My Notes

Fan Feng

1 Graph Convolutional Autoencoder on Large Graphs

(Inspired by *Inductive Representation Learning on Large Graphs* and *FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling*)

Traditional GCN:

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

in which

$$\tilde{A} = D^{-1/2}AD^{-1/2}$$

is the normalized adjacency matrix and σ is the activation function. However, when A is a large matrix, the computation is non-trivial and has a complexity of $O(n^3)$ *[fastest: $O(n^{2.37})$]* for each layer. Thus, sampling methods are essential for applying GCN on really large graphs.

For SGD senario, the loss function should be an expectation w.r.t. the data distribution D:

$$L = \mathrm{E}_{x\sim D}[g(W;x)]$$

And in practice, an average is applied to replace expectation and gradient is also calculated on this:

$$L_{\mathrm{emp}} = \frac{1}{n}\sum_{i=1}^{n} g\left(W;x_i\right), \quad x_i \sim D, \forall i$$

Here, W denotes the model parameter to be optimized and x is the input.

For GCN, we can also write the layers in the following format:

$$h^{(l+1)}(v) = \sigma(\sum_{u} \tilde{A}(v,u)h^{(l)}(u)W^{(l)})$$

in which v, u are nodes of the graph, and $h^{(l)}(u)$ indicates node u's attributes in the l-th layer. $\tilde{A}(v,u)$ denotes the position (v,u)'s value of matrix $\tilde{A}$.

Considering estimating the summation in the Monte Carlo manner, we can pick up one node from the $(l+1)$-th layer (w.l.o.g., choosing v), then randomly choose t_l nodes $(u_1^{(l)},\ldots,u_{t_l}^{(l)})$ from layer l with replacement, in which $P(choosing\ u_j) = \frac{\tilde{A}(v,u_j)}{\sum_{u_j}\tilde{A}(v,u_j)}$:

$$\tilde{h}^{(l+1)}(v) := \frac{\sum_{u^{(l)}}\tilde{A}\left(v,u^{(l)}\right)}{t_l}\sum_{j=1}^{t_l}\tilde{h}^{(l)}\left(u_j^{(l)}\right)W^{(l)}$$

in which $\tilde{h}^{(l)}(u)$ is the estimator of node v's attributes for the l-th layer when sampling t_l times.

And obviously **(is it right?)**, $\tilde{h}^{(l+1)}(v)$ is an unbiased and consistent estimator of $h^{(l+1)}(v)$.

Thus, the gradient (w.r.t. $W^{(l)}$) $\nabla \tilde{H}^{(l+1)}(v)$ can be estimated by:

$$\nabla \tilde{h}^{(l+1)}(v) \sim \sum_{j=1}^{t_l} \nabla \{ \frac{\sum_{u^{(l)}} \tilde{A}\left(v, u^{(l)}\right)}{t_l} \left[\tilde{h}^{(l)}\left(u_j^{(l)}\right) W^{(l)} \right] \}$$

Thus we follow this strategy:

- Picking $b_{(l+1)}$ nodes for layer $(l+1)$ as a batch
- For each node v in this batch, sample t_l nodes from all its neighbors with probability $P(choosing\ u_j) = \frac{\tilde{A}(v,u_j)}{\sum_{u_j} \tilde{A}(v,u_j)}$ (with replacement)
- Calculate $\sum_{j=1}^{t_l} \nabla \{ \frac{\sum_{u^{(l)}} \hat{A}\left(v,u^{(l)}\right)}{t_l} \left[\tilde{h}^{(l)}\left(u_j^{(l)}\right) W^{(l)} \right] \}$ as an estimator of node v's gradient.
- Take average for the batch. Then do BP: $W \leftarrow W - \eta \nabla L_{\text{batch}}$

During the sampling procedure, if we want to better store the network structure for quick probability query, k-NN or ϵ-NN might be reasonable solutions.