機械学習勉強会2.0

GAN

GANの概要

GAN (Generative Adversarial Networks)

画像生成手法の一つ

progressive GAN

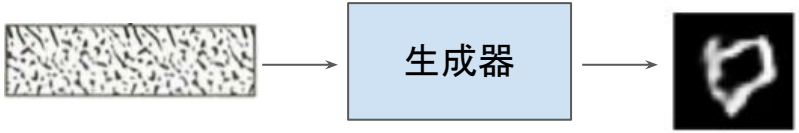
https://arxiv.org/abs/1710.10196

cycle GAN

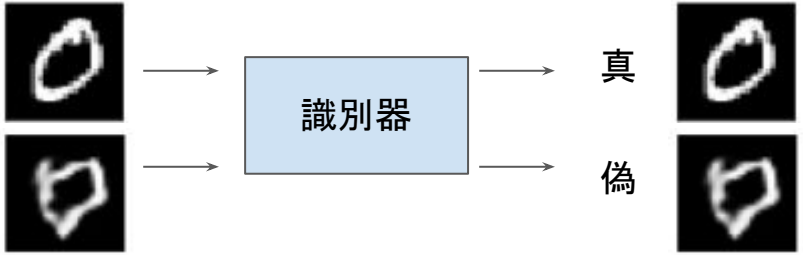
https://arxiv.org/abs/1703.10593

GANの概要

生成器：本物のデータと見分けが
付かないような偽のデータを作り出す

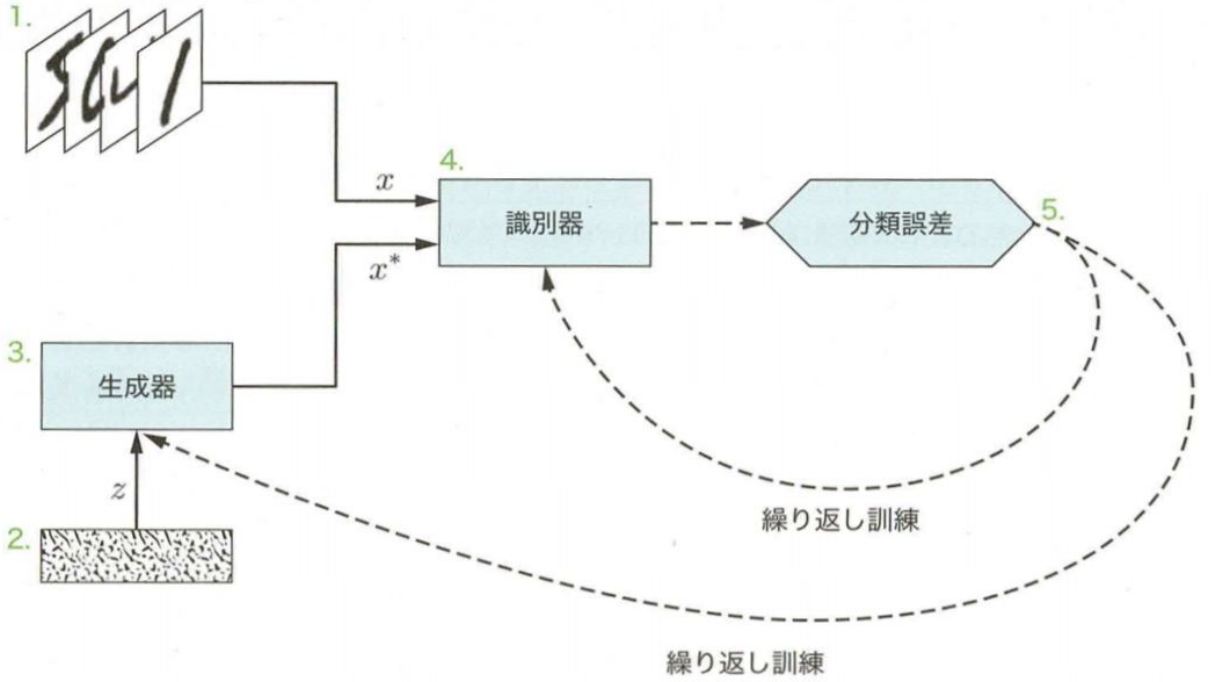


識別器：本物のデータと、生成器が作り出した
偽のデータを区別する



敵対的：生成器と識別器がゲームのような競争的な動作をする

GAN全体図



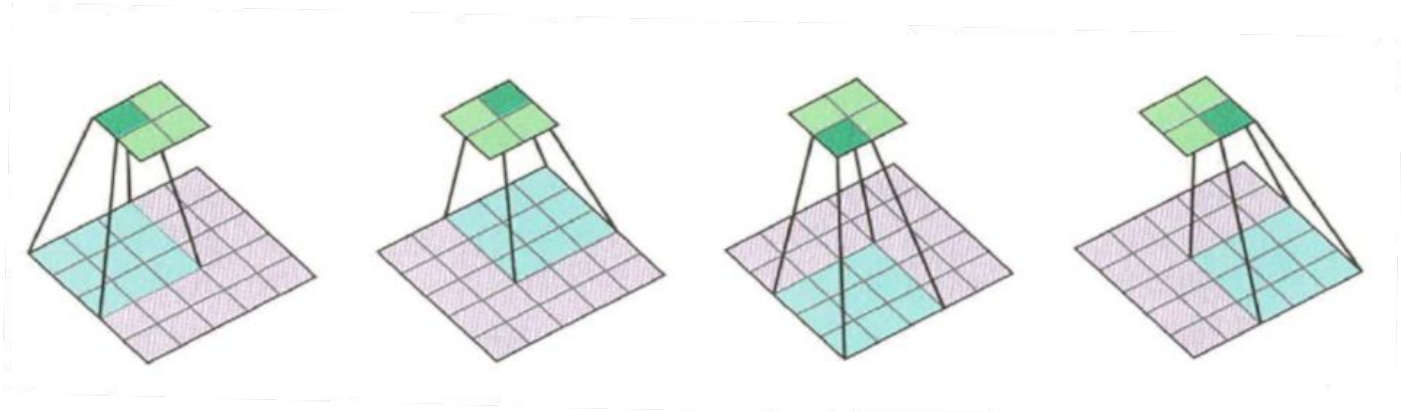
ナッシュ均衡に
落ち着くまで繰り返す。

実践GAN 敵対的生成ネットワークによる深層学習 初版 p. 8、Jakub Langr, Vladimir Bok 著、大和田 茂 訳、マイナビ出版

DCGAN (Deep Convolutional GAN)

ConvNetとGANを組み合わせたもの

2016年にAlec Radford, Luke Metz, Soumith Chintalaらによって提案 [1)]



“A Guide to Convolution Arithmetic for Deep Lerning”, https://arxiv.org/abs/1603.07285

1) https://arxiv.org/abs/1511.06434

GAN演習

演習

google colaboratoryでの実習を行う際には、下記リンクよりアクセス下さい。
https://colab.research.google.com/github/academeia/machine-learning-seminar_2020/blob/master/GAN/GAN.ipynb

ランタイムのタイプを変更することで、GPUによる計算が可能です。

初期設定、各種インポート

```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.2.0'
```

```
# To generate GIFs
!pip install imageio
```

```
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time

from IPython import display
```

データの取得、バッチ化

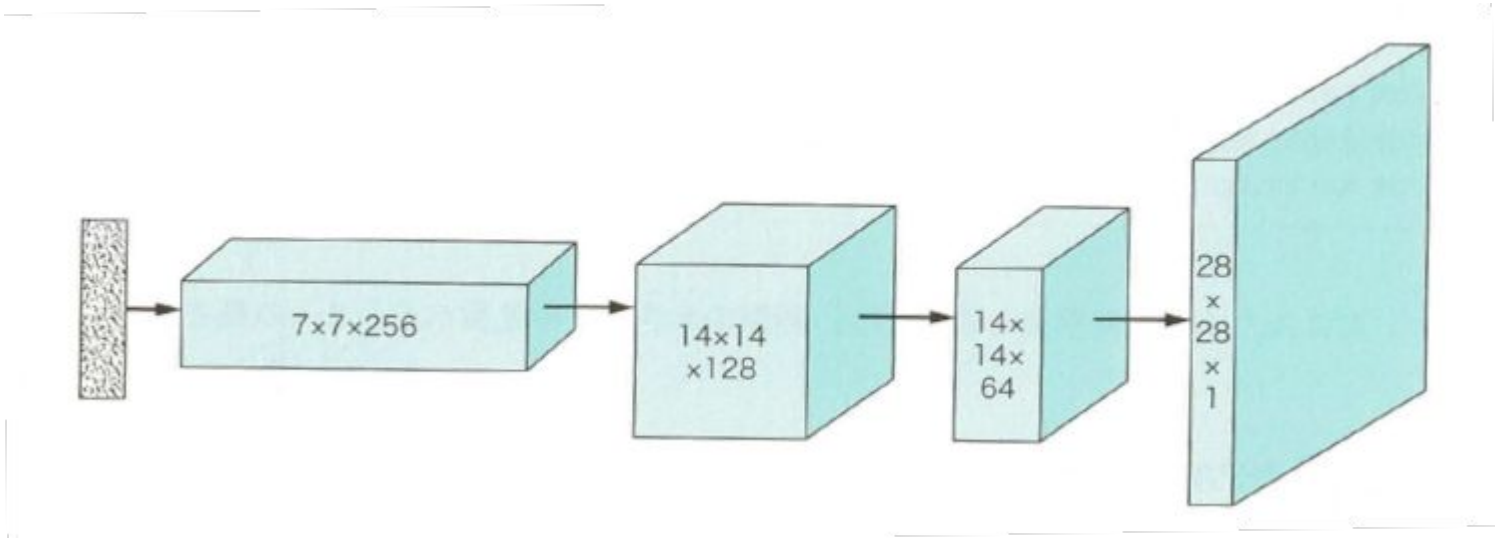
```
(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
```

```
train_images = train_images.reshape(train_images.shape[0],
                                    28, 28, 1).astype('float32')
train_images = (train_images - 127.5) / 127.5
                                  # Normalize the images to [-1, 1]
```

```
BUFFER_SIZE = 60000
BATCH_SIZE = 256
```

```
# Batch and shuffle the data
train_dataset = \
tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batc
h(BATCH_SIZE)
```

生成器のモデル概略



生成器のモデル関数 (1)

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)
                                        # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1),
                                     padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
```

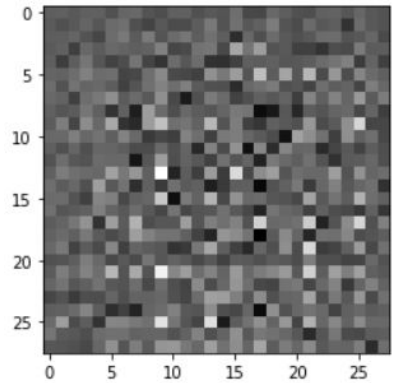
生成器のモデル関数 (2)

```
    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2),
                                     padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2),
                                     padding='same', use_bias=False,
                                     activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

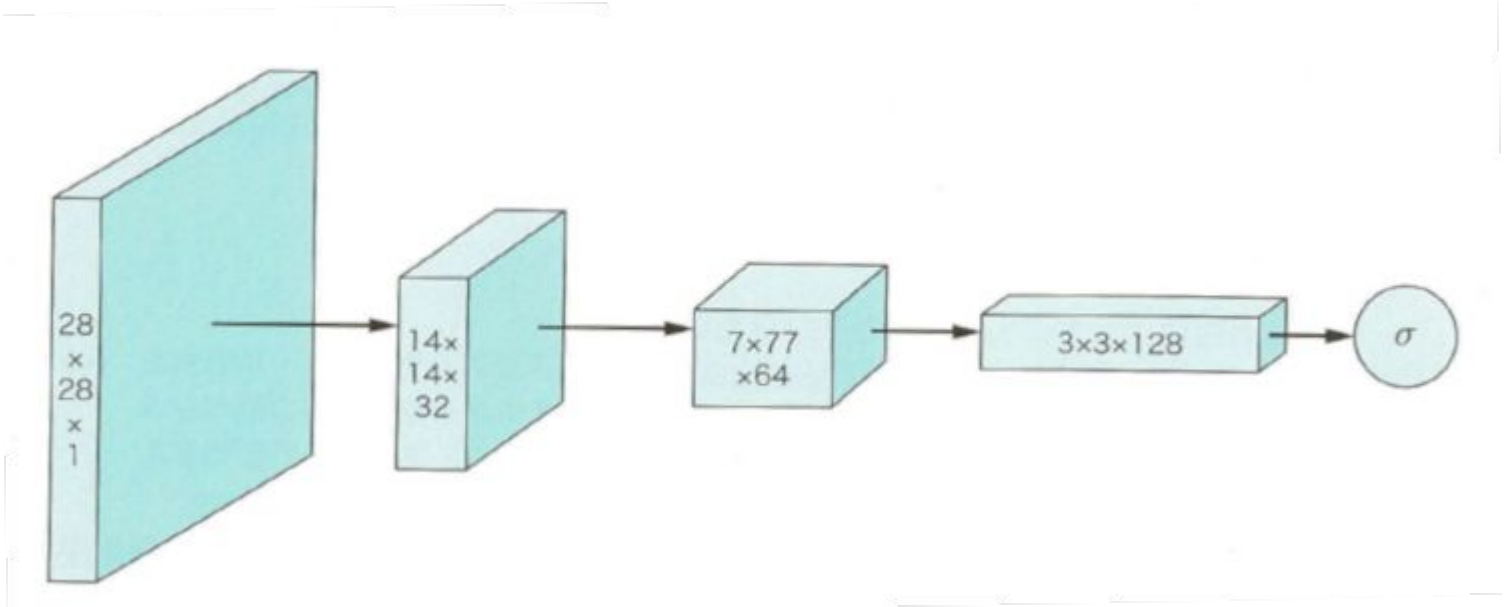
生成器のモデル、初期画像

```
generator = make_generator_model()

noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

```
<matplotlib.image.AxesImage at
```



識別器のモデル概略



識別器のモデル関数

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                     input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

識別器、交差エントロピー

```
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)
```

```
tf.Tensor([[-0.00156255]], shape=(1, 1), dtype=float32)
```

```
# This method returns a helper function to compute cross entropy loss
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

交差エントロピー

$$H(p,q) = -\sum_{x} p(x)\log q(x)$$

p: 正解の確率分布、*q*: 予測の確率分布

損失関数、最適化アルゴリズム

```
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

```
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

$$H(p,q) = -\sum_{x} p(x)\log q(x)$$

チェックポイント、各種パラメーター

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,

discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
```

```
EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16

# We will reuse this seed overtime (so it's easier)
# to visualize progress in the animated GIF)
seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

訓練の各ステップ (1)

```
# Notice the use of `tf.function`
# This annotation causes the function to be "compiled".
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)
```

訓練の各ステップ (2)

```
    gradients_of_generator = \
        gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = \
        disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator,
                                            generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,

discriminator.trainable_variables))
```

訓練用関数 (1)

```
def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch)

    # Produce images for the GIF as we go
    display.clear_output(wait=True)
    generate_and_save_images(generator, epoch + 1, seed)
```

画像の生成と保存

```
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)

  fig = plt.figure(figsize=(4,4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

訓練

```
train(train_dataset,
EPOCHS)
```

```
Time for epoch 10 is 711.0001904964447 sec
```

```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir)
)
```