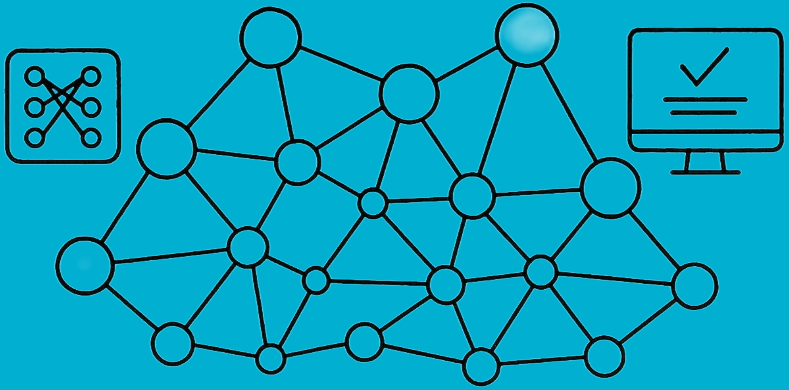
Graph Machine Learning in All Its Glory!

Shreya Khurana



PyOhio 2025

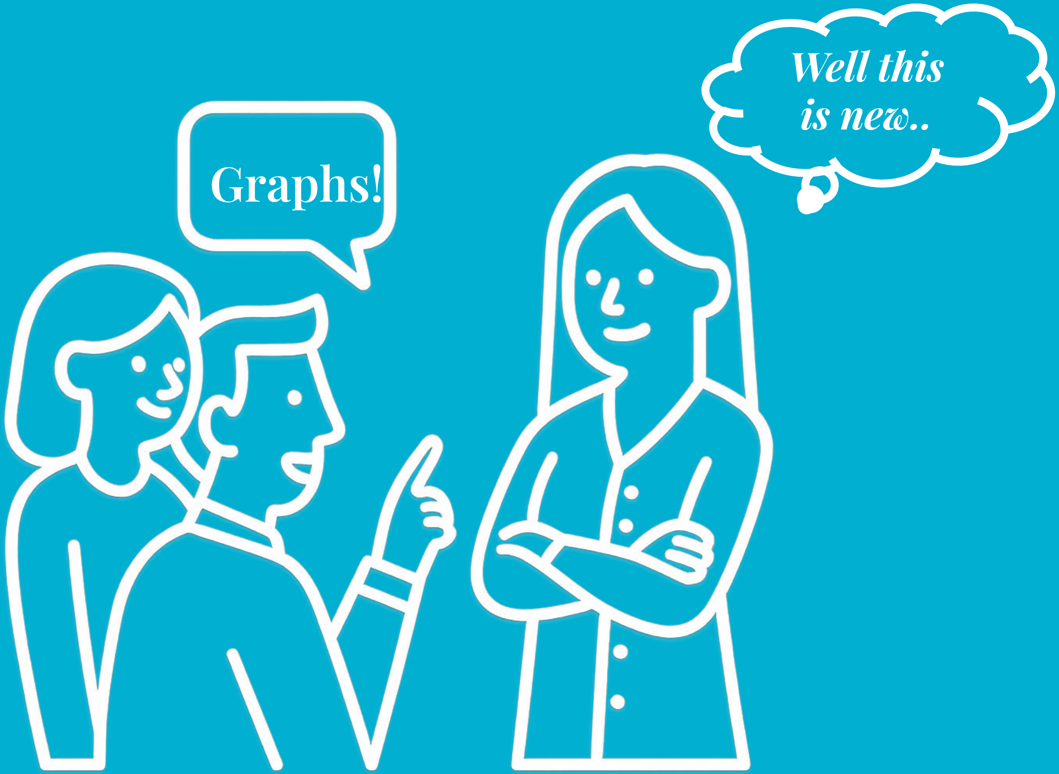
About Me

- AI Scientist at Intuit (2023 - Now)
  - Trust & Safety: Fraud detection
  - Time series forecasting and anomaly detection
- Data Scientist at GoDaddy (2019 - 2023)
  - NLP Text generation
  - Recommendation and ranking

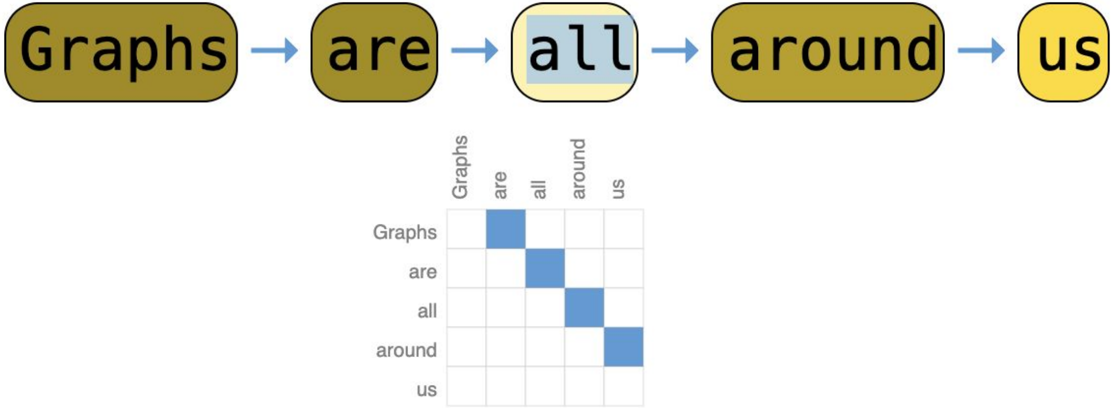
- Traditional ML + deep learning
- Models in deployment
- First in-person conference *in a while!*

- Live in San Francisco
- Interested in a lot of things

What this talk is: How I first got into graph ML

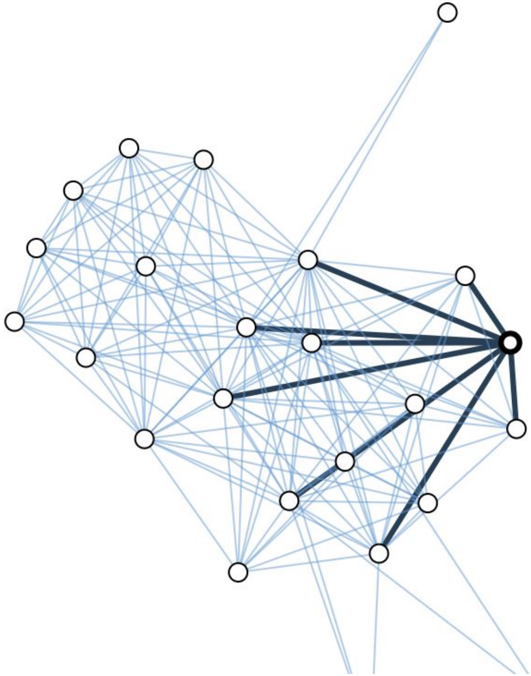


Graphs Around Us



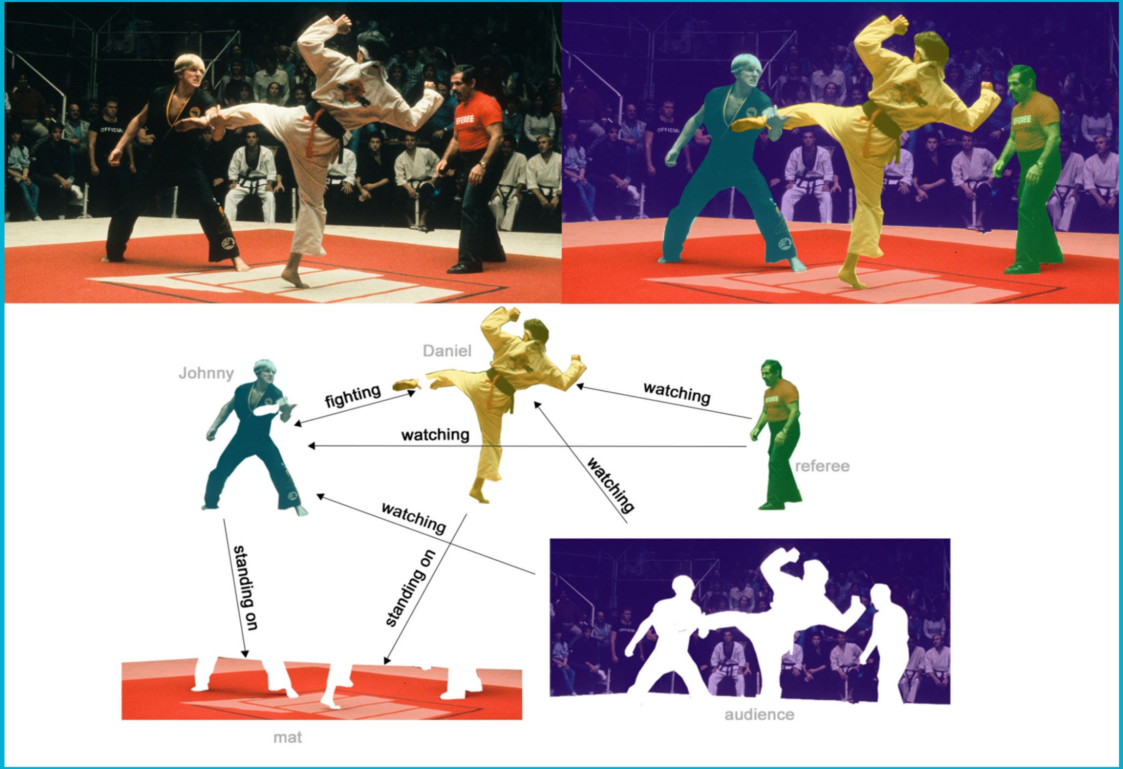
https://distill.pub/2021/gnn-intro/

Graphs Around Us



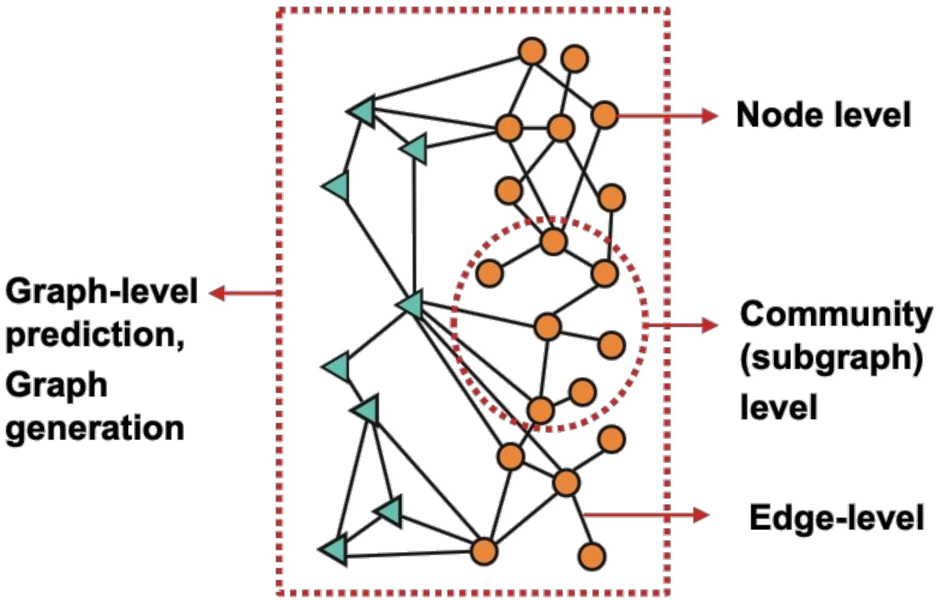
https://distill.pub/2021/gnn-intro/

Graphs Around Us



https://distill.pub/2021/gnn-intro/

How do we formulate an ML problem on graphs?



Node classification

Graph classification

Link prediction

Applications: Node Classification

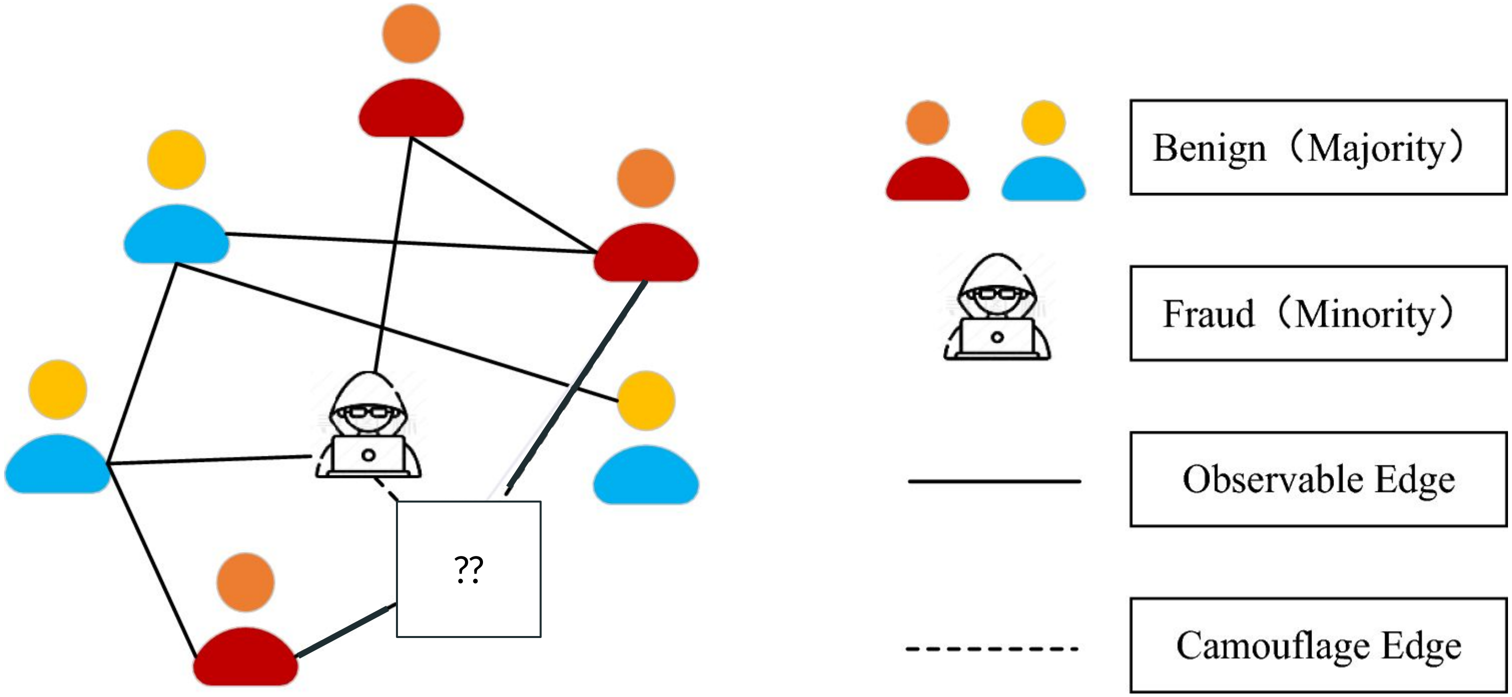


Image credits: Wen, J., Tang, X. & Lu, J. An imbalanced learning method based on graph tran-smote for fraud detection. *Sci Rep* 14, 16560 (2024). https://doi.org/10.1038/s41598-024-67550-4

Applications: Link Prediction

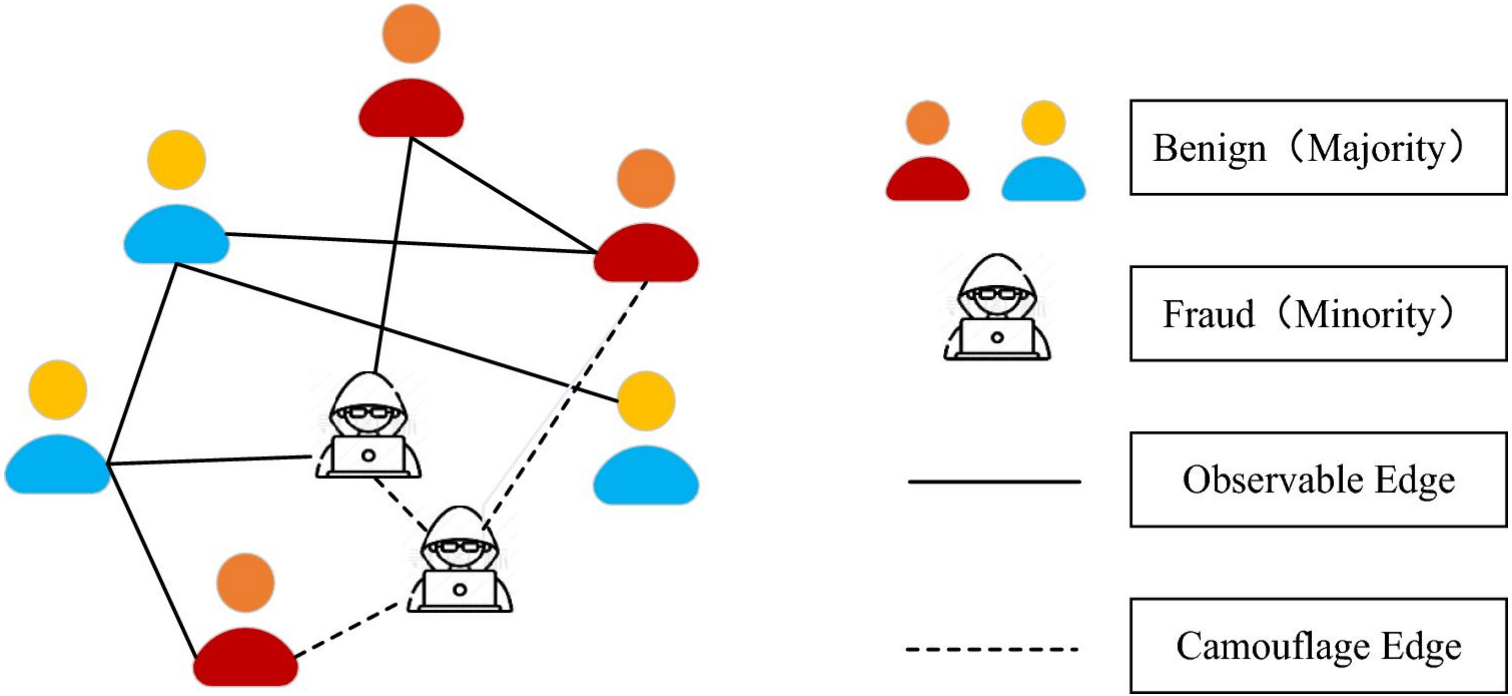


Image credits: Wen, J., Tang, X. & Lu, J. An imbalanced learning method based on graph tran-smote for fraud detection. *Sci Rep* 14, 16560 (2024). https://doi.org/10.1038/s41598-024-67550-4

Graph Neural Networks

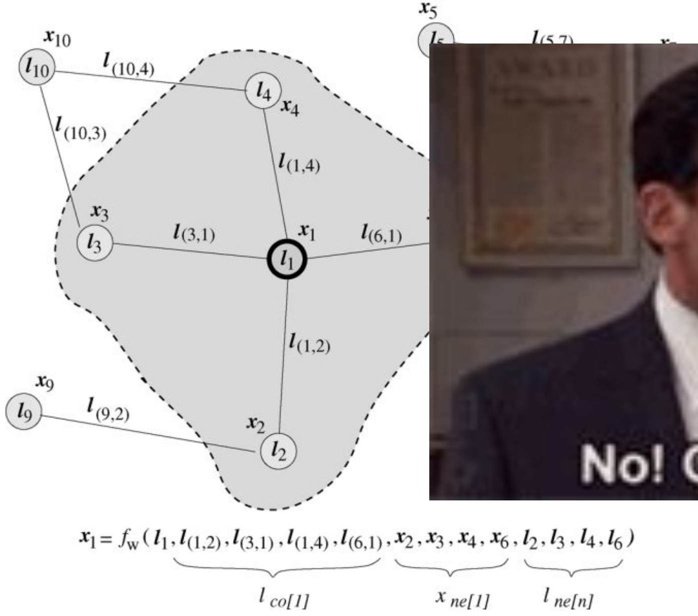
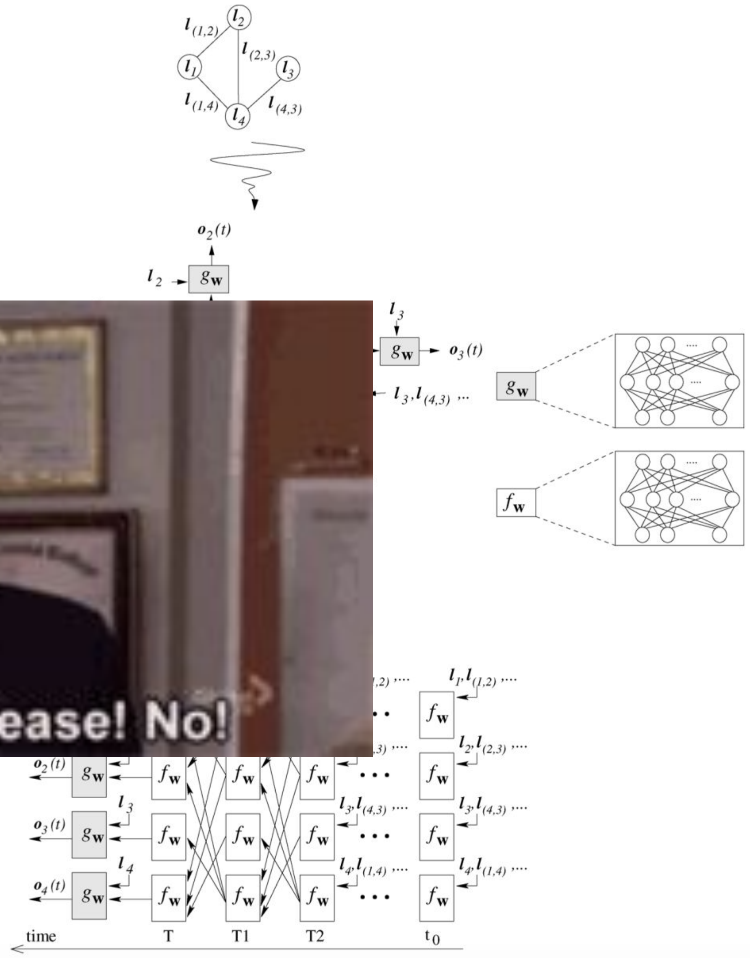


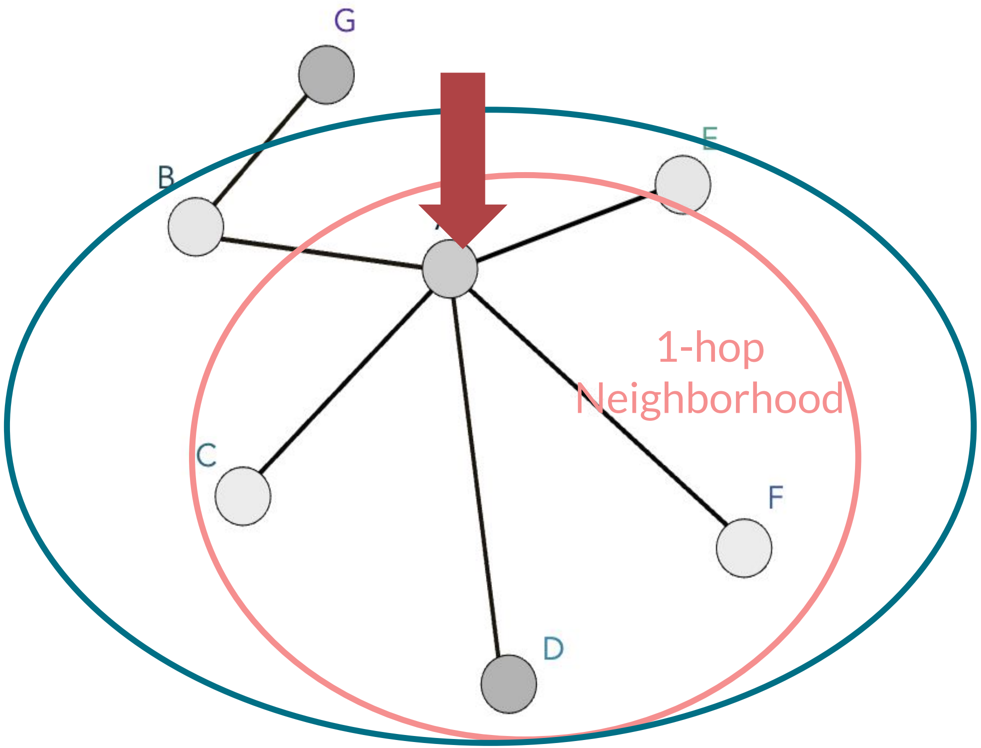
Fig. 2. Graph and the neighborhood of a node. The state x_1 of the node 1 depends on the information contained in its neighborhood.



Some Amazing References

1. A Gentle Introduction to Graph Neural Networks
2. Understanding Convolutions on Graphs
3. Graph Convolutional Networks | Thomas Kipf | Google DeepMind
4. The Graph Neural Network Model | IEEE Journals & Magazine
5. [1706.02216] Inductive Representation Learning on Large Graphs

Building Intuition: K-Hop Neighborhoods



Building Intuition: Message Passing

Connected nodes share similarities

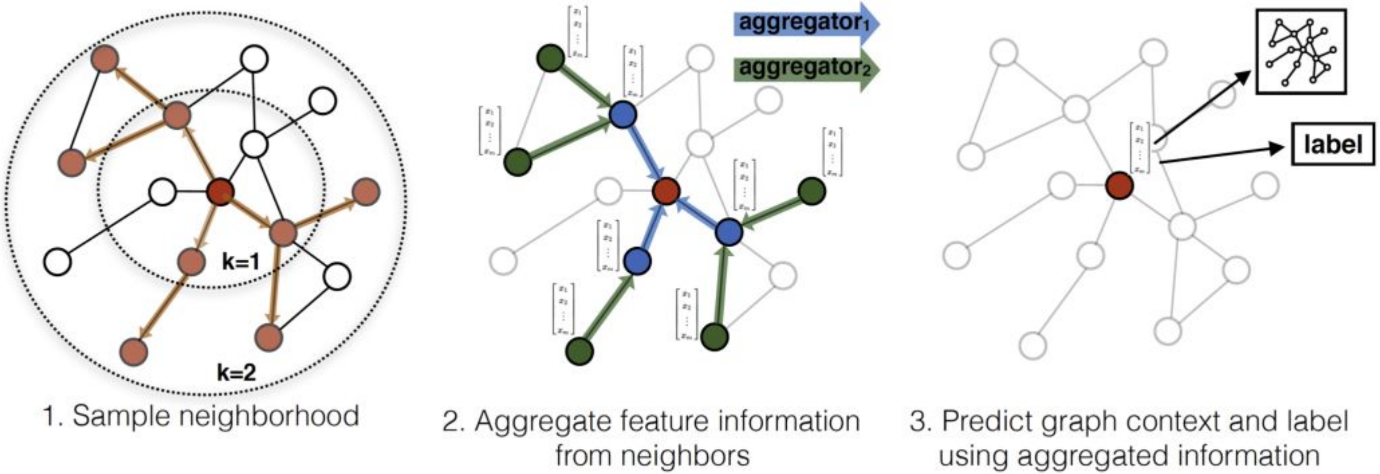
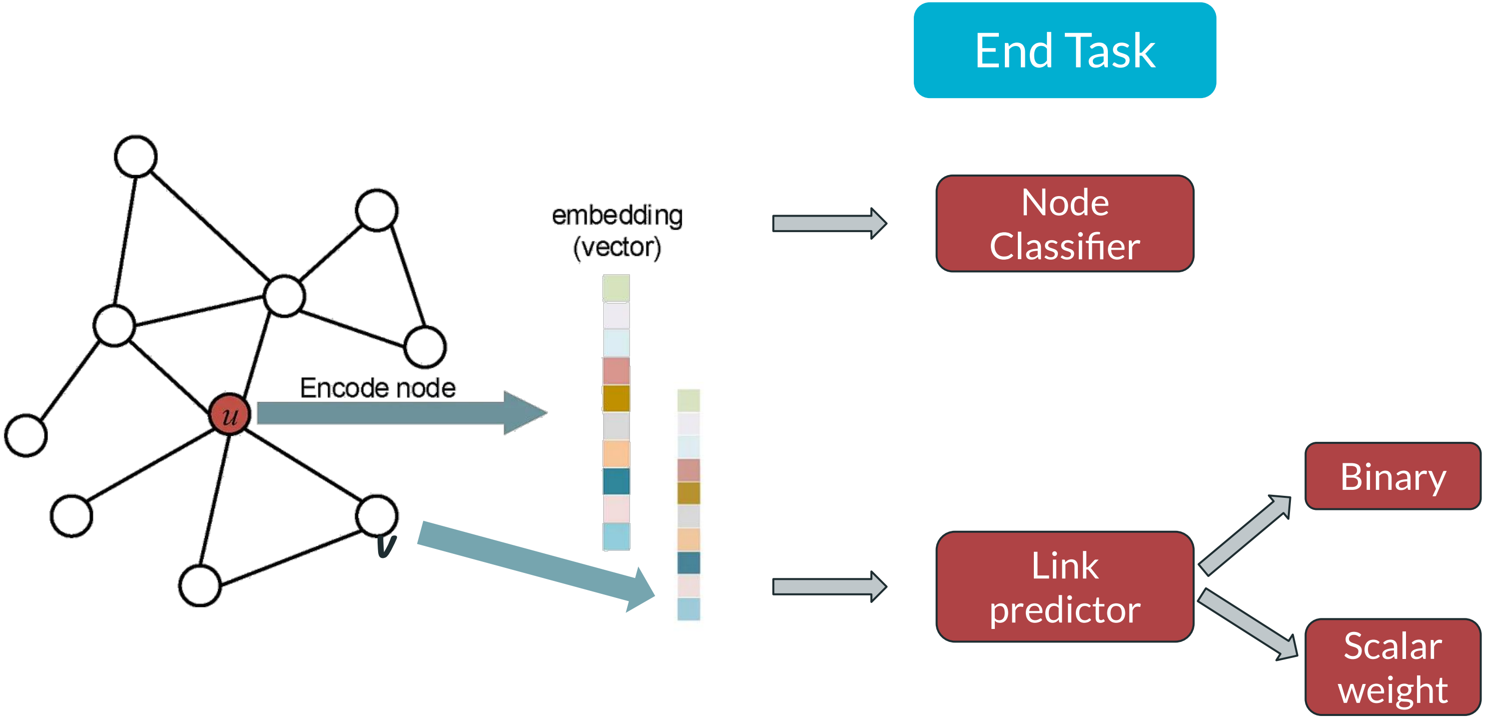


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

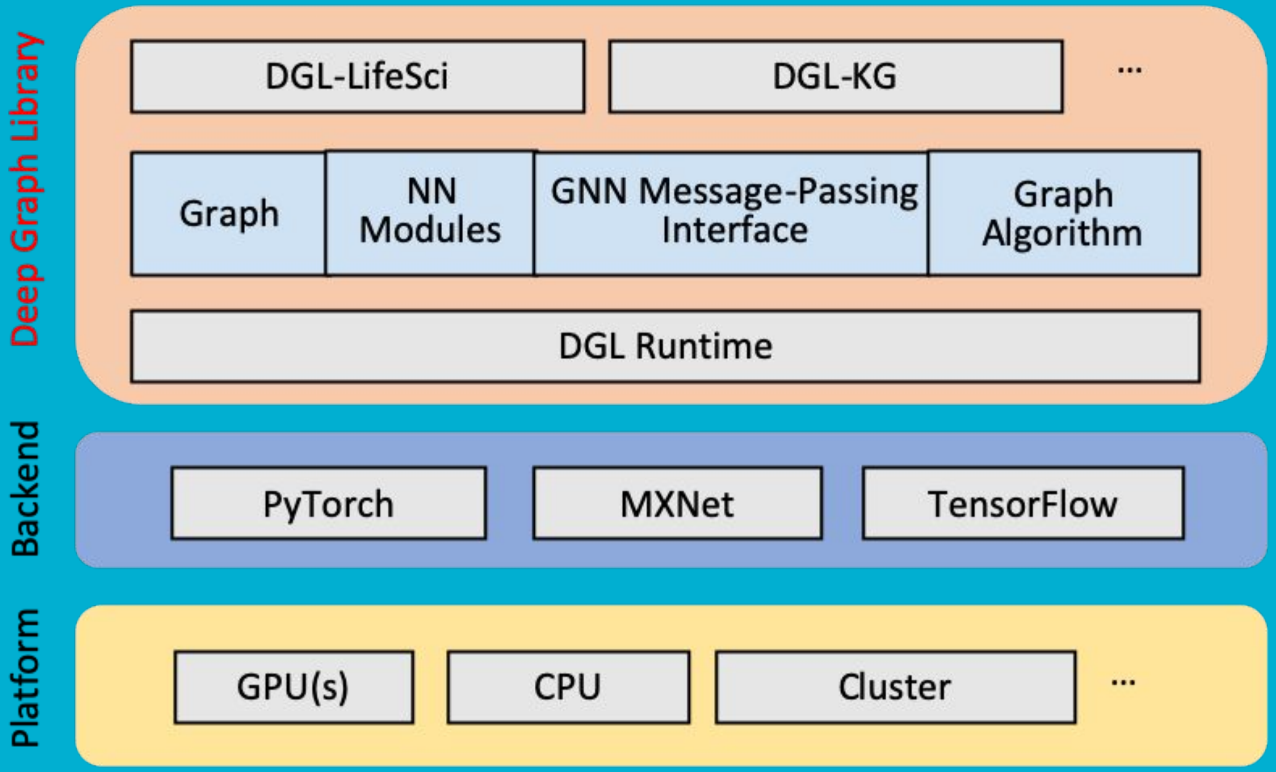
Hamilton., et al. (2017). Inductive Representation Learning on Large Graphs. arXiv:1706.02216

Building Intuition: Embeddings



Show Me How It's Done:
Movie Recommendations

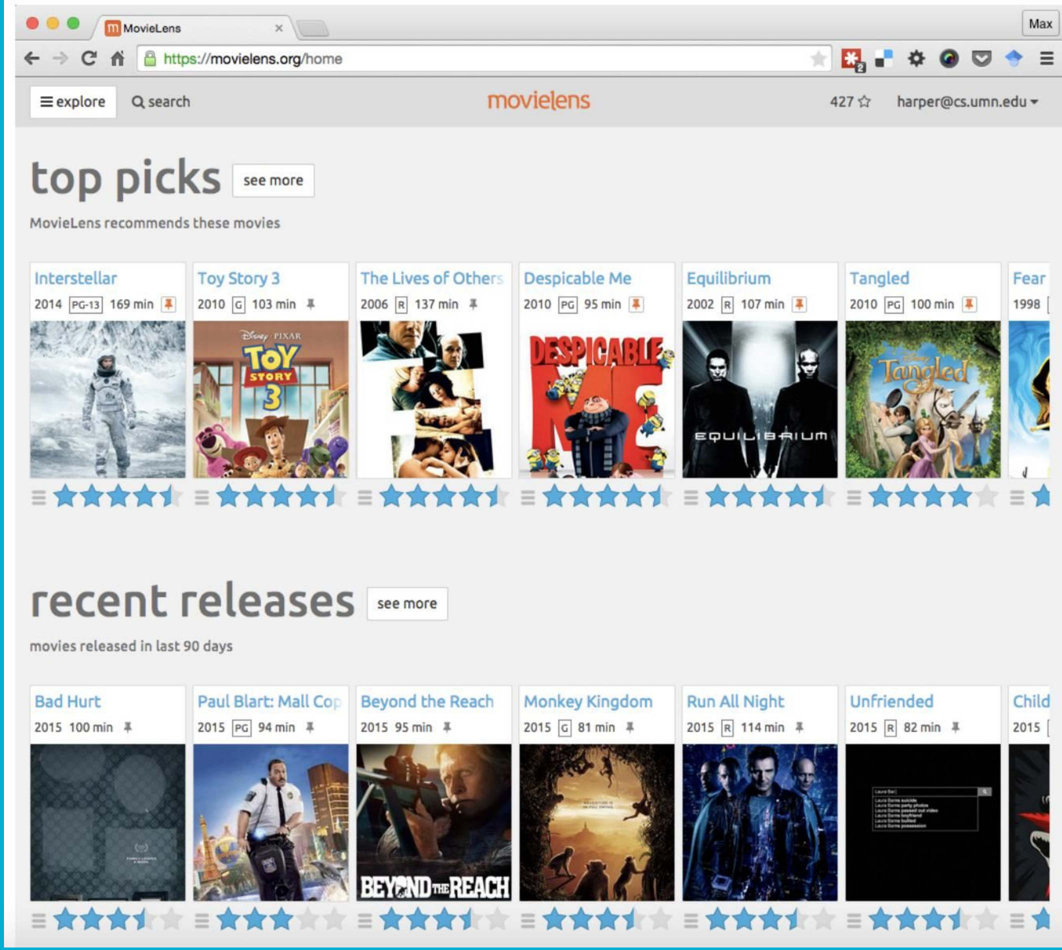
Deep Graph Library



Dataset

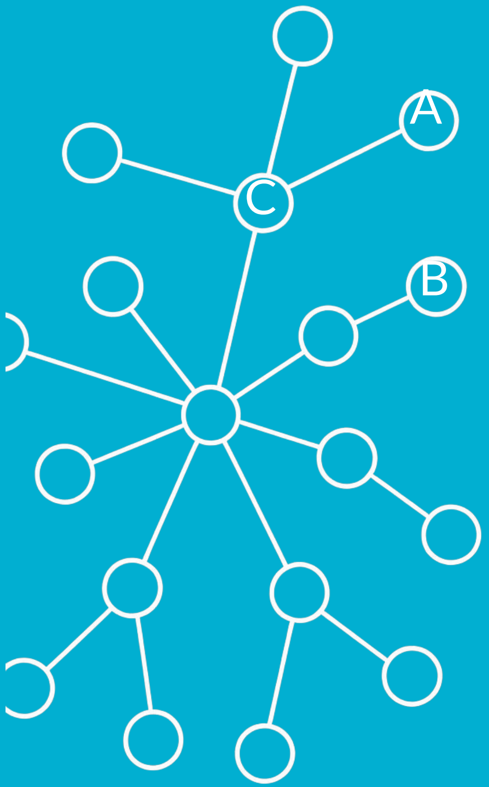
- MovieLens 100K
  - Number of users: 943
  - Number of movies: 1682
  - Number of ratings: 100000

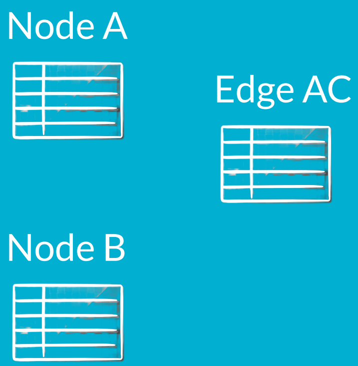
- Recommend Movies to Users
  - Predict ratings
  - If User -> Movie predicted rating is high, recommend



Building your dataset for GNNs

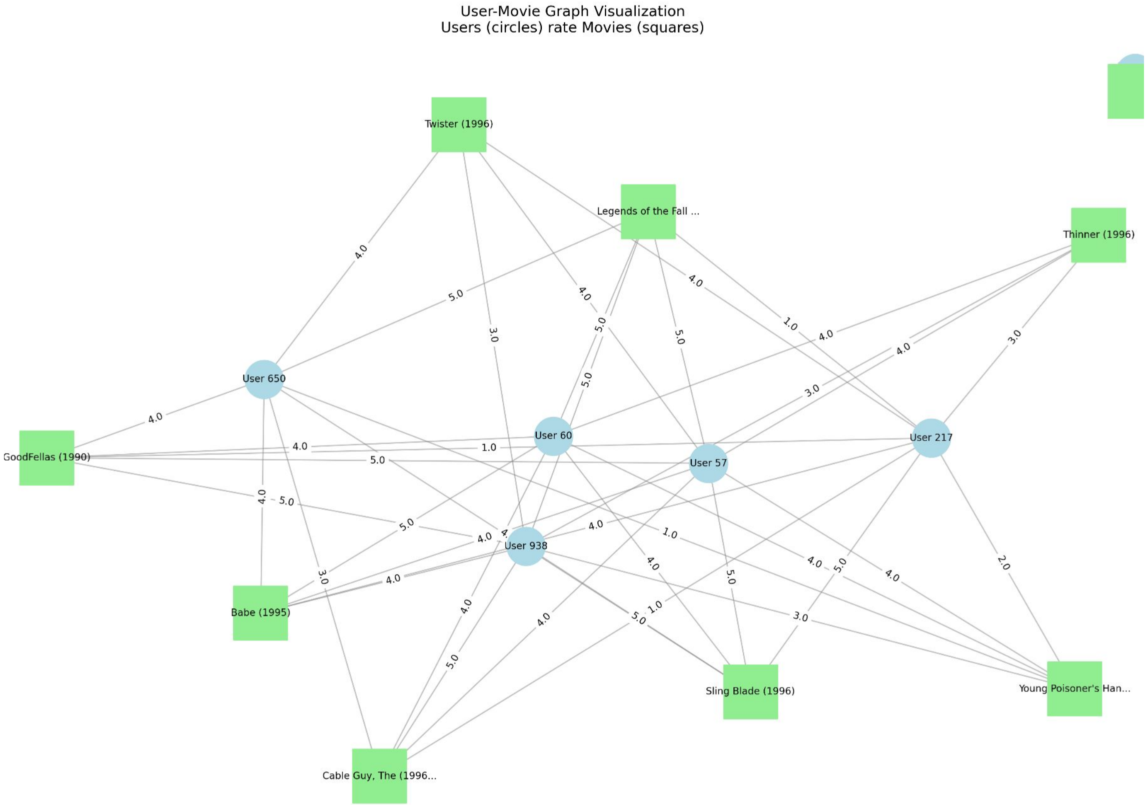
1. Node Features
2. Edge features
3. Adjacency matrix or interaction matrix (directed / undirected / weighted)

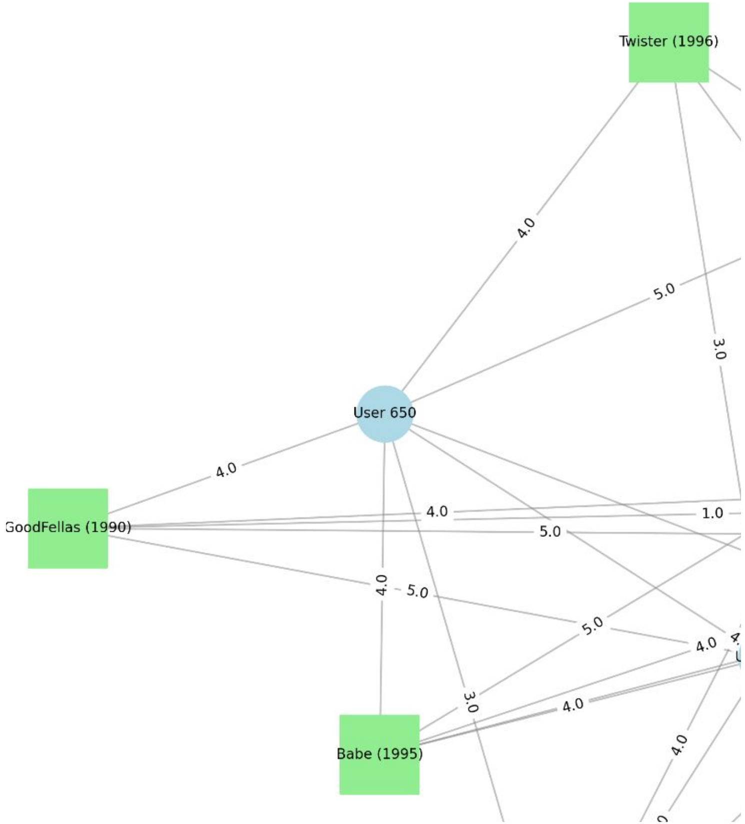




From/ To	A	B	C
A	0	0	1
B	0	0	0
C	1	0	0

Structuring Our Graph





Show Me How It's Done:
No, for real now...

```
class GNNRecommender(nn.Module):
    def __init__(self, num_users, num_movies, embedding_dim=128,
num_layers=3):
        super(GNNRecommender, self).__init__()
        self.user_embeddings = nn.Embedding(num_users, embedding_dim)
        self.movie_embeddings = nn.Embedding(num_movies, embedding_dim)

        self.layers = nn.ModuleList()
        for i in range(num_layers):
            self.layers.append(
                dgl.nn.SAGEConv(
                    embedding_dim,
                    embedding_dim,
                    aggregator_type='mean',
                    activation=F.relu
                )
            )
```

Initialize embeddings

Message passing layers

```python
self.predictor = nn.Sequential(
    nn.Linear(embedding_dim * 2, embedding_dim),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(embedding_dim, embedding_dim // 2),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(embedding_dim // 2, 1)
```

We train embeddings, but we need a final rating for the recommendation task

This predictor is trained to predict rating from embeddings of a given (user, movie) pair

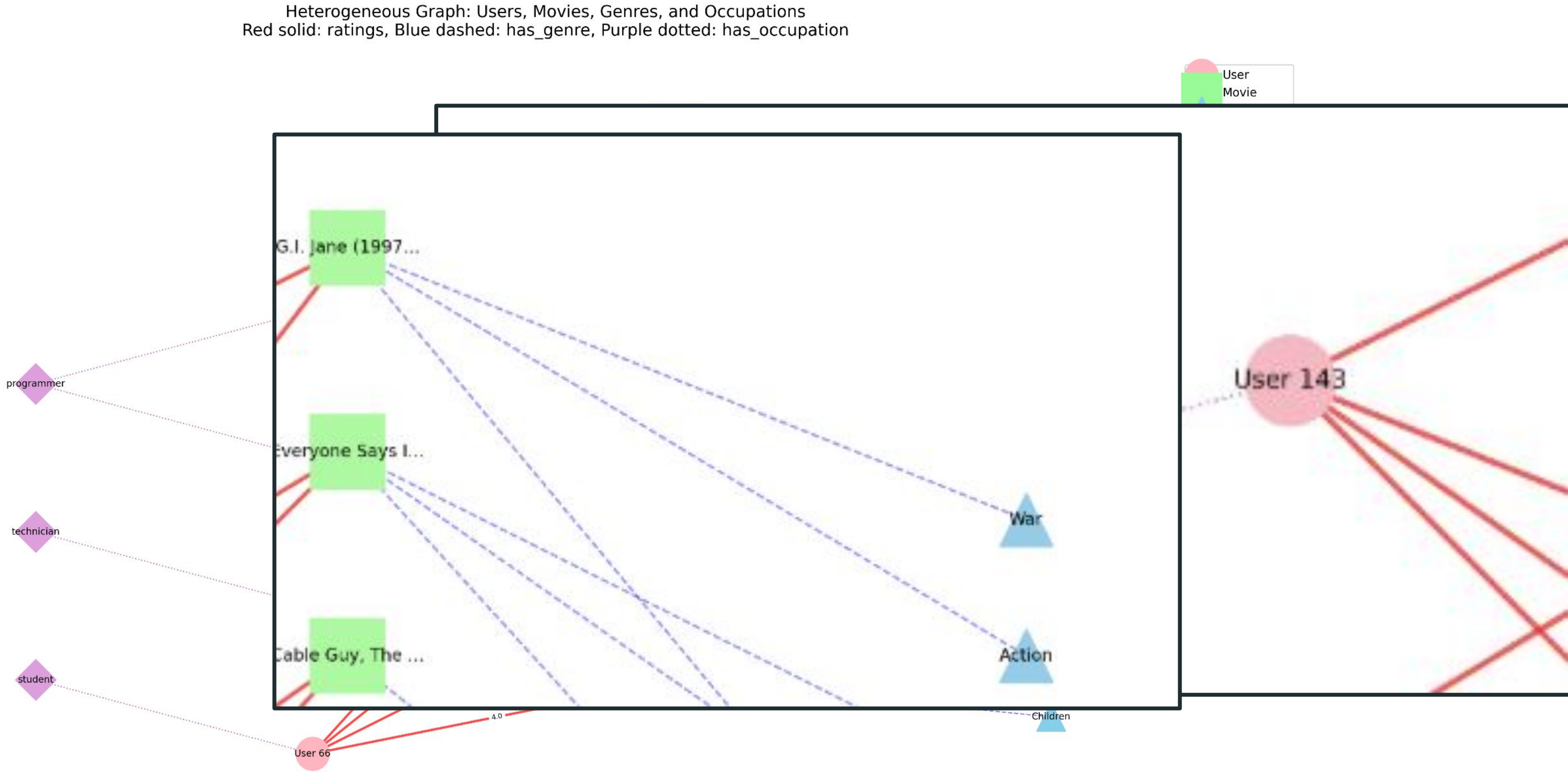
Adding Node Features as Node Data

```
user_features = torch.cat([
    normalize_ages(users_df['age']),
    one_hot_gender,
    one_hot_occupation
], dim=1)

movie_features = torch.cat([
    normalize_years(movies_df['year']),
    movies_df[genre_columns].values], dim=1)
```

```
class GNNRecommender(nn.Module):
    def __init__(self, in_feats, h_feats):
        super().__init__()
        self.conv1 = SAGEConv(in_feats,
h_feats, "mean")
        self.conv2 = SAGEConv(h_feats,
h_feats, "mean")
```

Adding Node Features in the Graph Structure



Adding Node Features: Node Attributes vs Nodes

Node Attributes

- Dense, continuous features example: Age, embeddings
- Computational efficiency: Fewer nodes = faster training
- Standard ML pipeline: Easier to incorporate and compare

Individual Nodes

- Sparse, categorical features example: Genres, state
- Dynamic features: merging, hierarchical
- Interpretability: what drives recommendations
    - Multi-hop reasoning: "Users who like Action movies also like Thriller movies"

Train and Test Graph Splitting

```
def split_data(ratings_df, test_size=0.2):
    ratings_df = ratings_df.sort_values('timestamp')
    train_data = []
    test_data = []
    for user_id, user_ratings in ratings_df.groupby('user_id'):
        n_ratings = len(user_ratings)
        n_test = max(1, int(test_size * n_ratings))  # At least 1 test rating per user
        user_test = user_ratings.iloc[-n_test:]
        user_train = user_ratings.iloc[:-n_test]

        train_data.append(user_train)
        test_data.append(user_test)
…
    return train_df, test_df
```

Graph Creation from Dataframes

```python
def create_graph_from_ratings(ratings_df, num_users, num_movies):
    """Create a graph from ratings dataframe."""
    user_nodes = torch.tensor(ratings_df['user_id'].values)
    movie_nodes = torch.tensor(ratings_df['movie_id'].values)

    # Create edges (user to movie)
    src = torch.cat([user_nodes, movie_nodes + num_users])
    dst = torch.cat([movie_nodes + num_users, user_nodes])

    # Create the graph
    g = dgl.graph((src, dst), num_nodes=num_users + num_movies)

    return g, user_nodes, movie_nodes
```

Training, finally!

```
train_g, train_users, train_movies = create_graph_from_ratings(train_df, num_users,
num_movies)
test_g, test_users, test_movies = create_graph_from_ratings(test_df, num_users, num_movies)

def train_model(..):
    optimizer = torch.optim.AdamW(model.parameters(), lr=lr, weight_decay=0.01)
    train_losses = []
    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()

        pred_ratings = model(train_g, train_users, train_movies)
        train_loss = F.mse_loss(pred_ratings, train_ratings.float())
```

Loss Function / Evaluation Metrics

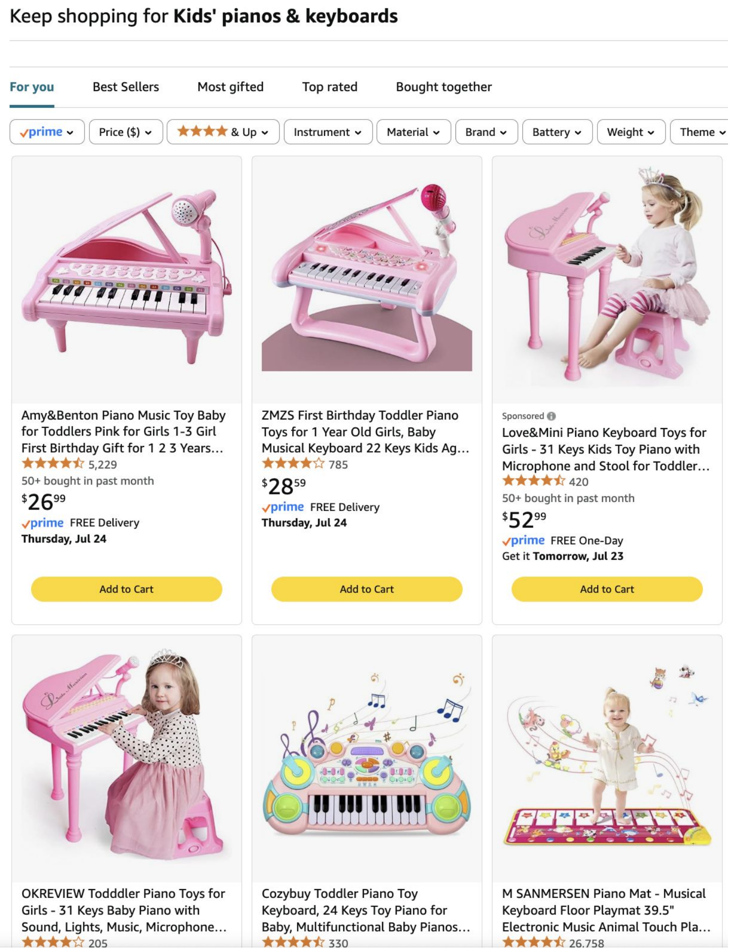
- Binary existence of edge: Classification metrics
  - Cross entropy loss
- Edge weight: Regression metrics
  - RMSE / MSE
- Recommendation metrics:
  - HITS @ K
  - Precision @ K
  - Mean Average Precision

Where to look twice

- Homogeneous vs heterogeneous graphs
  - Specify which edge to predict
- Explicit vs Implicit Negative Feedback

Positive and Negative Graphs

- RecSys NEED some negative data to help the model learn both sides – what the user likes vs doesn't
  - Movie ratings case study is simple
- Cases where we don't have explicit negative
  - Assume negative feedback when we don't have data
  - Example: Amazon search results (clicks vs views)
- We create negative graphs based on this assumption



```python

# Split edge set for training and testing
u, v = g.edges()

eids = np.arange(g.num_edges())
eids = np.random.permutation(eids)
test_size = int(len(eids) * 0.1)
train_size = g.num_edges() - test_size
test_pos_u, test_pos_v = u[eids[:test_size]], v[eids[:test_size]]
train_pos_u, train_pos_v = u[eids[test_size:]], v[eids[test_size:]]

# Find all negative edges and split them for training and testing
adj = sp.coo_matrix((np.ones(len(u)), (u.numpy(), v.numpy())))
adj_neg = 1 - adj.todense() - np.eye(g.num_nodes())
neg_u, neg_v = np.where(adj_neg != 0)

neg_eids = np.random.choice(len(neg_u), g.num_edges())
test_neg_u, test_neg_v = (
    neg_u[neg_eids[:test_size]],
    neg_v[neg_eids[:test_size]],
)
train_neg_u, train_neg_v = (
    neg_u[neg_eids[test_size:]],
    neg_v[neg_eids[test_size:]],
)
```

Positive graph

Adjacency Matrix

Inverse of Adjacency Matrix

Negative edges

Negative graph

Where to look twice

- Homogeneous vs heterogeneous graphs
    - Specify which edge to predict
- Explicit vs Implicit Negative Feedback
- Transductive vs Inductive Training
    - What to do when new nodes need to be added like new users or movies?
- Number of Conv layers = Number of neighbors to average
    - If k >> 1, all nodes will carry all and the same information which is the graph average
- Beware of frameworks!

Thank You!

Slides + code: https://github.com/ShreyaKhurana/pyohio-2025/

LinkedIn: https://www.linkedin.com/in/shreya-khurana/