Homework 3: Graph Models and Generative Models

Deep Learning (84100343-0)
Autumn 2022
Tsinghua University

This homework contains three parts: Graph Neural Networks (GNN) [**40pts**], Generative Adversarial Networks (GAN) [**60pts**] and Diffusion Models [**60pts**]. **Task on Graph Models is compulsory. For GAN and Diffusion Models, you can choose any one of them according to your interest.**

1 Graph Neural Networks (GNN) and Node2Vec

Graphs are the basic data structure. In the real world, objects cannot be fully understood without considering their connections with others. Graphs' edges are usually sparse, which makes it challenging to apply dense-input deep networks directly. Graph models in deep learning are specialized to get stronger expressiveness on graph information and have been used in antibacterial discovery, recommendation systems, and physics simulations.

1.1 Dataset Split in Node Classification

In this part, each data point is an independent sentence. Hence, it is easy to split the dataset (e.g., 80% as the training set and the remaining 20% as the validation set). While in a graph, different nodes are **not independent** because of message passing. In this case, there are two options.

- **Transductive Setting**: the input graph can be observed in all the dataset splits (training, validation, and test set). In other words, we only split the labels.
- **Inductive Setting**: we break the edges between splits to get multiple graphs. These graphs are thus independent.

1.2 Dataset and Library

- We adopt the citation network dataset *Cora* from [6]. In a citation network, nodes represent documents, and edges represent citation links.
- We will use Pytorch Geometric (PyG) in this assignment. You can install it according to the tutorial. Besides, it's **strongly** recommended to read through PyG's documentation and code.

1.3 Tasks and Scoring

You need to finish the following tasks on the given dataset:

- **Task A**: We provide implementations of **GCN**[2], **GAT**[4] and **Node2Vec**[1] for you. Read through and run `gcn.py`, `gat.py`, `node2vec.py`, and report the performance of these methods. [**20pts**]
- **Task B**: Implement **DeepGCN**[3] **or GIN**[5] (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**]

2 Generative Adversarial Networks (GAN)

Generative Adversarial Networks are widely applied in vision tasks, such as Image Synthesis, Video Inpainting, and Visual Style Transfer. A typical image-generation GAN contains a generator network

(G) and a discriminator network (D). G is trained to generate fake images that can fool D, while D aims to distinguish the real images and the synthetic images. We use the following formula to describe this process:

$$\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z\sim p(z)} [\log (1 - D(G(z)))] \tag{1}$$

where $z \sim p(z)$ are the random noise samples, $G(z)$ are the generated images using the neural network generator G, and $D(\cdot)$ is the output of the discriminator, which specifies the probability of an input being real.

In this task, we recommend you to implement the alternative update process optimizing a GAN:

- Update the generator G to maximize the probability of the discriminator given the outputs of G:

$$\max_G \mathbb{E}_{z\sim p(z)} [\log D(G(z))] \tag{2}$$

- Update the discriminator D to maximize the probability to make the correct choice on both real and generated data.

$$\max_D \mathbb{E}_{x\sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z\sim p(z)} [\log (1 - D(G(z)))] \tag{3}$$

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 in Figure 1. [**15pts**]

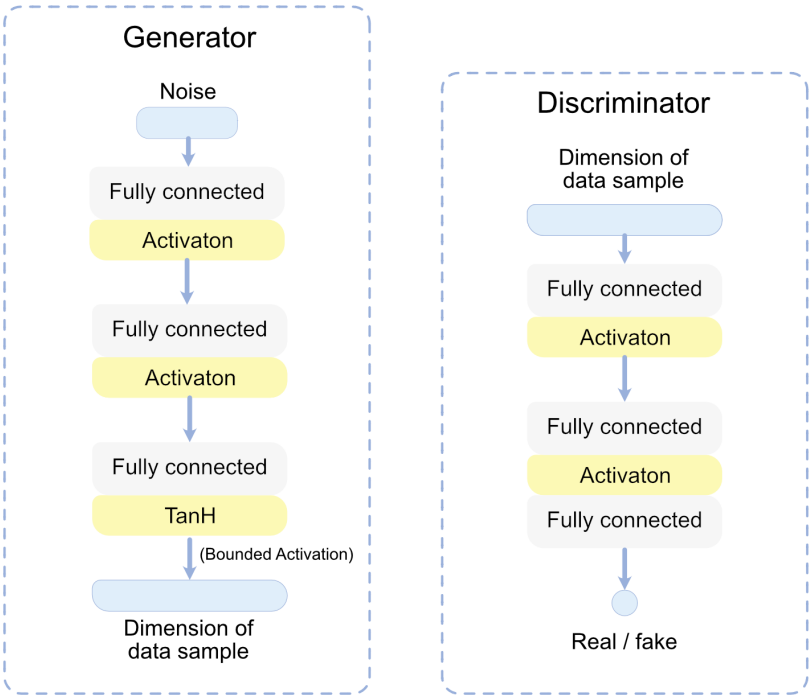

Figure 1: GAN structure

Loss Implementation: The training loss according to 2 and 3 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**]

2.2 Least Squire GAN

There are two main directions to modify a GAN: loss function and network backbone. For the loss function, Least Square GAN (LS-GAN) re-design the generator loss with:

$$\ell_G = \frac{1}{2}\mathbb{E}_{z\sim p(z)}\left[(D(G(z))-1)^2\right] \tag{4}$$

and the discriminator loss with:

$$\ell_D = \frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}\left[(D(x)-1)^2\right] + \frac{1}{2}\mathbb{E}_{z\sim p(z)}\left[(D(G(z)))^2\right] \tag{5}$$

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**]

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 in Figure 2. Now you are required to finish $build\_dc\_classifier$ and $build\_dc\_generator$, and provide generated images with DC-GAN in your report. [**20pts**]

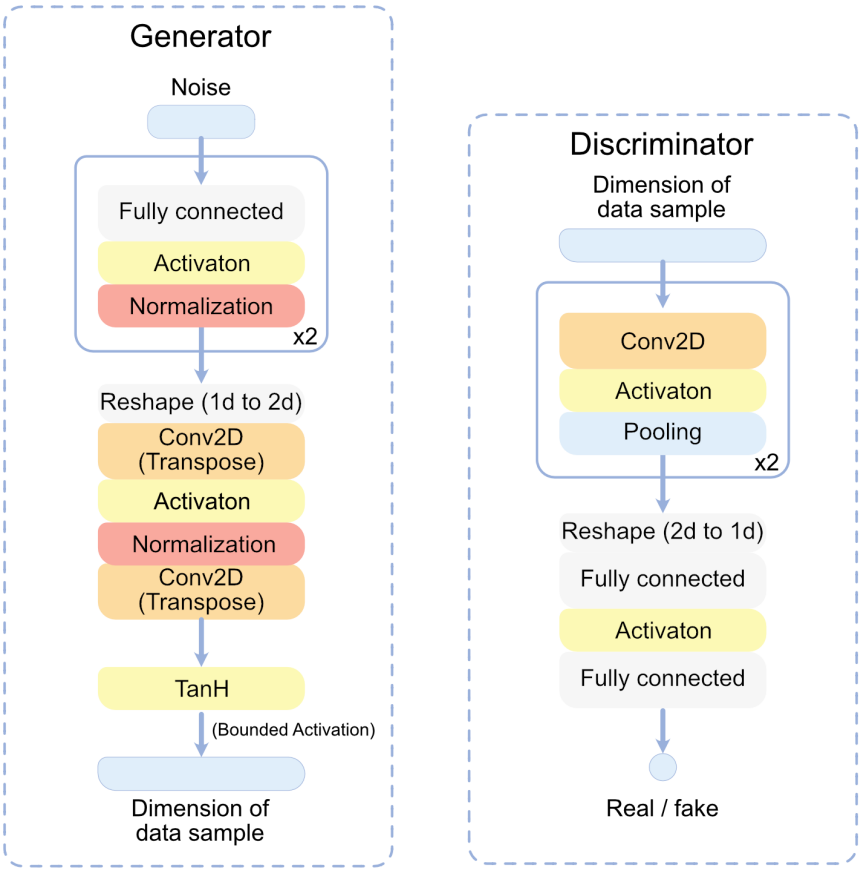

Figure 2: DC-GAM structure

3 Diffusion Models

In this part, we will consider a denoising diffusion probabilistic model with a forward process $q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$ and a reverse process $p_\theta(x_{0:T}) = \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$. We parameterize above processes with:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{6}$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t), \sigma_\theta^2 I) \tag{7}$$

- Suppose $\beta_t = c$ with a fixed $c : 0 \leq c < 1$ for all $t \in \mathrm{Z}$, we can extend the forward process to $T = \infty$. Prove that x_t converge to $\mathcal{N}(0, I)$ in distribution when $t \to \infty$. **[10pts]**
- During training, diffusion models aim to minimize the variational upper bound $L_t = D_{KL}(q(x_{t-1}|x_t, x_0)\|p_\theta(x_{t-1}|x_t))$. Please show that $q(x_{t-1}|x_t, x_0)$ follows a normal distribution and calculate the relative mean and variance. **[10pts]**

3.1 Diffusion Model Implementation

Diffusion models usually share the same training procedure. Thus a pre-trained model checkpoint can be adapted into different sampling strategies. For example, researchers are interested in the sampling scheduler designs, which can accelerate the sampling process with a limited decrease in performance and no need for additional training.

In this task, we prepare some open-source checkpoints of diffusion models and the sampling method from diffusion denoising probabilistic models (DDPM). Your tasks are listed as follows:

- **DDPM Case Study**: Generate image samples with different diffusion steps. Visualize the generated images and score them with your subjective assessment.**[10pts]**
- **Diffusion Denoising Implicit Model**: DDIM is a widely used diffusion scheduler. DDIM can be treated as a generalized DDPM with variational sampling variance in the reverse process. Implement the DDIM sampler, and do the same case study as DDPM. Then, compare the visualizations and the assessment results of DDIM samplers and DDPM samplers. **[10pts]**
- **Classifier Guided Diffusion Model**: CGDM is proven to outperform GAN on class-conditioning generation benchmarks. We provide the code of classifier guidance for DDPM. Change the conditional intensity and subjectively evaluate its effect on diversity, image quantity, and class correspondence. **[10pts]**

Notice that:

- `README.md` provides additional instructions on downloading checkpoints and running recipes.
- Considering the inference speed of diffusion models is slow, this task will not be evaluated for numerical performance. But if you are interested in the evaluation process of image generation models, we also provide a code with several metrics.
- Please **DO NOT** upload model checkpoints in your homework. Only result images should be contained in your report.
- We recommend 64×64 model checkpoints for the generation to save your time.

3.2 Text-to-image failure case study

Although the performance of diffusion models is astonishing, several failure cases of diffusion models can help to further improve existing models. **Stable Diffusion** is a successful open-sourced text-to-image generative model. You can easily experience this application with the inference API from Huggingface: https://huggingface.co/runwayml/stable-diffusion-v1-5.

- You need to find five text prompts that Stable Diffusion fails and provide your short explanation (like Figure 3). (Note that you should avoid using prompts that contain sensitive keywords) **[10pts]**

Figure 3: Text prompt: In December, Qatar Avenue with many pedestrians. [1]

4 Submit Format

Submit your *code and report* as an Archive (zip or tar). The report is supposed to cover your **question answering**, the model **technical details**, **experimental results**, and necessary **references**.

References

[1] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

[2] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[3] G. Li, M. Muller, A. Thabet, and B. Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

[5] K. Xu, W. Hu, and Leskovec. How powerful are graph neural networks? In *ICLR*, 2019.

[6] Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

[1] As we know, Qatar is holding the 2022 world camp, and its temperature is still above 20 centigrade even in December. But, in the generated picture, it seems like people are wearing heavy clothes, and snow covers the trees, which is impossible.