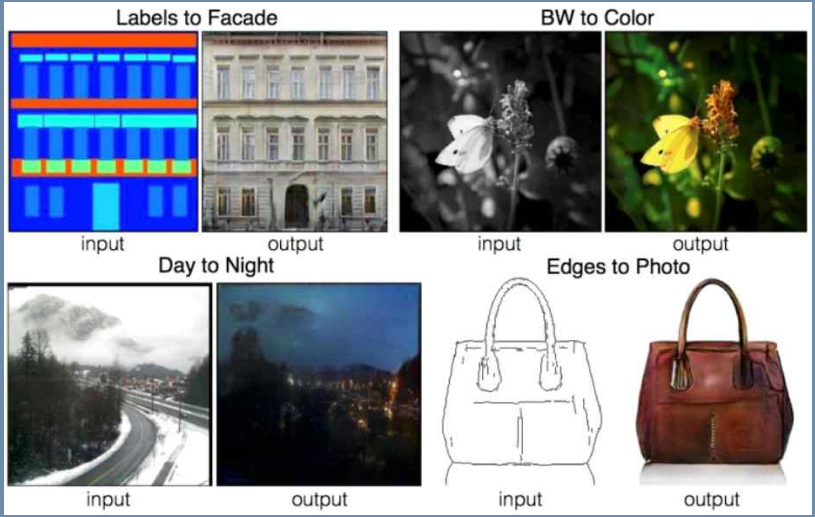
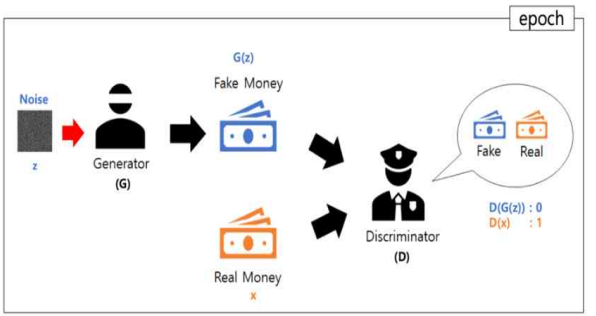
GAN

(Generative Adversarial Networks)

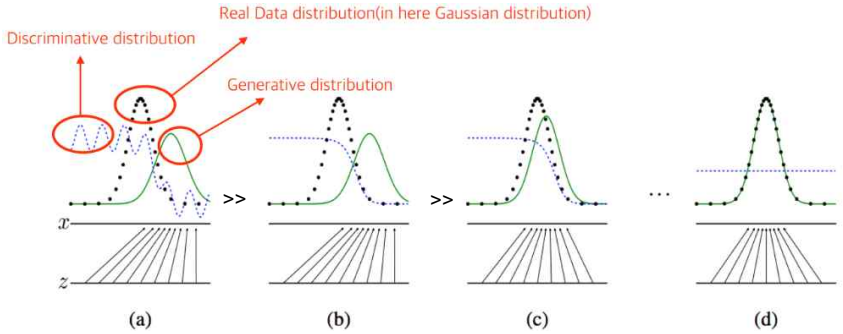
GANはGenerative Adversarial Networksの 略で、
Ian Goodfellow によって2014 年に初めて神経情報処理システム学会 (NIPS) で提案され、
その後、様々な老文に派生して発展し、イメージ生成、映像生成、テキスト生成などに多様に応用されています。



GANはGeneratorとDiscriminatorという2つのネットワークで構成されており、この2つのネットワークを敵対的に学習させて目的を達成します。例えば、生成モデルは本物の紙幣と似た偽札を作って警察を騙そうとする偽造紙幣犯と同じで、逆に判別モデルは偽造紙幣犯が作り出した偽札を探知しようとする警察と似ています。こうした競争が続くにつれ、偽造紙幣犯は警察を騙せなかったデータを、警察は偽造紙幣犯に騙されたデータをそれぞれ入力してもらい、敵対的に学習するようになるのです。このゲームでの競争は、偽造紙幣が本物の紙幣と区別されないまで、つまり与えられた標本が実際の標本になる確率が0.5に近い値を持つまで続きます。 偽物だと確信する場合は判別器の確率値が0、実際だと確信する場合は判別器の確率値が1を表すことになり、判別器の確率値が0.5ということは偽物か本物かを判別しにくいことを意味するようになります。



Generative Modelは、元のデータの分布を近似できるように学習



$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

V(G) : 値を最小限化するように学習（赤ボックス式ではGeneratorを使用しないことで省略）

V(D) : 値を最大化するように学習

原本データから1つのデータ(x)を抜いて、Discriminatorに入れてlogを取った期待値

1つのノイズ（z）を抜いて、Generatorに入れて、偽データを生成した上でDiscriminatorに入れた値をマイナスして1を足した値の対数を取った期待値

必要ライブラリ、データの読み込み

```
import torch
import torch.nn as nn
import numpy as np
from torchvision import transforms
from torch.autograd import Variable
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
from torchvision import datasets
import torchvision
```

```
transform = transforms.Compose([
        transforms.Resize(28),
        transforms.CenterCrop(28),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.ToTensor(),
        transforms.Normalize(mean=(  0.5), std=( 0.5))
])
dataset = datasets.MNIST(root="./dataset", train=True, download=True, transform=transform)
data_loader = torch.utils.data.DataLoader(dataset, batch_size=64, shuffle=True, num_workers=4)
```

```python
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()

        self.label_emb = nn.Embedding(10, 10)

        self.model = nn.Sequential(
            nn.Linear(794, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.3),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x, labels):
        x = x.view(x.size(0), 784)
        c = self.label_emb(labels)
        x = torch.cat([x, c], 1)
        out = self.model(x)
        return out.squeeze()
```

Discriminatorクラス定義
活性化関数:LeakyReLU使用
最後にSigmiod関数を使って0~1を持つように設定

```python
class Generator(nn.Module):
    def __init__(self):
        super().__init__()

        self.label_emb = nn.Embedding(10, 10)

        self.model = nn.Sequential(
            nn.Linear(110, 256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(256, 512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(512, 1024),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Linear(1024, 784),
            nn.Tanh()
        )

    def forward(self, z, labels):
        z = z.view(z.size(0), 100)
        c = self.label_emb(labels)
        x = torch.cat([z, c], 1)
        out = self.model(x)
        return out.view(x.size(0), 28, 28)
```

Generatorクラス定義
最後にTanhを使って-1~1を持つように設定

loss, optimizer関数設定

```
criterion = nn.BCELoss()
d_optimizer = torch.optim.Adam(discriminator.parameters(), lr=1e-4)
g_optimizer = torch.optim.Adam(generator.parameters(), lr=1e-4)
```

Generatorのloss値計算関数

```
def generator_train_step(batch_size, discriminator, generator, g_optimizer, criterion):
    g_optimizer.zero_grad()
    z = Variable(torch.randn(batch_size, 100)).cuda()
    fake_labels = Variable(torch.LongTensor(np.random.randint(0, 10, batch_size))).cuda()
    fake_images = generator(z, fake_labels)
    validity = discriminator(fake_images, fake_labels)
    g_loss = criterion(validity, Variable(torch.ones(batch_size)).cuda())
    g_loss.backward()
    g_optimizer.step()
    return g_loss.data
```

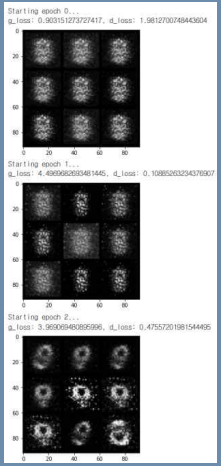
Discriminatorのloss値計算関数

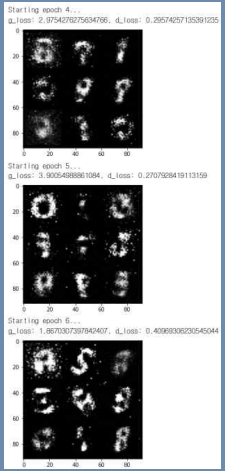
```
def discriminator_train_step(batch_size, discriminator, generator, d_optimizer, criterion, real_images, labels):
    d_optimizer.zero_grad()

    # train with real images
    real_validity = discriminator(real_images, labels)
    real_loss = criterion(real_validity, Variable(torch.ones(batch_size)).cuda())

    # train with fake images
    z = Variable(torch.randn(batch_size, 100)).cuda()
    fake_labels = Variable(torch.LongTensor(np.random.randint(0, 10, batch_size))).cuda()
    fake_images = generator(z, fake_labels)
    fake_validity = discriminator(fake_images, fake_labels)
    fake_loss = criterion(fake_validity, Variable(torch.zeros(batch_size)).cuda())

    d_loss = (real_loss + fake_loss) / 2
    d_loss.backward()
    d_optimizer.step()
    return d_loss.data
```

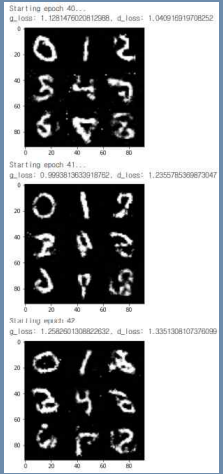
학습

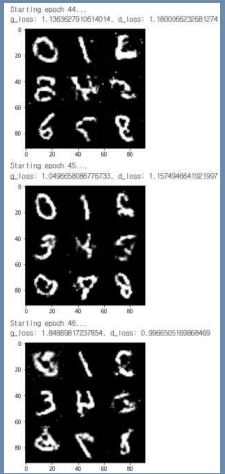
```
num_epochs = 30
n_critic = 5
display_step = 300
for epoch in range(num_epochs):
    print('Starting epoch {}...'.format(epoch))
    for i, (images, labels) in enumerate(data_loader):
        real_images = Variable(images).cuda()
        labels = Variable(labels).cuda()
        generator.train()
        batch_size = real_images.size(0)
        d_loss = discriminator_train_step(len(real_images), discriminator,
                                          generator, d_optimizer, criterion,
                                          real_images, labels)


        g_loss = generator_train_step(batch_size, discriminator, generator, g_optimizer, criterion)

    generator.eval()
    print('g_loss: {}, d_loss: {}'.format(g_loss, d_loss))
    z = Variable(torch.randn(9, 100)).cuda()
    labels = Variable(torch.LongTensor(np.arange(9))).cuda()
    sample_images = generator(z, labels).unsqueeze(1).data.cpu()
    grid = make_grid(sample_images, nrow=3, normalize=True).permute(1,2,0).numpy()
    plt.imshow(grid)
    plt.show()
```

学習結果

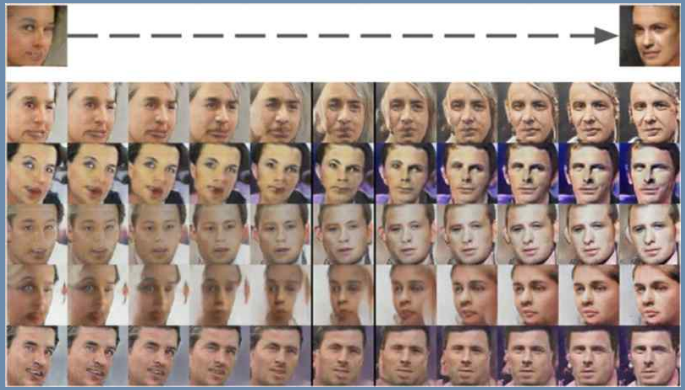








生成されたFAKEイメージ



生成モデルとしてのGANがデータを偶然作り出すのか、データを完璧に理解している価値のあるモデルなのかを調べるために、Gの出力が人の顔だとした時、左を眺める顔を作り出すz(左)たちの平均ベクトルと右を見ている顔に対応するz(右)たちの平均を計算し、この二つのベクトル間の軸を中間からinterpolationて生成者で入力するとゆっくり回転する顔が出てくることが確認できます。
この結果は、GAN生成者が学習したディープラーニングアルゴリズムが正確にデータの意味を理解し、データの確率分布を正確に表現していることで、入力での若干の変化が出力でもスムーズな変化で表現できるという事実を示しています。

限界点

既存のGANの限界点は大きく2つに分かれます。

1.（性能評価）

GAN モデルの性能を客観的数値で表現できる方案が不在GANの場合、結果自体が新しく作られたデータであるため比較可能な定量的尺度がない。

2.（性能改善）

GANは既存のネットワーク学習方法と異なる構造で学習が不安定。 GANはSaddle Problemあるいは Minmaxを解かなければならない生まれつき不安定な構造。

しかし、この2つの短所をすべて改善し、GANの後続研究が 発展できるように
DCGAN(Deep Convolutional GAN)が開発されました。

GANの種類