Image Generation with VAE and GAN

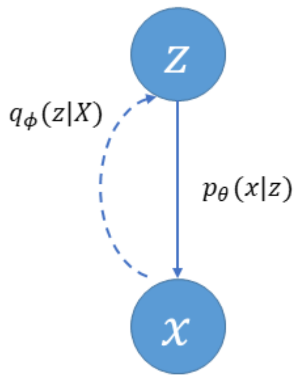
Background

In this homework, we will focus on two popular types of generative models, i.e., Variational Auto-Encoder (VAE) and Generative Adversarial Network (GAN). **为了减轻作业量，不要求实现GAN的部分。**

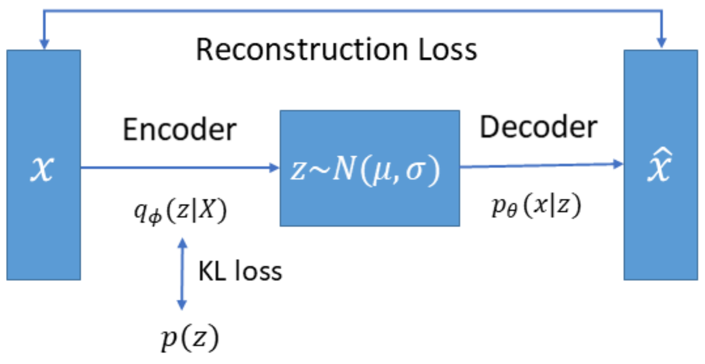
VAE

A VAE is a probabilistic model based on autoencoders, which takes high-dimensional input data and compresses it into low-dimensional representations. Unlike a traditional autoencoder which maps the input to a latent vector, a VAE maps the input data to a parameterized probability distribution, such as a Gaussian distribution with a mean and a variance. This approach produces a continuous, structured latent space, which is useful for image generation.

The graphical model of VAE is shown as follows, where z is a latent variable, x is an observed variable, θ is the parameters of the conditional distribution $p(x|z)$ (also called as the decoder network), and ϕ is the parameters of the approximate posterior distribution $q(z|x)$ (also called as the encoder network) . **In this homework, x is an image from MNIST.** A VAE models the probability $p(x) = \int_z p(z)p(x|z)$, where $p(z)$ is a prior distribution of the latent variable z. **In this homework, the prior distribution $p(z)$ is a unit Gaussian $\mathcal{N}(0, I)$.**



(a) Graphical Models



(b) Training Framework

Encoder Network ($q(z|x;\phi)$)

The encoder defines the approximate posterior distribution $q(z|x)$, which takes input as an observation and outputs a set of parameters for specifying the distribution of the latent representation z. **In this homework, we simply model the distribution as a diagonal Gaussian** (which means each dimension of z is independent of each other). Thus the network outputs two parameters: the mean μ and the log standard deviation $\log \sigma$, where σ^2 is the diagonal values of the Gaussian distribution's variance matrix. (We use log-variance instead of the variance directly for numerical stability.) That is, $q(z|x)$ can be formulated as $\mathcal{N}(\mu, diag(\sigma^2))$, where $diag(\sigma^2)$ is a matrix whose diagonal is σ^2 and the other elements are zero.

Decoder Network ($p(x|z;\theta)$)

The decoder defines the observation probability $p(x|z)$, which takes a latent sample z as input and produces the distribution of the observed variable. There are two methods to model the distribution, and we will introduce them in the reconstruction loss.

Loss Function

VAEs are trained by maximizing the evidence lower bound (ELBO) (See our slides or reference[1] for detailed proofs):

$$\log p(x) \geq ELBO = E_{q(z|x)}[\log \frac{p(x,z)}{q(z|x)}] = E_{q(z|x)}[\log p(x|z)] - KL(q(z|x)||p(z))$$

where $KL(\cdot||\cdot)$ denotes KL-divergence. Therefore, we can rewrite the loss function $\mathcal{L}$ as the sum of the reconstruction loss $\mathcal{L}^{rec}$ and the KL-divergence loss $\mathcal{L}^{kl}$:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}^{rec} + \mathcal{L}^{kl} \\ \mathcal{L}^{rec} &= -E_{q(z|x)}[\log p(x|z)] \\ \mathcal{L}^{kl} &= KL(q(z|x)||p(z)) \end{aligned}$$

Reconstruction Loss

Let $f_\theta(z)$ be the decoder output with input z. There are two commonly used ways to minimize the reconstruction loss:

1. One method is to model $p(\cdot|z;\theta)$ as the Gaussian distribution $\mathcal{N}(f_\theta(z), I)$, where the decoder $f_\theta(z)$ outputs the mean of the Gaussian. In this case, we can minimize the reconstruction loss by minimizing the mean square errors of the reconstruction, i.e., to minimize $||x - x^{'}||_2$, where $x^{'} = f_\theta(z)$ is a reconstruction of x.
2. Another method, **which we adopt in this homework**, is to use the decoder to compute for each pixel the probability of being 1 (assume the pixel value belongs to $[0,1]$). In this case, we can minimize the binary cross entropy between the original images and the reconstructions, i.e., to minimize $-[\sum_i x_i \log x_i^{'} + (1-x_i)\log(1-x_i^{'})]$, where $x^{'} = f_\theta(z)$ is a reconstruction of x, and i indexes over all pixels in x / $x^{'}$.

KL Loss

The KL-divergence between $\mathcal{N}(\mu, diag(\sigma^2))$ and $\mathcal{N}(0, I)$ can be computed in closed form as:

$$KL(\mathcal{N}(\mu, diag(\sigma^2))||\mathcal{N}(0,I)) = \frac{1}{2}\sum_{i=1}^{k}\left(\mu_i^2 + \sigma_i^2 - 2\log\sigma_i - 1\right)$$

where k is the dimensionality of z.

Reparameterization Trick

To obtain a sample z for the encoder-decoder training, we can encode the data x and then sample from the latent distribution $\mathcal{N}(\mu, diag(\sigma^2))$. Then we decode the latent z and reconstruct the data x.

However, the sampling operation creates a bottleneck because backpropagation cannot flow through a random node. To address the problem, we use the reparameterization trick. In this homework, we obtain z as follows:

$$z = \mu + \sigma \odot \epsilon$$

where $\odot$ is element-wise multiplication and ϵ is an random variable from the unit Gaussian $\mathcal{N}(0, I)$.

Network Architecture

For the encoder network, we use four convolutional layers followed by two fully-connected layers. In the decoder network, we mirror this architecture by using two fully-connected layers followed by four transposed convolutional layers. The details will be introduced later.

Training

- During each iteration, we pass the image to the encoder to obtain μ and $\log \sigma$ of the approximate posterior $q(z|x)$.
- Apply the *reparameterization trick* to sample z from $q(z|x)$.
- Pass the sampled z to the decoder to obtain the reconstructed image $f_\theta(z)$.
- Calculate the KL loss $\mathcal{L}^{KL}$ and reconstruction loss $\mathcal{L}^{rec}$, and make a optimizing step.

Inference

Besides reconstruction, VAE can generate novel images from the prior distribution.

- Sample z from the prior distribution p(z), i.e., $\mathcal{N}(0, I)$.
- Pass the sampled z to the decoder to obtain the generated image $f_\theta(z)$.

To keep track of the learning progression, we will generate a fixed batch of latent vectors that are drawn from a Gaussian distribution (i.e. `fixed_noise`) . In the training loop, we will periodically generate images from `fixed_noise`, and show them in Tensorboard.

GAN（选做）

GAN is a framework for teaching a DL model to capture the training data’s distribution. It is made of two distinct models, a *generator* and a *discriminator*. The job of the generator is to spawn ‘fake’ images that look like the training images. The job of the discriminator is to look at an image and output whether it is a real training image or a fake image from the generator. During training, the generator is constantly trying to outsmart the discriminator by generating better and better fakes, while the discriminator is working to become a better detective and correctly classify the real and fake images. The equilibrium of this game is when the generator is generating perfect fakes that look as if they came directly from the training data, and the discriminator is left to always guess at 50% confidence that the generator output is real or fake.

A DCGAN is a direct extension of the GAN, except that it explicitly uses convolutional and transposes convolutional layers in the discriminator and generator, respectively. **In this homework, we implement a DCGAN.**

Discriminator

The discriminator, D, is a binary classification network that takes an image as input and outputs a scalar probability that the input image is real (as opposed to fake). Here, D takes an input image, processes it through a series of Conv2d, BatchNorm2d, and LeakyReLU layers, and outputs the final probability through a Sigmoid activation function. This architecture can be extended with more layers if necessary, but there is significance to the use of the strided convolution, BatchNorm, and LeakyReLUs. The DCGAN paper mentions it is a good practice to use strided convolution rather than pooling to downsample because it lets the network learn its own pooling function. BatchNorm and LeakyRelu functions also promote healthy gradient flow which is critical for the learning process of both G and D.

Generator

The generator, G, is designed to map the latent space vector (z, drawn from a unit Gaussian distribution $\mathcal{N}(0, I)$) to data-space. Since our data are images, converting z to data-space means creating an image with the same size as the training images. In practice, this is accomplished through a series of strided two dimensional convolutional transpose layers, each paired with a 2d BatchNorm layer and a Relu activation. The output of the generator is fed through a tanh function to return it to the input data range of $[-1, 1]$. It is worth noting the existence of the BatchNorm functions after the conv-transpose layers, as this is a critical contribution of the DCGAN paper. These layers help with the flow of gradients during training.

Weight Initialization

From the DCGAN paper, the authors specify that all model weights shall be randomly initialized from a Gaussian distribution with mean=0, stdev=0.02.

Loss Function

The loss function is:

$$\min_G \max_D V(D, G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_g(z)}[\log(1 - D(G(z)))]$$

Training

Training is split up into two main parts. Part 1 updates the Discriminator and Part 2 updates the Generator.

- Part 1: Practically, we want to maximize ($\log D(x) + \log(1 - D(G(z)))$). **First**, we will construct a batch of real samples from the training set, forward pass through D, calculate the loss ($\log D(x)$), then calculate the gradients in a backward pass. **Secondly**, we will construct a batch of fake samples with the current generator, forward pass this batch through D, calculate the loss ($\log(1 - D(G(z)))$), and *accumulate* the gradients with a backward pass.
- Part 2: As stated in the original paper, we want to train the Generator by minimizing ($\log(1 - D(G(z)))$) in an effort to generate better fakes. However, the objective suffers from the gradient vanishing problem. **As a fix, we maximize ($\log D(G(z))$) instead.** (This is the log-D trick in our slides.)

To keep track of the generator's learning progression, we will generate a fixed batch of latent vectors that are drawn from a Gaussian distribution (i.e. `fixed_noise`) . In the training loop, we will periodically input this fixed_noise into G, and over the iterations we will see images formed out of the noise.

Requirements

- Python 3
- PyTorch >= 1.1
- torchvision
- tensorboard
- scipy < 1.4 // make sure spicy < 1.4 is installed, otherwise computing FID score will be extremely slow (refer to https://zhuanlan.zhihu.com/p/110053716 for more details)

Dataset Description

`dataset.py` defines a class to load the MNIST data and transform the size of each image from 28×28 to $1 \times 32 \times 32$.

- **For VAE, Pixel values are normalized in $[0,1]$.**
- **For DCGAN, Pixel values are normalized in $[-1,1]$.**

Download MNIST dataset and Inception model and put all files under `codes/`.

Python Files Description

In this homework, we provide unfinished implementation of VAE and DCGAN using **PyTorch**. Both programs share the same code structure:

- `main.py`: the main script for running the whole program.
- `VAE.py/GAN.py`: the main script for model implementation, including some utility functions.
- `dataset.py`: the script that loads the data
- `trainer.py`: the script that trains a generative model

Usage

- Training: `python main.py --do_train`. It will load the latest checkpoint (initialize if not existed) and run the training process. After training, the model will be evaluated by FID.
- Test: `python main.py`. It will load the latest checkpoint (initialize if not existed) and evaluate the model by FID.
- Tensorboard: `tensorboard --logdir ./runs`. It will open a web server to show the training curves.

VAE:

You are supposed to:

1. Implement a VAE of the following architecture in `VAE.py`. `latent-dim` denotes the dimension of the latent variable.

Layer id	Layer type	Output volume/dimension	Kernel size	Stride	Padding	Output padding
Enc-1	Conv+ReLU	$16 \times 32 \times 32$	3	1	1	\
Enc-2	Conv+ReLU	$32 \times 16 \times 16$	3	2	1	\
Enc-3	Conv+ReLU	$32 \times 16 \times 16$	3	1	1	\
Enc-4	Conv+ReLU	$16 \times 8 \times 8$	3	2	1	\
Enc-5	FC+ReLU	2*latent-dim	\	\	\	\
Enc-6	FC	2*latent-dim (See Note below)	\	\	\	\
Dec-1	FC+ReLU	latent-dim	\	\	\	\
Dec-2	FC	$16 \times 8 \times 8$	\	\	\	\
Dec-3	Transposed conv+ReLU	$32 \times 16 \times 16$	3	2	1	1
Dec-4	Transposed conv+ReLU	$32 \times 16 \times 16$	3	1	1	0
Dec-5	Transposed conv+ReLU	$16 \times 32 \times 32$	3	2	1	1
Dec-6	Transposed conv+Sigmoid	$1 \times 32 \times 32$	3	1	1	0

Note: Enc-6's output should be split into two parts: the first part is μ, the second part is $\log \sigma$, so both μ and $\log \sigma$ has latent-dim elements.

2. Implement the `reparemeterize()` and `forward()` function in `VAE.py`
3. Implement the `loss_fn()` in `trainer.py`.

   You are required to average the losses over all instances in a batch.

DCGAN（选做）：

为了减轻作业量，GAN的部分可以选做。做了最多加两分。

You are supposed to:

1. Implement the Generator of the following architecture in `GAN.py`. BN2d denotes BatchNorm2d. Both `latent-dim` and `hidden-dim` are hyper parameters of the Generator. **In this homework, we set `hidden-dim` to 16.**

   The input latent variable z is of shape $[\text{latent-dim}, 1, 1]$

Layer id	Layer type	Output volume	Kernel size	Stride	Padding
1	Transposed conv + BN2d + ReLU	4*hidden-dim $\times 4 \times 4$	4	1	0
2	Transposed conv + BN2d + ReLU	2*hidden-dim $\times 8 \times 8$	4	2	1
3	Transposed conv + BN2d + ReLU	hidden-dim $\times 16 \times 16$	4	2	1
4	Transposed conv + Tanh	$1 \times 32 \times 32$	4	2	1

2. Implement the `train_step()` function in `trainer.py`. **See the comments in codes for some hints.**

   You are required to average the losses over all instances in a batch.

Report

In the experiment report, you need to answer the following questions:

1. Set the dimension of latent variable of 3, 10, 100, and train three VAEs separately. **Therefore, you need to run at least three experiments in this homework.**
   1. The code we provide will plot a number of graphs (e.g. loss curves) during training on the Tensorboard. Show them in your report. (6 curves for a VAE. Make sure that TAs can see each curves clearly.)
   2. The code we provide will also paint some images on the Tensorboard. Show images from the validation set, their reconstructions, and model-generated images **at the last record step**. (Three groups of images for a VAE. Make sure that TAs can see each graph clearly.)
   3. **State the experimental settings of each curve/image clearly, which include `latent_dim`, `batch_size`, and `num_train_steps`.**

2. We use FID score to evaluate the quality of model-generated images. FID score measures the similarity between model-generated images and real images. Specifically, we extract features from each image with a well-trained model, fit a multivariate Gaussian to the feature vectors of generated images and real images separately, and compute the similarity between the two Gaussians. **The lower FID means the performance is better.** For more details, refer to the reference[3].

   Report the FID score of the four trained models mentioned above. (The best VAE's FID should be lower than 40.)

3. Discuss how the dimension of the latent variable influences the performance of VAEs.

   Bonus point (optional 0.5 points): Change the architecture of the encoder/decoder of a VAE (e.g., using only fully connected layers, or using less layers, etc.), discuss how the architecture of encoder/decoder influences the performance.

The following requirements are optional （**选做，最多加2分**）:

4. **(Optional)** Set the dimension of the latent variable to 100 and train a DCGAN.
   1. The code we provide will plot a number of graphs (e.g. loss curves) during training on the Tensorboard. Show them in your report. (5 curves are required. Make sure that TAs can see each curves clearly.)
   2. The code we provide will also paint some images on the Tensorboard. Show the model-generated images **at the last record step**. One group of images is required. Make sure that TAs can see each graph clearly.)
   3. **State the experimental settings of each curve/image clearly, which include `latent_dim`, `batch_size`, and `num_train_steps`.**

5.. **(Optional)** Report the FID score of the GAN. (The best GAN's FID should be lower than 80.)

6. **(Optional)** Has your GAN converged to the Nash equilibrium? See the training curves and explain why.

7. **(Optional)** Comparing the generated images from DCGAN and VAE-100 (the VAE with latent variable of size 100). Which ones are blurry and which ones are sharp? Try to explain the phenomenon from the training objectives.

8. **(Optional)** Interpolation in the latent space is a common way to qualitatively evaluate how well a GAN or VAE learns the latent representations. Linear interpolation can be conducted by first sampling two latent points denoted as z_1 and z_2, and then decoding an image from $z_1 + \frac{i}{K}(z_2 - z_1)$ for $i \in \{0, 1, \ldots, K\}$ (One can also try setting i to a value bigger than K or smaller than 0 to evaluate its generalization, which is the case of linear extrapolation.)

   Choose one model (VAE-100 or GAN), add some codes and apply linear interpolation in the latent space. Show the interpolated images and discuss the performance of the generated images.

   Bonus point (optional 0.5 points): Conduct the interpolation experiment on two models. Which one learns better latent representations, DCGAN or VAE-100?

9. Other explorations, for example, other datasets, different architectures and so on. At most **2 points** (the bonus mentioned above are also included).

NOTE: Since the training time may be long, you are not requried to tune the hyper-parameters. The default hyper-parameters are enough to produce reasonable results (if you have implemented the model correctly, especially for the loss, which should be averaged over the batch).

Code Checking

We introduce a code checking tool this year to avoid plagiarism. You **MUST** submit a file named `summary.txt` along with your code, which contains what you modified and referred to. You should follow the instructions below to generate the file:

1. Fill the codes. Notice you should only modify the codes between `# TODO start` and `# TODO end`, the other changes should be explained in `README.md`. **DO NOT** change or remove the lines start with `# TODO`.
2. Add references if you use or refer to a online code, or discuss with your classmates. You should add a comment line just after `# TODO start` in the following formats:
   1. If you use a code online: `# Reference: https://github.com/xxxxx`
   2. If you discuss with your classmates: `# Reference: Name: Xiao Ming Student ID: 2018xxxxxx`

   You can add multiple references if needed.

   Warning: You should not copy codes from your classmates, or copy codes directly from the Internet, especially for some codes provided by students who did this homework last year. In all circumstances, you should at least write more than 70% codes. (You should not provide your codes to others or upload them to Github before the course ended.)

   警告:作业中不允许复制同学或者网上的代码，特别是往年学生上传的答案。我们每年会略微的修改作业要求，往年的答案极有可能存在错误。一经发现，按照学术不端处理(根据情况报告辅导员或学校)。在任何情况下，你至少应该自己编写70%的代码。在课程结束前，不要将你的代码发送给其他人或者上传到github上。
3. Here is an example of your submitted code:

```
def forward(self, input):
    # TODO START
    # Reference: https://github.com/xxxxx
    # Reference: Name: Xiao Ming Student ID: 2018xxxxxx your codes...
    # TODO END
```

4. At last, run python `./code_analyze/analyze.py` , the result will be generated at `./code_analyze/summary.txt`. Open it and check if it is reasonable. A possible code checking result can be:

```
######################## # Filled Code ######################## #
..\codes\layers.py:1
# Reference: https://github.com/xxxxx
# Reference: Name: Xiao Ming Student ID: 2018xxxxxx your codes...
########################
# References
########################
# https://github.com/xxxxx
# Name: Xiao Ming Student ID: 2018xxxxxx
########################
# Other Modifications
########################
# _codes\layers.py -> ..\codes\layers.py
# 8 - self._saved_tensor = None
# 8 + self._saved_tensor = None # add some thing
```

Submission Guideline

You need to submit both report and codes, which are:

- **Report**: well formatted and readable summary including your results, discussions and ideas. Source codes should not be included in report writing. Only some essential lines of codes are permitted for explaining complicated thoughts.
- **Codes**: organized source code files with README for **extra modifications** (other than `TODO`) or specific usage. Ensure that TAs can successfully reproduce your results following your instructions. **DO NOT include model weights/raw data/compiled objects/unrelated stuff over 50MB.**
- **Code Checking Result**: You should only submit the generated `summary.txt`. **DO NOT** upload any codes under `code_analysis`. However, TAs will regenerate the code checking result to ensure the correctness of the file.

You should submit a `.zip` file name after your student number, organized as below:

- `Report.pdf/docx`
- `summary.txt`
- `codes/`
    - `VAE/`
        - `*.py`
        - `README.md/txt`
    - `GAN/` **(Optional)**
        - `*.py`
        - `README.md/txt`

Deadline: Nov. 15th

TA contact info:

- Shao Zhihong (邵智宏)， szh19@mails.tsinghua.edu.cn

Reference

[1] CARL DOERSCH. Tutorial on Variational Autoencoders. arXiv preprint arXiv:1606.05908, 2016.

[2] Alec Radford, Luke Metz, Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv preprint arXiv:1511.06434, 2015

[3] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. arXiv preprint arXiv:1706.08500,2017