令和 3 年度　特別研究報告書

敵対的生成ネットワークを用いた点群データ生成の検討

龍谷大学　理工学部　情報メディア学科

T180492 山本敬士

指導教員　三好力　教授

内容梗概

機械学習の発達、特に敵対的生成ネットワークの登場により、画像の識別をはじめとする従来の深層学習の研究に加え、画像そのものを生成する研究が行われるようになった。また、その派生として自然言語の文章や音楽などの生成についても同様に研究されている。

そこで、本研究では敵対的生成ネットワークを用いた 3D データの生成を提案する。既存の 3D データの生成手法ではボクセル形式が用いられているが、本研究では点群データ形式を採用した新たな手法を提案する。また、提案した手法を用いて学習を進めた際に、生成することができるのか検証する。この検証には比較用として全結合の多層パーセプトロンのみで構成される単純な敵対的生成ネットワークを用意しそれと比較することで提案手法の有用性を測る。そして実データと学習後の結果を比較することで、実用的な生成に用いることができるか検討を行う。

目次

第1章.　　はじめに

本研究では、画像生成などに用いられている Generative Adversarial Networks[1](GAN, 敵対的生成ネットワーク)と呼ばれる既存技術を応用し、3D データを生成するニューラルネットワークの提案とその有用性について検討することを目的とする。

GAN を用いて 3D データを生成する既存の研究として、3D-GAN[2]がある。この研究ではボクセル形式と呼ばれる小さな立方体の集まりで形状を表現する手法がとられているが、3D データの形式にはボクセル形式のほかにポリゴン形式や点群データ形式などがある。本研究では、x,y,z の 3 つの座標値を持つ点の集まりで表現する点群データ形式を用いて既存手法の 3D-GAN とは異なるアプローチで 3D データを生成する手法を検討する。この場合、従来の手法で用いられている 3 次元空間の畳み込みは困難なため、点群データのクラス分類などを目的に研究された PointNet の構造を GAN に組み込んで応用する手法を提案する。

第2章.　　先行研究・既存技術

2.1　GAN

GAN[1](Generative Adversarial Network, 敵対的生成ネットワーク)はニューラルネットワークの構成の 1 種。データに含まれる特徴を学習することで、実在しないデータを生成することができる。Generator(生成器)と Discriminator(識別器)の2つのニューラルネットワークを図 2.1 に示すように組合せて構成される。

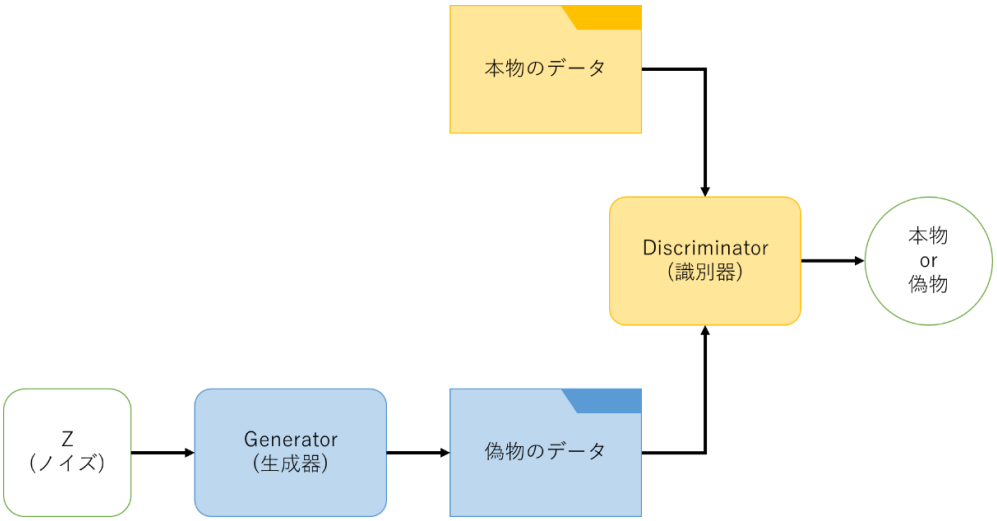

図 2.1 GAN の構成図

Generator はノイズ z から偽物のデータを生成し、Discriminator は Generator が生成した偽物のデータと本物のデータを判別する役割を持つ。2 つのネットワークが互いに競い合うことでそれぞれの性能が向上するように学習していく。

- Generator の学習
  Generator の損失関数L_gは以下のように表される。

$$L_g = \frac{1}{m}\sum_{i=1}^{m} \log\left(1 - D\left(G(z_i)\right)\right)$$

ここで、mはバッチ数、D は Discriminator、G は Generator、zは入力ノイズである。Generator が生成した偽データ G(z)を Discriminator が識別したとき、本物(1)と判定されると最小になる。Discriminator のパラメーターは固定して学習する。

- Discriminator の学習
  Discriminator の損失関数L_dは以下のように表される。

$$L_d = \frac{1}{m}\sum_{i=0}^{m}\left(logD(x_i) + \log\left(1 - D\left(G(z_i)\right)\right)\right)$$

  xは本物のデータを指す。Generator の学習と違い、この損失関数の最大化を目指して学習させる。これによって Generator と Discriminator が敵対関係となる。$\log D(x_i)$は本物のデータに対して 1 を、$\log\left(1 - D\left(G(z_i)\right)\right)$は偽物のデータに対して 0 を出力すると最大となる。

GAN の長所は、学習後の Generator に対してランダムノイズを入力することで、生成されるデータにランダム性が生まれることである。これによって出力ごとに異なるデータが生成されることになる。短所は学習が不安定なことである。Generator と Discriminator の均衡が崩れ学習が進まなくなる勾配消失や、一度 Discriminator を騙せるデータを出力すると、Generator が同じようなデータしか生成しなくなるモード崩壊といった問題がある。

2.2　3D-GAN

3D-GAN[2]は GAN の派生手法の1つで、3 次元畳み込みを利用したネットワークで構成される。生成器の概要図を下図 2.2 に示す。

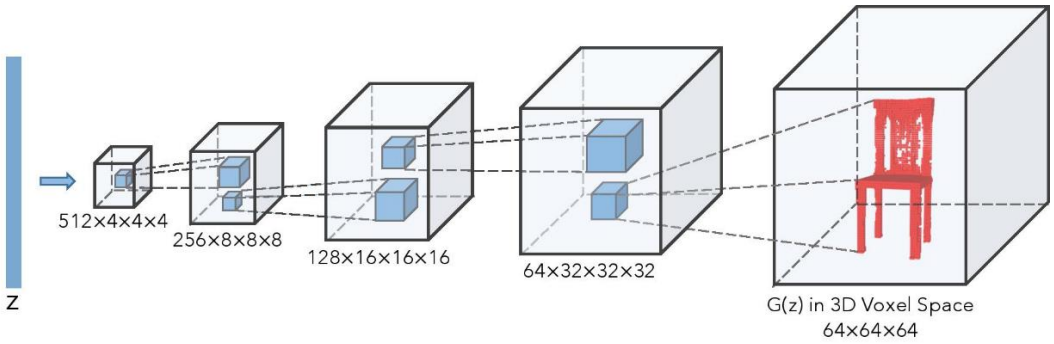

図 2.2 3D-GAN の Generator 構造([2]より引用)

生成器は初期入力のノイズ z を 3 次元空間にあてはめ、3 次元畳み込みとアップサンプリングを繰り返し最終的にボクセルのある確率を分割された空間ごとに出力する。識別器は 3 次元畳み込みとダウンサンプリングを繰り返したのち、特徴量の数を絞り最終的に偽物か本物かを識別する。

3D-GAN は従来の画像向けの 2 次元畳み込みを用いた GAN を 3 次元空間に拡張したもので、生成器側が少しずつ解像度を上げ、識別器側は徐々に解像度を落としていくという基本的な流れは同じである。違う点として、従来の画像用 GAN の生成器は各ピクセルが RGB の 3 チャンネルの出力を持つのに対して、3D-GAN では各ボクセル空間においてボクセルが存在する確率という1つの特徴量をそれぞれ持つようになっている。

2.3　PointNet

PointNet[3]は点群データのカテゴリ分類を行うニューラルネットワークである。また、物体のパーツごとへの分割、広い空間における物体の分類などに応用が可能である。PointNet ネットワークの構造を下図 2.3 に示す。

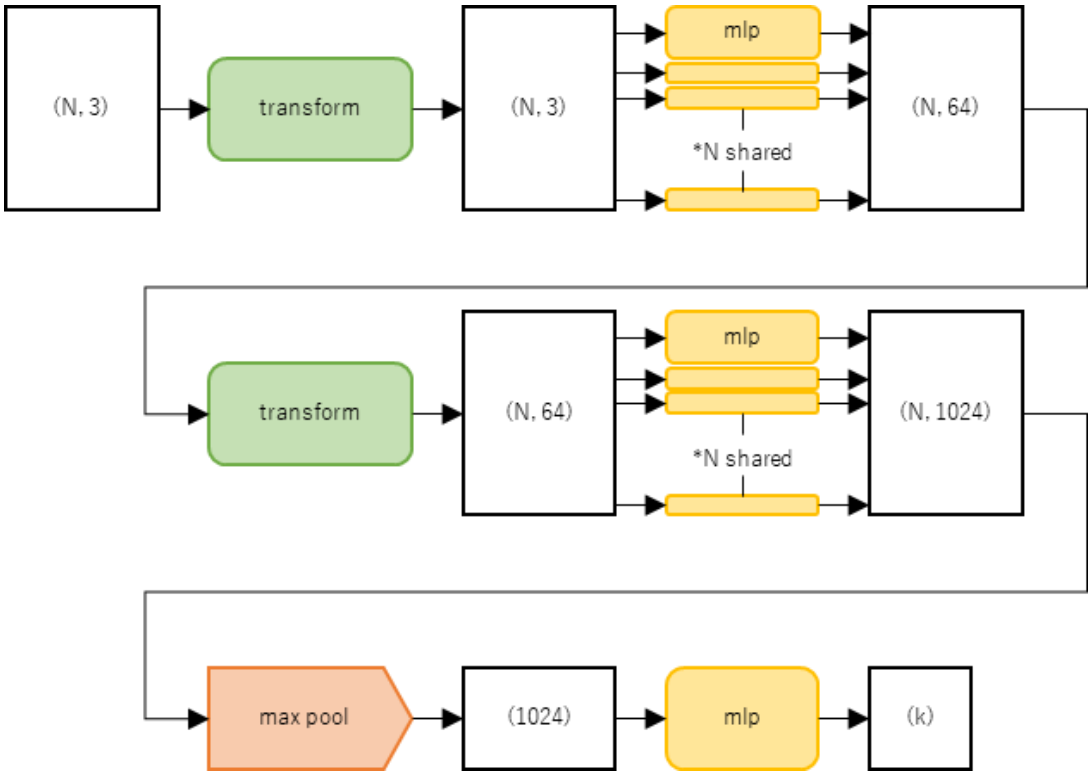

図　2.3 PointNet の構造図([3]を元に作成)

N は頂点数、k は分類するカテゴリ数、mlp は多層パーセプトロンを表す。transform は入力(N,M) (M は特徴量の数)から変換行列(M,M)を求め、その2つの積を出力とするネットワークで、図 2.4 のように構成される。

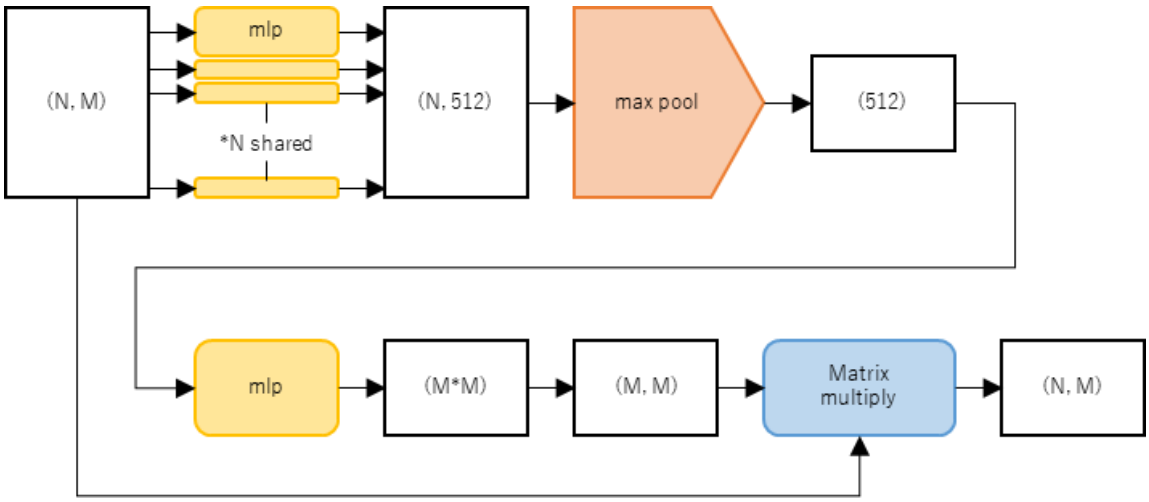

図 2.4 PointNet の transform 層の構造([3]を元に作成)

PointNet の特徴は後半部分や transform 層にみられる N*X の入力を X まで減らす部分である。ここには max pooling や average pooling などが用いられ、これらは入力の 1 次元目(頂点数 N の部分)の順序に影響を受けない。これによって順不変である点群データについて順序を入れ替えても結果が等しくなるようになっている。この特徴から引用元[3]では symmetric function と呼称されている。

2.4 既存手法の問題点

既存手法である 3D-GAN の問題点として以下の 2 点が挙げられる。

- 出力されるデータのサイズ

  画像におけるピクセルの数と同じようにボクセルデータにも分割サイズの解像度が存在する。3D-GAN の場合、64*64*64 の解像度が最終出力となっている。この解像度を上げれば生成されるモデルの精度も上げられるが、ボクセルの場合 3 次元なため、解像度を 2 倍にするとデータサイズは 8 倍と出力データのサイズが爆発的に増加してしまう。

- 教師用データ作成のための変換

  1章でも述べた通り、3D データにはいくつかの形式がある。広く使われているものとしては、映像やゲーム等の CG に用いられるポリゴン形式、LiDAR を用いたスキャンでは一般的な点群データ形式、3D-CAD の幾何形状を用いる形式などがある。それらと比べてボクセル形式のデータは利用例が多いとは言えない。したがって実在のデータを用いるにはこれらのデータをボクセル形式に変換する必要が出てくる。このとき、特にポリゴン形式や点群データ形式は物体の表面を表すデータなため、立方体の集まりで物体の中身までデータとして保持するボクセル形式への変換は相性があまりよくない。

第3章.　　提案手法

3.1　既存手法との比較

2.4 の問題に対して、既存手法とは異なる点群データ形式を採用することで問題を解決できないか検討した。まず、出力データの増大について、3D-GAN の解像度 64*64*64 の場合の出力データサイズは各ボクセルのデータを 1byte(8bit)とすると、262144byte となる。このとき、同サイズで点群データが表現可能な頂点数は各頂点に x,y,z それぞれ 4byte(32bit)の浮動小数点を用いたとすると、約 21800 頂点である。各ボクセルのデータを 1bit まで落とすと、同サイズの点群データの頂点数は約 2730 頂点となる。したがってこれらの頂点数以下で出力すればデータサイズの削減となる。本研究ではこれを満たす点群データの出力ネットワーク構成を目指す。

次に教師データについて、点群データを用いることで変換が不要または容易になる。LiDAR スキャンによるデータはほぼそのまま利用でき、ポリゴン形式のデータに対しても、ポリゴン上の頂点やメッシュ上に含まれる点をサンプリングすることで点群データに変換できる。

3.2　識別器の構造

識別器には PointNet の構造を用いた。下図 3.1 は識別器の構造図である。

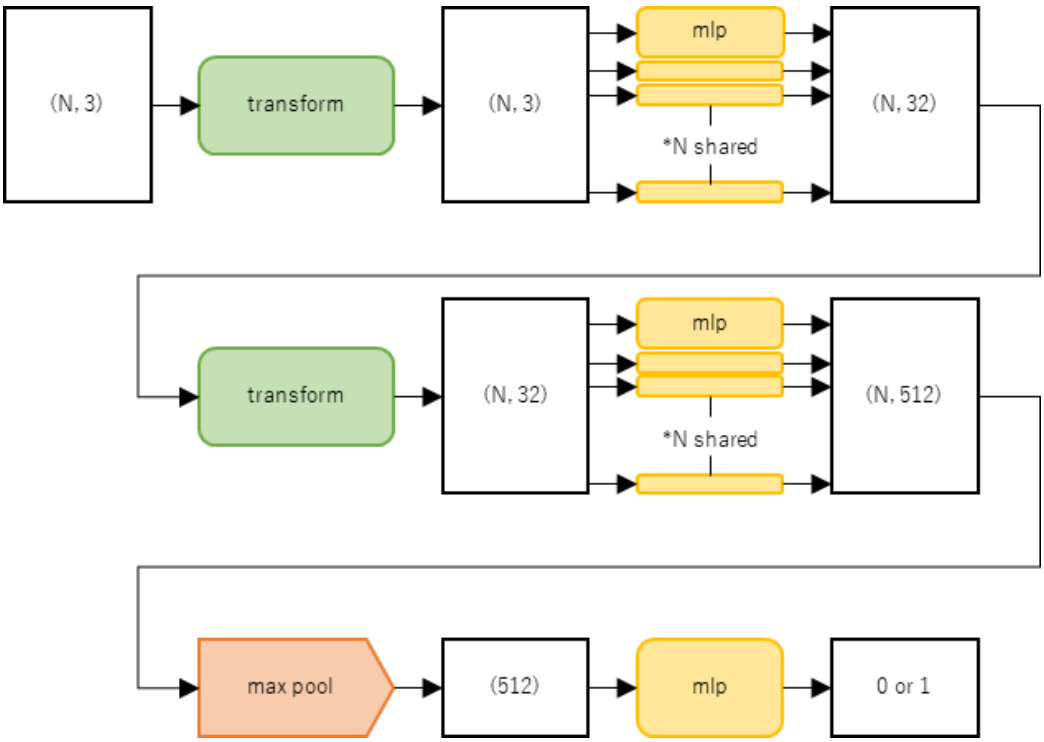

図 3.1 識別器の構造

3.3　生成器の構造

生成器の構造は下図 3.2 のとおりである。

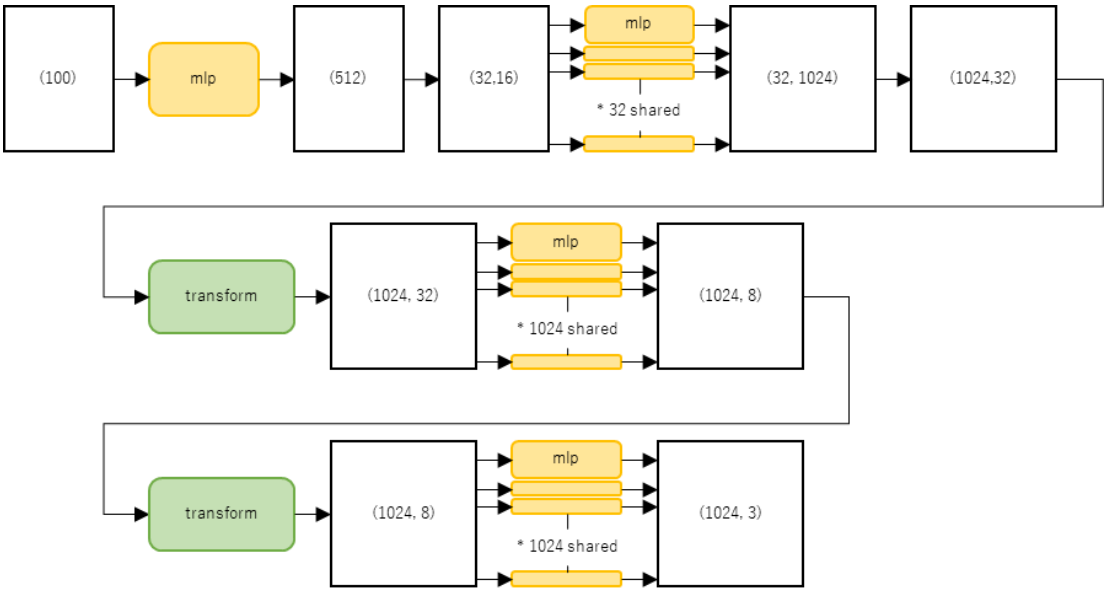

図 3.2 生成器の構造

第4章.　　実験

4.1　実験環境

本実験では、Python 用機械学習ライブラリの Tensorflow と Keras を用いた。実行に用いた計算機環境を表 4.1 に示す。

表 4.1 計算機の構築環境

OS	Windows 10 (x64)
メモリ	16GB
CPU	Intel(R) Core(TM) i7 4790 @ 3.60GHz
GPU	NVIDIA GeForce GTX 1050 Ti 4GB
Python	3.8.3
Tensorflow	2.2.0
CUDA	10.1.105
cuDNN	7.6.5

4.2　実験概要

PointNet の構造を用いる提案手法モデルと多層パーセプトロンのみで構成されるモデル（以下、比較用モデル）の2つを用意し学習過程や結果を比較することで PointNet の構造が GAN に流用可能かを検証し、また実際の点群データと比べて自然で違和感のない点群データを出力でき、従来手法の 3D-GAN に代わるほど実用的かどうかを検証する。以上の 2 点が実験目的である。

4.3　実験内容

機械学習用の 3D モデルデータセット ModelNet10[4]を利用し、頂点や面から点群をサンプリングして点群データに変換したものを教師データに用いる。このサンプリングには Python 用 3D データ向けライブラリである trimesh を用いており、全 3D データ共通で 1024 頂点をサンプリングした。ModelNet10 のデータとそのサンプリング例を図 4.1、4.2 に示す。

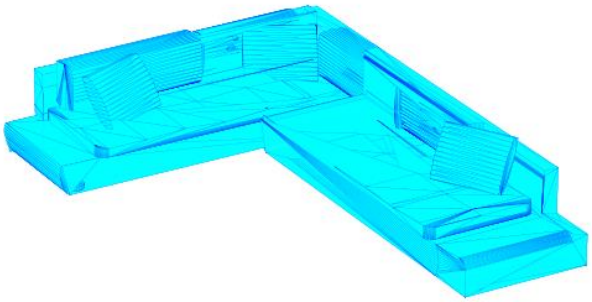

図 4.1 ModelNet10 の 3D データ例

図 4.2 上図 4.1 の点群サンプリング後の様子

学習条件はミニバッチ数 32、エポック数 2000 として提案手法モデルと比較用モデルの 2 つを ModelNet10 で分類される 10 カテゴリをそれぞれ学習させる。各エポックでは識別器を実データで学習、識別器を生成器の出力で学習、生成器を学習の3つがある。識別器の学習では生成器、生成器の学習では識別器のパラメーターを固定して片側ずつ学習を進める。

4.4　実験結果

提案手法の学習過程の損失変化を図 4.3 に、比較用モデルの学習過程の損失変化を図 4.4 に示す。提案手法の学習過程の識別器正答率を図 4.5 に、比較用モデルの学習過程の識別器正答率を図 4.6 に示す。

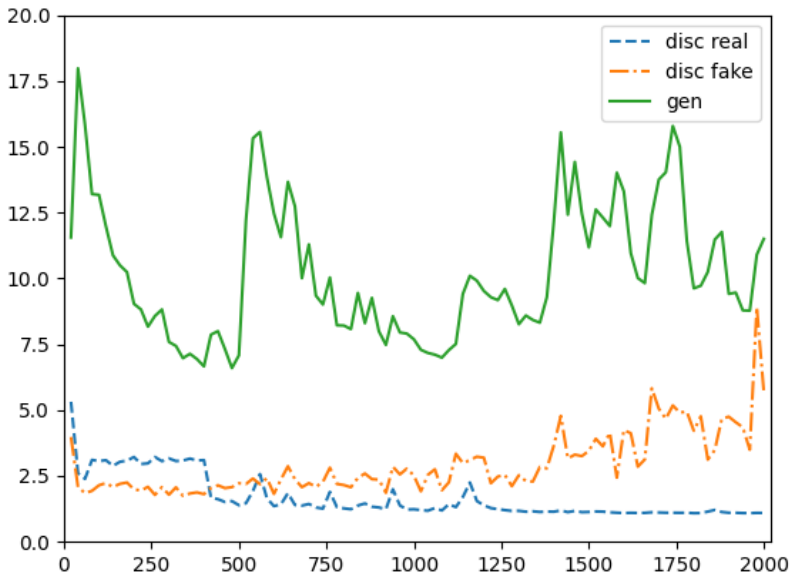

図 4.3　提案手法モデルの学習過程の損失

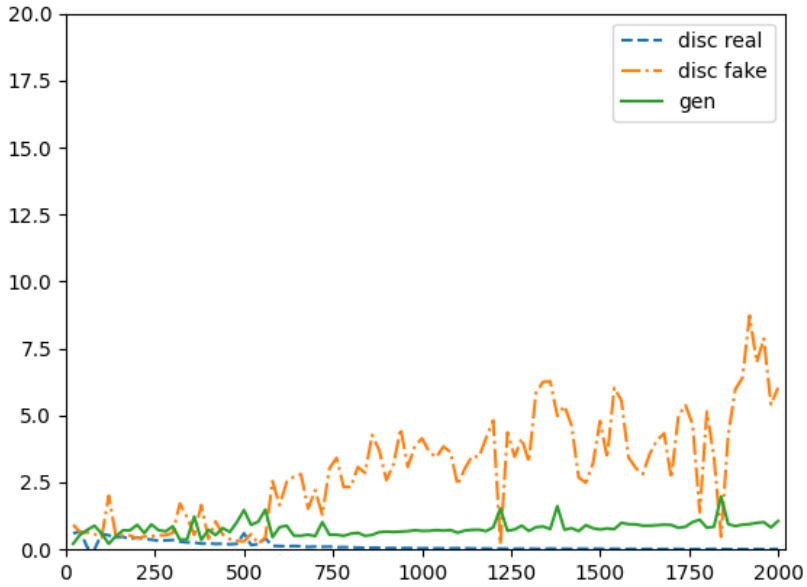

図 4.4　比較用モデルの学習過程の損失

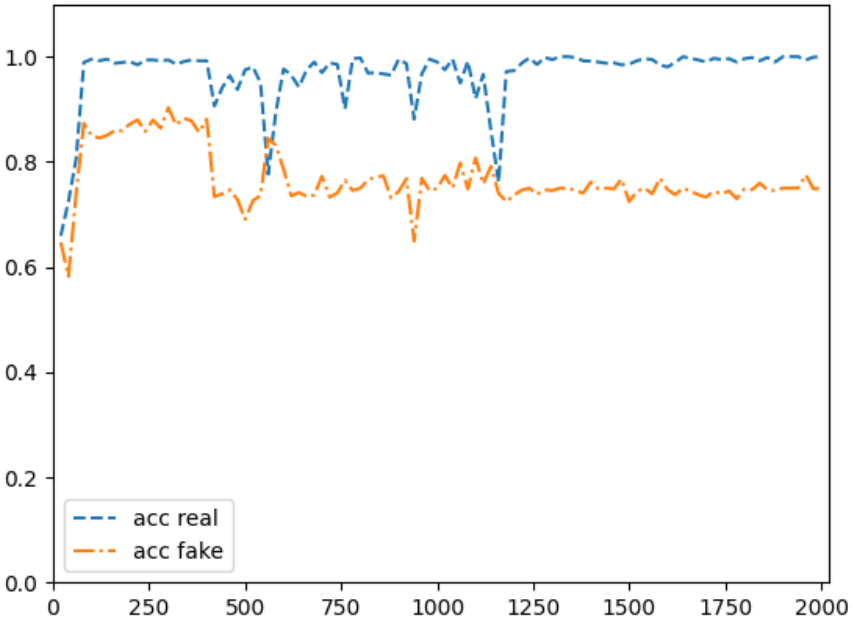

図 4.5 提案手法の学習過程における識別率変化

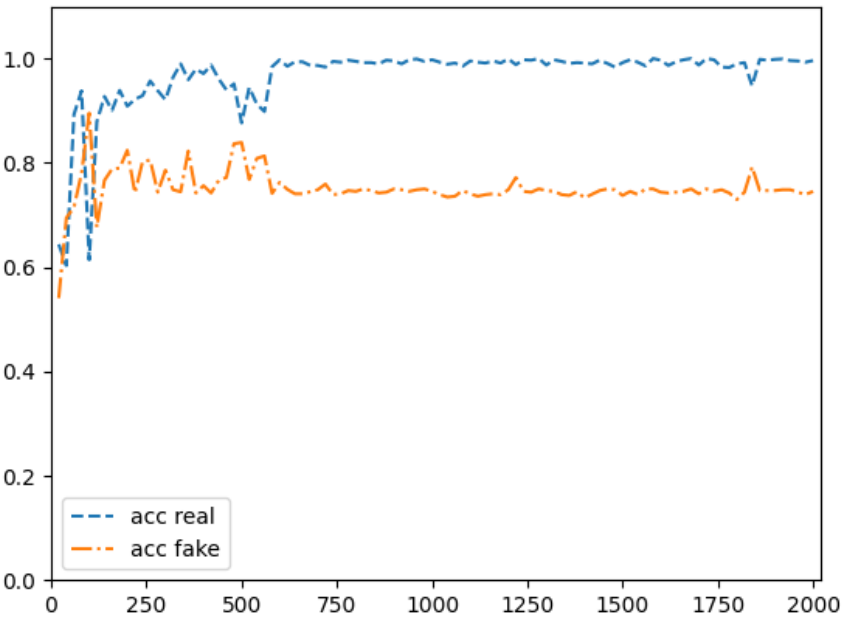

図 4.6 比較用モデルの学習過程における識別率変化

GAN の評価手法である FID(Fréchet Inception Distance)を用いて評価を行う。FID は事前に別で学習させた分類モデルを用いて測定される実データとの分布の差で、小さいほど実データの分布に近いことを示す。以下の式で求めることができる。

$$FID = \|\mu_r - \mu_f\| + Tr\left(\sum r + \sum f - 2\left(\sum r \sum f\right)^{-\frac{1}{2}}\right)$$

ここで、μ_r, μ_fはそれぞれ実データと GAN の生成データを分類モデルに入力したときの特徴ベクトルの平均、$\sum r, \sum f$はそれぞれの共分散行列を表す。Trは行列のトレースである。
カテゴリごとの FID スコアを表 4.2 に、グラフを図 4.7 に示す。

表 4.2 FID スコア

カテゴリ名	bathtub	bed	chair	desk	dresser	monitor	night stand	sofa	table	toilet
提案手法	5816.1	2511.2	329.4	1738.9	2418.0	2245.2	3621.4	1016.1	864.8	941.1
多層パーセプトロン	4990.0	2524.6	3409.9	1408.6	3036.1	4081.1	767.0	4871.9	1643.6	1752.2

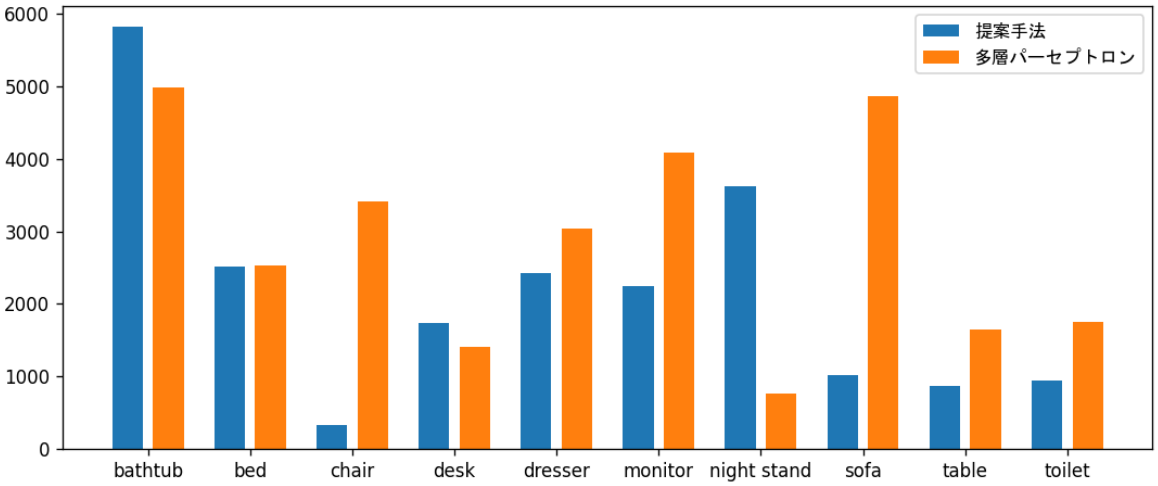

図 4.7 FID スコアの比較

生成された点群データと実データの例を図 4.8 に示す。

図 4.8 点群データの例

第5章.　　考察

実験結果より、図 4.3、4.4 の学習過程の損失変化について、提案手法モデルと比較用モデルで共通して実データを識別器に入力したときの損失(disc real)が一貫して低い値を保っている。また、生成データを識別器に入力したときの損失(disc fake)が上下にぶれている。したがって双方のモデルにおいて識別器は実データに対してかなり精度が高い状態であり、生成データについては生成器との敵対的な学習によって損失が増減を繰り返しており、理想的な学習過程だと考えられる。提案手法モデルと比較用モデルの大きな違いは生成器の損失(gen)である。提案手法モデルではエポックが進むごとに損失が増減を繰り返しているのに対して、比較用モデルでは損失がほぼ一定の値で変動が見られない。つまり、敵対的生成ネットワークの学習における生成器と識別器の敵対ができていない状態であると考えられる。

次に図 4.5、4.6 の学習過程の識別率の変化について、両方のモデルで実データにたいする識別率(acc real)が 0.9 以上、生成データに対する識別率(acc fake)は 0.75 前後になっている。先の損失変化についての考察で実データに対して適応しすぎていることが見られたが、このデータからもこの傾向が読み取れる。敵対的生成ネットワークが理想的な状態まで学習を進め、実データと比べて自然なデータを生成できるほどになった場合、識別器は実データ、生成データどちらに対しても識別率が 0.5 程度になるはずである。従って、今回の実験では識別器が生成器よりも学習しすぎているといえる。

FID スコアについては図 4.7 より、特に chair、sofa のカテゴリで提案手法モデルが比較用モデルより FID が小さく、逆に night stand では比較用モデルが提案手法モデルより優れた結果となった。それ以外のカテゴリでは提案手法モデルと比較用モデルで大きな差はみられなかった。

最後に図 4.8 の出力結果について、比較用モデルでは最大値または最小値に偏った立方体のような形状にしかならなかったのに対して、提案手法モデルは比較用モデルと比べて点群が一つの集合のように集まっている。しかしながら、その集合が実データと同じような自然な形状とは言えず、実用的なレベルには至っていないと考えられる。FID スコアが night stand や bathtub で比較用モデルが優れていたのは立方体のような形状がそれらと似ていたのが原因ではないかと推測される。

第6章.　　結論

3D データを生成する敵対的生成ネットワークとして、既存手法である 3D-GAN とは異なったアプローチの点群データ形式を用いたネットワークを提案した。提案手法には点群データ分類モデルの PointNet を応用した構造を用いた。提案手法の有用性を検証するために全結合の多層パーセプトロンのみで構成される単純な GAN を比較対象とし、それぞれ ModelNet10 の 3D データを用いて学習させる実験を行った。その結果、比較用モデルと比べ提案手法のほうがオブジェクトとして一つの集まりになっている傾向が見られた。また定量的評価の指標として FID を用いたが、こちらは提案手法が比較用モデルより優れた結果を残すカテゴリがいくつか見られた。しかし、その形状は実データとの類似性が低くとどまる結果であった。

以上のことから、3D 点群データの生成としての提案手法には、現段階では実用性は確認できなかった。

今後の課題として、出力結果を評価する際に定量的な FID スコアに加え、定性的な評価を行うことや、実用的な 3D 形状を出力できるようにネットワーク構造に工夫を加えて精度を向上させることが考えられる。

謝辞

本論文を作成するにあたり、多くのご指導、ご助言をいただきました三好　力　教授に厚くお礼申し上げます。

第7章.　　参考文献

[1] Generative Adversarial Networks
Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio

[2] Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
Jiajun Wu*, Chengkai Zhang*, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum

[3] PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation
Charles R. Qi, Hao Su, Kaichun Mo, Leonidas J. Guibas

[4] 3D ShapeNets: A Deep Representation for Volumetric Shapes (ModelNet10)
Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, Jianxiong Xiao.

[5] Point cloud classification with PointNet – Keras （2021/9/15）
https://keras.io/examples/vision/pointnet/

付録

- 学習用ソースコード(一部[5]より引用)

```
import os
import datetime
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dot, Permute, Dense, Flatten, Reshape, BatchNormalization, GlobalMaxPooling1D, Conv1D, Input
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam

label_names = ["bathtub", "bed", "chair", "desk", "dresser", "monitor", "night_stand", "sofa", "table", "toilet"]
NUM_POINTS = 1024
NUM_CLASSES = 10
BATCH_SIZE = 32
NOIZE_SIZE = 100

tf.random.set_seed(1234)
np.random.seed(seed=1234)

class OrthogonalRegularizer(keras.regularizers.Regularizer):
    def __init__(self, num_features, l2reg=0.001):
        self.num_features = num_features
        self.l2reg = l2reg
        self.eye = tf.eye(num_features)
    def __call__(self, x):
        x = tf.reshape(x, (-1, self.num_features, self.num_features))
        xxt = tf.tensordot(x, x, axes=(2, 2))
        xxt = tf.reshape(xxt, (-1, self.num_features, self.num_features))
        return tf.reduce_sum(self.l2reg * tf.square(xxt - self.eye))

def conv_bn(x, filters):
    x = Conv1D(filters, kernel_size=1, padding="valid")(x)
    x = BatchNormalization(momentum=0.0)(x)
    return tf.keras.layers.Activation("relu")(x)

def dense_bn(x, filters):
    x = Dense(filters)(x)
    x = BatchNormalization(momentum=0.0)(x)
    return tf.keras.layers.Activation("relu")(x)

def tnet(inputs, num_features):
    bias = keras.initializers.Constant(np.eye(num_features).flatten())
    reg = OrthogonalRegularizer(num_features)
    x = conv_bn(inputs, 32)
    x = conv_bn(x, 64)
    x = conv_bn(x, 512)
    x = GlobalMaxPooling1D()(x)
    x = dense_bn(x, 256)
    x = dense_bn(x, 128)
    x = Dense(
        num_features * num_features,
        kernel_initializer="zeros",
        bias_initializer=bias,
        activity_regularizer=reg,
    )(x)
    feat_T = Reshape((num_features, num_features))(x)
    return Dot(axes=(2, 1))([inputs, feat_T])

def build_discriminator():
    inputs = Input(shape=(NUM_POINTS, 3))
    x = tnet(inputs, 3)
    x = conv_bn(x, 32)
    x = conv_bn(x, 32)
    x = tnet(x, 32)
    x = conv_bn(x, 32)
    x = conv_bn(x, 128)
    x = conv_bn(x, 512)
    x = GlobalMaxPooling1D()(x)
    x = dense_bn(x, 128)
    x = dense_bn(x, 64)
    outputs = Dense(1, activation="sigmoid")(x)
    model = Model(inputs=inputs, outputs=outputs, name="discriminator")
    return model

def build_generator():
    inputs = Input((NOIZE_SIZE,))
    x = dense_bn(inputs, 128)
    x = dense_bn(x, 512)
    x = Reshape((32,16))(x)
    x = Conv1D(64, kernel_size=1, activation="relu")(x)
    x = Conv1D(256, kernel_size=1, activation="relu")(x)
    x = Conv1D(1024, kernel_size=1, activation="relu")(x)
    x = Permute((2, 1))(x)
    x = tnet(x, 32)
    x = conv_bn(x, 32)
    x = conv_bn(x, 16)
    x = conv_bn(x, 8)
    x = tnet(x, 8)
    x = conv_bn(x, 8)
    outputs = Conv1D(3, kernel_size=1, activation="sigmoid")(x)
    model = Model(inputs=inputs, outputs=outputs, name="generator")
    return model
```

```
def build_discriminator_mpl():
    inputs = Input(shape=(NUM_POINTS, 3))
    x = Flatten()(inputs)
    x = dense_bn(x, 1024)
    x = dense_bn(x, 256)
    x = dense_bn(x, 256)
    x = dense_bn(x, 64)
    outputs = Dense(1, activation="sigmoid")(x)
    model = Model(inputs=inputs, outputs=outputs, name="discriminator_alt")
    return model

def build_generator_mpl():
    inputs = Input((NOIZE_SIZE,))
    x = dense_bn(inputs, 128)
    x = dense_bn(x, 512)
    x = Dense(NUM_POINTS*3, activation="sigmoid")(x)
    outputs = Reshape((NUM_POINTS,3))(x)
    model = Model(inputs=inputs, outputs=outputs, name="generator_alt")
    return model

def build_gan(generator, discriminator, name=None):
    model = Sequential(name=name)
    model.add(generator)
    model.add(discriminator)
    return model

def train_category(discriminator, generator, gan, basedir, epochs, train_points, category_id):
    train_category_type = label_names[category_id]
    os.makedirs(os.path.join(basedir, train_category_type), exist_ok=True)
    f_loss = open(os.path.join(basedir, train_category_type, "loss.csv"), "w")
    f_acc = open(os.path.join(basedir, train_category_type, "acc.csv"), "w")

    for epoch in range(1,epochs+1):
        z = np.random.rand(BATCH_SIZE, NOIZE_SIZE)
        gen_points = generator.predict(z)
        batch_index = np.random.randint(0, train_points.shape[0], (BATCH_SIZE,))
        d_loss_real = discriminator.evaluate(train_points[batch_index], np.ones((BATCH_SIZE, 1)), verbose=0)
        d_loss_fake = discriminator.evaluate(gen_points, np.zeros((BATCH_SIZE, 1)), verbose=0)
        g_loss = gan.evaluate(z, np.ones((BATCH_SIZE, 1)), verbose=0)
        if (d_loss_real[0] > d_loss_fake[0] and d_loss_real[0] > g_loss) or d_loss_real[1] < 0.6:
            d_loss_real = discriminator.train_on_batch(train_points[batch_index], np.ones((BATCH_SIZE, 1)))
        if (d_loss_fake[0] > d_loss_real[0] and d_loss_fake[0] > g_loss) or d_loss_fake[1] < 0.6:
            d_loss_fake = discriminator.train_on_batch(gen_points, np.zeros((BATCH_SIZE, 1)))
        if (g_loss > d_loss_real[0] and g_loss > d_loss_fake[0]) or g_loss > 3.0:
            g_loss = gan.train_on_batch(z, np.ones((BATCH_SIZE, 1)))

        print ("epoch:{:>4} [D real loss: {}, acc.: {}] [D fake loss: {}, acc.: {}] [G loss: {}]".format(epoch, d_loss_real[0], d_loss_real[1], d_loss_fake[0], d_loss_fake[1], g_loss))
        print("{},{},{}".format(d_loss_real[0], d_loss_fake[0], g_loss), file=f_loss)
        print("{},{}".format(d_loss_real[1], d_loss_fake[1]), file=f_acc)

        if epoch % 100 == 0:
            os.makedirs(os.path.join(basedir, train_category_type, "epoch"   + str(epoch)), exist_ok=True)
            for i in range(BATCH_SIZE):
                with open(os.path.join(basedir, train_category_type, "epoch"   + str(epoch), "{:02}.obj".format(i)), "w") as f:
                    print("o point", file=f)
                    for v in gen_points[i]:
                        print("v {} {} {}".format(v[0]*2-1.0, v[1]*2-1.0, v[2]*2-1.0), file=f)

    f_acc.close()
    f_loss.close()

def train(discriminator, generator, epochs, datetime_str, model_type):
    basedir = os.path.join(datetime_str, model_type)
    os.makedirs(basedir, exist_ok=True)

    discriminator.save_weights(os.path.join(basedir, "discriminator_weight_init.h5"))
    generator.save_weights(os.path.join(basedir, "generator_weight_init.h5"))

    discriminator.compile(loss="binary_crossentropy", optimizer=Adam(), metrics=["accuracy"])

    discriminator.trainable = False
    gan = build_gan(generator, discriminator)
    gan.compile(loss='binary_crossentropy', optimizer=Adam())

    for i in range(NUM_CLASSES):
        print("Model Type:{} Category:{}".format(model_type, label_names[i]))
        train_points = np.load(os.path.join("data", str(NUM_POINTS), label_names[i], "train", "point.npy"))
        discriminator.load_weights(os.path.join(basedir, "discriminator_weight_init.h5"))
        generator.load_weights(os.path.join(basedir, "generator_weight_init.h5"))

        train_category(discriminator, generator, gan, basedir, epochs, train_points, i)

        discriminator.save_weights(os.path.join(basedir, label_names[i], "discriminator_weight.h5"))
        generator.save_weights(os.path.join(basedir, label_names[i], "generator_weight.h5"))

if __name__ == "__main__":
    datetime_str = datetime.datetime.now().strftime('%Y%m%d_%H%M')

    #PointNet train
    generator = build_generator()
    discriminator = build_discriminator()
    train(discriminator, generator, 2000, datetime_str, "PointcloudGAN")

    #MPL train
    generator_mpl = build_generator_mpl()
    discriminator_mpl = build_discriminator_mpl()
    train(discriminator_mpl, generator_mpl, 2000, datetime_str, "MPL")
```