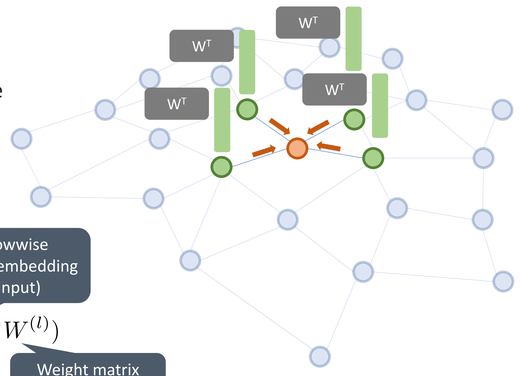
FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling

Jie Chen, Tengfei Ma, Cao Xiao. IBM Research

GCN (Graph Convolutional Net)

Graph convolution is similar to image convolution: Sum up the contributions of the neighbors to update the representation of itself.



Rowwise node embedding (output)

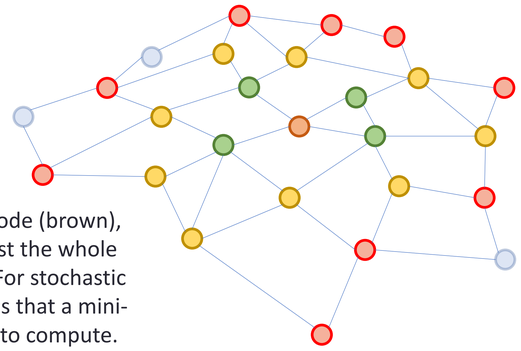
Rowwise node embedding (input)

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)})$$

Adjacency matrix (normalized)

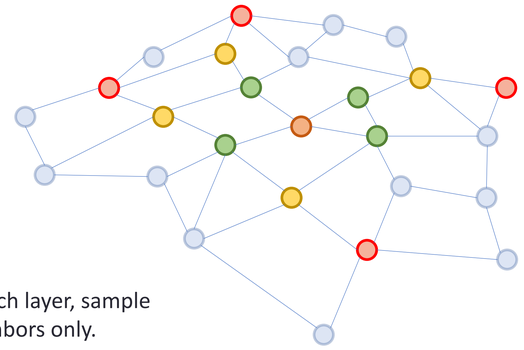
Weight matrix (model parameter)

Scalability Challenge



Starting from a single node (brown), after a few layers, almost the whole graph will be touched. For stochastic optimization, this means that a mini-batch is very expensive to compute.

Proposed Solution: FastGCN



We propose that for each layer, sample a fixed number of neighbors only.

Empirical Risk Minimization

- In standard learning theory, we learn the parameters of a model by minimizing the risk:

$$\text{expected risk } f(w) = \mathrm{E}[\ell(w;\xi)], \qquad \text{empirical risk } f_{\text{emp}}(w) = \frac{1}{n}\sum_{i=1}^{n}\ell(w;\xi_i).$$

loss function; model parameter; training example and label

- For large models, the minimization is typically done through stochastic gradient descent: Update the model with one training example (or a mini-batch of examples) each time:

$$w_{k+1} = w_k - \gamma_k \nabla \ell(w_k, \xi_{k_i})$$

- The problem for graphs is that one node is related to many other nodes, hence the sample gradient is very expensive to compute.

Generalize to Integral Transform

- Let the graph G have a vertex set V
- We generalize it to an infinite graph G' with vertex set V', and an associated probability space (V', F, P) …
- … such that G is induced from G' and the nodes in V are iid samples of V' according to probability measure P
- Now, one layer of GCN …

embedding vectors

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)})$$

- … becomes

embedding function

$$h^{(l+1)}(v) = \sigma\left(\int \hat{A}(v,u)h^{(l)}(u)W^{(l)}\,dP(u)\right)$$

Sampling Each Layer

- We may perform Monte Carlo approximation for the integral in each layer. For the *l*-th layer, use t_l iid samples ~ P:

$$h^{(l+1)}_{t_{l+1}}(v) = \sigma\left(\frac{1}{t_l}\sum_{j=1}^{t_l}\hat{A}(v,u_j^{(l)})h^{(l)}_{t_l}(u_j^{(l)})W^{(l)}\right)$$

- In practice, this sampling means using t_l iid nodes ~ Uniform to approximate matrix multiplication (because vertex set is already iid ~ P)

$$H^{(l+1)}_{t_{l+1}}(v,:) = \sigma\left(\frac{n}{t_l}\sum_{j=1}^{t_l}\hat{A}(v,u_j^{(l)})H^{(l)}_{t_l}(u_j^{(l)},:)W^{(l)}\right)$$

- Furthermore, for variance reduction, we may use t_l iid nodes ~ Q to perform sampling. The optimal distribution Q is not efficient to compute, but setting Q to be proportional to the squared column norms of the normalized adjacency matrix A works fairly well in practice

Theoretical Result

- Recall that we approximate the loss function f by sampling the *l*-th layer with t_l nodes:

$$f = \mathrm{E}_{v\sim P}[\ell(w;h^{(M)}(v))] \xrightarrow{\text{approx}} f_{t_0,t_1,\ldots,t_M} = \frac{1}{t_M}\sum_{i=1}^{t_M}\ell(w;h^{(M)}_{t_M}(u_i^{(M)}))$$

- Theorem: The approximator is strongly consistent:

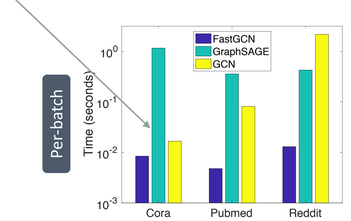
$$\lim_{t_0,t_1,\ldots,t_M\to\infty} f_{t_0,t_1,\ldots,t_M} = f \quad \text{with probability one}$$

- The result may be easily generalized to the gradient:

$$\lim_{t_0,t_1,\ldots,t_M\to\infty} \nabla f_{t_0,t_1,\ldots,t_M} = \nabla f \quad \text{with probability one}$$

Experimental Results

Optimizing the implementation of GraphSAGE for small graphs improves running time to here



batch size: 400 100 400

Table 4: Total training time (in seconds).

	Cora	Pubmed	Reddit
FastGCN	2.7	15.5	638.6
GraphSAGE-GCN	72.4	259.6	3318.5
GCN (batched)	6.9	210.8	58346.6
GCN (original)	1.7	21.4	NA

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

out of memory

Experimental Results

Importance sampling is consistently better