Homework 3: Graph Models and Generative Models

Deep Learning (84100343-0)

Tsinghua University

Date: December 19, 2022

1 Graph Neural Networks (GNN) and Node2Vec

Task A

We provide implementations of **GCN**, **GAT** and **Node2Vec** for you. Read through and run `gcn.py`, `gat.py`, `node2vec.py`, and report the performance of these methods. [**20pts**]

Answer

I read and ran the scripts (on GPU). Table 1 shows the running time, and Figure 1 visualizes the training procedures.

The following characteristics can be found:

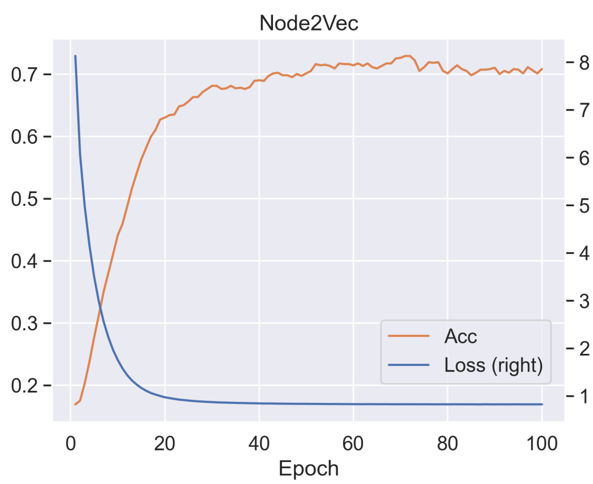
1. The training procedure of Node2Vec is unsupervised, with its code structure quite different from GCN and GAT, two supervised methods.
2. There is no `edge_attr` in the *Cora* dataset, so GCN and GAT actually use the same information.
3. In terms of clock time, GCN and GAT run significantly faster than Node2Vec, as random walks are computationally more expensive. Node2Vec seems to benefit from parallel evaluation.
4. In terms of epochs, three models are similar, with GAT converging slightly faster (~20 epochs). The test accuracy of GCN and GAT can reach 0.8+, while Node2Vec is lower (~0.7). Overfitting is evident.
5. As shown in Figure 1d, Node2Vec generates node embeddings highly correlated to class labels.

Model	**User time(s)**	**Wall time(s)**
GCN	3.34	4.56
GAT	1.59	2.51
Node2Vec	101.28	22.10

Table 1: Script running time

(a) Graph Convolutional Network (GCN)

(b) Graph Attention Network (GAT)



(c) Node2Vec: Biased Walks

(d) Node2Vec Visualization

Figure 1: Learning curves of graph models

Task B

Implement **DeepGCN** or **GIN** (You **only** need to implement **one** of them to get full grades), and report the performance. (**Hint**: PyG has implemented basic layers for you) [**20pts**]

Answer

I have chosen to implement **DeepGCN**, which originated from **DeepGCN** by Li et al. (2019) and **DeeperGCN** by Li et al. (2020). Inspired by CNN, DeepGCN tackles over-smoothing by adding skip connections, dense connections and dilated convolutions. DeeperGCN made further improvement for deeper models:

- ⋆ GENeralized Graph Convolution (`GENConv`) for message aggregation:

$$\mathbf{x}_i' = \mathrm{MLP}\left(\mathbf{x}_i + \mathrm{AGG}\left(\left\{\mathrm{ReLU}\left(\mathbf{x}_j + \mathbf{e_{ji}}\right) + \epsilon : j \in \mathcal{N}(i)\right\}\right)\right)$$

  where AGG can be `SoftMax` or `PowerMean`.

- ⋆ Pre-activation residual connections (`"res+"`):

$$\text{Normalization} \longrightarrow \text{Activation} \longrightarrow \text{Dropout} \longrightarrow \text{GraphConv} \longrightarrow \text{Res}$$

- ⋆ Message normalization (`MsgNorm`) over aggregated messages:

$$\mathbf{x}_i' = \mathrm{MLP}\left(\mathbf{x}_i + s \cdot \|\mathbf{x}_i\|_2 \cdot \frac{\mathbf{m}_i}{\|\mathbf{m}_i\|_2}\right)$$

The implementation is adapted from the official PyG example (`examples/ogbn_proteins_deepgcn.py`). I have used a 14-layer DeepGCN with large dropout rates (0.5), applying a learnable parameter `t` for softmax aggregation and a learnable scaling factor of message normalization. Accuracy curves are shown in Figure 2.

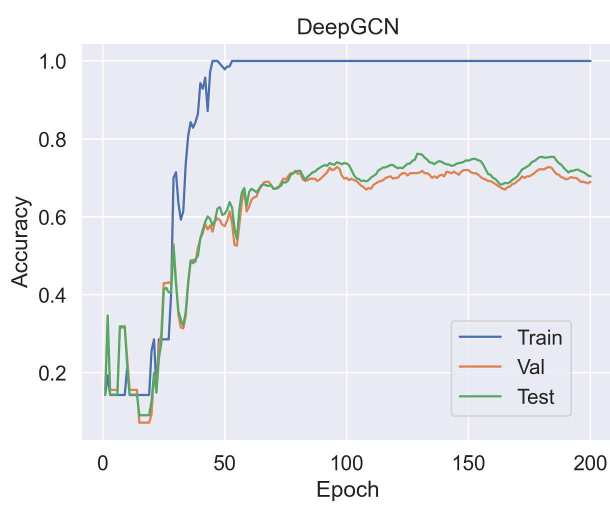


Figure 2: Learning curves of DeepGCN

The initial accuracy fluctuates, probably due to initialization. As the training set is quite small, a deeper model doesn't outperform a conventional GCN. Test accuracy can reach 0.7620 (epoch 129).

The training completed in 23.61 s (Wall time).

2 Generative Adversarial Networks (GAN)

2.1 GAN Implementation

Model Implementation: In our provided code, you can finish the implementation of *sample_noise*, *discriminator*, and *generator* in `gan_pytorch.py` following the hints in code notations. We present an example GAN structure. [**15pts**]

Loss Implementation: The training loss can be implemented in *discriminator_loss* and *generator_loss* respectively. You can run `gan.py` to go through the full training and sampling process, and show your generated images in your report. [**15pts**]

Answer

My model was implemented according to the given structure, using ReLU activation and a final Tanh layer in the generator, while using LeakyReLU activation in the discriminator (per GAN architecture guidelines). The training completed in 86.29 s (Wall time). As shown in Figure 3, *D* and *G* were indeed competing with each other, resulting in oscillating loss.

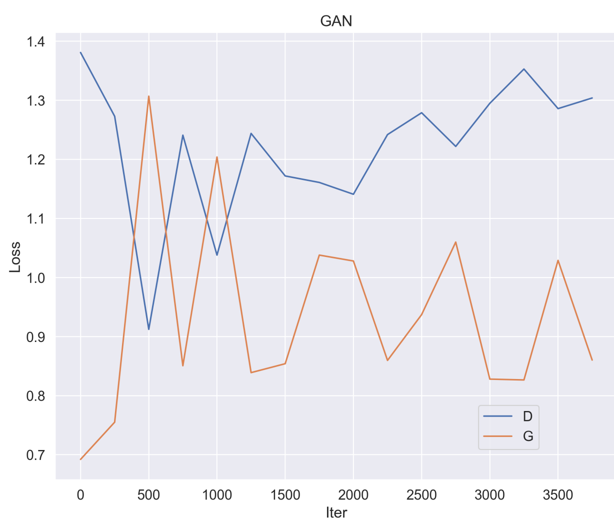


Figure 3: Learning curves of GAN

As can be seen from Figure 4, the generator produced increasingly clear hand-written digits from random noises. The final images are not smooth enough, containing random white pixels.

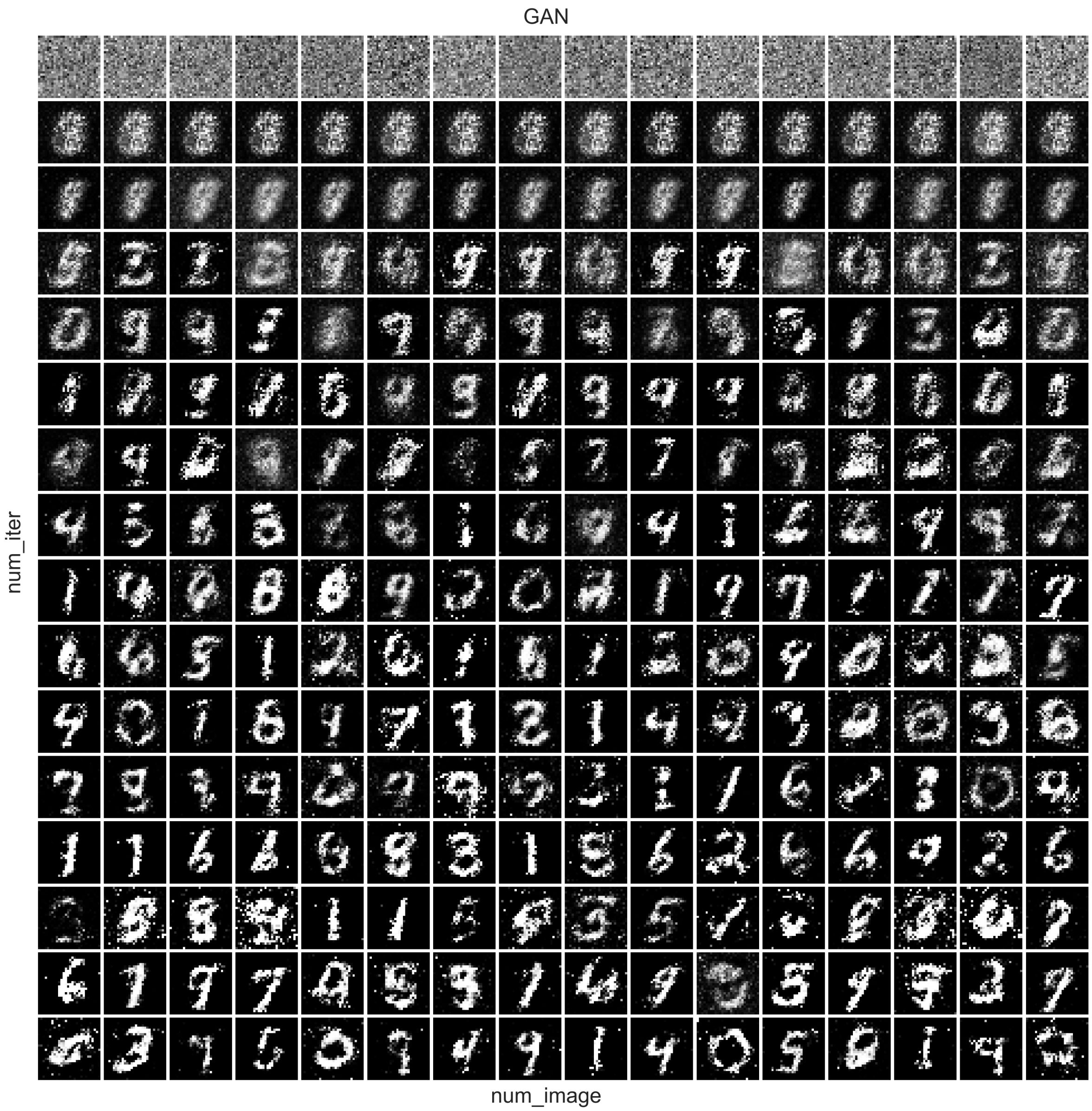


Figure 4: Image generated by GAN

2.2 Least Square GAN

LS-GAN is claimed to be more stable due to smoother gradients. Please implement LS-GAN in *ls_discriminator_loss* and *ls_generator_loss*, and show images generated by LS-GAN in your report. [**10pts**]

Answer

Loss functions are similarly defined using `mse_loss`. The training completed in 81.48 s (Wall time), and the loss curves in Figure 5 almost converge.

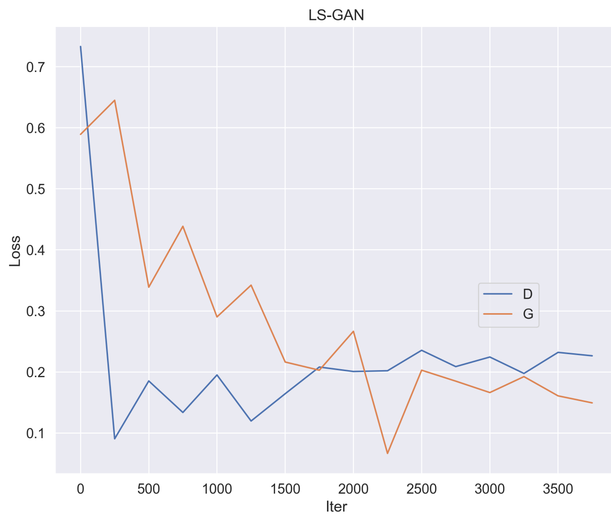


Figure 5: Learning curves of LS-GAN

It can be seen from Figure 6 that LS-GAN can converge slightly faster, but generated images don't improve much in quality. Some digits are even harder to discern.

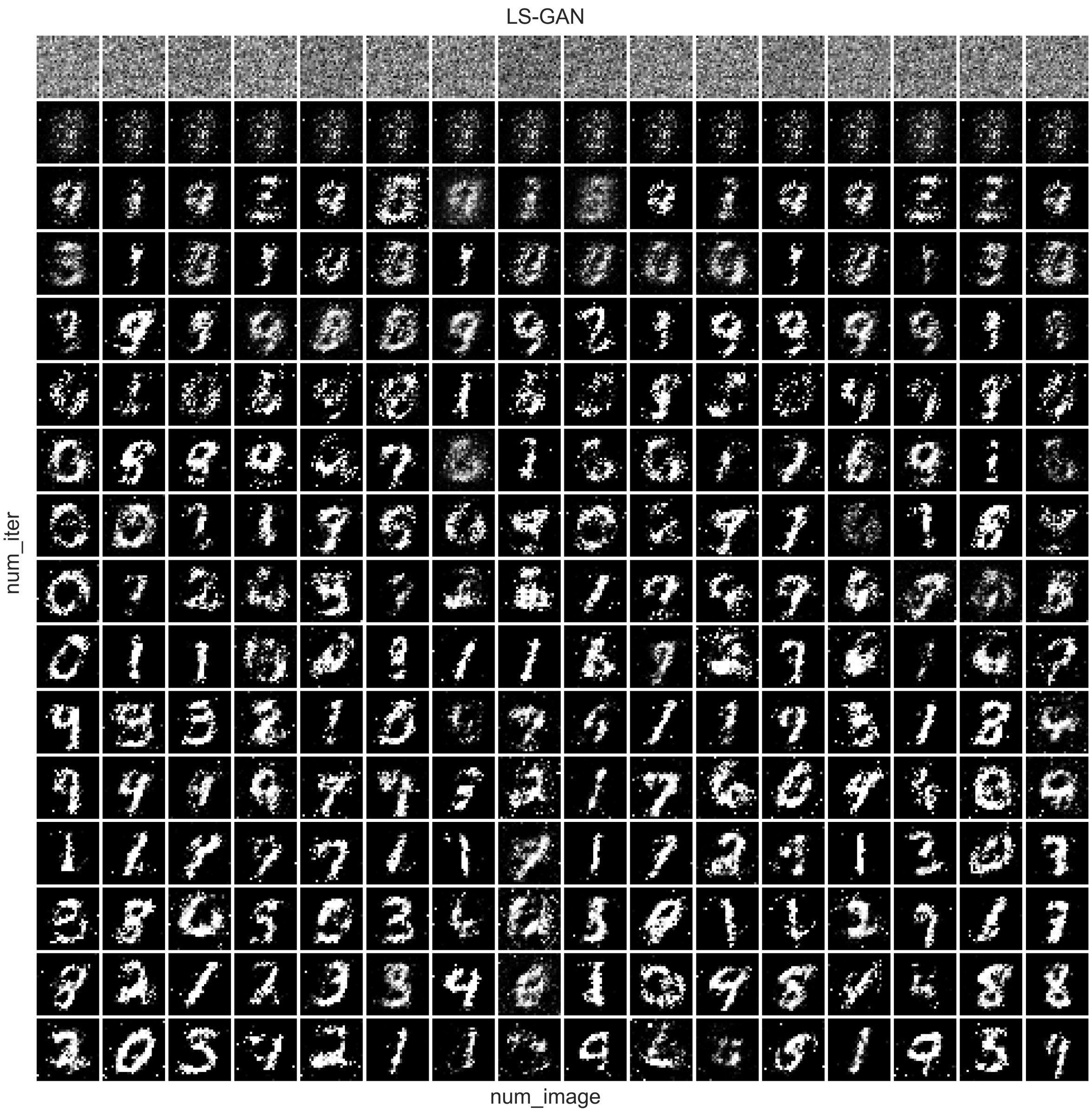


Figure 6: Image generated by LS-GAN

2.3 Deeply Convolutional GAN

Deeply Convolutional GAN (DC-GAN) introduces convolution networks, which greatly enhance the performance of Image Synthesis. We provide an example network structure. Now you are required to finish *build_dc_classifier* and *build_dc_generator*, and provide generated images with DC-GAN in your report. [**20pts**]

Answer

DC-GAN is structurally more complex than the previous two models. Initially I used large `ConvTranspose` kernels (= 7) in the generator, which failed to generate seemingly authentic images, giving rise to high G loss and extremely low D loss. Instead, a kernel size of 4 works fine here.

The training completed in 100.15 s (Wall time), and Figure 7 shows that G is actually the "winner".

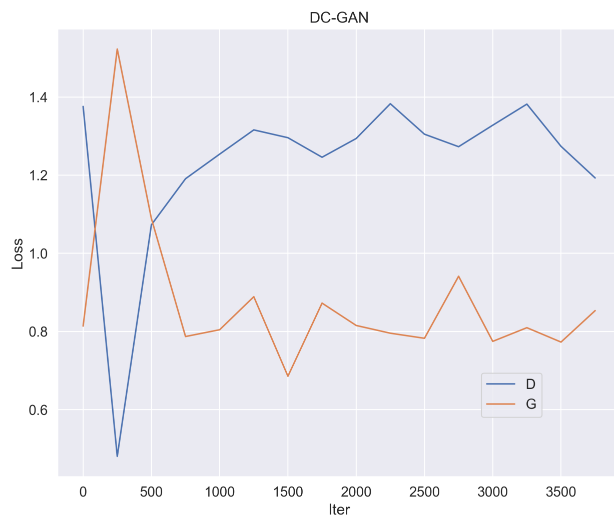


Figure 7: Learning curves of DC-GAN

Figure 8 demonstrates the effectiveness of DC-GAN. Images are very smooth compared with the previous two models, containing hardly any noticeable noise pixels.

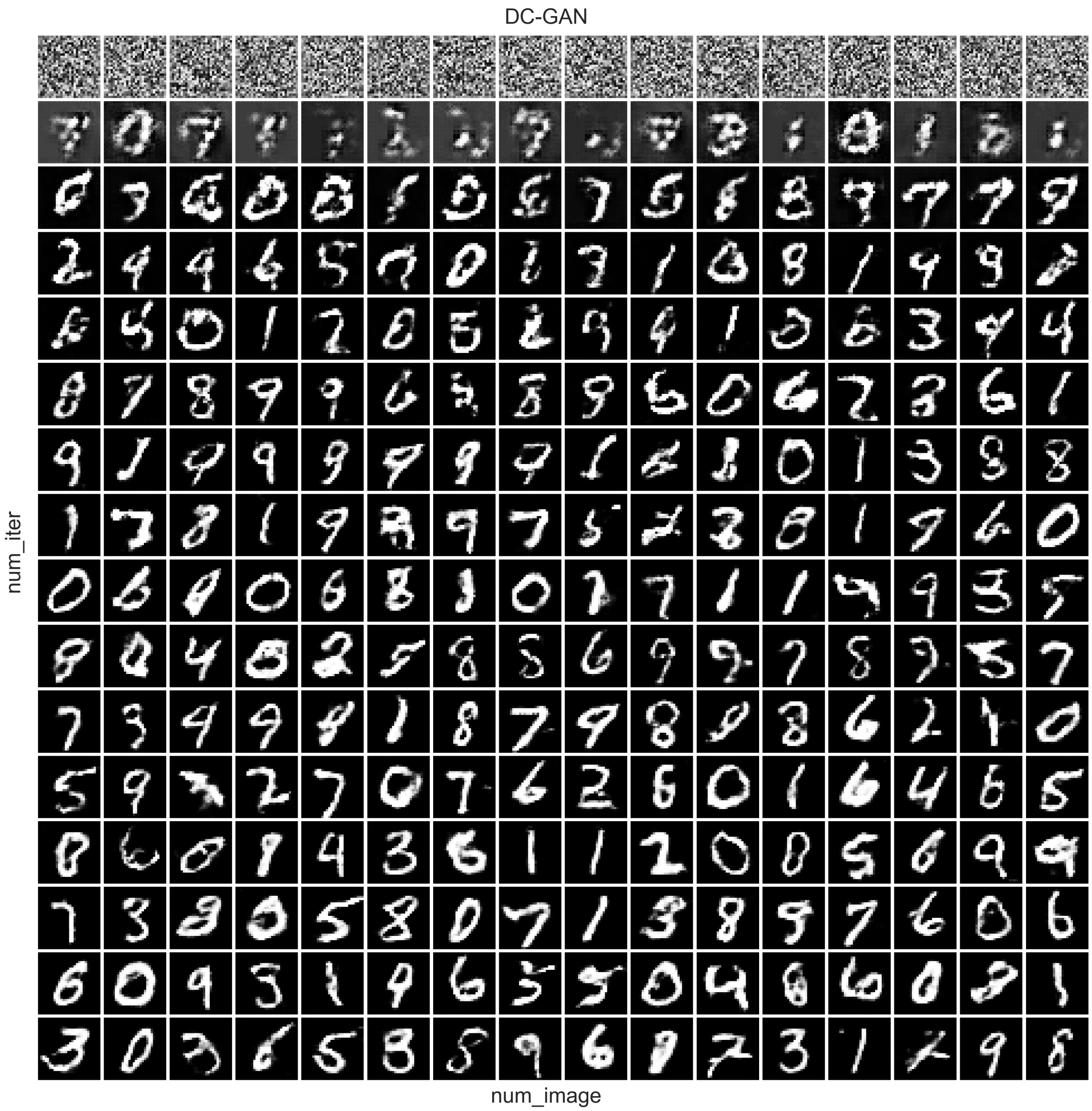


Figure 8: Image generated by DC-GAN

References

Li, Guohao, Chenxin Xiong, Ali Thabet, and Bernard Ghanem, "DeeperGCN: All You Need to Train Deeper GCNs," 2020.

— **, Matthias Müller, Ali Thabet, and Bernard Ghanem**, "DeepGCNs: Can GCNs Go as Deep as CNNs?," 2019.