2019sp-cs-8501-Deep2Read Scribe Notes: FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling

Scribe: Arshdeep Sekhon

June 1, 2019

1 Motivation

The Problem: Graph Convolutional Network requires using the entire dataset in a transductive setting to compute loss function. This is because the graph representation computation is not independent for the vertices: all need to be calculated together with neighborhood information aggregation. This incurs huge memory (requires storage of entire data in memory) and time cost, worsened by the recursive layer/cycle computations.

This Paper: The main point is to convert the non-independent computation for each vertex embedding into integrals of an embedding function on vertices *iid* sampled from a probability distribution. This can be converted into a monte carlo estimation of a loss function summed over iid observations. This sampling based loss formulation enables SGD for optimization over batches of data.

2 Method

Assume there is a (possibly infinite) graph G' with the vertex set V' associated with a probability space (V', F, P), such that for the given graph G, it is an induced subgraph of G' and its vertices are iid samples of V' according to the probability measure P. All vertices share the same probability measure P that defines a sampling distribution. A regular GCN:

$$\tilde{H}^{(l+1)} = \hat{A}H^{(l)}W^{(l)}, \quad H^{(l+1)} = \sigma(\tilde{H}^{(l+1)}), \quad l = 0, \ldots, M-1, \quad L = \frac{1}{n}\sum_{i=1}^{n} g(H^{(M)}(i,:)). \tag{1}$$

M denotes the number of cycles or layers. This paper:

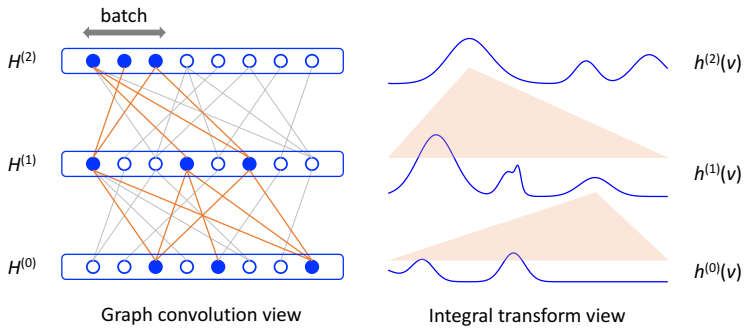


Figure 1: Two views of GCN. (a) graph convolution view (b) integral transform view

$$\tilde{h}^{(l+1)}(v) = \int \hat{A}(v,u)h^{(l)}(u)W^{(l)}\,dP(u), \quad h^{(l+1)}(v) = \sigma(\tilde{h}^{(l+1)}(v)), \quad l = 0,\ldots,M-1, \tag{2}$$

$$L = \mathrm{E}_{v\sim P}[g(h^{(M)}(v))] = \int g(h^{(M)}(v))\,dP(v). \tag{3}$$

$P(u)$ is the sampling probability for the vertices. To reduce variance of the Monte Carlo Estimator, use importance sampling $P(u) = q(u)$ based on the degree distribution, instead of uniform distribution:

$$q(u) = \|\hat{A}(:,u)\|^2 / \sum_{u'\in V} \|\hat{A}(:,u')\|^2, \quad u \in V$$

2.1 The batched algorithm

Algorithm 1 FastGCN batched training (one epoch), improved version

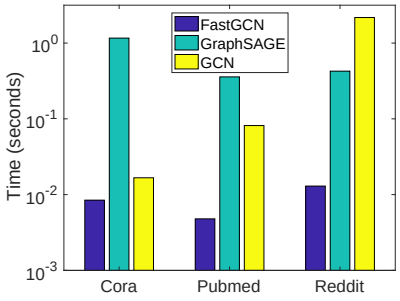
For each vertex u, compute sampling probability $q(u) \propto \|\hat{A}(:,u)\|^2$
for each batch **do**
  For each layer l, sample t_l vertices $u_1^{(l)}, \ldots, u_{t_l}^{(l)}$ according to distribution q
  for each layer l **do** ▷ Compute batch gradient ∇L_{batch}
    If v is sampled in the next layer,

$$\nabla \tilde{H}^{(l+1)}(v,:) \leftarrow \frac{1}{t_l} \sum_{j=1}^{t_l} \frac{\hat{A}(v,u_j^{(l)})}{q(u_j^{(l)})} \nabla \left\{ H^{(l)}(u_j^{(l)},:) W^{(l)} \right\}$$

  end for
  $W \leftarrow W - \eta \nabla L_{\text{batch}}$ ▷ SGD step
end for

GraphSAGE authors propose restricting the immediate neighborhood size for each layer to reduce computation and memory. In the worst case, the size of the expanded neighborhood is the product of the t_l's. On the other hand, FastGCN samples vertices - the total number of involved vertices is at most the sum of the t_l's.

3 Results



Micro F1 Score

	Cora	Pubmed	Reddit
FastGCN	0.850	0.880	0.937
GraphSAGE-GCN	0.829	0.849	0.923
GraphSAGE-mean	0.822	0.888	0.946
GCN (batched)	0.851	0.867	0.930
GCN (original)	0.865	0.875	NA

Figure 2: Per-batch training time in seconds (left) and prediction accuracy (right).

4 Conclusion

This paper presents FastGCN, a method to convert previous GCN loss into a sampling of vertices based loss formulation. This allows batched and inductive setting for GCN, reducing time cost.