

# 105. 深層学習の適用方法 (GAN)

秋葉洋哉

2024 年 7 月 15 日

## 1 GAN

### 1.1 概要

GAN(Generative Adversarial Networks) は、生成モデルの一つであり、生成モデルは、データの分布を推定し、新しいデータを生成するためのモデルである。GAN は、生成器 (Generator) と識別器 (Discriminator) の 2 つのネットワークを用いて学習を行う。ある分布  $p_z$  を Generator  $G$  に入力したときの出力  $p_g$  を、Discriminator  $D$  に入力する。すると、 $D$  は、 $p_g$  が  $p_z$  から生成されたデータであるか、実際のデータであるかを判定する。 $G$  は、 $D$  が  $p_g$  を実際のデータと誤認識するように学習し、 $D$  は、 $G$  が生成したデータを実際のデータと区別するように学習する。このように、 $G$  と  $D$  が互いに学習することで、生成モデルを学習することができる。

この 2 者は、ゼロサムゲームの関係にある。つまり、 $G$  が勝つ確率と  $D$  が勝つ確率の和は 1 になる。 $G$  は、 $D$  が  $p_g$  を実際のデータと誤認識する確率を最小化し、 $D$  は、 $G$  が生成したデータを実際のデータと区別する確率を最小化することを目指す。

- $G$  : 相手が勝利する確率を最小化する作戦を取る
- $D$  : 自分の勝利する確率を最大化する作戦を取る

これを数式で表すと、以下のようになる。

$$\min_G \max_D V(D, G) \tag{1}$$

$$\text{s.t. } V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \tag{2}$$

ただし、 $V(D, G)$  は、 $D$  と  $G$  のゲームの価値関数であり、 $p_{data}(x)$  は、実際のデータの分布、 $p_z(z)$  は、生成器に入力するノイズの分布である。また、 $x$  は、実際のデータ、 $z$  は、ノイズである。また、 $\mathbb{E}$  は、期待値を表す。

生成データが本物のようなデータを生成するのはなぜだろうか。以下では、 $p_g = p_{data}$  であると仮定したときに、価値関数が最適化されていることを示す。まず、Generator の出力がすべて真のデータであったとす

る。すると、 $V$  は以下のように展開できる。

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(x))] \quad (3)$$

$$= \int_x p_{data}(x) \log D(x) dx + \int_z p_z(z) \log(1 - D(x)) dz \quad (4)$$

$$= \int_x (p_{data}(x) \log D(x) + p_g(x) \log(1 - D(x))) dx \quad (5)$$

$$= \int_x a \log(y) + b \log(1 - y) dx \quad (6)$$

ただし、 $a = p_{data}(x)$ 、 $b = p_g(x)$ 、 $y = D(x)$  と置いた。この時の極大値は、

$$\frac{\partial V}{\partial y} = \frac{a}{y} - \frac{b}{1 - y} = 0 \quad (7)$$

$$\therefore y = \frac{a}{a + b} \quad (8)$$

$$= \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \quad (9)$$

となる。この条件を満たすとき、 $V$  は最大値を取る。さらに、価値関数の  $D(x)$  を  $\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$  に置き換えると、

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (10)$$

$$= \mathbb{E}_{x \sim p_{data}(x)} \log \left[ \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \right] + \mathbb{E}_{x \sim p_g(x)} \log \left[ \frac{p_g(x)}{p_{data}(x) + p_g(x)} \right] \quad (11)$$

$$= \mathbb{E}_{x \sim p_{data}(x)} \log \left[ \frac{2p_{data}(x)}{p_{data}(x) + p_g(x)} \right] + \mathbb{E}_{x \sim p_g(x)} \log \left[ \frac{2p_g(x)}{p_{data}(x) + p_g(x)} \right] - 2 \log 2 \quad (12)$$

$$= 2JS(p_{data} || p_g) - 2 \log 2 \quad (13)$$

ただし、 $JS$  は、ジェンセン・シャノンダイバージェンスであり、 $JS$  が最小となるとき、 $p_{data}(x) = p_g(x)$  となる。つまり、この価値関数に従うことで、 $p_{data}(x) = p_g(x)$  となる時に、最適解となることがわかる。

ここで、クロスエントロピー誤差関数について思い出してみると、クロスエントロピー誤差関数は、

$$L = -y \log p - (1 - y) \log(1 - p) \quad (14)$$

で表せた。ただし、 $y$  は、正解ラベル、 $p$  は、予測値である。例えば、真のデータを  $p$  に入力するときは、 $y = 1$  となり、 $p = D(x)$  が入力される。この時、 $L = -\log D(x)$  となる。また、Generator によって生成されたデータを  $p$  に入力するときは、 $y = 0$  となり、 $p = D(G(z))$  が入力される。この時、 $L = -\log(1 - D(G(z)))$  となる。これらの 2 つの  $L$  を足し合えると

$$L = -\log D(x) - \log(1 - D(G(z))) \quad (15)$$

となる。これは単一のデータの場合の式 (2) と等価である。つまり、GAN の学習は、クロスエントロピー誤差関数の期待値を用いて、 $D$  と  $G$  の学習を行っていくことになる。

## 1.2 GAN の学習

GAN の学習は、以下の手順で行う。

1. Generator のパラメータ  $\theta_g$  を固定し、真のデータと生成データを  $m$  個ずつ取り出す。
2. Discriminator のパラメータ  $\theta_d$  を勾配上昇法で更新する。
3. Discriminator のパラメータ  $\theta_d$  を固定し、生成データを  $m$  個取り出す。
4. Generator のパラメータ  $\theta_g$  を勾配降下法で更新する。

それぞれの更新式は以下で表せる。

$$\theta_d \leftarrow \theta_d + \alpha \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \right] \quad (16)$$

$$\theta_g \leftarrow \theta_g - \alpha \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^{(i)}))) \quad (17)$$

ただし、 $\alpha$  は、学習率である。GAN では、 $\theta_d$  を  $k$  回更新し、 $\theta_g$  を 1 回更新することが一般的である。GAN の学習過程を図 1 に示す。

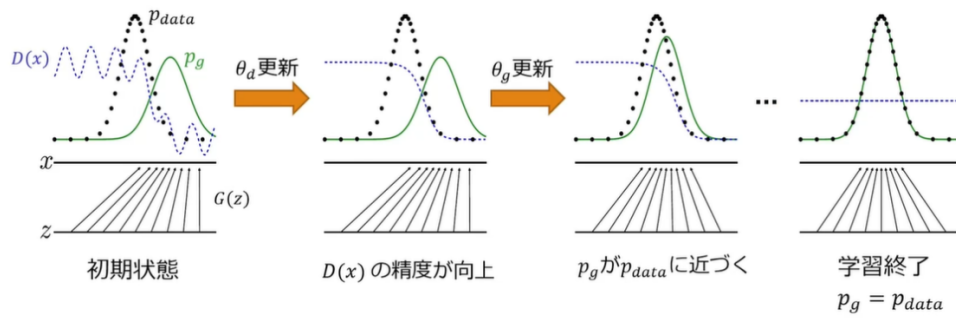


図 1: GAN の学習過程

#### Discriminator の実装例

```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
def discriminator_loss(real_output, fake_output):
    # 本物の画像に対する Discriminator の予測を 1 の配列と比較
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    # 生成された画像に対する Discriminator の予測を 0 の配列と比較
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

## 2 DCGAN

### 2.1 概要

DCGAN(Deep Convolutional Generative Adversarial Networks) は、GAN の一種であり、CNN(Convolutional Neural Networks) を用いて画像生成を行うモデルである。DCGAN は、以下の構造制約を設けることで、生成品質を向上させることができる。

- Generator : Pooling 層の代わりに、転置畳み込み層 (Transposed Convolutional Layer) を用いるを用い、最終層に tanh, 他の層に ReLU を用いる。
- Discriminator : Pooling 層の代わりに、畳み込み層を用い、最終層に Sigmoid, 他の層に LeakyReLU を用いる。
- 共通事項 : 中間層に全結合層を用いず、Batch Normalization を用いる。

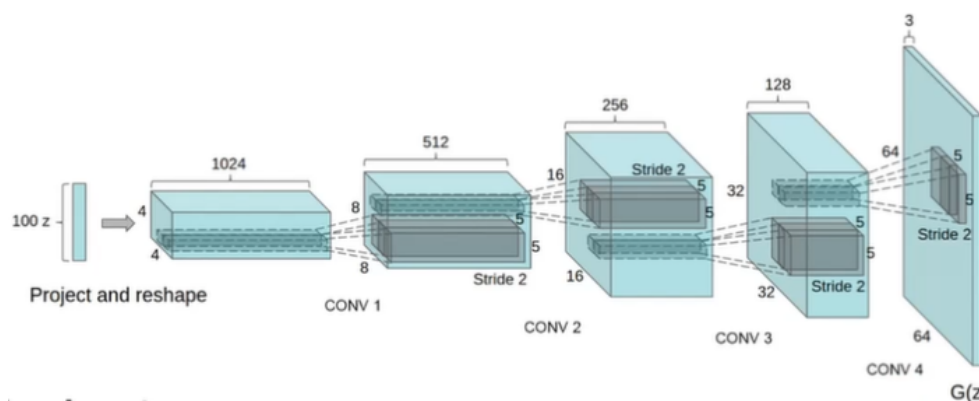


図 2: DCGAN の構造: 乱数を与えると、 $4 \times 4 \times 1024$  の画像を生成し、そのチャンネル数を減らして縦横を増やしていくことで、 $64 \times 64 \times 3$  の RGB 画像を出力する。

### 2.2 Fast Bi-layer Neural Synthesis of One-Shot Realistic Head Avatars

Zakharov et al. (2020) では、DCGAN を用いて、1 枚の顔画像から、顔の動画 (Avatar) を高速で生成するモデルを提案した。この論文では、初期化部と推論部から顔アバターを生成するモデルを提案した。初期化では、人物の特徴を抽出し、1 アバターにつき 1 回だけ実行される。推論部では、初期化部で生成された特徴を元に、所望の動きを付ける。時間フレームの分だけ推論部が実行され、動画が生成される。従来のモデルと比較すると、初期化の計算コストを大きくして、推論部の計算コストを削減することで、高速でアバターを生成することができるようになった。

推論部では、緻密な輪郭と荒い顔画像を別々に生成して結合することで、高速にアバターを生成することができるようになった。

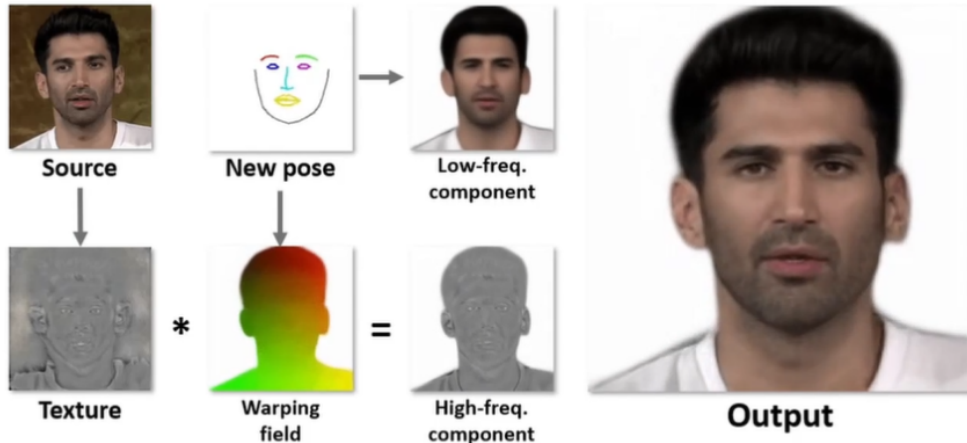


図 3: Avatar の生成過程

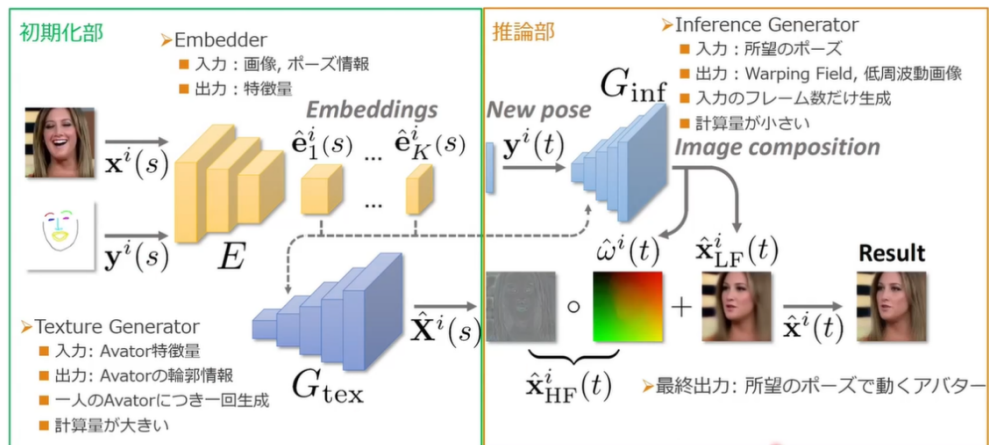


図 4: Avatar の生成過程

### 3 CGAN(conditional GAN)

#### 3.1 概要

CGAN(conditional GAN) は、条件付き生成モデルであり、生成器に条件パラメータを与えることで、GAN では指定できなかった生成したい画像のクラスを指定できる。CGAN の損失関数は、以下のように表せる。

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x|y)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z|y)))] \quad (18)$$

ただし、 $y$  は、条件である。例えば、犬、という条件を与えると、Generator は犬の画像を生成することができる。一方、Discriminator は、生成された画像かどうか、とその画像がラベルのクラスに分類されるかどうか、の2条件を判別する。その組み合わせの通りは、以下のようになる。

- G が生成した犬の画像 かつ ラベルが犬 と識別した場合→ 不正解
- G が生成した犬の画像 かつ ラベルが犬以外 と識別した場合→ 不正解
- 真の犬の画像 かつ ラベルが犬 と識別した場合→ 正解
- 真の犬の画像 かつ ラベルが犬以外 と識別した場合→ 不正解

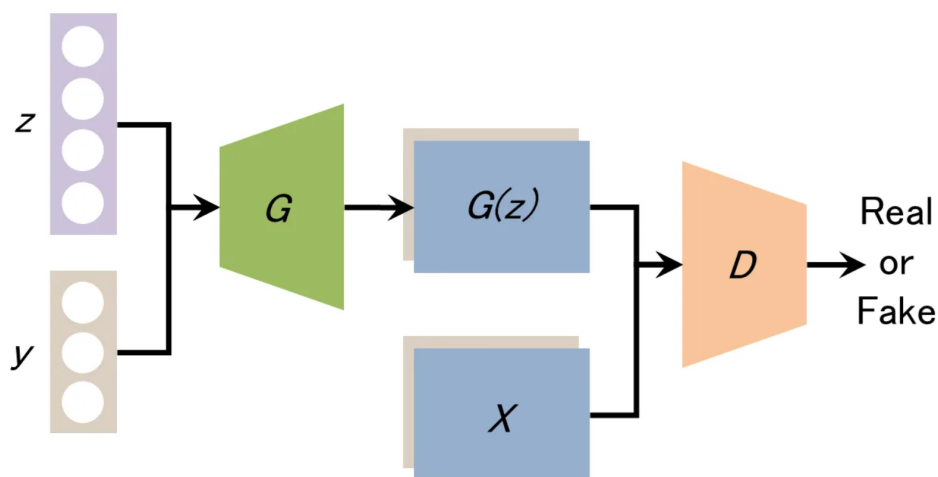


図 5: CGAN の構成

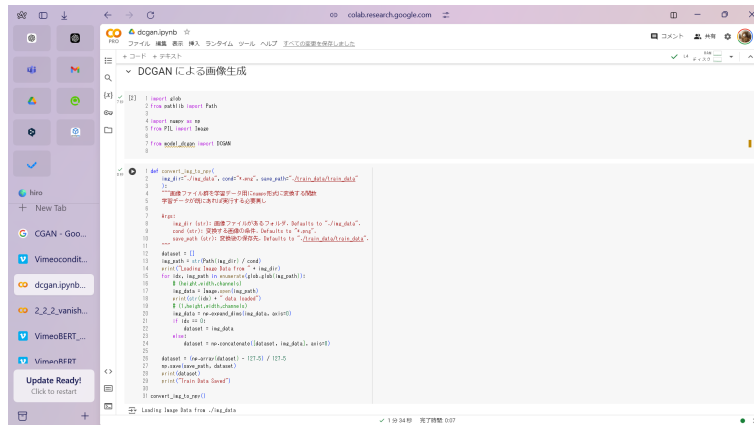
#### ■参考文献

1. 岡谷貴之/深層学習 改訂第2版 [機械学習プロフェッショナルシリーズ]/ 講談社サイエンティフィク/ 2022-01-17
2. 深層畳み込み敵対的生成ネットワーク (DCGAN) <https://www.tensorflow.org/tutorials/generative/dcgan?hl=ja>
3. CGAN (Conditional GAN): 条件付き敵対的生成ネットワーク NegativeMindException <https://blog.nativemind.com/2019/10/05/conditional-gan/>

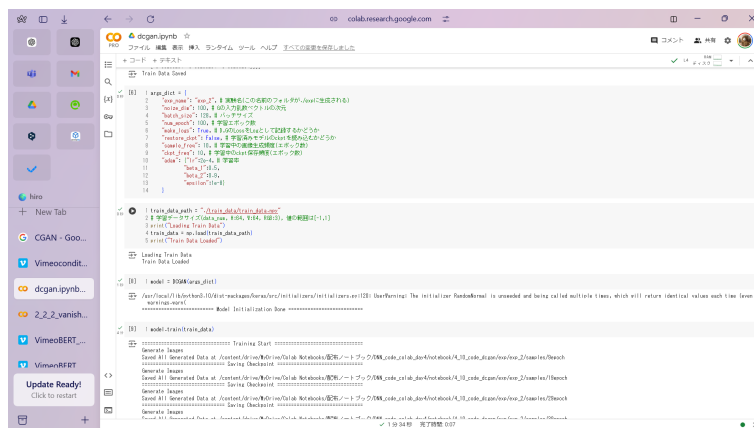
4. Github @garridoq/gan-guide <https://github.com/garridoq/gan-guide>

## 4 実装演習キャプチャ

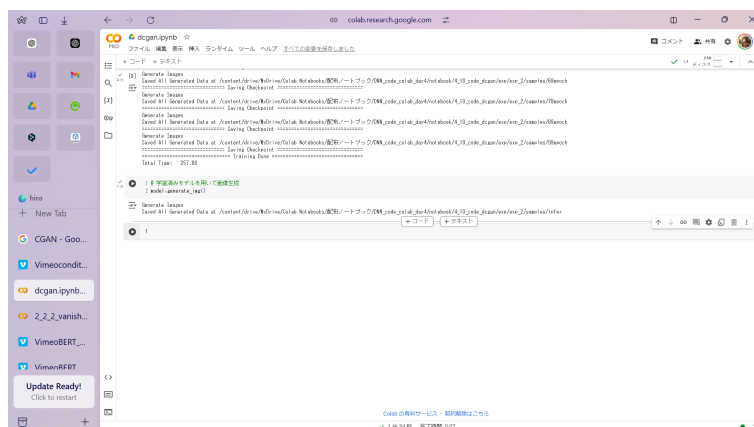
### 4.1 GAN



The first screenshot shows the initial code in the Colab notebook. The title is "DCGANによる画像生成". The code defines a function `convert_img_data` that takes a file path and a name, and returns a tuple of image data and labels. It then loads the data from the file path and converts it to a numpy array. The code also defines a function `train_data_loader` that returns a generator function for the training data.



The second screenshot shows the output of the training process. The code has been executed, and the output shows the progress of the training. The output includes the number of images generated, the number of images saved, and the number of images loaded. The output also shows the progress of the training, including the number of epochs completed and the current loss.



The third screenshot shows the final output of the GAN. The code has been executed, and the output shows the generated images. The output includes the number of images generated, the number of images saved, and the number of images loaded. The output also shows the progress of the training, including the number of epochs completed and the current loss.

図 6: (コードはうまく回ったが、うまく生成できなかった。)