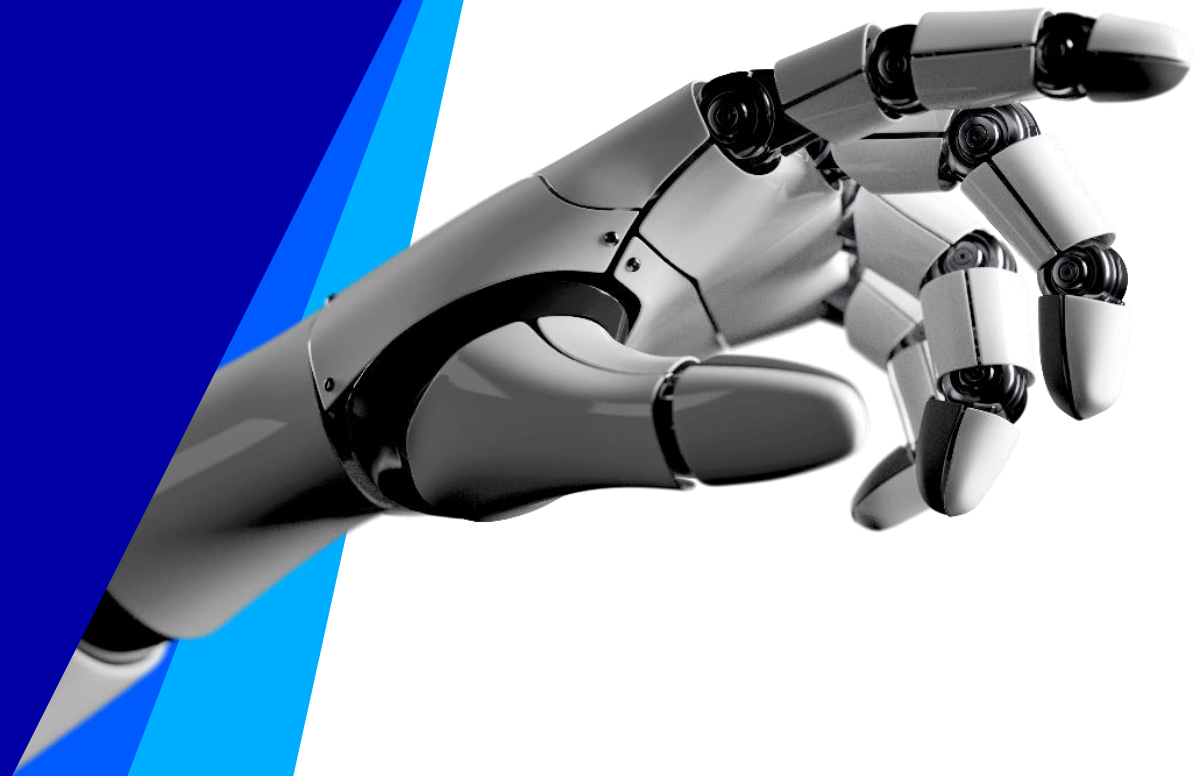


Session 16

PERFORMANCE
EN FINANCEMENT
DE L'INNOVATION

Node embedding techniques

- Auto-encoder· Shallow encoder + dot-product decoder
- Random Walk embeddings· Node2Vec, DeepWalk
- Graph Auto-encoder, Graph VAE

Graph Classification

France · Espagne · Royaume-Uni · Portugal · Belgique · Italie · Canada · Brésil · Chili · Colombie

Graph Classification

- Goal· Assign a label/class to an entire graph
- Example of classification· molecular graphs (drug likeliness, toxicity prediction)
- Two stage graph classification
  - Pretrain node embeddings using techniques like Node2Vec and compute graph representation using a readout function (global sum/max/mean) on pre-trained graph embeddings
  - Predict graph label by applying an MLP on the graph embeddings
- End-to-end approach
  - Perform node representation + graph classification jointly using GNNs
    - Node representation — H = GNN(X, A) — (GCN, GAT, SAGEConv, …)
    - Graph-level representation — s=Readout(H) — (sum/avg/max)
    - Prediction using an MLP — $\hat{y} = MLP(s)$
    - Compute loss — $L(\hat{y}, y) = CE(\hat{y}, y)$
    - Optimize loss using SGD

Pytorch session

- Graph classification with pytorch + pyG

Scaling GNN

Based on (Graph Deep leaning book, Ma & Tang , chapiter 7), (Lecture 17 CS224W, Leskovec 2021)

Limitation of vanilla GNN

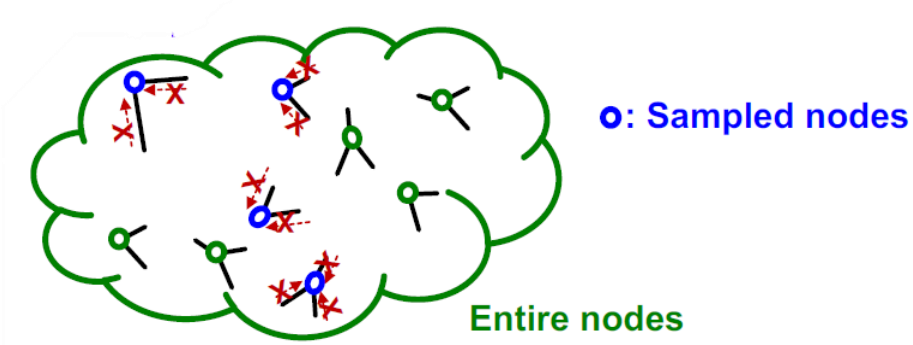
$$H^{(l)} = \sigma\left(\hat{A}H^{(l-1)}W^{(l)}\right), \text{ for } l = 1, \ldots, L$$

- During the forward pass, the representations for all nodes and all the parameters need to be stored in memory (for backprop)
  - $\hat{A}$ $O(|E|)$ => storing edge indexes
  - $H^{(l)}$ $O(L\,|V|\,d)$ => memory for all the intermediate node embeddings
  - $W^{(l)}$ $O(L\,d^2)$ => memory for all weight matrices
  - Total $O(|E| + L\,|V|d + L\,d^2) \sim O(L\,|V|d + L\,d^2)$
  - When the size of the graph is large (i.e $|E|$ or $|V|$ are large), it becomes impossible to fit them into the memory (social networks, youtube, amazon, …)
- Solution· Use Stochastic Gradient Descent (SGD) or Mini-batch SGD

Graph Deep leaning book, Ma & Tang chapiter 7

Random node sampling

- Sample a mini-batch of M (<<|V|) nodes independently



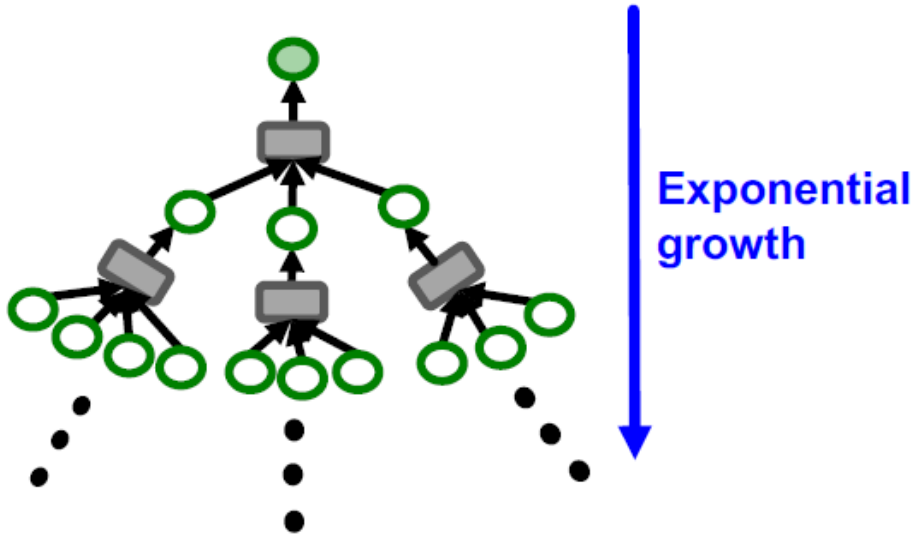
- Sampled nodes tend to be isolated from each other
- Recall· GNN generate node embeddings by aggregating neighboring node features.
  - GNN does not access to neighboring nodes within the mini-batch
- Standard SGD (iid assumption) cannot effectively train GNNs

Memory complexity· $O(L\,Md\ +\ L\,d^2)$

L-hop neighbors

- Idea· for a GNN of L layers, to compute embedding of a single node, all we need is the L-hop neighborhood

- let's consider the following training strategy
  - Randomly sample M (<<|V|) nodes
  - For each sampled node v
    - Get L-Hop neighborhood
    - Compute v's embedding
  - Compute average loss for M nodes
  - Perform SGD

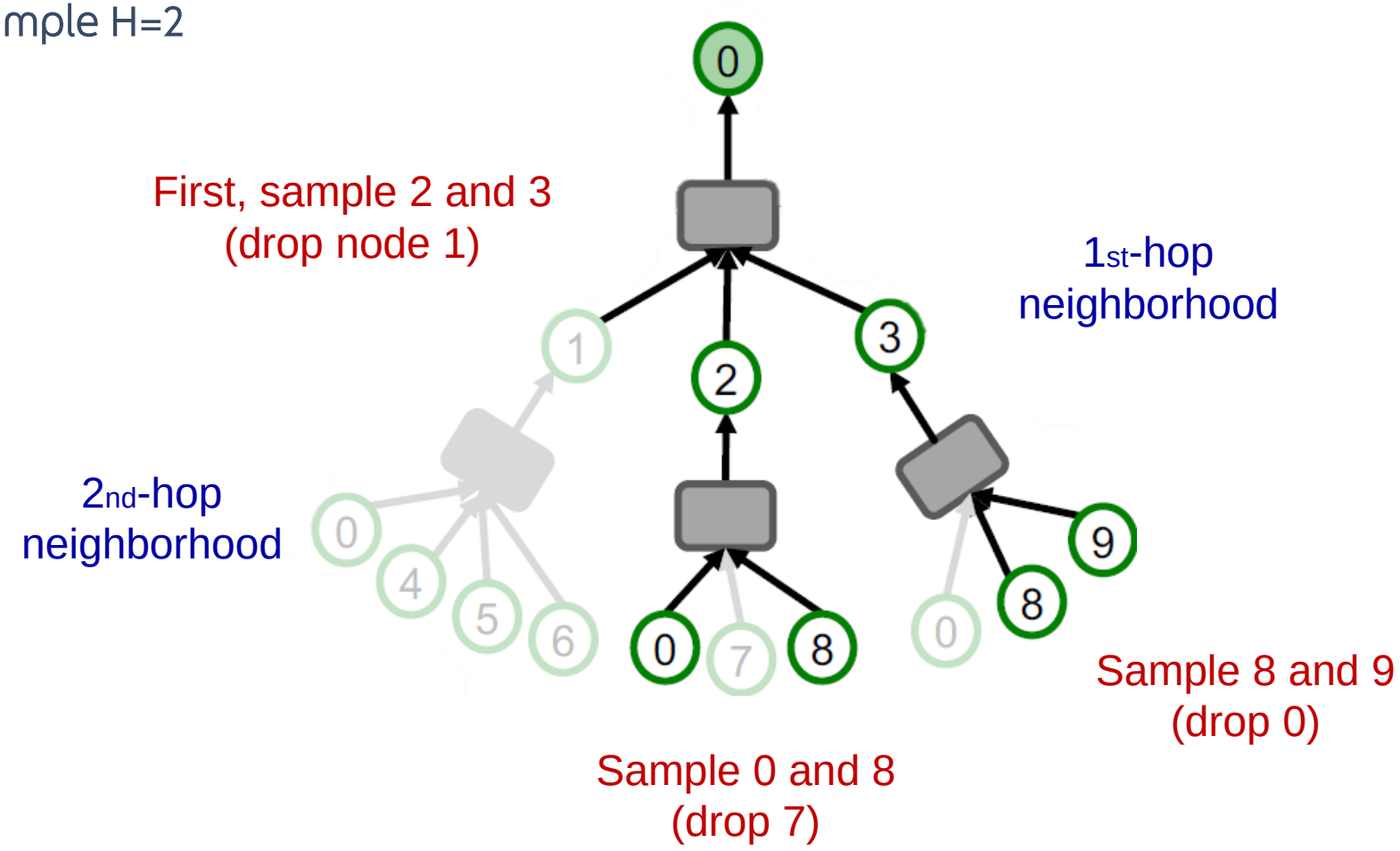


- Problems·
  - Does not scale· the number of neighbors grow exponentially
  - Need to aggregate lot of information just to compute one node embedding

Memory complexity· $O(d^L\ Md\ +\ L\ d^2)$ where d is the average degree of each nodes

GraphSage neighbor sampling

- Idea· sampling at most H neighbors at each hop
- Example H=2

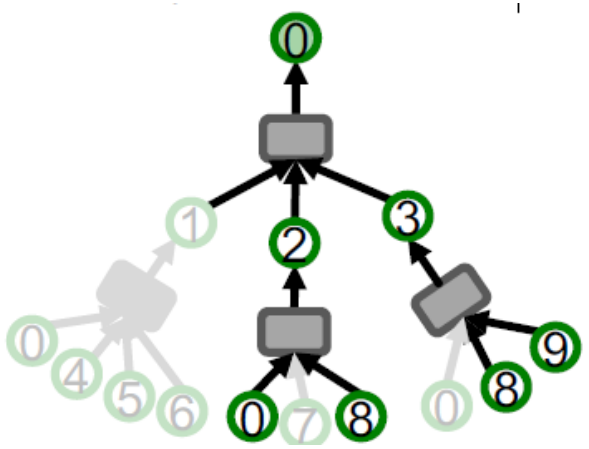


Neighbor sampling for L-layer GNN

- For l=1, … , L
  - For each node in k-hop neighborhood
  - (Randomly) sample at most H_l neighbors

1st-hop neighborhood Sample $H_1 = 2$ neighbors

2nd-hop neighborhood Sample $H_2 = 2$ neighbors



- Trade-off· Smaller H leads to more efficient neighbor aggr. but larger variance
- Modern GNN librairies already provide efficient implementation of Neighbor sampling (pytorch geometric, DGL, …)

Memory complexity· $O(H^L\ Md\ +\ L\ d^2)$, H is the number of sampled neighbors per node.

Pytorch session

- GraphSAGE + Neighbor sampling for node classification on pytorch + pyG

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$; | Initial node features

for $k = 1...K$ **do**

  for $v \in \mathcal{V}$ **do**

    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$; | Sample + Aggregate neighbors

    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$ | Update node representation

  end

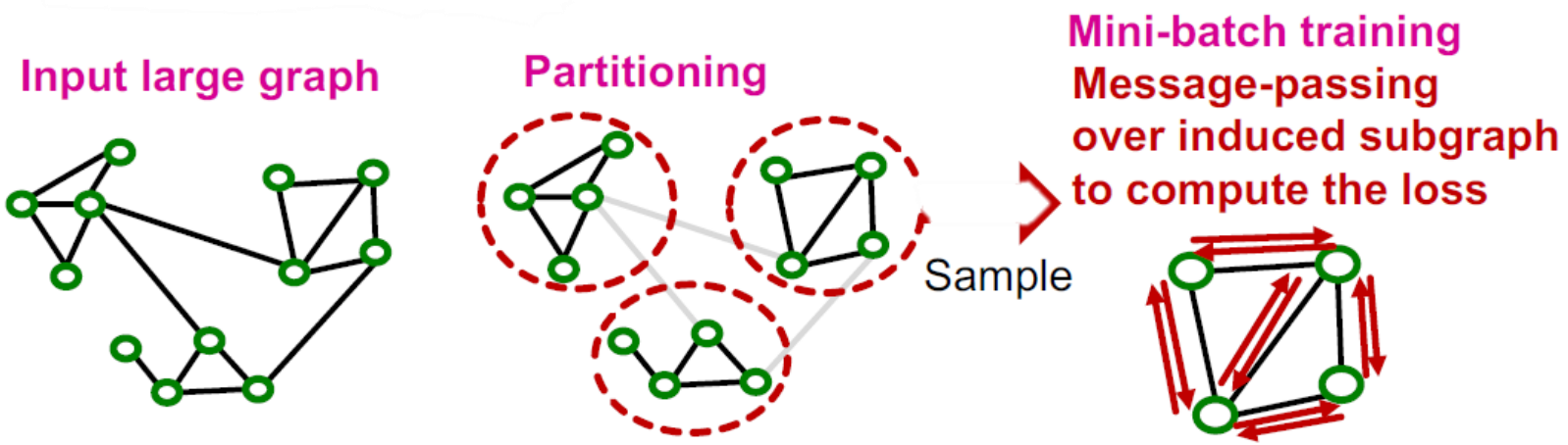
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$

end

$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$ | Final node representation

Cluster-GCN

- Given a large graph, partition it into groups of nodes or subgraphs (using scalable methods such as METIS) => avoid exponential neighborhood expansion
- Sample one node group at a time
- Apply GNN on the subgraph

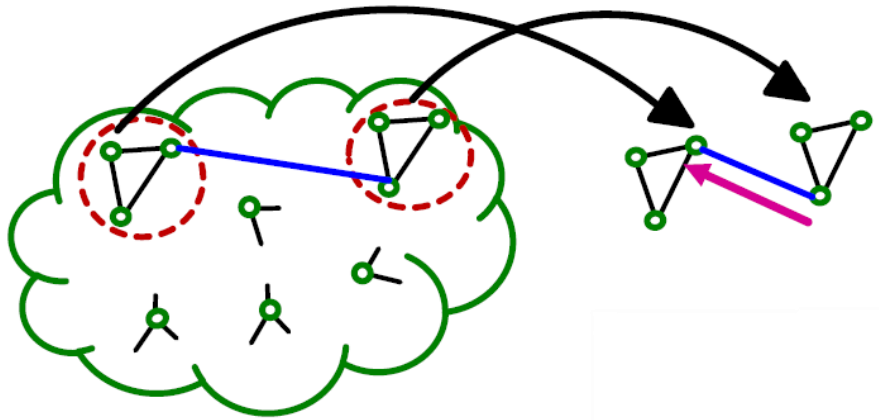


Memory complexity: $O(LMd + Ld^2)$, M the batch size (subgraphs size)

Chiang, Wei-Lin, et al. "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks." SIGKDD 2019

Cluster-GCN

- Problems of Cluster-GCN·
  - Induced subgraph removes between group links which can hurt performance
  - Sampled nodes are not diverse enough to be represent the entire graph structure => bad estimation of the gradient (high variance) => slow convergence
- Solution·
  - Partition graph into small groups and
  - Aggregate multiple node groups per mini-batch (the links between the chosen clusters are added back)
  - => this make make the variance across batches smaller => improve convergence



Pytorch session

- Cluster-GCN for node classification with pytorch + pyG

GraphSAINT random walk sampler

- Given a large graph, sample subgraph using a Random walk sampler (starting from r randomly samples nodes)
- Apply GNN on the subgraph

function RW($\mathcal{G}$,r,h) ▷ Random walk sampler
  $\mathcal{V}_{\text{root}} \leftarrow r$ root nodes sampled uniformly at random (with replacement) from $\mathcal{V}$
  $\mathcal{V}_s \leftarrow \mathcal{V}_{\text{root}}$
  for $v \in \mathcal{V}_{\text{root}}$ **do**
    $u \leftarrow v$
    for $d = 1$ to h **do**
      $u \leftarrow$ Node sampled uniformly at random from u's neighbor
      $\mathcal{V}_s \leftarrow \mathcal{V}_s \cup \{u\}$
    end for
  end for
  $\mathcal{G}_s \leftarrow$ Node induced subgraph of $\mathcal{G}$ from $\mathcal{V}_s$
end function

Memory complexity: $O(LMd + L d^2)$, M the batch size (subgraphs size)

Zeng, Hanqing, et al. "GraphSAINT: Graph Sampling Based Inductive Learning Method." ICLR 2020.

Pytorch session

- GraphSaint for node classification with pytorch + pyG

Summary

- Full batch training
  - Pro· best in term of gradient estimation
  - Con· Slow training, large graphs does not fi to GPU
  - Mem complexity· $O(L\,|V|d\, +\, L\,d^2)$
- Mini-batch Random node sampling
  - Pro· scale
  - Con· spasity, poor estimation of the gradient
  - Mem complexity· $O(L\,Md\, +\, L\,d^2)$
- Mini-batch L-hop neighbors
  - Pro· good estimation of the gradient
  - Con· does not scale (exponential neighborhood expansion)
  - Mem complexity· $O(d^L\,Md\, +\, L\,d^2)$
- Neighbor sampling
  - Trade-off· Smaller H leads to more efficient neighbor aggr. but larger variance
  - Mem complexity· $O(H^L\,Md\, +\, L\,d^2)$
- Cluster-GCN & GraphSAINT
  - Pro· Scale
  - Con· Lack of node diversity => poor estimation of the gradient
  - Mem complexity· $O(LMd\, +\, L\,d^2)$

Scalable GNN in the real world

- Food recommendation at UBER EATS
  - Blogpost https://eng.uber.com/uber-eats-graph-learning/
    - Use 2-layer GraphSAGE with Neighbor Sampling

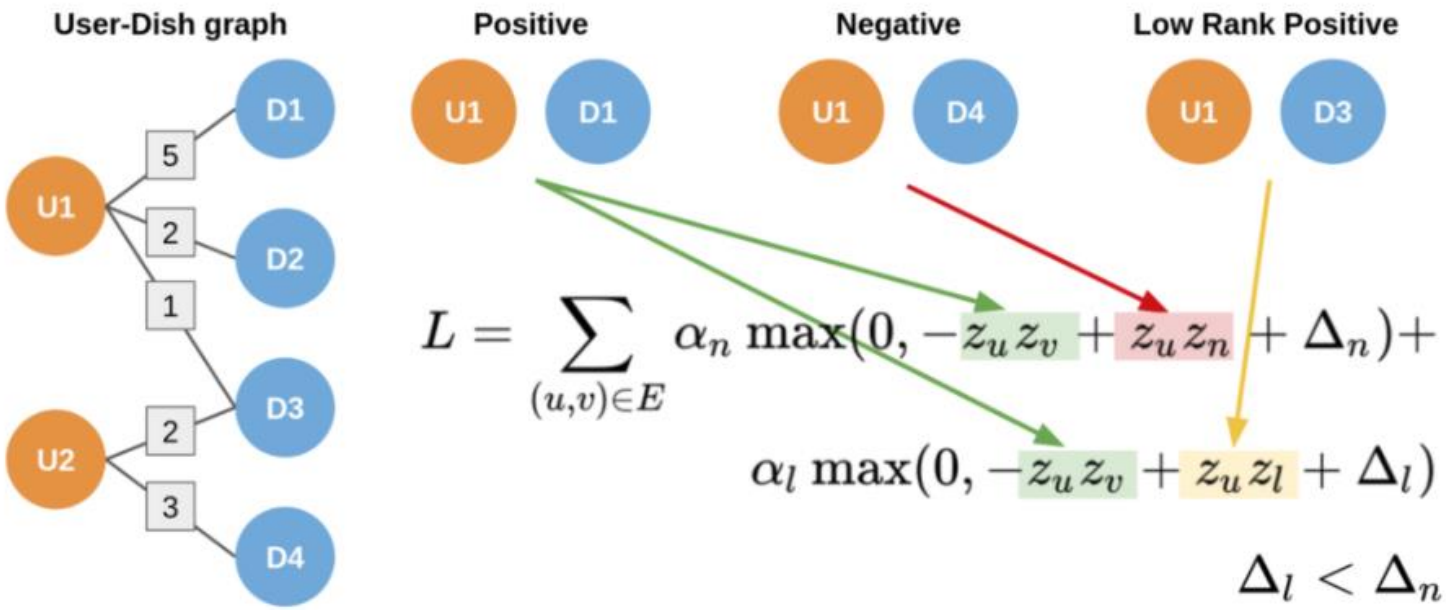


Figure 4: Our Uber Eats recommendation system leverages max-margin loss augmented with low rank positives.

Example of scalable GNN in the real world

- Product recommendation
  - Alibaba product recsys · https·//arxiv.org/abs/1803.02349
  - Amazon product recsys· https·//dl.acm.org/doi/pdf/10.1145/3340531.3412732
- Facebook· Graph embeddings for fake account detection
- Pinterest's Pinsage· GNN for visual similarity
- See Sergey Ivanov ' s blogpost for a review