ECS795P Deep Learning and Computer Vision, 2022

Course Work 2: Unsupervised Learning by Generative Adversarial Nets

Introduction

Aim:
(1) to obtain practical knowledge and hands-on understanding of the basic concepts in Generative Adversarial Nets(GAN);
(2) to obtain practical experience on how to implement basic GAN using PyTorch.

Start: Download and install PyTorch from its official website:
For Linux: https://pytorch.org/get-started/locally/#linux-installation .; For Mac: https://pytorch.org/get-started/locally/#mac-installation ;
For Windows: https://pytorch.org/get-started/locally/#windows-installation .

Tasks: three subtasks are involved:

1. Coding: to add your code blocks in the required sections based on Sec.2.2 in this guideline and annotations in coding files; (30% of this CW)
2. Report: to complete the questions in report; (50% of this CW)
3. Online assessment: to answer one question and to conduct one exercise, which are randomly selected from below. It will be carried out during the lab demo session in WK11; (20% of this CW)

Platform: python + PyTorch

Basic material:
Some of online materials for PyTorch-code may help you better complete this coursework (if you are not familiar with PyTorch, you can follow this step by step)
https://pytorch.org/tutorials/
https://github.com/yunjey/pytorch-tutorial https://github.com/MorvanZhou/PyTorch-Tutorial/blob/master/tutorial-contents- notebooks/404_autoencoder.ipynb

1. Understanding basic concepts in GAN models

Objective: To become familiar with the basic knowledge of the GAN model and its basic usages.

1.1 The questions to think over:

Reference: https://towardsdatascience.com/understanding-generative-adversarial-networks-4dafc963f2ef
Reference:

1. What are the two basic parts in GAN?

A generative adversarial network (GAN) is composed of two parts:

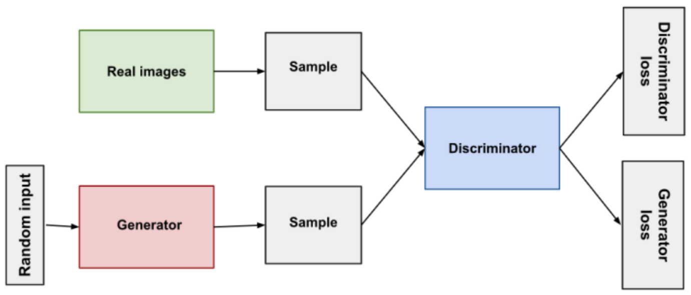
Generator:

The generator improves its ability to generate plausible data. The instances that are generated serve as negative training examples for the discriminator.

Discriminator:

The discriminator learns to distinguish between real and fake data generated by the generator. The generator is penalised by the discriminator for producing implausible results.

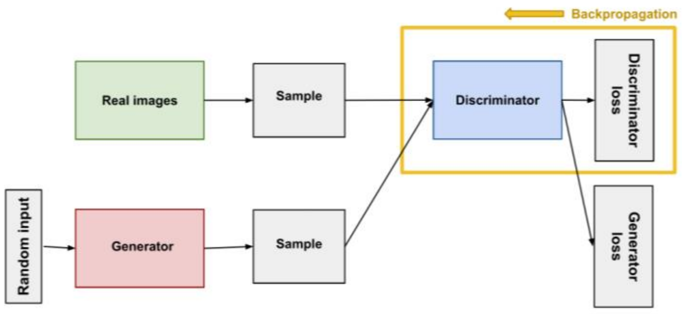
Here's a picture of the whole system:



2. What are the specific objectives of these two parts?

Discriminator:

In a GAN, the discriminator is simply a classifier. It attempts to distinguish between real data and data generated by the generator. It could use any network architecture suitable for the type of data it's classifying.

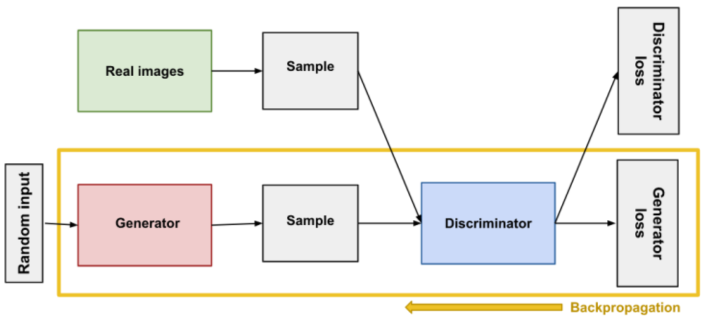


The Generator:

The generator component of a GAN learns to generate fake data by incorporating discriminator feedback. It learns to manipulate the discriminator so that its output is classified as real.

Generator training necessitates a closer integration of the generator and the discriminator than discriminator training necessitates. The part of the GAN that trains the generator is as follows:

- random input
- generator network that transforms the random input into a data instance
- discriminator network that classifies the generated data discriminator output
- generator loss that penalises the generator for failing to fool the discriminator



3. What is the basic loss function of GAN?

A GAN can be equipped with two loss functions: one for generator training and one for discriminator training. The generator and discriminator losses are derived from a single measure of probability distribution distance. However, in both of these schemes, the generator can only influence one term in the distance measure: the term that reflects the distribution of the fake data. As a result, during generator training, we eliminate the other term that reflects the distribution of the real data.

Even though they are derived from the same formula, the generator and discriminator losses look different in the end.

Minimax GAN loss

The term "minimax GAN loss" refers to the simultaneous minimax optimization of the discriminator and generator models. The generator attempts to minimise the following function, whereas the discriminator attempts to maximise it:

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

As previously stated, the discriminator attempts to maximise the average of the log probability of real images and the log of the inverse probability of fake images.

discriminator: maximize log D(x) + log(1 – D(G(z)))

For fake images, the generator attempts to minimise the log of the inverse probability predicted by the discriminator. This encourages the generator to generate samples with a low likelihood of being a forgery.

generator: minimize log(1 – D(G(z)))

4. What is the training process of GAN?

GAN training components

In general, a training phase has two major subparts that are completed sequentially.

Pass 1: Train the discriminator and freeze the generator (freezing means false training). The network only uses forward pass and does not use backpropagation). We ignore the generator loss and only use the discriminator loss during discriminator training, which penalises the discriminator for misclassifying real faces as fake or generated faces as real. Backpropagation is used to update the weights of the generator. The weights of the generator are not updated.

Pass 2: Train the generator and freeze the discriminator. We use the generator loss during generator training to penalise the generator if it fails to fool the discriminator and generates a face that the discriminator classifies as fake. During generator training, the discriminator is frozen, and only the weights of the generator are updated via backpropagation.

1.2 The exercises to conduct:

1. Check details of *GAN_pytorch.py,* understand the whole framework.
   For example:
   1) Dataset: Which dataset is used? How to load this dataset?
   2) Model: Can you plot/draw the basic architecture in this case?
   3) Loss: What is the loss function in this example?
   4) Training: How to train this network?
   5) Test: How to test this model?
2. Change the learning rate to 0.01 and train for a few epochs to understand how learning rate will influence the model outcome.
3. Change the batch size to 256 and discuss how the batch size influences the model performance regarding its training speed and test accuracy.

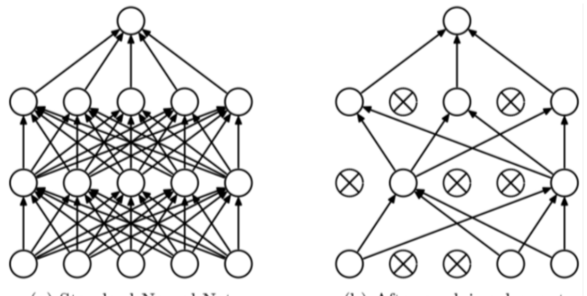
2. Generative Adversarial Networks with PyTorch

Objective: To become familiar with GAN and re-implement the original GAN model.

2.1 The questions to think over:

1. What is dropout in deep learning and its advantages.

Dropout is a technique that removes neurons from the neural network or 'ignores' them during training; in other words, different neurons are temporarily removed from the network. Dropout modifies the idea of learning all the weights in the network during training to learning only a fraction of the weights in the network. The figure below shows that during the standard training phase, all neurons are involved, whereas during the dropout phase, only a few select neurons are involved, with the rest 'turned off.' To prevent some neurons from dominating the process, different sets of neurons are activated after each iteration.


(a) Standard Neural Net (b) After applying dropout.

Advantages:

- This method keeps neural networks from overfitting.
- It prevents all neurons in a layer from optimising their weights at the same time. This adaptation, performed in random groups, prevents all neurons from convergent on the same goal, thereby decorrelating the weights.
- The use of dropout is that the activations of the hidden units become sparse, which is also desirable.

https://ai-pool.com/a/s/dropout-in-deep-learning#:~:text=Dropout%20is%20a%20technique%20that,network%20on%20a%20temporary%20basis.

https://www.oreilly.com/library/view/machine-learning-for/9781786469878/252b7560-e262-49c4-9c8f-

5b78d2eec420.xhtml#:~:text=The%20main%20advantage%20of%20this,goal%2C%20thus%20decorrelating%20the%20weights.

2. List some typical optimizers in deep learning.

   TYPES OF OPTIMIZERS :

   - Gradient Descent

     This is one of the oldest and the most common optimizer used in neural networks, best for the cases where the data is arranged in a way that it possesses a convex optimization problem. It will try to find the least cost function value by updating the weights of your learning algorithm and will come up with the best-suited parameter values corresponding to the Global Minima. This is done by moving down the hill with a negative slope, increasing the older weight, and positive slope reducing the older weight.

   - Stochastic Gradient Descent

     This is a Gradient Descent optimizer variant that can work with data and solve non-convex optimization problems. The problem with such data is that the cost function is based on local minima that are incompatible with your learning algorithm. Instead of batch processing, this optimizer executes one update at a time. As a result, it's usually much faster, and the cost function gets smaller with each iteration (EPOCH). It performs frequent, high-variance updates, causing the objective function (cost function) to fluctuate significantly. As a result, the gradient shifts to a possible Global Minima.

   - Adagrad

     This is an algorithm in which the learning rate is crucial in determining the updated parameter values. This optimizer, unlike Stochastic Gradient Descent, uses a different learning rate for each iteration. It also uses different learning rates for each of the parameters rather than the same one for all of them.

   - Adadelta

     This is an extension of the Adaptive Gradient optimizer that addresses its aggressive nature of decreasing the learning rate infinitesimally. Instead of using the previous squared gradients, the sum of gradients is defined as a reducing weighted average of all previous squared gradients (weighted averages), limiting the learning rate to a very small value.

   - RMSprop

     Both the optimising algorithms, RMSprop (Root Mean Square Propagation) and Adadelta, were created around the same time to solve Adagrad's problem of destructive learning rates.

     Both, however, use the same method to determine the learning rate at time t for each iteration, which is an Exponential Weighted Average.

     Geoffrey Hinton proposed RMSprop, an adaptive learning rate method that divides the learning rate by an exponentially weighted average of squared gradients.It is recommended to set gamma to 0.95 because it has produced good results in the majority of cases.

   - Adam

     This is the Adaptive Moment Estimation algorithm, which also computes adaptive learning rates for each parameter at each iteration. The parameter values are determined using a combination of Gradient Descent with Momentum and RMSprop.

     When the algorithm was first introduced, there was a list of appealing advantages of using Adam on non-convex optimization problems, which made it the most commonly used optimizer.

https://medium.com/analytics-vidhya/this-blog-post-aims-at-explaining-the-behavior-of-different-algorithms-for-optimizing-gradient-46159a97a8c1

3. What optimizer we used for training in this case.

   For our training purpose the Adam optimizer has been used.

   Adam: This is the Adaptive Moment Estimation algorithm, which also computes adaptive learning rates for each parameter at each iteration. The parameter values are determined using a combination of Gradient Descent with Momentum and RMSprop. When the algorithm was first introduced, there was a list of appealing advantages of using Adam on non-convex optimization problems, which made it the most commonly used optimizer.

   Advantages:

   It comes with combining the benefits of both Gradient with Momentum and RMSProp such as:

   - low memory requirements,
   - appropriate for non-stationary objectives,
   - works best with large data and
   - parameters with efficient computation.

   Working:

   This works by storing an exponential weighted average of the past squared derivative of loss with respect to the weight at time t-1, in addition to the same methodology as adaptive learning rate.

   Parameters:

   It has several parameters, which are β1, β2, and ε(epsilon).

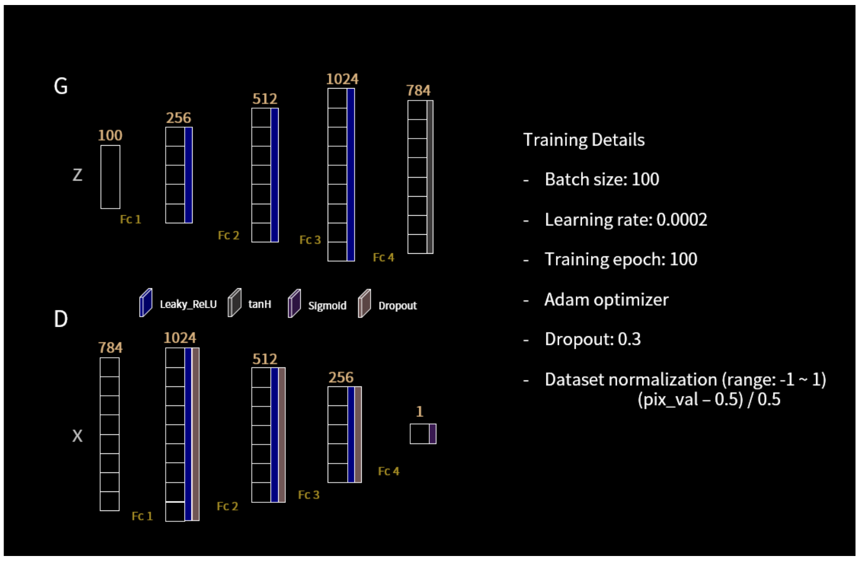
   β1 → initial restricting parameters for Momentum

   β2 → initial restricting parameters for RMSprop

4. How do D-loss and G-loss change during training? Visualize how the D-loss and G-loss change during training and explain the reason.

2.2 The exercises to conduct:

1. [coding] Change the architecture of discriminator and generator as follow:



Training Details

- Batch size: 100
- Learning rate: 0.0002
- Training epoch: 100
- Adam optimizer
- Dropout: 0.3
- Dataset normalization (range: -1 ~ 1)
  (pix_val – 0.5) / 0.5

2. Remove dropout function for this architecture and plot its training loss.
3. **[coding]** Show the generated images at the 10th epoch, the 20th epoch, the 50th epoch, the 100th epoch.

The answers to this section can be found in the code executed.