Network Embeddings [1]

Presented by **Ting-An Chen**

Advisor: De-Nian Yang, Ming-Syan Chen

Dec. 12, 2019

Outline

- Introduction - Network embedding (NE)
- Classification
- Classic works
  - DeepWalk [2]
  - LINE [4]
  - Node2vec [5]
  - SDNE [6]
  - GCN [7]
- Conclusion
- Reference

Introduction

- Network
  - Diverse information formed as network structures
  - Tasks - to predict missing information, such as

    - Node property -> classification

    - Edge property -> link prediction

    - Community -> clustering
  - Problem - abundant information and complex inference
  - Technique - network embeddings

Introduction

- Network embeddings (NEs)
  - A.k.a. a procedure of information compression
  - Notion - to find a mapping function to convert each node to a low-dimensional latent representation

Introduction

- Characteristics of network embeddings
  - **Adaptability** - networks are evolving, but embeddings should not be learned again and again
  - **Scalability** - embedding algorithms should be able to process large-scale networks in a short time period
  - **Community aware** - the distance between latent representations reflects the relationship of nodes in real networks
  - **Low dimensional** - compress the information and speed up the inference
  - **Continuous** - continuous latent space shows more robustness than discrete one

Classification of Network Embeddings

- Attributes info
  - Non-attributed v.s. Attributed
- Labels (ground truth)
  - Unsupervised v.s. Semi-supervised v.s. Supervised
- Node properties
  - Homogeneous (unique) v.s. Heterogeneous (multi-types)

DeepWalk [2]

- DeepWalk – the 1ST work of graph embedding for deep learning
- Ideas came from SkipGram [3]
- SkipGram
  - A word embedding (word2vec, word to vector) method in NLP (Neural Language Processing)
  - Idea – to learn the embedding of a word from its context
  - Assumption – words with same context are similar in some extent

Dog	climbed	the	tree

Cat	climbed	the	tree

DeepWalk [2]

- SkipGram
  - Dataset – a sentence, e.g. "**The dog barked at the mailman**"
    - Center word: **dog**
  - Context words
    - Skip_window: number of words far from center words are considered
      - E.g. skip_window = 2, then ['**The**', '**barked**', '**at**'], context words, are considered
    - Num_skips: number of context words sampled to learn the embedding of center word, **dog**
      - E.g. num_skips = 2, randomly sample 2 words in ['**The**', '**barked**', '**at**'], such as ['**barked**', '**at**']
  - Initial vectors
    - One-hot encoding of all words in dataset

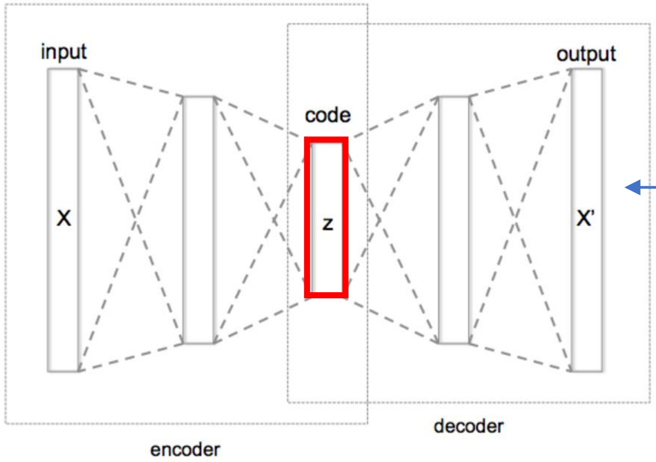
	dog	apple	…	cat
The	0	0		0
dog	1	0	…	0
…				

DeepWalk [2]

- SkipGram
  - Center word: **dog** - Input
  - Sampled context words: [‘**barked**’, ‘**at**’] - Outputs
  - Initial vectors: one-hot encoding
  - Training pair: (input word one-hot, output word one-hot), e.g. (**dog** one-hot, **barked** one-hot) and (**dog** one-hot, at one-hot)

DeepWalk [2]

- SkipGram
  - Training pair: (x, y), e.g. (**dog** one-hot, **barked** one-hot) and (**dog** one-hot, `at` one-hot)
  - Model - autoencoder



Ground truth

Min. Loss function, L
e.g. MSE,
cross entropy, etc.

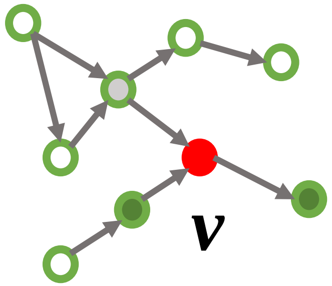
$z = W\mathrm{x}$

$x' = W'\mathrm{z}$

Gradient descent to update the weight matrix:
W* <- W* - η * ∂L/∂W
W'* <- W'* - η * ∂L/∂W'

DeepWalk [2]

- Skip-gram v.s. DeepWalk

	Skip-gram	DeepWalk
Data	**Sentences**, e.g. “The dog **barked at** the mailman”	**Graph** with nodes and edges 

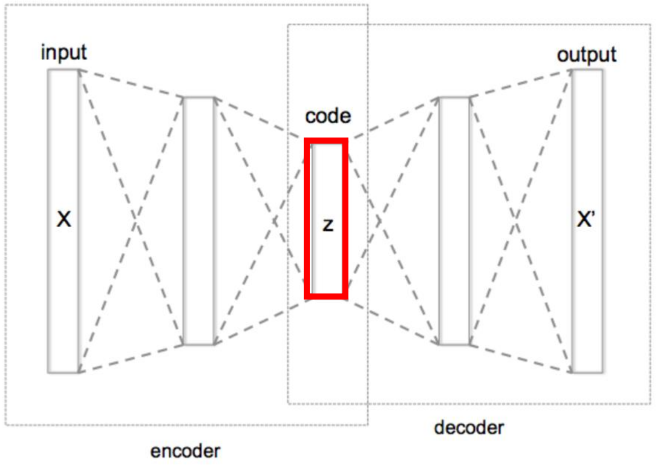
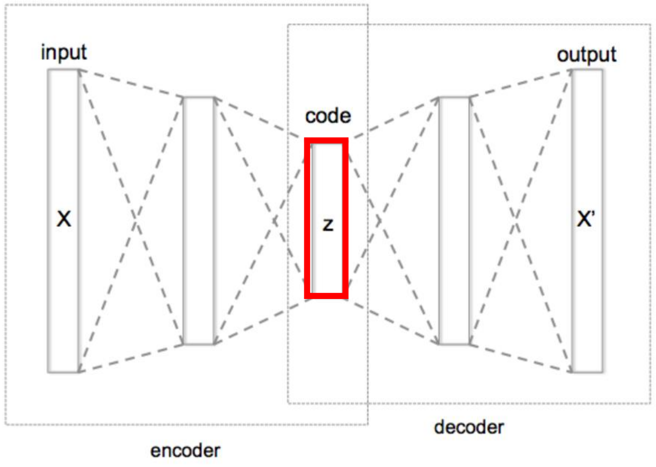
DeepWalk [2]

- Skip-gram v.s. DeepWalk

	Skip-gram	DeepWalk
Random Walks	x	For each vertex ***v***, Times of walks ***r*** Walk length ***t***
Window	Skip_window = 2 [‘The’, ‘barked’, ‘at’] Num_skips = 2 e.g. [‘barked’, ‘at’]	Window size ***w***
Initial vector	One-hot vector of each word	One-hot vector of each node

DeepWalk [2]

- Skip-gram v.s. DeepWalk

	Skip-gram	DeepWalk
Model & Training		

DeepWalk = Random Walk + Skip-gram

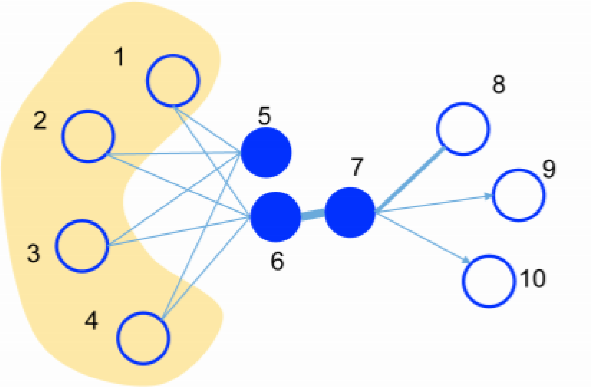
DeepWalk [2]

- DeepWalk - Summary
  - Objective - to find node embeddings by means of the info of neighbors, neighbors of neighbors, etc.
  - Methods - Random walks + Skip-gram
  - Random walks
    - To sample some nodes as a node embedding's info
    - DFS (Depth First Search)
    - Given #walks per vertex, walk length and window size
    - "Truncated" random walks in specific, due to the limitation of window size
  - Skip-gram
    - Nodes indexed are transformed as one-hot vectors
    - Autoencoder

LINE [4]

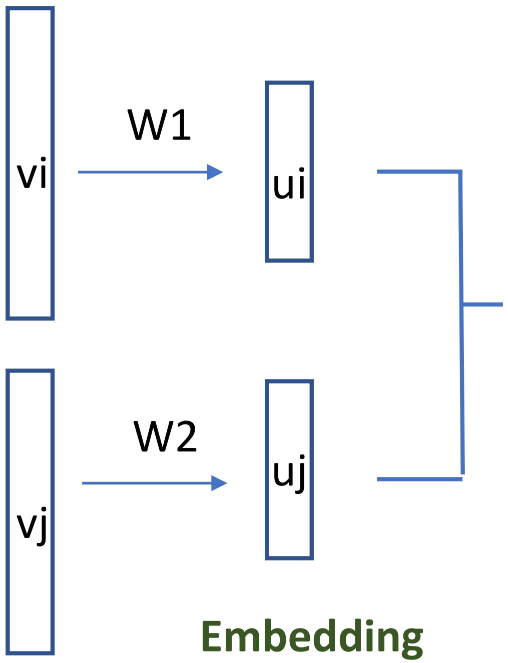
- Deficiency of DeepWalk
  - Random walks rather than considering the weights on edges (*)
  - Low efficiency for large scale networks (**)

- Line: Large-scale Information Network Embedding
  - Notions
    - Considering 1ST order and 2ND order proximity (**) [BFS]
    - Edges sampling (positive / negative sample) (*)
  - Data
    - Pairs of nodes with 1ST order or 2ND order relationship
    - Sampled following the weights on edges (positive/negative sampling)



LINE [4]

- Model



Embedding

One-hot

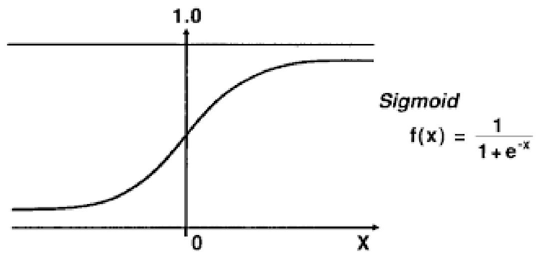
Loss function (KL-divergence with some constants omitted)

$$-\sum_{j \in N(i)} w_{ji} \log p_1(v_j, v_i) \quad \text{OR} \quad -\sum_{j \in N(i)} w_{ji} \log p_2(v_j | v_i).$$

Output

Sigmoid (Dot-product)

w_{ji}



Ground truth

First-order

1, if nodes linked

0, o.w.

$p_1(v_j, v_i)$.

Second-order

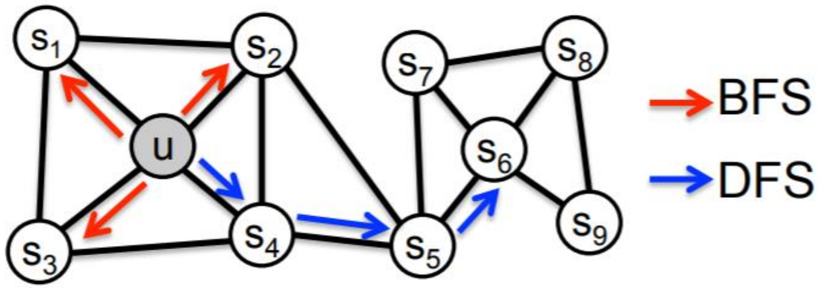
1, if co-friends exist

0, o.w.

$p_2(v_j|v_i)$

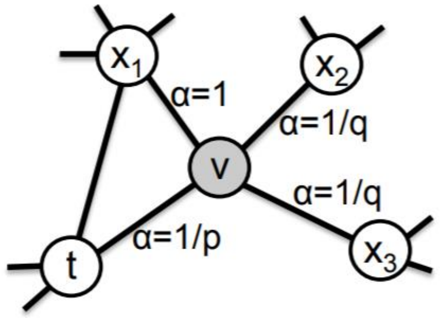
node2vec [5]

- BFS
  - Homophily – same community
  - E.g. LINE
- DFS
  - Structural equivalence – similar structure but not the same community
  - E.g. DeepWalk



node2vec [5]

- Node2vec Walks – unbiased random walks
  - Combine with the properties of BFS (community) and DFS (structure)
  - Designs
    - 0. Current step (v)
    - 1. Can return back to the last step (t) – p: return parameter
    - 2. Next step (x) – q: in-out parameter
    - 3. Shortest path in degrees of nodes (a, b) =: d(a, b)
    - 4. α = 1/p, if d(t, x) = 0, i.e., x = t [return mechanism]
    - 5. α = 1, if d(t, x) = 1 [BFS mechanism (community)]
    - 6. α = 1/q, if d(t, x) = 2 [DFS mechanism (structure)]

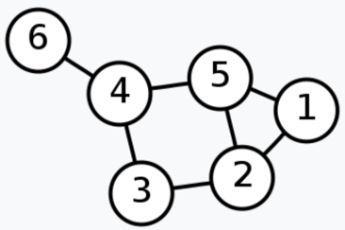


$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$$

SDNE [6]

- SDNE: Structural Deep Network Embedding
  - Notion 1.
    - 1^{ST} order – neighbors
    - 2^{ND} order – co-neighbors
  - Notion 2. Autoencoder – similar to DeepWalk
  - Notion 3. Initial vectors – vectors from the adjacency matrix (info of neighbors) rather than one-hot vectors

Labelled graph	Degree matrix	Adjacency matrix
	$\begin{pmatrix} 2&0&0&0&0&0\\ 0&3&0&0&0&0\\ 0&0&2&0&0&0\\ 0&0&0&3&0&0\\ 0&0&0&0&3&0\\ 0&0&0&0&0&1 \end{pmatrix}$	$\begin{pmatrix} 0&1&0&0&1&0\\ 1&0&1&0&1&0\\ 0&1&0&1&0&0\\ 0&0&1&0&1&1\\ 1&1&0&1&0&0\\ 0&0&0&1&0&0 \end{pmatrix}$

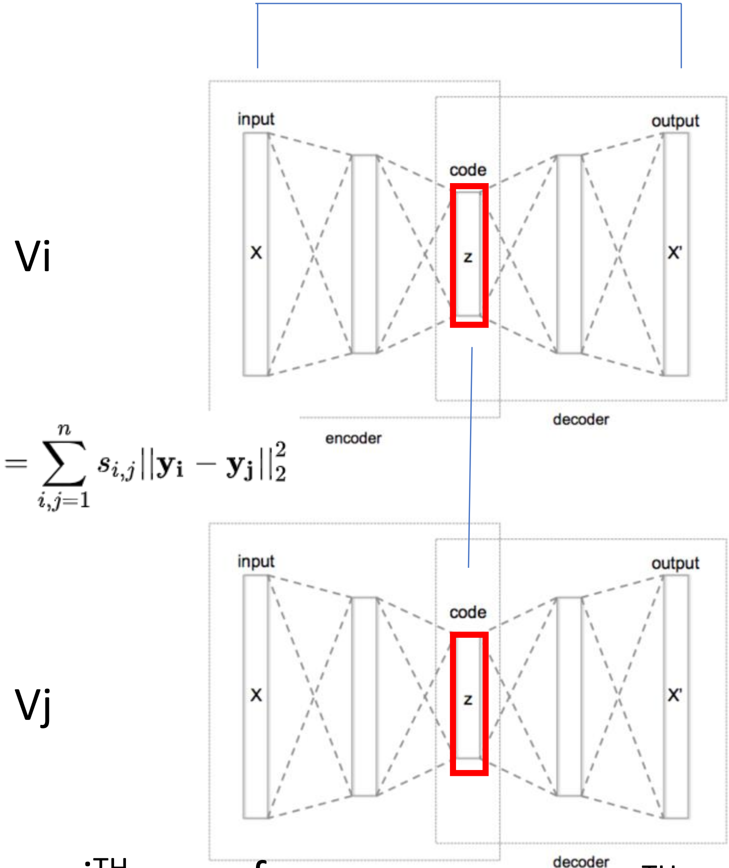
https://en.wikipedia.org/wiki/Laplacian_matrix

SDNE [6]

- Model

$$L_{2nd} = \sum_{i=1}^{n} ||(\hat{x}_i - x_i) \odot \mathbf{b_i}||_2^2 = ||(\hat{X} - X) \odot B||_F^2$$

If $s_{i,j} = 0, b_{i,j} = 1$, else $b_{i,j} = \beta > 1$.



$$L_{1st} = \sum_{i,j=1}^{n} s_{i,j}||\mathbf{y_i}^{(K)} - \mathbf{y_j}^{(K)}||_2^2 = \sum_{i,j=1}^{n} s_{i,j}||\mathbf{y_i} - \mathbf{y_j}||_2^2$$

i^TH row of adjacency matrix, S

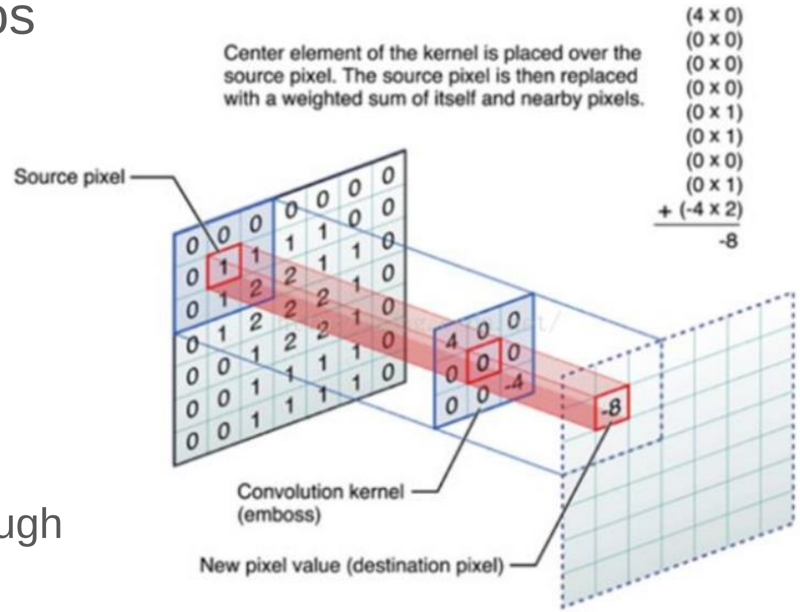
i^TH row of adjacency matrix, S

Sparsity of networks
➔ #non-zero >> #zero elements
➔ prone to reconstruct as zero matrix (X_hat)
➔ Larger penalty for non-zero elements of (X_hat - X)

$$L_{mix} = L_{2nd} + \alpha L_{1st} + \nu L_{reg}$$

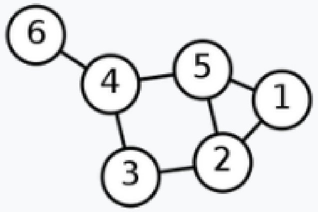
GCN [7]

- GCN: Graph Convolution Network
- Before GCN, what is CNN (Convolutional Neural Network)
  - Be used to extract the features of images, or even videos
  - Notion 1. Local characteristics
    - Important info is in local, rather than in the whole image
    - ➔ Kernels (filters) trained to find out the common local features
  - Notion 2. Uncertain location of important info
    - Info can be in any location of an image
    - ➔ Sliding stride to determine the strides when kernels slide through
  - Features what CNN extracts are in Euclidean space
  - How about the feature extraction in non-Euclidean space, such as graph?



GCN [7]

- GCN: Graph Convolution Network
  - Matrix as the representative of a graph
    - Degree matrix, D – number of links to the other nodes
    - Adjacency matrix, A – connection to the other nodes
    - Laplacian matrix (simple version) – L = D – A
      - In Physics, it can be used to show the energy loss
      - In social network, it shows message propagation

Labeled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

GCN [7]

https://purelyvivid.github.io/2019/07/07/GCN_1/

- A GCN layer can be defined as (can be any other forms) $H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$
  - $Y = LX = DX - AX$
    - L = D – A, Laplacian matrix
    - X, node embeddings
    - DX: messages each node owns before the propagation
    - AX: for each node, the amount of messages will be taken away from the neighbors
  - $\hat{A} = A + I$
    - Self-owned info
  - Local aggregation similar to CNN
    - However, number of local connections of each node for GCN is different. ➔ Normalization is in need.
    - $L^{rw} = D^{-1}L$ - algorithmic average
    - $L^{sym} = D^{-0.5}LD^{-0.5}$ - geometric average

Conclusion

- Aggregate neighbors’ or other nodes’ info
- Random walks and its variants ➔ graph Laplacian
- Limitation of GCN – memory and training time
    - GraphSAGE [8]
    - Cluster-GCN [9]

- What has not been mentioned?
    - Nodes with attribute info
    - Edge embeddings
    - Heterogeneous network embeddings

Reference

[1] H. Chen, B. Perozzi, R. Al-Rfou, and S. Skiena. A tutorial on network embeddings. arXiv preprint arXiv:1808.02590, 2018.

[2] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 701–710. ACM, 2014.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. CoRR, abs/1301.3781, 2013.

[4] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Largescale information network embedding. In Proceedings of the 24th International Conference on World Wide Web, pages 1067–1077. ACM, 2015.

[5] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 855–864. ACM, 2016.

Reference

[6] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1225–1234. ACM, 2016.

[7] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. of ICLR, 2017.

[8] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In NIPS.

[9] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, “Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks,” in Proc. of KDD. ACM, 2019