INSTRUCTION

Graphs are ubiquitous in the real world, e.g., social networks, molecular structures, and world wide webs. Recent years, the machine learning community has made substantial efforts on many graph-based learning tasks, such as node classification and link prediction. Unlike Convolutional Neural Networks (CNN) which perform the end-to-end learning on grid-structured images, graph-based models have to face the data where nodes are arbitrarily connected with others. There have been several works that generalize convolutional networks on graphs, e.g., Burna et al. (2014); Duvenaud et al. (2015); Kearnes et al. (2016); Defferrard et al. (2016). Graph Convolutional Networks (GCN) (Kipf & Welling, 2016) apply the adjacency matrix of the graph as a filter that directs information to propagate over the edges. The feedforward process is formulated as

\[ H^{l+1} = \sigma(\hat{A}H^lW^l) \]  

where \( \hat{A} \) denotes the normalized adjacency matrix, \( H^l \) represents the node embeddings in \( l \)-th layer, \( W^l \) is the learnable weights of the model, and \( \sigma \) is nonlinear activation. As the it performs propagation at layer \( l = 1, ..., L \), the embedding of each node is updated by neighborhood mixing, which is equivalent to the pooling mechanism in convolution. GCN and its many variations aim to learn meaningful vector representations for nodes and edges, which are further fed to downstream generic models on some classification or regression tasks.

RELATED WORK

There are several issues with GCNs. First, training GCNs requires the entire graph as input, thus it is a transductive model. Lack of inductivity isolates the model from many applications which only allow a subset of the data visible during training (Yang et al, 2016). Second, the poor scalability makes it computationally prohibitive to apply the model on hefty datasets. To circumvent these problems, many sample-based approaches have come into rescue.

GraphSAGE (Hamilton et al., 2017). The key idea of this approach is to aggregate feature information from a node’s neighborhood. First, instead of caching the entire graph in training, it creates minibatches of nodes and edges from the adjacency matrix by traversing and batching a relatively large number of nodes and edges from the adjacency matrix at each layer \( l \). Then at each layer of propagation, it uniformly samples a fixed-size set of neighbors from \( B_l \) to keep the computational footprint of each batch fixed. The authors proposed several trainable aggregator functions, e.g., Mean aggregator, LSTM aggregator, and Pooling aggregator, to perform feature mixing and update the hidden state of each node accordingly. The GraphSAGE architecture is an inductive model and can take advantage of the SGD and other algorithms for batch training, which is naturally scalable.

FastGCN (Chen et al, 2018). Similar to the previous one, this approach also builds minibatches prior to training. However, it is allegedly the choice of sampling strategies that boosts the predictive performance so well. Specifically, unlike uniform sampling used by GraphSAGE, this approach samples nodes from the batched candidates according to some pre-computed distribution \( q(u) \). In Proposition 4 of the paper, the distribution \( q(u) \) is obtained by

\[ q(u) = \frac{\|\hat{A}(\cdot, u^{(i)})\|^2}{\sum_{u' \in V} \|\hat{A}(\cdot, u')\|^2}, \quad u \in V \]  

(2)
where $\hat{A}$ is the normalized adjacency matrix of the batch, and $V$ is the node set. The sampling distribution $q$ remains unchanged for all layers as a constant factor. A similar GCN module then performs convolution on the sampled nodes and the model is trained via SGD.

Apart from the sample-based methods, there are also some additional components that can be plugged into the aforementioned architectures.

**Attention mechanism** (Velickovic et al., 2018). As Eq. 1 shows, GCNs take the influence from neighbors equally since the values on each row of $\hat{A}$ are the same. The graph attention networks, on the other hand, add an attention layer before the convolution, evaluating the importance of a node’s neighbors. It assigns each neighbor a normalized score, which is used in subsequent weighted convolution layer. Ideally, the attention mechanism should be able to identify a partial of neighbors which share the characteristics with the center node on some similarity metrics.

**THIS PROJECT**

In the existing approaches, the sampling process acts as a batch generator, which is in fact isolated from the learning phase. Here, we propose a new method that integrates a attention-based sampler with the convolution module, in which we can train the two parts jointly. As Fig.1 shows, we create a candidate pool from a node’s higher-order neighbors. The sampler is expectedly to select the nodes from the pool based on distributions learned from the pool itself, then feed the sampled nodes to a convolution module to further learn the representation of the center node. On the other hand, considering that the sampled nodes no long carry any connectivity features (edges are dumped), we will add a recurrent network that incorporates the sequence of path into the attention mechanism for higher-order neighbors. Finally, we will try out different combinations of the recurrent layer and sampler to explore the insight of the model.

**REFERENCE**


![Figure 1 Sampling from higher-order neighbors.](image)