Meysam Agah

Graph Convolutional Neural Networks for Web-Scale Recommender Systems

TABLE OF CONTENT

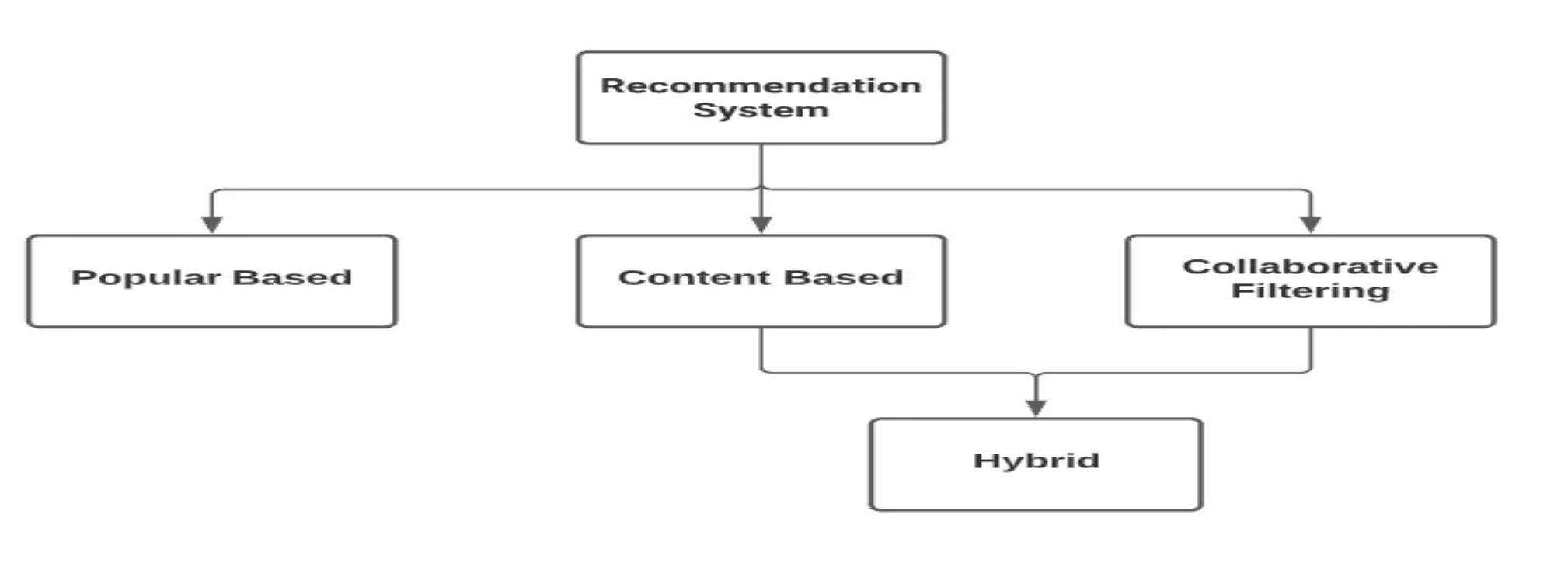
ABSTRACT

- Recent advancement in deep neural network led to better performance for recommender system but making these methods practical and scalable remains a challenge.
- Here we describe a recommendation engine developed on Pinterest.
- Our developed algorithm PinSage combines efficient random walks and graph convolutions to generate embeddings of nodes that incorporate both graph structure as well as node feature information.

ABSTRACT

- Compared to prior GCN approaches, we develop a novel method based on highly efficient random walks to structure the convolutions and design a novel training strategy that relies on harder-and-harder training examples to improve robustness and convergence of the model.
- PinSage deployed at pinterest and trained on 7.5 billion examples on a graph with 3 billion nodes representing pins and boards, and 18 billion edges.
- According to offline metrics, user studies and A/B tests, PinSage generates higher-quality recommendations than comparable deep learning and graph-based alternatives.

QUICK REVIEW ABOUT RECOMMENDER-SYSTEM METHODS



INTRODUCTION

- Deep learning methods play an important role in recommender system applications.
- New deep models can complement or even replace old recommendation algorithm like collaborative filtering.
- These algorithms has high utility because they can re-used in various recommendation tasks.

INTRODUCTION

- For example, item embeddings learned using a deep model can be used for item-item recommendation.
- Recent years significant developments made on GCN Recommender systems which is fundamental for recommendation applications.
- The idea behind GCNs is to learn how to iteratively aggregate feature information from local graph neighborhoods using neural networks

INTRODUCTION

- single **convolution** operation transforms and aggregates feature information from a node’s one-hop graph neighborhood, and by stacking multiple such convolutions information can be propagated across far reaches of a graph.
- Unlike purely content-based deep models, GCNs leverage both content information as well as graph structure. GCN-based methods have set a new standard on countless recommender system benchmarks.

INTRODUCTION

- The main challenge is to scale both the training as well as inference of GCN-based node embeddings to graphs with billions of nodes and tens of billions of edges.
- Scaling up GCNs is difficult because many of the core assumptions underlying their design are violated when working in a big data environment.

INTRODUCTION

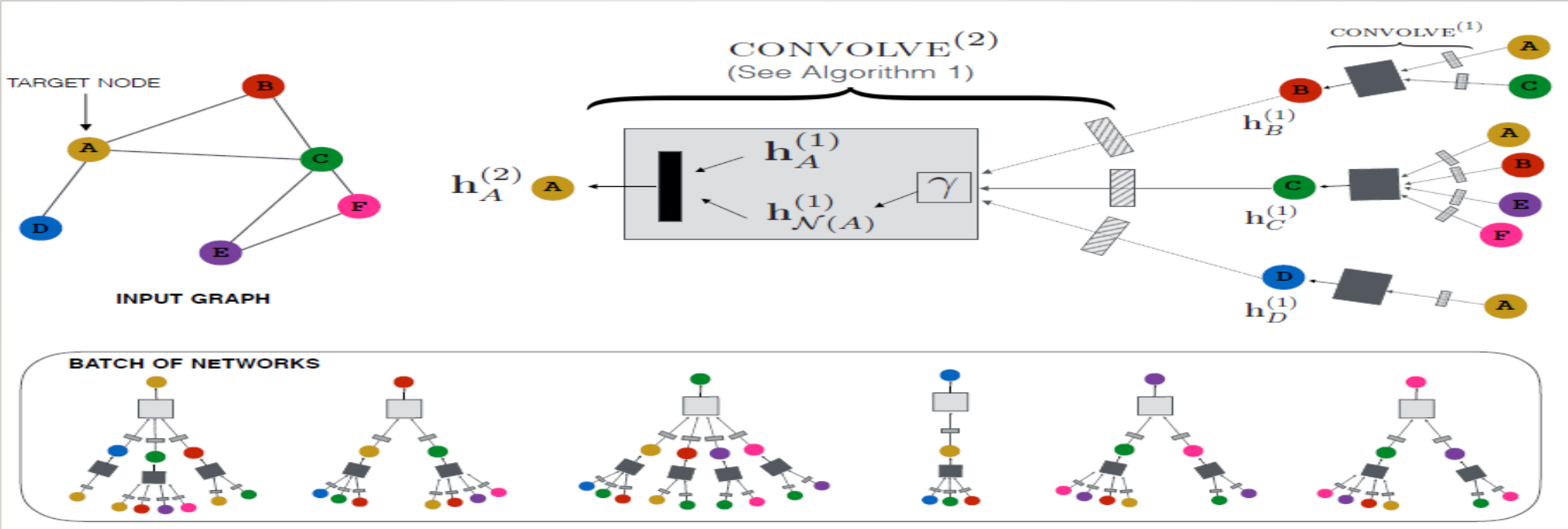


Figure 1: Overview of our model architecture using depth-2 convolutions.

INTRODUCTION

- Present work: Here we present a highly-scalable GCN framework that we have developed and deployed in production at Pinterest. Our framework, a random-walk-based GCN named PinSage, operates on a massive graph with 3 billion nodes and 18 billion edges—a graph that is 10, 000× larger than typical applications of GCNs. PinSage leverages several key insights to drastically improve the scalability of GCNs:

INTRODUCTION

- On-the-fly convolutions: Traditional GCN algorithms perform graph convolutions by multiplying feature matrices by powers of the full graph Laplacian. In contrast, our PinSage algorithm performs efficient, localized convolutions by sampling the neighborhood around a node and dynamically constructing a computation graph from this sampled neighborhood. These dynamically constructed computation graphs (Fig. 1) specify how to perform a localized convolution around a particular node, and alleviate the need to operate on the entire graph during training.

INTRODUCTION

- Producer-consumer minibatch construction: We develop a producer-consumer architecture for constructing minibatches that ensures maximal GPU utilization during model training. A large-memory, CPU-bound producer efficiently samples node network neighborhoods and fetches the necessary features to define local convolutions, while a GPU-bound TensorFlow model consumes these pre-defined computation graphs to efficiently run stochastic gradient decent.

INTRODUCTION

- Efficient MapReduce inference: Given a fully-trained GCN model, we design an efficient MapReduce pipeline that can distribute the trained model to generate embeddings for billions of nodes, while minimizing repeated computations.
- In addition to these advancements in scalability, we also introduce new training techniques and algorithmic innovations.

INTRODUCTION

- Constructing convolutions via random walks: we develop a new technique using short random walks to sample the computation graph
- Importance pooling: At this method we weigh the importance of node features in this aggregation based upon random walk similarity measures, leading to a 46% performance gain in offline evaluation metrics.
- Curriculum training: algorithm is fed harder-and-harder examples during training, resulting in a 12% performance gain.

INTRODUCTION

- At our deployed PinSage users interact with pins, which are visual bookmarks to online content
- Users organize these pins into boards, which contain collections of similar pins
- Altogether, Pinterest is the world's largest user-curated graph of images, with over 2 billion unique pins collected into over 1 billion boards.

INTRODUCTION

Compared to other deep content-based recommendation algorithms:

- In offline ranking metrics we improve over the best performing baseline by more than 40%
- in head-to-head human evaluations our recommendations are preferred about 60% of the time
- and the A/B tests show 30% to 100% improvements in user engagement across various settings.

RELATED WORK

- The notion of neural networks for graph data was first outlined in Gori et al. (2005), Scarselli et al. (2009)
- Gated Graph Sequence Neural Networks addressed limitation
- with the work of Bruna et al. (2013) approach of GCN originated.
- I terms of algorithm design this paper is related to Hamilton et al.(2017a)’s GraphSAGE algorithm

METHOD

- In this section, we describe the technical details of the PinSage architecture and training as well as a MapReduce pipeline to efficiently generate embeddings using a trained PinSage model
- The key computational workhorse of our approach is the notion of localized graph convolutions To generate the embedding for a node, we apply multiple convolutional modules that aggregate feature information from the node’s local graph neighborhood.

METHOD

- Each module learns how to aggregate information from a small graph neighborhood, and by stacking multiple such modules, our approach can gain information about the local network topology. Importantly, parameters of these localized convolutional modules are shared across all nodes, making the parameter complexity of our approach independent of the input graph size.

METHOD
PROBLEM SETUP

- Our task is to generate high-quality embeddings or representations of pins that can be used for recommendation.
- In order to learn these embeddings, we model the Pinterest environment as a bipartite graph consisting of nodes in two disjoint sets, I (containing pins) and C (containing boards).
- In addition to the graph structure, we also assume that the pins/items $u \in I$ are associated with real-valued attributes, $x_u \in R_d$

METHOD
PROBLEM SETUP

- in the case of Pinterest, we have that pins are associated with both rich text and image features. Our goal is to leverage both these input attributes as well as the structure of the bipartite graph to generate high-quality embeddings. These embeddings are then used for recommender system candidate generation via nearest neighbor lookup.
- Generally, when we describe the PinSage algorithm, we simply refer to the node set of the full graph with $V = I \cup C$ and do not explicitly distinguish between pin and board nodes, using the more general term “node” whenever possible.

METHOD
MODEL ARCHITECTURE

- Forward propagation algorithm: consider the task of generating an embedding, z_u for a node u, which depends on the node's input features and the graph structure around this node.
- The core of our PinSage algorithm is a localized convolution operation, where we learn how to aggregate information from u's neighborhood. This procedure is detailed in Algorithm 1

METHOD
MODEL ARCHITECTURE

Algorithm 1: CONVOLVE

Input : Current embedding $\mathbf{z}_u$ for node u; set of neighbor embeddings $\{\mathbf{z}_v | v \in \mathcal{N}(u)\}$, set of neighbor weights $\boldsymbol{\alpha}$; symmetric vector function $\gamma(\cdot)$

Output : New embedding $\mathbf{z}_u^{\text{NEW}}$ for node u

$\mathbf{n}_u \leftarrow \gamma\left(\{\text{ReLU}\left(\mathbf{Q}\mathbf{h}_v + \mathbf{q}\right) \mid v \in \mathcal{N}(u)\}, \boldsymbol{\alpha}\right)$;

$\mathbf{z}_u^{\text{NEW}} \leftarrow \text{ReLU}\left(\mathbf{W} \cdot \text{CONCAT}(\mathbf{z}_u, \mathbf{n}_u) + \mathbf{w}\right)$;

$\mathbf{z}_u^{\text{NEW}} \leftarrow \mathbf{z}_u^{\text{NEW}} / \|\mathbf{z}_u^{\text{NEW}}\|_2$

METHOD
MODEL ARCHITECTURE

- Importance-based neighborhoods: In PinSage in order to select neighbors to convolve in algorithm 1, we simulate random walk from node u and compute the L1-normalized visit count of nodes visited by random walk. Then neighborhood of u is defined according to top nodes with highest normalized visit counts.

METHOD
MODEL ARCHITECTURE

- Stacking convolutions: Each time we apply the convolve operation we get a new representation for a node, and we can stack multiple such convolutions on top of each other in order to gain more information about the local graph structure around node u. In particular, we use multiple layers of convolutions, where the inputs to the convolutions at layer k depend on the representations output from layer $k - 1$ and where the initial representations are equal to the input node features. Note that the model parameters in Algorithm 1 (Q, q, W, and w) are shared across the nodes but differ between layers.

METHOD
MODEL ARCHITECTURE

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 $z_u, \forall u \in \mathcal{M}$

/* 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

METHOD
MODEL ARCHITECTURE

/* 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

METHOD
MODEL TRAINING

- We train PinSage in a supervised fashion using a max-margin ranking loss. In this setup, we assume that we have access to a set of labeled pairs of items L, where the pairs in the set, (q, i) ∈ L, are assumed to be related—i.e., we assume that if (q, i) ∈ L then item I is a good recommendation candidate for query item q. The goal of the training phase is to optimize the PinSage parameters so that the output embeddings of pairs (q, i) ∈ L in the labeled set are close together.

METHOD
MODEL TRAINING

- We first describe our margin-based loss function in detail. Following this, we give an overview of several techniques we developed that lead to the computation efficiency and fast convergence rate of PinSage, allowing us to train on billion node graphs and billions training examples. And finally, we describe our curriculum-training scheme, which improves the overall quality of the recommendations.

METHOD
MODEL TRAINING

- Loss function: The basic idea is that we want to maximize the inner product of positive examples, i.e., the embedding of the query item and the corresponding related item. At the same time we want to ensure that the inner product of negative examples—i.e., the inner product between the embedding of the query item and an unrelated item—is smaller than that of the positive sample by some pre-defined margin.

METHOD
MODEL TRAINING

- The loss function for a single pair of node embeddings (z_q, z_i) : (q, i) ∈ L is thus:

$$J_G(z_q z_i) = E_{n_k \sim P_n(q)} \max\{0, z_q . z_{n_k} - z_q . z_i + \Delta\} \quad (1)$$

where $P_n(q)$ denotes the distribution of negative examples for item q, and Δ denotes the margin hyper-parameter. We shall explain the sampling of negative samples below.

METHOD
MODEL TRAINING

- Multi-GPU training with large minibatches: To make full use of multiple GPUs on a single machine for training, we run the forward and backward propagation in a multi-tower fashion. With multiple GPUs, we first divide each minibatch (Figure 1 bottom) into equal-sized portions. Each GPU takes one portion of the minibatch and performs the computations using the same set of parameters. After backward propagation, the gradients for each parameter across all GPUs are aggregated together, and a single step of synchronous SGD is performed.

METHOD
MODEL TRAINING

- Due to the need to train on extremely large number of examples (on the scale of billions), we run our system with large batch sizes, ranging from 512 to 4096. We use a gradual warmup procedure that increases learning rate from small to a peak value in the first epoch according to the linear scaling rule. Afterwards the learning rate is decreased exponentially.

METHOD
MODEL TRAINING

- Producer-consumer minibatch construction: each GPU process needs access to the neighborhood. Accessing the data in CPU memory from GPU is not efficient. So, we use a re-indexing technique to create a sub-graph $G' = (V', E')$ which will be involved in the computation of the current minibatch

METHOD
MODEL TRAINING

- The adjacency list of G′ and the small feature matrix containing only node features relevant to computation of the current minibatch are fed into GPUs at the start of each minibatch iteration, so that no communication between the GPU and CPU is needed during the convolve step, greatly improving GPU utilization.

METHOD
MODEL TRAINING

- Sampling negative items: To improve efficiency when training with large batch sizes, we sample a set of 500 negative items to be shared by all training examples in each minibatch. This drastically saves the number of embeddings that need to be computed during each training step, compared to running negative sampling for each node independently.

METHOD
MODEL TRAINING

- for each positive training example we add “hard” negative examples, i.e., items that are somewhat related to the query item q, but not as related as the positive item i. We call these “hard negative items”. They are generated by ranking items in a graph according to their Personalized PageRank scores with respect to query item q. the hard negative examples are more similar to the query than random negative examples, and are thus challenging for the model to rank, forcing the model to learn to distinguish items at a finer granularity.

METHOD
MODEL TRAINING

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.

METHOD
NODE EMBEDDINGS VIA MAPREDUCE

- many nodes are repeatedly computed at multiple layers when generating the embeddings for different target nodes. To ensure efficient inference, we develop a MapReduce approach that runs model inference without repeated computations.

METHOD
NODE EMBEDDINGS VIA MAPREDUCE

- The MapReduce pipeline has two key parts:
- (1) One MapReduce job is used to project all pins to a lowdimensional latent space, where the aggregation operation will be performed.
- (2) Another MapReduce job is then used to join the resulting pin representations with the ids of the boards they occur in, and the board embedding is computed by pooling the features of its neighbors

METHOD
NODE EMBEDDINGS VIA MAPREDUCE

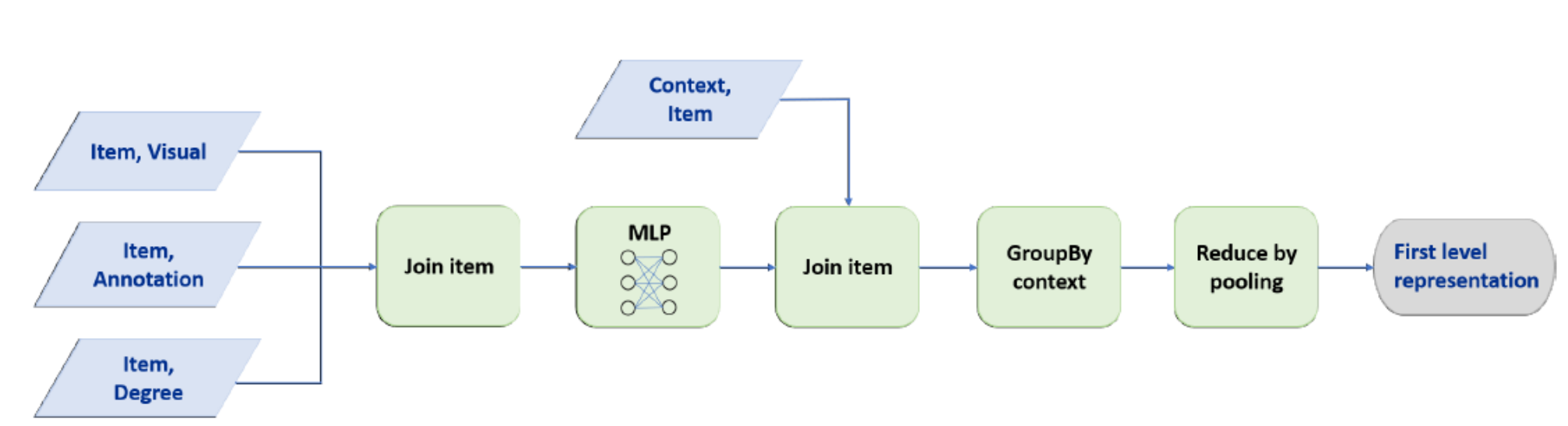


Figure 3: Node embedding data flow to compute the first layer representation using MapReduce. The second layer computation follows the same pipeline, except that the inputs are first layer representations, rather than raw item features.

METHOD
EFFICIENT NEAREST-NEIGHBOR LOOKUPS

- given a query item q, the we can recommend items whose embeddings are the K-nearest neighbors of the query item’s embedding. Approximate KNN can be obtained efficiently via locality sensitive hashing. After the hash function is computed, retrieval of items can be implemented with a two-level retrieval process based on the Weak AND operator

EXPERIMENTS

- To demonstrate the efficiency of PinSage and the quality of the embeddings it generates, we conduct a comprehensive suite of experiments on the entire Pinterest object graph, including offline experiments, production A/B tests as well as user studies

EXPERIMENTS
OFFLINE EVALUATION

- For each positive pair of pins (q, i) in the test set, we use q as a query pin and then compute its top K nearest neighbors NN_q from a sample of 5 million test pins. We then define the hit-rate as the fraction of queries q where i was ranked among the top K of the test sample
- We also evaluate the methods using Mean Reciprocal Rank (MRR), which takes into account of the rank of the item j among recommended items for query item q:

$$MRR = \frac{1}{n}\sum_{(q,i)\in L}\frac{1}{\lceil R_{i,q}/100 \rceil} \quad (2)$$

EXPERIMENTS
OFFLINE EVALUATION

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

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**

EXPERIMENTS
USER STUDIES

- In the user study, a user is presented with an image of the query pin, together with two pins retrieved by two different recommendation algorithms. The user is then asked to choose which of the two candidate pins is more related to the query pin. Users are instructed to find various correlations between the recommended items and the query item, in aspects such as visual appearance, object category and personal identity. If both recommended items seem equally related, users have the option to choose “equal”. If no consensus is reached among 2/3 of users who rate the same question, we deem the result as inconclusive.

EXPERIMENTS
USER STUDIES

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.

EXPERIMENTS
PRODUCTION A/B TEST

- Lastly, we report on the production A/B test experiments on the task of homefeed recommendations. The metric of interest is repin rate, which measures the percentage of homefeed recommendations that have been saved by the users. It means that a given pin presented to a user at a given time was relevant enough for the user to save that pin to one of their boards so that they can retrieve it later. We find that PinSage consistently recommends pins that are more likely to be re-pinned by the user than the alternative methods. Depending on the particular setting, we observe 10-30% improvements in repin rate over the Annotation and Visual embedding based recommendations.

CONCLUSION

- PinSage is a random-walk GCN. PinSage is a highly-scalable GCN algorithm capable of learning embeddings for nodes in web-scale graphs containing billions of objects.
- According to offline metrics, user studies and A/B tests all demonstrating a substantial improvement in recommendation performance by PinSage
- This work demonstrates the impact that graph convolutional methods can have in a production recommender system

REFERENCES

- [1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016).
- [2] A. Andoni and P. Indyk. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In FOCS.
- [3] T. Bansal, D. Belanger, and A. McCallum. 2016. Ask the GRU: Multi-task learning for deep text recommendations. In RecSys. ACM.
- [4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. 2009. Curriculum learning. In ICML.
- [5] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. 2003. Efficient query evaluation using a two-level retrieval process. In CIKM.

REFERENCES

- [6] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. 2017. Geometric deep learning: Going beyond euclidean data. IEEE Signal Processing Magazine 34, 4 (2017).
- [7] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. 2014. Spectral networks and locally connected networks on graphs. In ICLR.
- [8] J. Chen, T. Ma, and C. Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. ICLR (2018).
- [9] P. Covington, J. Adams, and E. Sargin. 2016. Deep neural networks for youtube recommendations. In RecSys. ACM.
- [10] H. Dai, B. Dai, and L. Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In ICML.

REFERENCES

- [11] M. Defferrard, X. Bresson, and P. Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In NIPS.
- [12] A. Van den Oord, S. Dieleman, and B. Schrauwen. 2013. Deep content-based music recommendation. In NIPS.
- [13] D. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru- Guzik, and R. P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In NIPS.
- [14] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec. 2018. Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time. WWW (2018).
- [15] M. Gori, G. Monfardini, and F. Scarselli. 2005. A new model for learning in graph domains. In IEEE International Joint Conference on Neural Networks.

REFERENCES

- [16] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L.Wesolowski, A. Kyrola, A. Tulloch,Y. Jia, and K. He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1Hour. arXiv preprint arXiv:1706.02677 (2017).
- [17] A. Grover and J. Leskovec. 2016. node2vec: Scalable feature learning for networks. In KDD.
- [18] W. L. Hamilton, R. Ying, and J. Leskovec. 2017. Inductive Representation Learning on Large Graphs. In NIPS.
- [19] W. L. Hamilton, R. Ying, and J. Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. IEEE Data Engineering Bulletin (2017).
- [20] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley. 2016. Molecular graph convolutions: moving beyond fingerprints. CAMD 30, 8.

REFERENCES

- [21] T. N. Kipf and M. Welling. 2017. Semi-supervised classification with graph convolutional networks. In ICLR.
- [22] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. 2015. Gated graph sequence neural networks. In ICLR.
- [23] T. Mikolov, I Sutskever, K. Chen, G. S. Corrado, and J. Dean. 2013. Distributed representations of words and phrases and their compositionality. In NIPS.
- [24] F. Monti, M. M. Bronstein, and X. Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. In NIPS.
- [25] OpenMP Architecture Review Board. 2015. OpenMP Application Program Interface Version 4.5. (2015).
- [26] B. Perozzi, R. Al-Rfou, and S. Skiena. 2014. DeepWalk: Online learning of social representations. In KDD.

REFERENCES

- [27] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The graph neural network model. IEEE Transactions on Neural Networks 20, 1 (2009), 61–80.
- [28] K. Simonyan and A. Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).
- [29] R. van den Berg, T. N. Kipf, and M. Welling. 2017. Graph Convolutional Matrix Completion. arXiv preprint arXiv:1706.02263 (2017).
- [30] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. 2018. GraphRNN: Generating Realistic Graphs using Deep Auto-regressive Models. ICML (2018).
- [31] M. Zitnik, M. Agrawal, and J. Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics (2018).

Thanks