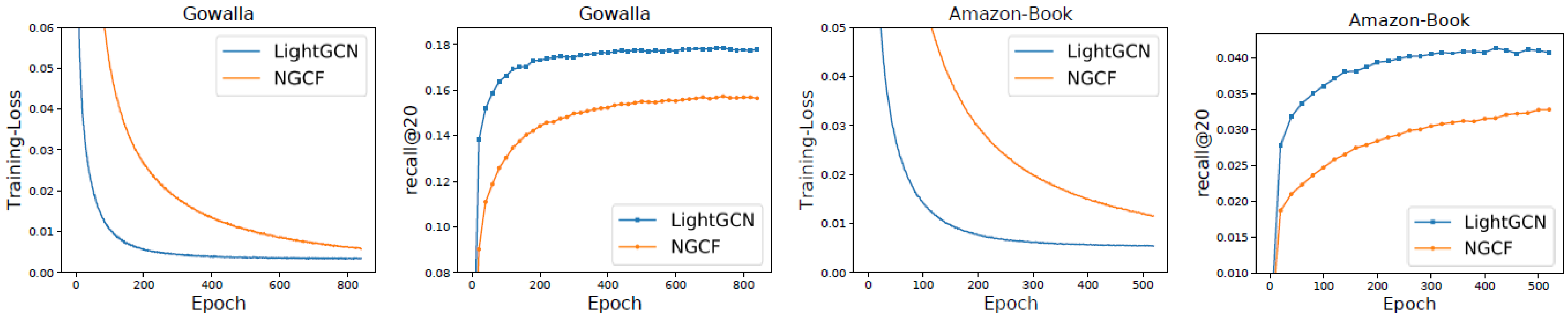



Image source: He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020, July). LightGCN: Simplifying and powering graph convolution network for recommendation. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval (pp. 639-648).

# Simplification Wave (2020+)

*LightGCN (He et al., 2020)*

**NGCF embedding update (layer k+1)**

$$\mathbf{e}_u^{(k+1)} = \sigma\left(\underbrace{\mathbf{W}_1\mathbf{e}_u^{(k)}}_{\text{Self-connection}} + \overbrace{\sum_{i\in N(u)} \frac{1}{\sqrt{N(i)N(u)}}\left(\mathbf{W}_1\mathbf{e}_i^{(k)} + \underbrace{\mathbf{W}_2\left(\mathbf{e}_u^{(k)} \odot \mathbf{e}_i^{(k)}\right)}_{\text{Personalisation/ interaction term}}\right)}^{\text{Aggregated messages to user u from neighbouring items}}\right)$$

- For NGCF, after $L$ rounds of propagation, you **concatenate** all learned embeddings to create the final embeddings:

$$\mathbf{e}_u = \left[\mathbf{e}_u^{(0)}, \mathbf{e}_u^{(1)}, \ldots, \mathbf{e}_u^{(L)}\right] \text{ per user}$$

$$\mathbf{e}_i = \left[\mathbf{e}_i^{(0)}, \mathbf{e}_i^{(1)}, \ldots, \mathbf{e}_i^{(L)}\right] \text{ per item.}$$

# Simplification Wave (2020+)

*LightGCN (He et al., 2020)*

**NGCF embedding update**

$$\mathbf{e}_u^{(k+1)} = \sigma\left(\mathbf{W}_1\mathbf{e}_u^{(k)} + \sum_{i\in N(u)} \frac{1}{\sqrt{N(i)N(u)}}\left(\mathbf{W}_1\mathbf{e}_i^{(k)} + \mathbf{W}_2\left(\mathbf{e}_u^{(k)} \odot \mathbf{e}_i^{(k)}\right)\right)\right)$$

**LightGCN embedding update**

$$\mathbf{e}_u^{(k+1)} = \sum_{i\in N(u)} \frac{1}{\sqrt{N(i)N(u)}}\mathbf{e}_i^{(k)}$$

- LightGCN final embeddings:

$\mathbf{e}_u = \sum_{l=0}^{L} \alpha_l \mathbf{e}_u^{(l)}$ per user

$\mathbf{e}_i = \sum_{l=0}^{L} \alpha_l \mathbf{e}_i^{(l)}$ per item

# Simplification Wave (2020+)

*LightGCN (He et al., 2020)*

**Why does this work?**

- Learning $W_1$ and $W_2$ and applying a non-linear activation is useful in semi-supervised node classification (original use of GCN architecture) but not CF.
- By dropping self-loops, embeddings at different layers stay distinct. Summing those layers preserves diversity, preventing all nodes collapsing to the same mean (“over-smoothing”).
- Fewer parameters leads to a simpler loss landscape. Training converges faster and with less risk of overfitting.

# Simplification Wave (2020+)

*LightGCN (He et al., 2020)*

## Is this still a GCN?

**GCN**

$$H^{(l+1)} = \sigma\left(\widetilde{D}^{-1/2}\tilde{A}\widetilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

- $\tilde{A} = A + I$
- $\widetilde{D}$ is the degree matrix of $\tilde{A}$

**LightGCN**

$$E^{(l+1)} = D^{-1/2}AD^{-1/2}E^{(l)}$$

- No learnable weight matrix
- No activation function
- No self-loops
- $D$ is degree matrix of $A$

# Simplification Wave (2020+)

*UltraGCN (Mao et al., 2021)*

- Strips the LightGCN formulation → "ultra-simplified" formulation
- Pure embedding model with graph regularisation.
- Removes message passing mechanism → approximates the limit of infinite-layer graph convolutions through a constraint loss.
- As $L \to \infty$, that process converges: each node's final embedding satisfies a **fixed-point equation:**

$$\boxed{E = D^{-1/2}\, A D^{-1/2} E} \rightarrow \boxed{E \approx PE}$$

# Simplification Wave (2020+)

*UltraGCN (Mao et al., 2021)*

- Loss function: $L = L_O + \lambda L_C + \gamma L_I$
- $L_0$: Ranking binary cross-entropy loss (rank true user-item pairs higher than negative sampled pairs)
- $L_C$: User-item constraint loss
- $L_I$: Item-item constraint loss

$$\mathcal{L}_C = -\sum_{(u,i)\in N^+} \beta_{u,i} \log\left(\sigma(e_u^\top e_i)\right) - \sum_{(u,j)\in N^-} \beta_{u,j} \log\left(\sigma(-e_u^\top e_j)\right)$$

Where $L_C$ effectively minimises the difference $\|e_u - (PE)_u\|_2$ where $(PE)_u = \sum_{i\in N(u)} \beta_{u,i} e_i$ as $L_c \propto -\sum_{(u,i)\in N^+} \beta_{u,i} e_u^T e_i$

# Simplification Wave (2020+)

*UltraGCN (Mao et al., 2021)*

**Is this still a GCN? No**

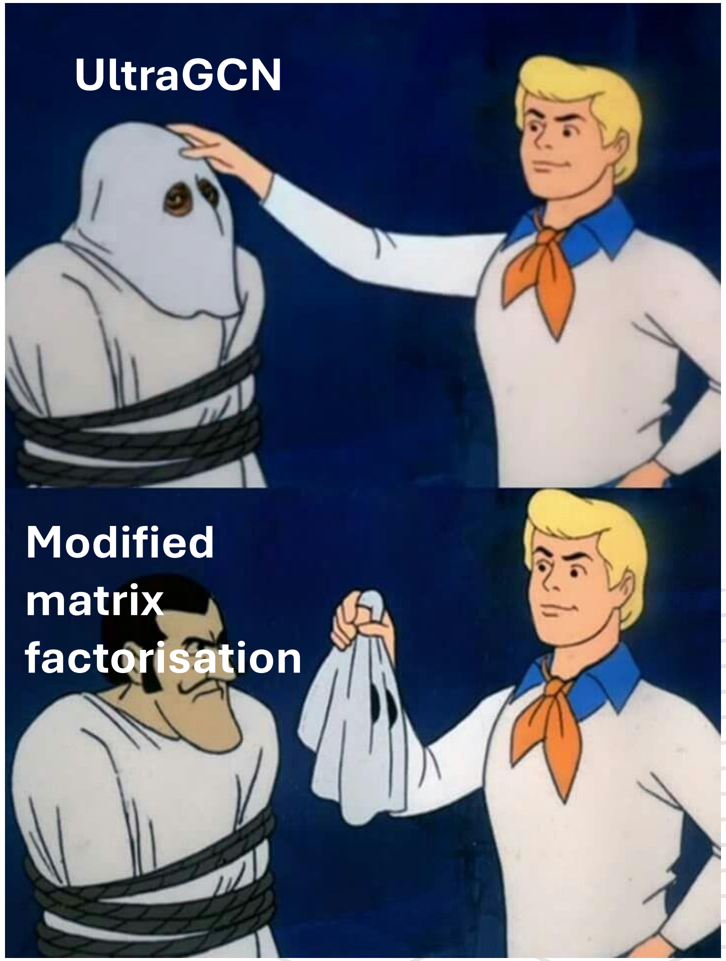
- ❌ No neighbour aggregation
- ❌ No multi-layer propagation
- ❌ No graph convolution
- ✅ Just learns embeddings and applies **graph-regularised loss**

**What is UltraGCN then?**

Latent factor model enhanced with embedded graph information.

# Isn't UltraGCN Just Old-School CF in Disguise?

- Embeddings-only model like standard MF but informed by graph structure.
- Constraint losses derived from GCN fixed-point
- Graph structure baked in via $\beta_{u,i}$ weights (user-item) and $\omega_{i,j}$ weights (item-item) weights
- Runs as fast as MF yet captures multi-hop proximity in one shot.



# Agenda

- Types of recommender systems
- Recap of GNNs and GCNs
- A history of graph-based recommender models
  - Early collaborative filtering methods
  - Complex GNN models
  - Simplified graph-aware recommender systems
- **Conclusion and references**

# Conclusion

- **Graph awareness matters** – Modelling interactions as a graph captures valuable structural signals that traditional methods miss.

# Conclusion

- **Graph awareness matters** – Modelling interactions as a graph captures valuable structural signals that traditional methods miss.
- **Simplicity wins in production** – LightGCN and UltraGCN simplify the learning process and still match or beat deeper GNNs while slashing training and inference cost.

# Conclusion

- **Graph awareness matters –** Modelling interactions as a graph captures valuable structural signals that traditional methods miss.
- **Simplicity wins in production –** LightGCN and UltraGCN simplify the learning process and still match or beat deeper GNNs while slashing training and inference cost.
- **Complexity is a cost, not a goal** – ML often adds depth and complexity, but in a production recommender system you should only pay for complexity that measurably improves your key metrics.

# References

**Bayesian Personalised Ranking**

Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2012). BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*.

**SVD++**

Koren, Y. (2008, August). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 426-434).

**GCN**

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

**NGCF**

Wang, X., He, X., Wang, M., Feng, F., & Chua, T. S. (2019, July). Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval* (pp. 165-174).

**LightGCN**

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020, July). LightGCN: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval* (pp. 639-648).

# References


**PinSage**

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018, July). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 974-983).

**GraphSAGE**

Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

**GCMC**

Berg, R. V. D., Kipf, T. N., & Welling, M. (2017). Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*.

**UltraGCN**

Mao, K., Zhu, J., Xiao, X., Lu, B., Wang, Z., & He, X. (2021, October). UltraGCN: ultra simplification of graph convolutional networks for recommendation. In *Proceedings of the 30th ACM international conference on information & knowledge management* (pp. 1253-1262).