知能プログラミング演習 I 第 4 回レポート

平成 31 年 8 月 2 日
学籍番号 29114154
氏名 PHAM DUY

1　レポートのテーマと構成

- テーマ
  最近の深層学習について話題となっている generative adversarial network(GAN) の画像生成
- 構成
  - GAN の内容
  - keras を用いた GAN の生成モデルの実装

2　内容

2.1　概要

GAN(Generative Adversarial Networks 日本語訳:「敵対的生成ネットワーク」) は生成モデルの一種であり、データから特徴を学習することで、実在しないデータを生成したり、存在するデータの特徴に沿って変換できる。GAN は、正解データを与えることなく特徴を学習する「教師なし学習」の一手法として注目されている。そのアーキテクチャの柔軟性から、アイデア次第で広範な領域に摘用できる。応用研究や理論的研究も急速に進んでおり、今後の発展が大いに期待されている。

図 1: GAN による画像生成

2.2 GAN の発展の背景とその特徴

- 発展の背景

  GAN は、イアン・グッドフェローらが 2014 年に発表した論文で、2 つのネットワークを競わせながら学習させるアーキテクチャとして提案された。この斬新な構成により、従来の生成モデルより鮮明で本物らしい画像生成が可能になった。

  さらに 2015 年には、畳み込みニューラルネットワーク（CNN）で見られるような畳み込み層をネットワークに適用した DCGAN（Deep Convolutional GAN）が提案された。後述のように、GAN は学習時に不安定になるケースが見られ、意味をなさない画像を生成したり、生成データの種類が偏るなどの課題があった。

- その特徴

  GAN は生成モデルであり、データの特徴を抽出して学習し、実在しないデータを生成できる。生成モデルに分類される手法としては、変分オートエンコーダやボルツマンマシンなども以前からあるが、GAN はそれらの手法と比べてより鮮明な画像の生成が可能である。

2.3 GAN の学習の仕組み

GAN は、2 つのニューラルネットワークで構成される。

GAN は生成器と識別器がお互い、訓練中にナッシュ均衡を達成しようとする。GAN のアーキテクチャを図 1 に示す。生成器 G の動作原理は実際のデータの潜在分布を極力適合させるために偽のデータを生成することである。一方で、識別器 D の動作原理は偽か実際のデータか正しく見分けることである。生成器の入力はランダムノイズベクトル z(基本的には一様分布もしくは正規分布) である。ノイズは多次元ベクトルである偽のサンプル G(z) を得るために生成器 G を介して新しいデータ空間にマッピングされる。また、識別器 D は二値分類器でありデータセットからの実際のサンプルもしくは生成器 G から生成された偽のサンプルを入力として受け取る。そして、識別器 D の出力は実際のデータである確率である。識別器 D が実際のものか偽のものかどうかわからなくなった時、GAN は最適な状態になる。この時点で、実際のデータ分布を学習した生成器モデル G が得られる。

2.4 GAN の学習モデル

$J^{(\mathrm{G})}$ と $J^{(\mathrm{D})}$ をそれぞれ識別器損失関数と生成器の損失関数とする。識別器 D が二値分類器として定義され、損失関数はクロスエントロピーで示される。定義は式 (1) の通り。

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{x\sim pdata} log D(x) - \frac{1}{2}\mathbb{E}_z log(1 - D(G(z))) \tag{1}$$

ここで、x は実際のサンプルを示し、z は $G(z)$ を生成器 G で生成するためのランダムノイズベクトル、$\mathbb{E}$ は期待 (期待値、expectation) である。$D(x)$ は D が x を実際のデータとみなす確率、$D(G(z))$ は D が G によって生成されたデータを特定する確率を示す。D の目的はデータの出所を正しく突き止めることであるため、$D(G(z))$ が 0 に近づくことを目標とするが、G は 1 に近づくことを目的とする。この考えに基づいて、2 つのモデル間には対立が存在する (ゼロサムゲーム)。したがって、生成器の損失

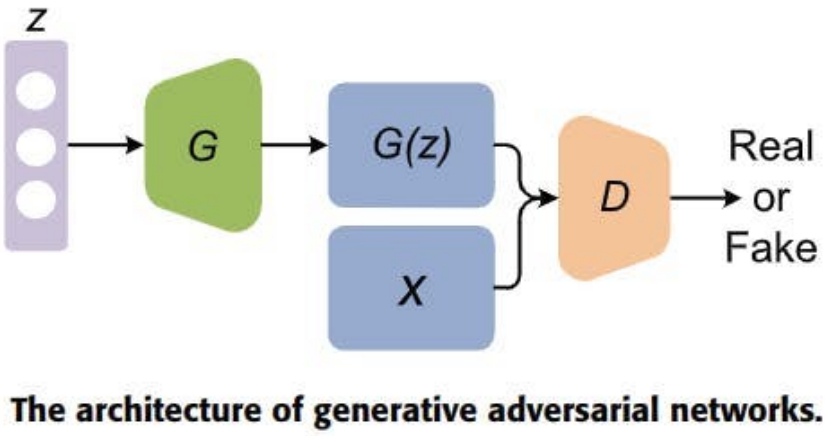

図 2: GAN のアーキテクチャ

は識別機によって式 (2) の様に導出される。

$$J^{(G)} = -J^{(D)} \tag{2}$$

結果的に、GAN の最適化問題は minimax ゲームに変換される。定義は式 (3) の通り。

$$\min_G max_D(D,G) = \mathbb{E}_{x\sim pdata(x)}[logD(x)] + \mathbb{E}_{z\sim p(z)}[log(1-D(G(z)))] \tag{3}$$

訓練プロセス中に G 中のパラメーターは D の更新プロセスのパラメーターと一緒に更新される。$D(G(z)) = 0.5D(G(z)) = 0.5$ である時、識別機はこれらの 2 つの分布間の差異を特定することができなくなる。この状態では、モデルが大域的最適解を達成するだろう。

2.5 派生 GAN モデル

オリジナルの GANs の欠陥により、様々な派生 GANs モデルが提案され、これらの派生 GANs モデルはアーキテクチャ最適化ベースの GANs と目的関数最適化ベースの GANs の 2 種類のグループに分けられる（図 3 ）。このセクションでは、いくつかの派生モデルの詳細について紹介する。

TABLE 1. Classification of GANs models.

Architecture Optimization Based GANs	Convolution based GANs	DCGAN [12]
	Condition based GANs	CGANs [13]; InfoGAN [14]; ACGAN [15]
	Autoencoder based GANs	AAE [16]; BiGAN [17]; ALI [18]; AGE [19]; VAE-GAN [20]
Objective Function Optimization Based GANs	unrolled GAN [21]; f-GAN [22]; Mode-Regularized GAN [23]; Least-Square GAN [24]; Loss-Senstive GAN [25]; EBGAN [26]; WGAN [27]; WGAN-GP [28]; WGAN-LP [29]	

図 3: 派生 GAN モデル

GAN の応用研究

- **Conditional GAN（条件付きベースの GAN）**
  通常の GAN のデータ生成は、ランダムにサンプルされるので、生成されるデータの種類を制御できない。Conditional GAN は学習時にラベルを与えることで、推論時にもラベルを入力してデータを生成できる。図 4 はその例で、数字の種類を指定してデータを生成している。

図 4: 条件付きベースの GAN によるデータ生成（MNIST データセット）

- **SRGAN**
  低解像度の画像を高解像度に復元する、超解像を目的とした GAN である。Bicubic 法のような従来の超解像手法による復元では、ぼやけた画像になりやすい。SRGAN では、GAN の特性を利用することで、ぼやけの少ない画像の復元が可能である。

- **pix2pix**
  pix2pix は画像間の特徴を変換する。図 5 では、セグメンテーションされた景色に対して、たとえば「航空写真を地図化する」「日中の写真を夜間のシーンに変換する」などを行っており、入力時に変換前の画像をラベルとして与える conditional な設定と考えられる。「ある特徴をもつ画像から、別の特徴へ変換する」と捉えると、応用アイデアが広がる。

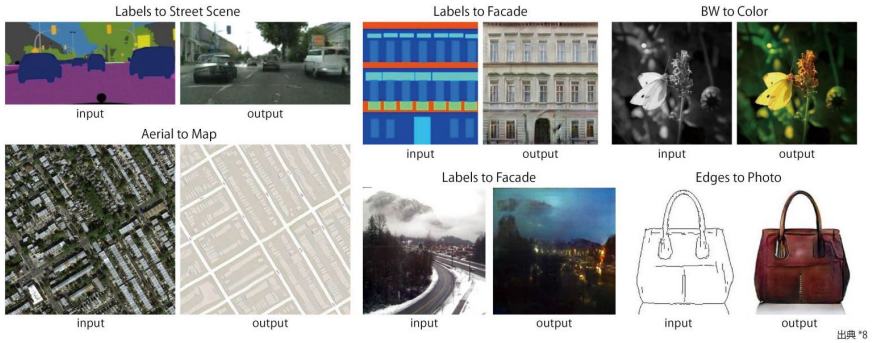

図 5: pix2pix により画像間の特徴を変換する

- **Cycle GAN**
  pix2pix と同じく、画像の特徴間の変換を実行する。ウマとシマウマを変換した図 6 がこの手法を使用している。ここでは学習データの変換元と変換先の対応付けが必要なく、共通した特徴（ドメインと呼ぶ）をもつ画像を集めて、Generator と Discriminator のそれぞれに学習させることで、各特徴間を相互に変換する。

図 6: Cycle GAN による馬とシマウマの変換

3　keras を用いた GAN の生成モデルの実装

今回 ConditionalGAN を Keras で実装する。ソースコード

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras
from keras.models import Sequential, Model
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Activation, Flatten, In
from keras.datasets import mnist
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
import numpy as np
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=config)
import time


def G_model(Height, Width, channel=3):
    inputs = Input((100,))
    in_h = int(Height / 4)
    in_w = int(Width / 4)
    x = Dense(in_h * in_w * 128, activation='tanh', name='g_dense1')(inputs)
    x = BatchNormalization()(x)
    x = Reshape((in_h, in_w, 128), input_shape=(128 * in_h * in_w,))(x)
    x = UpSampling2D(size=(2, 2))(x)
    x = Conv2D(64, (5, 5), padding='same',
               activation='tanh', name='g_conv1')(x)
    x = UpSampling2D(size=(2, 2))(x)
    x = Conv2D(channel, (5, 5), padding='same',
               activation='tanh', name='g_out')(x)
    model = Model(inputs, x, name='G')
    return model


# の定義Discreminator
def D_model(Height, Width, channel=3):
    inputs = Input((Height, Width, channel))
    x = Conv2D(64, (5, 5), padding='same',
               activation='tanh', name='d_conv1')(inputs)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Conv2D(128, (5, 5), padding='same',
               activation='tanh', name='d_conv2')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Flatten()(x)
    x = Dense(1024, activation='relu', name='d_dense1')(x)
    x = Dense(1, activation='sigmoid', name='d_out')(x)
    model = Model(inputs, x, name='D')
    return model


def Combined_model(g, d):
    model = Sequential()
    model.add(g)
```

```
    model.add(d)
    return model

# for output
def save_images(imgs, index, dir_path):
    # Argment
    #  img_batch = np.array((batch, height, width, channel)) with value range [-1, 1]
    B, H, W, C = imgs.shape
    batch = imgs * 127.5 + 127.5
    batch = batch.astype(np.uint8)
    w_num = np.ceil(np.sqrt(B)).astype(np.int)
    h_num = int(np.ceil(B / w_num))
    out = np.zeros((h_num*H, w_num*W), dtype=np.uint8)
    for i in range(B):
        x = i % w_num
        y = i // w_num
        out[y*H:(y+1)*H, x*W:(x+1)*W] = batch[i, ..., 0]
    fname = str(index).zfill(len(str(3000))) + '.jpg'
    save_path = os.path.join(dir_path, fname)

    plt.imshow(out, cmap='gray')
    plt.title("iteration: {}".format(index))
    plt.axis("off")
    plt.savefig(save_path)

if __name__ == '__main__':
    g = G_model(Height=28, Width=28, channel=1)
    d = D_model(Height=28, Width=28, channel=1)
    c = Combined_model(g=g, d=d)

    g_opt = keras.optimizers.Adam(lr=0.0002, beta_1=0.5)
    d_opt = keras.optimizers.Adam(lr=0.0002, beta_1=0.5)

    g.compile(loss='binary_crossentropy', optimizer='SGD')
    d.trainable = False
    for layer in d.layers:
        layer.trainable = False
    c.compile(loss='binary_crossentropy', optimizer=g_opt)
    d.trainable = True
    for layer in d.layers:
        layer.trainable = True
    d.compile(loss='binary_crossentropy', optimizer=d_opt)
```

```
# MNIST DATASET
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = (X_train.astype(np.float32) - 127.5)/127.5
X_train = X_train[:, :, :, None]
train_num = X_train.shape[0]
train_num_per_step = train_num // 64

# config
Minibatch = 64
Save_test_img_dir = "./"
iterator = 30000
# trainning
for ite in range(iterator):
    start = time.time()
    ite += 1
    # Discremenator training
    train_ind = ite % (train_num_per_step - 1)
    y = X_train[train_ind * Minibatch: (train_ind+1) * Minibatch]
    input_noise = np.random.uniform(-1, 1, size=(64, 100))
    g_output = g.predict(input_noise, verbose=0)
    X = np.concatenate((y, g_output))
    Y = [1] * 64 + [0] * 64
    d_loss = d.train_on_batch(X, Y)
    # Generator training
    input_noise = np.random.uniform(-1, 1, size=(Minibatch, 100))
    g_loss = c.train_on_batch(input_noise, [1] * Minibatch)
    end = time.time()
    print("ite {}, d_loss {}, g_loss {}".format(ite,d_loss,g_loss))

save_images(g_output, index=5, dir_path=Save_test_img_dir)
```

出力結果

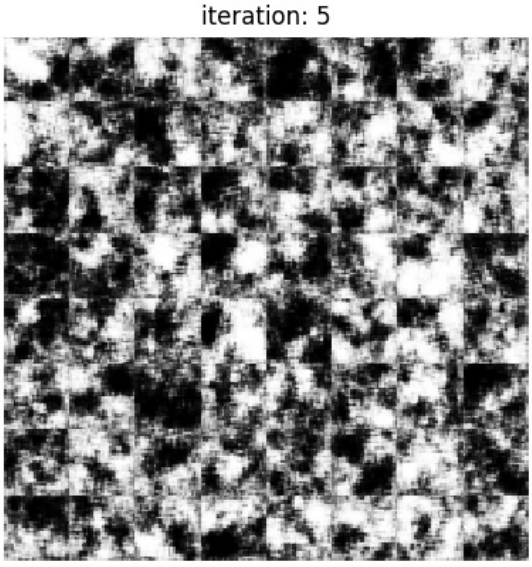

図 7: ConditionalGAN による画像生成の例（MNIST DATASET より）