Graph Convolutional Networks for Web-Scale Recommender Systems

Author: Rex Ying et al.

Bui Tien Cuong – 2019-35731

Presentation Overview

Introduction

PinSage

Experiments

Future Directions

Introduction

Recommendation Systems

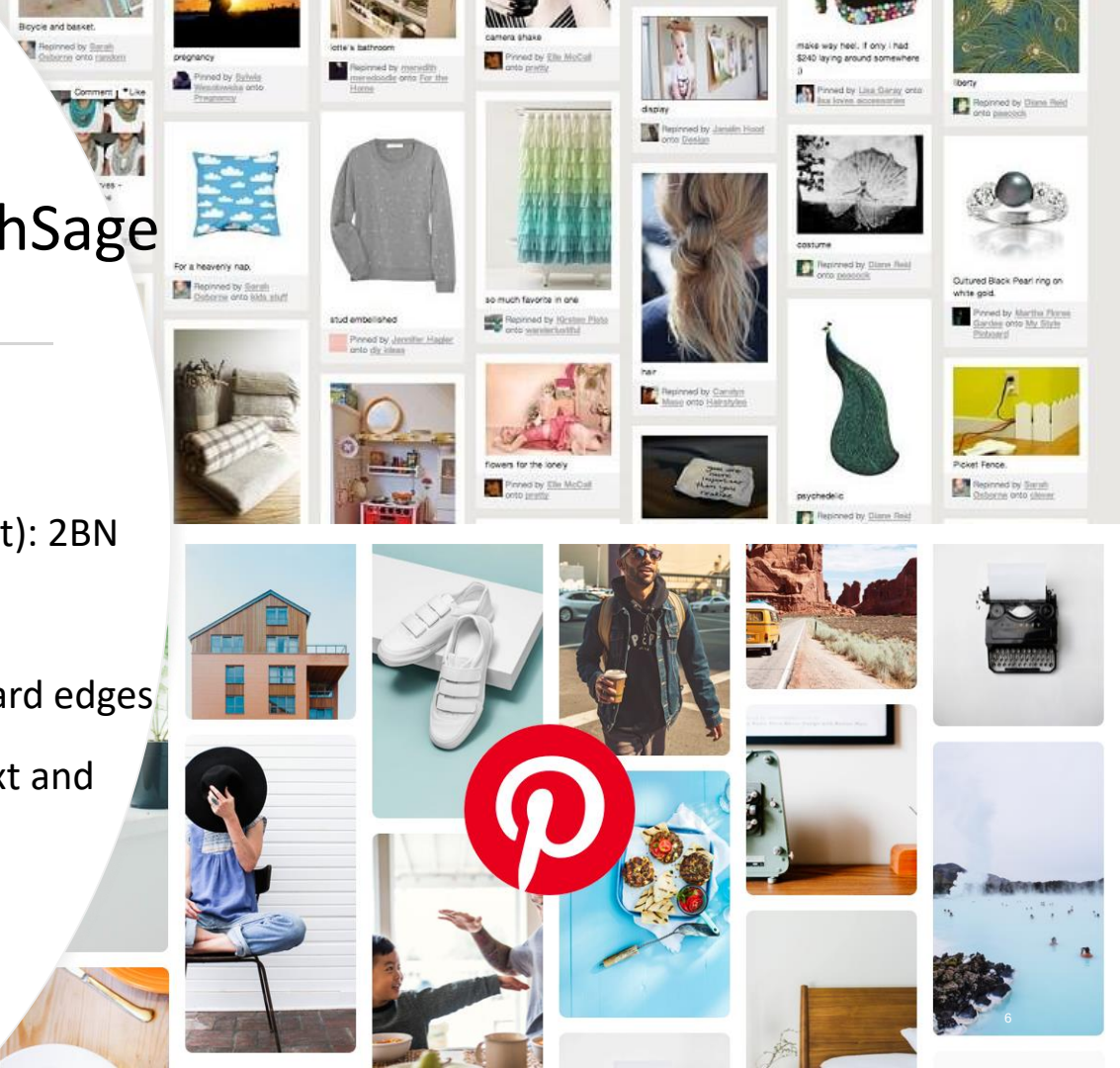
- Graph data are essential for recommendation systems
- Represent relationships between objects (user-item, person-person, …)
- Representation learning achieves significant improvements
- Learn latent features with DNN-based models via node embeddings
- GCN-based methods are efficient on recommendation systems

Challenges

- GCN-based methods are usually designed for small graphs
- Training and inference on large graphs are problematic
- Hard to use full graph Laplacian during training on recommendation systems:
  - Billions of nodes
  - Constantly evolving

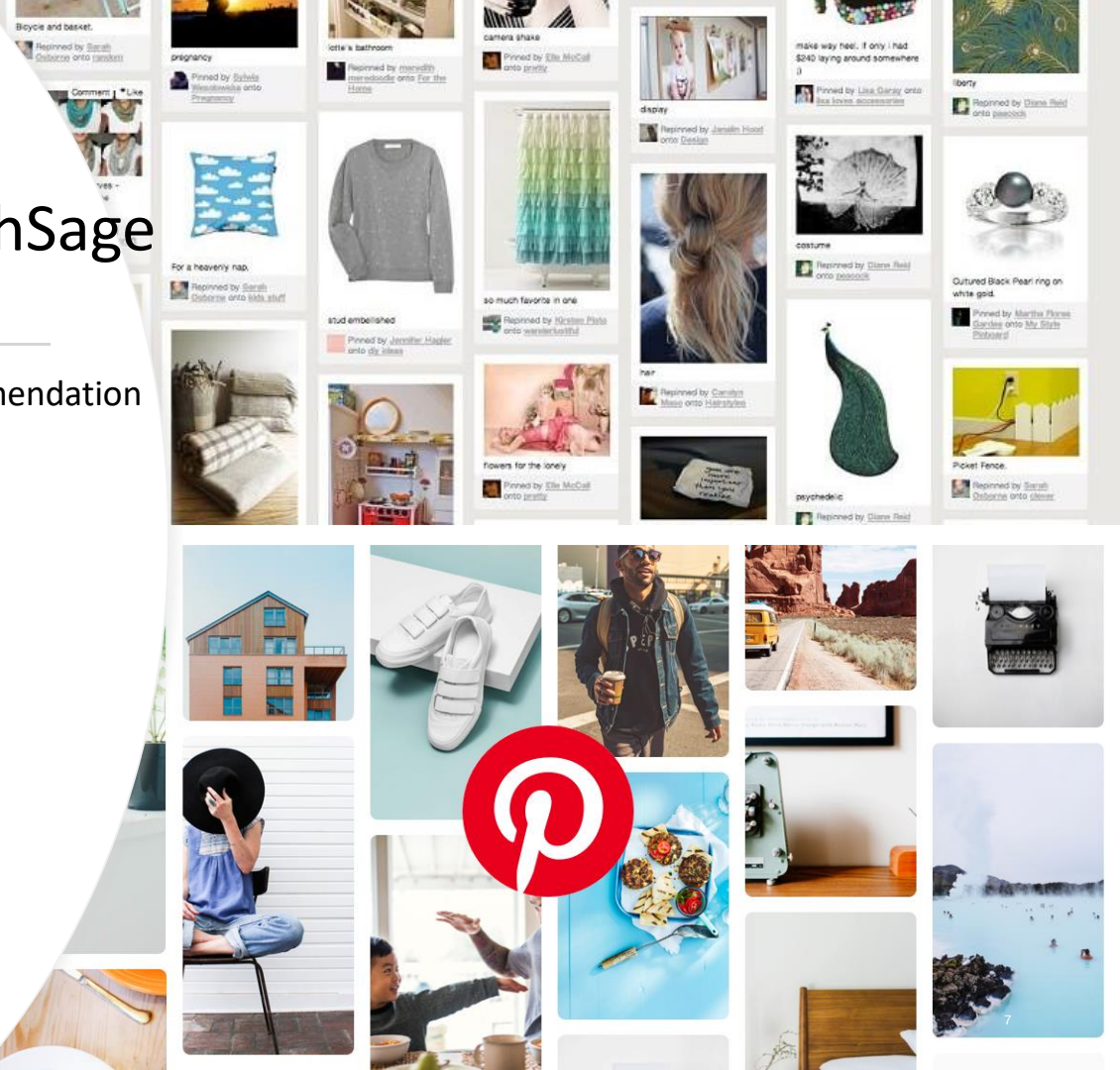
PinSage = Pinterest + GraphSage

- Pinterest: content discovery application
  - Pins – ***I*** (visual links to online content): 2BN
  - Boards – ***C*** (collections of pins): 1BN
- Bipartite graph (**V** = ***I*** U ***C***): 18BN pin-board edges
- A pin u has real-valued attributed $\boldsymbol{x}_u$ (text and image features)



PinSage = Pinterest + GraphSage

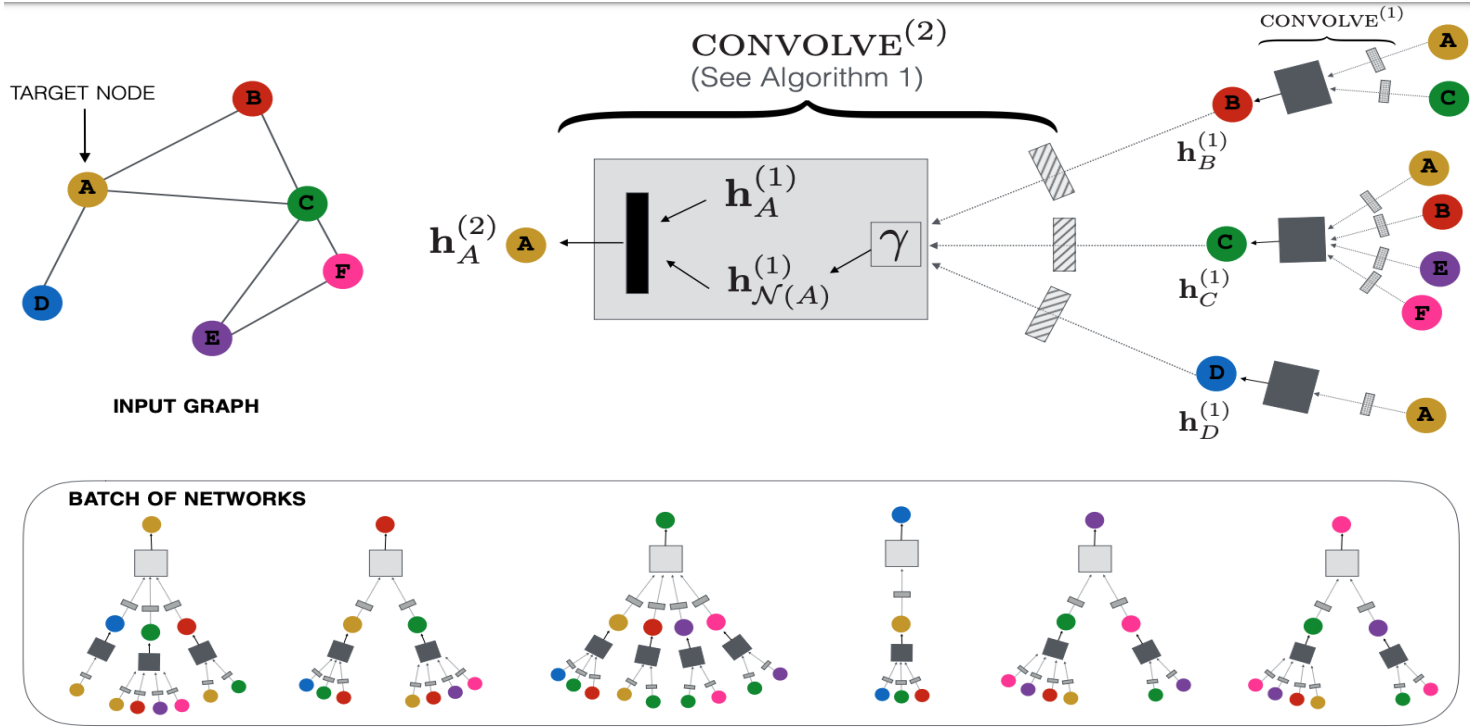
- Apply GraphSage Concepts to web-scale recommendation at Pinterest
- Leverage random walks to sample nodes
- Add 2-layer supports to each node
- Generate embeddings sampled nodes
- At the publication time:
  - Largest application of graph embeddings
  - 3BN nodes, 18BN edges
  - 10000x bigger than regular GCN applications



Key insights

- Localized convolutions:
  - Sampling node through random walks (with personalized PageRank scores)
  - Share parameters across nodes
- Importance pooling: use scores to weight node features
- Curriculum training: increasing difficulty of examples
- Efficiency: map-reduce minibatches
- AGL: an industrial-purpose graph from Ant Financial applies these concepts

Two layers of support



Embeddings for each node are computed by a different network, but parameters are shared among boxes with same shading.

Importance-based neighbor sampling

- Previous methods: k-hop graph neighborhoods
- PinSage:
  - Start random walk from u
  - Compute L1-normalized visit count of nodes
  - ***N(u)*** = T most “influential” neighbors of node u (having the highest visit counts) -> set of weights α

Training

- Labelled pairs of items: L = {(*q,i*) | item i is a good recommendation candidate for query q}
- Goal: output embeddings of q and i are close in the embedding space

Minibatch Algorithm

Algorithm 2: MINIBATCH

Input : Set of nodes $\mathcal{M} \subset \mathcal{V}$; depth parameter K; neighborhood function $\mathcal{N} : \mathcal{V} \rightarrow 2^{\mathcal{V}}$

Output: Embeddings $\mathbf{z}_u, \forall u \in \mathcal{M}$

/* Sampling neighborhoods of minibatch nodes. */

$\mathcal{S}^{(K)} \leftarrow \mathcal{M}$;

for $k = K, \ldots, 1$ **do**

  $\mathcal{S}^{(k-1)} \leftarrow \mathcal{S}^{(k)}$;

  for $u \in \mathcal{S}^{(k)}$ **do**

    $\mathcal{S}^{(k-1)} \leftarrow \mathcal{S}^{(k-1)} \cup \mathcal{N}(u)$;

  end

end

/* Generating embeddings */

$\mathbf{h}_u^{(0)} \leftarrow \mathbf{x}_u, \forall u \in \mathcal{S}^{(0)}$;

for $k = 1, \ldots, K$ **do**

  for $u \in \mathcal{S}^{(k)}$ **do**

    $\mathcal{H} \leftarrow \left\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u)\right\}$;

    $\mathbf{h}_u^{(k)} \leftarrow \text{CONVOLVE}^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathcal{H}\right)$

  end

end

for $u \in \mathcal{M}$ **do**

  $\mathbf{z}_u \leftarrow \mathbf{G}_2 \cdot \text{ReLU}\left(\mathbf{G}_1 \mathbf{h}_u^{(K)} + \mathbf{g}\right)$

end

Negative Sampling

- Approximate the normalization factor of edge likelihood
- Sample 500 negative items shared across all training examples in each minibatch
- Include “hard” negative examples:
  - Somewhat relevant to q, but not as related as i
  - Randomly sample items with Personalized PageRank score $\in$ [2000, 5000]

Figure 2: Random negative examples and hard negative examples. Notice that the hard negative example is significantly more similar to the query, than the random negative example, though not as similar as the positive example.

Loss function

- Use the concept of triangle loss
  - Maximize inner product of positive examples (q is related to i)
  - Minimize inner product of negative examples (q is unrelated to i)
- For a pair of embeddings (z_q, z_i): (q, i) $\in L$, loss function is:

$$J_{\mathcal{G}}(\mathbf{z}_q \mathbf{z}_i) = \mathbb{E}_{n_k \sim P_n(q)} \max\{0, \mathbf{z}_q \cdot \mathbf{z}_{n_k} - \mathbf{z}_q \cdot \mathbf{z}_i + \Delta\}$$

Curriculum Learning

- Using negative items achieves faster convergence
- First epoch: no negative items used → find area in parameter space with small loss
- Gradually add negative items, model focuses on learning to distinguish between highly related and somewhat related items
  - At epoch n, have n - 1 hard negative items for each item

Hit-rate and MRR performance

Method	Hit-rate	MRR
Visual	17%	0.23
Annotation	14%	0.19
Combined	27%	0.37
max-pooling	39%	0.37
mean-pooling	41%	0.51
mean-pooling-xent	29%	0.35
mean-pooling-hard	46%	0.56
PinSage	**67%**	**0.59**

Table 1: Hit-rate and MRR for PinSage and content-based deep learning baselines. Overall, PinSage gives 150% improvement in hit rate and 60% improvement in MRR over the best baseline.[5]

Accuracy performance

Methods	Win	Lose	Draw	Fraction of wins
PinSage vs. Visual	28.4%	21.9%	49.7%	56.5%
PinSage vs. Annot.	36.9%	14.0%	49.1%	72.5%
PinSage vs. Combined	22.6%	15.1%	57.5%	60.0%
PinSage vs. Pixie	32.5%	19.6%	46.4%	62.4%

Table 2: Head-to-head comparison of which image is more relevant to the recommended query image.

Speed Performance

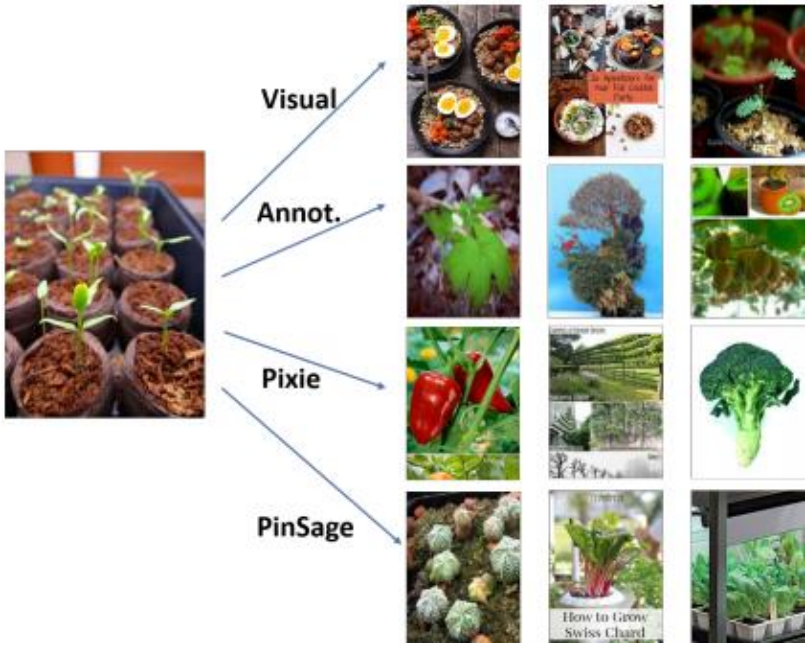
Batch size	Per iteration (ms)	# iterations	Total time (h)
512	590	390k	63.9
1024	870	220k	53.2
2048	1350	130k	48.8
4096	2240	100k	68.4

Table 3: Runtime comparisons for different batch sizes.

# neighbors	Hit-rate	MRR	Training time (h)
10	60%	0.51	20
20	63%	0.54	33
50	67%	0.59	78

Table 4: Performance tradeoffs for importance pooling.

Examples of Recommended Pins



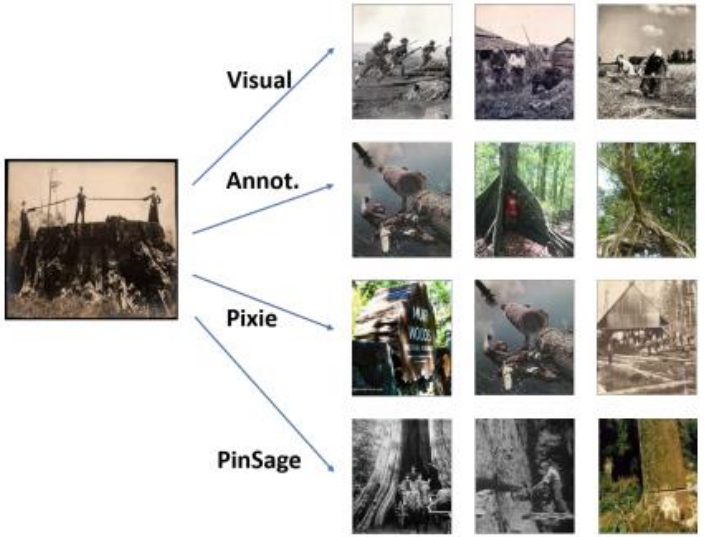


Figure 5: Examples of Pinterest pins recommended by different algorithms. The image to the left is the query pin. Recommended items to the right are computed using Visual embeddings, Annotation embeddings, graph-based Pixie, and PinSage.

Implementation

- My implementation:
  - https://github.com/alexbui91/pinsage
- Reference code:
  - https://gist.github.com/BarclayII

Future Directions

- The whole training process is based on (*q, i*) pairs, so would be interesting to improve informativeness of this kind of link
- Related boards as well, not only pins
- Weight relationship by:
  - Frequency of user's interaction with other pins that are close in the t-SNE representation
  - Some function of user statistics
- Complete code implementation:
  - Solve the heterogeneous node-attribute problem

THANK YOU !