This Talk

1) Node embeddings
   - Map nodes to low-dimensional embeddings.

2) Graph neural networks
   - Deep learning architectures for graph-structured data

3) Applications
Part 3: Applications
Outline for This Section

- Recommender systems
  - RW-GCNs: GraphSAGE-based model to make recommendations to millions of users on Pinterest.

- Computational biology
  - Decagon: Predicting polypharmacy side-effects with graph neural networks.

- Practical insights
  - Code repos, useful frameworks, etc

- Future directions
RW-GCNs: Graph Convolutional Networks for Web-Scale Recommender Systems

Based on material from:
- Graph is dynamic: need to apply to new nodes without model retraining
- Rich node features: content, image
Graph Neural Nets for RecSys

- Two sources of information in traditional recommender systems:
  - **Content features**: User and item features, in the form of images, categories etc.
  - **Network structure**: User-item interactions, in the form of graph/network structure.

- Graph neural networks naturally incorporate both!!
Application: Pinterest

Human curated collection of pins

**Pins**: Visual bookmarks someone has saved from the internet to a board they’ve created.

**Pin features**: Image, text, link
Task: Recommend related pins to users.

Challenges:
- Massive size: 3 billion pins and boards, 16 billion interactions
- Heterogeneous data: Rich image and text features
RW-GCN Overview

- Random-Walk GCNs = RW-GCNs
- Architecture is an extension of GraphSAGE:

\[ h^{(2)}_A \]

\[ h^{(1)}_A \]

\[ \gamma \]

\[ h^{(1)}_{\mathcal{N}(A)} \]

element-wise mean or max

multilayer perceptrons (MLPs)
Overview of RW-GCN Pipeline

1. **Collect** billions of training pairs from user logs.

2. **Train** system to generate similar embeddings for training pairs.

3. **Generate** embeddings for all pins.

4. **Make recommendations** using nearest neighbor search in the embedding space (in real time).
RW-GCN Overview

- Train so that **pins that are consecutively clicked** have similar embeddings.
- Max-margin loss:

\[ \mathcal{L} = \sum_{(u,v) \in \mathcal{D}} \max(0, -z_u^T z_v + z_u^T z_n + \Delta) \]

- set of training pairs from user logs
- "positive"/true training pair
- "negative" sample
- "margin" (i.e., how much larger positive pair similarity should be compared to negative)
RW-GCN Efficiency

- 10,000X larger than any previous graph neural network application.

Key innovations:
1. Sub-sample neighborhoods for efficient GPU batching
2. Producer-consumer training pipeline
3. Curriculum learning for negative samples
4. MapReduce for efficient inference
Neighborhood Subsampling

INPUT GRAPH

TARGET NODE

AGGREGATE

BATCH OF NODES

Every node has unique compute graph. Can’t batch on GPU!
Neighborhood Subsampling

Compute graphs have same structure = efficient GPU batching
Neighborhood Subsampling

- **Random-walk**-based neighborhood
  - Approximates personalized PageRank (PPR) score.
  - Sampled neighborhood for a node is a list of nodes with the top-K PPR score.

- **Advantage:**
  - Algorithm finds the most relevant nodes(item) for high degree nodes
Producer-consumer Pipeline

- Select a batch of pins
- Run random walks
- Construct their computation graphs

- Multi-layer aggregations
- Loss computation
- Backprop
Curriculum Learning

- **Idea:** use harder and harder negative samples
- Include more and more hard negative samples for each epoch

Source pin | Positive | Easy negative | Hard negative
MapReduce Inference

- How to efficiently infer representations on nodes we have not seen during training time?
- **Key insight**: avoid repeated computation by sharing computation in MapReduce layers!

![Diagram showing the process of MapReduce Inference with nodes and layers labeled: Item, Visual, Item, Annotation, Item, Degree, Context, Item, Join item, MLP, Join item, GroupBy context, Reduce by pooling, First level representation.](representation-learning-on-networks-snap.stanford.edu/proj/embeddings-www-wwwwww-2018)
RW-GCN Performance

- 72% better recommendation quality than standard GraphSAGE model.

Key innovations:

1. **Weigh importance** of neighbors according to approximate PPR score.
2. **Use curriculum training** to provide harder and harder training examples over time.
**Set-up:** Rank true “next-clicked” pin against $10^9$ other candidates.

**MRR:** Mean reciprocal rank of true example.

**Baselines:** Deep content-based models
Example Recommendations

Visual

Annot.

RW-GCN
Decagon:
A Graph Convolutional Approach to Polypharmacy Side Effects

Based on material from:
Polypharmacy Side Effects

**Goal:** Predict side effects of taking multiple drugs.

<table>
<thead>
<tr>
<th>Individual medications</th>
<th>Patient’s side effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>No side effect</td>
<td>No side effect</td>
</tr>
<tr>
<td>No side effect</td>
<td></td>
</tr>
</tbody>
</table>

**Drug combination**

Polypharmacy side effect

\{ ![Head with headache] , ![Face with rash] \}
Polypharmacy Side Effects

- Polypharmacy is common to treat complex diseases and co-existing conditions
- High risk of side effects due to interactions
- 15% of the U.S. population affected
- Annual costs exceed $177 billion
- Difficult to identify manually:
  - Rare, occur only in a subset of patients
  - Not observed in clinical testing
Modeling Polypharmacy

- Systematic experimental screening of drug interactions is challenging
- **Idea:** Computationally screen/predict polypharmacy side effects
  - Use molecular, pharmacological and patient population data
  - Guide strategies for combination treatments in patients
Data: Heterogeneous Graphs

Drugs

- Doxycycline
- Simvastatin
- Mupirocin

Genes

- Ciprofloxacin

Drugs and Genes are connected through various relationships such as:
- \( r_1 \): Gastrointestinal bleed effect
- \( r_2 \): Bradycardia effect

Other connections include:
- Drug target interaction
- Physical protein binding

The graph representation allows for learning embeddings that capture the structure and relationships within these heterogeneous networks.
Task Description

- **Predict labeled edges between drugs**
  - i.e., predict the likelihood that an edge $(c, r_2, s)$ exists

- **Meaning:** Drug combination $(c, s)$ leads to polypharmacy side effect $r_2$
Neural Architecture: Encoder

- **Input**: graph, additional node features
- **Output**: node embeddings
Making Edge Predictions

- **Input:** Query drug pairs and their embeddings
- **Output:** predicted edges

Diagram:
- Drug: Green Circle
- Gene: Orange Circle
- Feature vector: Black Box
- Gastrointestinal bleed effect: $r_1$
- Bradycardia effect: $r_2$
- Drug target interaction: Green Dot
- Physical protein binding: Orange Dot
- polypharmacy side effects: $p(C, r_1, S)$
  $p(C, r_2, S)$
  $p(C, r_4, S)$
  $p(C, r_5, S)$
  $p(C, r_n, S)$
Experimental Setup

- **Data:**
  - **Molecular:** protein-protein interactions and drug target relationships
  - **Patient data:** Side effects of individual drugs, polypharmacy side effects of drug combinations

- **Setup:**
  - Construct a heterogeneous graph of all the data
  - **Train:** Fit a model to predict known associations of drug pairs and polypharmacy side effects
  - **Test:** Given a query drug pair, predict candidate polypharmacy side effects
### Prediction Performance

<table>
<thead>
<tr>
<th>Method</th>
<th>AUROC</th>
<th>AUPRC</th>
<th>AP@50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decagon (3-layer)</td>
<td>0.834</td>
<td>0.776</td>
<td>0.731</td>
</tr>
<tr>
<td>Decagon (2-layer)</td>
<td>0.809</td>
<td>0.762</td>
<td>0.713</td>
</tr>
<tr>
<td>RESCAL</td>
<td>0.693</td>
<td>0.613</td>
<td>0.476</td>
</tr>
<tr>
<td>Node2vec</td>
<td>0.725</td>
<td>0.708</td>
<td>0.643</td>
</tr>
<tr>
<td>Drug features</td>
<td>0.736</td>
<td>0.722</td>
<td>0.679</td>
</tr>
</tbody>
</table>

- Up to **54% improvement** over baselines
- **First opportunity** to computationally flag polypharmacy side effects for follow-up analyses
Practical Insights
A quick example: Using GraphSAGE for a supervised node classification task.

Key steps:

1. Preprocess network and training data.
2. Run GraphSAGE
GraphSAGE TensorFlow Ex.

- **Preprocessing**

```python
from networkx.readwrite import json_graph
import json
import numpy as np

Data = json_graph.node_link_data(G)
with open('data-G.json') as f:
    f.write(json.dumps(data))

class_map = {nodes[i]: labels[i] for i in range(len(nodes))}
with open('data-class_map.json') as f:
    f.write(json.dumps(data))

id_map = {nodes[i]: i for i in range(len(nodes))}
with open('data-id_map.json') as f:
    f.write(json.dumps(data))

np.save(feats, 'data-feats.npy')
```

- **Save graph**

- **Save labels**

- **Save nodes**

- **Save features**
GraphSAGE TensorFlow Ex.

- Example: PPI data (available in GraphSAGE repo)

- Run both training and evaluation (random split of data)

  ```
  python -m graphsage.utils ppi-G.json ppi-walks.txt
  python -m graphsage.supervised_train --train_prefix=./ --model=graphsage_mean
  ```

- Alternative models:
  - gcn, graphsage_seq, graphsage_maxpool

- Easy to customize using Tensorflow
Future Directions
(Sub)graph embedding

- Existing approaches
  - Pool learned node embeddings via element-wise max/mean/sum
  - Add a “virtual” node representing the entire (sub)graph

- Is there better pooling strategy?
  - Handle massive graphs?
  - Learn “coarsened” representations?
Dynamic graphs

- Many graphs evolve over time:
  - Recommender systems
  - Financial transaction and event graphs
  - Social networks

- Applications:
  - Predict graph evolution
  - Anomaly detection (e.g., fraud)
Dynamic graphs

- Challenges:
  - How to efficiently and incrementally update the learned representations?
  - How to incorporate edge timing?
  - How to “forget” old/irrelevant info?
Combinatorial Applications

- Efficient SAT solvers via graph embeddings (Selsam et al., 2018).
- Learn embeddings of clause and literals (form a bipartite graph)
- Graph embeddings for neural theorem proving?
Reinforcement Learning

- **Idea:** Allow agents to use node embedding information to make decisions
- **So far:** Used for combinatorial optimization (Dai et al., 2017) and question answering (Das et al., 2018)
- **New directions:**
  - Game playing?
  - Graph representations of dialogue state?
Using Graph Neural Networks

- **Popular Code Bases:**
  - GCN (Tensorflow):
    https://github.com/tkipf/gcn/
  - GraphSAGE (Tensorflow):
    https://github.com/williamleif/GraphSAGE
  - GraphSAGE (PyTorch):
    https://github.com/williamleif/graphsage-simple/
This Talk

1) Node embeddings
   - Map nodes to low-dimensional embeddings.

2) Graph neural networks
   - Deep learning architectures for graph-structured data

3) Applications