Generative Adversarial Network

Creating your own universe with unsupervised learning fashion

Huynh Van Duy, Truong Nhat Hoang, Tran Quoc Linh
{51703006, 51703092, 51703124}@student.tdtu.edu.vn

December 1, 2019

Faculty of Information Technology, Ton Duc Thang University

"GAN is the coolest idea in machine learning in the last twenty years.[1]" - Yann LeCun

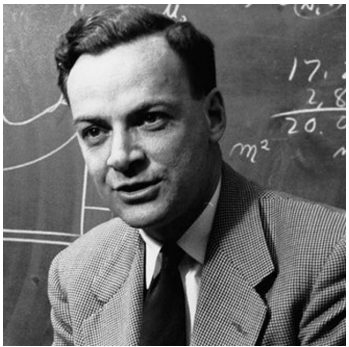
[1] LeCun, Yann. RL Seminar: The Next Frontier in AI: Unsupervised Learning. 2016

Table of contents

Introduction

Inspiration

Richard Feynman: *"What I cannot create, I do not understand"*
Generative Model: *"What I understand, I can create"*



Motivation

Challenge: Understand complex, unstructured inputs and describe our world.

Figure 1: Computer Vision

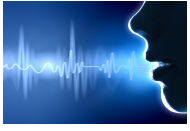


Figure 2: Speech Processing

Figure 3: Natural Language Processing

Figure 4: Robotic

Applications

Applications

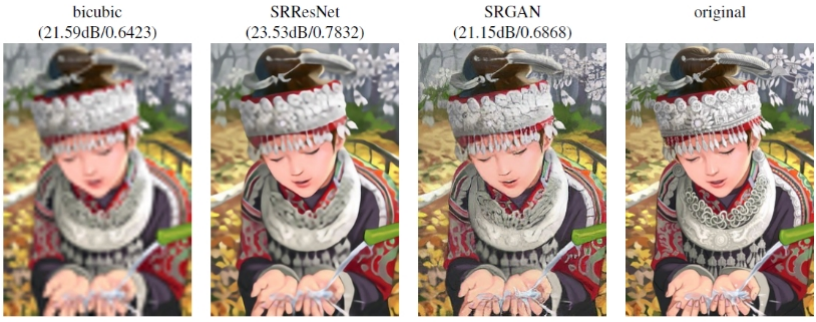


Figure 5: Image Super Resolution[2]

[2]Phillip Isola et al,. Image-to-Image Translation with Conditional Adversarial Networks. CVPR 2017

Applications

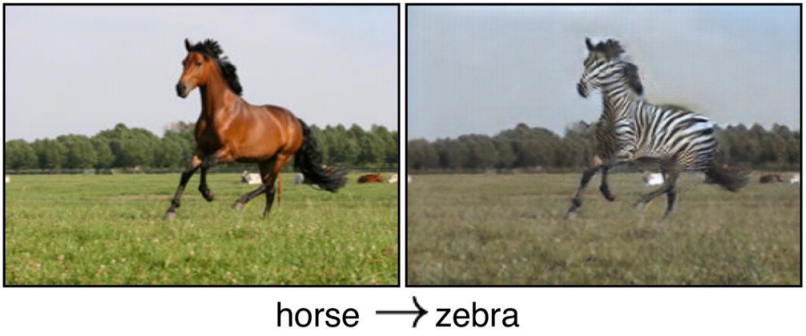


Figure 6: Image Translation[3]

[3] Jun-Yan Zhu et al,. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. ICCV 2017

Applications

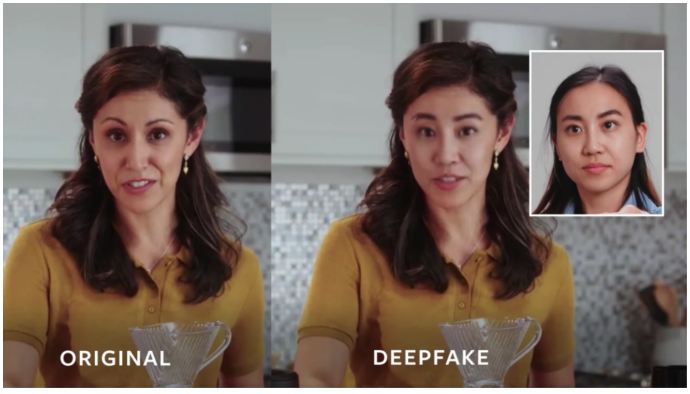


Figure 7: DeepFake[4]

[4] Facebook. DeepFake Detection Challenge. 2019

Applications

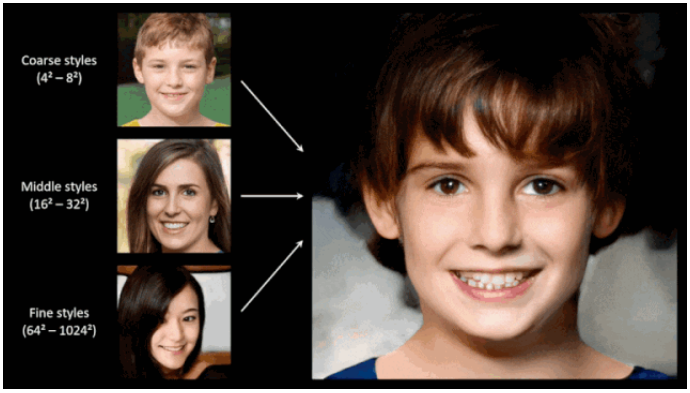


Figure 8: Style Transfer[5]

[5] Tero Karras et al, (NVIDIA Research). A Style-Based Generator Architecture for Generative Adversarial Networks. 2019

Applications

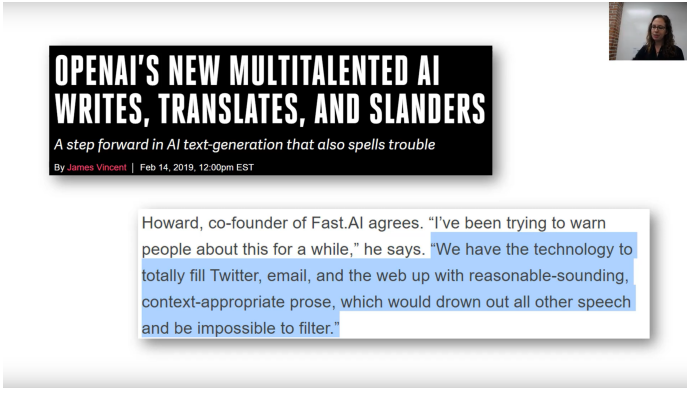


Figure 9: Text Generation[6]

[6] Alec Radford et al,. Language Models are Unsupervised Multitask Learners. 2019

Applications

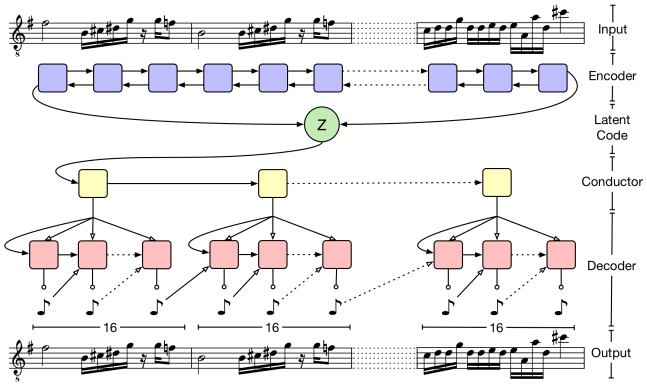


Figure 10: Music Generation[7]

[7] Adam Roberts et al,. A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music. ICML 2018

Related Work

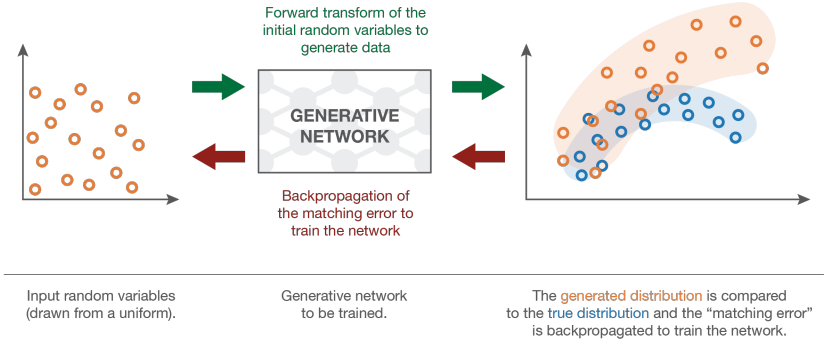
Generative and Discriminative Machine Learning

Let's think about classification problem in supervised way. Given pairs of (features, labels)/(inputs, targets) as (x, y).

- **Discriminative:** Learning to predict the labels from features. It learns the **conditional probability distribution** $\mathcal{P}(y|x)$. Ex: Logistic regression, support vector machine,..
- **Generative:** Learning to describe how the data distribution look like? It learns the **joint probability distribution** $\mathcal{P}(x|y)$. Ex: Naive bayes,..

Deep Generative Model

Generative Matching Network: Comparing the true and the generated probability distributions and backpropagating the difference through network[8].



[8] Joseph Rocca. Understanding Generative Adversarial Networks (GANs). 2019

Variational Autoencoder

There are 2 parts in VAE. Encoder will encode input into a latent space as a latent variable vector and decoder will use this vector to reconstruct input data.

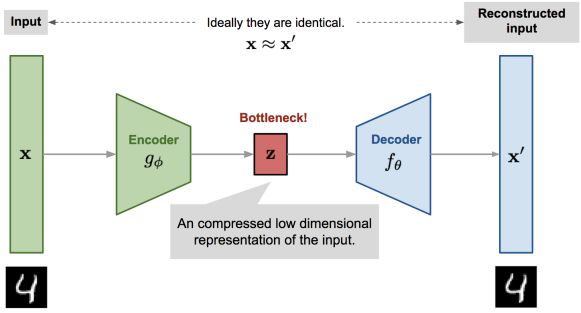


Figure 11: Illustration of autoencoder model architecture[9].

[9] Lilian Weng. From Autoencoder to Beta-VAE. 2018

Flow-based Generative Model

A flow-based generative model is constructed by a sequence of invertible transformations.

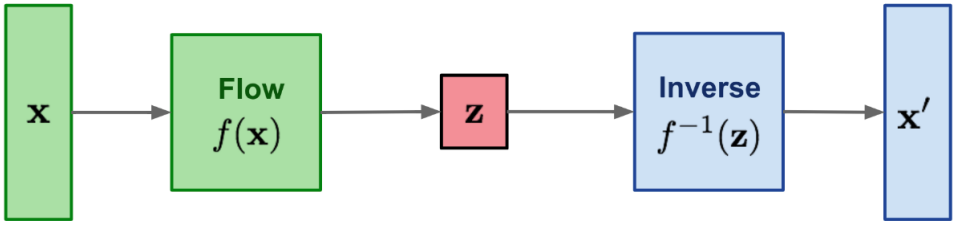


Figure 12: Illustration of Flow-based Generative Model[10].

[10] Lilian Weng. Flow-based Deep Generative Models. 2018

Generative Adversarial Network

Generative Adversarial Network

Goodfellow et al. (2014) proposed a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake[11].

[11] Ian J. Goodfellow et al., Generative Adversarial Nets. NIPS 2014.

Generative Adversarial Network

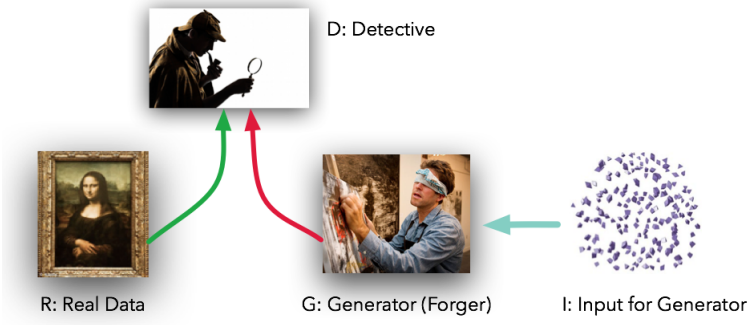
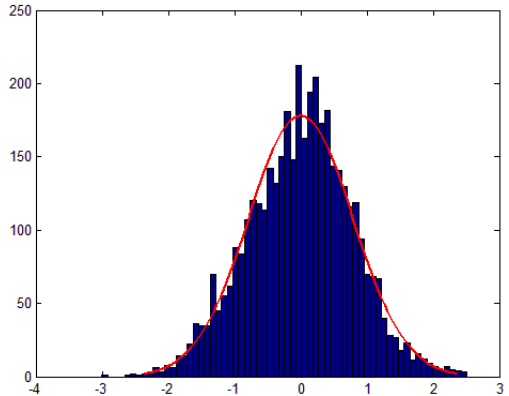


Figure 13: Thinking about discriminator as a detective who must be distinguish real from fake data. And generator is the one who try to fool the generator.

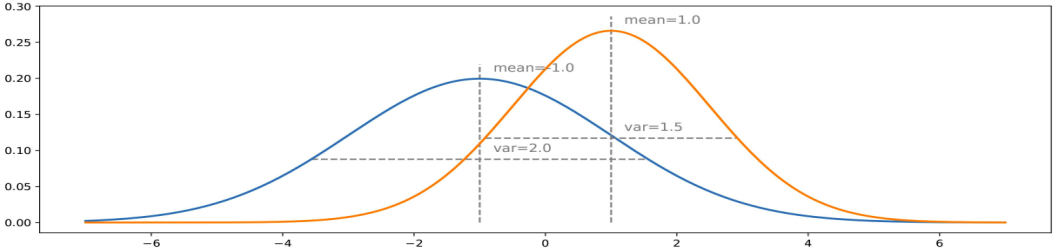
Generative Adversarial Network

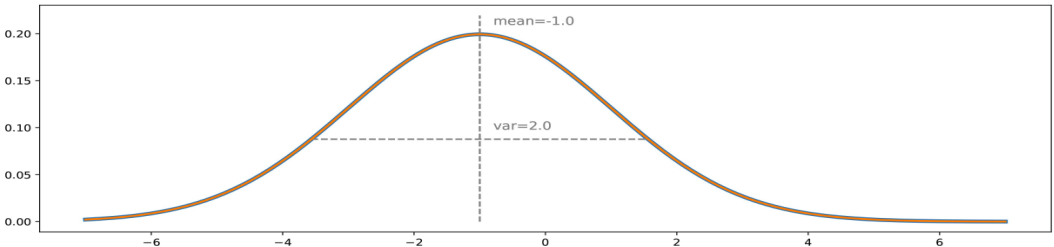
An **unsupervised dataset** serves as a training data for GANs. The generator trains based on whether it succeeds in fooling the discriminator.

$\Longrightarrow$ The generator needs to capture distrubution $\mathcal{P}_g$ over training data x.



Generative Adversarial Network





Generative Adversarial Network

- Sample z from input noise variables $\mathcal{P}_z(z) : z \sim \mathcal{P}_z(z)$
- Generate fake data from z: $x_{fake} = G(z; \theta_g)$. Note that G is a differentiable function represented by a multilayer perceptron with parameters θ_g.
- Another multilayer perceptron $D(x; \theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g.
  $\implies$ D and G play the following two-player minimax game with value function $V(G, D)$.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim \mathcal{P}_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim \mathcal{P}_z}[\log 1 - D(G(z)))]$$

Optimal solution: $\mathcal{P}_g = \mathcal{P}_{data}, D(x) = \frac{1}{2}$

Generative Adversarial Network

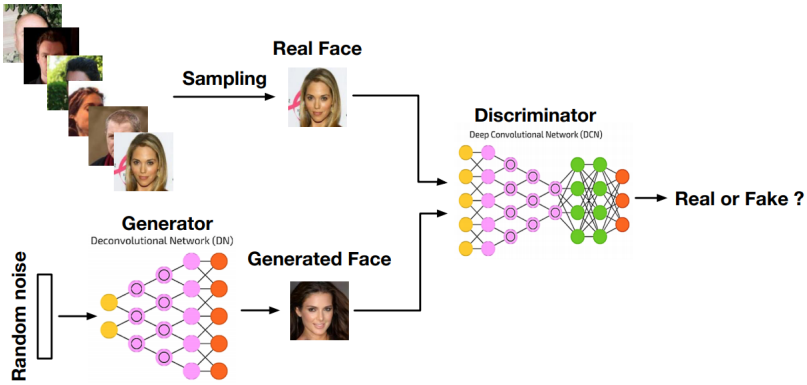


Figure 14: Architecture of a generative adversarial network[12].

[12] Lilian Weng. From GAN to WGAN. 2017.

GAN's Objective Function

- $x_{fake} \leftarrow G(z) \leftarrow z \sim \mathcal{P}_z$
- $x_{real} \leftarrow x \sim \mathcal{P}_{data}$
- $D(x) \leftarrow \mathcal{P}(y|x_{real})$
- $D(G(z)) \leftarrow \mathcal{P}(y|G(z)) \leftarrow \mathcal{P}(y|x_{real})$
- Discirminator: maximize $D(x)$, minimize $D(G(z))$.
- Generator: maximize $D(G(z))$.

GAN's Objective Function

Discriminator

- $D\_loss_{real} = \log D(x)$
- $D\_loss_{fake} = \log(1 - D(G(z)))$
- $D\_loss = \log D(x) + \log(1 - D(G(z)))$
- Total cost:

$$\frac{1}{m}\sum_{i=1}^{m} \log D(x^i) + \log(1 - D(G(z^i)))$$

Generator

- $G\_loss = \log(1 - D(G(z)))$ or $-\log D(G(z))$
- Total cost:

$$\frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z^i)))$$

or

$$\frac{1}{m}\sum_{i=1}^{m} -\log(D(G(z^i)))$$

Advantages and disadvantages

Advantages

Comparing to other generative models e.g., variational autoencoders, GANs offer advantages such as an ability to handle sharp estimated density functions, efficiently generating desired samples, eliminating deterministic bias and good compatibility with the internal neural architecture[13].

[13] Zhengwei Wang et al,. Generative Adversarial Networks: A Survey and Taxonomy. 2019.

Disadvantages[14]

- **Hard to train** - It is non-trivial for discriminator and generator to achieve Nash equilibrium during the training and the generator cannot learn the distribution of the full datasets well, which is known as mode collapse.
- **Hard to evaluate** - the evaluation of GANs can be considered as an effort to measure the dissimilarity between the real distribution $\mathcal{P}_r$ and the generated distribution $\mathcal{P}_g$. Unfortunately, the accurate estimation of pr is not possible.
- Problems about image quality, image diversity and stable training.

[14] Zhengwei Wang et al,. Generative Adversarial Networks: A Survey and Taxonomy. 2019.

GAN-variants

Architecture-variant GANs

- Fully-connected GAN (FCGAN) - The original GAN paper by Ian Goodfellow.
- Laplacian Pyramid of Adversarial Network (LAP-GAN): LAPGAN utilizes a cascade of CNNs within a Laplacian pyramid framework to generate high quality images.
- Deep Convolutional GAN (DCGAN): DCGAN is the first work that applied a deconvolutional neural network architecture for G.
- Boundary Equilibrium GAN (BEGAN) uses an autoencoder architecture for the discriminator which was first proposed in EBGAN.
- Progressive GAN (PROGAN) involves progressive steps toward the expansion of the network architecture.
- Self-attention GAN (SAGAN) deploys a self-attention mechanism in the design of the discriminator and generator architectures for GANs.
- BigGAN design is based on SAGAN and it has been demonstrated that the increase in batch size and the model complexity can dramatically improve GANs performance with respect to complex image datasets.

Architecture-variant GANs

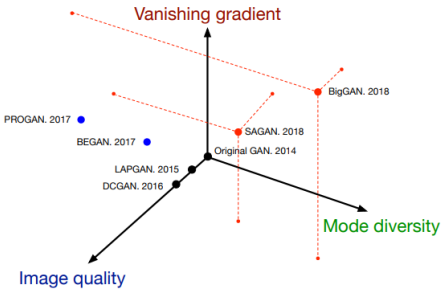


Figure 15: Summary of recent architecture-variant GANs for solving the three challenges. The challenges are categorized by three orthogonal axes. A larger value for each axis indicates better performance. Red points indicate GAN-variants which cover all three challenges, blue points cover two, and black points cover only one challenge.

Loss-variant GANs

- Wasserstein GAN (WGAN) use Earth Mover (EM) distance as a loss measure for optimization.
- Least Square GAN (LSGAN) use least square loss for discriminator instead of sigmoid cross entropy in the vanilla GAN.
- f-GAN summarizes that GANs can be trained by using an f-divergence such as Kullback-Leibler divergence, Jensen-Shannon divergence, Pearson divergence.
- Unrolled GAN (UGAN) is a design proposed to solve the problem of mode collapse for GAN during training.
- Geometric GAN using SVM separating hyperplane, which has the maximal margins between the two classes.
- Relativistic GAN (RGAN) is proposed as a general approach to devising new cost functions from the existing one i.e., it can be generalized for all integral probability metric.

Loss-variant GANs

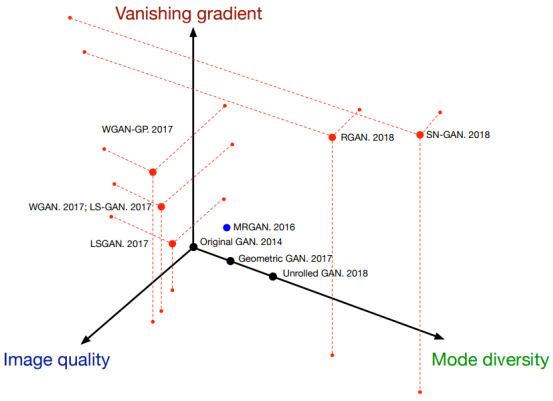
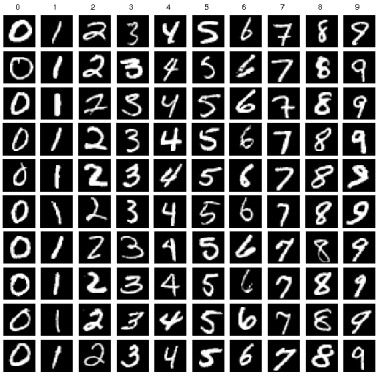


Figure 16: Current loss-variants for solving the challenges. Challenges are categorized in terms of three independent axes. Red points indicate the GAN-variant covers all three challenges, blue points cover two, and black points cover only one challenge. Larger value for each axis indicates better performance.

Experiments

MNIST database

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size (28×28) image.



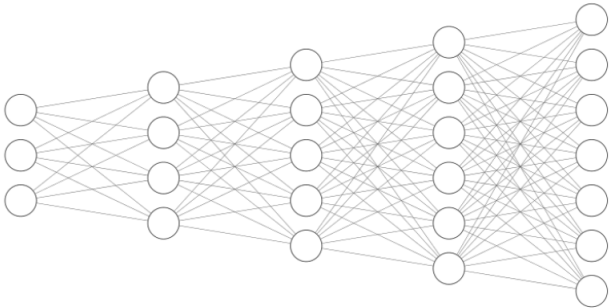
Baseline Model

For evaluating our proposed network, we use base line result from Deep GSN (2012)[15].

[15]Yoshua Bengio, Gregoire Mesnil, Yann Dauphin and Salah Rifai. Better Mixing via Deep Representations. 2012

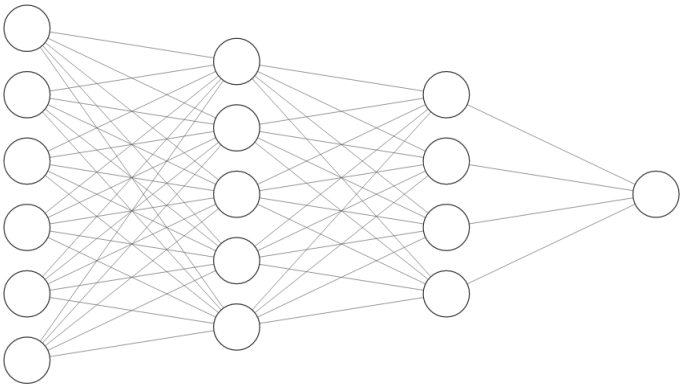
Our Architecture

We implement generator and discriminator as 2 fully connected networks. Generator has 5 FC layers with dim (128, 256, 512, 1024, 784 (28×28)). All first 4 layers use **LeakyReLU** activation function, the last one use **tanh** function.



Our Architecture

Discriminator has 4 FC layers (784, 512, 256, 1). **LeakyReLU** was applied in first 3 layers. In the last layer we use 1 neuron with **sigmoid** funcion.



Implementation

- Use PyTorch framework.
- Modular design, object-oriented programming style.
- Publish as a pip package, install and run easily.
- Our source code was published at: https://github.com/vndee/deepgen

Implementation

```python
class Generator(GeneratorBase):
    def __init__(self, latent_dim=100, img_shape=(1, 28, 28)):
        super(Generator, self).__init__()

        self.img_shape = img_shape
        self.latent_dim = latent_dim

        def block(in_feat, out_feat, normalize=True):
            layers = [nn.Linear(in_feat, out_feat)]

            if normalize:
                layers.append(nn.BatchNorm1d(out_feat, 0.8))

            layers.append(nn.LeakyReLU(0.2, inplace=True))
            return layers

        self.model = nn.Sequential(
            *block(latent_dim, 128, normalize=False),
            *block(128, 256),
            *block(256, 512),
            *block(512, 1024),
            nn.Linear(1024, int(np.prod(img_shape))),
            nn.Tanh()
        )

    def forward(self, z):
        img = self.model(z)
        img = img.view(img.size(0), *self.img_shape)
        return img
```

```python
class Discriminator(DiscriminatorBase):
    def __init__(self, img_shape=(1, 28, 28)):
        super(Discriminator, self).__init__()

        self.img_shape = img_shape

        self.model = nn.Sequential(
            nn.Linear(int(np.prod(img_shape)), 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, img):
        img_flat = img.view(img.size(0), -1)
        validity = self.model(img_flat)
        return validity
```

Usage

- Install deepgen package:
  pip3 install deepgen
- Create a file example.py like this and run:
  python3 example.py
- See console output and sample images in folder **images.**

```
import os
import torch

from deepgen.gan.gan import GAN
from torchvision import datasets
import torchvision.transforms as transforms

img_size = (1, 28, 28)
batch_size = 64

if __name__ == '__main__':
    data_loader = torch.utils.data.DataLoader(
        datasets.MNIST(
            '../data/',
            train=True,
            download=True,
            transform=transforms.Compose(
                [transforms.Resize(28), transforms.ToTensor(), transforms.Normalize([0.5], [0.5])]
            ),
        ),
        batch_size=batch_size,
        shuffle=True,
    )

    os.makedirs('images', exist_ok=True)

    model = GAN()

    print(model)
    his = model.train(data_loader=data_loader, n_epoch=5, sample_interval=10)
    print(his)
```

Result

Original MNIST

Deep GSN

GAN

Demo Applications

Visit http://truongnhathoang.media:8000 to play with a toy example.

Demo Applications

Running locally:

- Install some packages: torch, torchvision, django, ..
- Download model and put it in backend_core folder: https://bit.ly/2DB2eAc
- Run: **python3 manage.py runserver**

Discussion

- As we know GAN is really hard to train. We were trained too many times to achieve expected performance.
- Because of simple architecture our output are not a state-of-the-art generation model.
- We learned lots of things from coding, debugging to research in this project.

Future Work

- Discovering state-of-the-art research in generative adversarial networks.
- Improving performance of our model and training with other datasets.
- In this project it seems that we ignore the use of discriminator. We will consider the use of discriminator in the field of prevent adversarial attacks and security threats.

Q&A